Applying Formal Tools to Compositional Systems

Robert L. Constable, Kenneth Birman (Cornell),
Danny Dolev (Hebrew University)

Jason Hickey
Christoph Kreitz
Mark Hayden

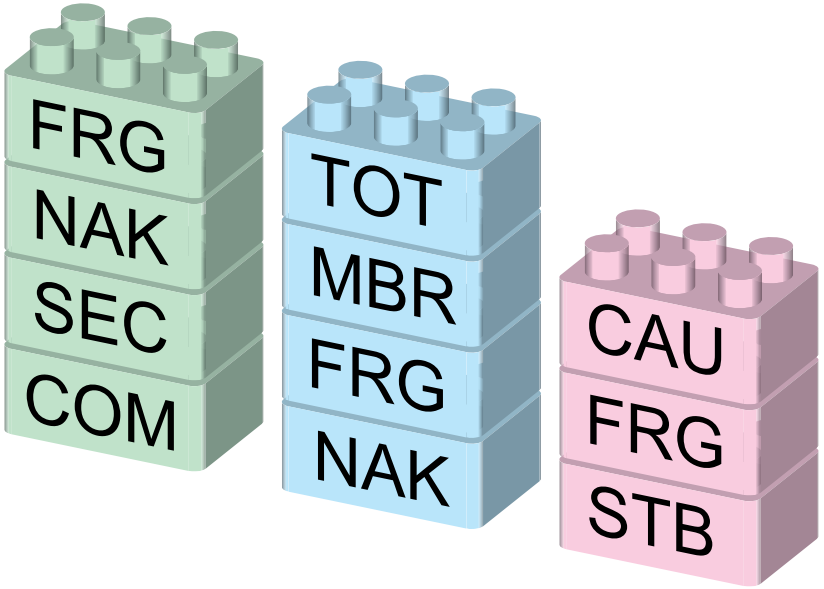
Outline

- **Issues in compositional systems**
- **Formal tools for addressing problems**
- **Formal methodology**

Modular Programming

Group Communication System

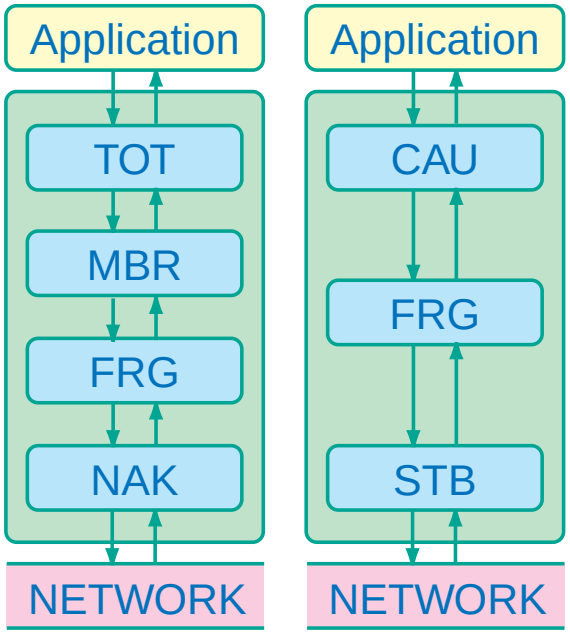
- **Flexible**
- **Easy to**
  - **design**
  - **maintain**
  - **verify**



- **Construction by *composition***
- **“Dynamic Composition and Wrapping with Ensemble,” Robbert Van Renesse**

Cost of modularity

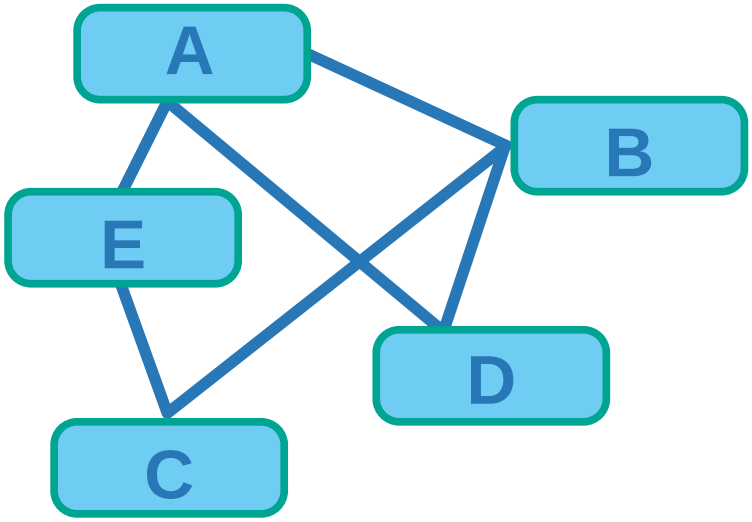
- **Performance**
  - **Redundant code**
  - **Abstraction enforcement**
- **Verification**
  - **Combinatorial number of configurations**



Locality

- **Modules define *local* behavior**
- **Global properties are hard to verify**
  - **Many module compositions**
- **Horus four years ago:**

  layers in C
  - **Frequent local optimization**
  - **Difficult global optimization**



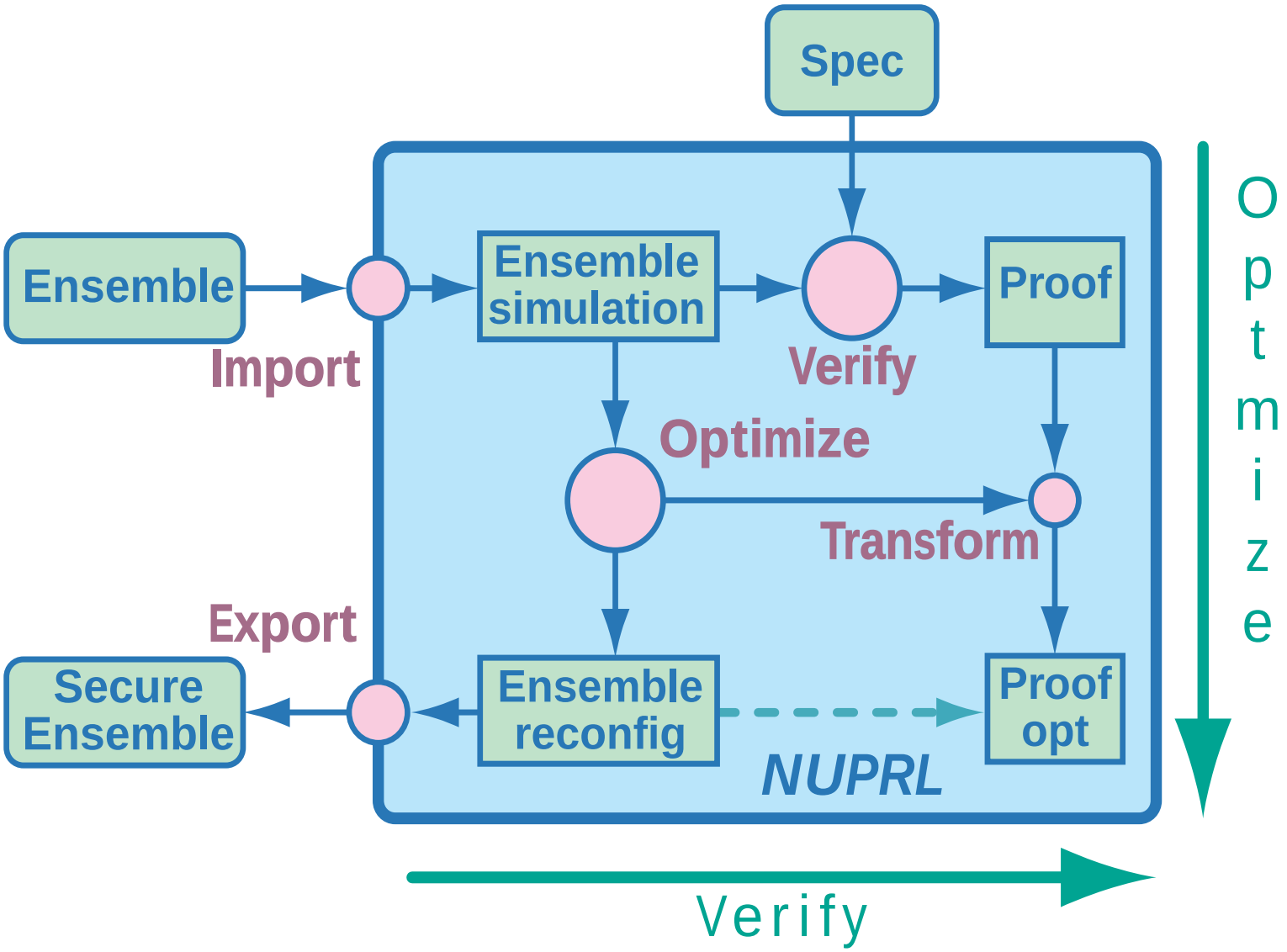
Solution

- **Create well-defined, simplified execution model**
  - **No global operations**
  - **Simplify functionality**
    - Layer threading
    - Message queuing
- **Use language with formal semantics**
  - **ML family of formal languages**
- **Enables:**
  - **Precise reasoning about behavior**
  - **Use of advanced formal tools**

ML successes

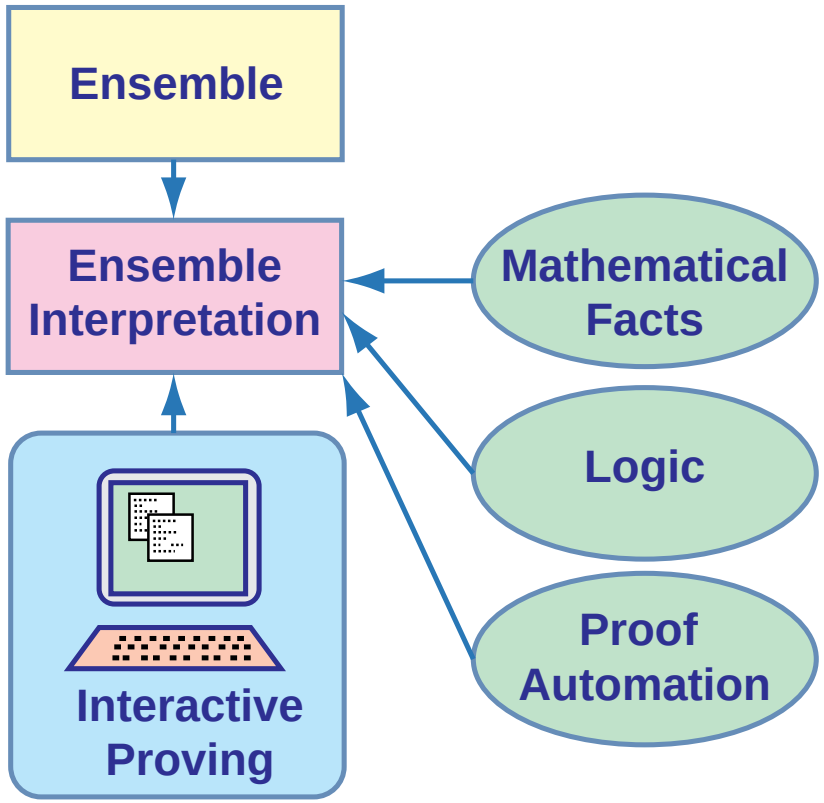
- **Formal & informal reasoning**
  - **Program transformation**
  - **Verification**
  - **7x reduction in code size typical**
- **Example**
  - **6 layer stack: 1500μs**
  - **Optimized: 30μs**
- **Java**

Formal analysis



Nuprl

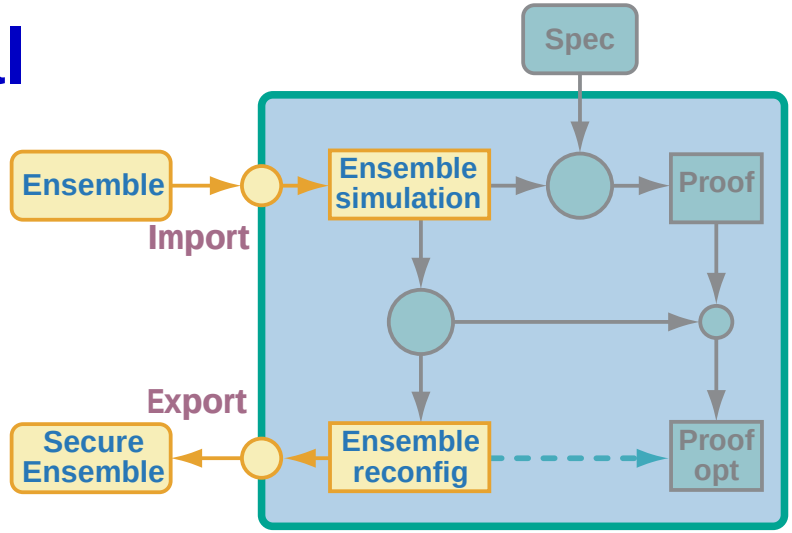
- **Deductive proof system**
  - **Interactive theorem proving**
  - **Heuristics, decision procedures, tactics**
- **Theory of types**
  - **Functional programming language**
  - **Expressive type system**
  - **Refinement of core ML**



Embedding

Mapping ML ⟷ formal language

- **ML is *not* directly formal**
- **Requirements**
  - Import & export algorithms
  - Formal ML semantics

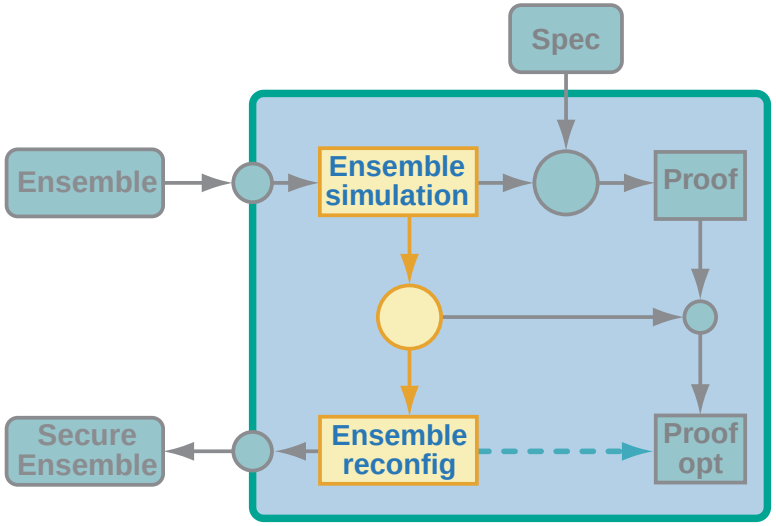


ML Interpretation

- **Interpretation of core OCaml**
- **Restricted calculus of effects**
- **Suggests formal ML extensions**

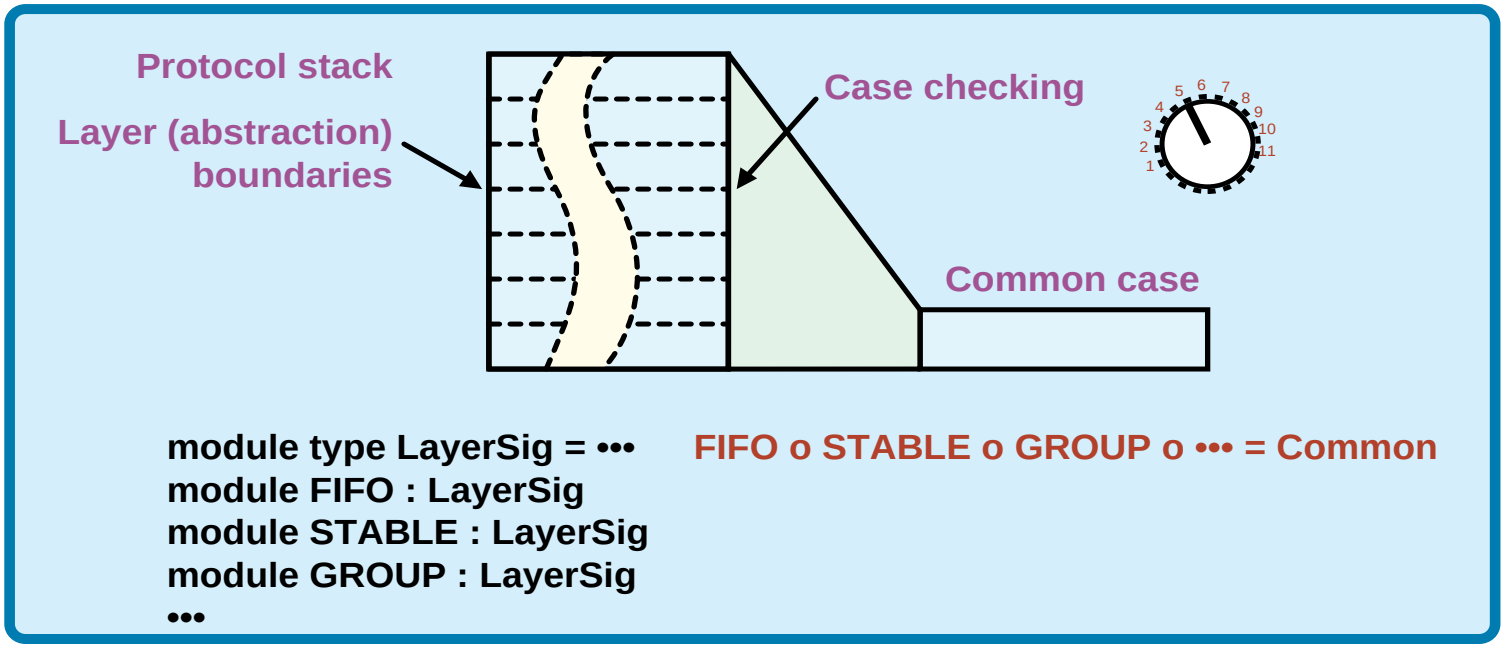
Program Transformation

- Global program optimization
- Tension with modularity
- Too many cases to do by hand
- Formal, correctness-preserving

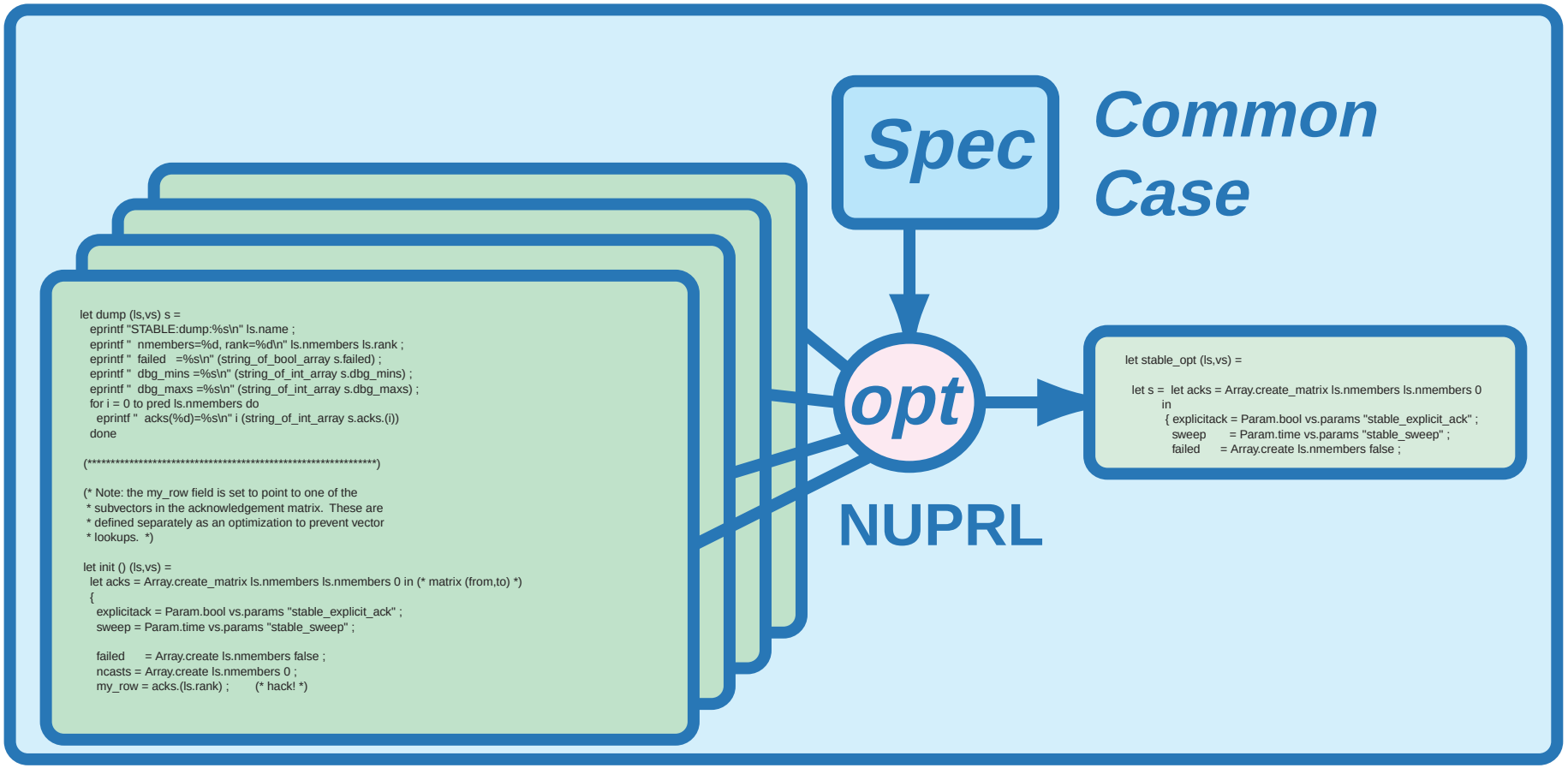


Transformations

- **Code Inlining**
- **Redundant code elimination**
- **Example: messages are not typically reordered**



Example: buffer management

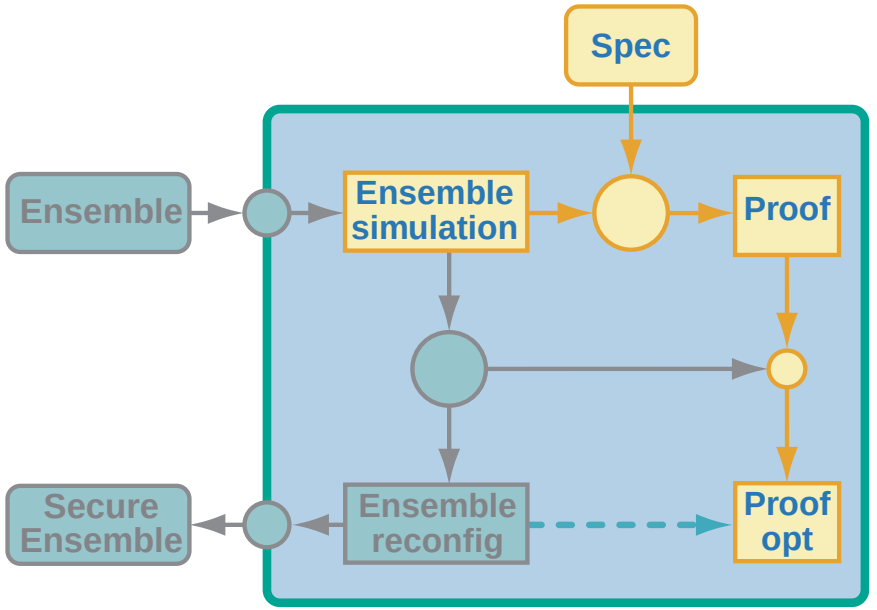


Tools for transformation

- **Computation rules for ML**
- **Derived inference rules**
- **Substitution/equality reasoning**
- **Tactics for transformation automation**

Verification

- **Verify original program**
- **Transform verification**



Conclusion

- **Formal methods address problems of composable systems**
- **Proof of concept**
  - **Ensemble embedding/optimization**
- **Future work: proof automation**
  - **More extensive verification strategies**
  - **More extensive optimization strategies**