The End of History?
Using a Proof Assistant to Replace Language Design with Library Design

Adam Chlipala
MIT CSAIL
SNAPL
April 2017

Joint work with: Benjamin Delaware, Samuel Duchovni, Jason Gross, Clément Pit—Claudel, Sorawit Suriyakarn, Peng Wang, and Katherine Ye

How We're Doing

Software bug causes launch failure

Hardware bug causes massive recall

Software bug leaks secret information

Software bug causes loss of life

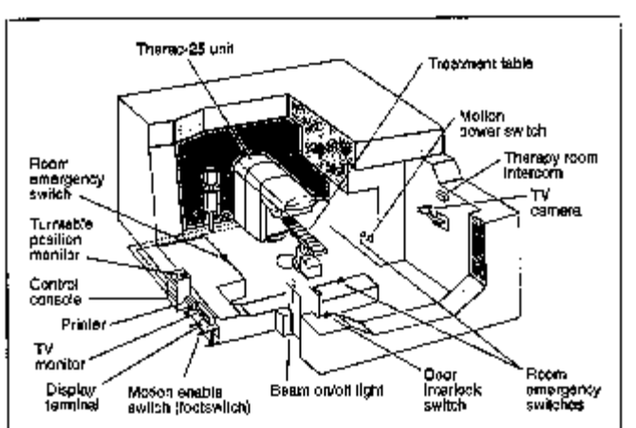
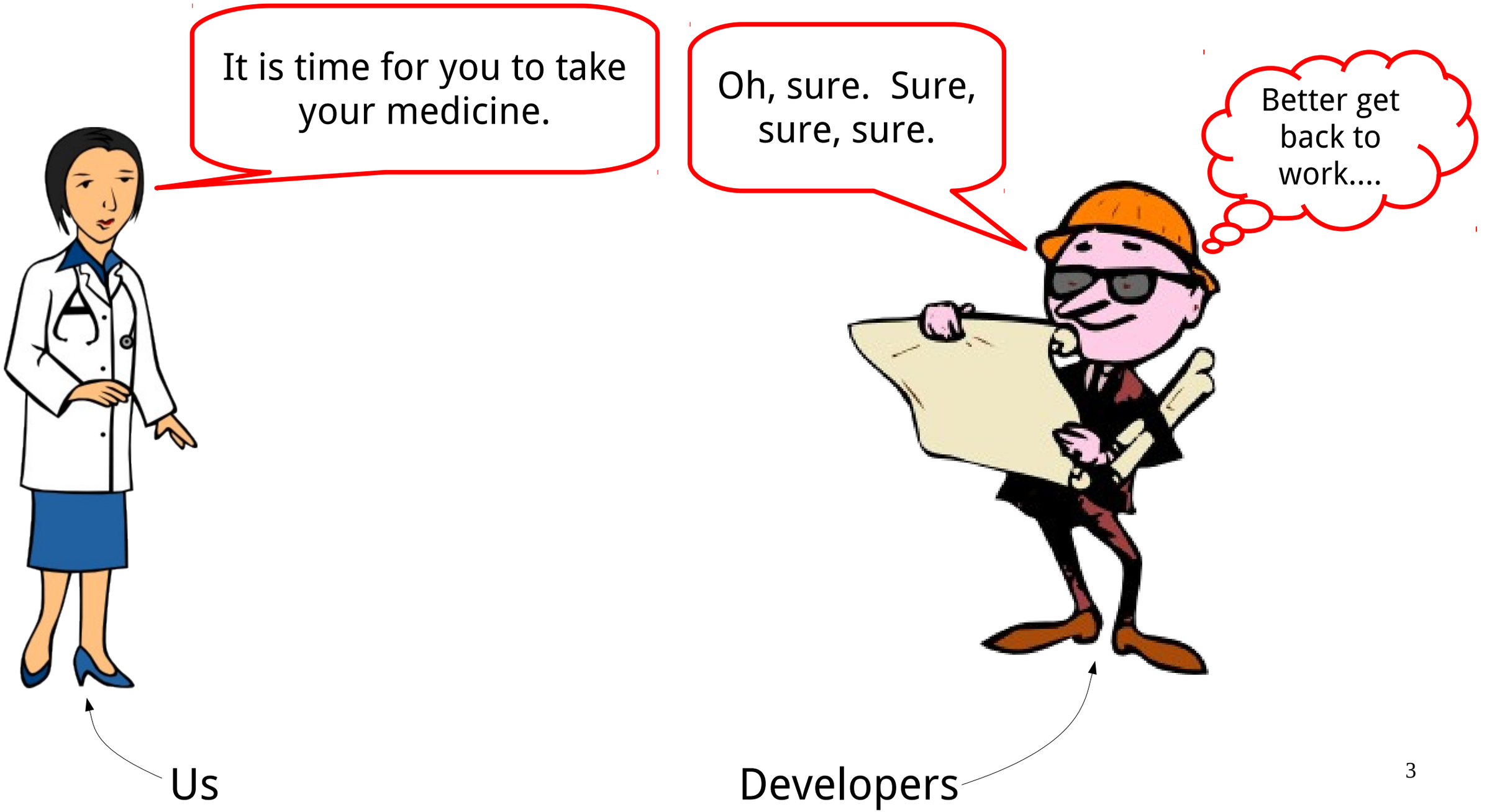


Figure 1. Typical Therac-25 facility.

<u>The time has come to settle for nothing less than high-assurance computer systems!</u>


It is time for you to take your medicine.
Oh, sure. Sure, sure, sure.
Better get back to work....
Us
Developers

Analog computers

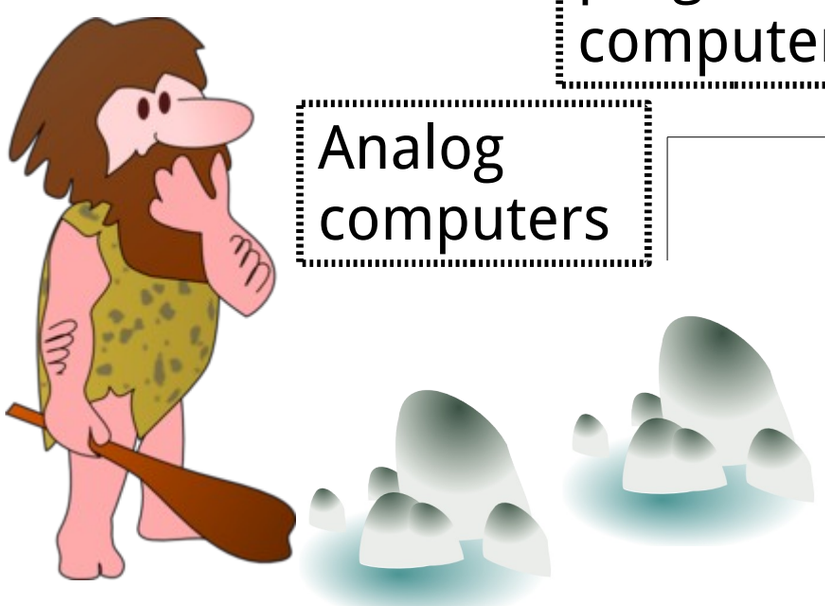
Stored-program computers

Assembly language

Structured programming

Data abstraction

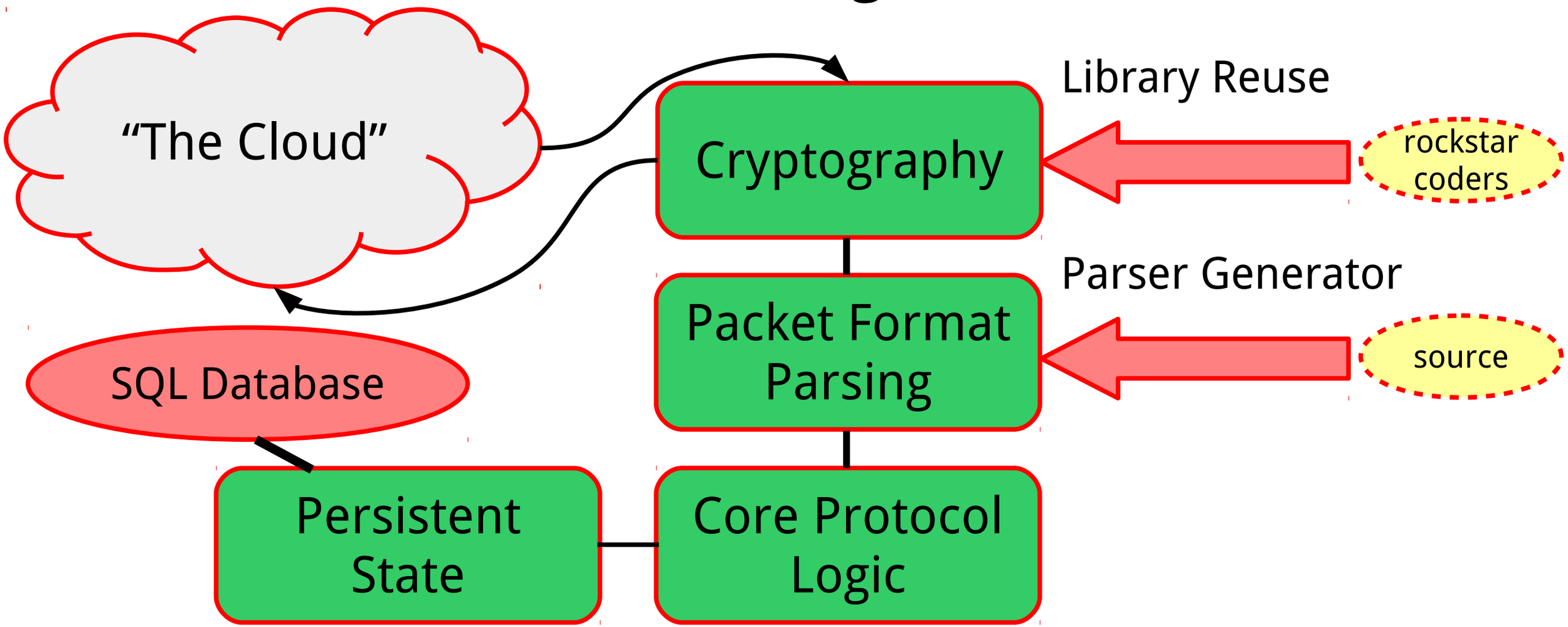
Formal methods



Formal specifications and proofs deserve to be *the new glue* holding together complex systems and helping us understand them and their parts.

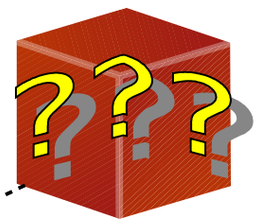
The design of systems should *change* to take advantage of formal methods to *raise the level of abstraction*.

State of the Art: Building an Internet Server



Complaints About: Talking to a Standard Server

And by the way, sometimes there are serious bugs.



Database is a black box, maintained by an elite cadre, often not doing quite what you need

SQL Database

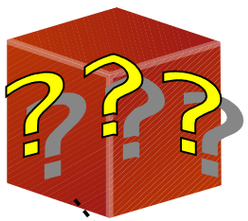
Yet another language, only understandable after reading a pile of documentation

Awkward API, often based on string manipulation, allowing code-injection vulnerabilities

Persistent State

Complaints About: Using a Domain-Specific Language

And by the way, sometimes there are serious bugs.



Compiler is a black box, maintained by an elite cadre, often not doing quite what you need

Parser Generator

source

Packet Format Parsing

Yet another language, only understandable after reading a pile of documentation

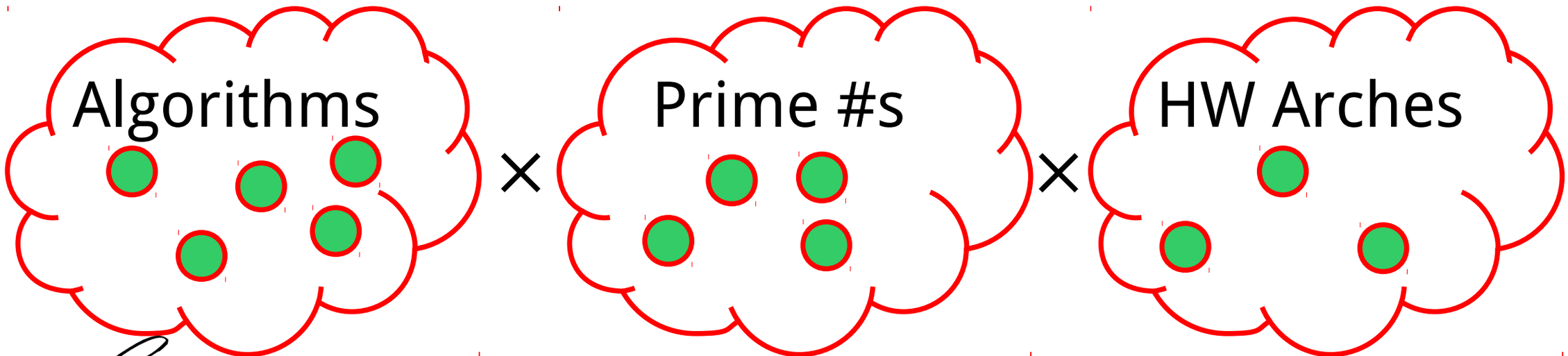
Core Protocol Logic

Awkward integration, with build processes instead of clean intra-language abstractions

What About *Embedded* DSLs?

Complaint	**Addressed?**
Yet another language	*Partly yes*, but still need to learn the semantics of the DSL, even if syntax may be standardized
Compiler is a black box	**No**!
Awkward integration	**Yes**!
Sometimes serious bugs	*Partly yes*, as we usually avoid type-safety bugs but not deeper semantic bugs

Complaints About: Using Libraries Coded by Wizards

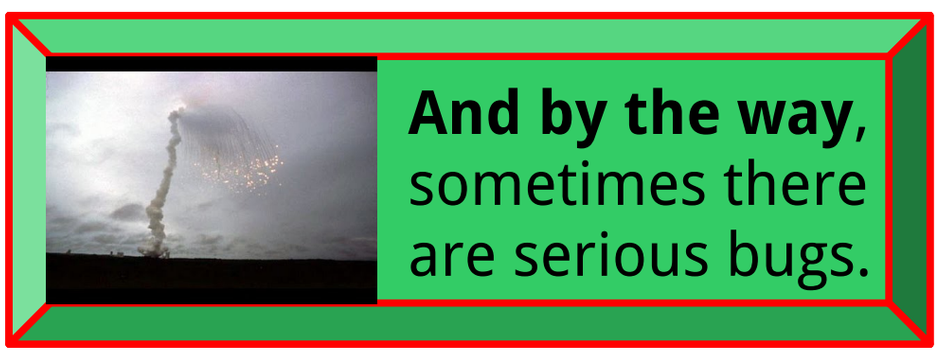


Labor-intensive adaptation, with each combination taking *at least* several days for an expert.

Library Reuse

Cryptography

rockstar coders



Rethinking the Programming Framework

Domain-Specific Notation

Semantics

Domain-Specific Notation

Semantics

Correctness bugs?
Ruled out by pervasive use of a proof assistant.

Framework

proof

proof

Semantics

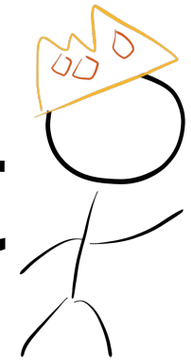
Semantics

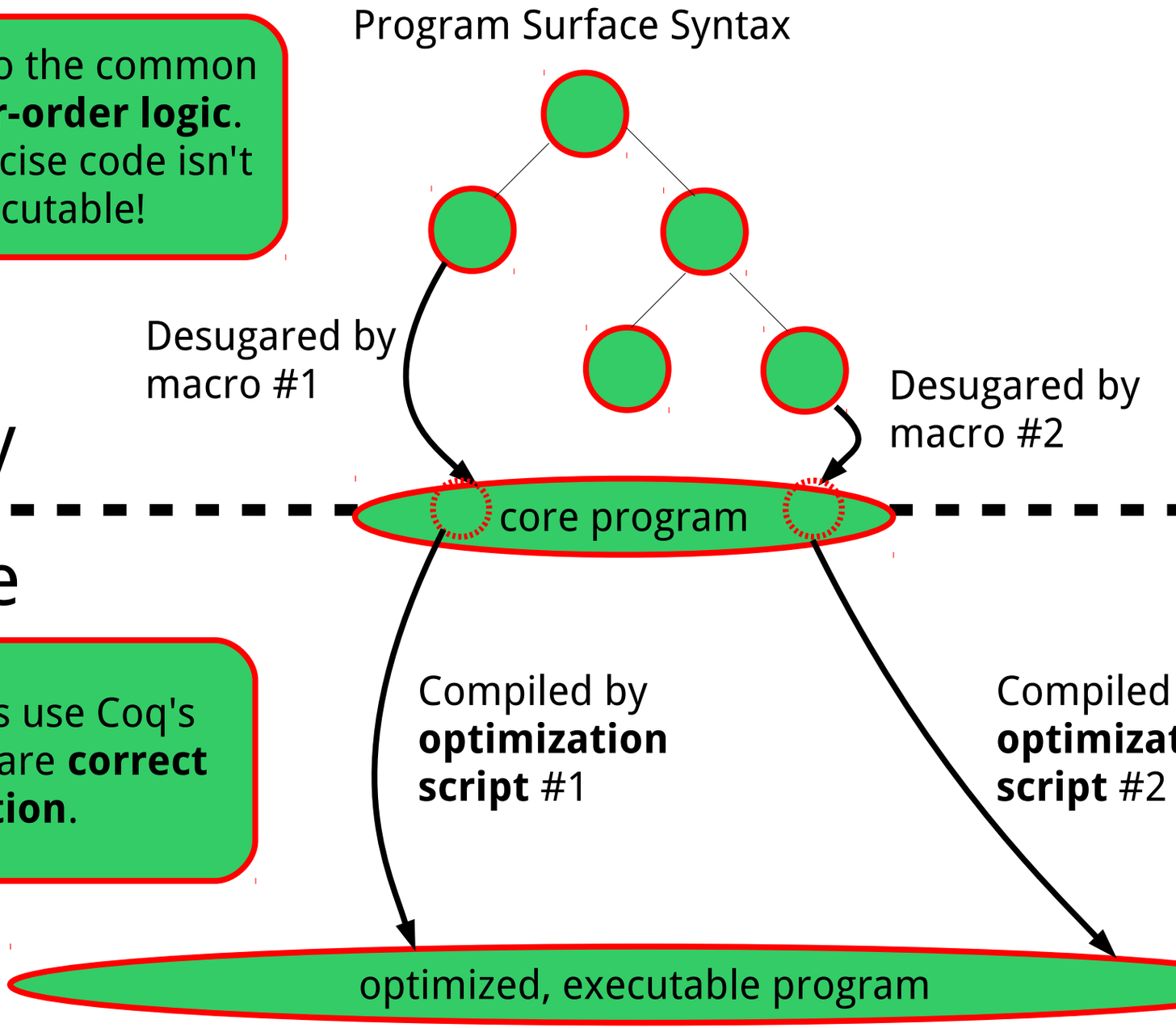
Domain-Specific Notation

Domain-Specific Notation

Macros desugar into the common language of **higher-order logic**. Often the most concise code isn't obviously executable!

Fiat





Functionality

Performance

Optimization scripts use Coq's tactic language and are **correct by construction**.

Fiat's Layers

1. **Coq**: logic and tactic language
2. **Computations**: nondeterministic functional programs
3. **Abstract data types**: encapsulated state
4. **Domains**: libraries for particular spec styles
5. **Applications**

Example Fiat Program

```
Definition BookStoreSpec : ADT BookStoreSig :=
  Eval simpl in
    Def ADT {
    rep := QueryStructure BookStoreSchema,
    Def Constructor0 "Init" : rep := empty,,
    (* … *)
    Def Method1 "NumOrders" (r : rep) (author : string) : rep * nat :=
        count <- Count (For (o in r!sORDERS) (b in r!sBOOKS)
                        Where (author = b!sAUTHOR)
                        Where (o!sISBN = b!sISBN)
                        Return ()); ret (r, count)
  }%methDefParsing.
```

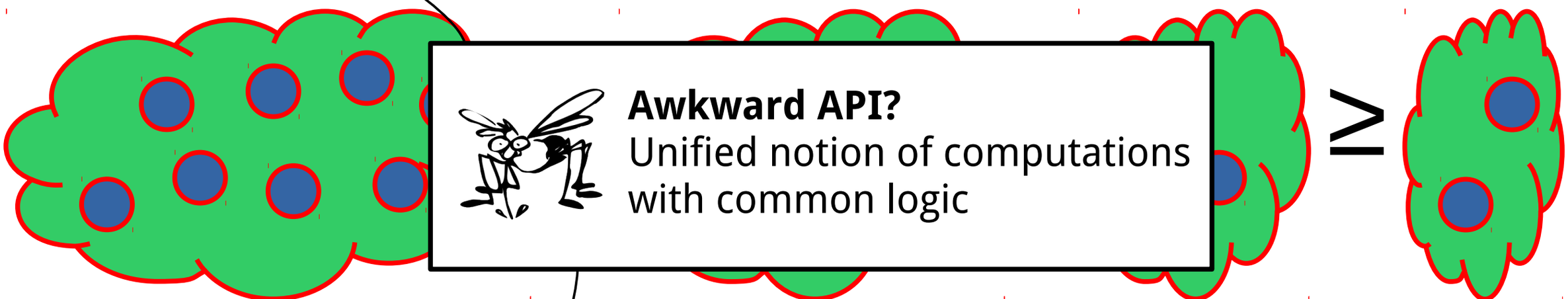
Spec

```
Theorem SharpenedBookStore :
  FullySharpened BookStoreSpec.
Proof.
  master_plan EqIndexTactics.
Defined.
```

Optimization Script

Core Fiat: **Computations** as Sets of Results

One allowable behavior, within a space of choices



Awkward API?
Unified notion of computations with common logic

Initial Spec
(nondeterministic, in logic)

Refined Spec
(some decisions made)

Functional

Imperative

Refinement (subset) relation, with Coq proofs

verified compilation to assembly

Bedrock

Computations naturally form a **monad**.

$$\mathsf{ret}\ v \stackrel{\text{def}}{=} \{v\}$$

$$x \leftarrow c_1;\ c_2(x) \stackrel{\text{def}}{=} \{v \in c_2(x) \mid x \in c_1\}$$

Example:

$x \leftarrow \{n \in \mathbb{N} \mid \exists m.\ n = 2 \times m\};$
$y \leftarrow \mathbb{N};$
$\mathsf{ret}\ (x + 2 \times y)$

In other words, choose an even number,
by some very indirect means!

Refinement of computations is just **subset**.

In other words,
resolving *nondeterminism*
is the same as
moving to a *smaller set*.

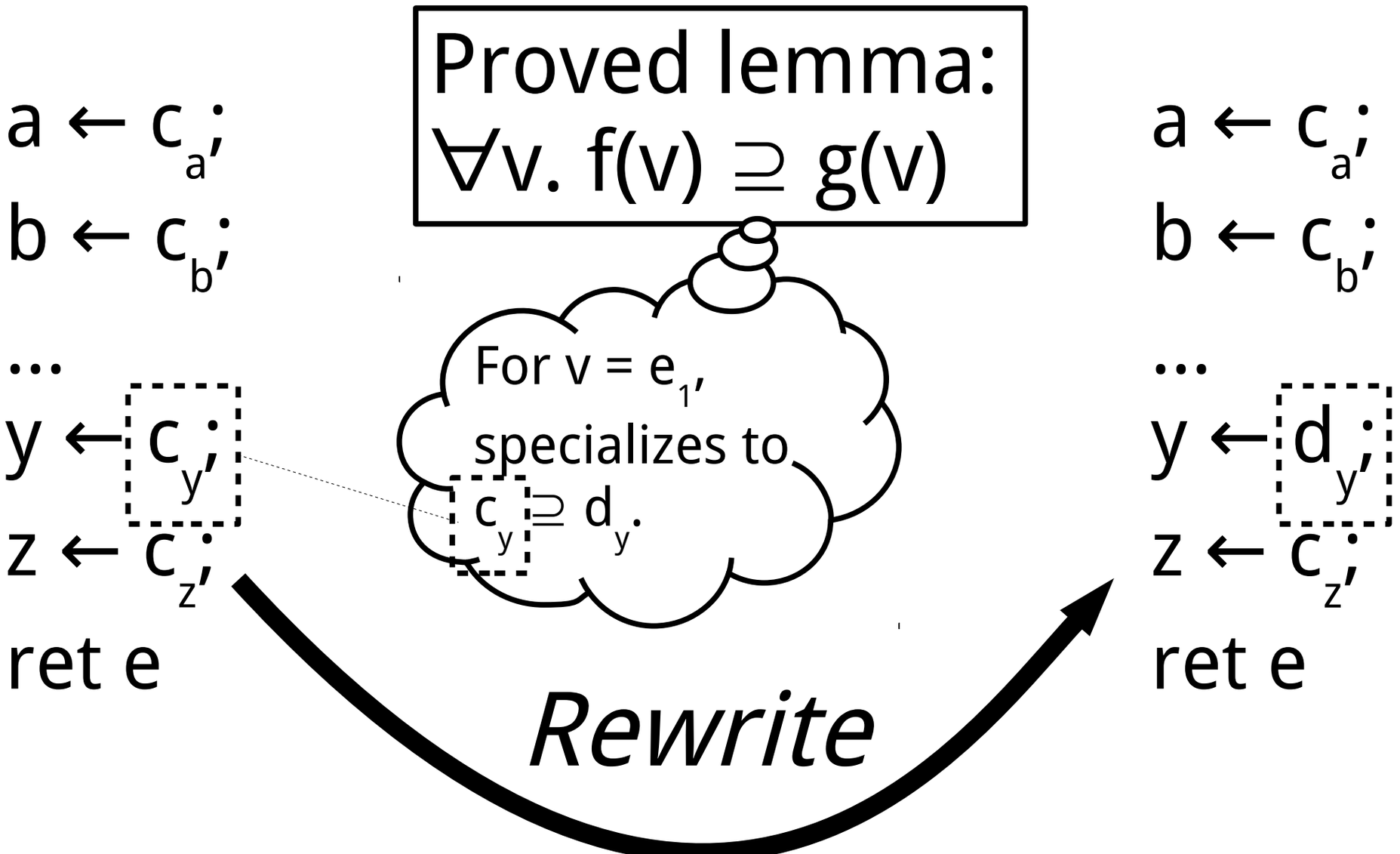
Example:

$x \leftarrow \{n \in \mathbb{N} \mid \exists m.\ n = 2 \times m\};$
$y \leftarrow \mathbb{N};$
$\mathrm{ret}\ (x + 2 \times y)$

$\supseteq\!\!/$ $\mathrm{ret}\ 42$

$\subseteq \{n \in \mathbb{N} \mid \mathrm{even}(n)\}$

Refinement is compatible with **rewriting**.



Fiat Principle #2: **Abstract Data Types** with computations for methods

Type of **private state**

ADT {
    rep = T
    method $m_1(x : D_1) : R_1 = c_1$
    method $m_2(x : D_2) : R_2 = c_2$
    …
    method $m_n(x : D_n) : R_n = c_n$
}

Method has computation as body, allowing **nondeterminism**.

Macros as Documentation

```
Def Method1 "NumOrders" (r : rep) (author : string) : rep * nat :=
    count <- Count (For (o in r!sORDERS) (b in r!sBOOKS)
                    Where (author = b!sAUTHOR)
```

Yet another language?
Readable macros desugaring into a common language

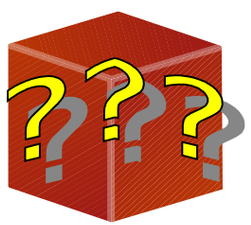
```
Count b ≡
results ← b;
return length(results)
```

```
return [a]
```

```
Where P b ≡
{l | P → l ∈ b ∧ ¬P → l = []}
```

Ingredients for Optimization Scripts

```
filter (λ(x, _) → f(x)) (join l1 l2)
= join
```



Compiler a black box?
Easy to add new
optimization rules w/ proofs

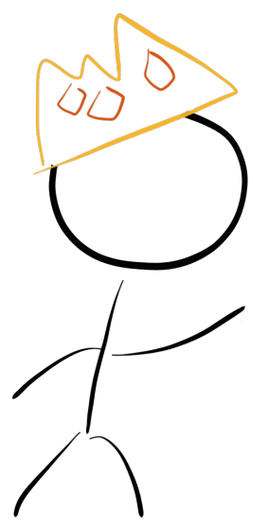
```
filter (λ(k, v) → k = k₀) (BST.enumerate t)
= BST.lookup t k₀
```

Separate **functionality** and **performance**

Functionality as macros desugaring to a common logic

Performance via proved optimization rules

`http://plv.csail.mit.edu/fiat/`



Work supported by:

I2O: HACMS and BRASS programs

National Science Foundation