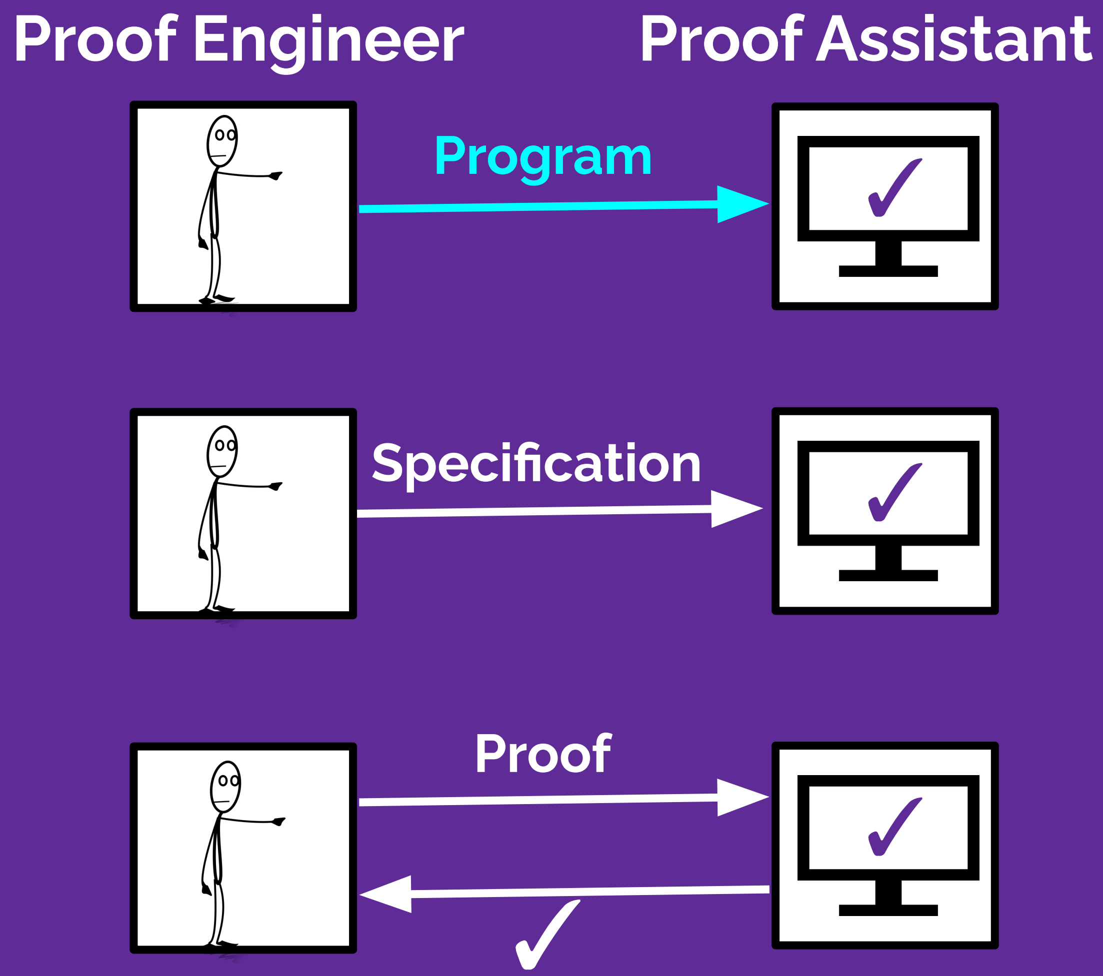
Language Models for Formal Proof

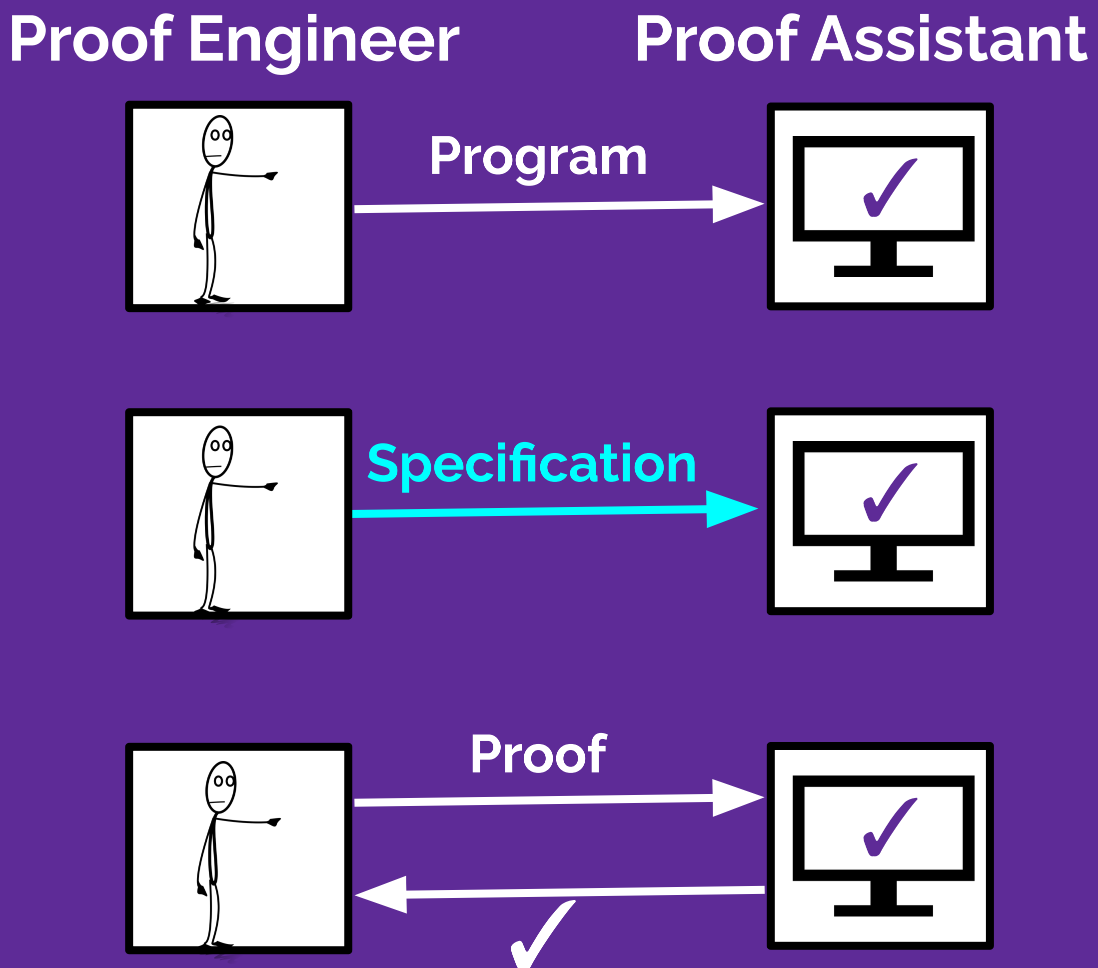
Talia Ringer
UIUC Computer Science

Proof Assistants

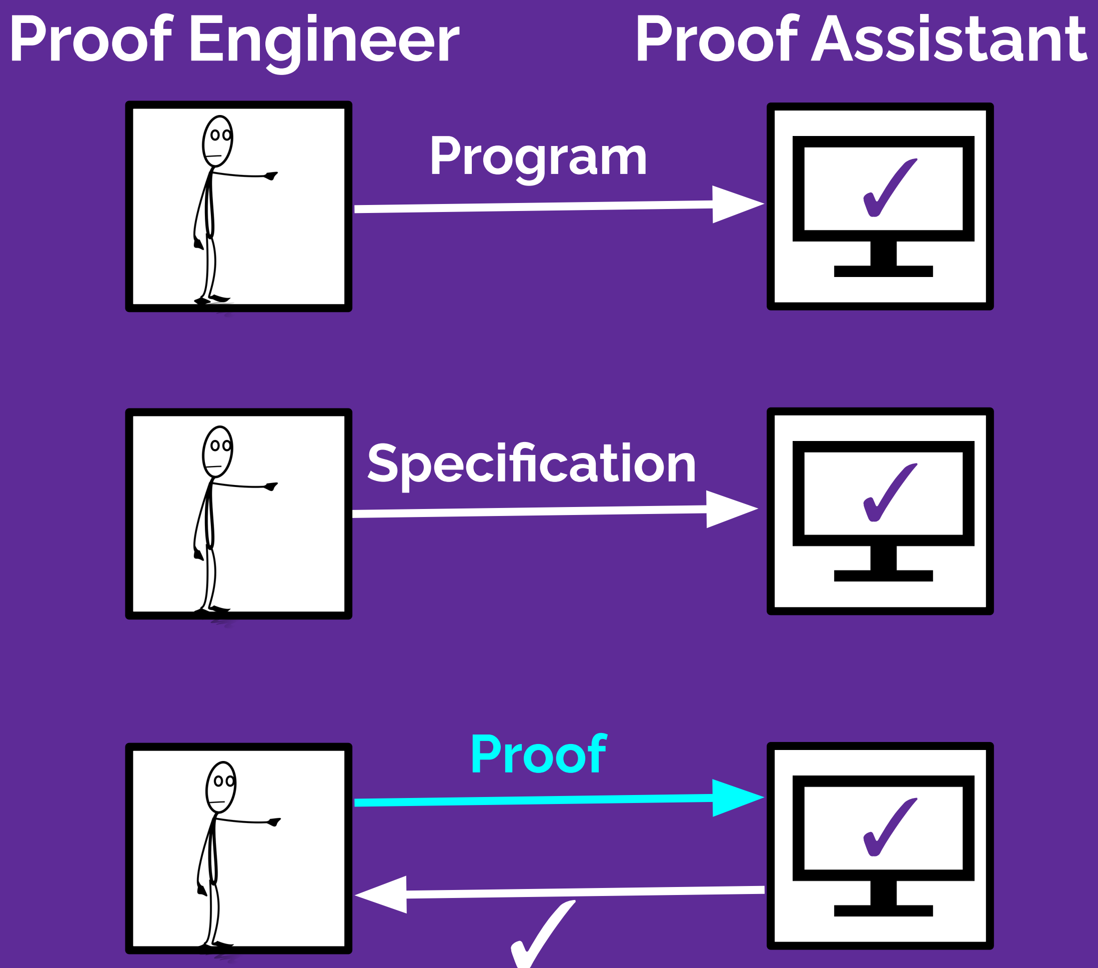
Proof Assistants



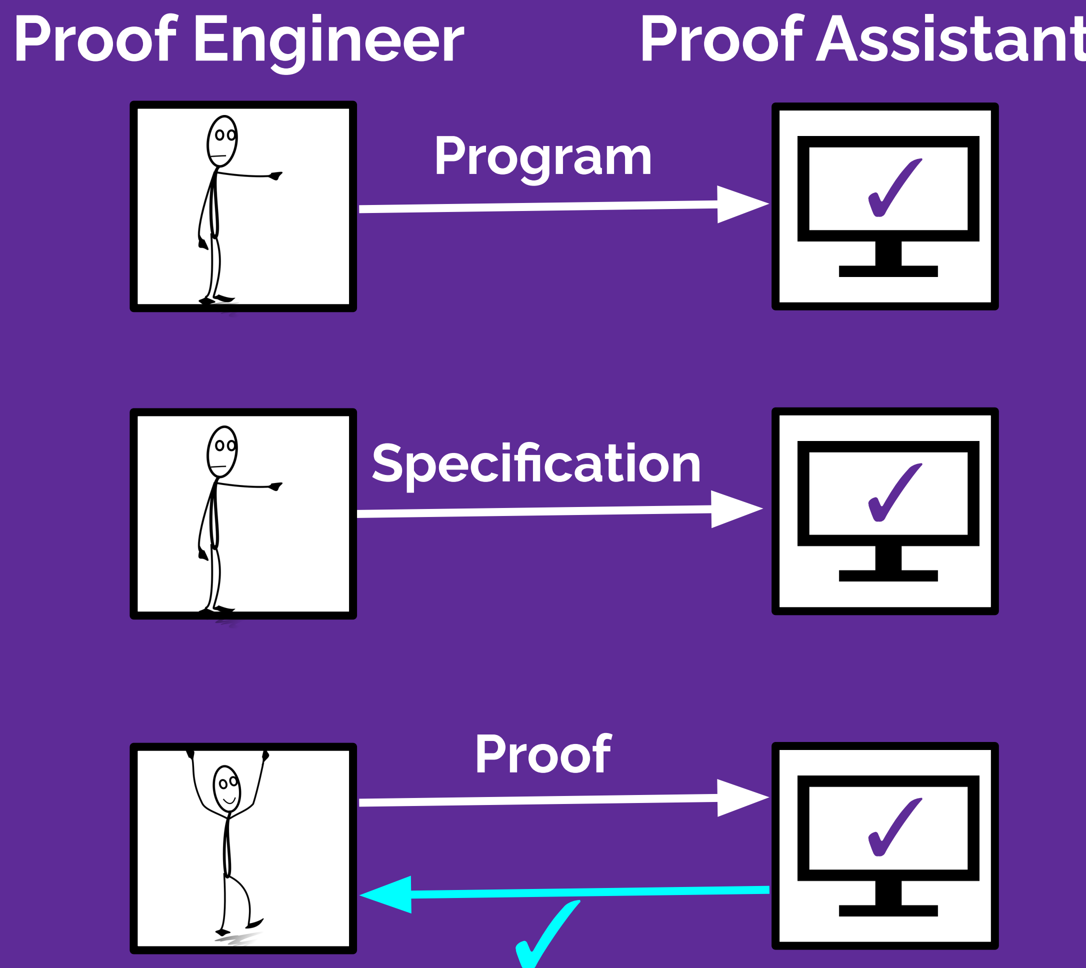
Proof Assistants



Proof Assistants

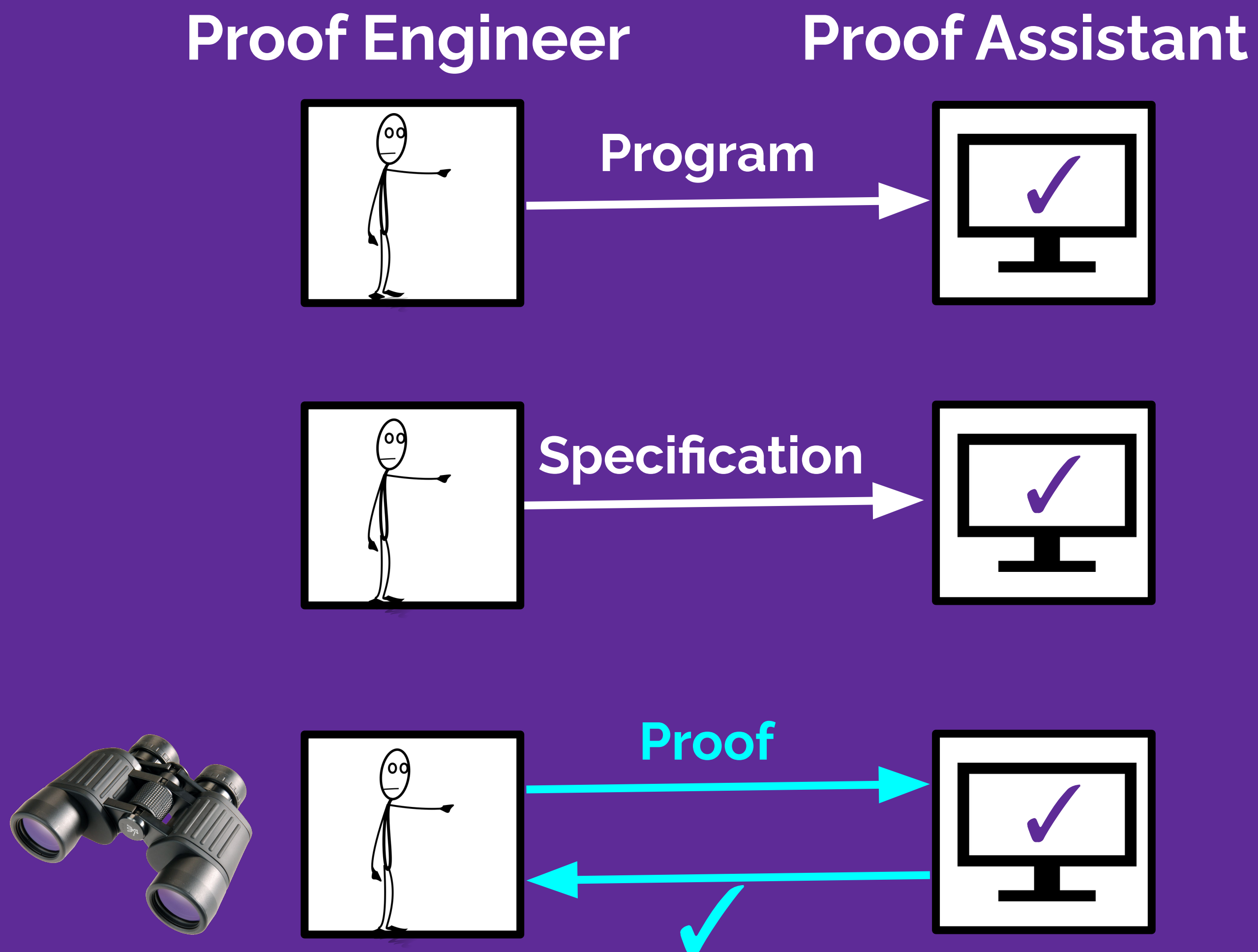


Proof Assistants

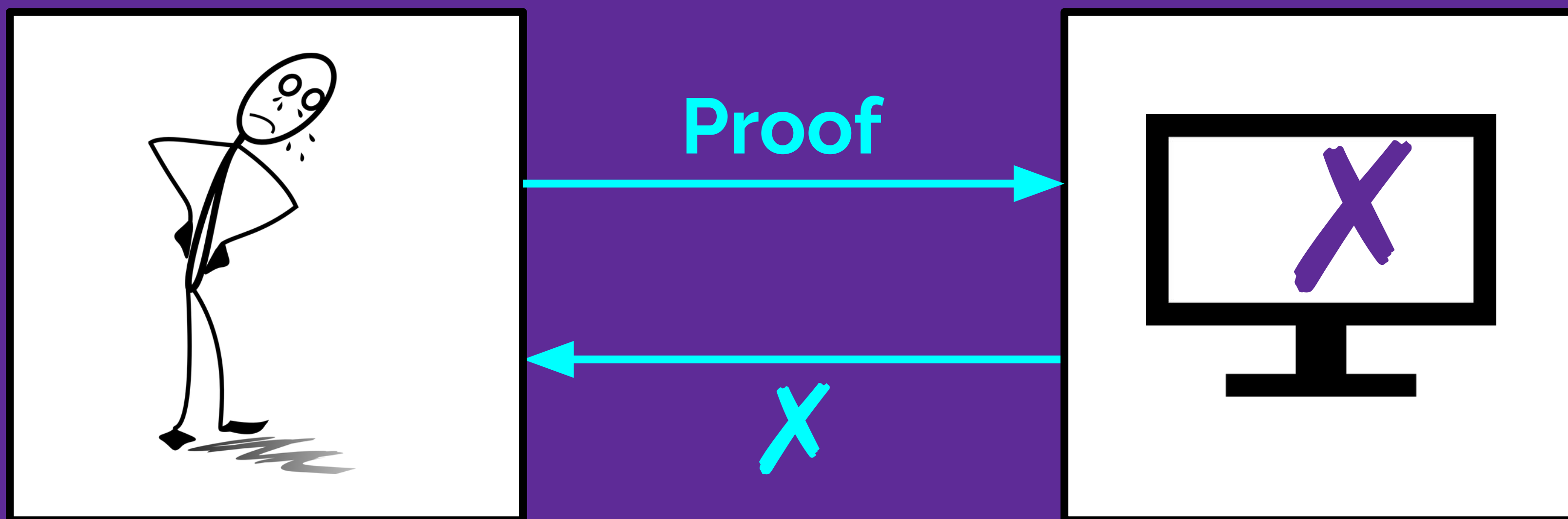


Then vs. Now

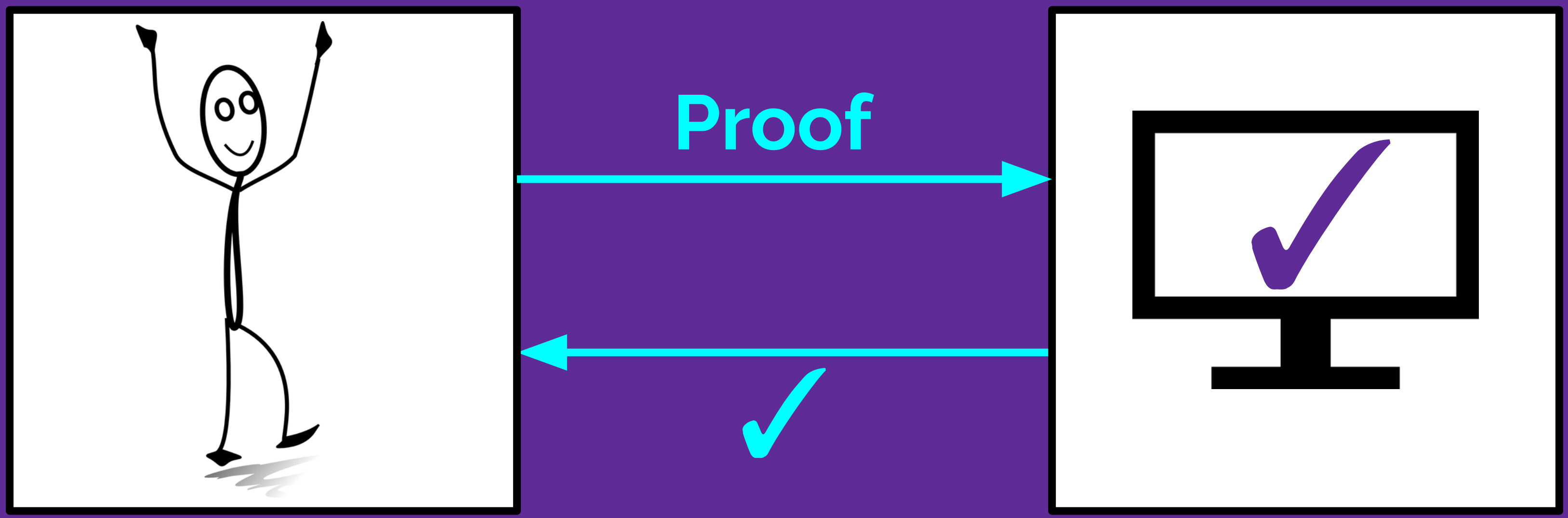
Then vs. Now


Proof Engineer
Proof Assistant
Program
Specification
Proof

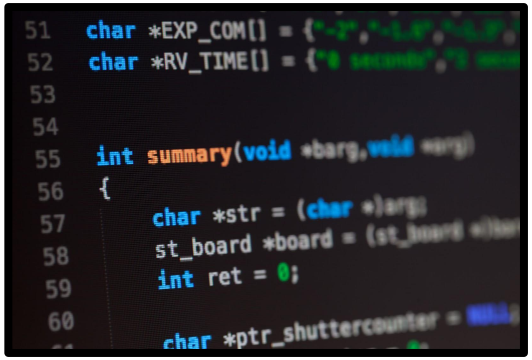
Then vs. Now



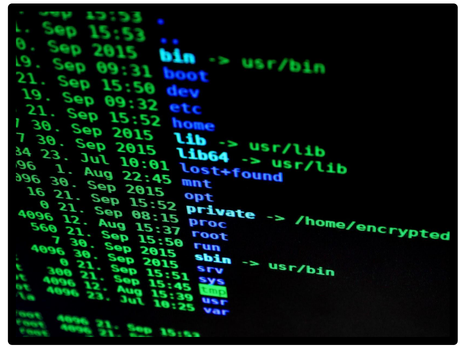
Then vs. Now



Then vs. Now

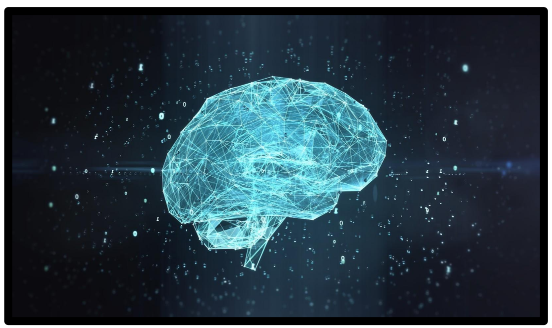


Compilers



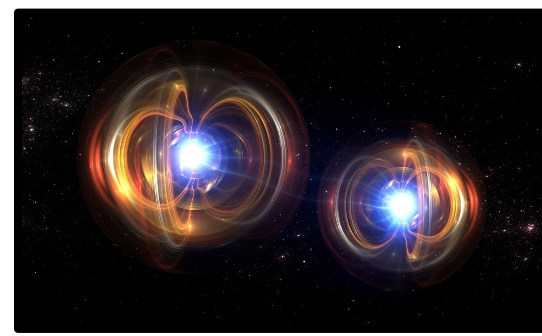
File Systems

Web Browsers



Machine Learning Systems

Operating Systems

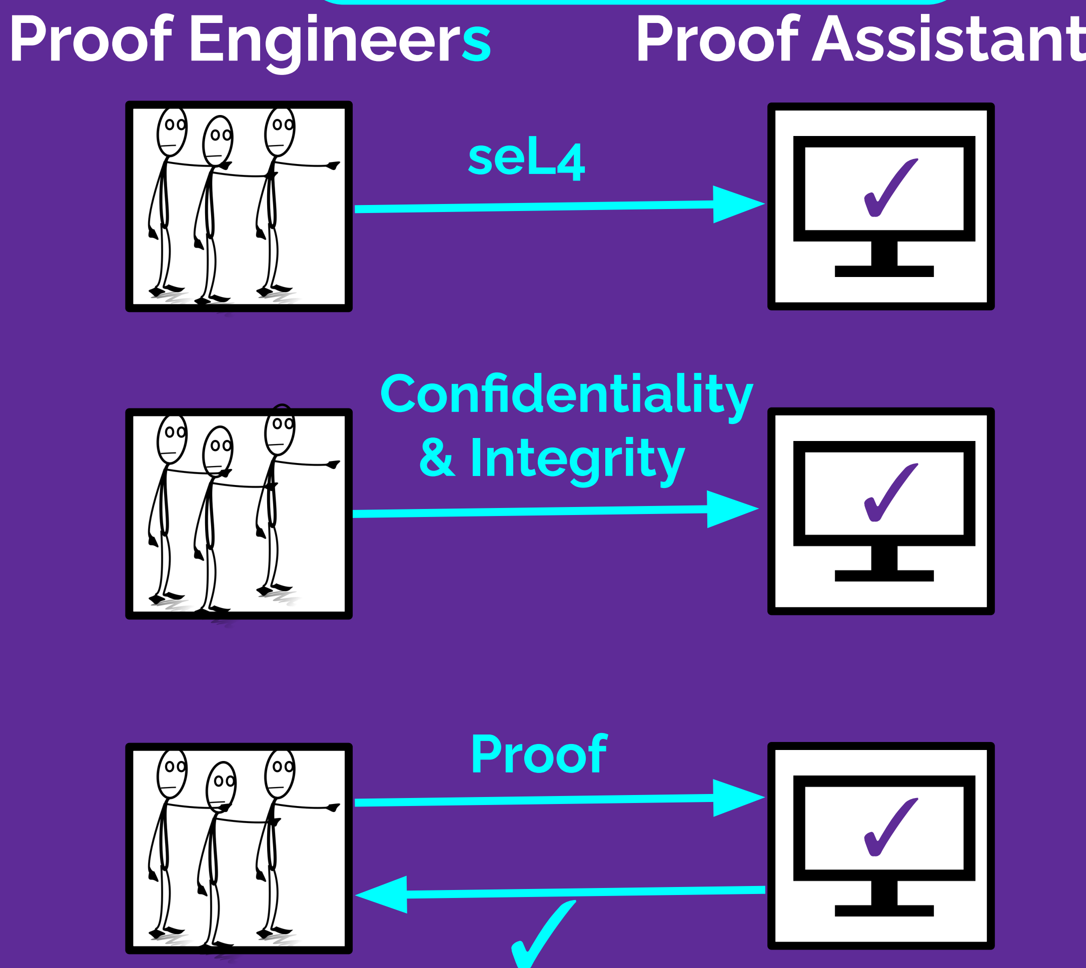


Quantum Optimizers

Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric and Zachary Tatlock (2019), QED at Large: A Survey of Engineering of Formally Verified Software, Foundations and Trends in Programming Languages: Vol. 5, No. 2-3, pp 102–281.

Then vs. Now

20+ person-years
~1,000,000 LOP



Proof automation makes it easier to develop and maintain verified systems using proof assistants.

Traditional automation:

+ predictable
+ dependable
+ understandable
- limited in scope
- takes expertise to extend

Language models:

- *un*predictable
- *not* dependable
- *not* understandable

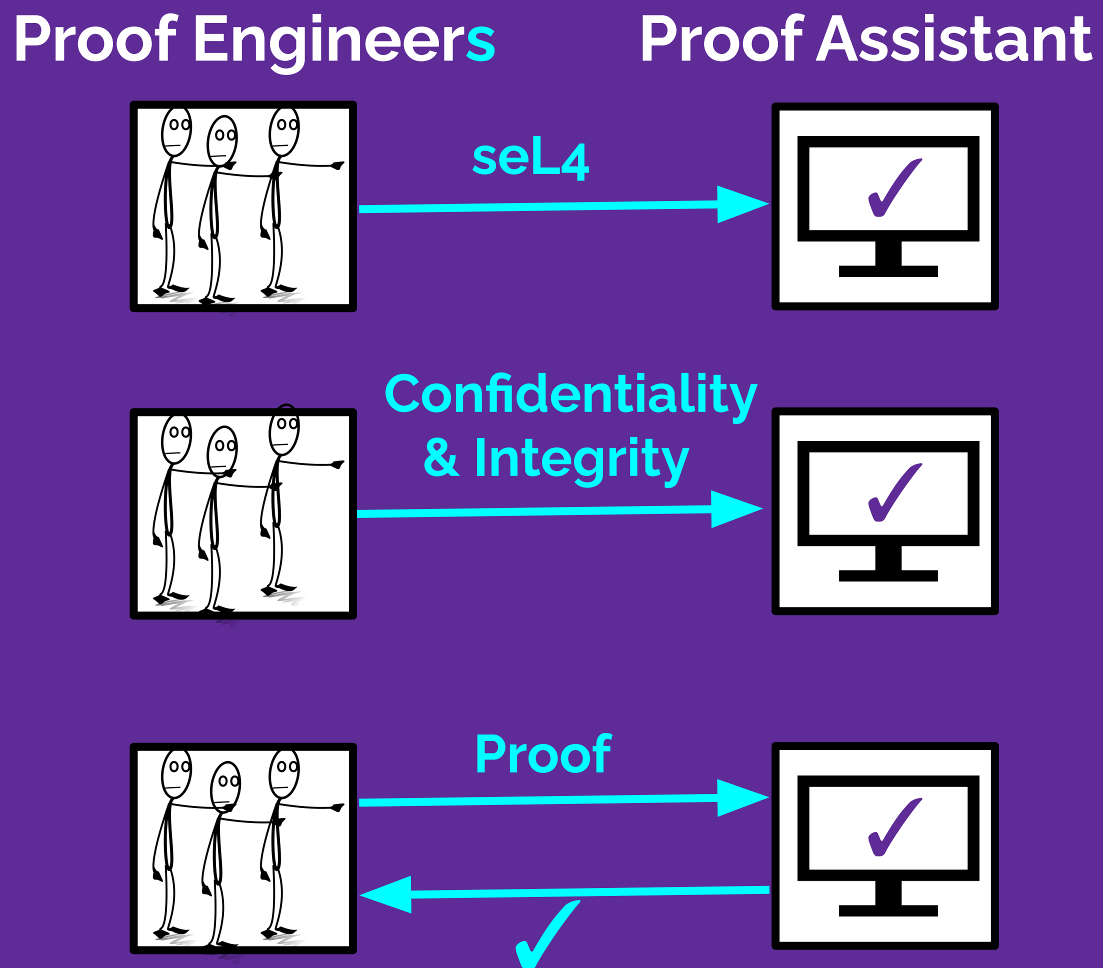
\+ *not very* limited in scope
\+ takes *little* expertise to extend

Best of both worlds?

+ predictable
+ dependable
+ understandable
+ *not very* limited in scope
+ takes *little* expertise to extend

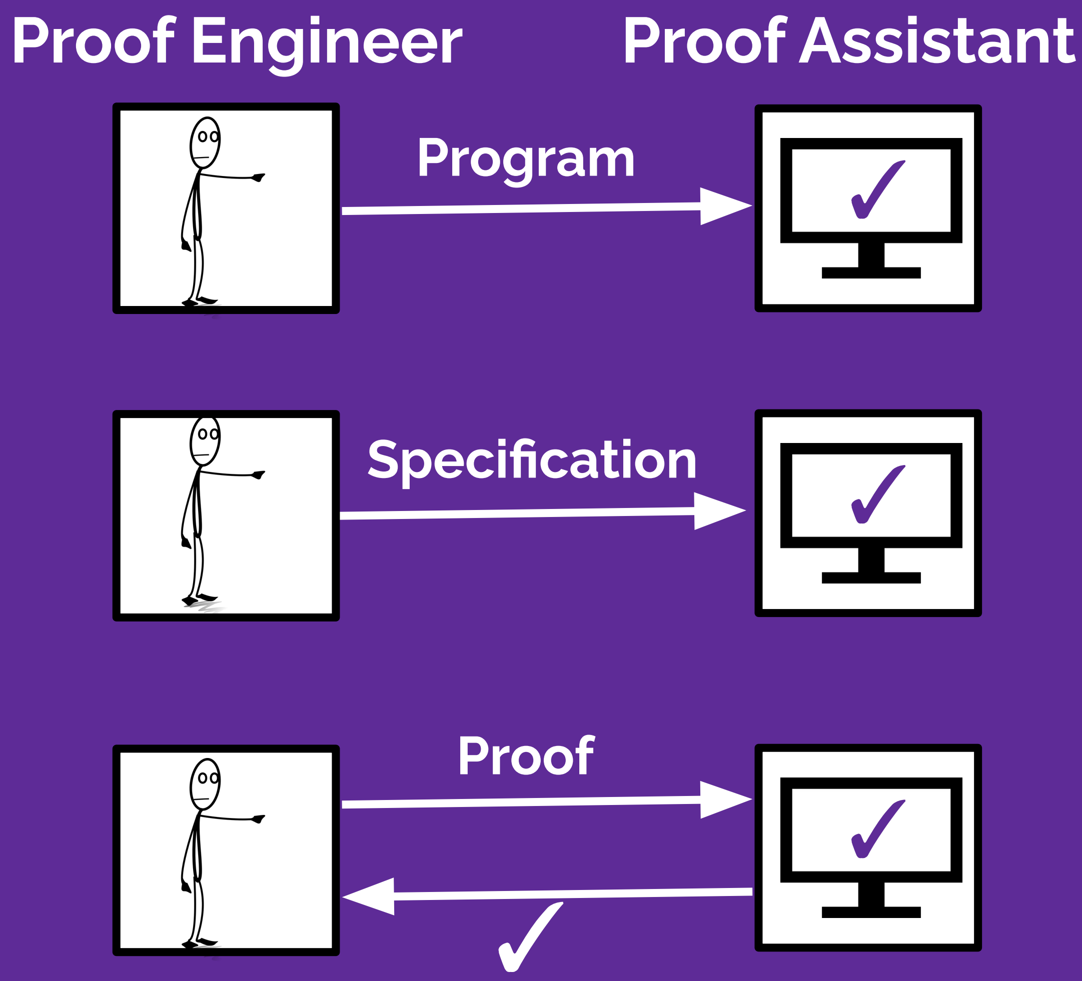
Now vs. **Future**

Not that much work, lots of help?



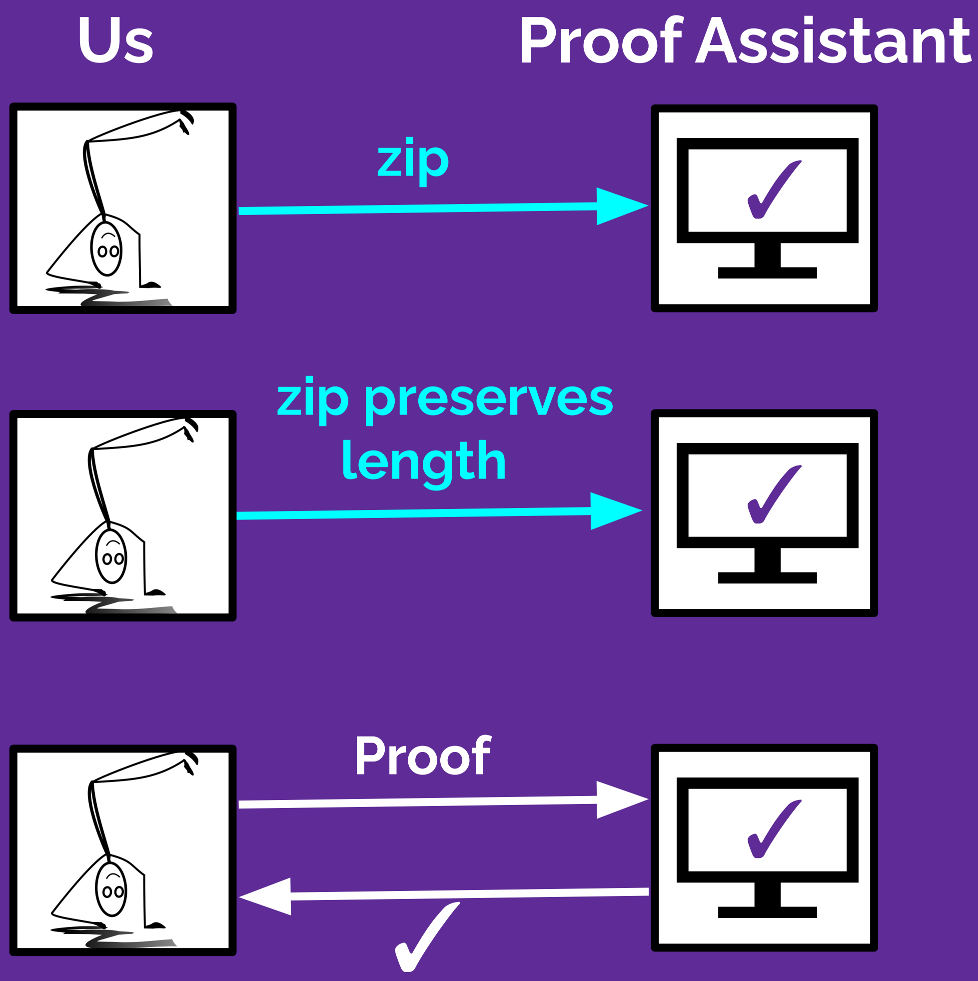
1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities



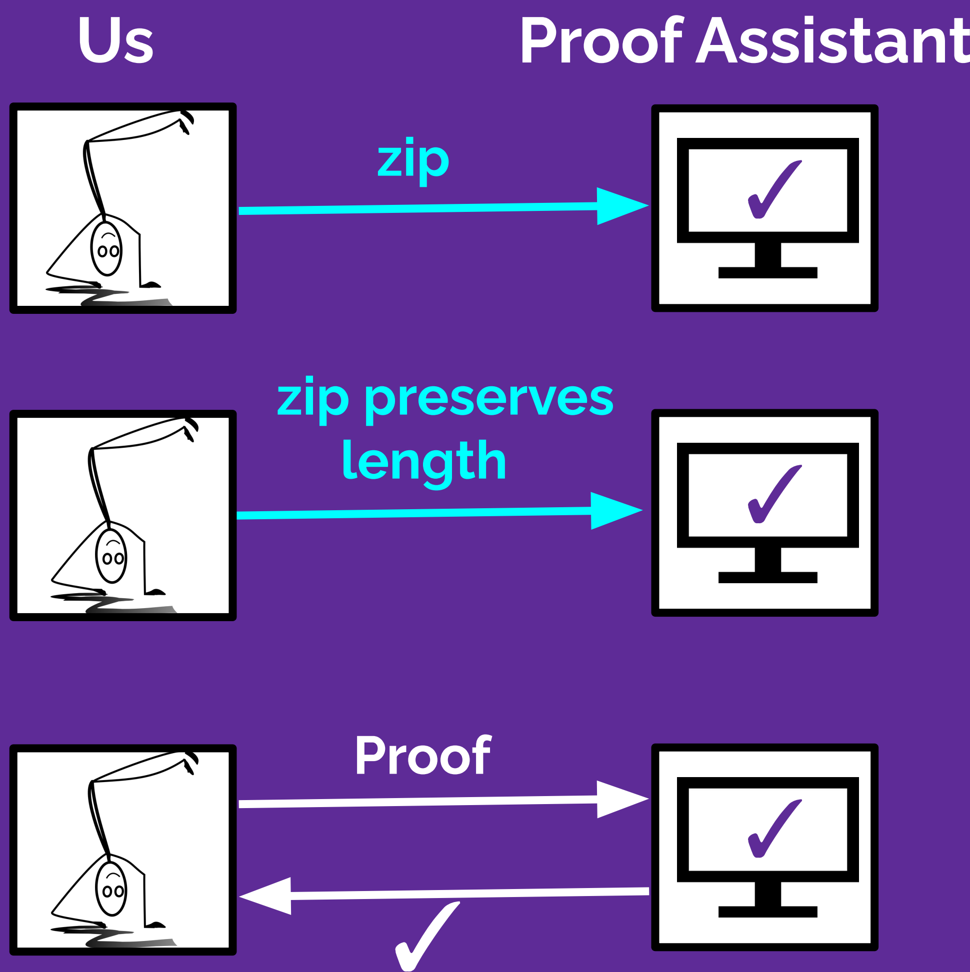
Proof Assistants (Part 1 of 5)

List Zip Preserves Length

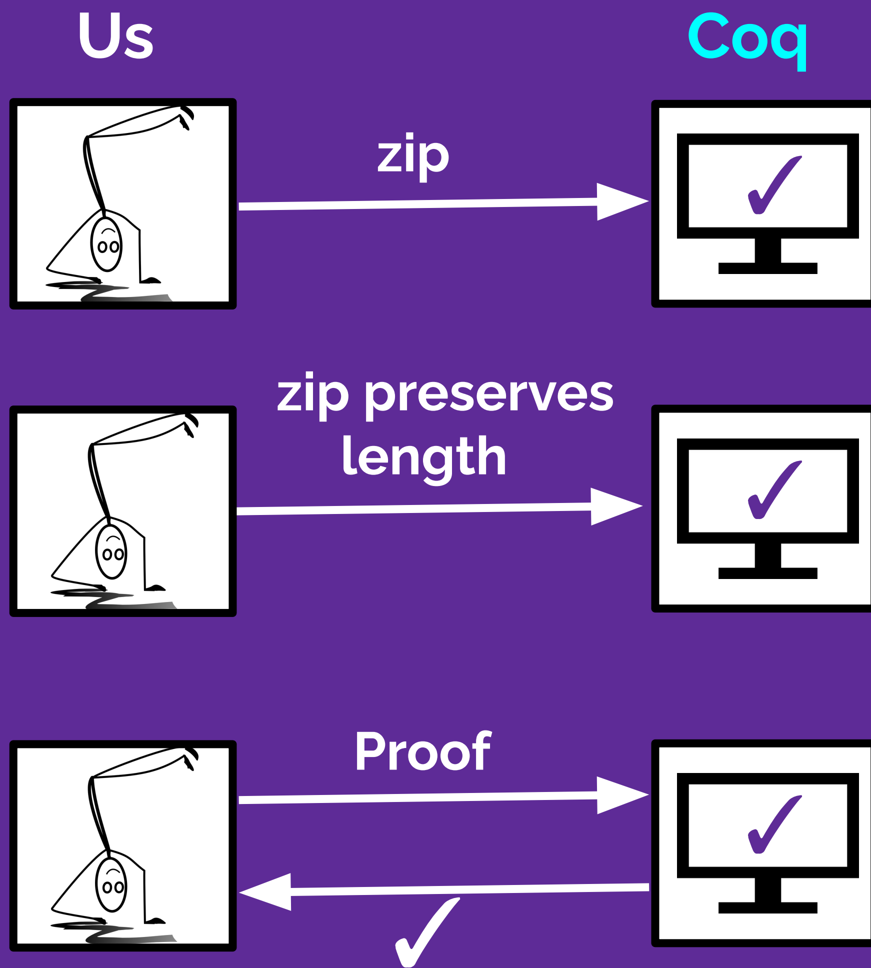


Proof Assistants (Part 1 of 5)

List Zip Preserves Length

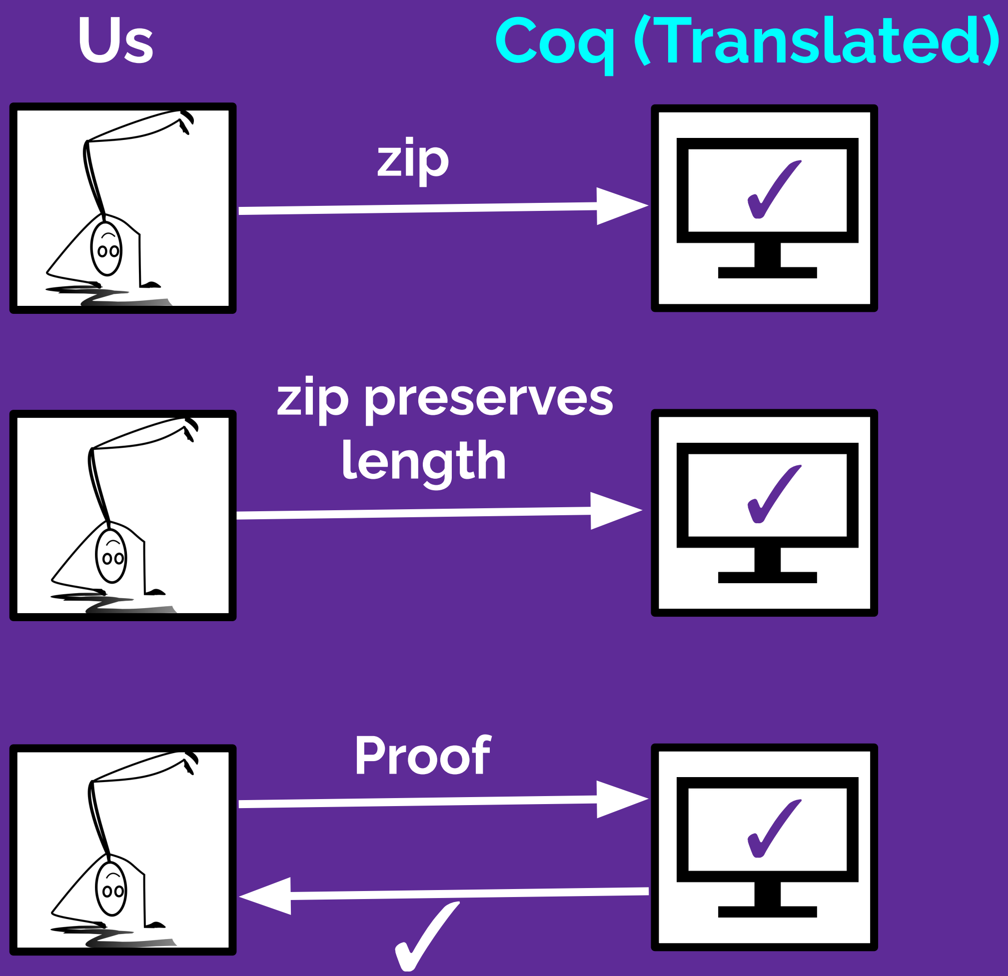


List Zip Preserves Length



Proof Assistants (Part 1 of 5)

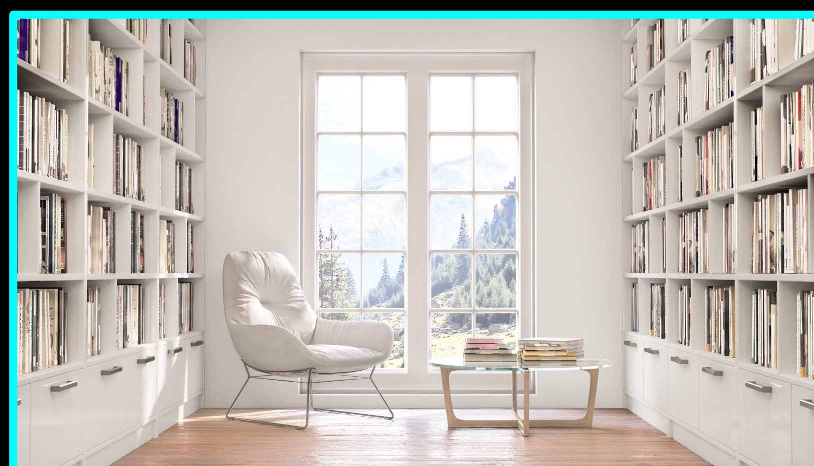
List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
list <T> :=
  | [ ] : list <T>
  | cons : T → list <T> → list <T>
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

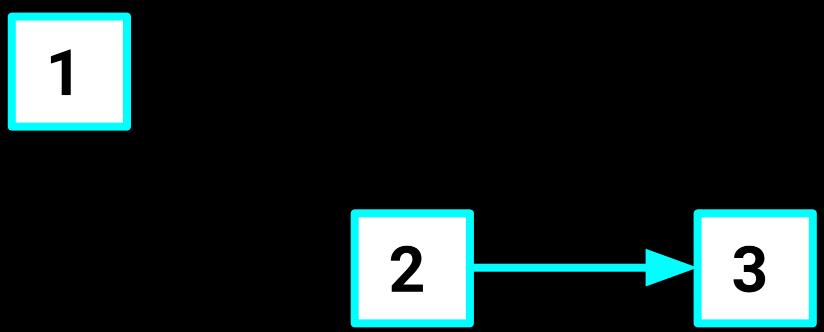
```
list <T> :=
  | [ ] : list <T>
  | cons : T → list <T> → list <T>
```

List Zip Preserves Length

list <T> :=
| **[ ] : list <T>**
| cons : T → list <T> → list <T>

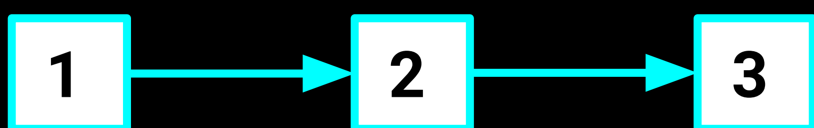
List Zip Preserves Length

list <T> :=
  | [ ] : list <T>
  | cons : **T → list <T>** → list <T>



List Zip Preserves Length

list <T> :=
  | [ ] : list <T>
  | **cons :** T → list <T> → **list <T>**



List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=
  if l = [ ] then
    0
  else
    1 + length (tail l)
```

Proof Assistants (Part 1 of 5)

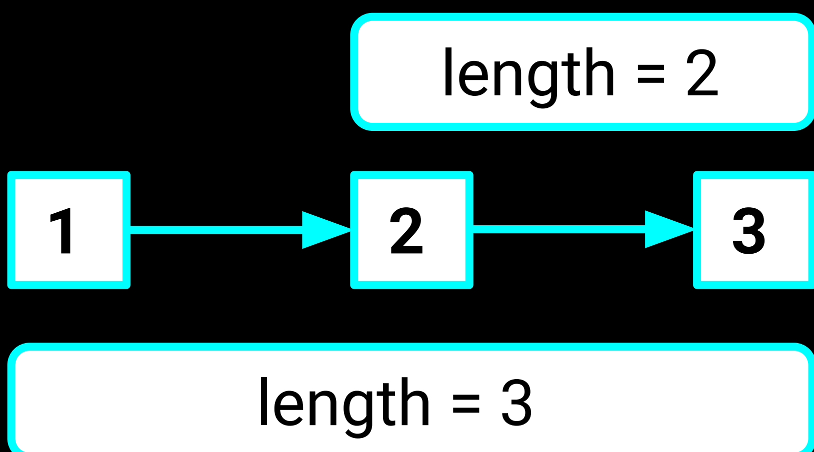
List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=
  if l = [ ] then
    0
  else
    1 + length (tail l)
```

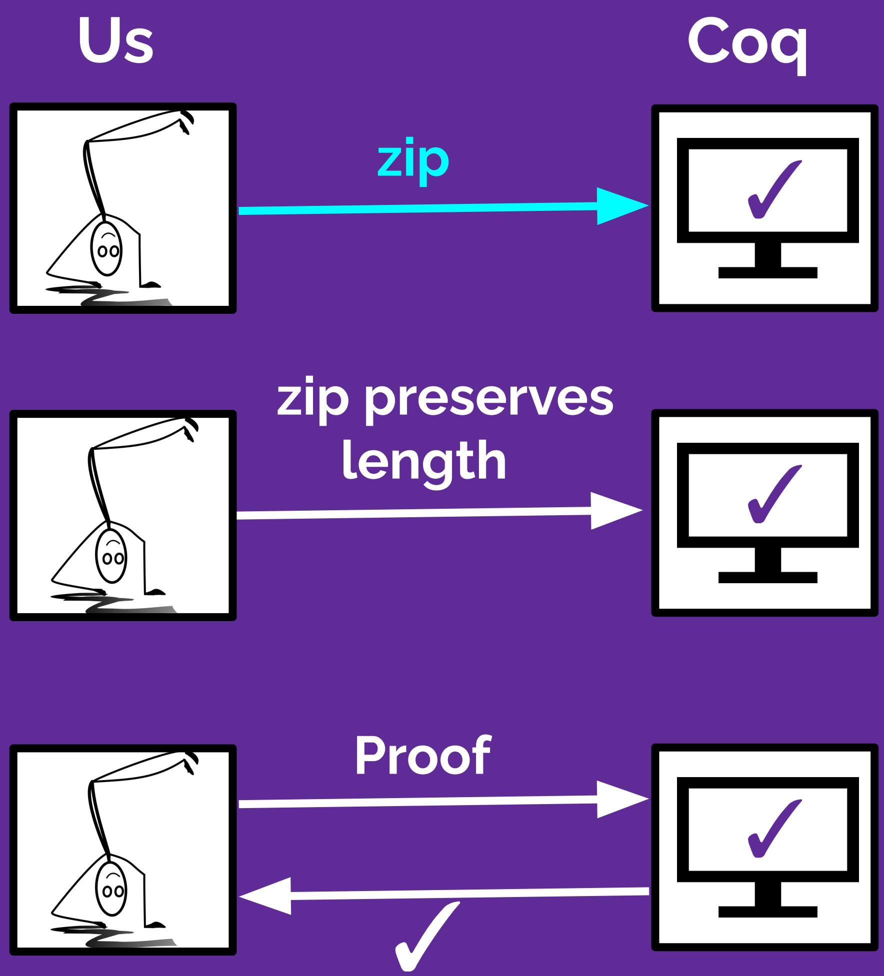
Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=
  if l = [ ] then
    0
  else
    1 + length (tail l)
```



List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

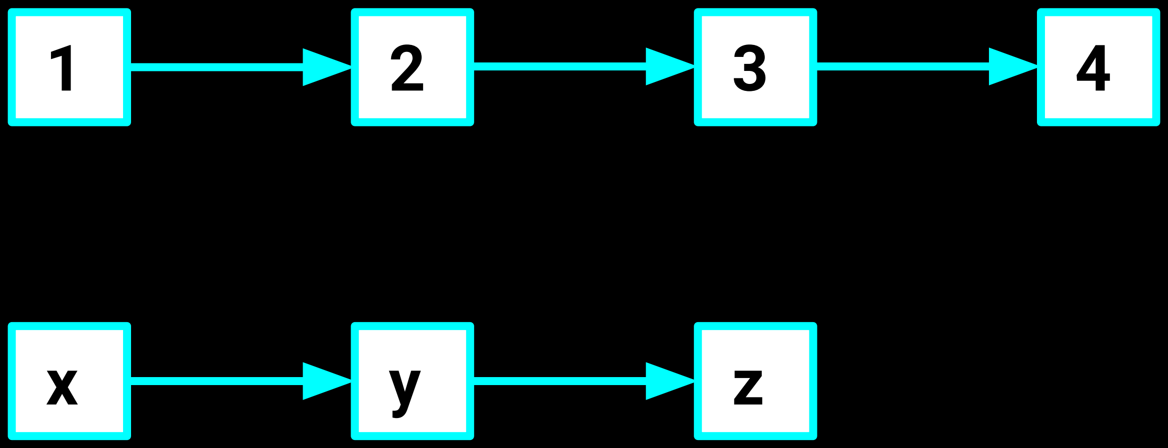
Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

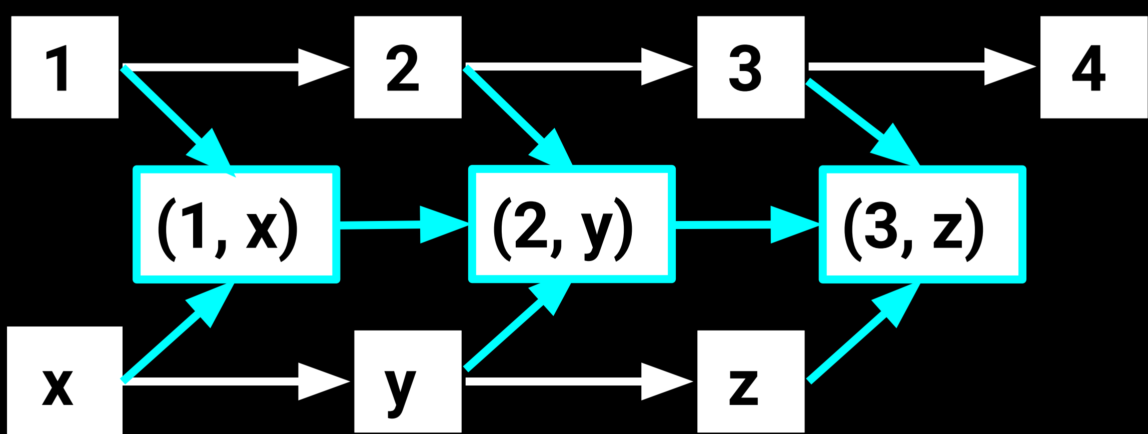
List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



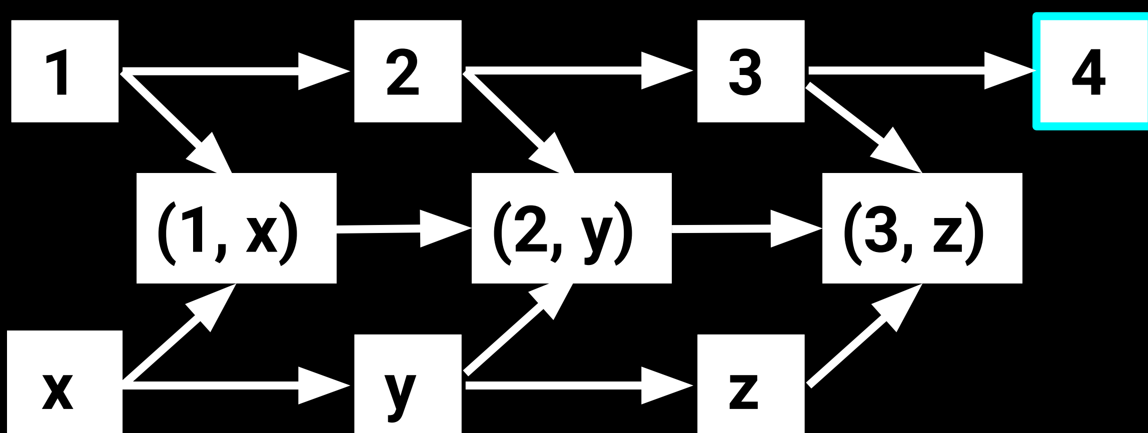
List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



List Zip Preserves Length

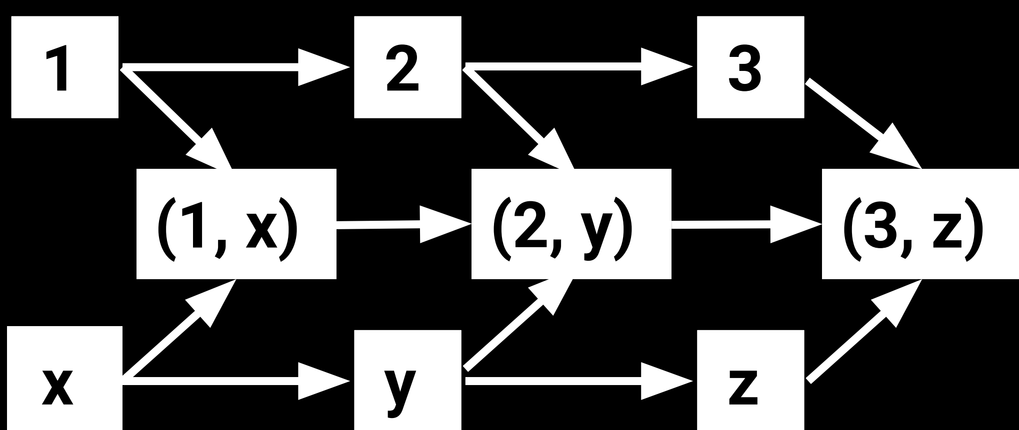
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

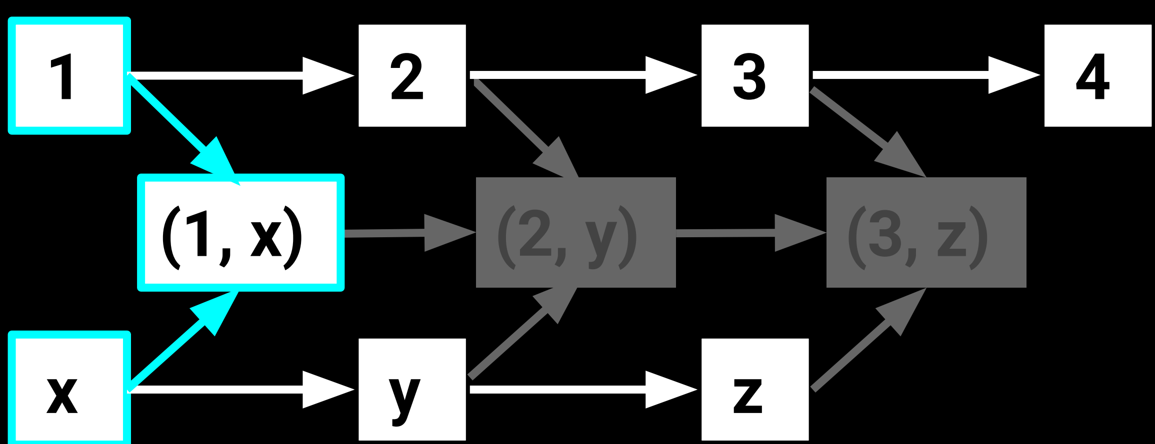


List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

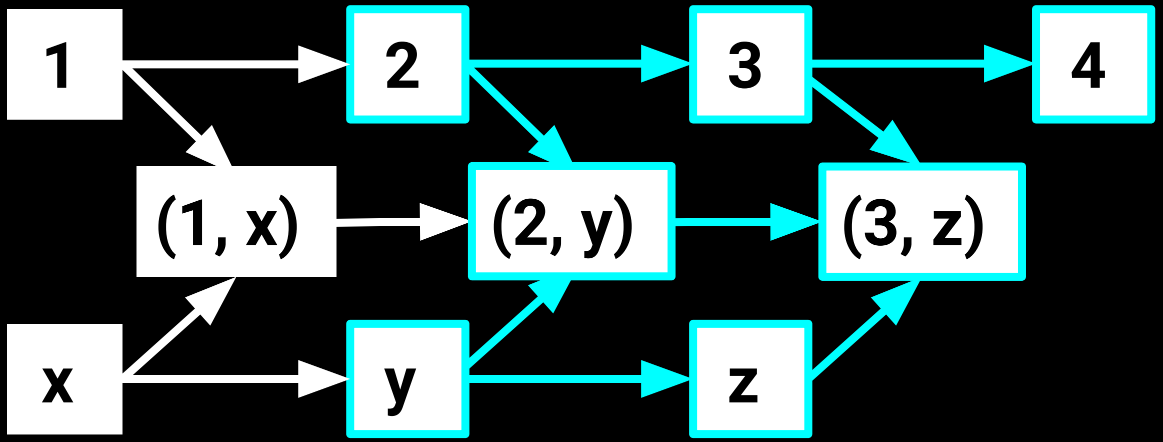
List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



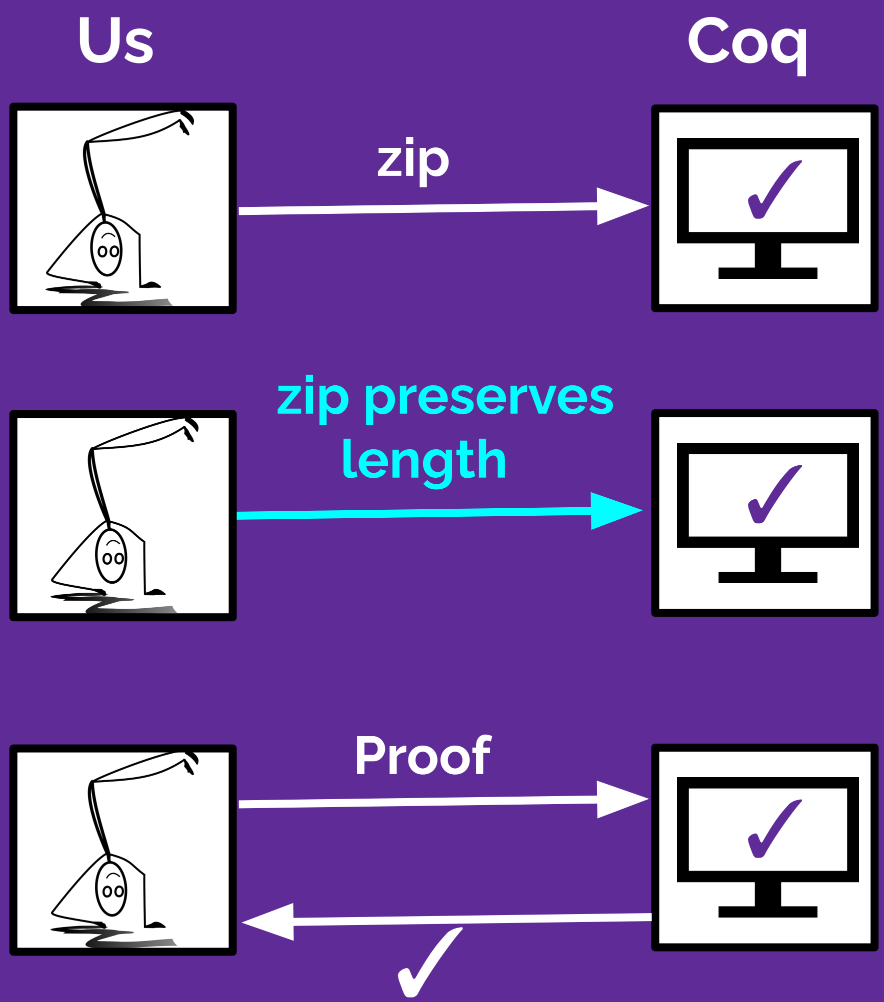
List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=
  if l1 = [ ] or l2 = [ ] then
    [ ]
  else
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Proof Assistants (Part 1 of 5)

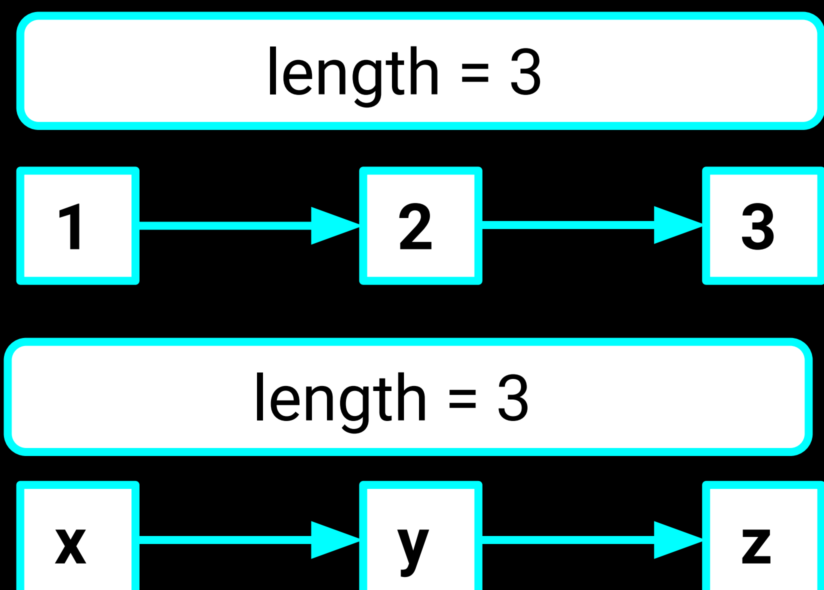
List Zip Preserves Length

```
Theorem zip_preserves_length :
  ∀ <A, B> (l1 : list <A>) (l2 : list <B>),
    length l1 = length l2 →
    length (zip l1 l2) = length l1.
```

Proof Assistants (Part 1 of 5)

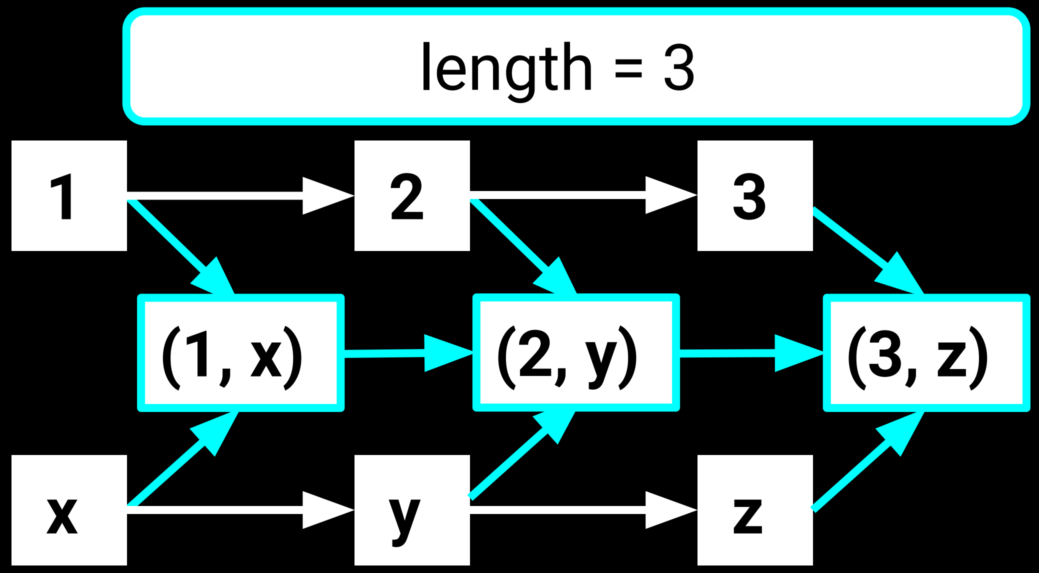
List Zip Preserves Length

```
Theorem zip_preserves_length :
  ∀ <A, B> (l1 : list <A>) (l2 : list <B>),
   length l1 = length l2 →
   length (zip l1 l2) = length l1.
```



List Zip Preserves Length

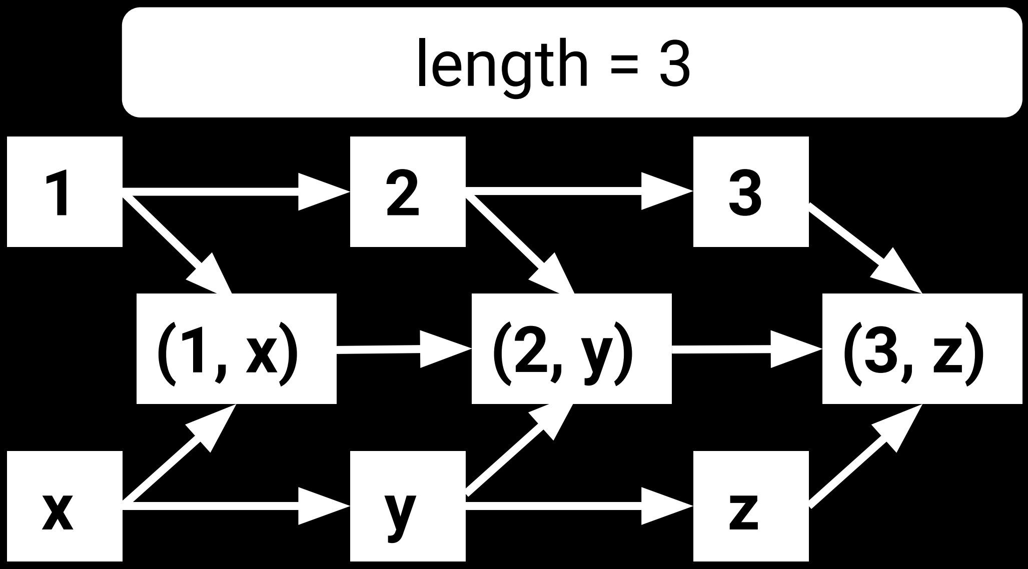
Theorem zip_preserves_length :
  ∀ <A, B> (l1 : list <A>) (l2 : list <B>),
   length l1 = length l2 →
   length (zip l1 l2) = length l1.



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

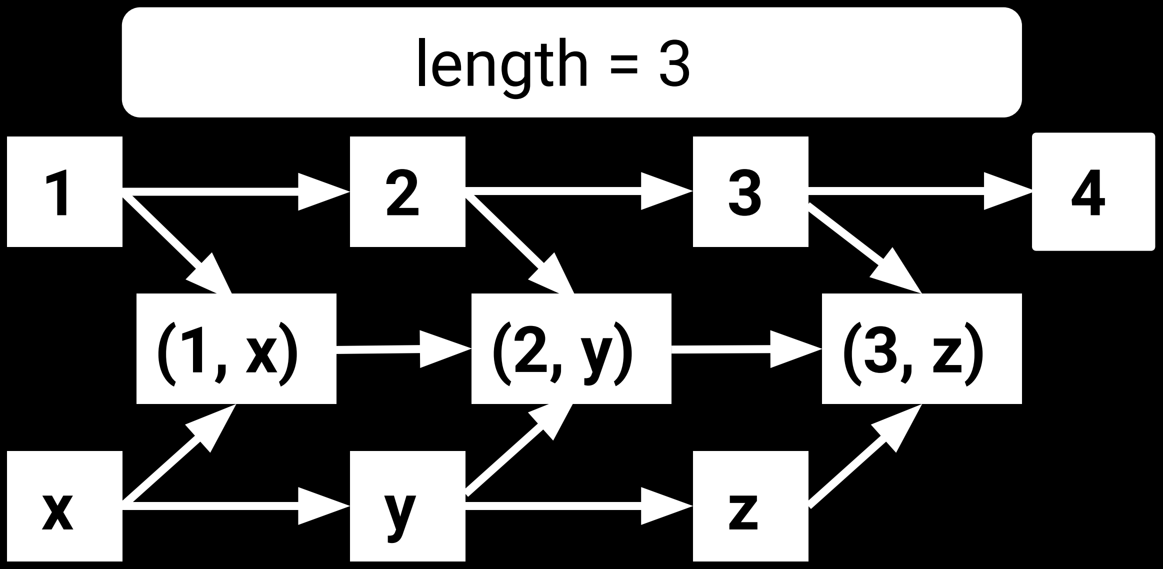
Theorem zip_preserves_length :
 ∀ <A, B> (l1 : list <A>) (l2 : list <B>),
  length l1 = length l2 →
  length (zip l1 l2) = length l1.



Proof Assistants (Part 1 of 5)

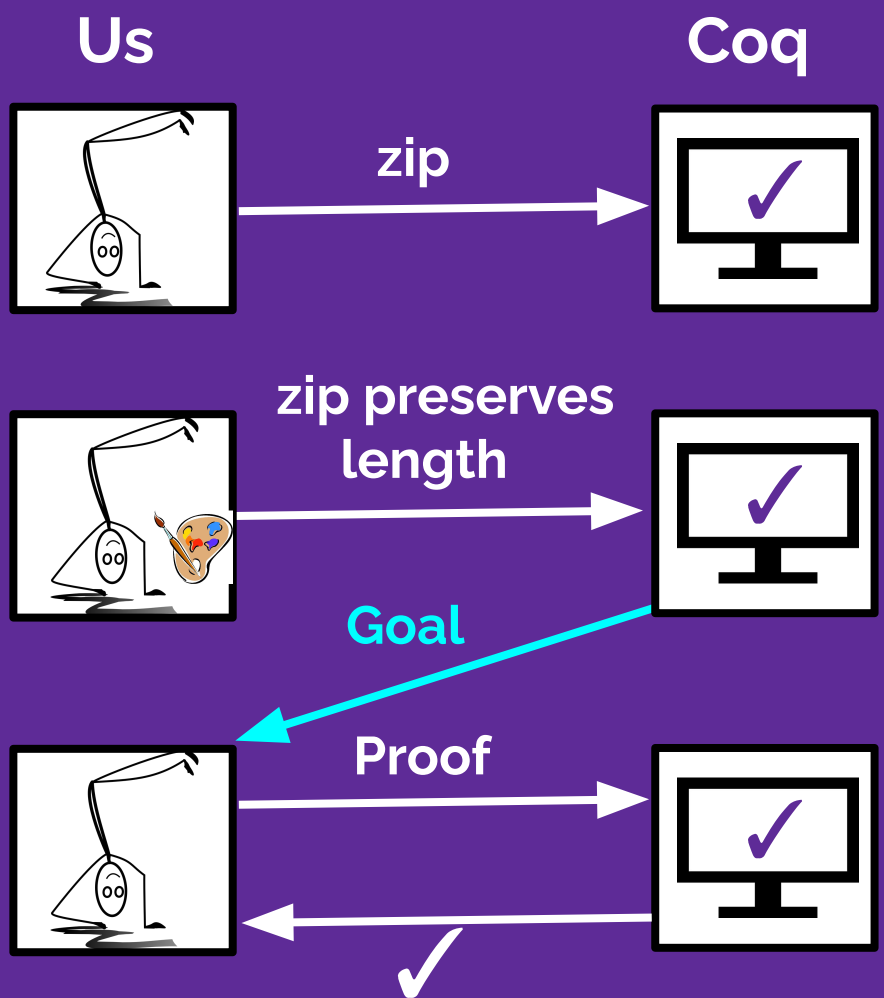
List Zip Preserves Length

Theorem zip_preserves_length :
∀ <A, B> (l1 : list <A>) (l2 : list <B>),
length (zip l1 l2) = **min** (length l1) (length l2).



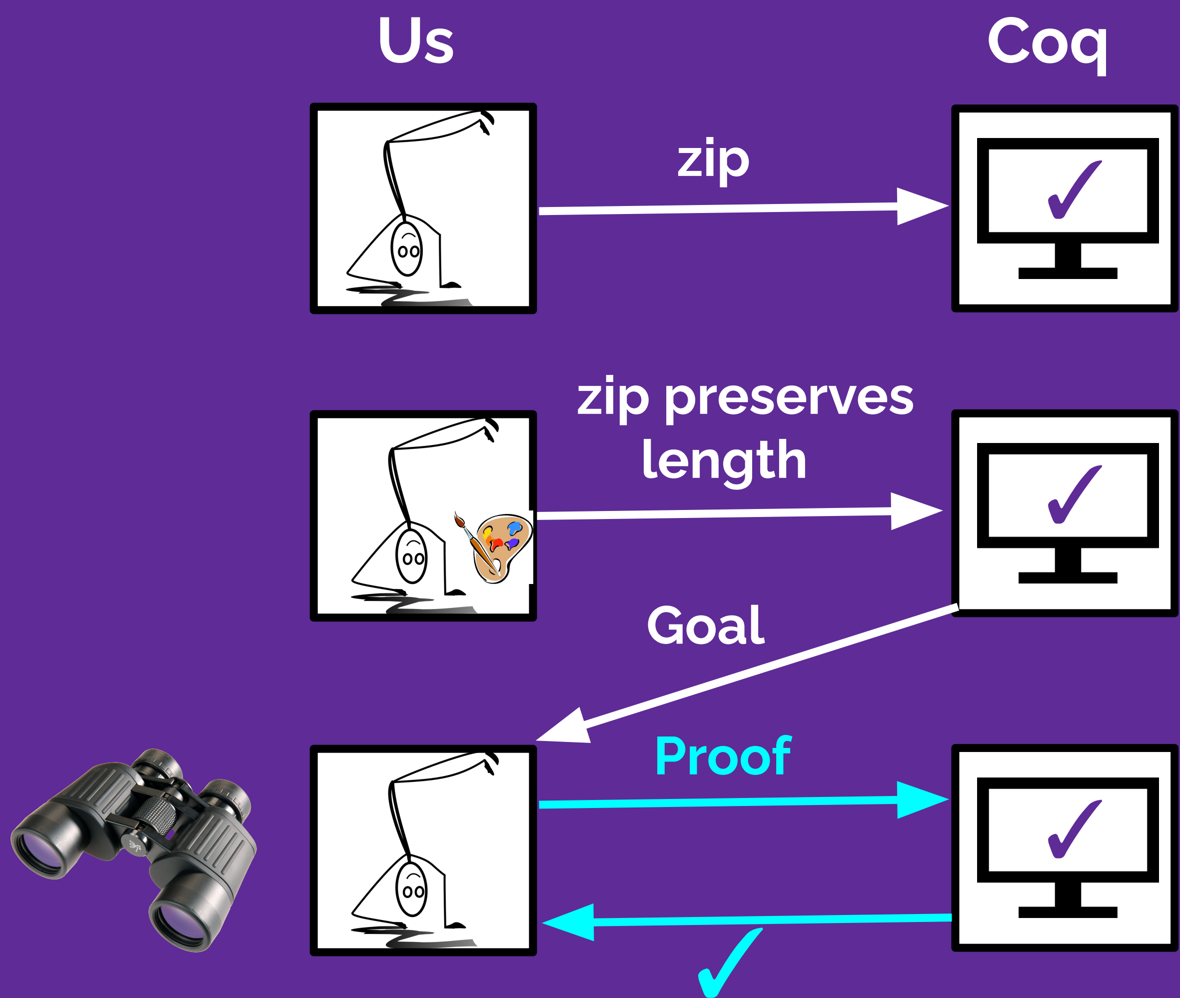
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



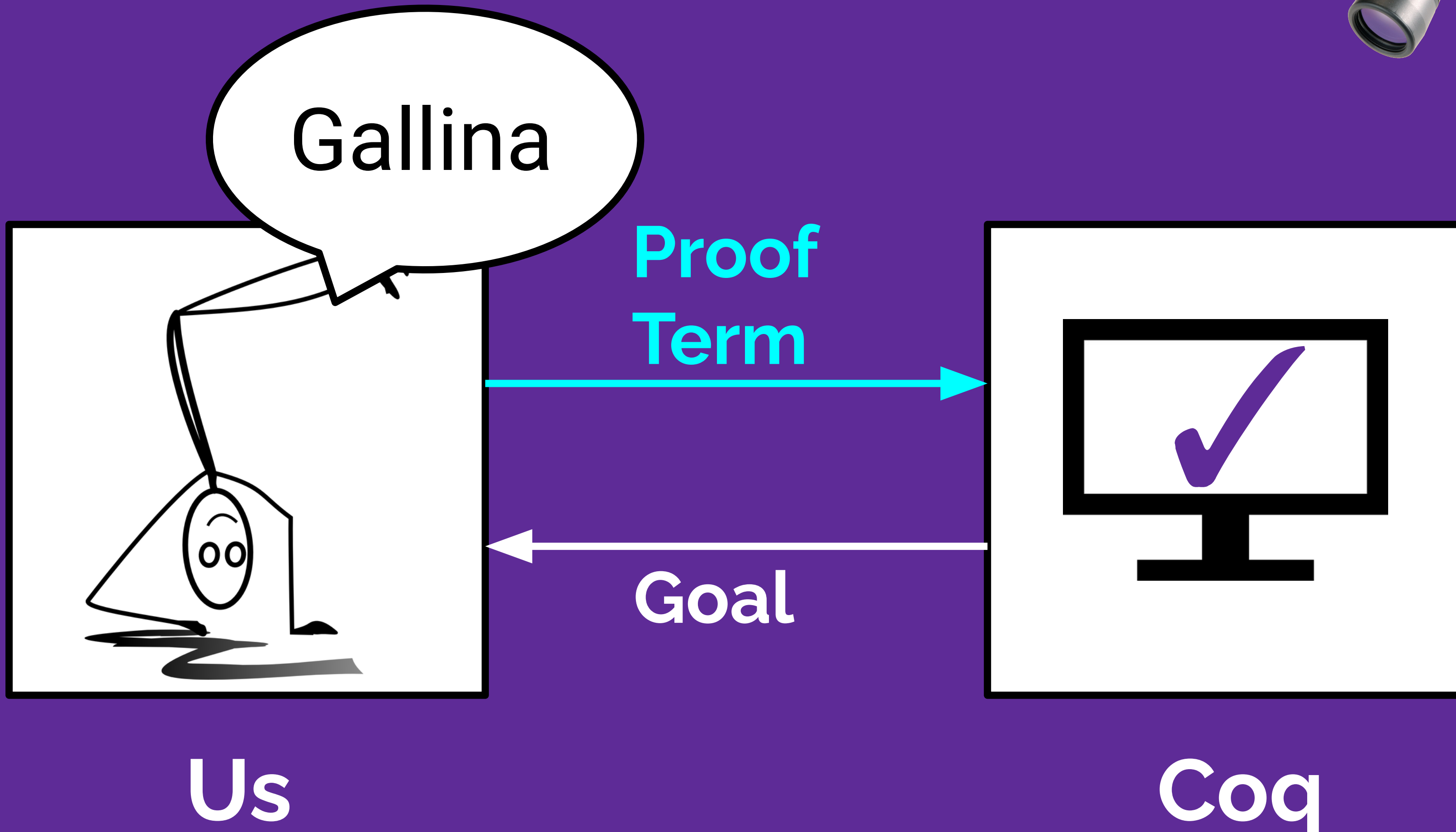
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



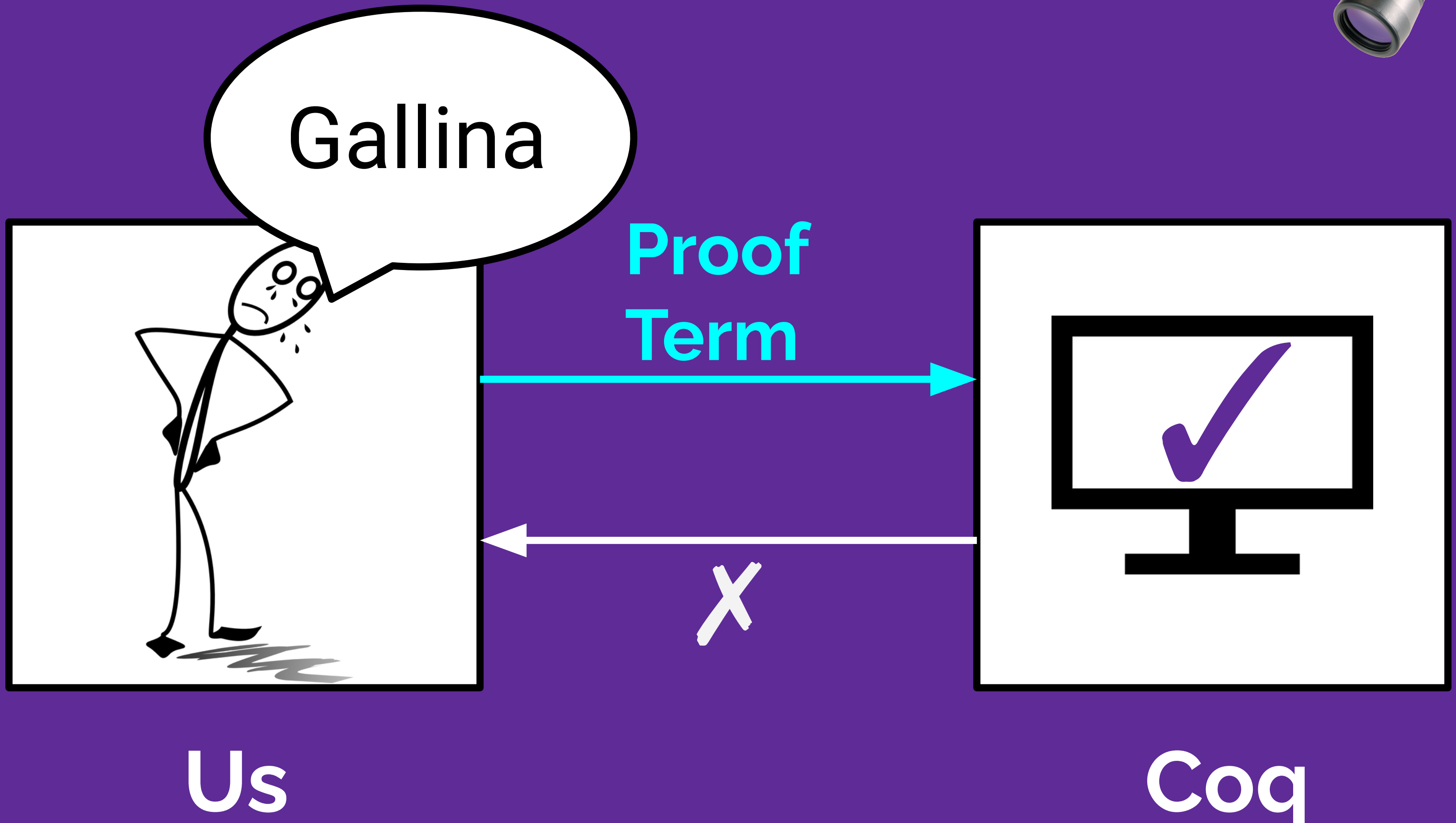
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



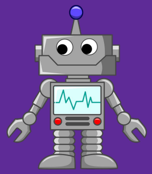
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



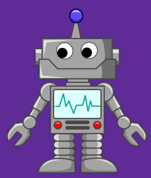
Proof Assistants (Part 1 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities



Proof Automation

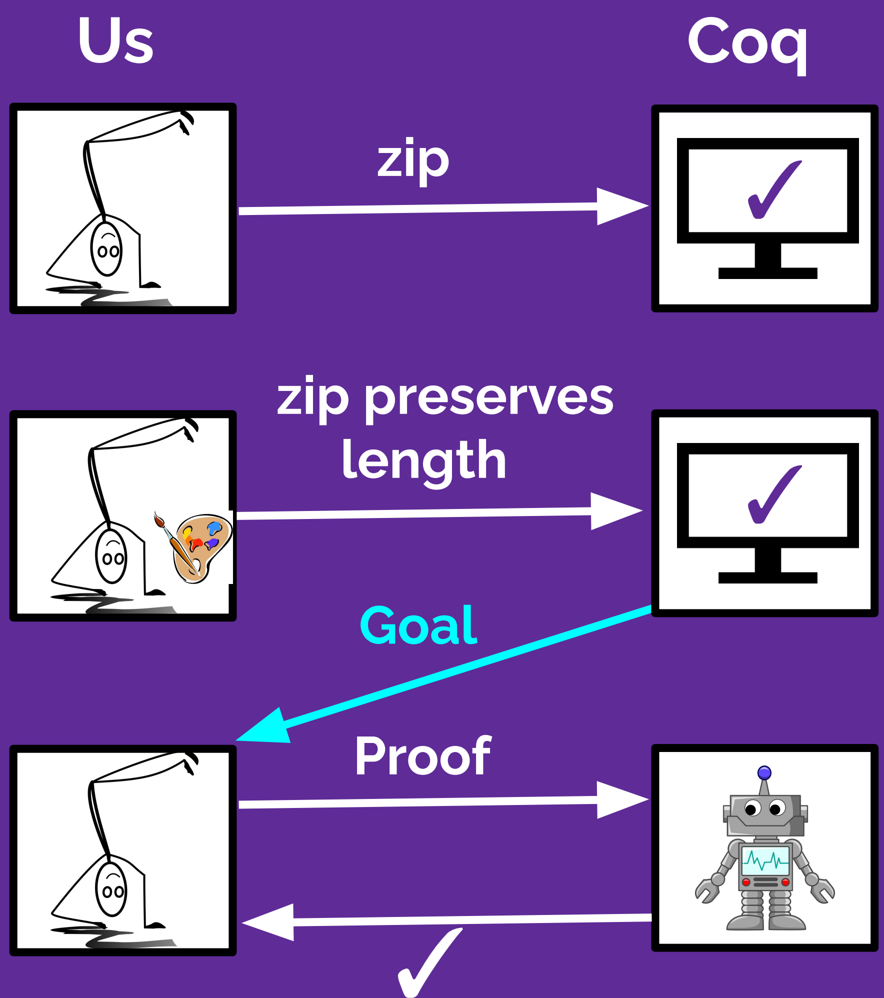
Traditional Automation (Part 2 of 5)



Proof Automation*

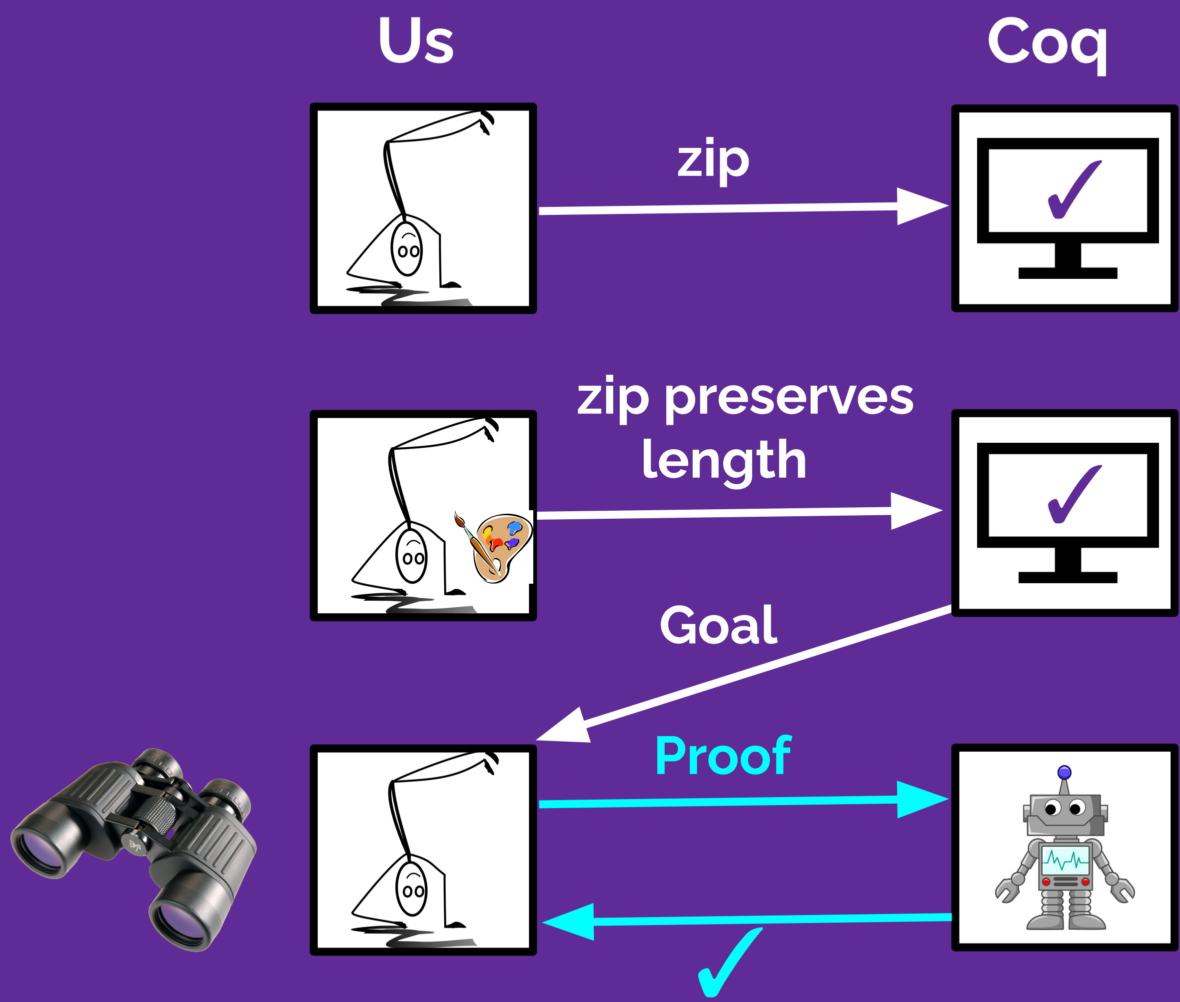
Traditional Automation (Part 2 of 5)

Proof Automation*



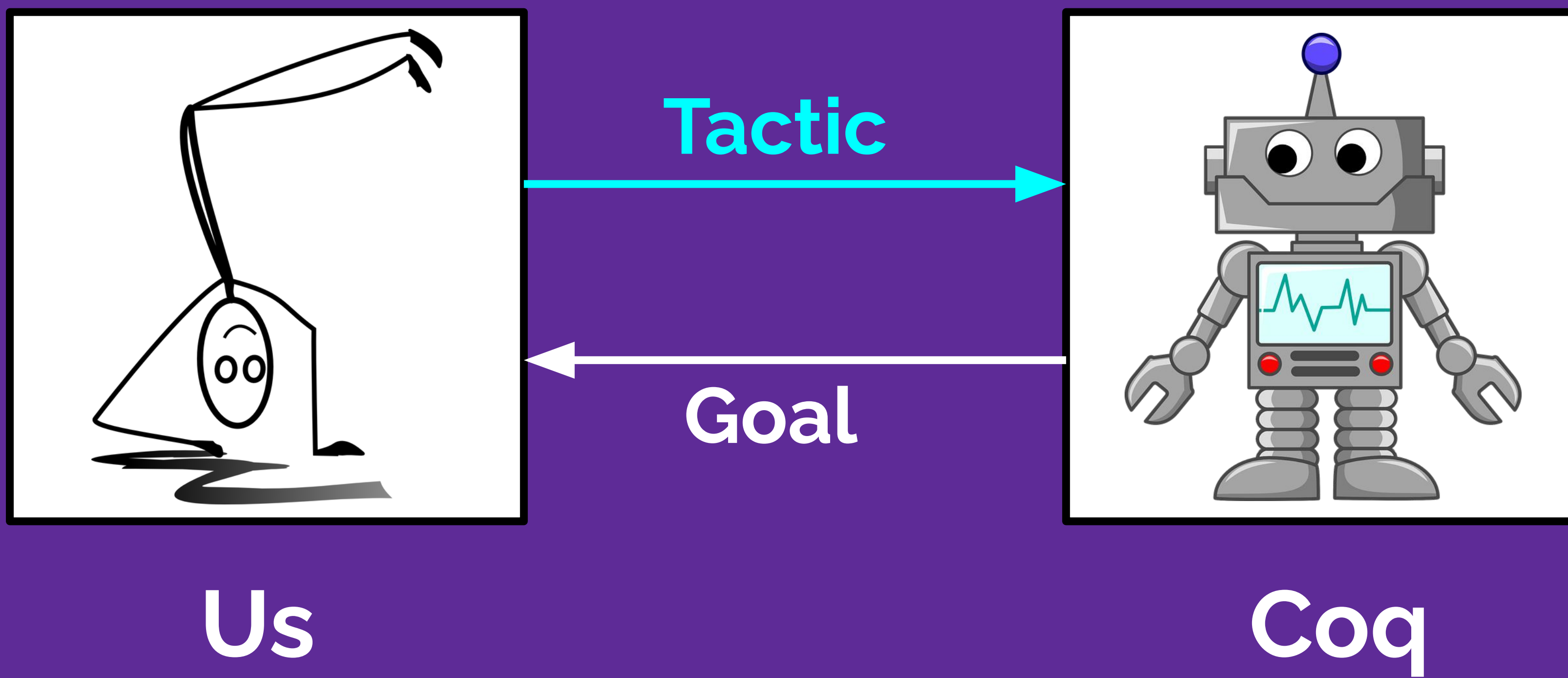
Traditional Automation (Part 2 of 5)

Proof Automation*



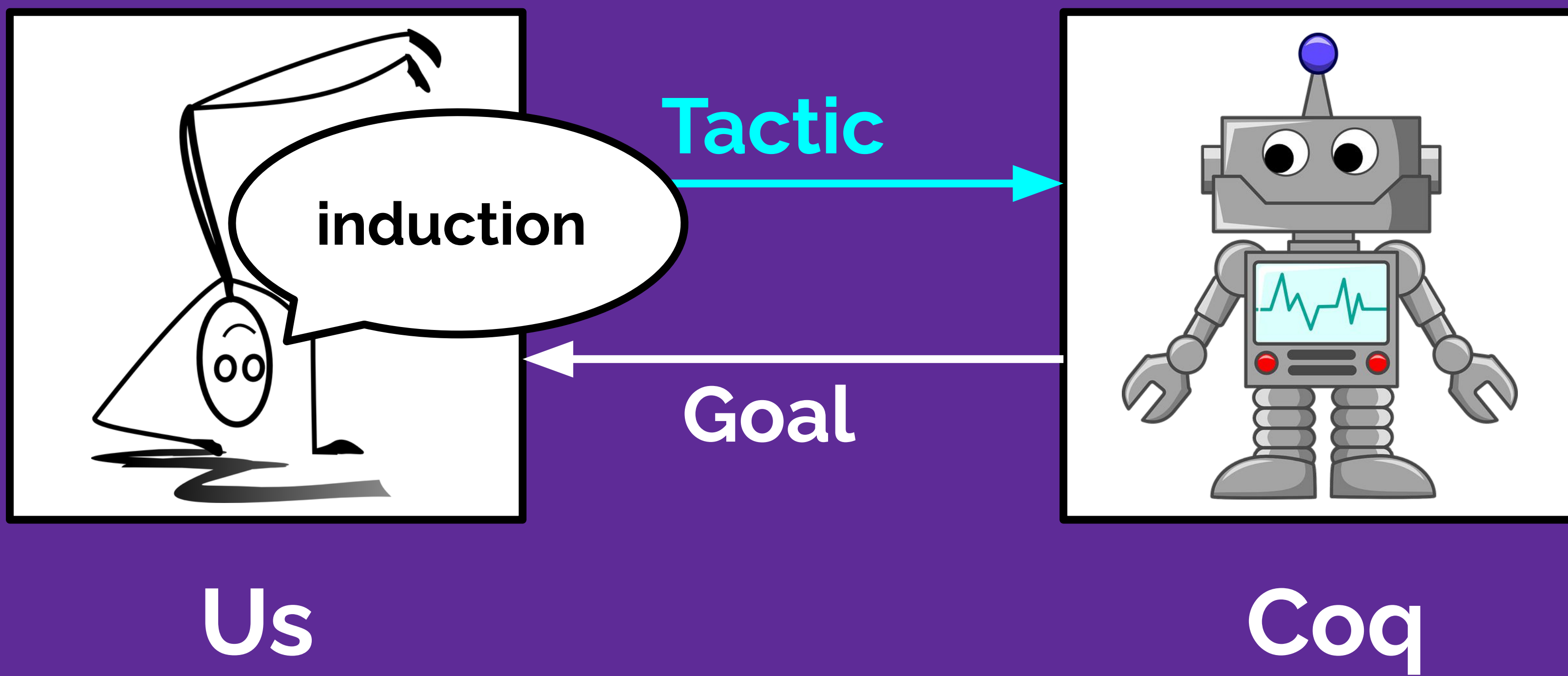
Traditional Automation (Part 2 of 5)

Proof Automation*



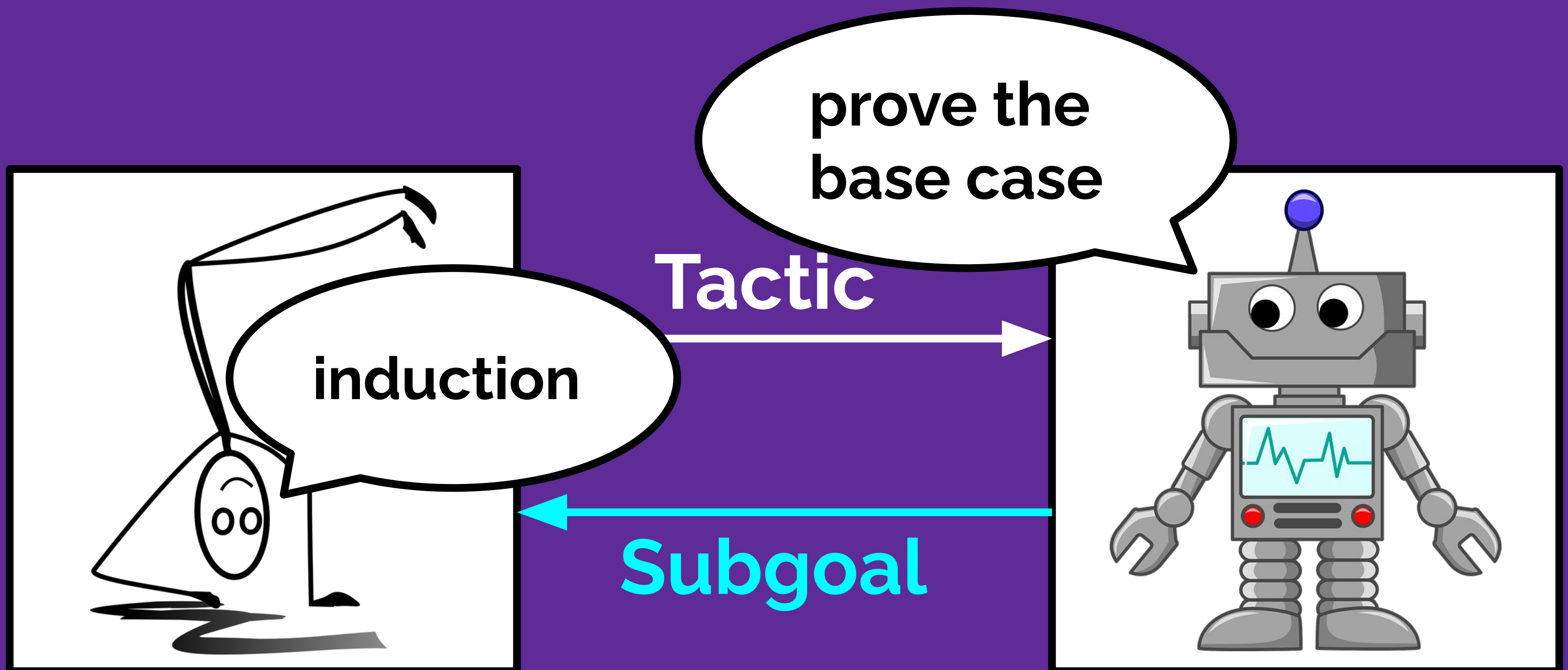
Traditional Automation (Part 2 of 5)

Proof Automation*



Traditional Automation (Part 2 of 5)

Proof Automation*

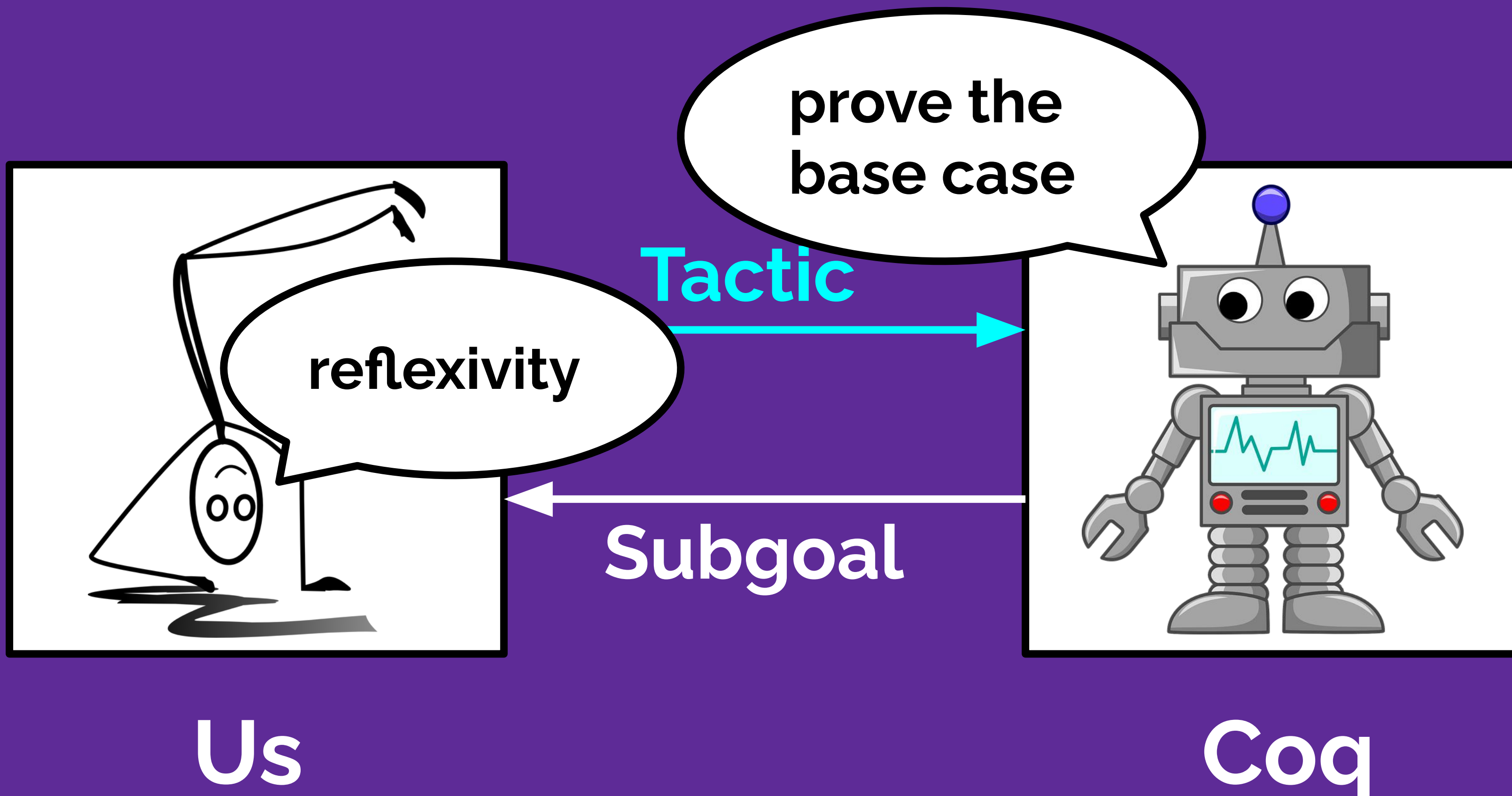


Us

Coq

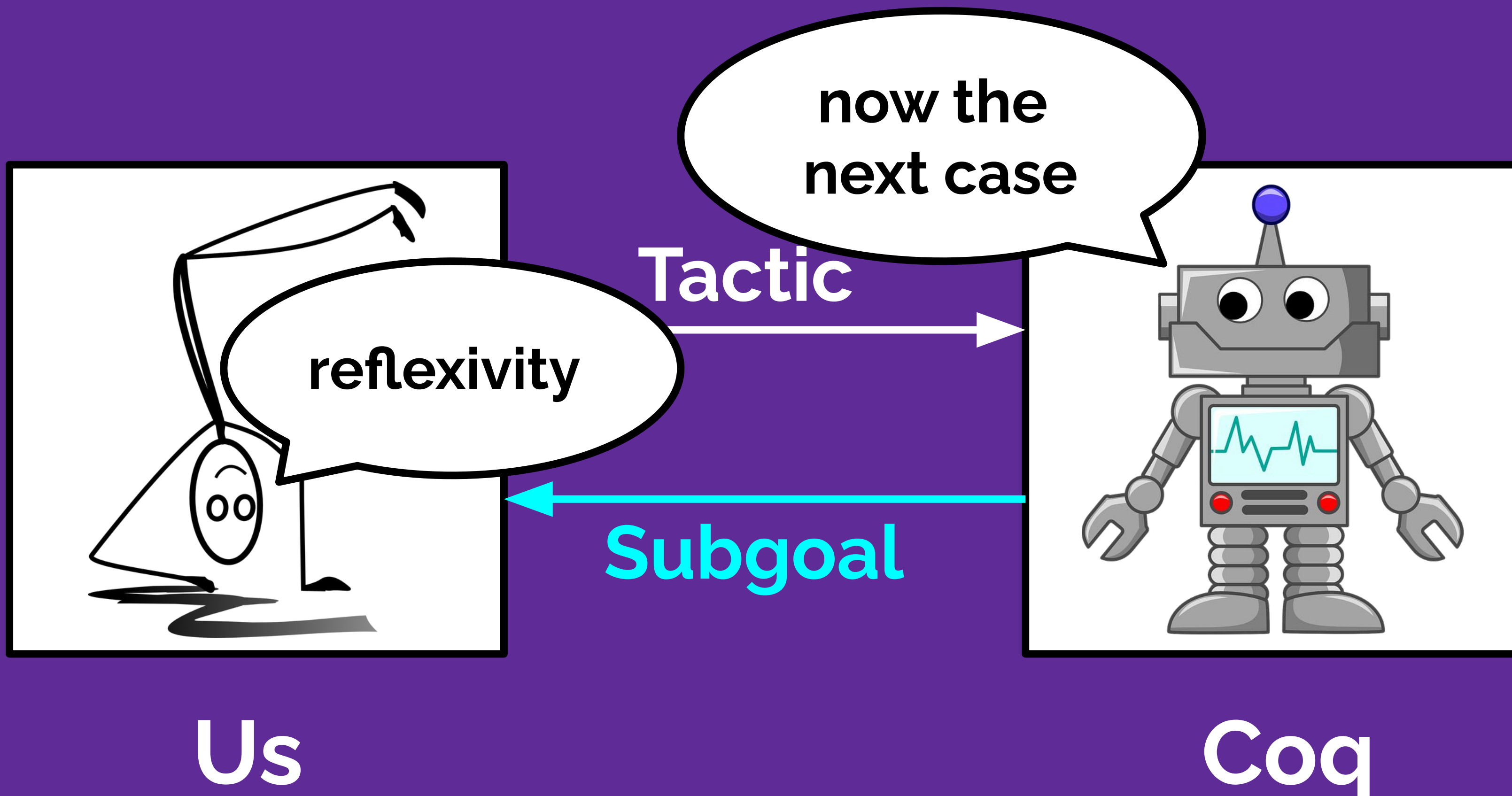
Traditional Automation (Part 2 of 5)

Proof Automation*



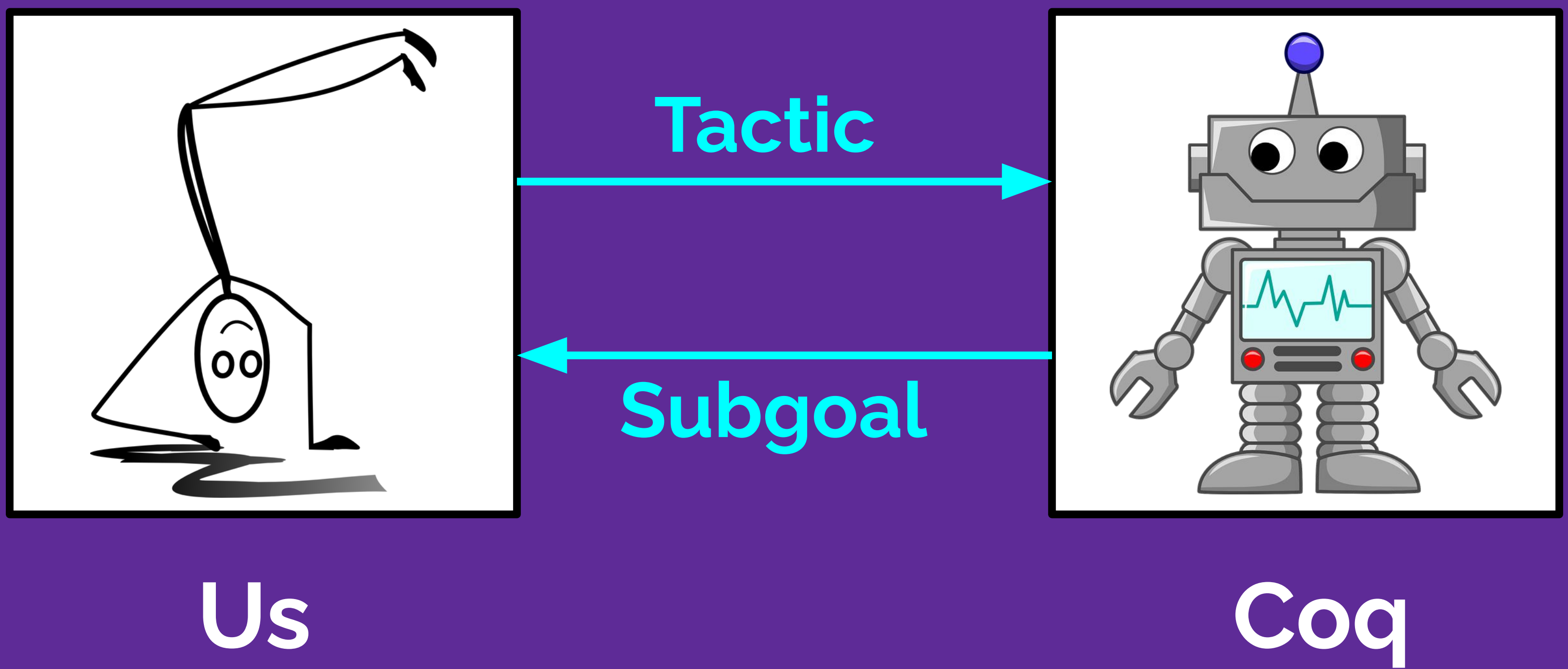
Traditional Automation (Part 2 of 5)

Proof Automation*



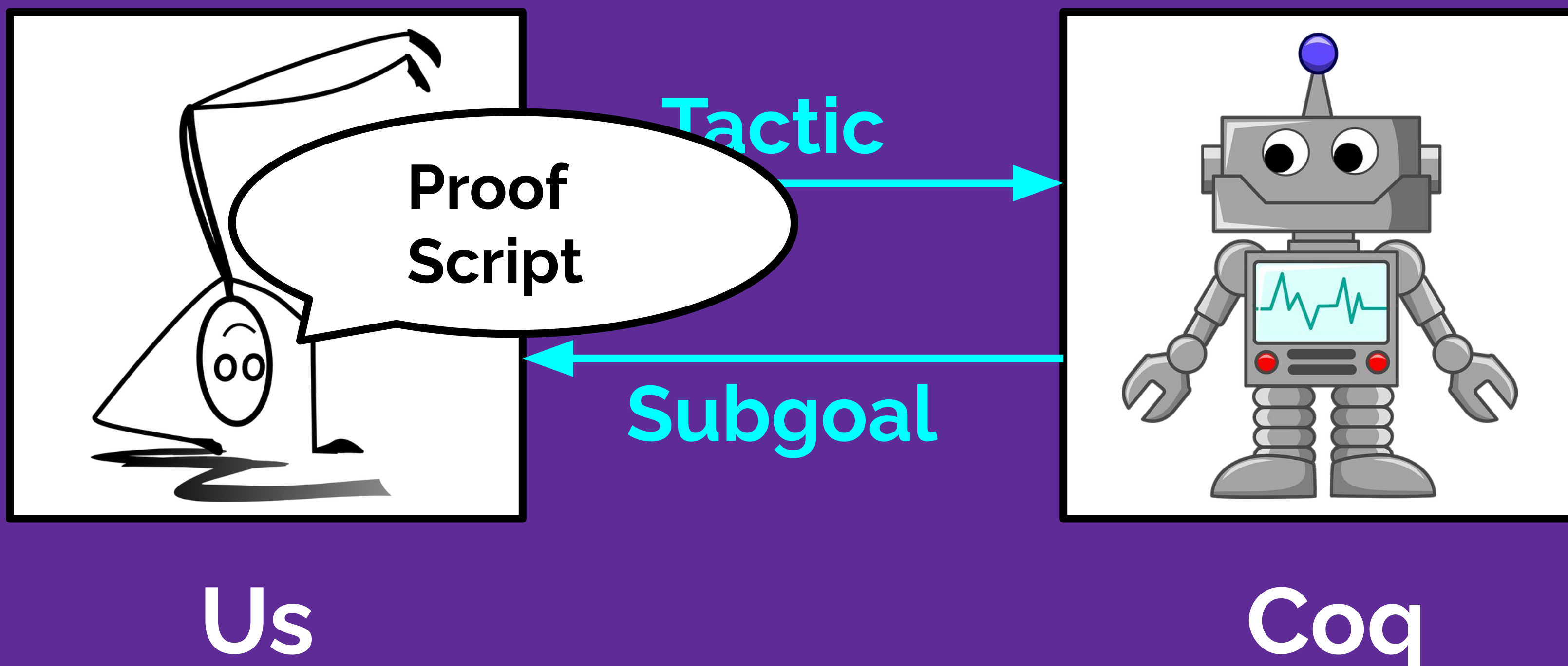
Traditional Automation (Part 2 of 5)

Proof Automation*



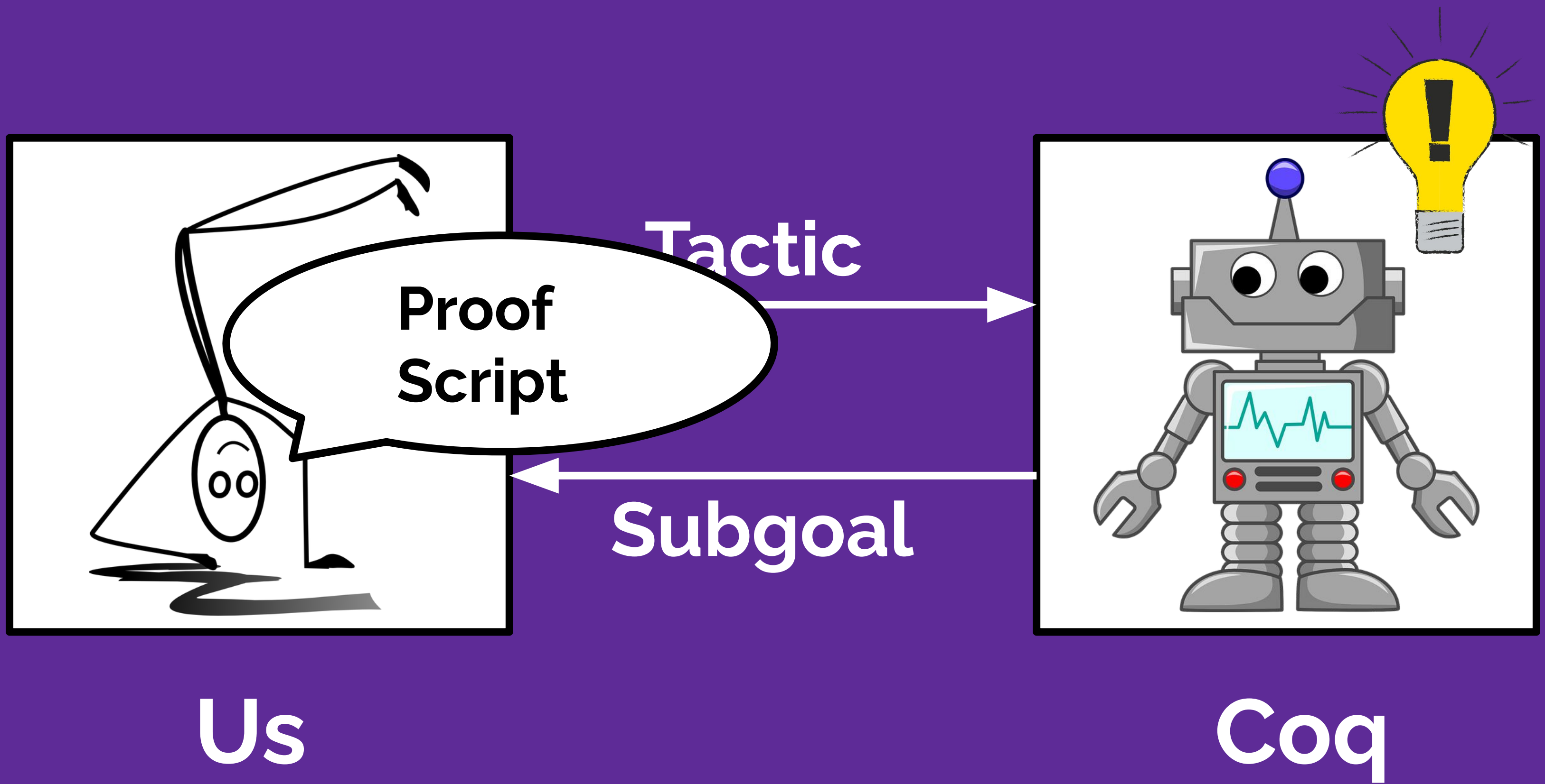
Traditional Automation (Part 2 of 5)

Proof Automation*



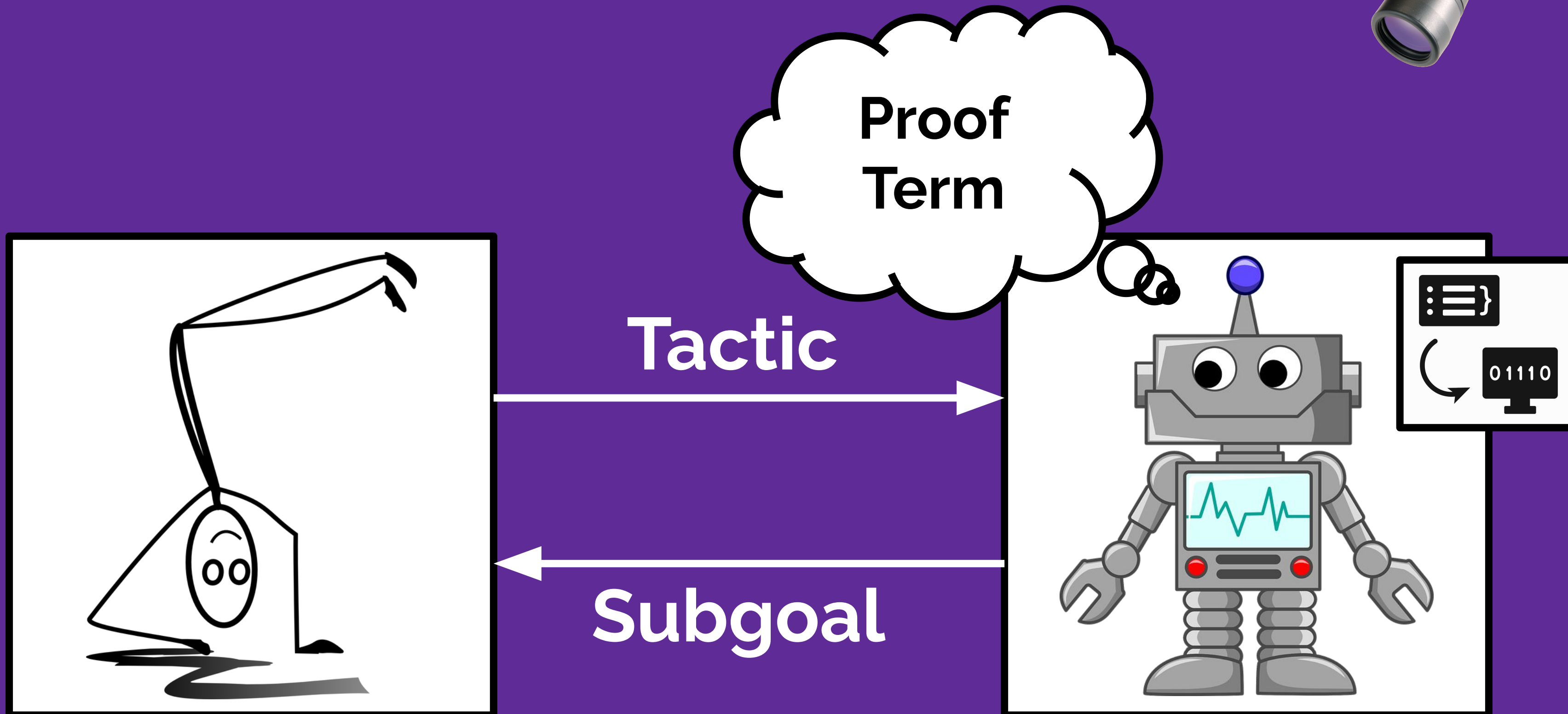
Traditional Automation (Part 2 of 5)

Proof Automation*



Traditional Automation (Part 2 of 5)

Proof Automation*

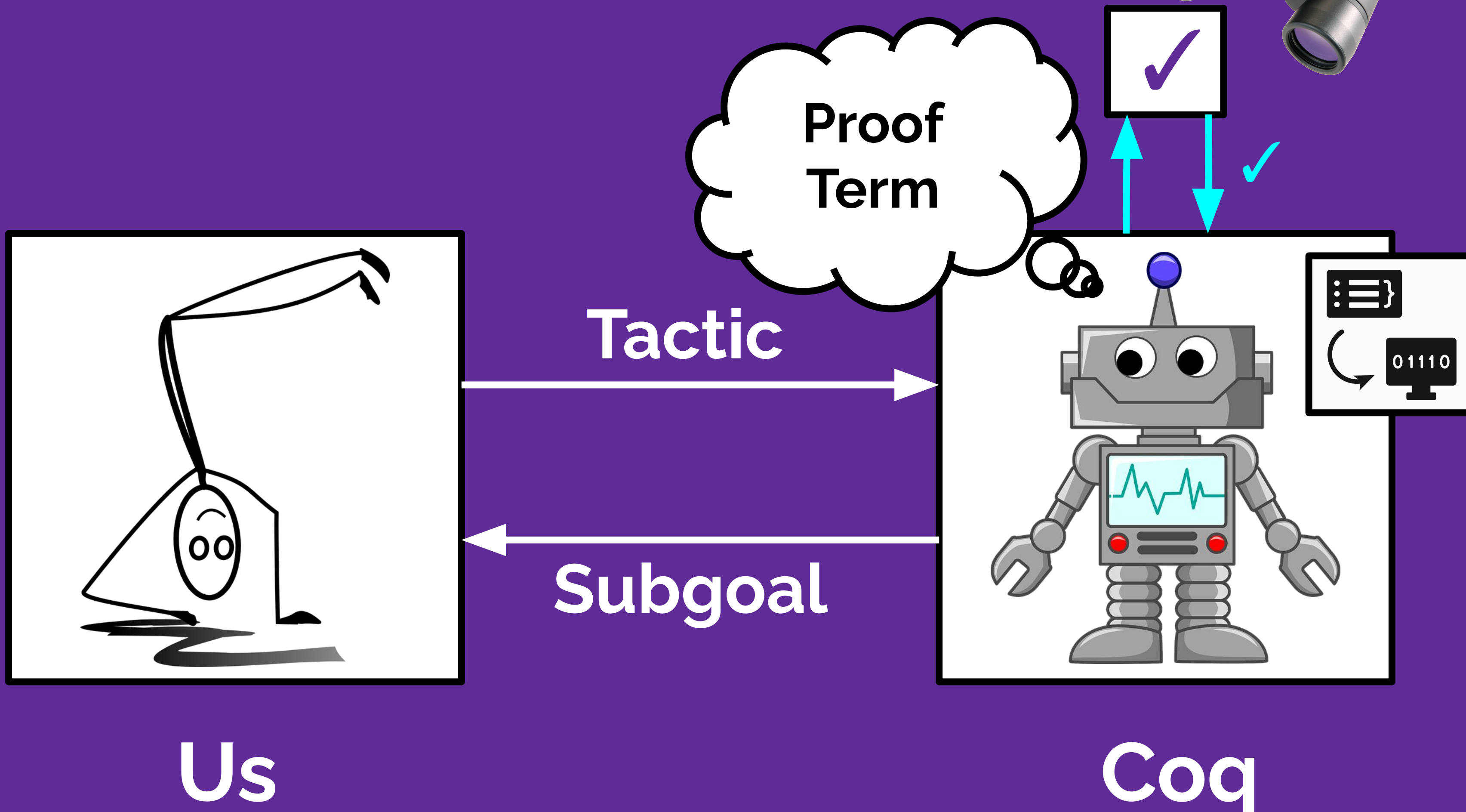


Us

Coq

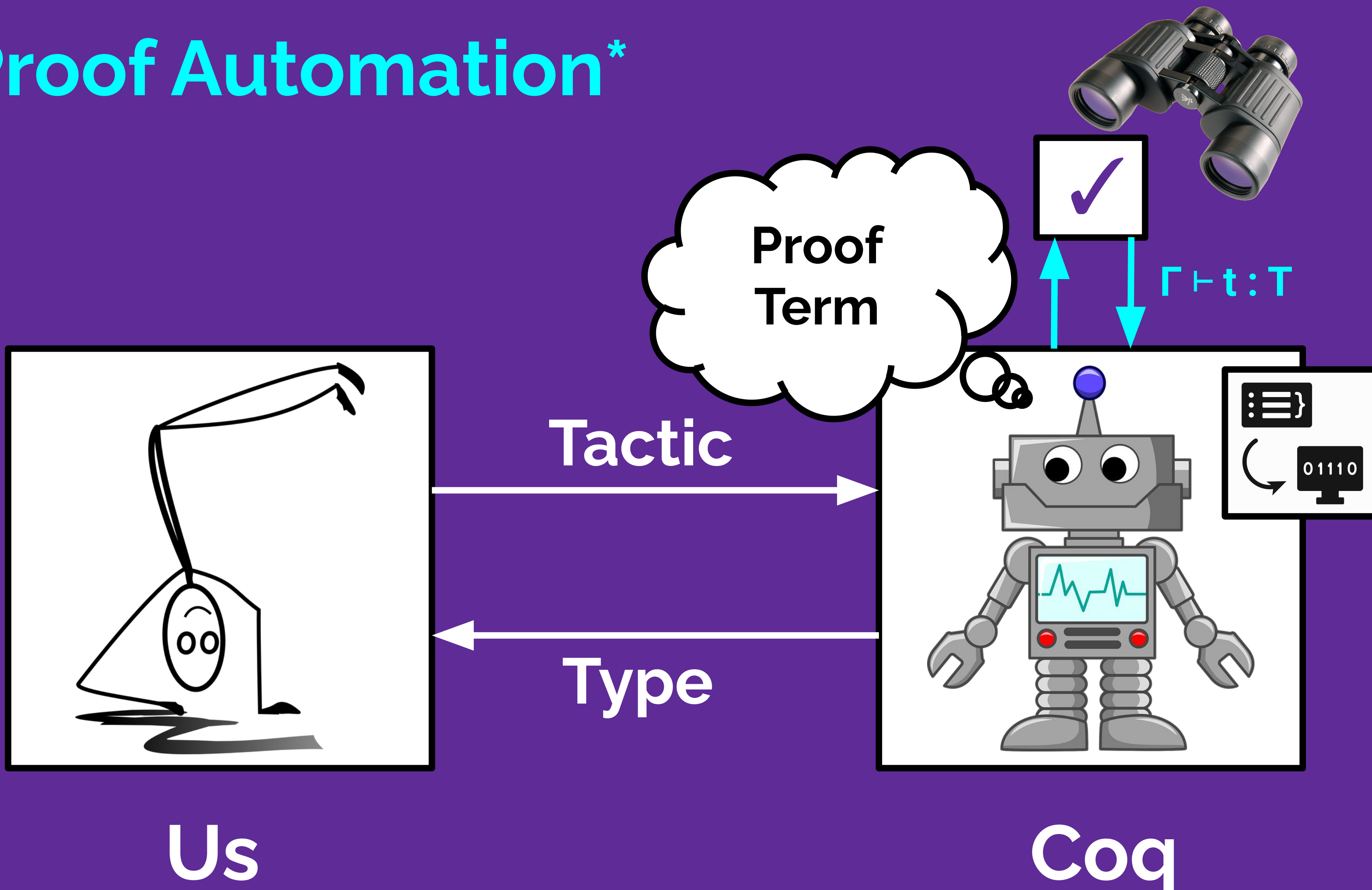
Traditional Automation (Part 2 of 5)

Proof Automation*



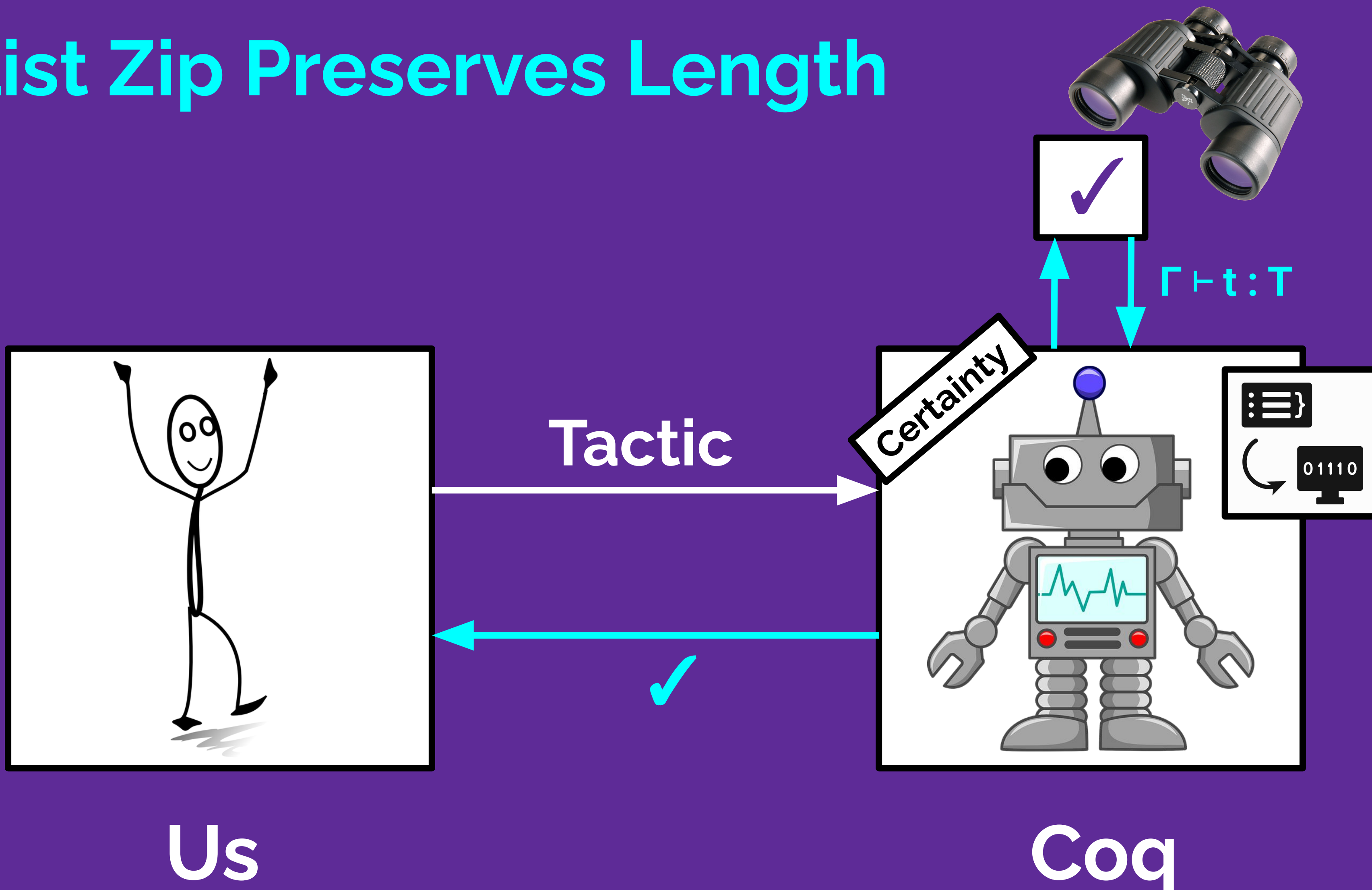
Traditional Automation (Part 2 of 5)

Proof Automation*



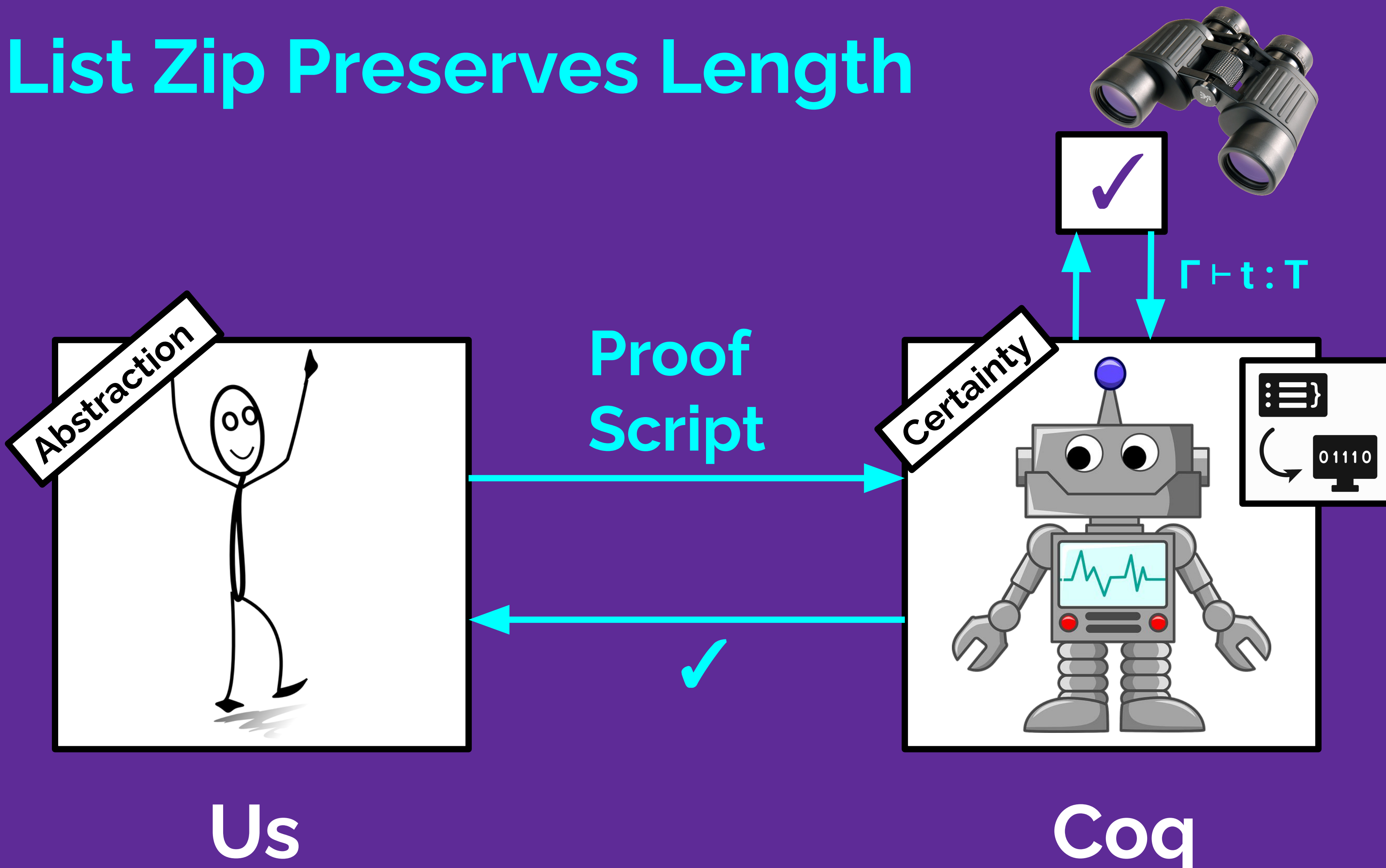
Traditional Automation (Part 2 of 5)

List Zip Preserves Length



Traditional Automation (Part 2 of 5)

List Zip Preserves Length



Traditional Automation (Part 2 of 5)

List Zip Preserves Length

```
intros T1 T2 l1. induction l1[1] as [|t1 tl1 IHtl1].
- auto.[2]
- intros l2. induction l2[3] as [|t2 tl2 IHtl2].
  + intros H. auto.[4]
  + intros H. simpl.  rewrite IHtl1; auto.[5]
```

Abstraction

```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>
  list_rect[1] (fun (l1 : list T1) => ...)
    (fun (l2 :  list T2) _ => eq_refl)[2]
    (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>
      list_rect[3] (fun (l2 : list T2) => ...)
        (fun (H : ...) => eq_sym H)[4]
        (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>
          fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...)[5])
        l2[3])
    l1[1]
    l2.
```

Certainty

Traditional Automation (Part 2 of 5)

List Zip Preserves Length

```
intros T1 T2 l1. induction l1[1] as [|t1 tl1 IHtl1].
- auto.[2]
- intros l2. induction l2[3] as [|t2 tl2 IHtl2].
  + intros H. auto.[4]
  + intros H. simpl.  rewrite IHtl1; auto.[5]
```

Abstraction

Induction => Induction Principles

```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>
  list_rect[1] (fun (l1 : list T1) => ...)
    (fun (l2 :  list T2) _ => eq_refl)[2]
    (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>
      list_rect[3] (fun (l2 : list T2) => ...)
        (fun (H : ...) => eq_sym H)[4]
        (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>
          fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...)[5])
      l2[3])
    l1[1]
    l2.
```

Certainty

Traditional Automation (Part 2 of 5)

List Zip Preserves Length

```
intros T1 T2 l1. induction l1[1] as [|t1 tl1 IHtl1].
- auto.[2]
- intros l2. induction l2[3] as [|t2 tl2 IHtl2].
  + intros H. auto.[4]
  + intros H. simpl.  rewrite IHtl1; auto.[5]
```

Abstraction

Induction => Induction Principles

```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>
  list_rect[1] (fun (l1 : list T1) => ...)
    (fun (l2 :  list T2) _ => eq_refl)[2]
    (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>
      list_rect[3] (fun (l2 : list T2) => ...)
        (fun (H : ...) => eq_sym H)[4]
        (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>
          fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...)[5])
        l2[3])
    l1[1]
    l2.
```

Certainty

Traditional Automation (Part 2 of 5)

Kinds of Automation

- Tactic languages
- Reflection
- Custom tactics
- Custom proof modes
- Proof procedures
- Plugins
- Proof repair
- Hammers

Traditional Automation (Part 2 of 5)

Kinds of Automation

- Tactic languages
- Reflection
- Custom tactics
- Custom proof modes
- Proof procedures
- Plugins
- Proof repair
- Hammers

Traditional Automation (Part 2 of 5)

This automation can do basically anything, yet still **preserve correctness**.

Traditional Automation (Part 2 of 5)

De Bruijn Criterion

Traditional Automation (Part 2 of 5)

Checking the Proof

Producing the Proof

Traditional Automation (Part 2 of 5)

Checking the Proof

Producing the Proof

Traditional Automation (Part 2 of 5)

Checking the Proof

Search Procedures

Producing the Proof

Traditional Automation (Part 2 of 5)

Checking the Proof

Search Procedures

Domain-Specific Heuristics

Producing the Proof

Traditional Automation (Part 2 of 5)

Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

Traditional Automation (Part 2 of 5)

Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Spoiler!

Producing the Proof

ChatGPT

Traditional Automation (Part 2 of 5)

Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

ChatGPT

Traditional Automation (Part 2 of 5)

Checking the Proof

Small & Human-Readable Logic Checker

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

ChatGPT

Traditional Automation (Part 2 of 5)

Checking the Proof

Small Logical Kernel

Search Procedures

Domain-Specific Heuristics

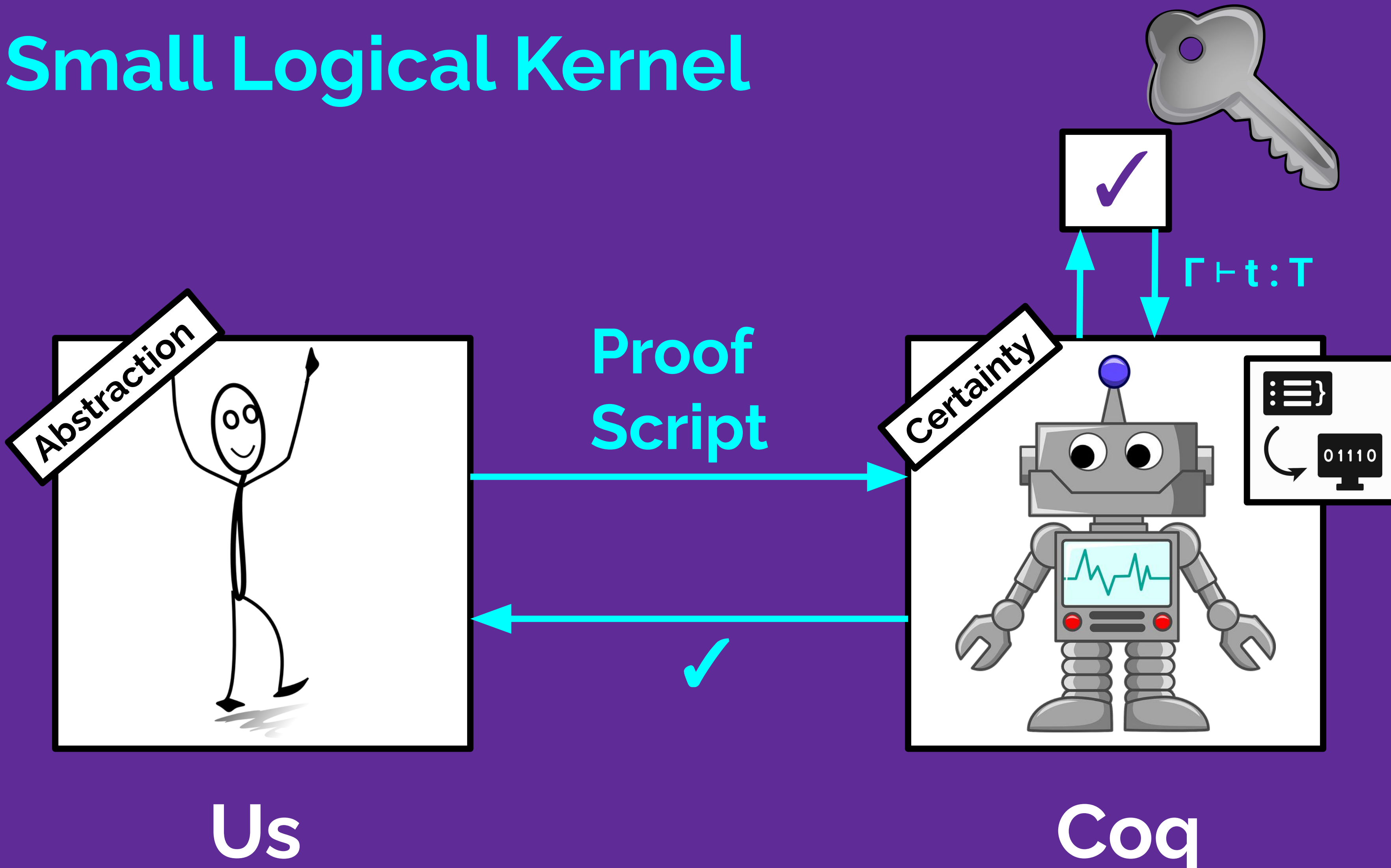
Proof Transformations

Producing the Proof

ChatGPT

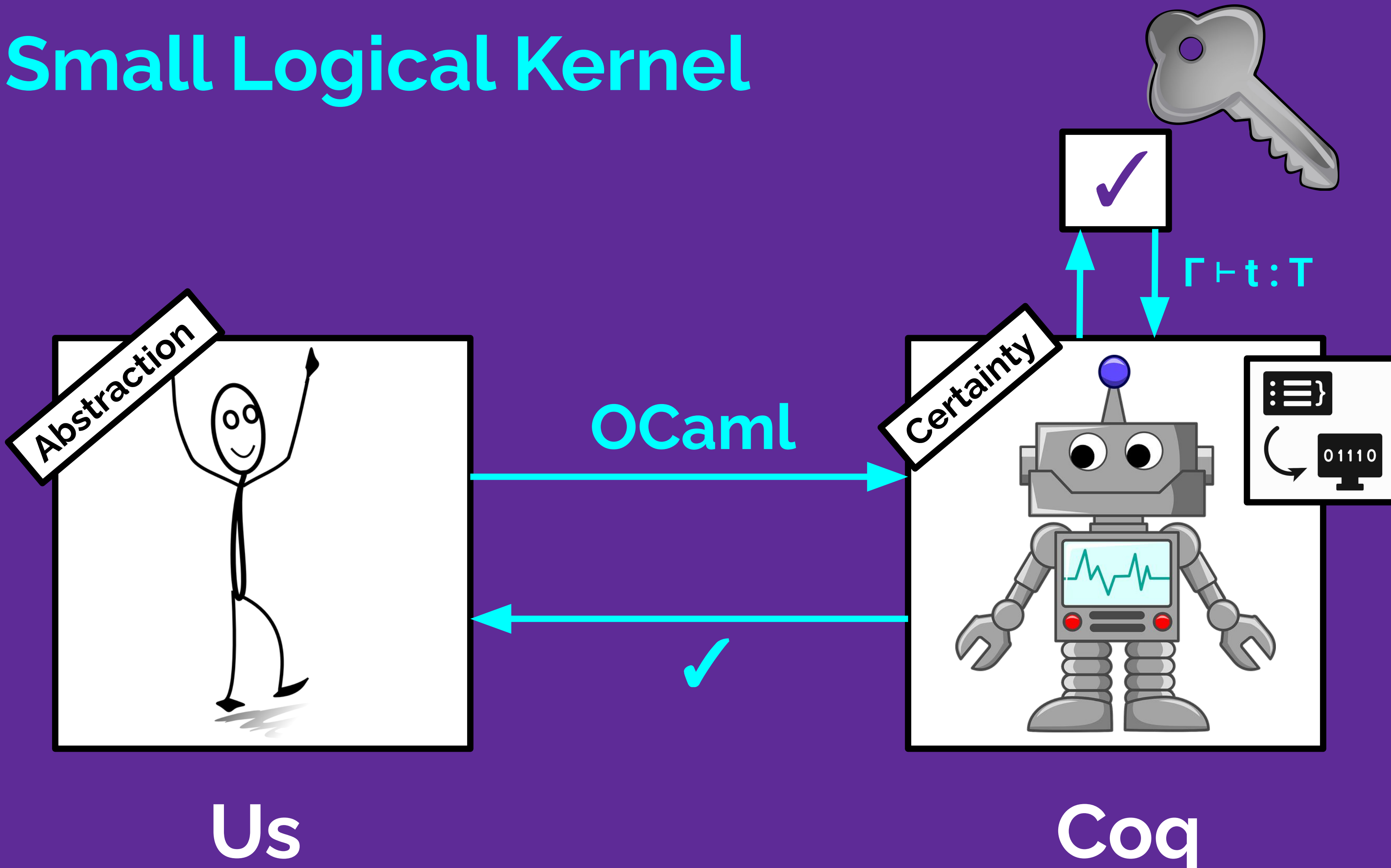
Traditional Automation (Part 2 of 5)

Small Logical Kernel



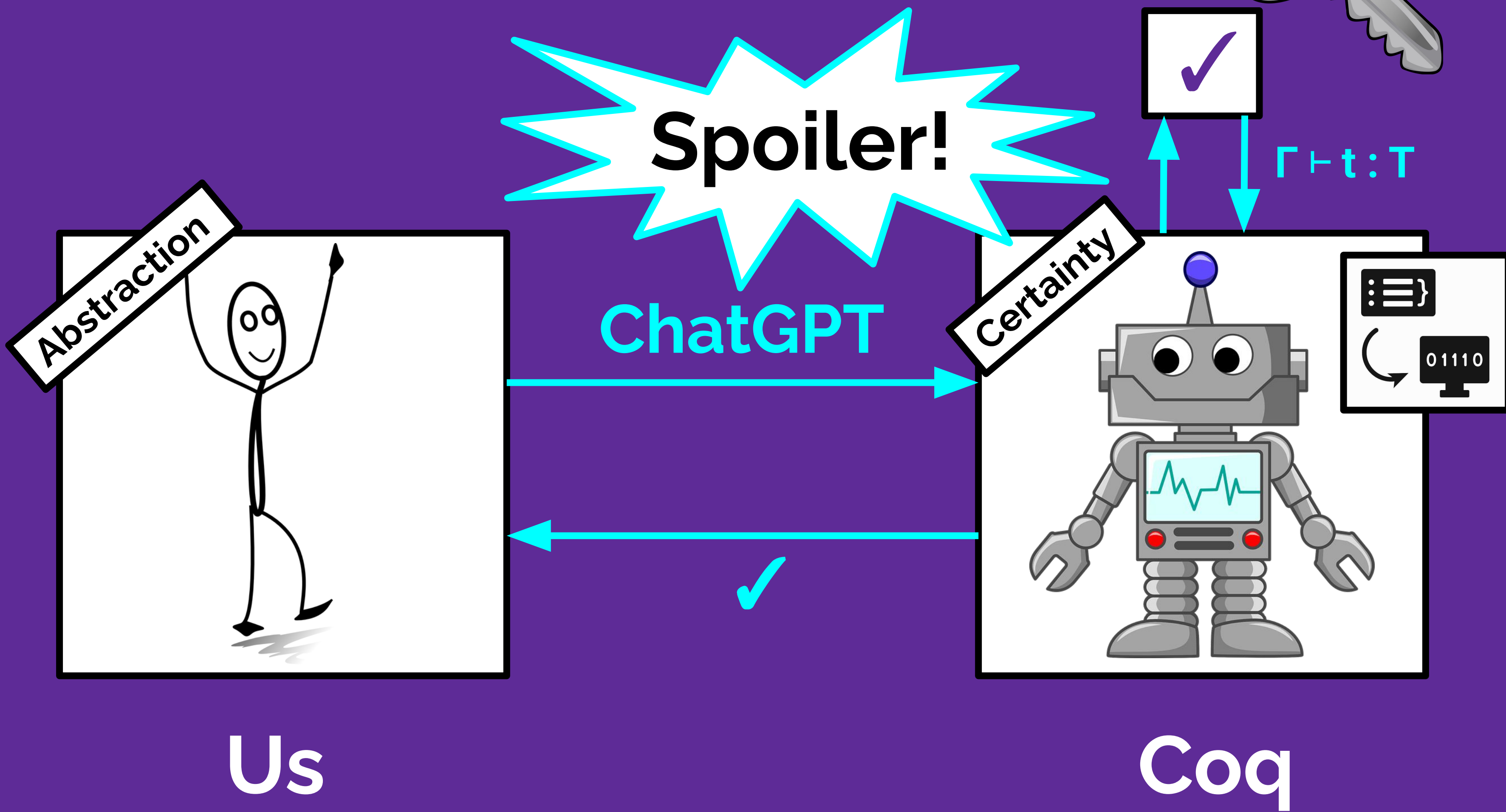
Traditional Automation (Part 2 of 5)

Small Logical Kernel



Traditional Automation (Part 2 of 5)

Small Logical Kernel



Traditional Automation (Part 2 of 5)

With de Bruijn, as long as you don’t touch the kernel, your automation is safe.

Traditional Automation (Part 2 of 5)

With de Bruijn, as long as you don’t touch the kernel, your automation is safe.*

(If your specification is OK, your kernel has no bugs, and you don’t introduce axioms)

Traditional Automation (Part 2 of 5)

With de Bruijn, as long as you don't touch the kernel, your automation is safe.*
The kernel and specification are the core trusted pieces, vetted by humans.

Traditional Automation (Part 2 of 5)

Traditional automation:

+ predictable
+ dependable
+ understandable
- limited in scope
- takes expertise to extend

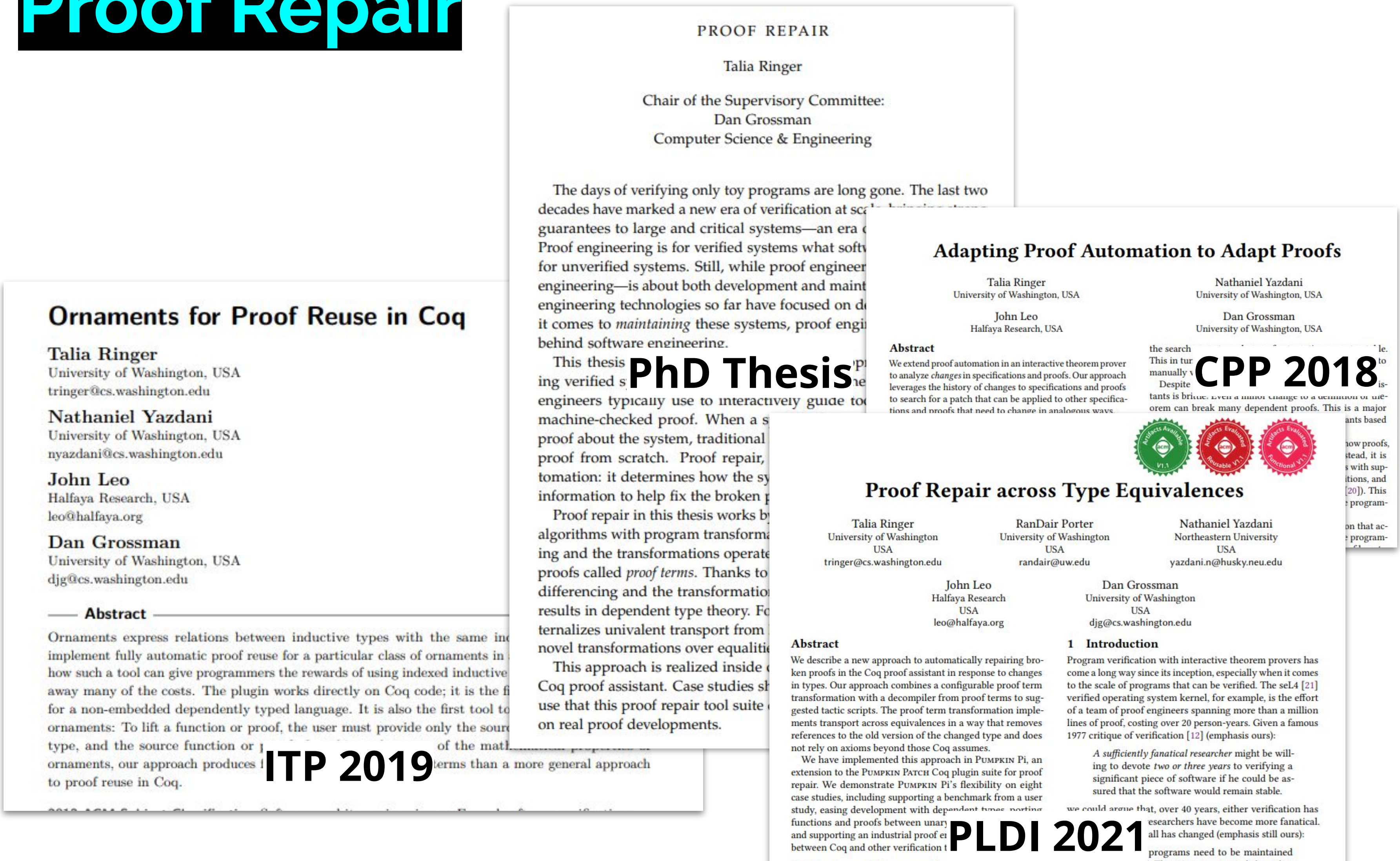
Traditional Automation (Part 2 of 5)

Traditional proof repair:

+ predictable
+ dependable
+ understandable
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair


PROOF REPAIR

Talia Ringer

Chair of the Supervisory Committee:
Dan Grossman
Computer Science & Engineering

The days of verifying only toy programs are long gone. The last two decades have marked a new era of verification at sca

Adapting Proof Automation to Adapt Proofs

Talia Ringer, University of Washington, USA
Nathaniel Yazdani, University of Washington, USA
John Leo, Halfaya Research, USA
Dan Grossman, University of Washington, USA

Abstract

We extend proof automation in an interactive theorem prover to analyze *changes* in specifications and proofs. Our approach leverages the history of changes to specifications and proofs to search for a patch that can be applied to other specifica-

Ornaments for Proof Reuse in Coq

Talia Ringer, University of Washington, USA, tringer@cs.washington.edu
Nathaniel Yazdani, University of Washington, USA, nyazdani@cs.washington.edu
John Leo, Halfaya Research, USA, leo@halfaya.org
Dan Grossman, University of Washington, USA, djg@cs.washington.edu

Abstract

Ornaments express relations between inductive types with the same in

Proof Repair across Type Equivalences

Talia Ringer, University of Washington, USA, tringer@cs.washington.edu
RanDair Porter, University of Washington, USA, randair@uw.edu
Nathaniel Yazdani, Northeastern University, USA, yazdani.n@husky.neu.edu
John Leo, Halfaya Research, USA, leo@halfaya.org
Dan Grossman, University of Washington, USA, djg@cs.washington.edu

Abstract

We describe a new approach to automatically repairing broken proofs in the Coq proof assistant in response to changes in types. Our approach combines a configurable proof term transformation with a decompiler from proof terms to suggested tactic scripts. The proof term transformation implements transport across equivalences in a way that removes references to the old version of the changed type and does not rely on axioms beyond those Coq assumes.

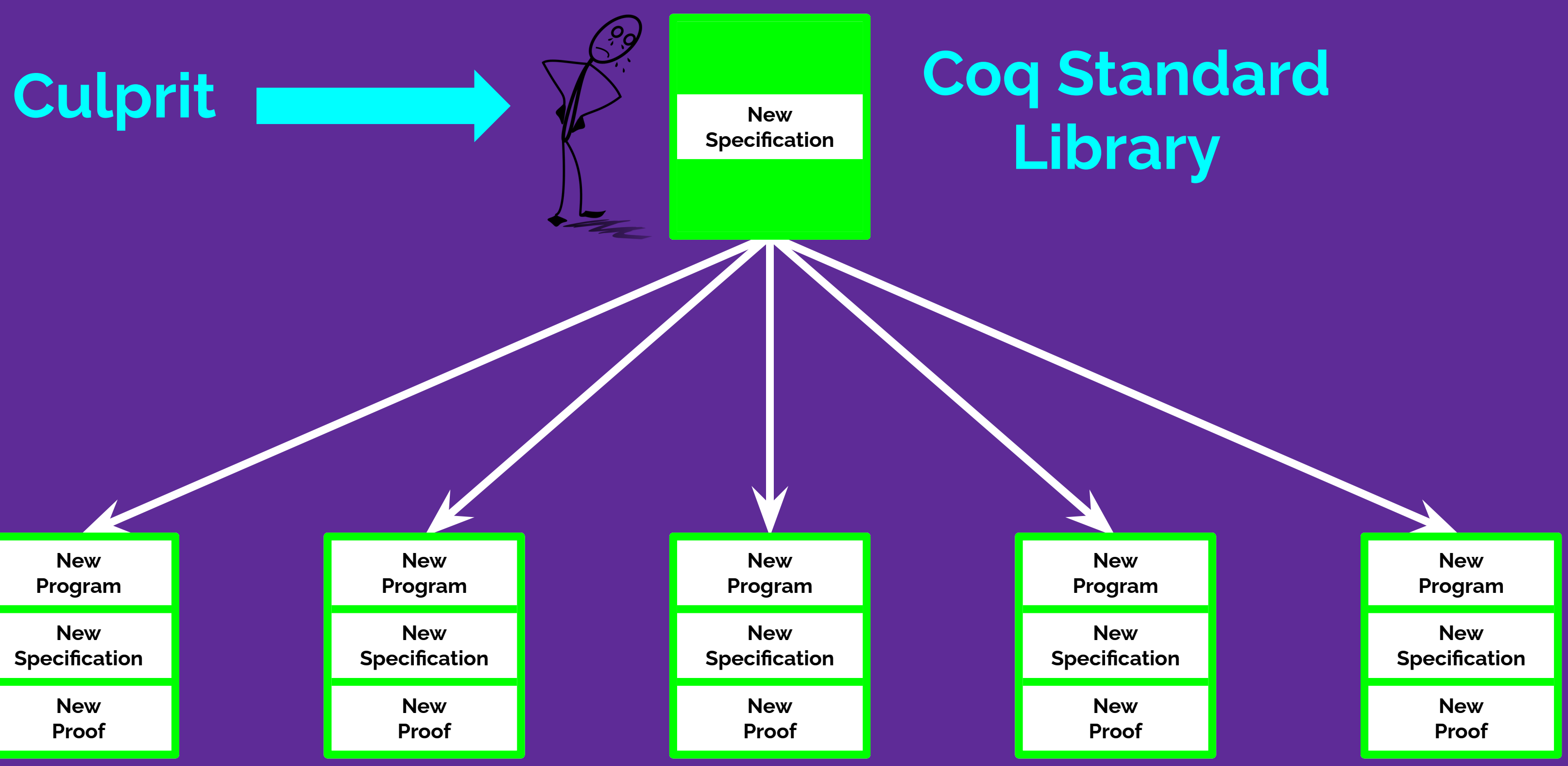
1 Introduction

Program verification with interactive theorem provers has come a long way since its inception, especially when it comes to the scale of programs that can be verified. The seL4 [21] verified operating system kernel, for example, is the effort of a team of proof engineers spanning more than a million lines of proof, costing over 20 person-years. Given a famous 1977 critique of verification [12] (emphasis ours):

A sufficiently fanatical researcher might be willing to devote *two or three years* to verifying a significant piece of software if he could be assured that the software would remain stable.

Traditional Automation (Part 2 of 5)

Proof Repair



Traditional Automation (Part 2 of 5)

You have changed a datatype, and now the standard library is broken!

Traditional Automation (Part 2 of 5)

Proof Repair

451 functions & proofs,
25 seconds

You have **changed** a **datatype**, and now the **standard library is broken**!

Traditional Automation (Part 2 of 5)

Proof Repair

list <T> :=
  | **[ ]** : list <T>
  | **cons** : T → list <T> → list <T>

(* Repair all 451 functions & proofs: *)
Repair Module Old.list **New.list** in StdLib.

Traditional Automation (Part 2 of 5)

Proof Repair

```
list <T> :=
  | cons : T → list <T> → list <T>
  | [ ] : list <T>
```

```
(* Repair all 451 functions & proofs: *)
Repair Module Old.list New.list in StdLib.
```

Traditional Automation (Part 2 of 5)

Proof Repair

```
list <T> :=
  | cons : T → list <T> → list <T>
  | [ ] : list <T>
```

```
(* Repair all 451 functions & proofs: *)
Repair Module Old.list New.list in StdLib.
```

Traditional Automation (Part 2 of 5)

Traditional proof repair:

+ predictable
+ dependable
+ understandable
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair – Predictable

PUMPKIN Pi supports any change described by a type equivalence.

The Univalent Foundations Program. 2013. **Homotopy Type Theory: Univalent Foundations of Mathematics.** Institute for Advanced Study.

Proof Repair – Predictable

PUMPKIN Pi supports any change described by a type equivalence.

The Univalent Foundations Program. 2013. **Homotopy Type Theory: Univalent Foundations of Mathematics.** Institute for Advanced Study.

Traditional Automation (Part 2 of 5)

Equivalences

$$\begin{tikzcd}[column sep=huge]
\text{Old.list T} \arrow[r, bend left, "\textbf{swap}"] & \text{New.list T} \arrow[l, bend left, "\textbf{swap_back}"]
\end{tikzcd}$$

Equivalences

swap **l**

l :Old.list T

New.list T

swap_back (swap **l**)

Equivalences

swap (swap_back l)

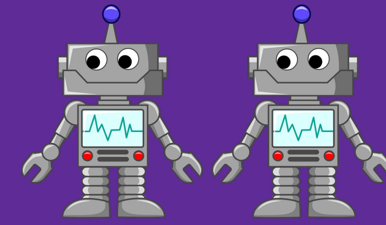
Old.list T l :New.list T

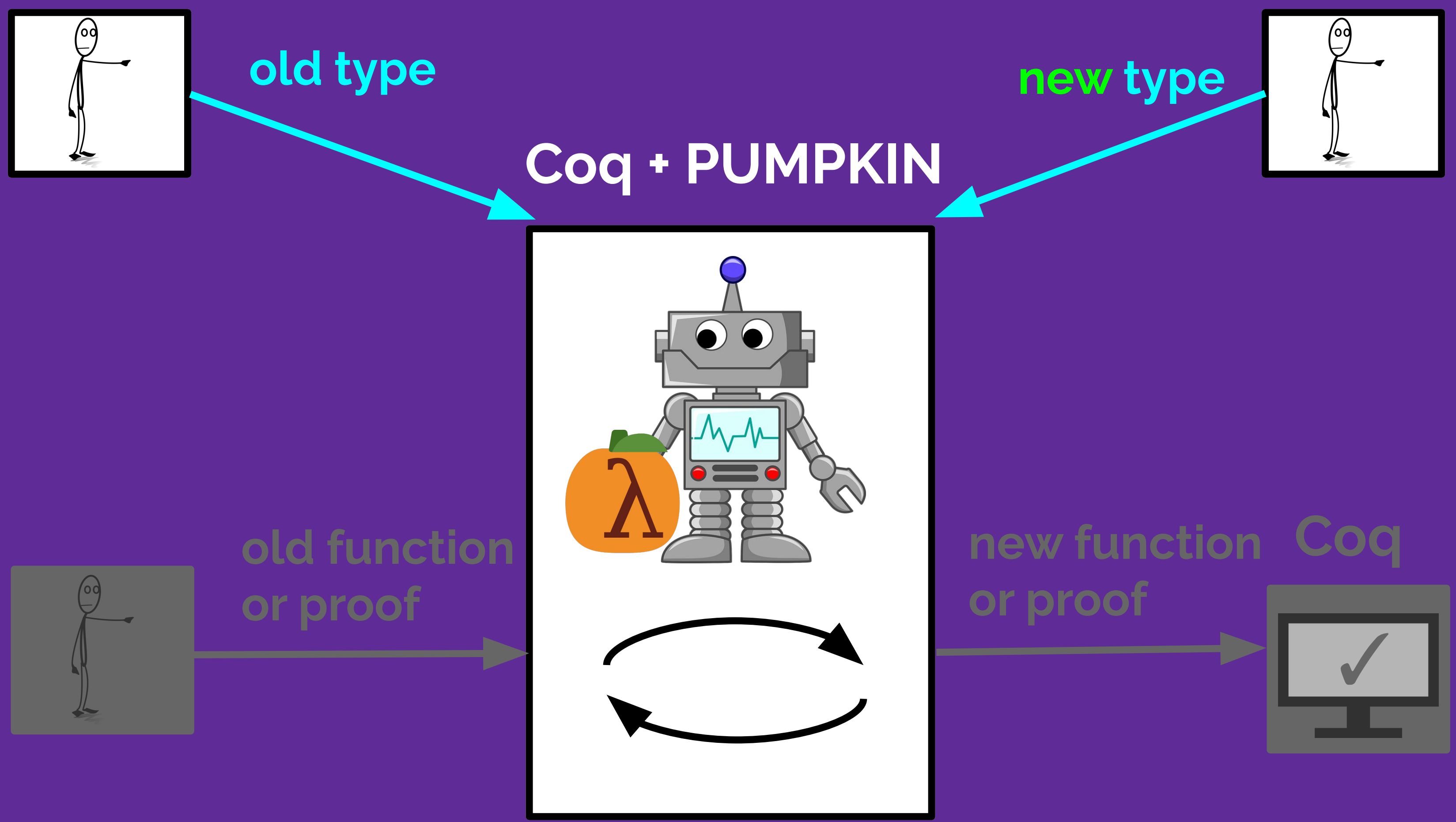
swap l

Traditional Automation (Part 2 of 5)

Equivalences

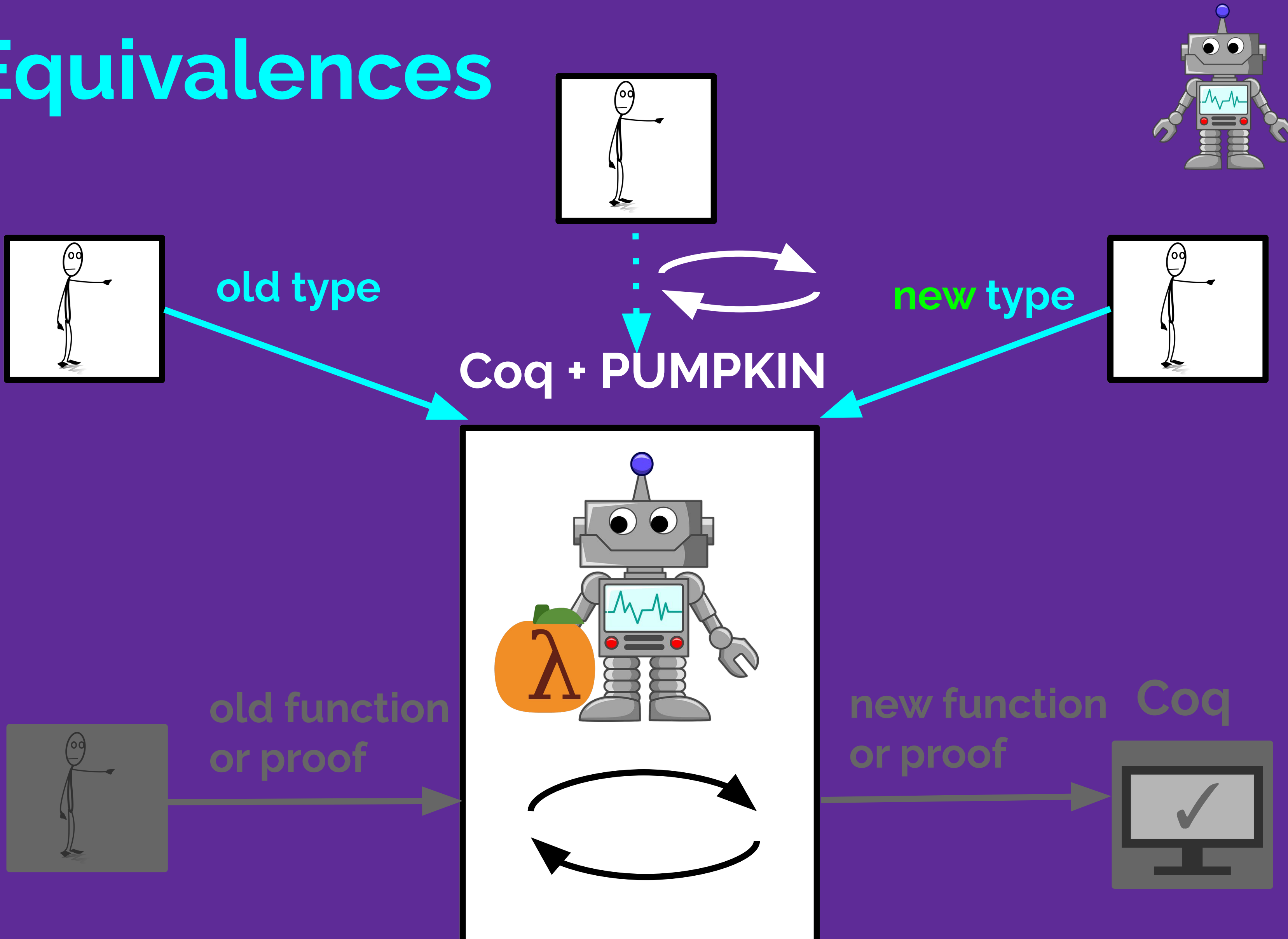
Equivalences





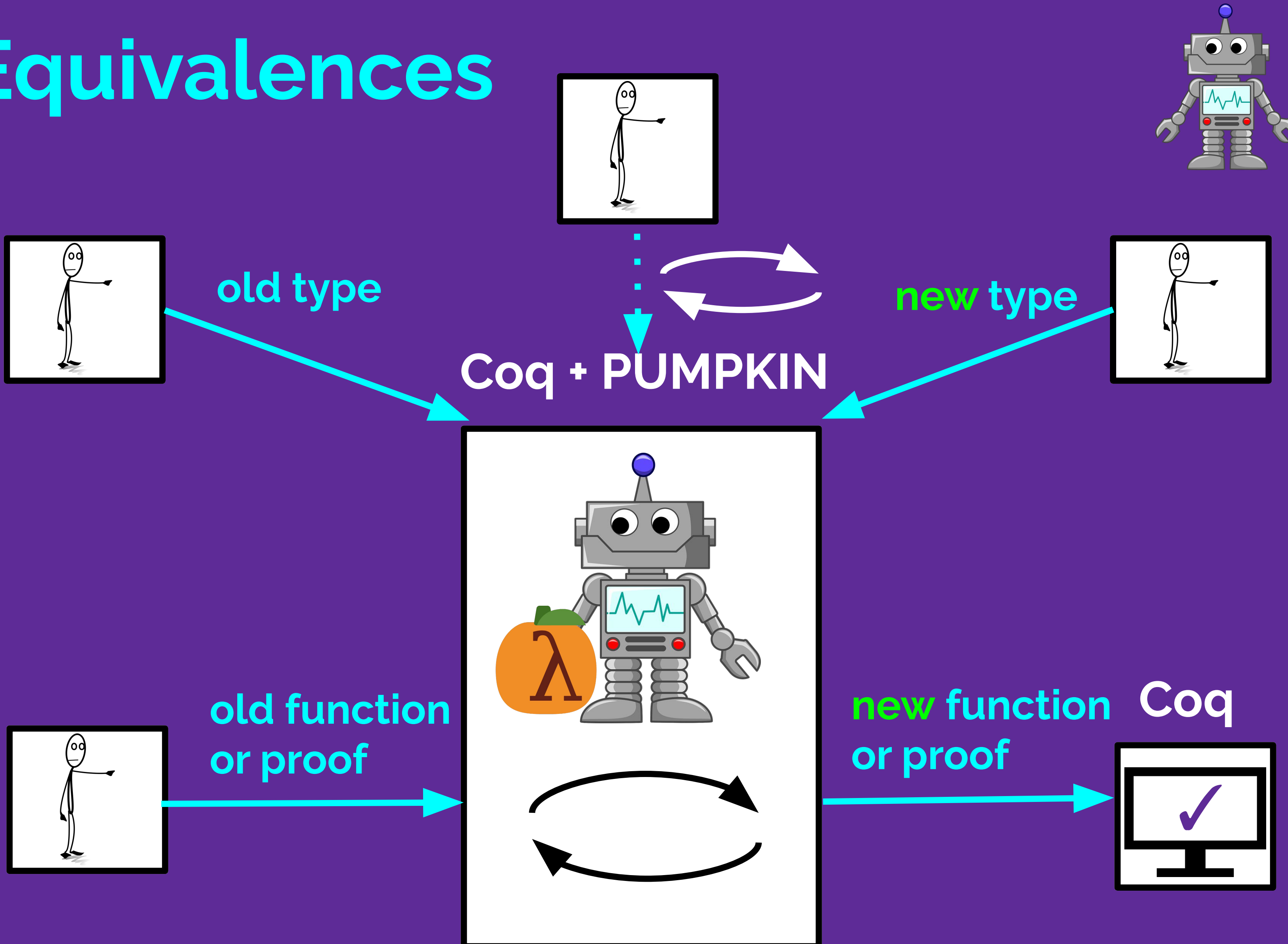
Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

PUMPKIN Pi is flexible & useful for real scenarios.

Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

Equivalences are even more expressive than they may sound.

Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

Adding New Information

Traditional Automation (Part 2 of 5)

Traditional proof repair:

+ predictable
+ dependable
+ understandable
- limited in scope
- takes expertise to extend

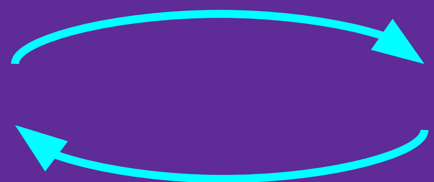
Traditional Automation (Part 2 of 5)

Traditional proof repair:

+ predictable
+ dependable
+ understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair – Understandable



Transport: Rewriting across Equivalences

The Univalent Foundations Program. 2013. **Homotopy Type Theory: Univalent Foundations of Mathematics.** Institute for Advanced Study.

Traditional Automation (Part 2 of 5)

Proof Repair – Understandable

Transport as a Proof Term Transformation

Traditional Automation (Part 2 of 5)

For type nerds: Deconstruct Equivalence (Lambek's Theorem)

Traditional proof repair:

+ predictable
+ dependable
+ understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair – Limited Scope

Proof Repair across Quotient Type Equivalences

Internal and External Views

COSMO VIOLA, University of Illinois Urbana-Champaign, USA
MAX FAN, University of Illinois Urbana-Champaign, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

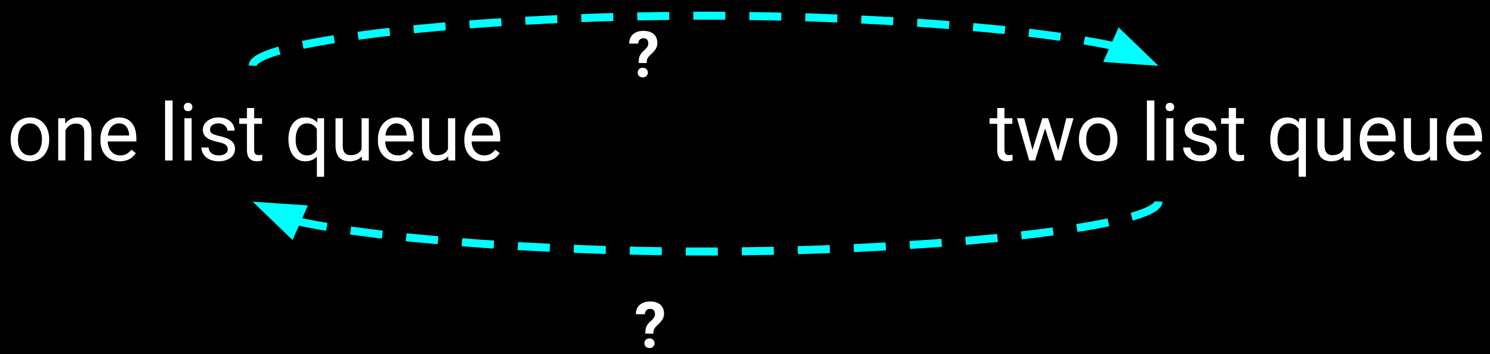
Proofs in proof assistants like Coq can be brittle, breaking easily in response to changes in the terms and types those proofs depend on. To address this, recent work introduced an algorithm and tool in Coq to automatically repair broken proofs in response to changes that correspond to type equivalences. However, many changes remained out of the scope of this algorithm and tool—especially changes in underlying *behavior*. We extend this proof repair algorithm so that it can express certain changes in behavior that were previously out of scope. We focus in particular on equivalences between *quotient types*—types equipped with a relation that describes what it means for any two elements of that type to be equal. Quotient type equivalences can be used to express interesting changes in representations of mathematical structures, as well as changes in the underlying implementations of data structures—two use cases highlighted by our case studies.

We extend this algorithm to support quotient type equivalences in two different ways: (1) internally to cubical type theory (applied to Cubical Agda), and (2) externally to CIC_ω (applied to Coq). While our approach in Coq comes equipped with prototype automation, it suffers notably from Coq's lack of quotient types—something we circumvent using Coq's setoid machinery and an extension to the proof repair algorithm to support the corresponding new proof obligations. In contrast, while our approach in Cubical Agda is completely manual, it takes advantage of cubical type theory's internal quotient types, which makes the algorithm straightforward. Furt[illegible] of correctness of repaired proofs, something not possible in gen[illegible]ween these two approaches, and demonstrate these tradeoffs on proof repair case studies for previously unsupported changes.

Under Submission

Traditional Automation (Part 2 of 5)

Quotient Type Equivalences



Traditional Automation (Part 2 of 5)

Traditional proof repair:

+ predictable
+ dependable
+ understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair – Hard to Extend

One PhD student,
one undergraduate,
one advisor,
2.5 years.
Is this sustainable?

Traditional Automation (Part 2 of 5)

Proof Repair – Hard to Extend

One PhD student,
one undergraduate,
one advisor,
2.5 years.
Is this sustainable?

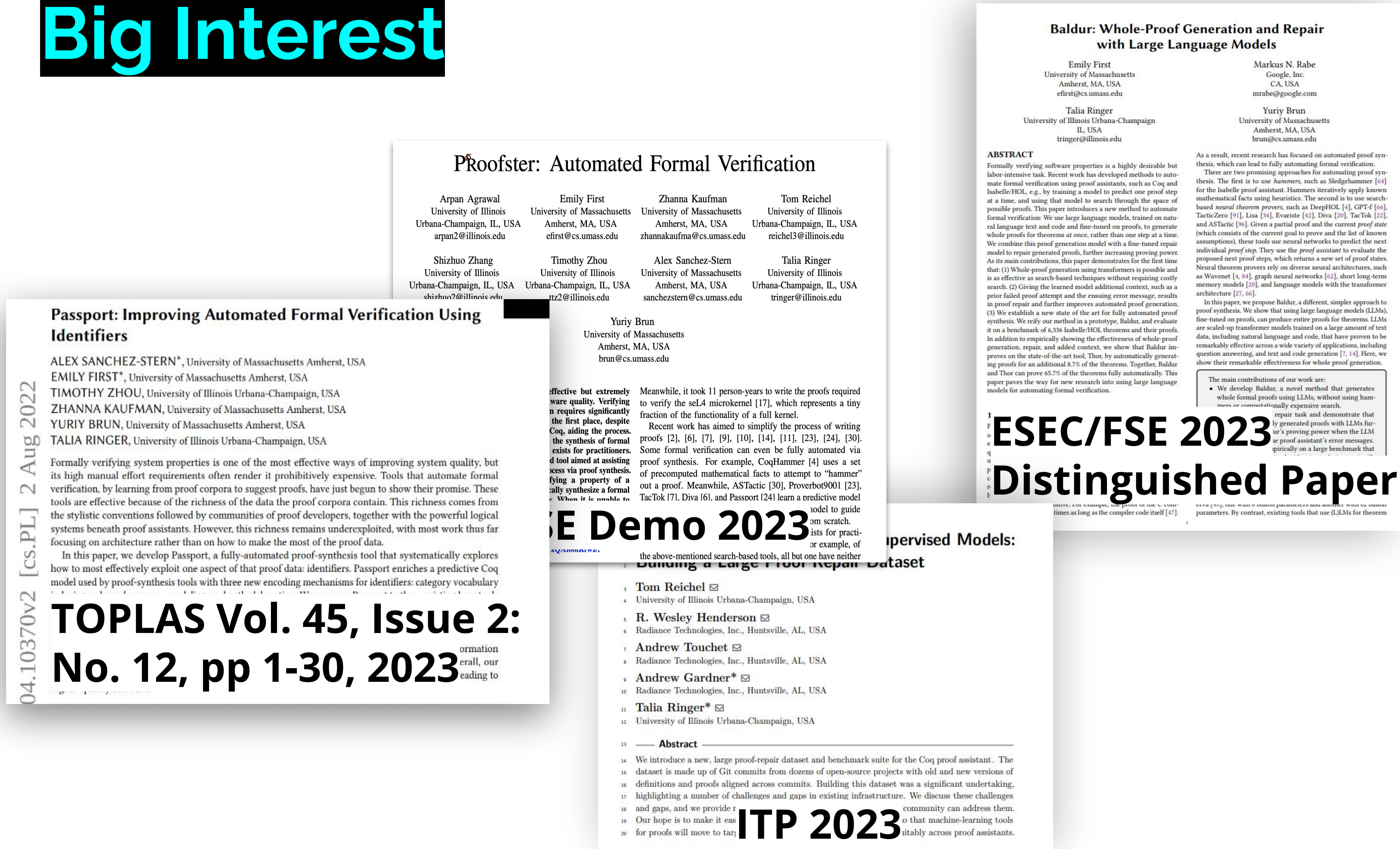
1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities

Language models:

- *un*predictable
- *not* dependable
- *not* understandable
+ *not very* limited in scope
+ takes *little* expertise to extend

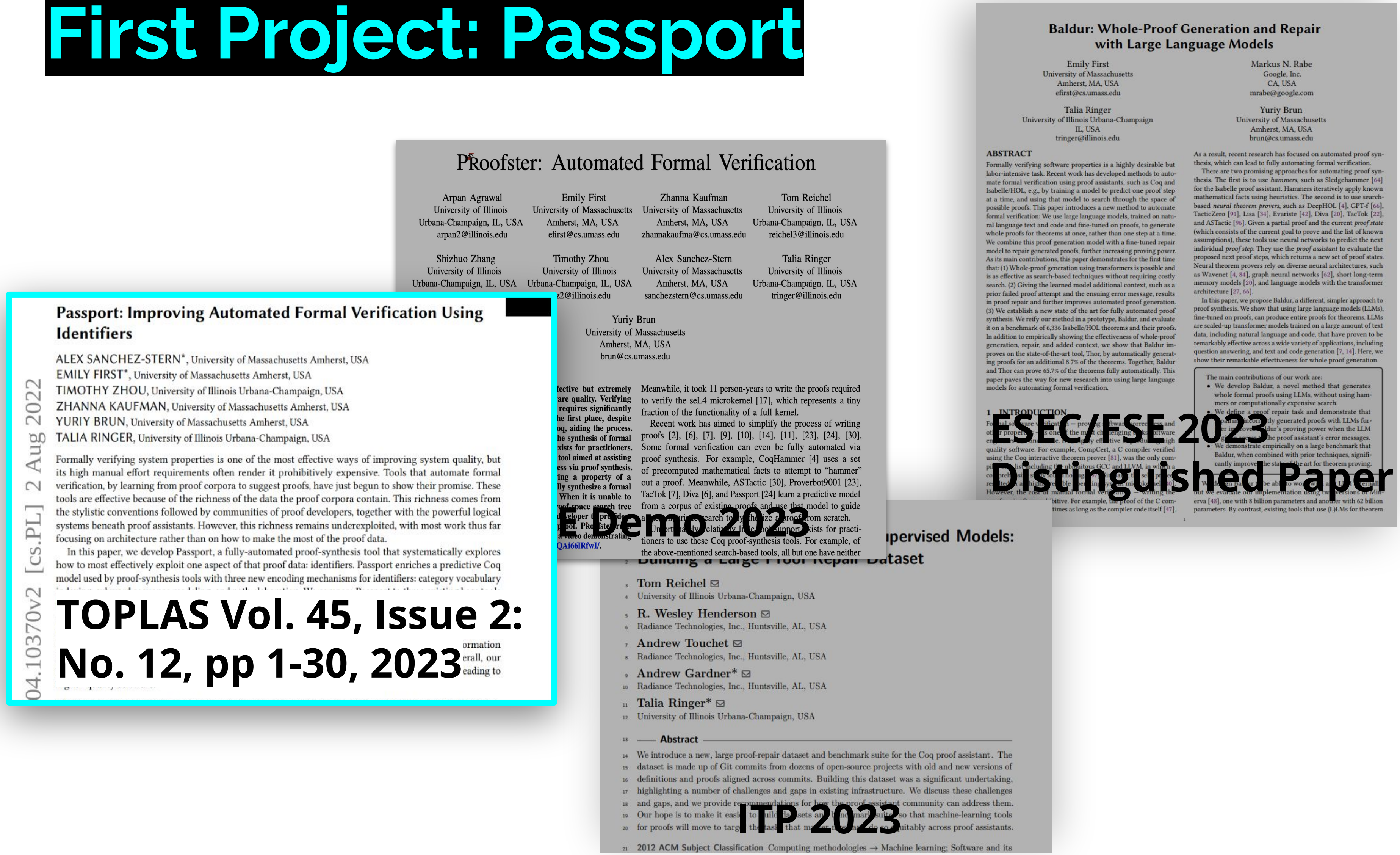
LM-Based Automation (Part 3 of 5)

Big Interest



LM-Based Automation (Part 3 of 5)

First Project: Passport

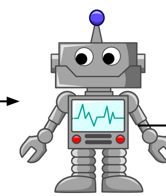


LM-Based Automation (Part 3 of 5)

First Project: Passport

Addition of real numbers is commutative

forall r1 r2: R,
Rplus r1 r2 = Rplus r2 r1



Next Tactic

LM-Based Automation (Part 3 of 5)

First Project: Passport

forall r1 r2: R,
Rplus r1 r2 = Rplus r2 r1

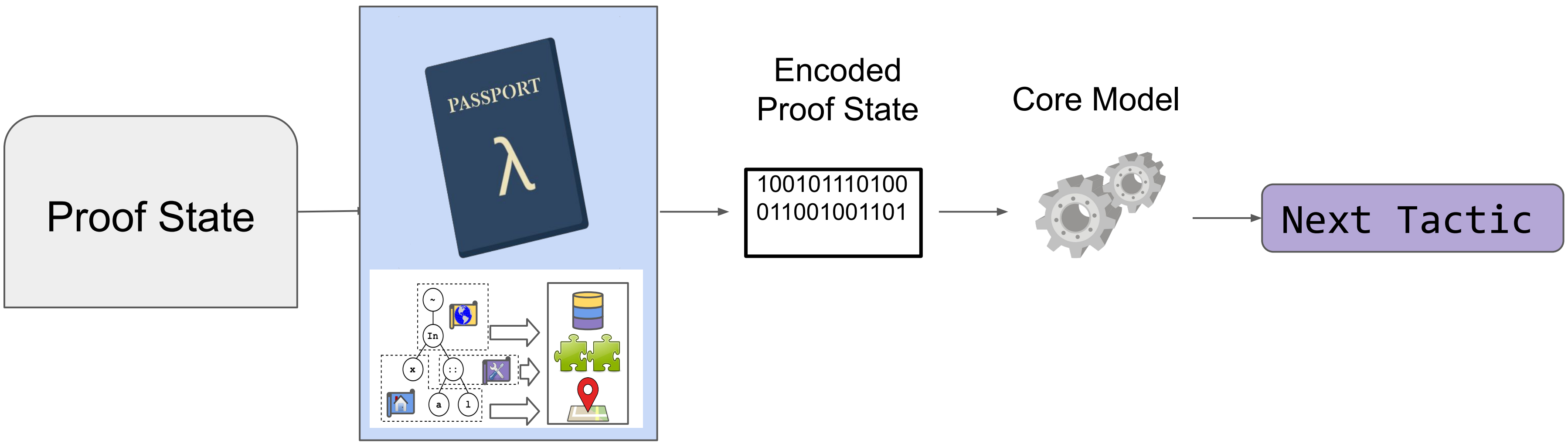
Category Vocabulary Indexing

Subword Sequence Modeling

Path Elaboration

LM-Based Automation (Part 3 of 5)

First Project: Passport



First Project: Passport

Language models:

- *un*predictable
- *not* dependable
- *not* understandable

\+ *not very* limited in scope

\+ takes *little* expertise to extend

LM-Based Automation (Part 3 of 5)

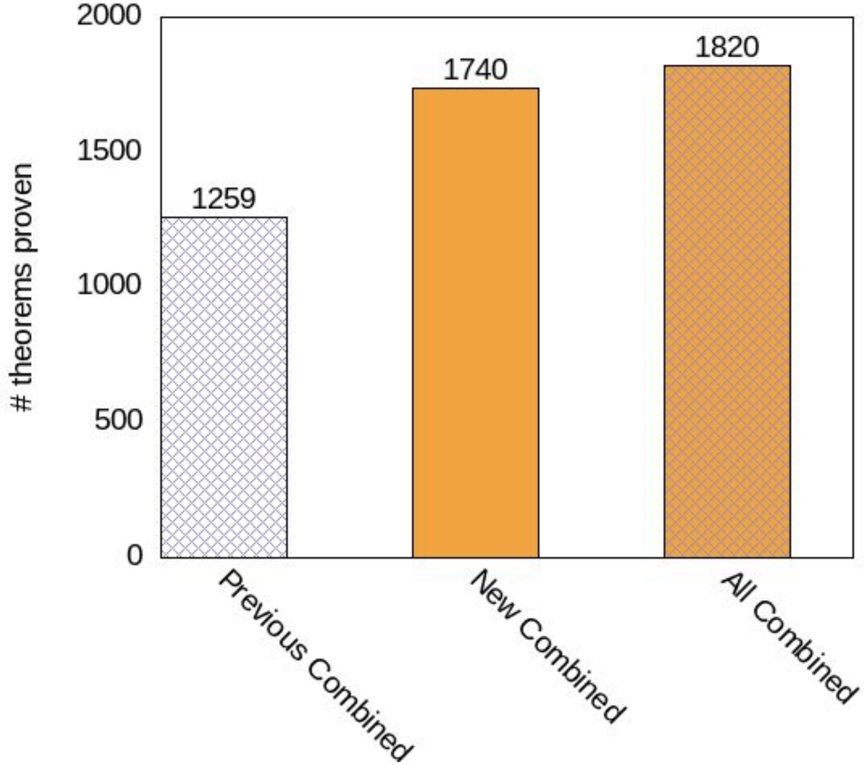
First Project: Passport – Big Scope

- Yang and Deng 2019
- **Math**ematical formalizations, proven correct **programs**, and Coq automation **libraries**
- 123 open-source Coq projects
- Trained on **97 projects** (**57,719 theorems**)
- Tested on **26 projects** (**10,782 theorems**)

CoqGym

First Project: Passport – Big Scope

We can prove **45% more** theorems than before!

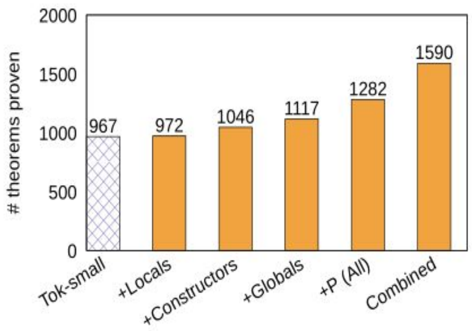


LM-Based Automation (Part 3 of 5)

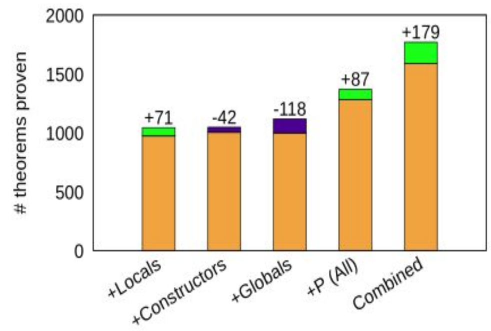
First Project: Passport – Big Scope

Diversity brings even higher returns!

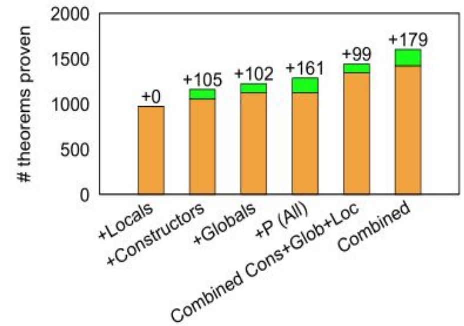
64% more theorems than the baseline!



(a) The impact of category vocabulary indexing on three identifier categories (without subwords or paths): local variables, type constructors, and global definitions.



(b) The impact of subword encoding on each of the categories of identifiers (with category vocabulary indexing but without paths).



(c) The impact of fully-qualified path encoding of type constructors and global definitions (with category vocabulary indexing but without subwords).

First Project: Passport – Easy to Extend

- Some **easy Python scripts** on top of someone else's **existing project**
- **Parallelized work** for different extensions between me and five other authors
- **Undergraduate** implemented most challenging extension in an order of **weeks**
- Scripts were **simple and fun** enough that I got excited when writing one in between drafting thesis chapters, ran into a couch, and broke my big toe

First Project: Passport

Language models:

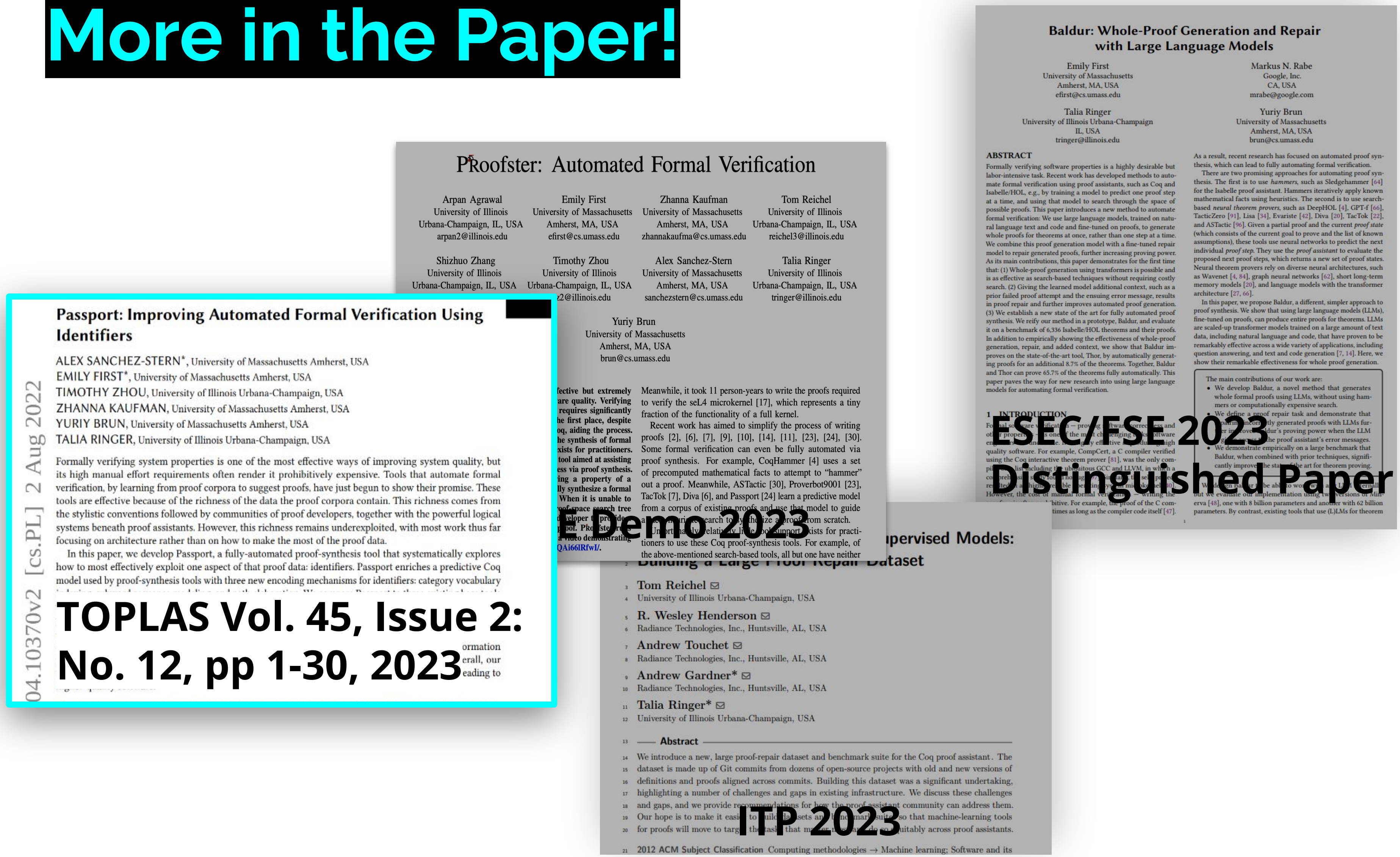
- *un*predictable
- *not* dependable
- *not* understandable
+ *not very* limited in scope
+ takes *little* expertise to extend

LM-Based Automation (Part 3 of 5)

First Project: Passport – Confusion

- Somehow, the ***name*** of the user running the training script impacted the **file order**, which impacted the **results** of training a model on **identical data** in an **identical way**
- We found a **nondeterminism bug in Pytorch**
- **Some combinations** of extensions worked **mysteriously poorly**, even though all together they helped
- Apparently **this is just life** with even small **LMs**? Is this life now? **Help?**

More in the Paper!



LM-Based Automation (Part 3 of 5)

Since Then

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Markus N. Rabe, Google, Inc., CA, USA, mrabe@google.com
Talia Ringer, University of Illinois Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

ESEC/FSE 2023 Distinguished Paper

PRoofster: Automated Formal Verification

Arpan Agrawal, University of Illinois, Urbana-Champaign, IL, USA, arpan2@illinois.edu
Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Zhanna Kaufman, University of Massachusetts, Amherst, MA, USA, zhannakaufma@cs.umass.edu
Tom Reichel, University of Illinois, Urbana-Champaign, IL, USA, reichel3@illinois.edu
Shizhuo Zhang, University of Illinois, Urbana-Champaign, IL, USA, shizhuo2@illinois.edu
Timothy Zhou, University of Illinois, Urbana-Champaign, IL, USA, ttz2@illinois.edu
Alex Sanchez-Stern, University of Massachusetts, Amherst, MA, USA, sanchezstern@cs.umass.edu
Talia Ringer, University of Illinois, Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

ICSE Demo 2023

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

arXiv:2204.10370v2 [cs.PL] 2 Aug 2022

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-30, 2023

Proof Repair Infrastructure for Supervised Models: Building a Large Proof Repair Dataset

Tom Reichel, University of Illinois Urbana-Champaign, USA
R. Wesley Henderson, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Touchet, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Gardner*, Radiance Technologies, Inc., Huntsville, AL, USA
Talia Ringer*, University of Illinois Urbana-Champaign, USA

ITP 2023

LM-Based Automation (Part 3 of 5)

Second Project: Proofster

Baldur: Whole-Proof Generation and Repair with Large Language Models

ESEC/FSE 2023 Distinguished Paper

Proofster: Automated Formal Verification

ICSE Demo 2023

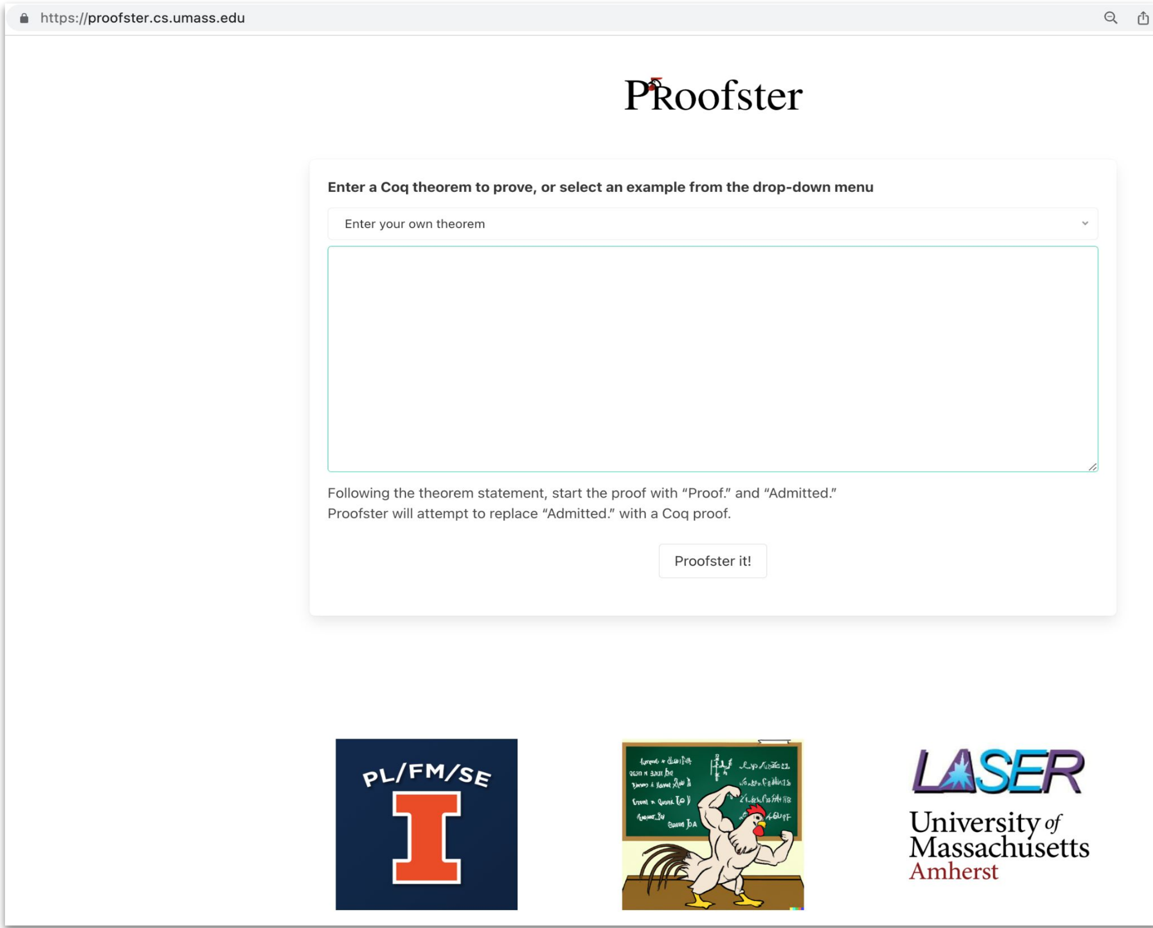
Passport: Improving Automated Formal Verification Using Identifiers

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-30, 2023

ITP 2023

LM-Based Automation (Part 3 of 5)

Second Project: Proofster



https://proofster.cs.umass.edu/

LM-Based Automation (Part 3 of 5)

Second Project: Proofster

```
Inductive ev: nat → Prop :=
    | ev_0 : ev 0
    | ev_SS (n: nat) (H: ev n) : ev (S (S n)).

Theorem ev_inversion: forall (n: nat),
    ev n →
    (n = 0) ∨ (exists n', n = S (S n') ∧ ev n').
Proof.
intros.
```

```
n : nat   H : ev n
────────────────────────────────
n = 0 ∨ (exists n' : nat, n = S (S n') ∧ ev n')
```

```
elim H.
left.
eauto.
```

```
n : nat   H : ev n
────────────────────────────────
forall n : nat,
ev n →
n = 0 ∨ (exists n' : nat, n = S (S n') ∧ ev n') →
S (S n) = 0 ∨
(exists n' : nat, S (S n) = S (S n') ∧ ev n')
```

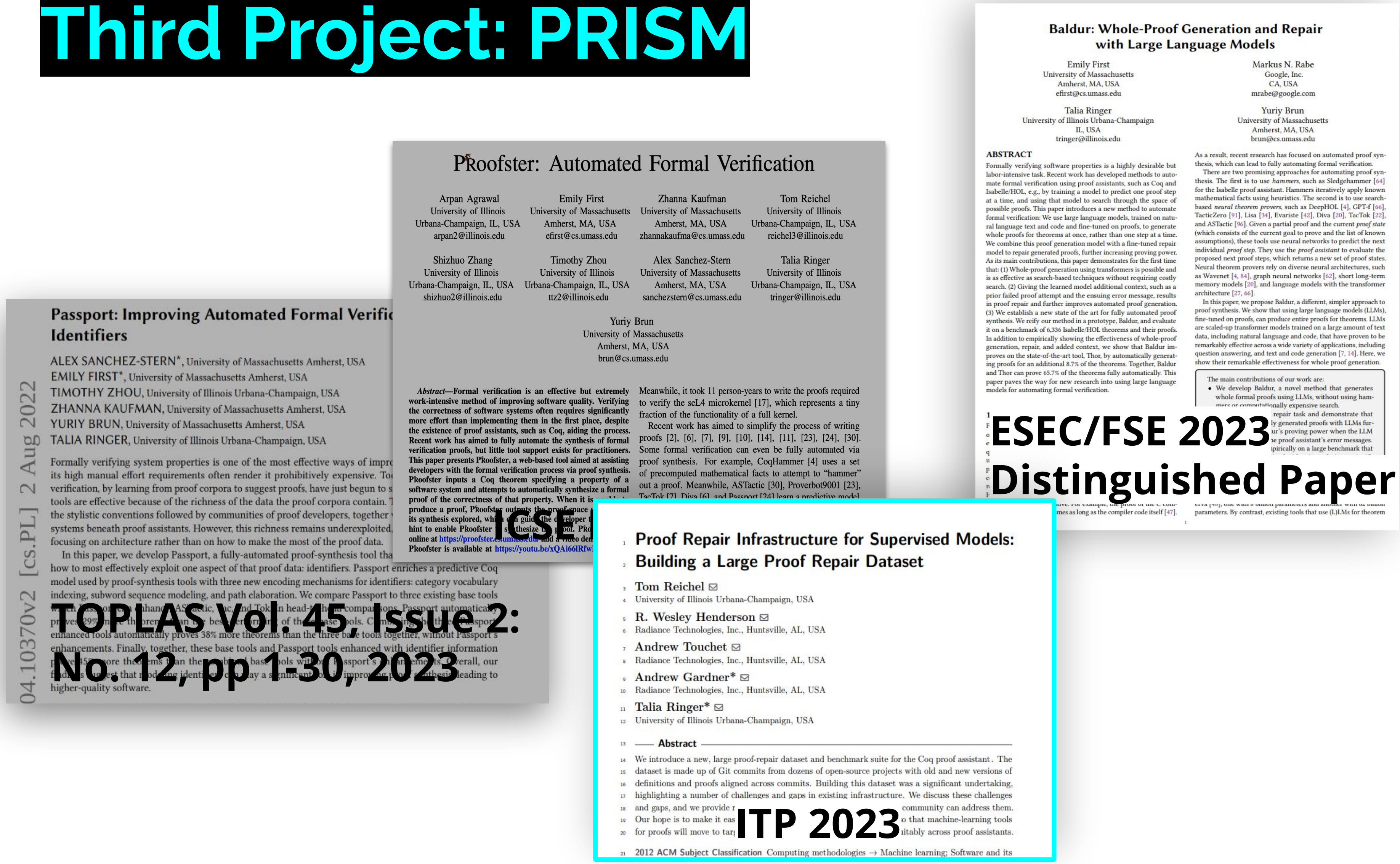
```
intros.
destruct H1.
```

```
n : nat   H : ev n   n0 : nat   H0 : ev n0   H1 : n0 = 0
────────────────────────────────
S (S n0) = 0 ∨
(exists n' : nat, S (S n0) = S (S n') ∧ ev n')
- - - - - - - - - - - - - - - - - - - - - - - -
S (S n0) = 0 ∨
(exists n' : nat, S (S n0) = S (S n') ∧ ev n')
```

```
eauto.
eauto.
Qed.
```

LM-Based Automation (Part 3 of 5)

Third Project: PRISM



LM-Based Automation (Part 3 of 5)

Third Project: PRISM

- **Dataset** for **proof repair** models for Coq
- **Actual proof repairs** by proof engineers
- **Collaboration with Radiance**
- **Massive infrastructure undertaking**
  - Building many different projects
  - … with many different Coq versions
  - … for many different commits
  - … and aligning data across commit pairs
- **WIP Training Repair Models**

Fourth Project: Baldur

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Markus N. Rabe, Google, Inc., CA, USA, mrabe@google.com
Talia Ringer, University of Illinois Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

ESEC/FSE 2023 Distinguished Paper

PRoofster: Automated Formal Verification

Arpan Agrawal, University of Illinois, Urbana-Champaign, IL, USA, arpan2@illinois.edu
Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Zhanna Kaufman, University of Massachusetts, Amherst, MA, USA, zhannakaufma@cs.umass.edu
Tom Reichel, University of Illinois, Urbana-Champaign, IL, USA, reichel3@illinois.edu
Shizhuo Zhang, University of Illinois, Urbana-Champaign, IL, USA, shizhuo2@illinois.edu
Timothy Zhou, University of Illinois, Urbana-Champaign, IL, USA, ttz2@illinois.edu
Alex Sanchez-Stern, University of Massachusetts, Amherst, MA, USA, sanchezstern@cs.umass.edu
Talia Ringer, University of Illinois, Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

ICSE

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-30, 2023

Proof Repair Infrastructure for Supervised Models: Building a Large Proof Repair Dataset

Tom Reichel, University of Illinois Urbana-Champaign, USA
R. Wesley Henderson, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Touchet, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Gardner*, Radiance Technologies, Inc., Huntsville, AL, USA
Talia Ringer*, University of Illinois Urbana-Champaign, USA

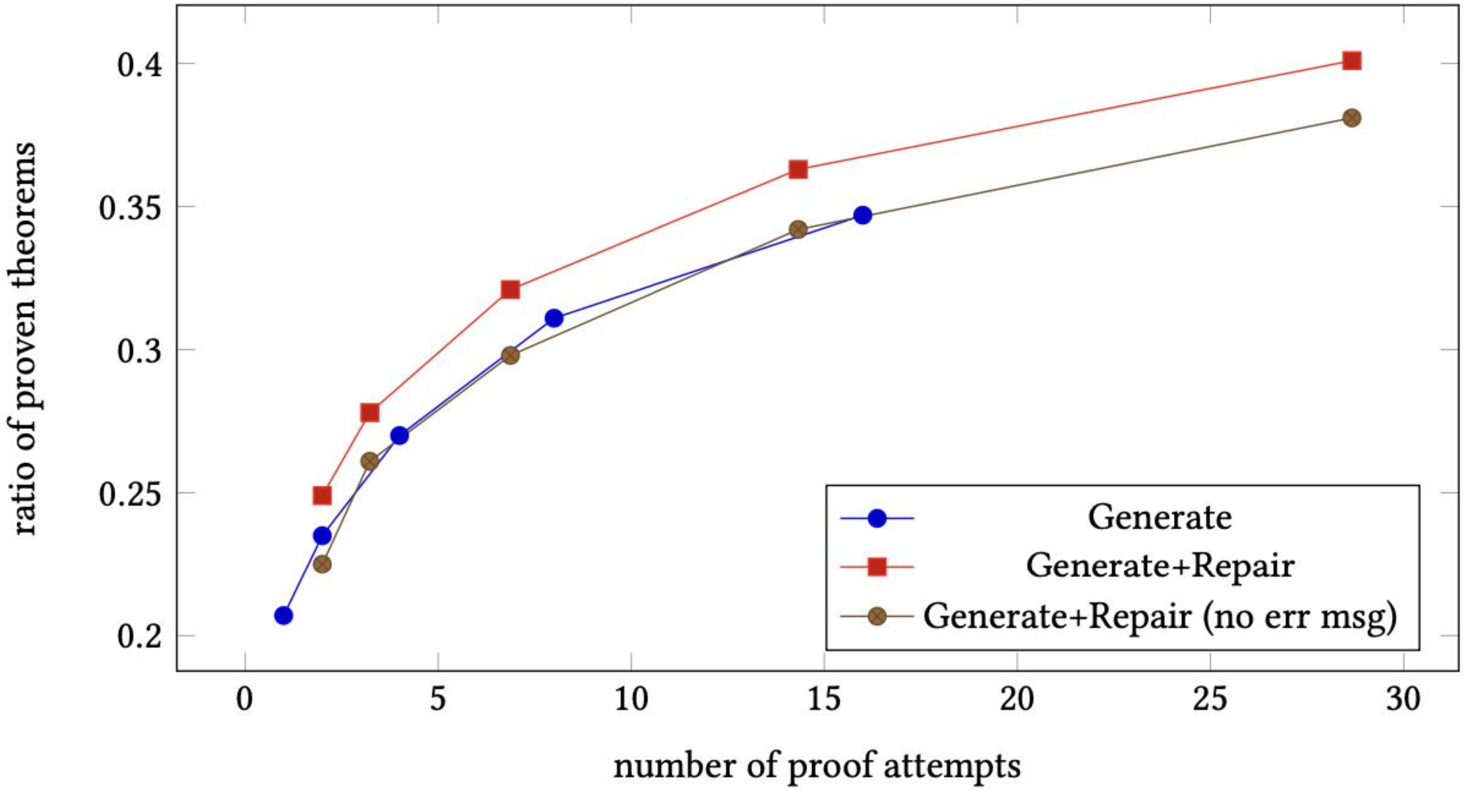
ITP 2023

LM-Based Automation (Part 3 of 5)

Fourth Project: Baldur

- Using an **LLM**, one could, conceivably, synthesize **entire proofs at once**.
- Collaborating with Google, we fine-tuned the Minerva model to synthesize proofs in Isabelle/HOL
- Evaluated on PISA dataset (theorems in Isabelle/HOL)

Fourth Project: Baldur



LM-Based Automation (Part 3 of 5)

Fourth Project: Baldur

- Baldur (without repair) can **synthesize whole proofs** for **47.9%** of the theorems, whereas search-based approaches prove **39.0%**.
- Baldur can **repair its own erroneous proof attempts** using the error message from the proof assistant, proving another **1.5%**.
- **Diversity continues to help**. Together with Thor, a tool that combines a model, search, and a hammer, Baldur can prove **65.7%**.

More in the Papers

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Markus N. Rabe, Google, Inc., CA, USA, mrabe@google.com
Talia Ringer, University of Illinois Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

ESEC/FSE 2023 Distinguished Paper

PRoofster: Automated Formal Verification

Arpan Agrawal, University of Illinois Urbana-Champaign, IL, USA, arpan2@illinois.edu
Emily First, University of Massachusetts, Amherst, MA, USA, efirst@cs.umass.edu
Zhanna Kaufman, University of Massachusetts, Amherst, MA, USA, zhannakaufma@cs.umass.edu
Tom Reichel, University of Illinois Urbana-Champaign, IL, USA, reichel3@illinois.edu
Shizhuo Zhang, University of Illinois Urbana-Champaign, IL, USA
Timothy Zhou, University of Illinois Urbana-Champaign, IL, USA
Alex Sanchez-Stern, University of Massachusetts, Amherst, MA, USA, sanchezstern@cs.umass.edu
Talia Ringer, University of Illinois Urbana-Champaign, IL, USA, tringer@illinois.edu
Yuriy Brun, University of Massachusetts, Amherst, MA, USA, brun@cs.umass.edu

…E Demo 2023

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-30, 2023

…pervised Models: Building a Large Proof Repair Dataset

Tom Reichel, University of Illinois Urbana-Champaign, USA
R. Wesley Henderson, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Touchet, Radiance Technologies, Inc., Huntsville, AL, USA
Andrew Gardner*, Radiance Technologies, Inc., Huntsville, AL, USA
Talia Ringer*, University of Illinois Urbana-Champaign, USA

ITP 2023

LM-Based Automation (Part 3 of 5)

Language models:

- *un*predictable
- *not* dependable
- *not* understandable

\+ *not very* limited in scope
\+ takes *little* expertise to extend

LM-Based Automation (Part 3 of 5)

Checking the Proof

Small Logical Kernel

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

LMs

LM-Based Automation (Part 3 of 5)

With de Bruijn, as long as you don't touch the kernel, your automation is safe.

LM-Based Automation (Part 3 of 5)

With de Bruijn, as long as you don't touch the kernel, your automation is safe.*
But *boy* does this make the development process suck.

Help at Every Stage

**With de Bruijn, as long as you don’t touch the kernel, your automation is safe.*
(If your specification is OK, your kernel has no bugs, and you don’t introduce axioms)**

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities

Already Neurosymbolic

Checking the Proof

Small Logical Kernel

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

LMs

Best of Both Worlds (Part 4 of 5)

But we want even more of the benefits of both kinds of automation.

Best of Both Worlds (Part 4 of 5)

Observation: We can do fairly well sometimes without **search**. Maybe we can use search at a **higher level** than before and get further returns?

Best of Both Worlds (Part 4 of 5)

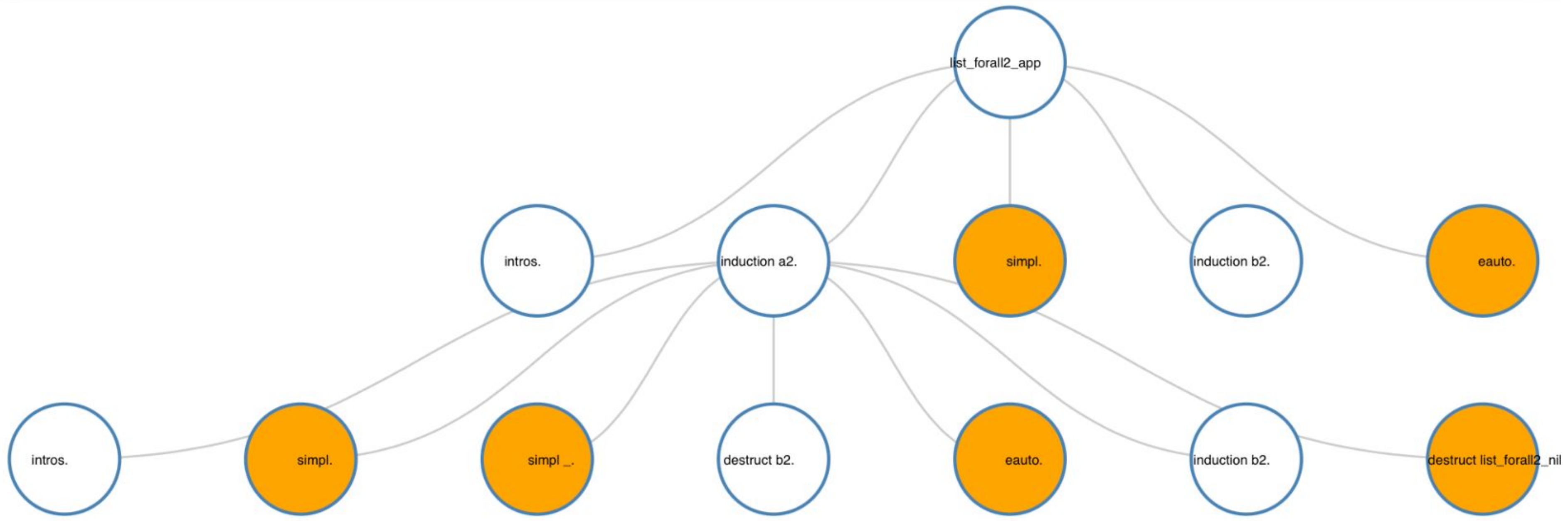
One idea: Move the search process *up* in abstraction.

Best of Both Worlds (Part 4 of 5)

One idea: Move the search process *up* in abstraction.

Best of Both Worlds (Part 4 of 5)

Proof Search



Best of Both Worlds (Part 4 of 5)

Conversational Action Search

Getting More out of Large Language Models for Proofs

Shizhuo Dylan Zhang[1], Emily First[2], and Talia Ringer[1]

[1] University of Illinois Urbana-Champaign, USA
[2] University of Massachusetts Amherst, USA

Abstract

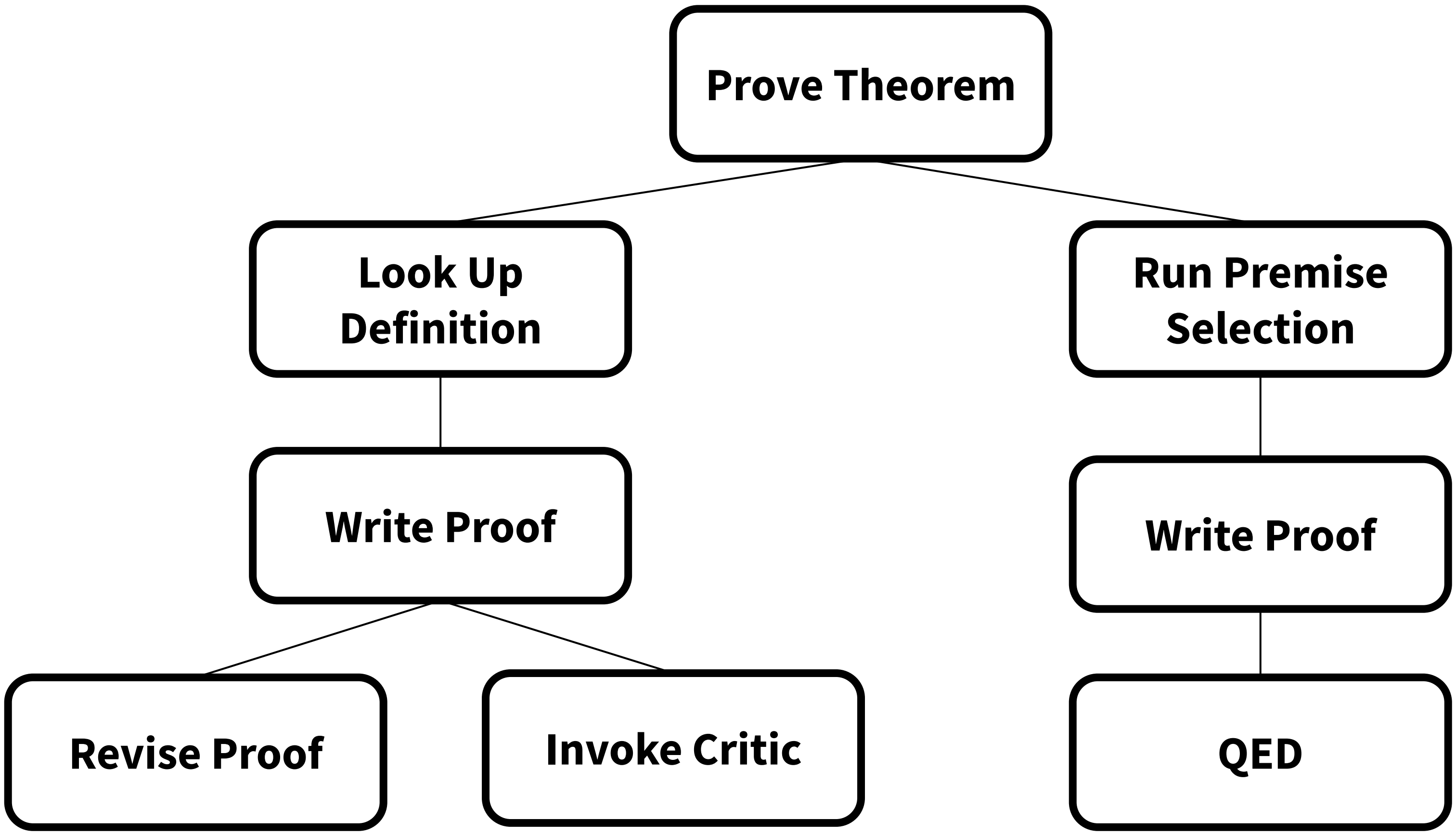
Large language models have the potential to simplify formal theorem proving and make it more accessible. But how to get the most out of these models is still an open question. To answer this question, we take a step back and explore the failure cases of these models using common prompting-based techniques. Our talk will discuss these failure cases and what they can teach us about ho

AITP 2023

ese models.

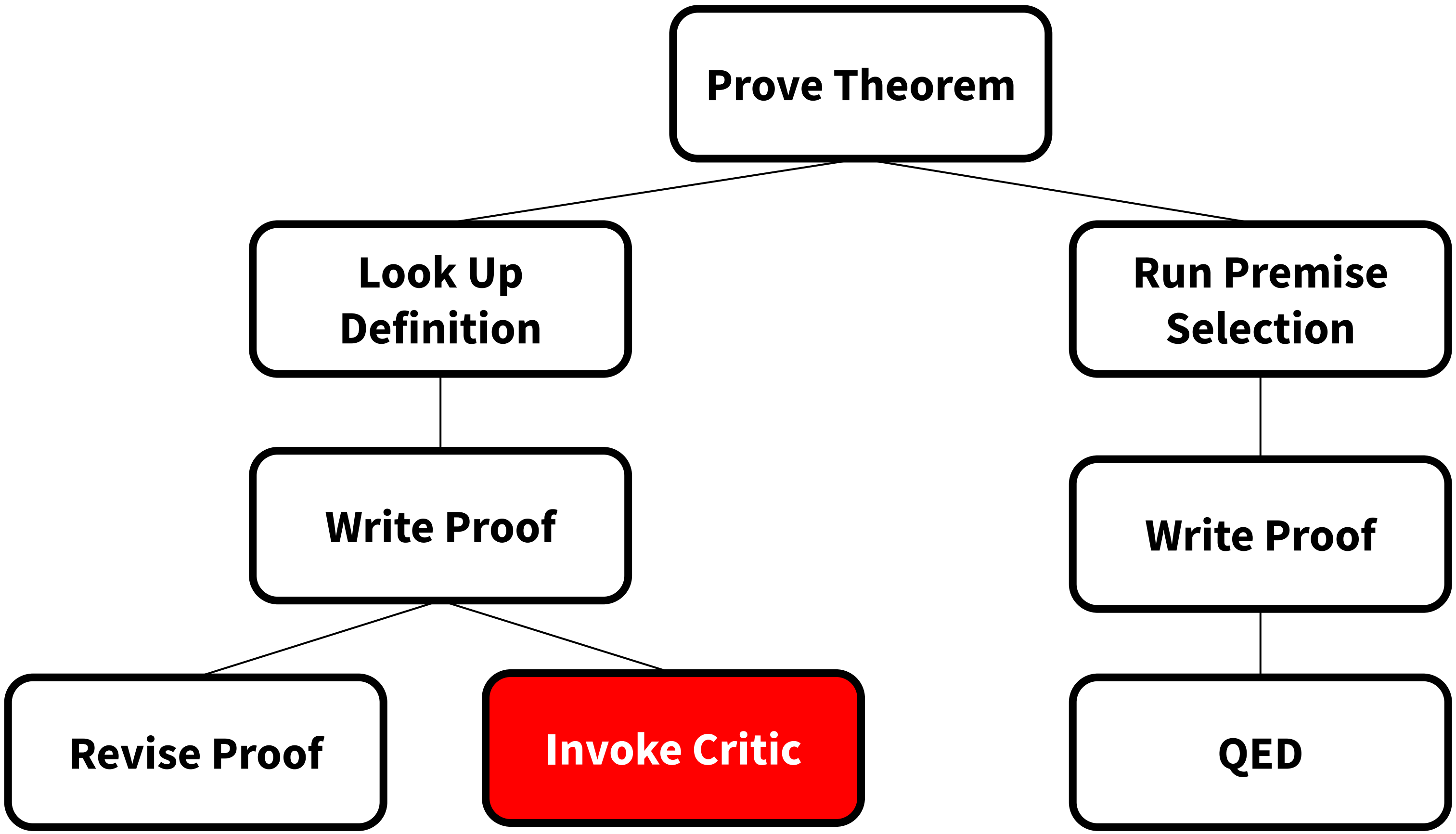
Best of Both Worlds (Part 4 of 5)

Conversational Action Search



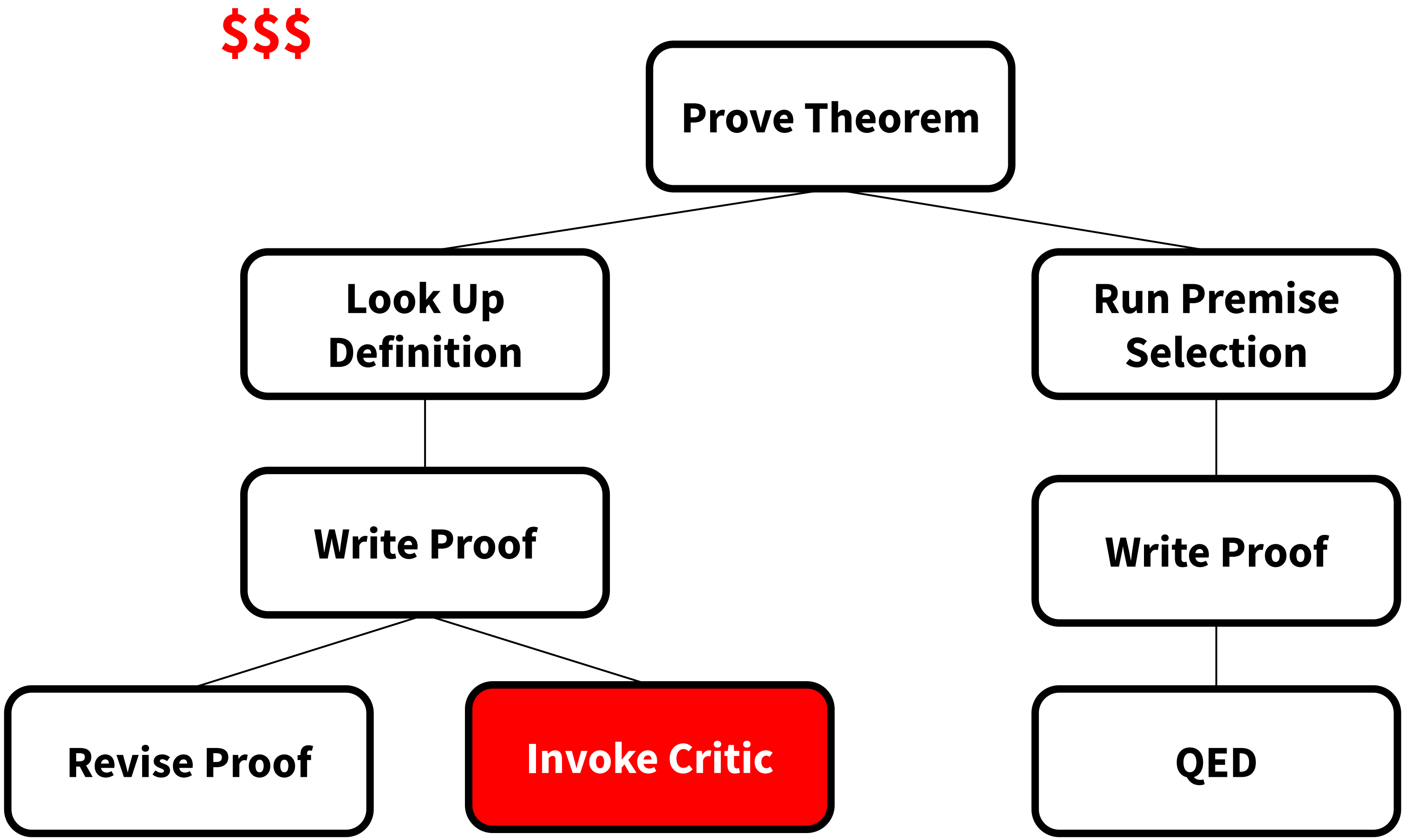
Best of Both Worlds (Part 4 of 5)

Conversational Action Search



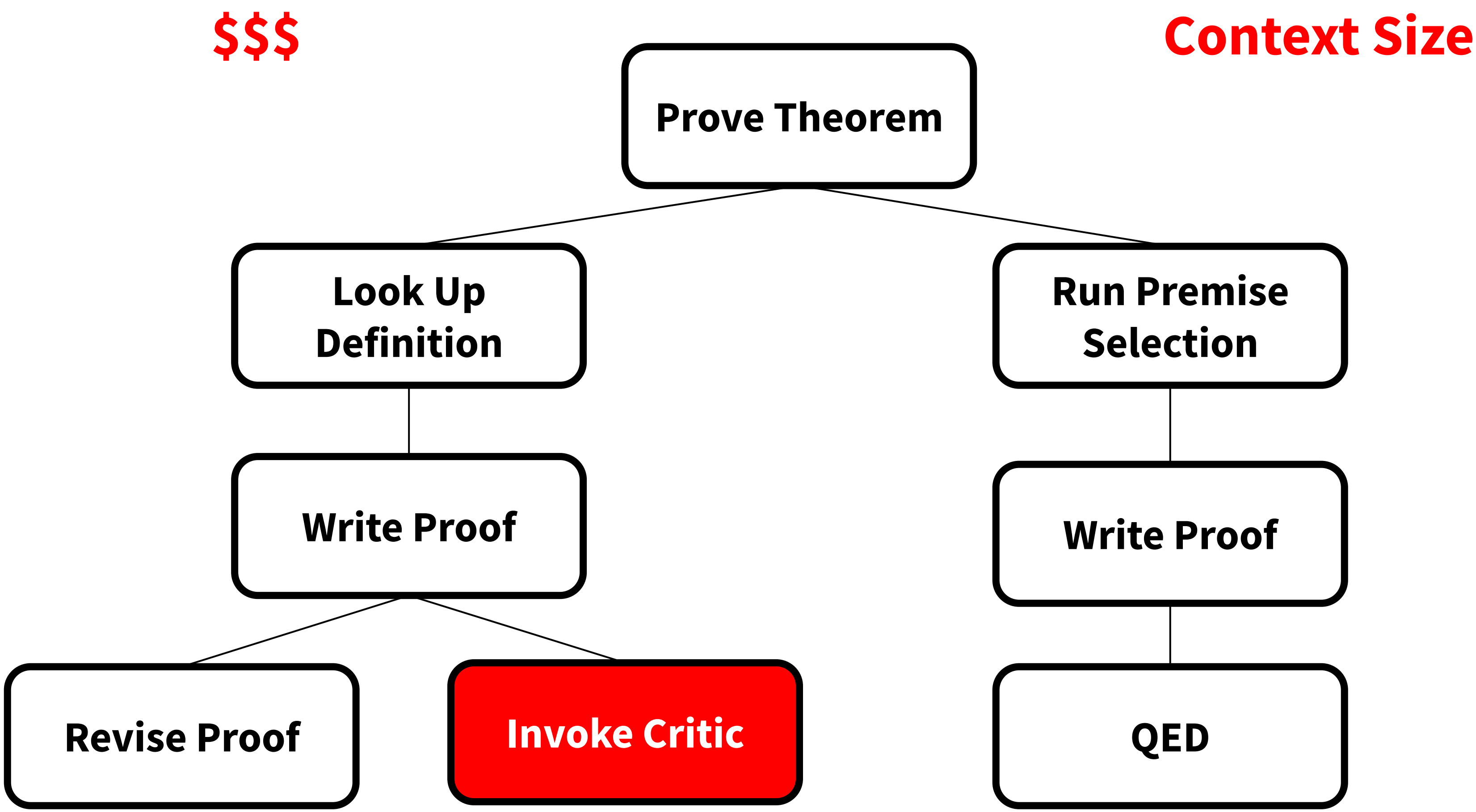
Best of Both Worlds (Part 4 of 5)

Conversational Action Search



Best of Both Worlds (Part 4 of 5)

Conversational Action Search



Best of Both Worlds (Part 4 of 5)

Conversational Action Search

Promising Results

Best of Both Worlds (Part 4 of 5)

Observation: Diversity in **models** helps, and **diversity** in **techniques** appears to help, too. Let's keep taking advantage of that.

Best of Both Worlds (Part 4 of 5)

Soon: Best of both worlds for proof repair, too.

Best of Both Worlds (Part 4 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. **Opportunities**

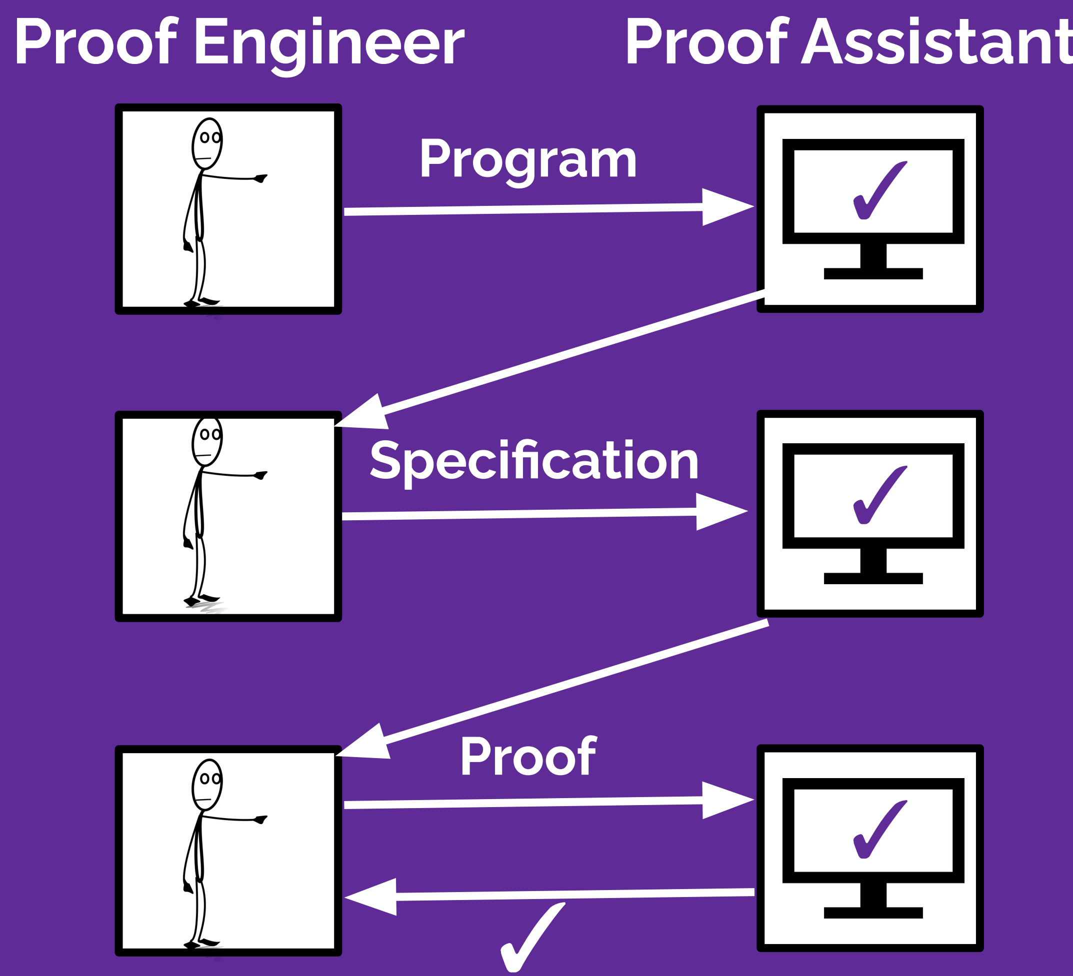
So far I've assumed the **specification** already exists.

What if LMs can help people *specify* software too? This is risky, but promising.

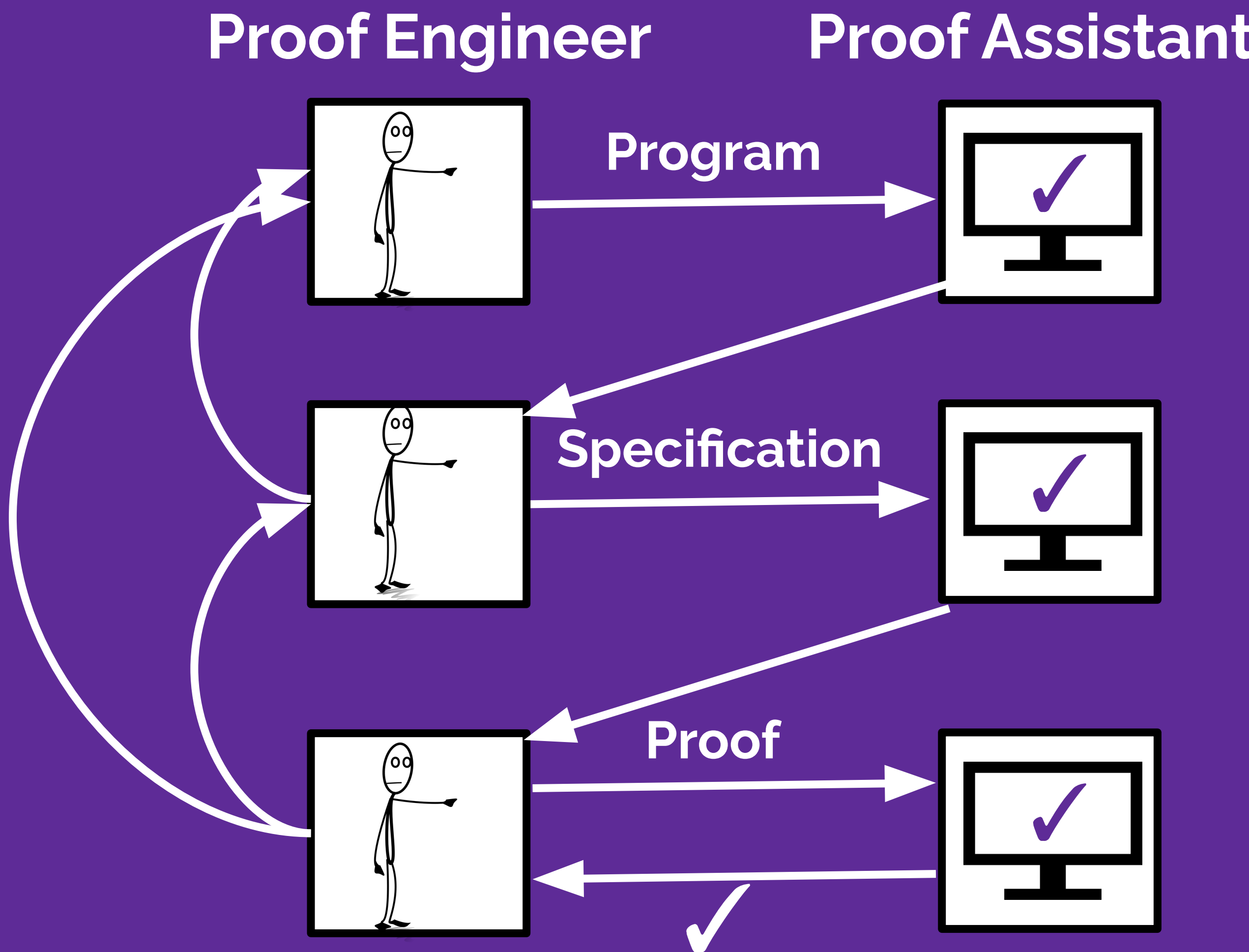
Opportunities (Part 5 of 5)

What if LMs can help people *specify* software too? This is risky, but promising.

Opportunities (Part 5 of 5)

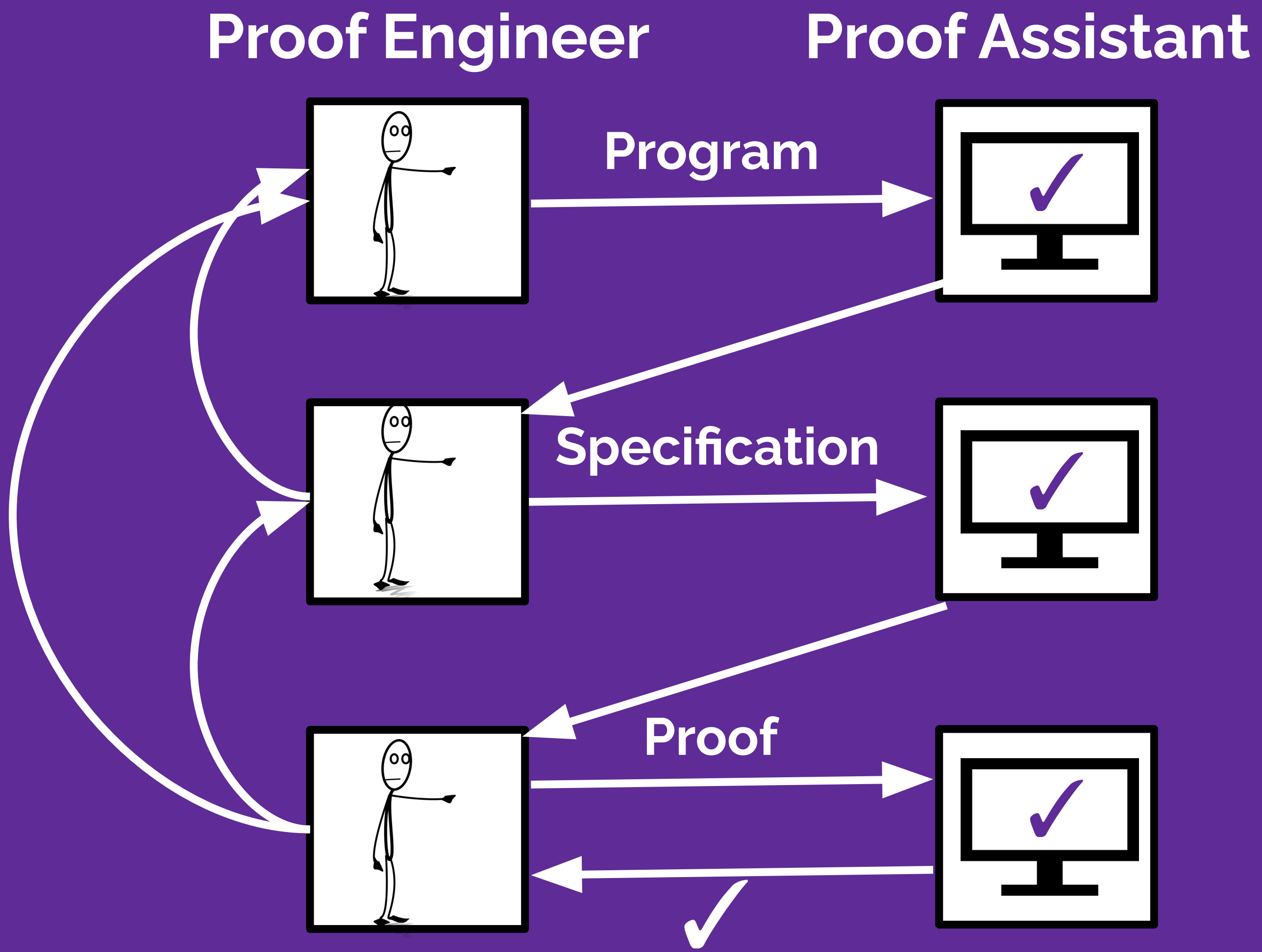


Opportunities (Part 5 of 5)



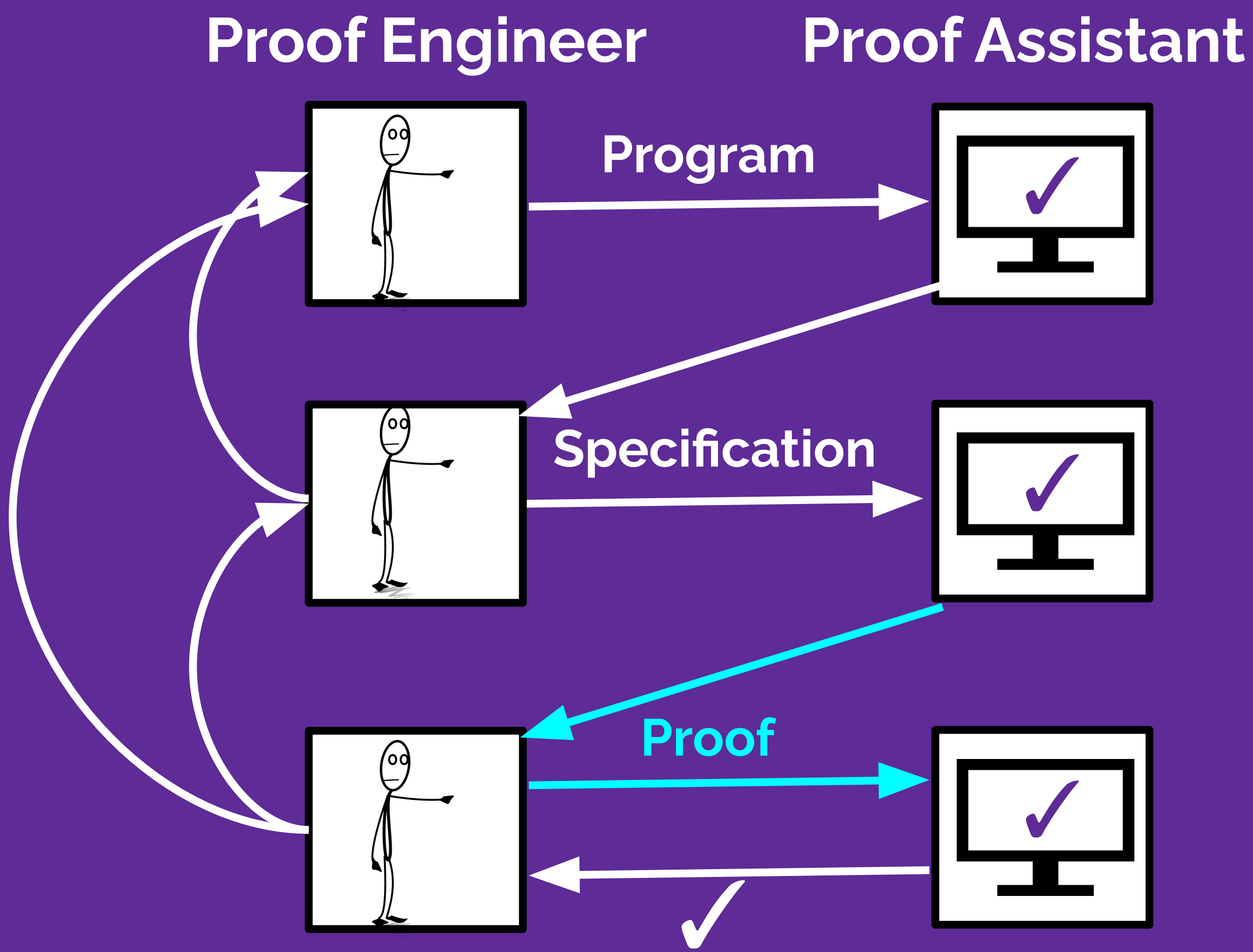
Opportunities (Part 5 of 5)

Help at Every Stage



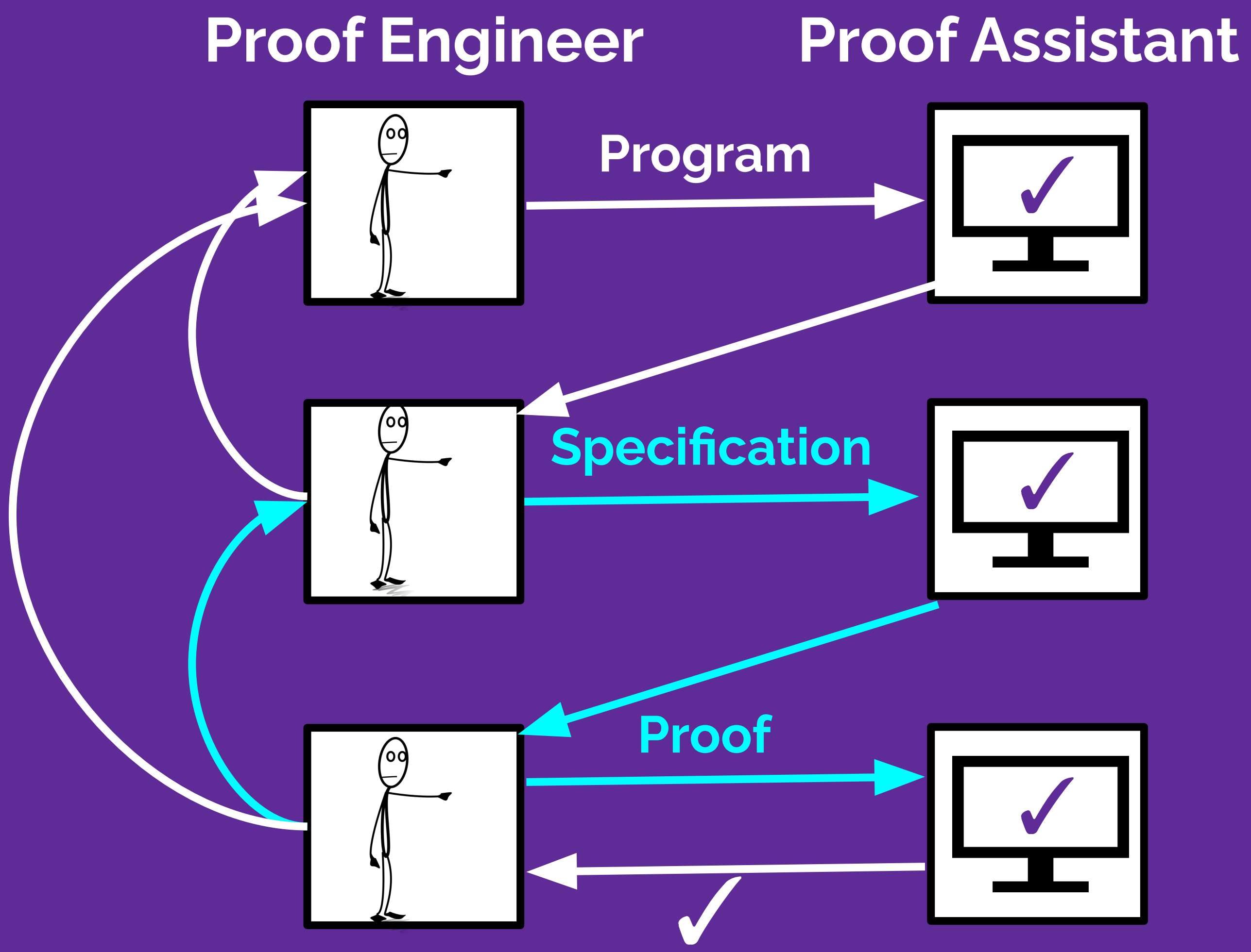
Opportunities (Part 5 of 5)

Help at Every Stage



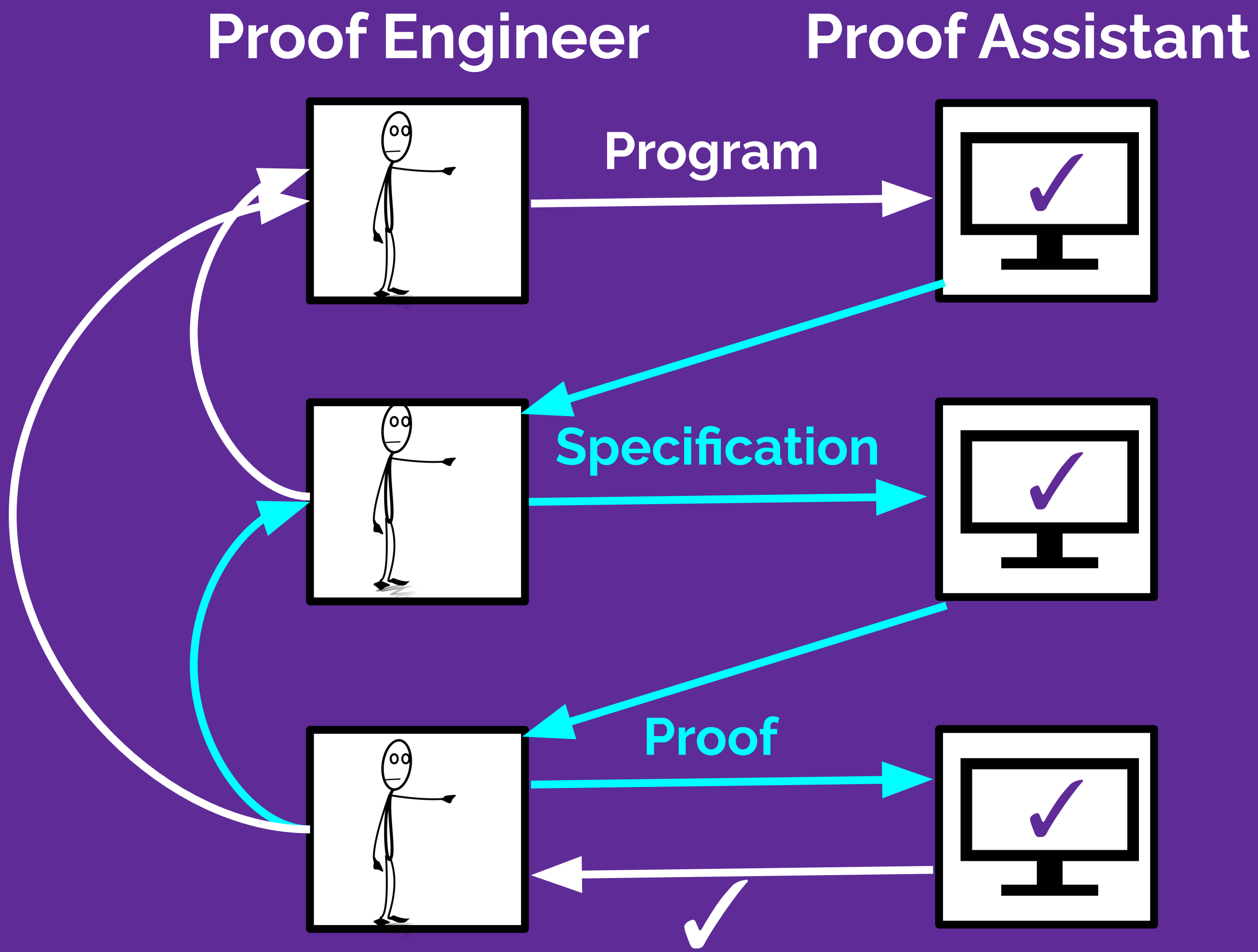
Opportunities (Part 5 of 5)

Help at Every Stage



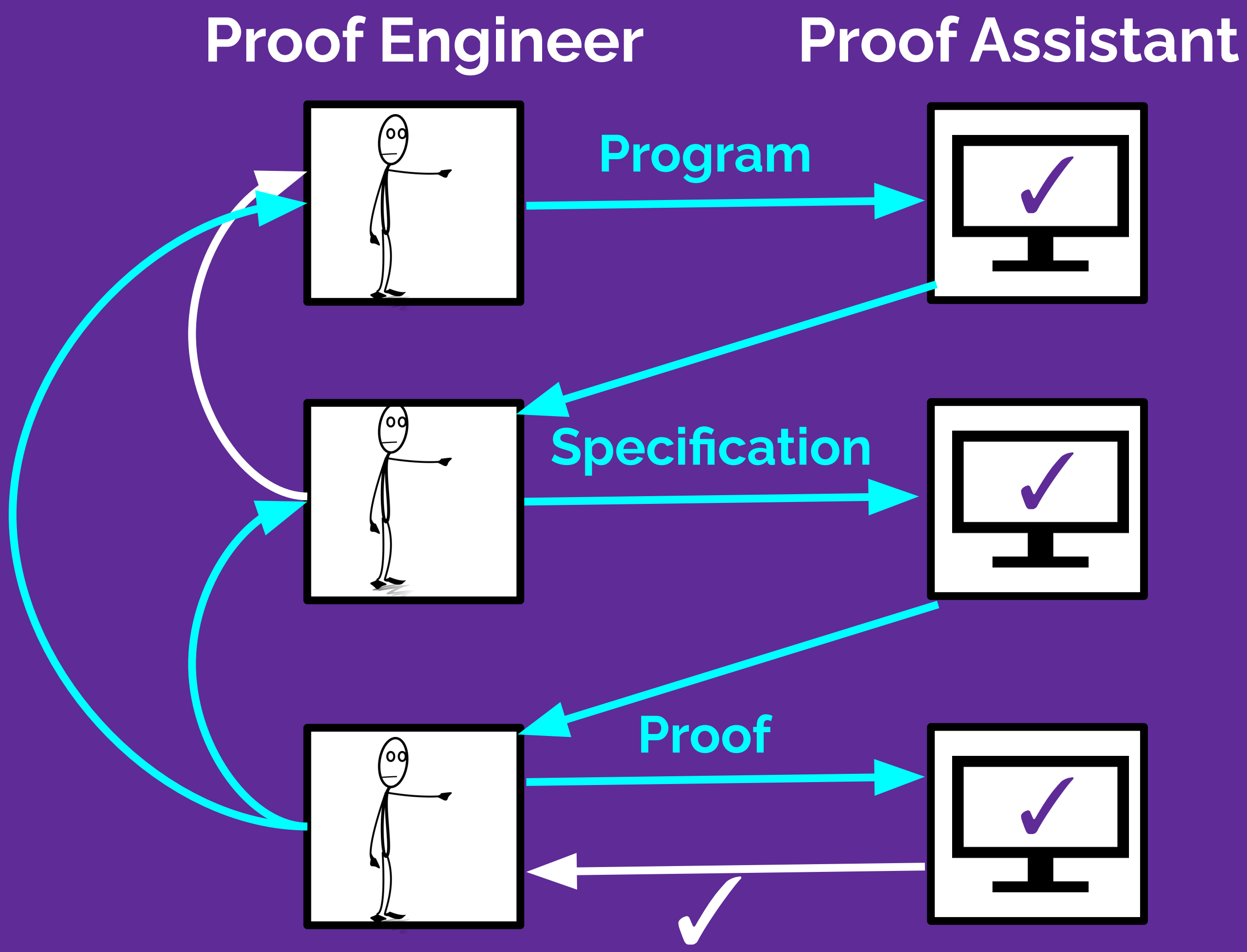
Opportunities (Part 5 of 5)

Help at Every Stage



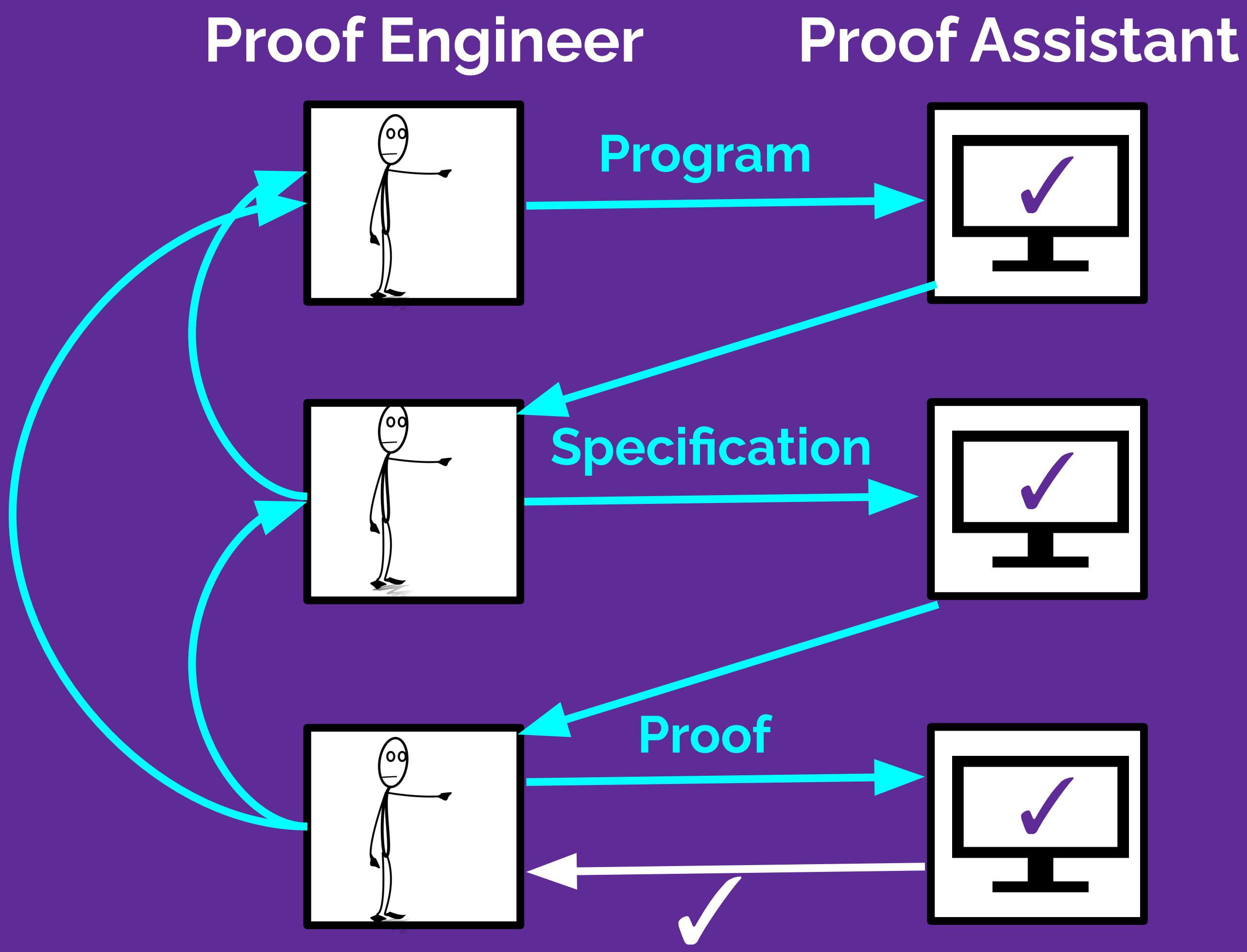
Opportunities (Part 5 of 5)

Help at Every Stage



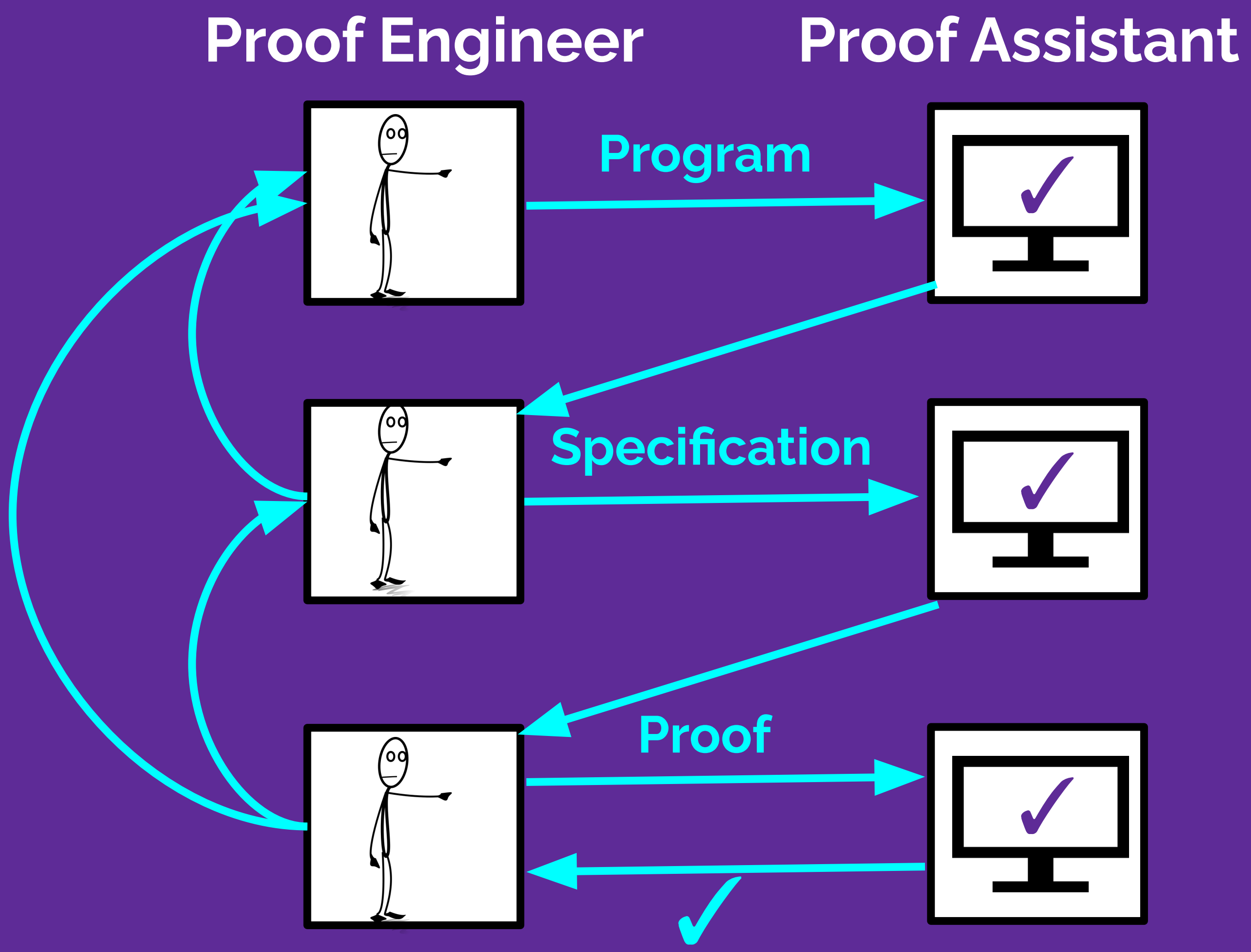
Opportunities (Part 5 of 5)

Help at Every Stage



Opportunities (Part 5 of 5)

Help at Every Stage



Opportunities (Part 5 of 5)

Help at Every Stage

**With de Bruijn, as long as you don't touch the kernel, your automation is safe.*
(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)**

Opportunities (Part 5 of 5)

Help at Every Stage

With de Bruijn, as long as you don't touch the kernel, your automation is safe.*
(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)

Opportunities (Part 5 of 5)

Key Challenge:
There is no oracle for a specification!

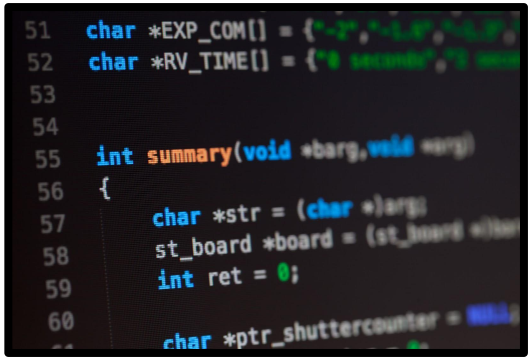
Opportunities (Part 5 of 5)

Key Challenge:

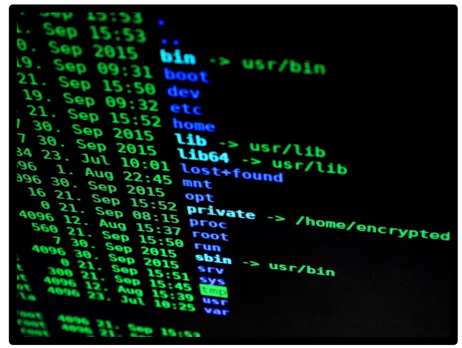
What tools can best help users make sense of generated specifications? What information presented in what ways best helps users ensure that they match their intentions?

Opportunities (Part 5 of 5)

More Trustworthy Software

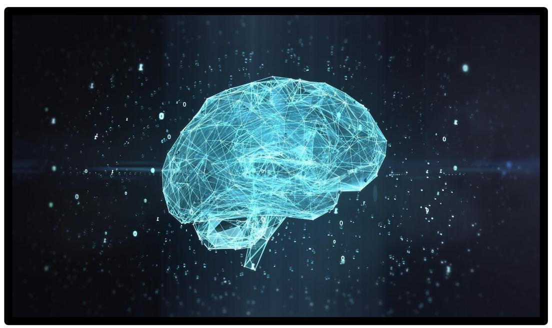


Compilers



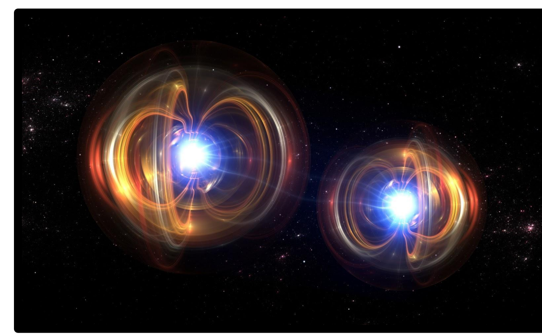
File Systems

Web Browsers



Machine Learning Systems

Operating Systems



Quantum Optimizers

Key Challenge:
What tools can best help users make sense of generated specifications? What information presented in what ways best helps users ensure that they match their intentions?