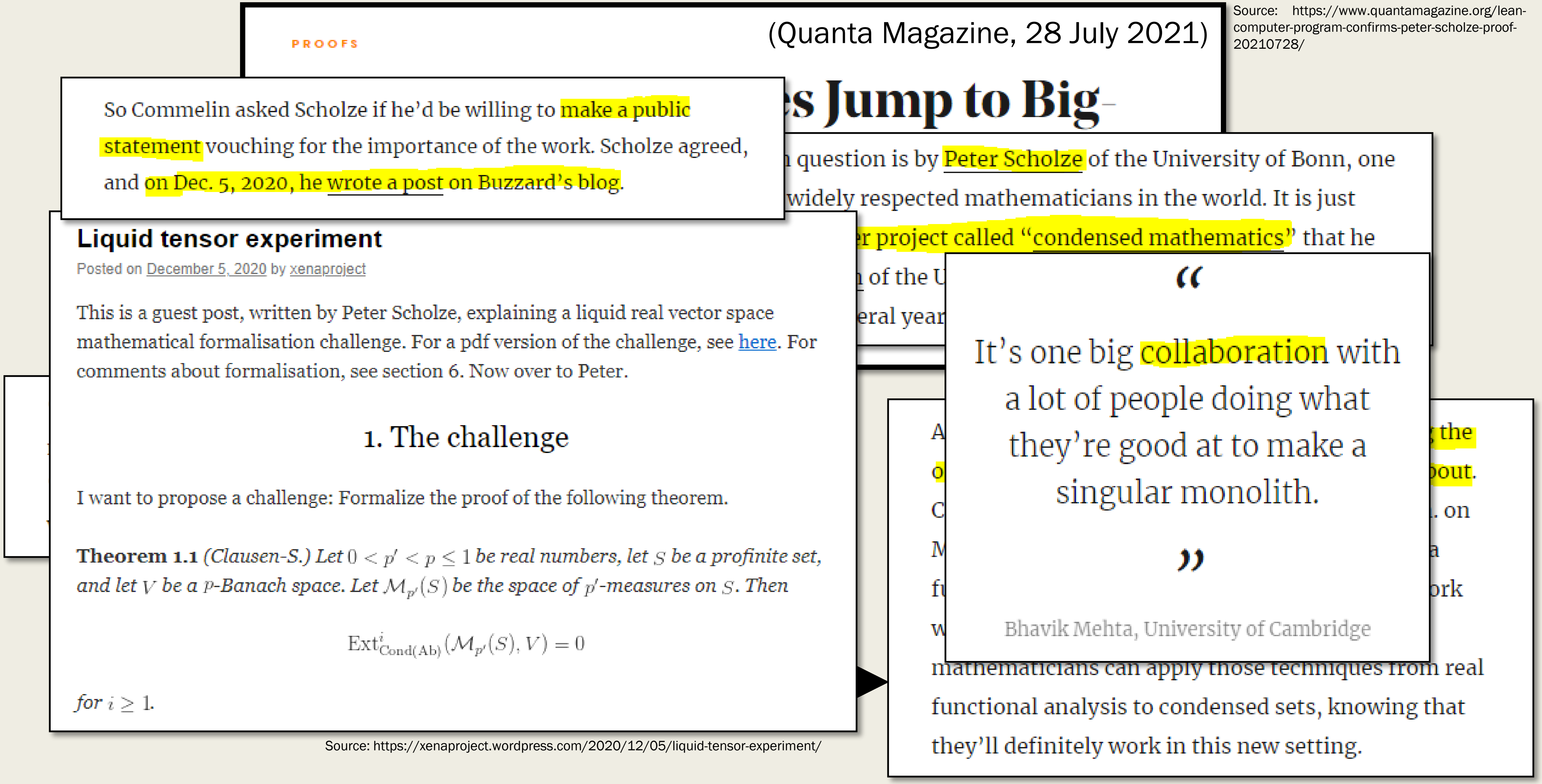
(Auto)Complete this Proof: Decentralized Proof Generation via Smart Contracts

Jin Xing Lim[1], Barnabé Monnot[2], Georgios Piliouras[1], and Shaowei Lin

[1] Singapore University of Technology and Design (SUTD) [2] Ethereum Foundation

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Recent Update in Formalized Mathematics



PROOFS

(Quanta Magazine, 28 July 2021)

Source: https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728/

...s Jump to Big-

So Commelin asked Scholze if he'd be willing to make a public statement vouching for the importance of the work. Scholze agreed, and on Dec. 5, 2020, he wrote a post on Buzzard's blog.

...question is by Peter Scholze of the University of Bonn, one
...widely respected mathematicians in the world. It is just
...r project called "condensed mathematics" that he
...of the U
...eral year

Liquid tensor experiment

Posted on December 5, 2020 by xenaproject

This is a guest post, written by Peter Scholze, explaining a liquid real vector space mathematical formalisation challenge. For a pdf version of the challenge, see here. For comments about formalisation, see section 6. Now over to Peter.

1. The challenge

I want to propose a challenge: Formalize the proof of the following theorem.

Theorem 1.1 *(Clausen-S.) Let $0 < p' < p \leq 1$ be real numbers, let S be a profinite set, and let V be a p-Banach space. Let $\mathcal{M}_{p'}(S)$ be the space of p'-measures on S. Then*

$$\mathrm{Ext}^i_{\mathrm{Cond(Ab)}}(\mathcal{M}_{p'}(S), V) = 0$$

for $i \geq 1$.

Source: https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/

> "It's one big collaboration with a lot of people doing what they're good at to make a singular monolith."
>
> Bhavik Mehta, University of Cambridge

A... the
o... bout.
C... . on
M... a
fu... ork
w...
mathematicians can apply those techniques from real functional analysis to condensed sets, knowing that they'll definitely work in this new setting.

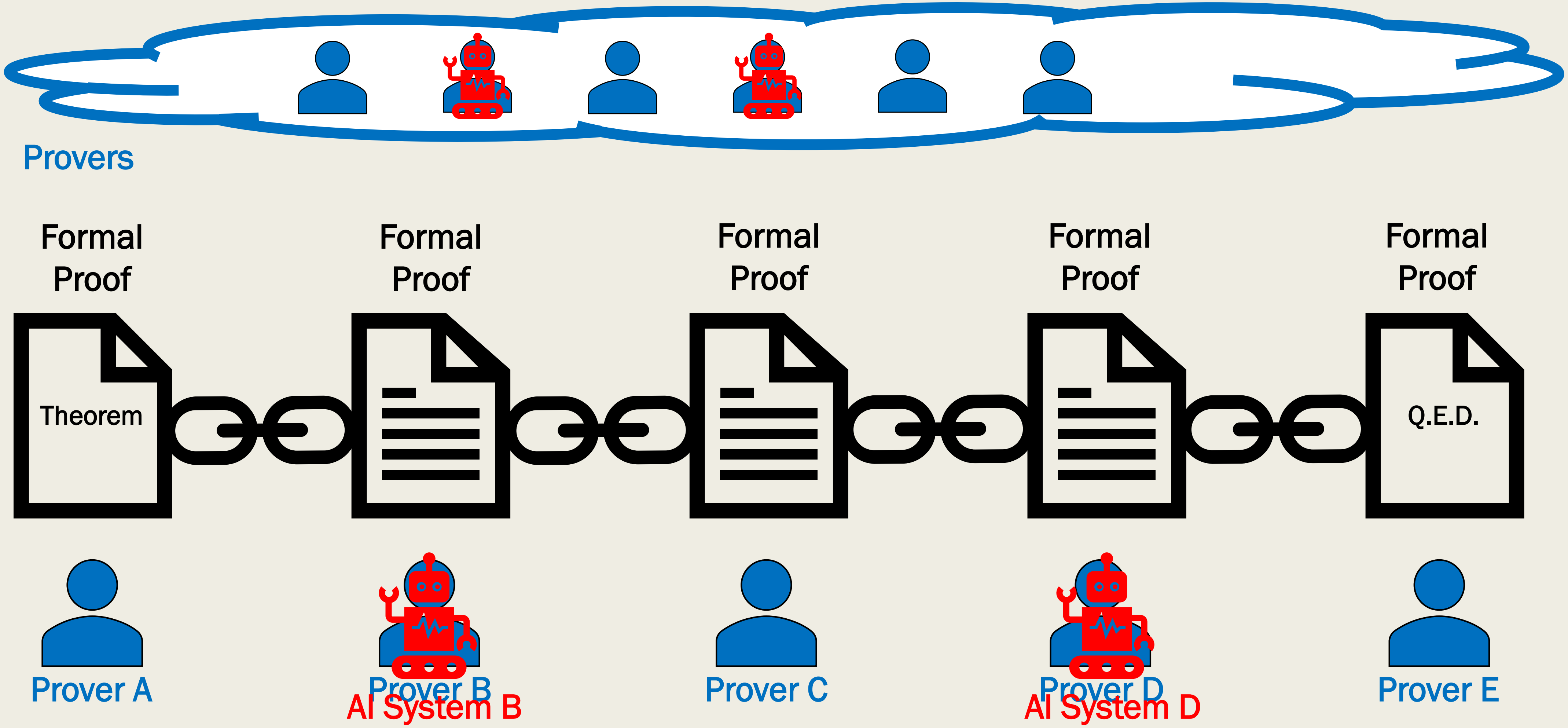
What did we learn?

- Importance of having a **top-down approach** where someone can state his problem statement and it is then broken down into smaller parts for contributors to prove
  - *What is a good way for someone to post his/her problem statement formally and allow contributors to work on it while having the end goal in mind?*

- Importance of **dissemination of partial results** and problems
  - *What is a good platform where contributors can post partial results, state the problems encountered during the proofs and get updates (almost) immediately?*

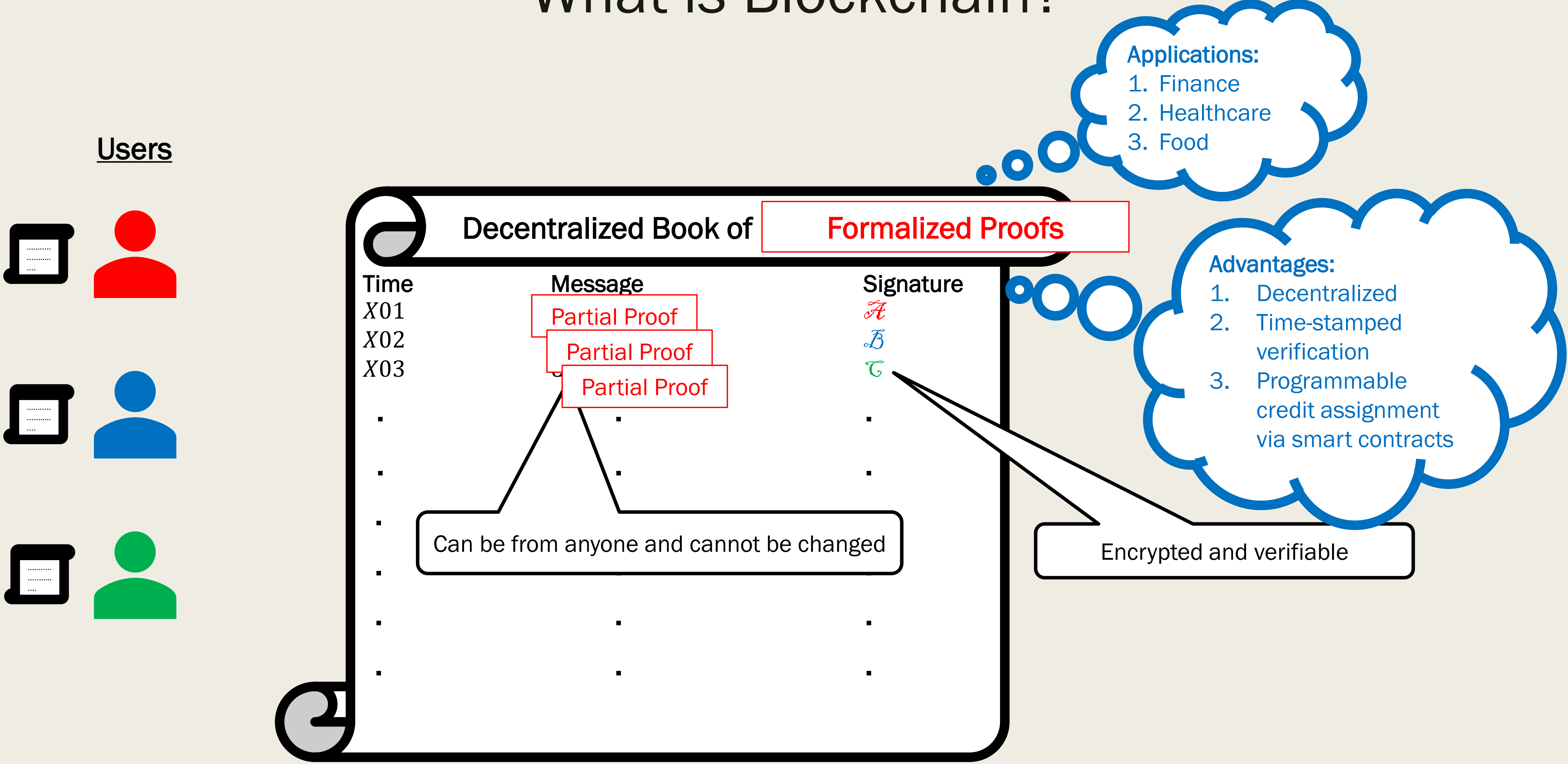
- Importance of **collaboration** between mathematicians/computer scientists
  - *How can we assign verified authorship to each of the partial results?*
  - *How can we incentivize and allocate rewards (if any) fairly to contributors?*

Proposed Solution:
Blockchain Your Own (Partial) Proof



AI System = Sledgehammer (IsabIle), TacticToe (HOL4), CoqHammer (Coq), Tactician (Coq), etc

What is Blockchain?



Users

Decentralized Book of **Formalized Proofs**

Time	Message	Signature
$X01$	Partial Proof	$\mathcal{A}$
$X02$	Partial Proof	$\mathcal{B}$
$X03$	Partial Proof	$\mathcal{C}$
.	.	.
.	.	.
.		
.		
.	.	.
.	.	.

Can be from anyone and cannot be changed

Encrypted and verifiable

Applications:

1. Finance
2. Healthcare
3. Food

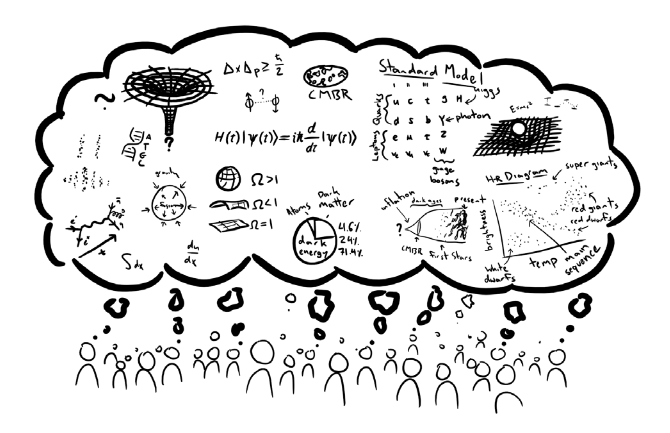
Advantages:

1. Decentralized
2. Time-stamped verification
3. Programmable credit assignment via smart contracts

Images' sources:
https://medium.com/@deepbreadth/the-future-of-common-knowledge-abaca3c04e4f
https://www.enago.com/academy/difference-contributor-and-co-author/
https://www.bravon.io/incentives-vs-gamification/

Why Blockchain?

Decentralization



Fast dissemination of (partial) results and problems
⇒ emergence of common knowledge

Time-stamped verification

Authorships of partial progresses from distributed collaborators can be verified

Credit assignments

Gamification to incentivise provers via smart contracts (e.g. Ethereum)

New token (e.g. ForMath Token) can be used to measure contributions to formalized mathematics

Related Works

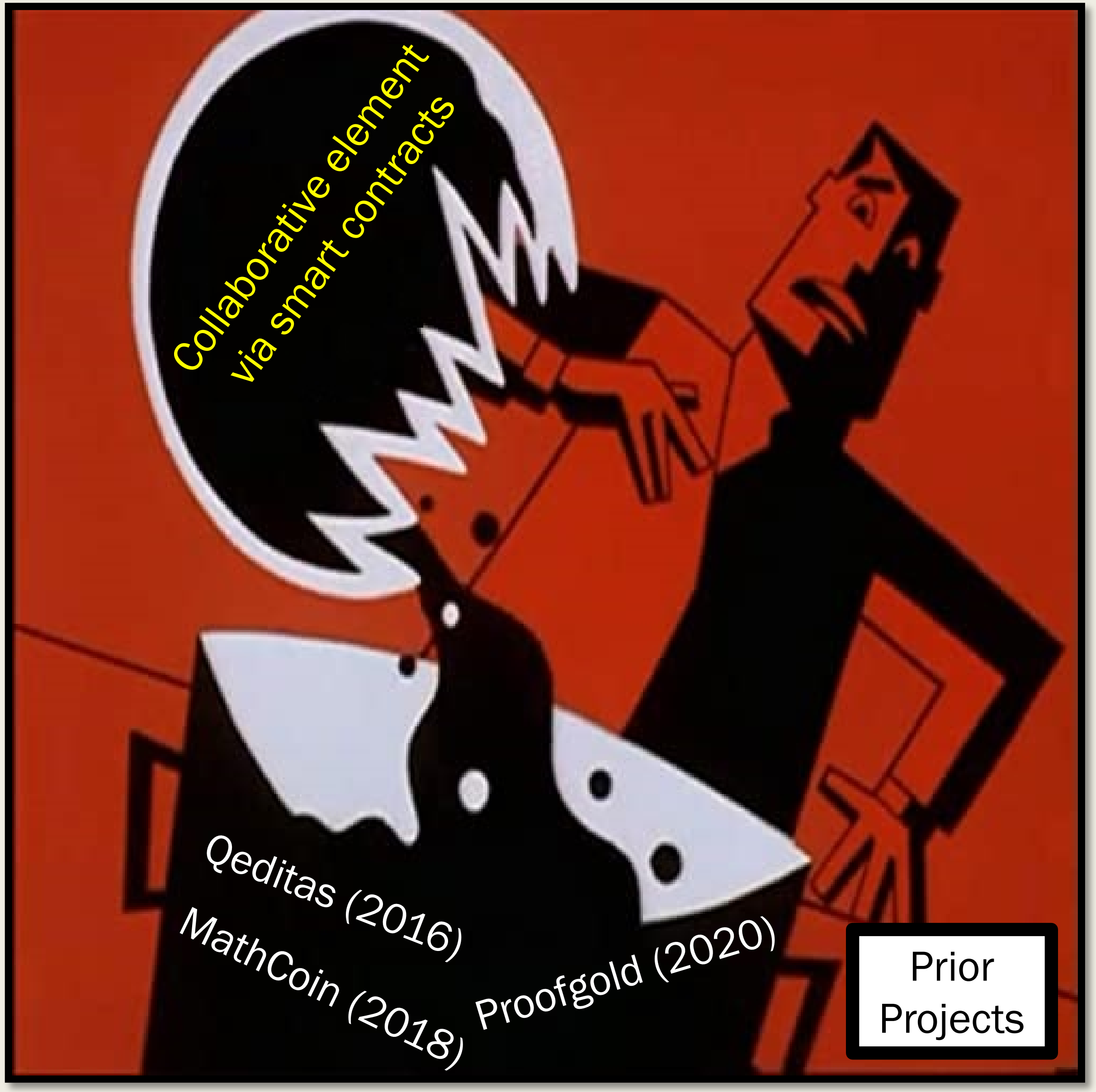


Image source: https://www.amazon.co.jp/-/en/Ci/dp/B07N2K26Y2

System Architecture

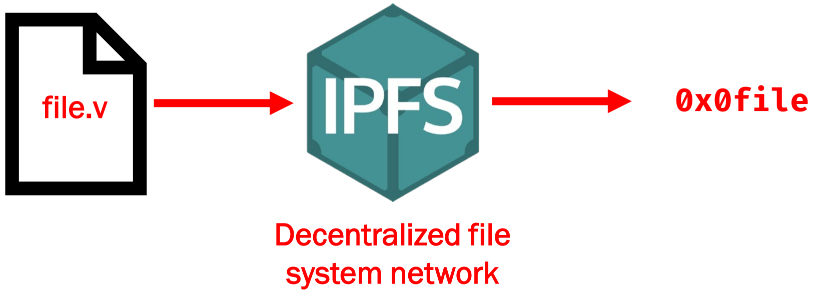
Incentive Layer
(What is rewarded?)

- Deploy incentive mechanisms via smart contracts
- Reward 1 sole prover via voting for main contributor VS split reward via some allocation rule
- Different approaches to score contributions, e.g.:
  - ➢ Token-Curated Registries (TCRs): incentivize participants to vote and rank importance of contributions

Client Layer
(What is meaningful?)

- Access and download records and contributions
- Interface to present the string diagram/directed acyclic graph built by imports declared
- Plug-in to the chosen proof assistant editor may be built
- Perform validity checks that cannot be handled by the data layer

Data Layer
(What is recorded?)

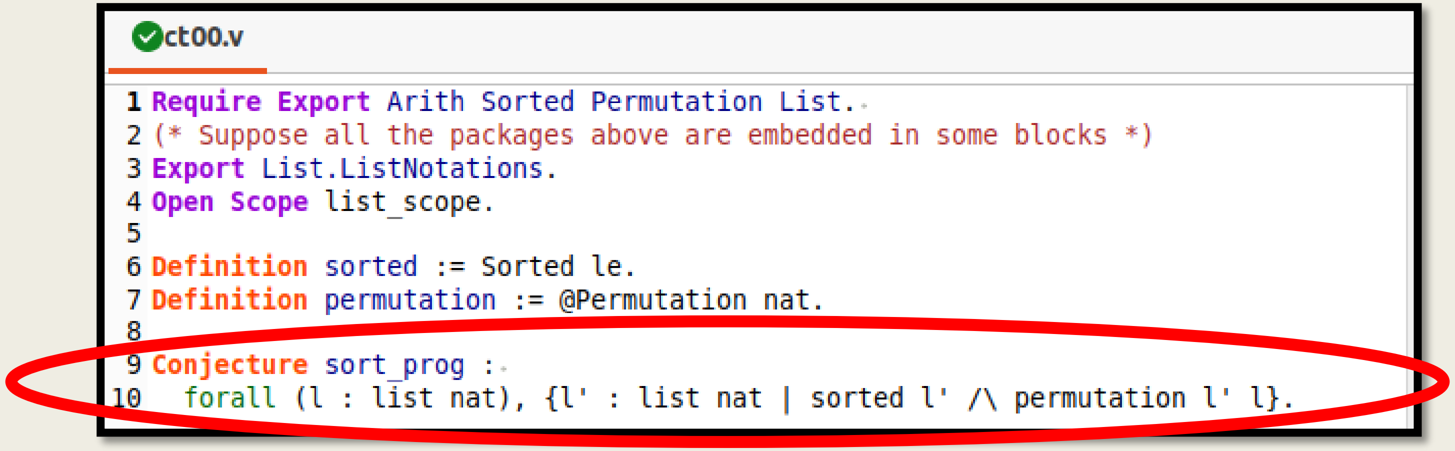


A **record** (on blockchain) contains:

1. Prover’s address
2. IPFS hash address of contribution
3. “imports” references
4. Contribution type (conjecture, partial proof, …)

Image source: https://en.wikipedia.org/wiki/InterPlanetary_File_System

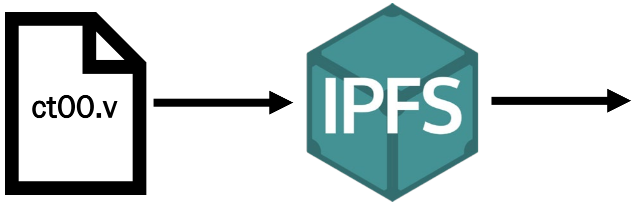
An Illustrative Example: Sort Program in Coq



```
Require Export Arith Sorted Permutation List.
(* Suppose all the packages above are embedded in some blocks *)
Export List.ListNotations.
Open Scope list_scope.

Definition sorted := Sorted le.
Definition permutation := @Permutation nat.

Conjecture sort_prog :
  forall (l : list nat), {l' : list nat | sorted l' /\ permutation l' l}.
```

ct00: Open problem asked by some “client”

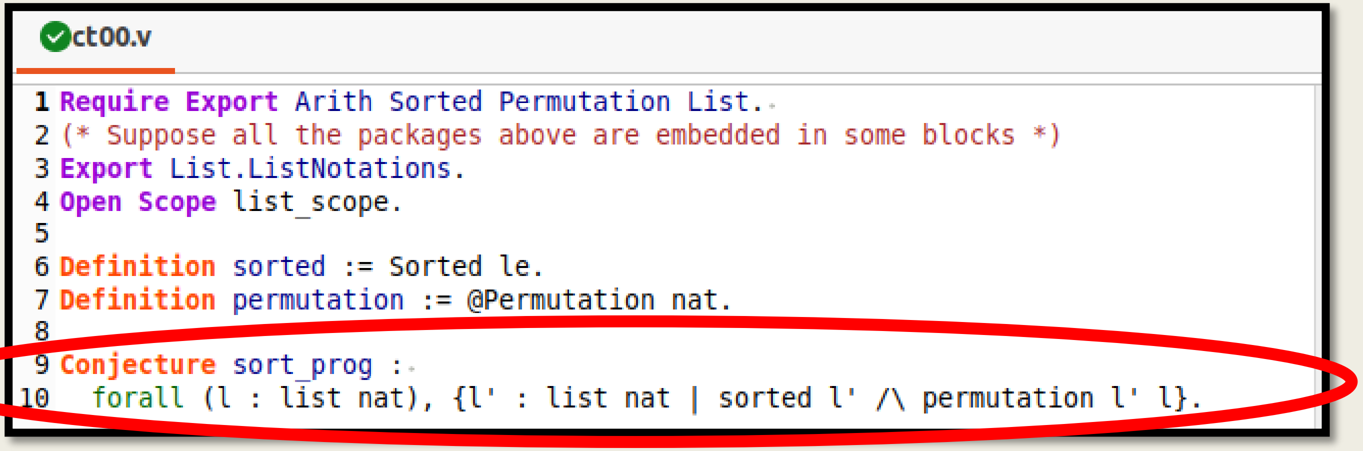
Data Layer
(What is recorded?)



0xEf6c15b611
3ca6D24422B6
C8bc18e702e9
08A572
(Hash address
of ct00.v in IPFS)

1. **Prover’s address:**
   0x3CD087B6F3f639847C94E36d75F52bd587FD78d1
2. **Contribution’s address:**
   0xEf6c15b6113ca6D24422B6C8bc18e702e908A572
3. **“imports” references:**
   hash addresses of Arith, Sorted, Permutation, List
4. **Contribution type:** Conjecture

An Illustrative Example: Sort Program in Coq

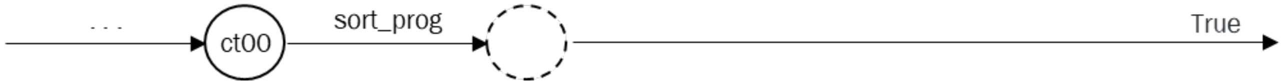


```
Require Export Arith Sorted Permutation List.
(* Suppose all the packages above are embedded in some blocks *)
Export List.ListNotations.
Open Scope list_scope.

Definition sorted := Sorted le.
Definition permutation := @Permutation nat.

Conjecture sort_prog :
  forall (l : list nat), {l' : list nat | sorted l' /\ permutation l' l}.
```

ct00: Open problem asked by some “client”

Client Layer
(What is meaningful?)

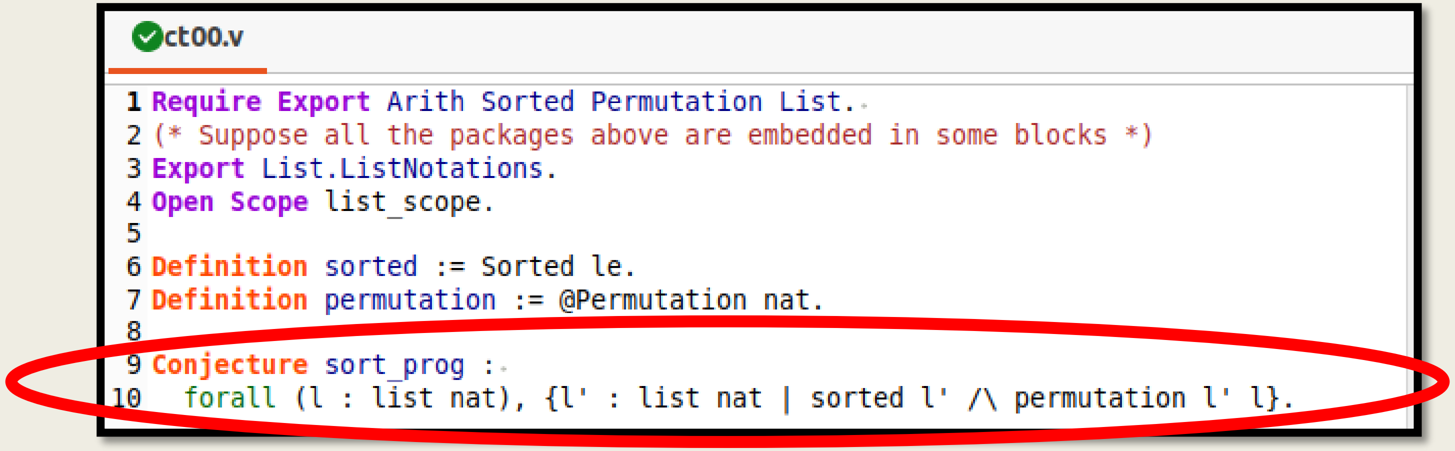
String diagram representation



Legend:

- Edge: Type
- Node (solid): Completed proof term
- Node (dotted): Incomplete proof term

An Illustrative Example: Sort Program in Coq



```
Require Export Arith Sorted Permutation List.
(* Suppose all the packages above are embedded in some blocks *)
Export List.ListNotations.
Open Scope list_scope.

Definition sorted := Sorted le.
Definition permutation := @Permutation nat.

Conjecture sort_prog :
  forall (l : list nat), {l' : list nat | sorted l' /\ permutation l' l}.
```

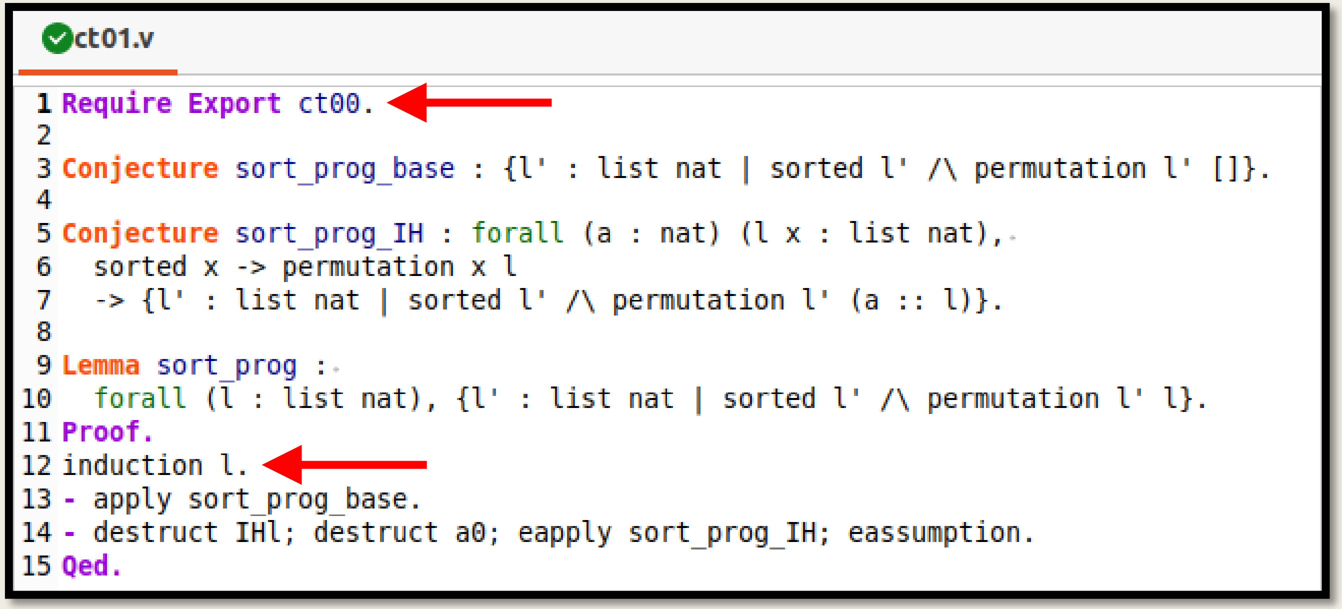
ct00: Open problem asked by some “client”

Incentive Layer
(What is rewarded?)

Smart contract

```
$if\ verify(sort\_prog) = True$:
    $if\ n(\{Provers\}) = 1$:
        $transfer(client, Prover, 10\ tokens)$;
    $else$:
        $allocation\_rule(Provers, 10\ tokens)$;
```

Contribution from Human



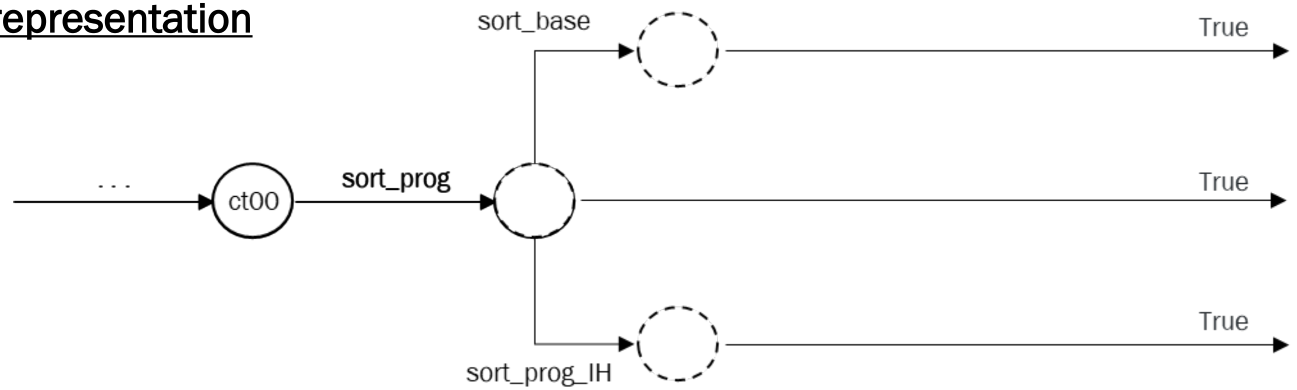
```
Require Export ct00.

Conjecture sort_prog_base : {l' : list nat | sorted l' /\ permutation l' []}.

Conjecture sort_prog_IH : forall (a : nat) (l x : list nat),
  sorted x -> permutation x l
  -> {l' : list nat | sorted l' /\ permutation l' (a :: l)}.

Lemma sort_prog :
  forall (l : list nat), {l' : list nat | sorted l' /\ permutation l' l}.
Proof.
induction l.
- apply sort_prog_base.
- destruct IHl; destruct a0; eapply sort_prog_IH; eassumption.
Qed.
```

Partial proof
(proof with gaps)

ct01: First partial proof by some prover A

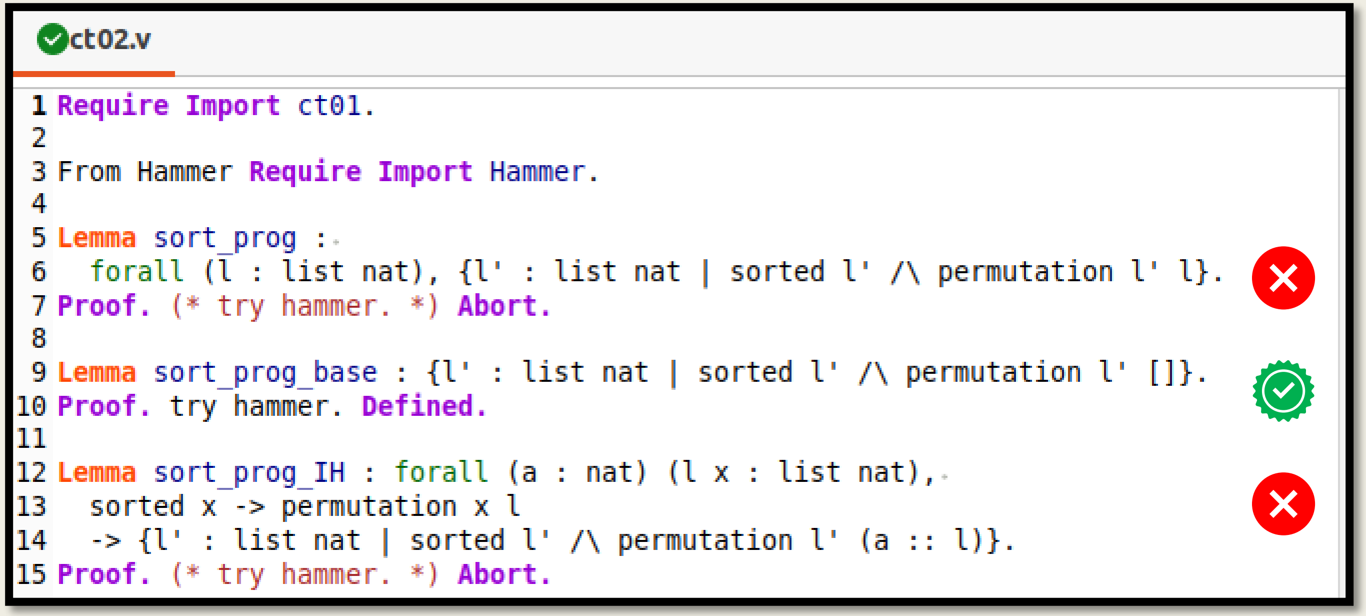
Client Layer
(What is meaningful?)

String diagram representation



Legend:
- Edge: Type
- Node (solid): Completed proof term
- Node (dotted): Incomplete proof term

Contribution from AI System

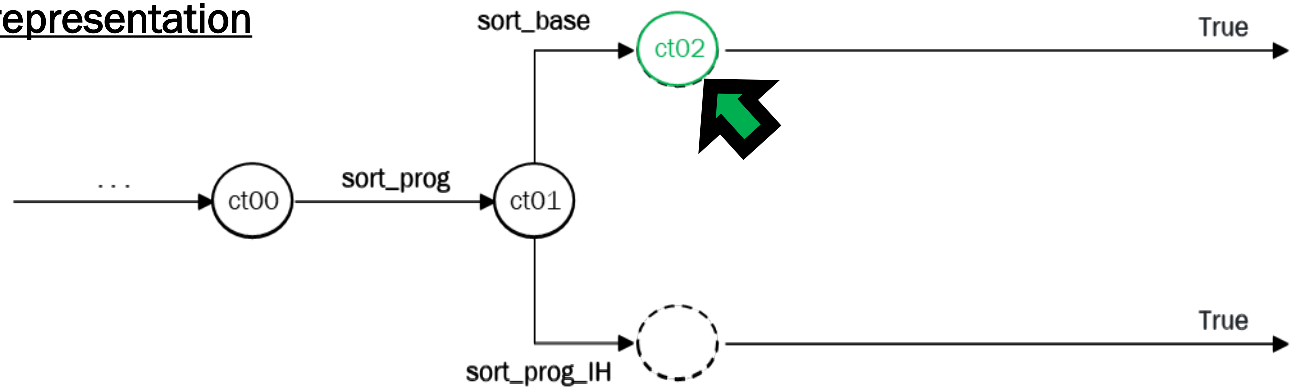


```
Require Import ct01.

From Hammer Require Import Hammer.

Lemma sort_prog :
  forall (l : list nat), {l' : list nat | sorted l' /\ permutation l' l}.
Proof. (* try hammer. *) Abort.

Lemma sort_prog_base : {l' : list nat | sorted l' /\ permutation l' []}.
Proof. try hammer. Defined.

Lemma sort_prog_IH : forall (a : nat) (l x : list nat),
  sorted x -> permutation x l
  -> {l' : list nat | sorted l' /\ permutation l' (a :: l)}.
Proof. (* try hammer. *) Abort.
```

ct02: Contribution by AI CoqHammer

Client Layer (What is meaningful?)

String diagram representation

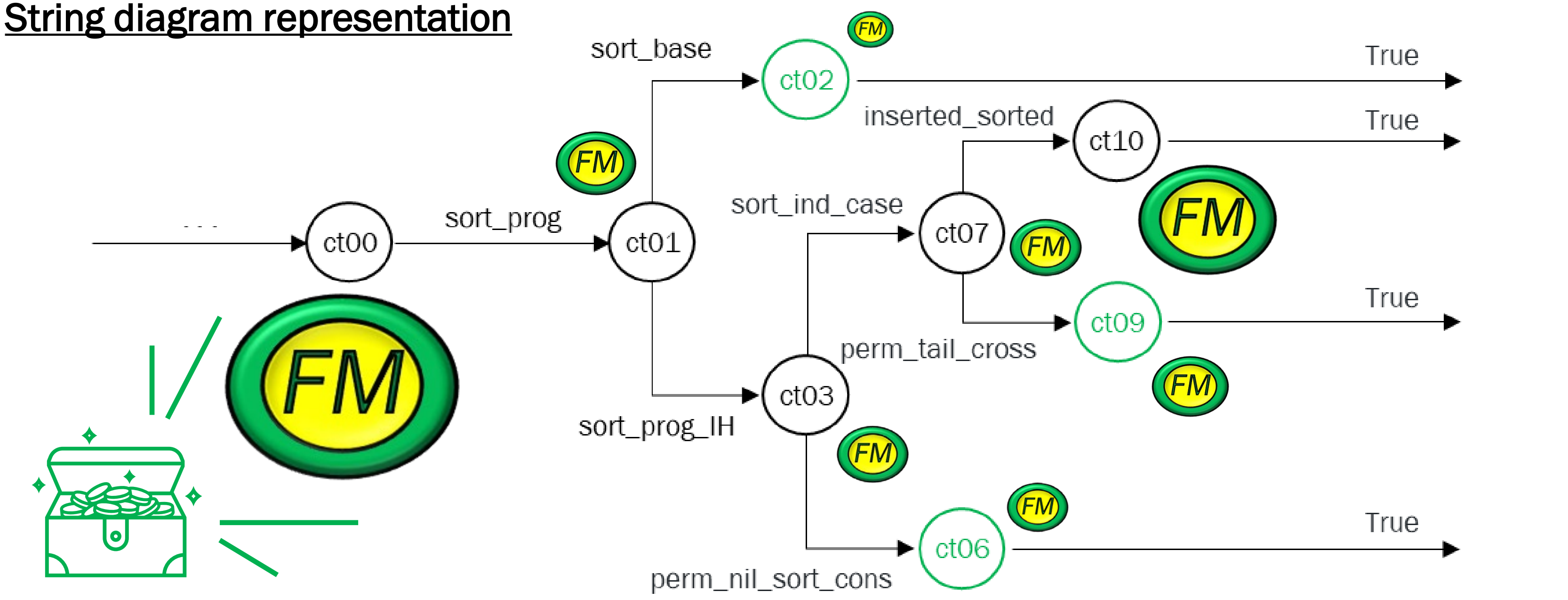


Legend:
- Edge: Type
- Node (solid): Completed proof term
- Node (dotted): Incomplete proof term
- Node (green): AI System

Insertion Sort from Human-AI Collaboration

Client Layer
(What is meaningful?)

String diagram representation



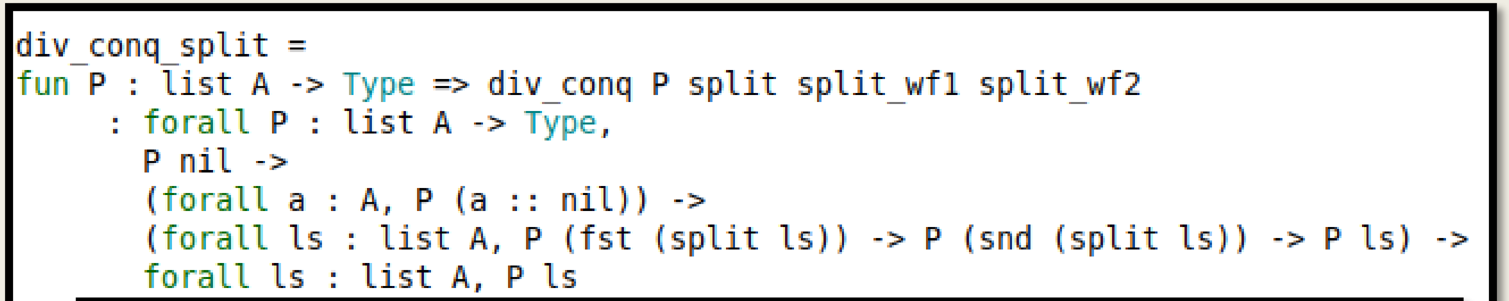
Legend:
- Edge: Type
- Node (solid): Completed proof term
- Node (dotted): Incomplete proof term
- Node (green): AI System

Incentive Layer
(What is rewarded?)

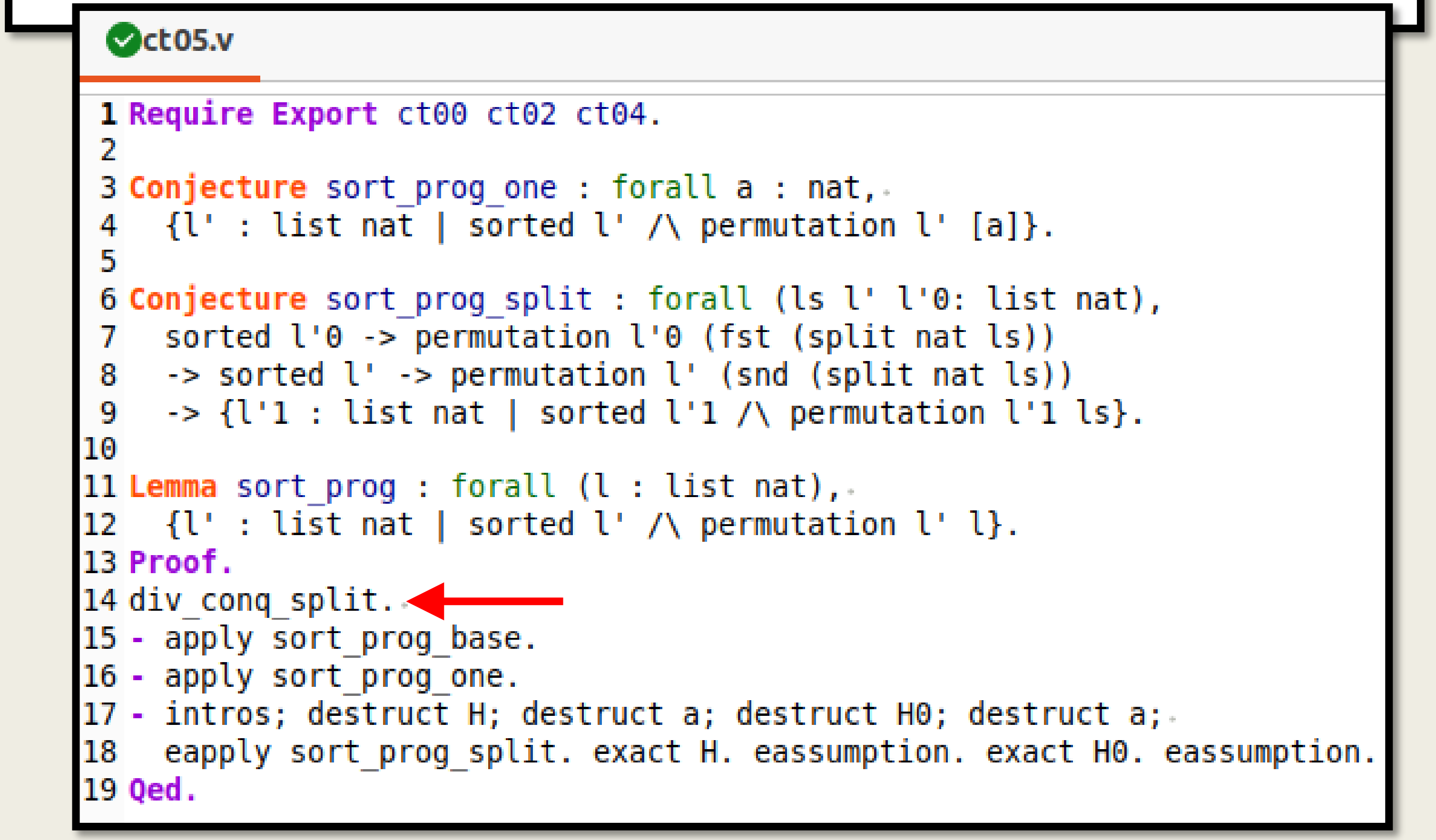
Smart contract

```
$if\ verify(sort\_prog) = True:$
    $if\ n(\{Provers\}) = 1:$
        $transfer(client, Prover, 10\ tokens);$
    $else:$
        $allocation\_rule(Provers, 10\ tokens);$
```

Same Theorem but Different Proof

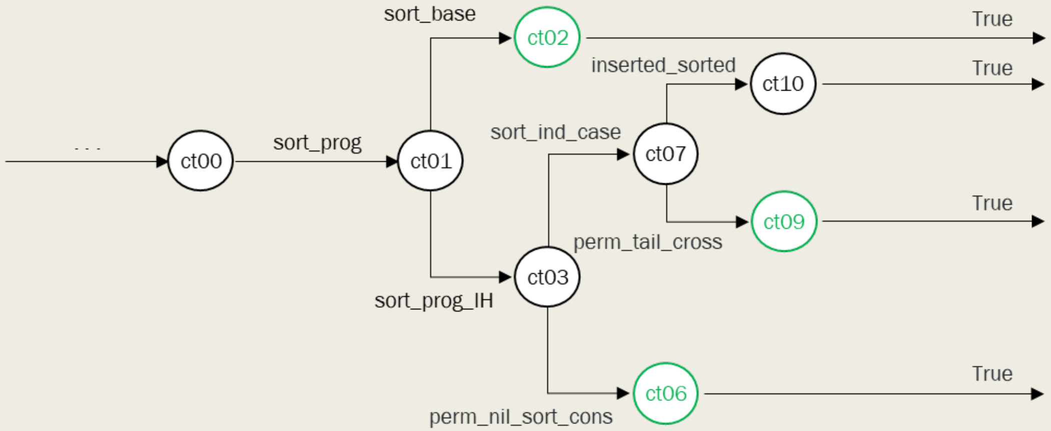


```
div_conq_split =
fun P : list A -> Type => div_conq P split split_wf1 split_wf2
     : forall P : list A -> Type,
       P nil ->
       (forall a : A, P (a :: nil)) ->
       (forall ls : list A, P (fst (split ls)) -> P (snd (split ls)) -> P ls) ->
       forall ls : list A, P ls
```

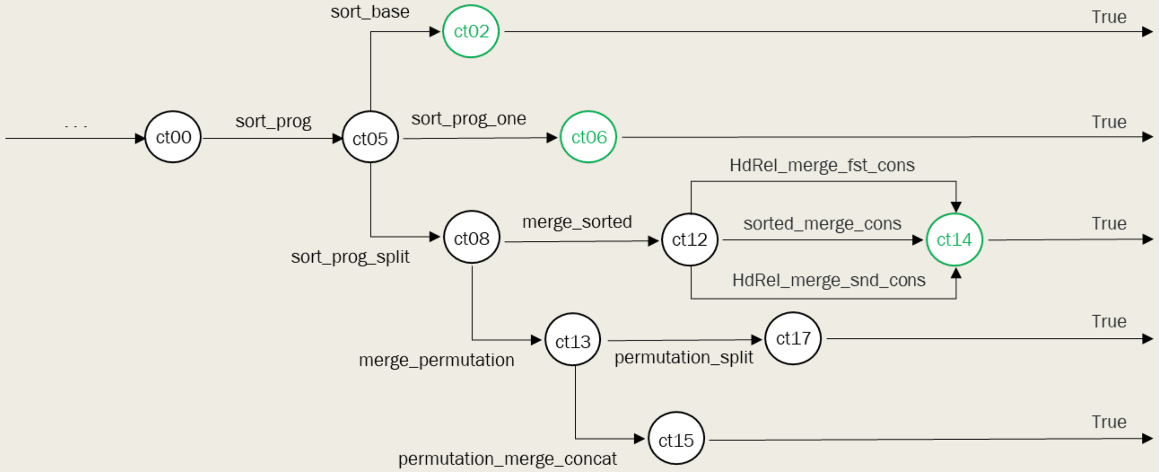


```
Require Export ct00 ct02 ct04.

Conjecture sort_prog_one : forall a : nat,
  {l' : list nat | sorted l' /\ permutation l' [a]}.

Conjecture sort_prog_split : forall (ls l' l'0: list nat),
  sorted l'0 -> permutation l'0 (fst (split nat ls))
  -> sorted l' -> permutation l' (snd (split nat ls))
  -> {l'1 : list nat | sorted l'1 /\ permutation l'1 ls}.

Lemma sort_prog : forall (l : list nat),
  {l' : list nat | sorted l' /\ permutation l' l}.
Proof.
div_conq_split.
- apply sort_prog_base.
- apply sort_prog_one.
- intros; destruct H; destruct a; destruct H0; destruct a;
  eapply sort_prog_split. exact H. eassumption. exact H0. eassumption.
Qed.
```

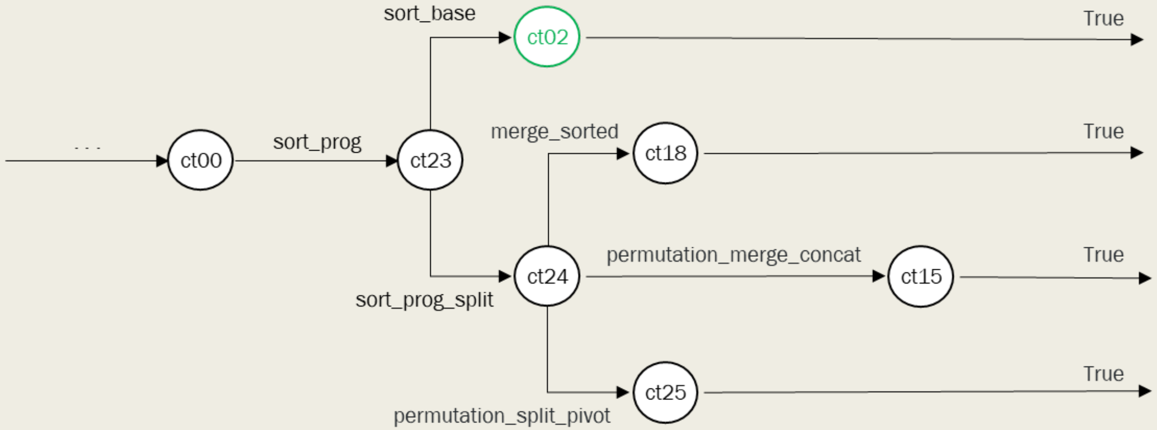
Decentralized Way of
Building Different Proofs Collaboratively



Insertion Sort



Merge Sort

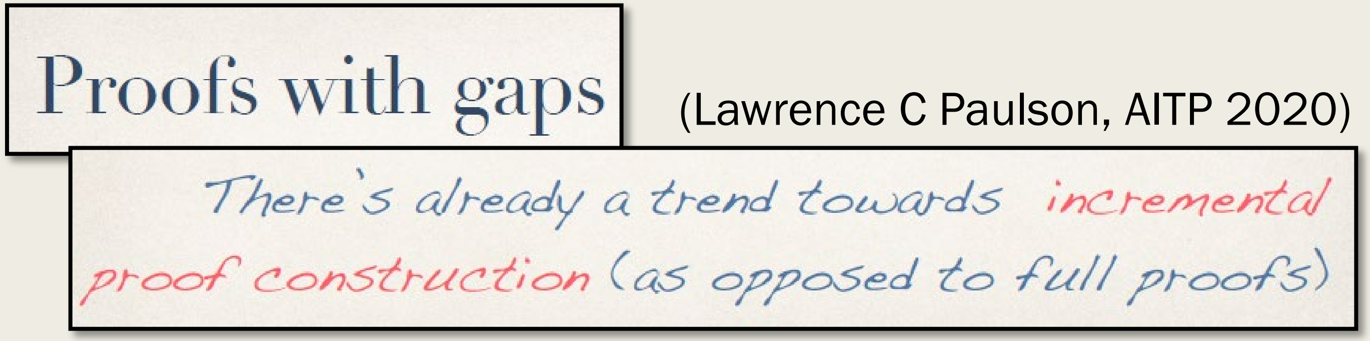


Quick Sort

All codes can be found on https://github.com/jinxinglim/coq-chain.

Conclusion

Challenge: To have humans and AI systems to collaborate in formalizing mathematics

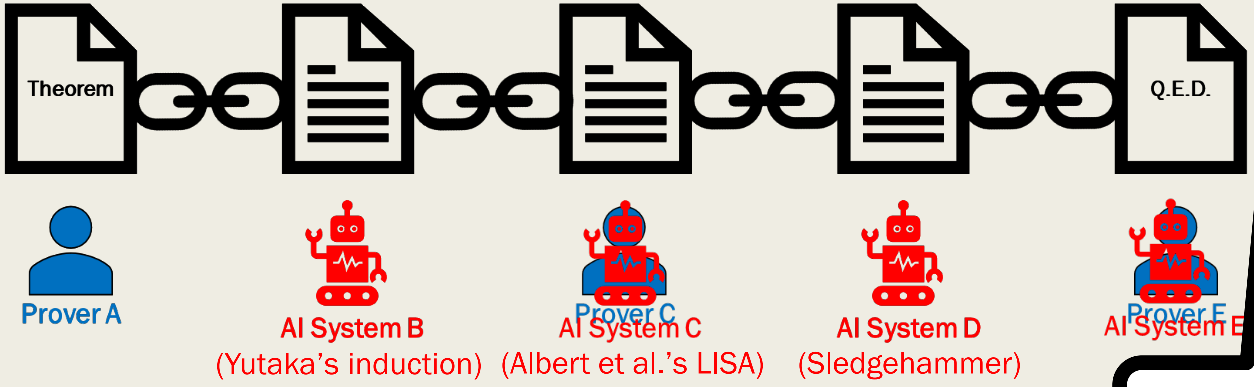


Proofs with gaps

(Lawrence C Paulson, AITP 2020)

There's already a trend towards incremental proof construction (as opposed to full proofs)

Solution: Use blockchain as the platform to unite collaborators (humans/AI systems) together



Unified proof/program synthesis

Future Directions:

- Ways to allocate rewards (if any) fairly to contributors (within same proo
- Ways to incentivise creations of new mathematical objects (definitions/tactics/propositions)

Image source:
https://juliandontcheff.wordpress.com/2017/12/06/artificial-stupidity-as-a-dba-limitation-of-artificial-intelligence/

Thank you!

Questions and Feedback