runtime verification

Universal Truth Framework

Grigore Rosu

Founder & CEO, RV

Professor of Computer Science, UIUC

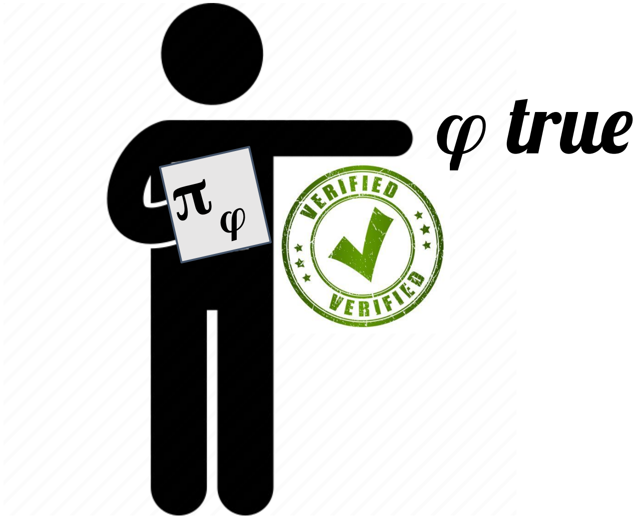
research.runtimeverification.com

March 2023

Universal Truth Framework – What?

Every claim made by framework is verifiably true!

- Claims come with independent, succinct, 3rd party checkable proof certificates



Claim = anything provable or computable:
program execution, work done or action,
formal correctness or security of code,
… mathematical theorem

Universal Truth Framework – So What? Many Many Applications … Sky's the Limit

Verifiable computing for *all* programming languages

- Execute your code securely in untrusted environments (e.g., in the cloud)

zkLANG for *any* programming language LANG, correct by construction

- zkEVM variants, Cairo (StarkWare), zkVM (RiscZero), zkLLVM (=nil; Foundation)

Formal verification, correctness, security audits, any other program claims, all become checkable certificates (instead of PDFs)

- You don't have to trust the developers or the auditors or anybody else

Critical procedures or devices (medical, aviation, automotive, robotics, blockchains) yield checkable certificates for their correct application

- Increase confidence in complex systems, complex processes, machines, AI

Universal Truth Framework – How?
K + SNARKS = Marriage Made in Heaven

Claim:

φ → $\mathbb{K}$ →

Can use other provers (Coq, Lean, Isabelle, Agda, Dedukti, etc.) or even AI (ChatGPT) to search for proofs

Mathematical long proof: Π_φ

```
#1. φ1                 // axiom
#2. φ2                 // axiom
...
#100. φ100             // axiom
...
#247. φ2 → φ248        // ...
#248. φ248   // modus ponens
...          // with #2, #247
#99999. φ              //
...
```

Huge!
GBs or TBs

→

mathematical proof checker (240 LOC)

SNARK-ed!

→

crypto proof: π_φ

256 bits

Universal Truth Framework – New Blockchain Tech? Blockchain of Truth!

Blockchains currently suffer from some limitations:

- Duplication of computation (all nodes execute same code)
- Hardwired programming or VM language, for all programs
- Security, correctness, formal verification are “external” activities, off-chain

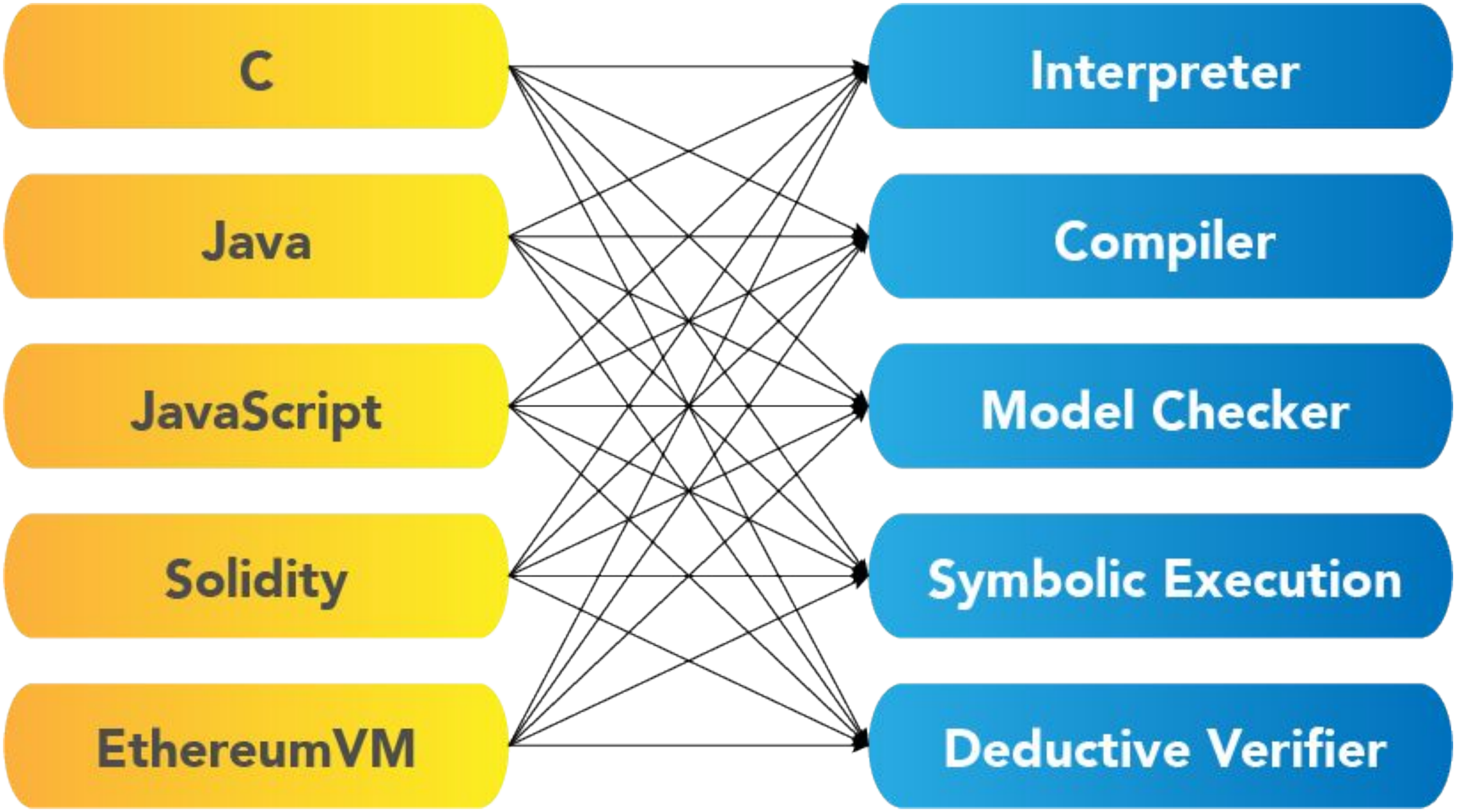
Will enable new generation of blockchains - Blockchain of Truth

- Allow arbitrary claims to be made, stored, checked; e.g. executions, correctness
- Write smart contracts in any programming or specification language
- Execute transaction code once and for all, locally; send SNARK certificate
- Any claim is backed by a mathematical proof, made succinct as a crypto proof

What is K and Why?

kframework.org

State of the Art:
(too) Many Languages, Many Tools

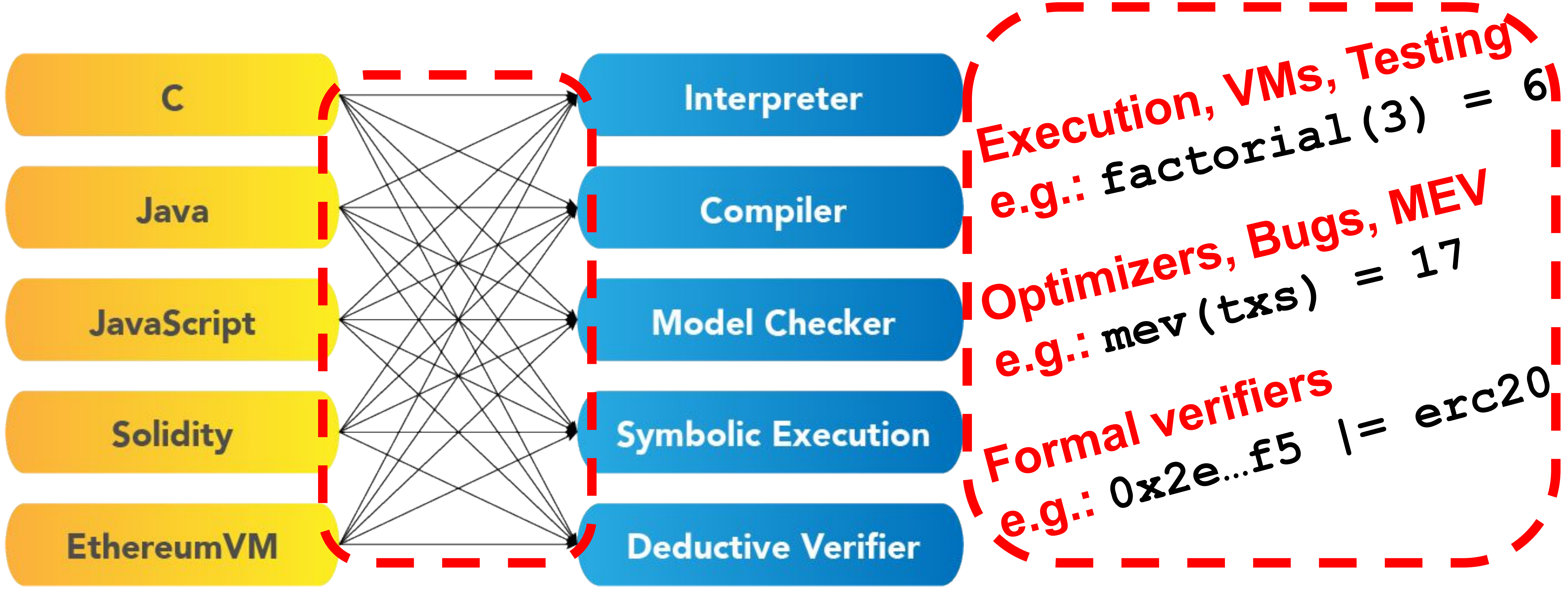


Execution, VMs, Testing
e.g.: `factorial(3) = 6`

Optimizers, Bugs, MEV
e.g.: `MEV(txs) = 17`

Formal verifiers
e.g.: `0x2e…f5 |= ERC20`

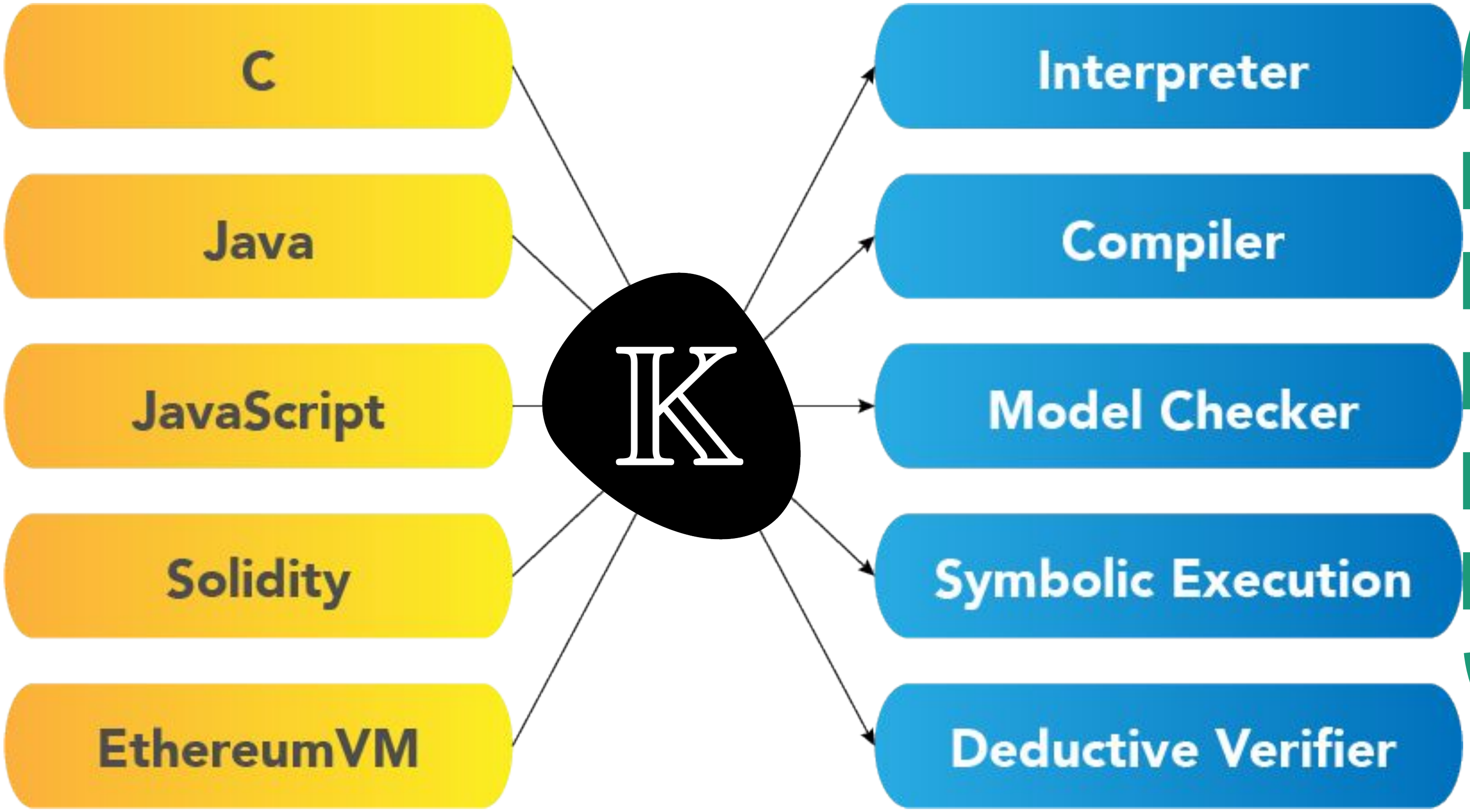
Pain Points:
Duplication, Errors, and Many <u>Claims</u> to Trust!

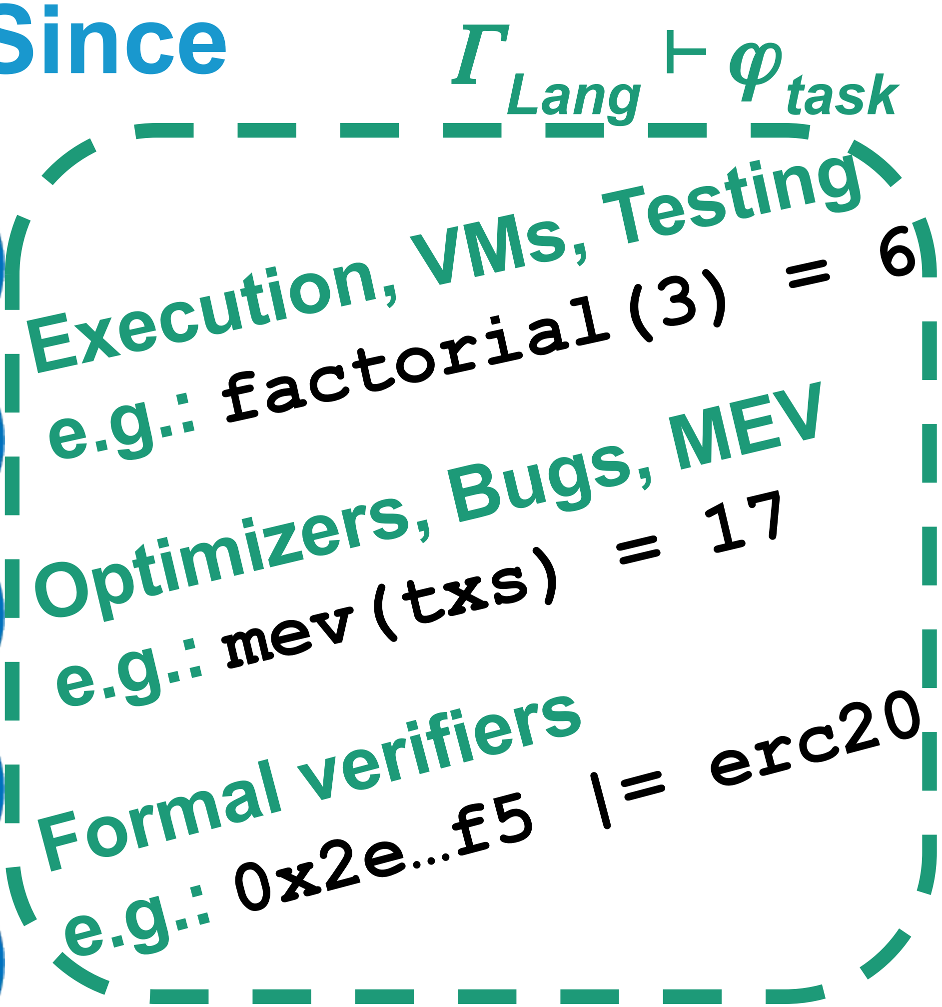


- Duplication of code and effort
- Wasted talent, error prone, out of sync

Claims: Functional, Safety, Security)
Blockchain tech falls here ^^^

Our Solution: K
Invented in 2003, Improved Ever Since





+ Separation of concerns
+ Intrinsic network effect

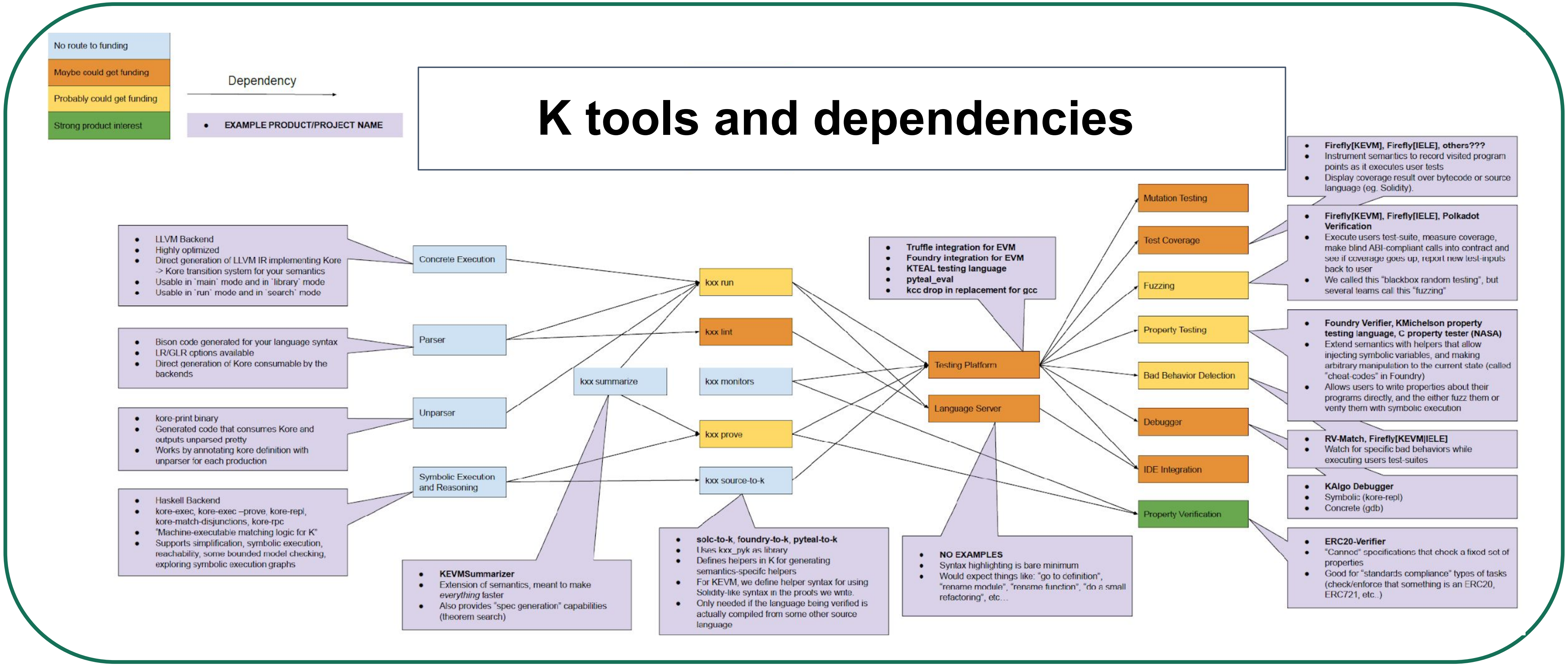
Everything in K is a proof, $\Gamma_{Lang} \vdash \varphi_{task}$
Computation is a special case of proof
Small(est) proof checker: 240 LOC

K is Large and Complex – Why Trust It?
You Shouldn't! Check The Proofs It Outputs!

500k+ lines of code, 4 different languages

Likely most complex formal methods system

Open source:
kframework.org



runtime verification

What's New in K?

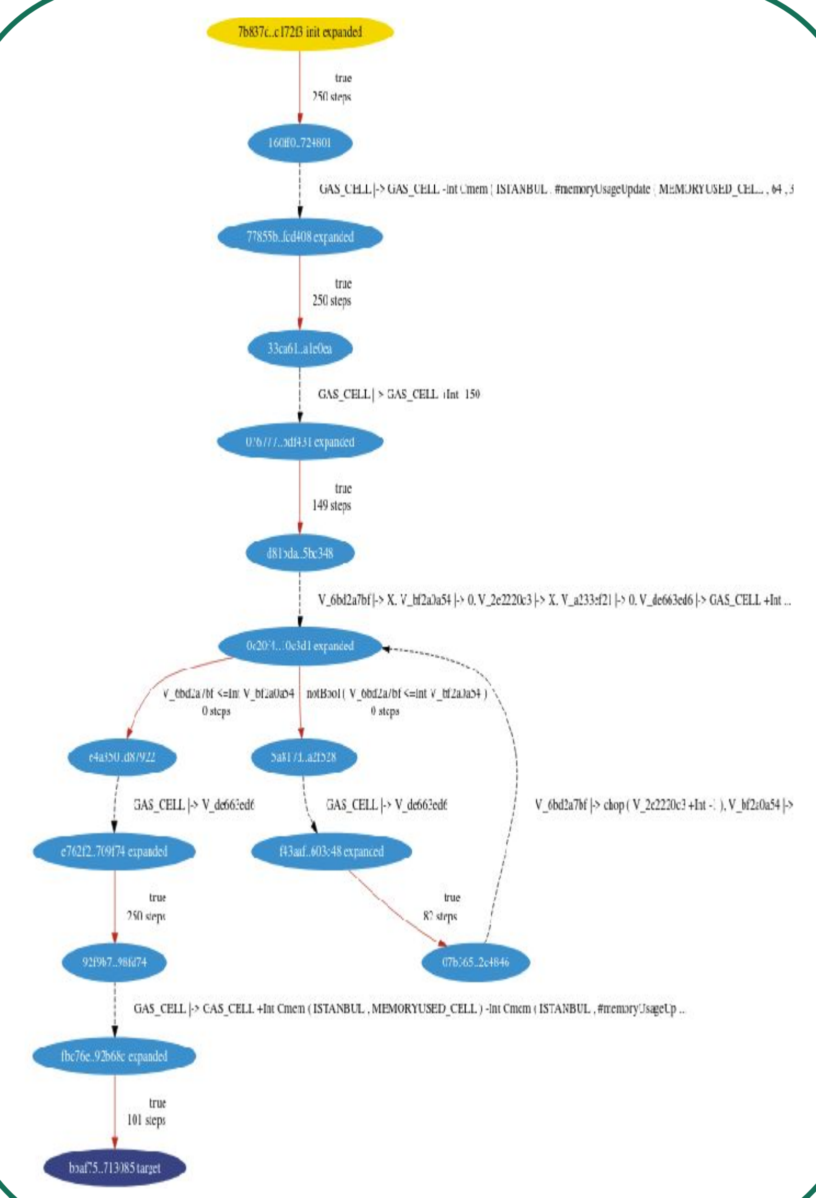
research.runtimeverification.com

K Summarizer

Input: PL semantics, say <u>KEVM</u>, and code fragment, say **`SumContract`**

```
contract SumContract {

    function sum(int n) external pure returns (int s) {
        s = 1;
        int i = n;
        while (i > 1) {
            s += i;
            i -= 1;
        }
    }
}
```



Output: A CFG comprising all symbolic behaviors of the program.
Semantics driven, correct by construction: each edge is a proved claim.

Not possible 6 months ago! Game changer.

KFoundry = K[EVM] + Cheat Codes

Foundry is an increasingly popular parametric property testing framework for Solidity

```
contract ContractTest is Test {

    SumContract cut;

    function setUp() public {
        cut = new SumContract();
    }

    function testSumProperty(int N) public {
        vm.assume(N >= 0); // vm.assume(N < 10);
        int r = cut.sum(N);
        assertEq(r, N * (N + 1) / 2);
    }
}
```

Starting from blank state, execute `setUp`, then save that state. Deploy any contracts needed, mint balances, etc…

From the state post `setUp`, execute each `test*` method with random inputs for the arguments.

Foundry tools generate random inputs for parameters and run the resulting tests.

KFoundry executes the parametric tests symbolically, using the K EVM semantics.
Parametric tests become formal specifications, and KFoundry formally verifies them!

Familiar UI. K hidden under the hood!

Formal Verification with Parametric Tests

Hoare Triples – foundation of formal verification:

```
(forall vars)
    {pre} code {post}
```

They can be expressed as property tests:

```
function testProperty(vars) {
  assume pre;
  code;
  assert post;
}
```

Example (Solidity sum of numbers up to N):

```
(forall N)
    {N > 0} r = sum(N) {r = N*(N+1)/2}
```

expressible as property test:

```
function testSumProperty(int N) public {
    vm.assume(N > 0);
    int r = cut.sum(N);
    assertEq(r, N * (N + 1) / 2);
}
```

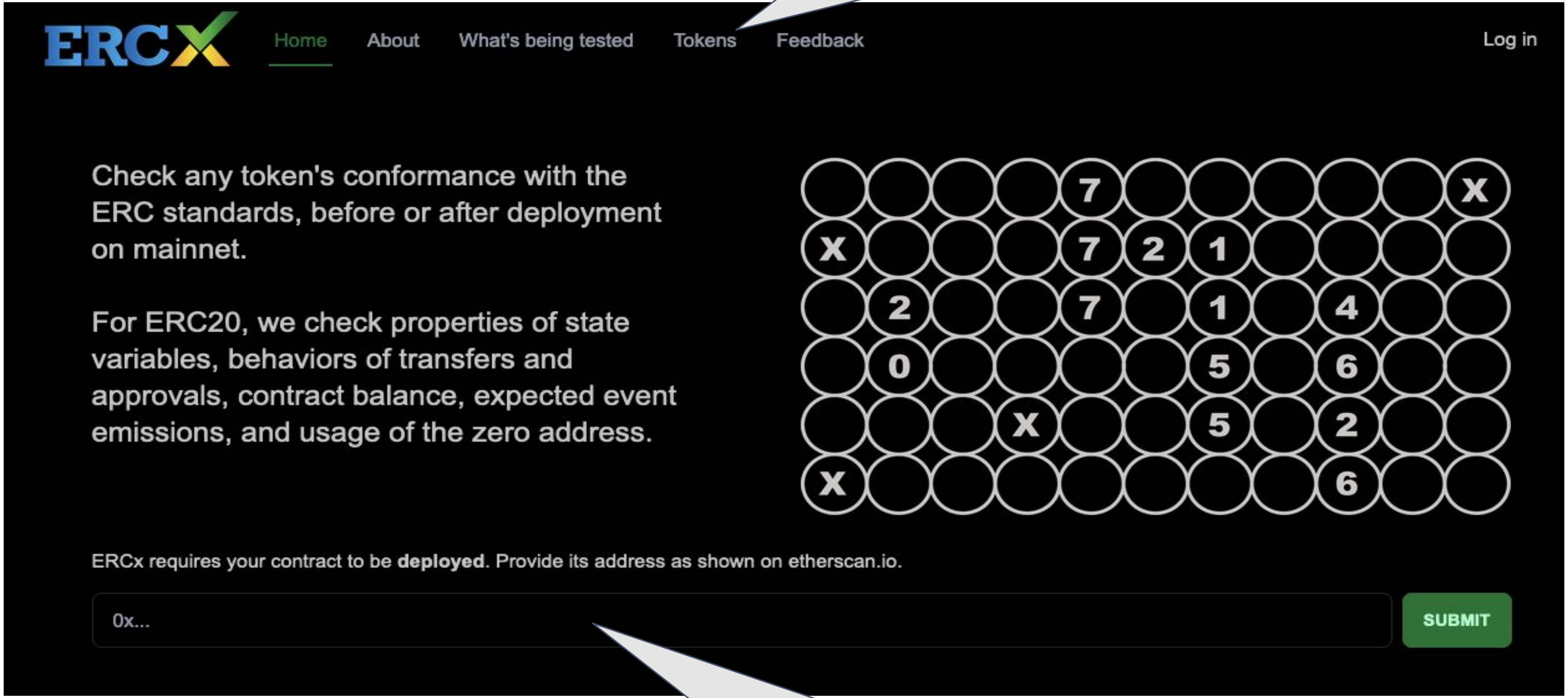
Passing property test symbolically = formal verification of Hoare triple !

Most of formal verification *is* symbolic property testing!

ERC-X Tool = KFoundry[ERC token specification]

ercx.runtimeverification.com

Deep dive investigation of ERC tokens deployed on mainnet.



Completely Automatic!

Submit your token, too. See how you stack up!

RV-Match = K[C]

Used for Verification of Solana Validator

Funded by:

NASA NSF BOEING TOYOTA JUMP CRYPTO

```c
typedef unsigned short ushort;
static inline ushort fd_ushort_rotate_right(ushort x, int n) {
  n &= 15;
  return (ushort)((x >> n) | (x << (16 - n)));
}

int main(int argc, char **argv) {
  ushort i = fd_ushort_rotate_right(60164, 0);
  return 0;
}
```

firedancer snippet

RV-Match:

- Instance of K concrete execution with C lang
- Automatic debugger for subtle bugs other tools can't find, with no false positives
- Seamless integration with unit tests, build infrastructure, and continuous integration
- Platform for analyzing programs, boosting standards compliance and assurance

Conventional compilers do not detect problem

```
$ gcc fd_ub_example.c -o gcc.out
$ ./gcc.out
$
$ kcc fd_ub_example.c -o kcc.out
$ ./kcc.out
fd_ub_example.c: In function `fd_ushort_rotate_right':
fd_ub_example.c:6:3: error: undefined behavior: result of signed left shift not representable in result type [-Wno-signed-left-shift-overflow]
    Refer to c18 §6.5.7/4 file:///src/.build/dist/lib/kcc/html/shifts.html
  called by fd_ub_example.c:10:14(main)
```

RV-Match's kcc tool precisely detects and reports error, and points to ISO C11 standard

Most comprehensive C semantics! ISO compliant.

runtime verification

Matching Logic
Proof Objects
SNARKed Proof Checker

Matching Logic = Foundation for K, Coq, Lean, …

Smallest logical foundation known for languages and formal verification!

- Invented in 2019 [published in LICS'19]
- 7 syntactic constructs, 15 proof rules
- Can define *any programming language (PL)*
- Can express *any claim about any program*

Everything K, Coq, Lean, etc., do is a provable matching logic theorem $\boldsymbol{\Gamma \vdash \varphi}$

- Thus, K, Coq, Lean, …, become powerful *methodologies* to build ML proofs [ICFP'20]
- We prefer K: computation = proof, fast (can be used as PL), many PLs formalized

FOL Rules	(Propositional 1)	$\varphi \to (\psi \to \varphi)$
	(Propositional 2)	$(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
	(Propositional 3)	$((\varphi \to \bot) \to \bot) \to \varphi$
	(Modus Ponens)	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}$
	(∃-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
	(∃-Generalization)	$\dfrac{\varphi \to \psi}{(\exists x.\, \varphi) \to \psi}\ x \notin FV(\psi)$
Frame Rules	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(Propagation$_\vee$)	$C[\varphi \vee \psi] \to C[\varphi] \vee C[\psi]$
	(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ with $x \notin FV(C)$
	(Framing)	$\dfrac{\varphi \to \psi}{C[\varphi] \to C[\psi]}$
Fixpoint Rules	(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
	(Prefixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
	(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{(\mu X.\, \varphi) \to \psi}$
Technical Rules	(Existence)	$\exists x.\, x$
	(Singleton)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

240 LOC Proof Checker – Smallest Ever!

We use MetaMath

- metamath.org; 20+ implementations
- Defined entire matching logic
- Encode claims and proof objects
- Reduce claim correctness to mathematical proof checking: $\Gamma \vdash \varphi$ (axioms Γ, theorem φ)

We re-implemented MetaMath:

- In RiskZero's Rust fragment
- Then generate recursive STARK
- Collaboration with RiskZero (Tim Carstens) and Univ. of Illinois (Andrew Miller + Grigore Rosu)

[published in CAV'21, OOPSLA'23]

```
$c \imp ( ) #Pattern |- $.

$v ph1 ph2 ph3 $.
ph1-is-pattern $f #Pattern ph1 $.
ph2-is-pattern $f #Pattern ph2 $.
ph3-is-pattern $f #Pattern ph3 $.
imp-is-pattern
  $a #Pattern ( \imp ph1 ph2 ) $.

axiom-1
  $a |- ( \imp ph1 ( \imp ph2 ph1 ) ) $.

axiom-2
  $a |- ( \imp ( \imp ph1 ( \imp ph2 ph3 ) )
        ( \imp ( \imp ph1 ph2 )
               ( \imp ph1 ph3 ) ) ) $.

${
  rule-mp.0 $e |- ( \imp ph1 ph2 ) $.
  rule-mp.1 $e |- ph1 $.
  rule-mp   $a |- ph2 $.
$}
```

Matching logic syntax and proof system (240 LOC in total)

```
imp-refl $p |- ( \imp ph1 ph1 )
$=
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  imp-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern imp-is-pattern
  imp-is-pattern ph1-is-pattern
  ph1-is-pattern ph1-is-pattern
  imp-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  imp-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern axiom-2
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  axiom-1 rule-mp ph1-is-pattern
  ph1-is-pattern axiom-1 rule-mp
$.
```

Claims with proofs (machine checked)

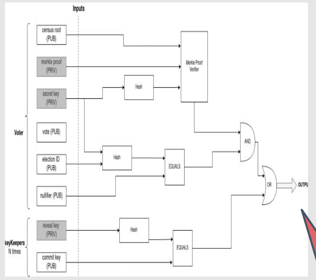
More General, Universal, yet Smaller Circuit (than Language-Specific Solutions)

zkEVM vs zk[K[EVM]]

Only Execution Claims: φ

Trust me, zkEVM is a <u>correct</u> implementation of some (hypothetical) EVM formal semantics!

zkEVM
(complex)



zkEVM is just one example, but Cairo, zkVM of RiskZero, zkLLVM of =nil;, etc., suffer from the same problem

30k to
1.5M+ LOC

crypto proof: π_φ

Any Claim: φ

Plug-and-Play any language instead of EVM.

Cairo
RISC-V
LLVM
. . .
Plutus
. . .

EVM → K

Formal semantics of EVM, ideally on the blockchain and vetted by some organization.

Π_φ

~240 LOC

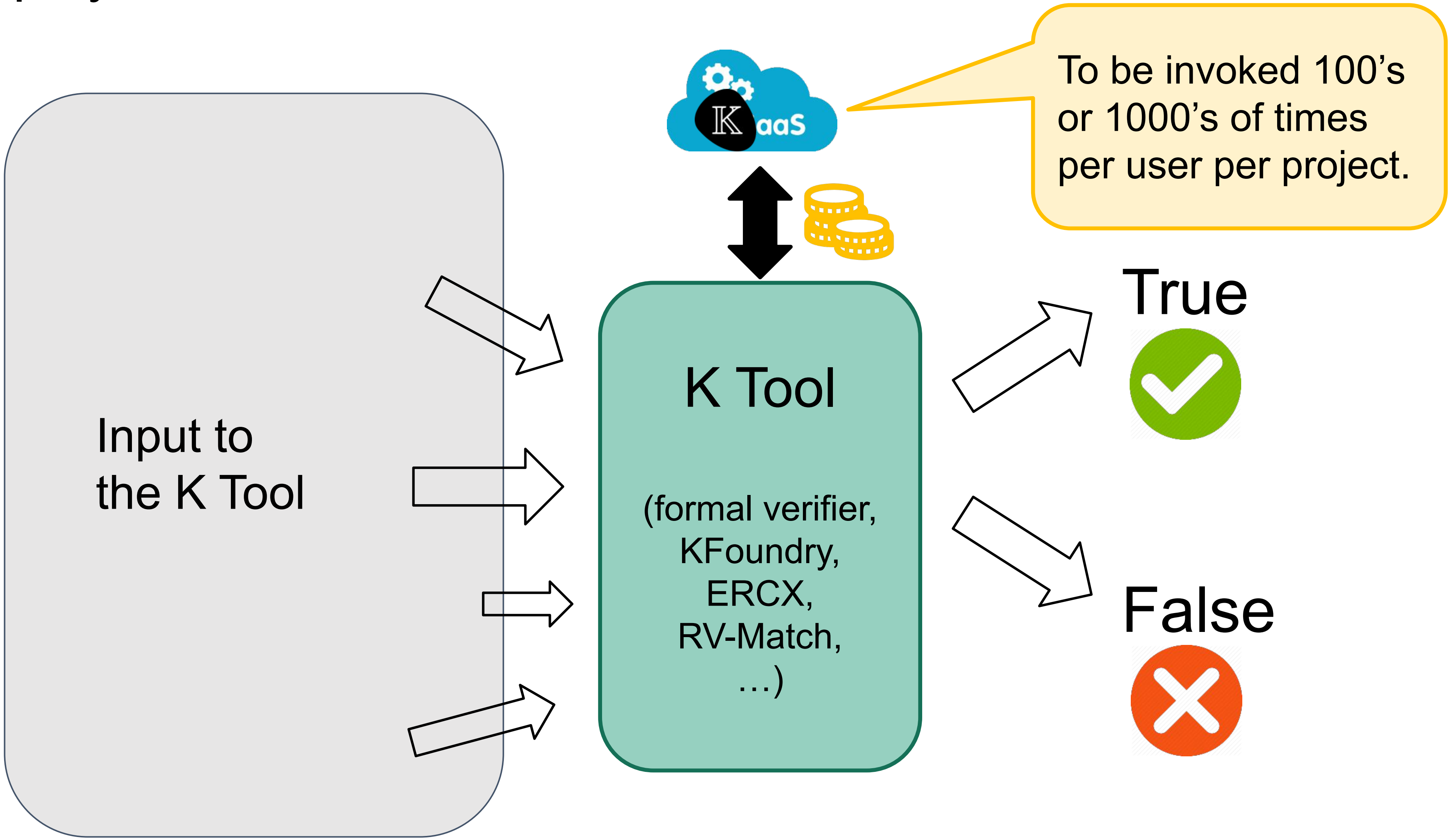
SNARK-ed simple matching logic proof checker. Same for all languages and claims!

crypto proof: π_φ

Coming Products (2023-2024)

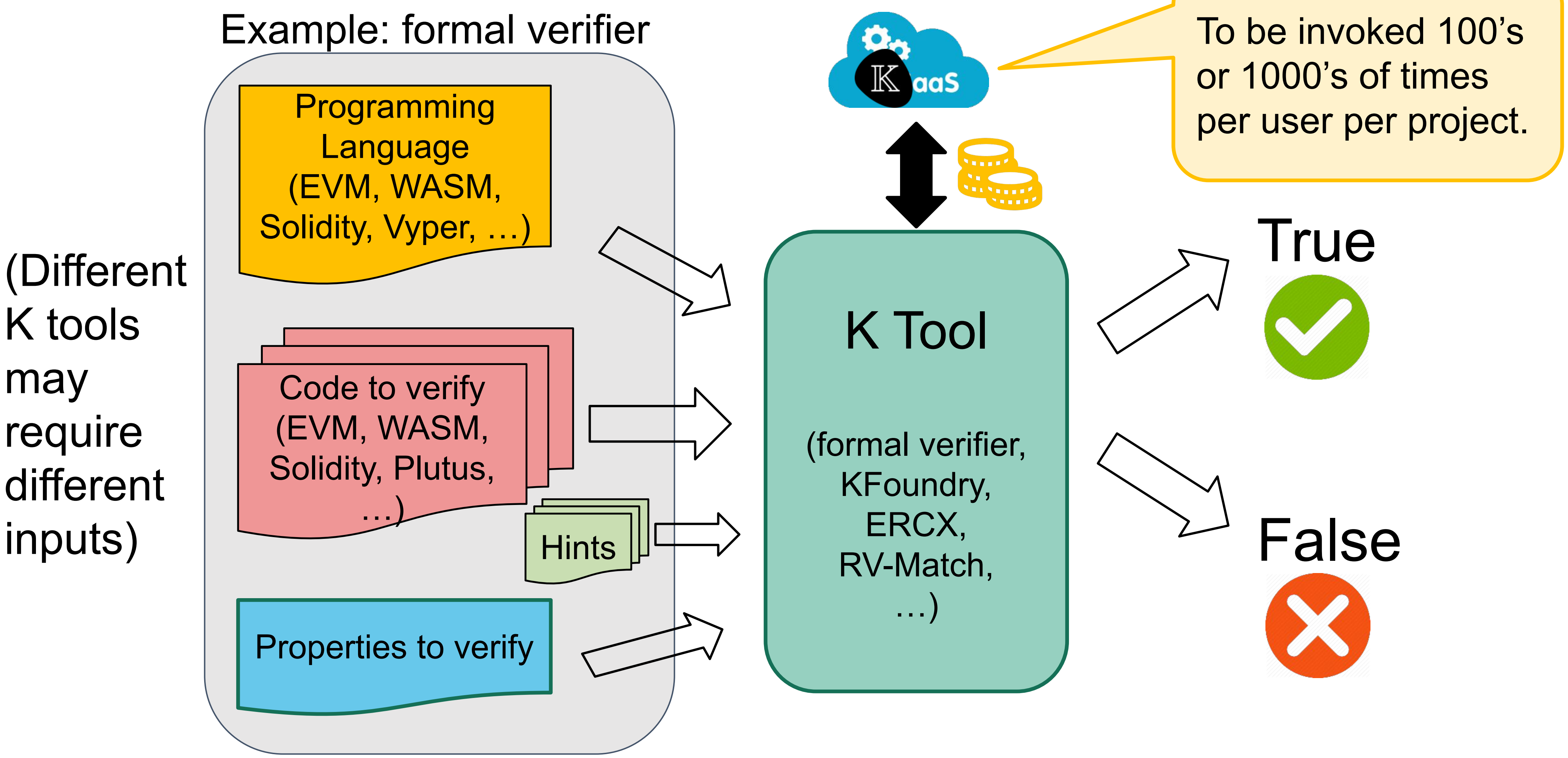
2023 Product: K Prover as a Service (KaaS)

Pre-deployment recurrent revenue



2023 Product: K Prover as a Service (KaaS)

Pre-deployment recurrent revenue



2023 Product: Invariant Monitoring & Recovery

Post-deployment recurrent revenue

Major outcome of formal verification audits: **specifications / invariants**

So we got the most difficult component of runtime monitoring … for free !

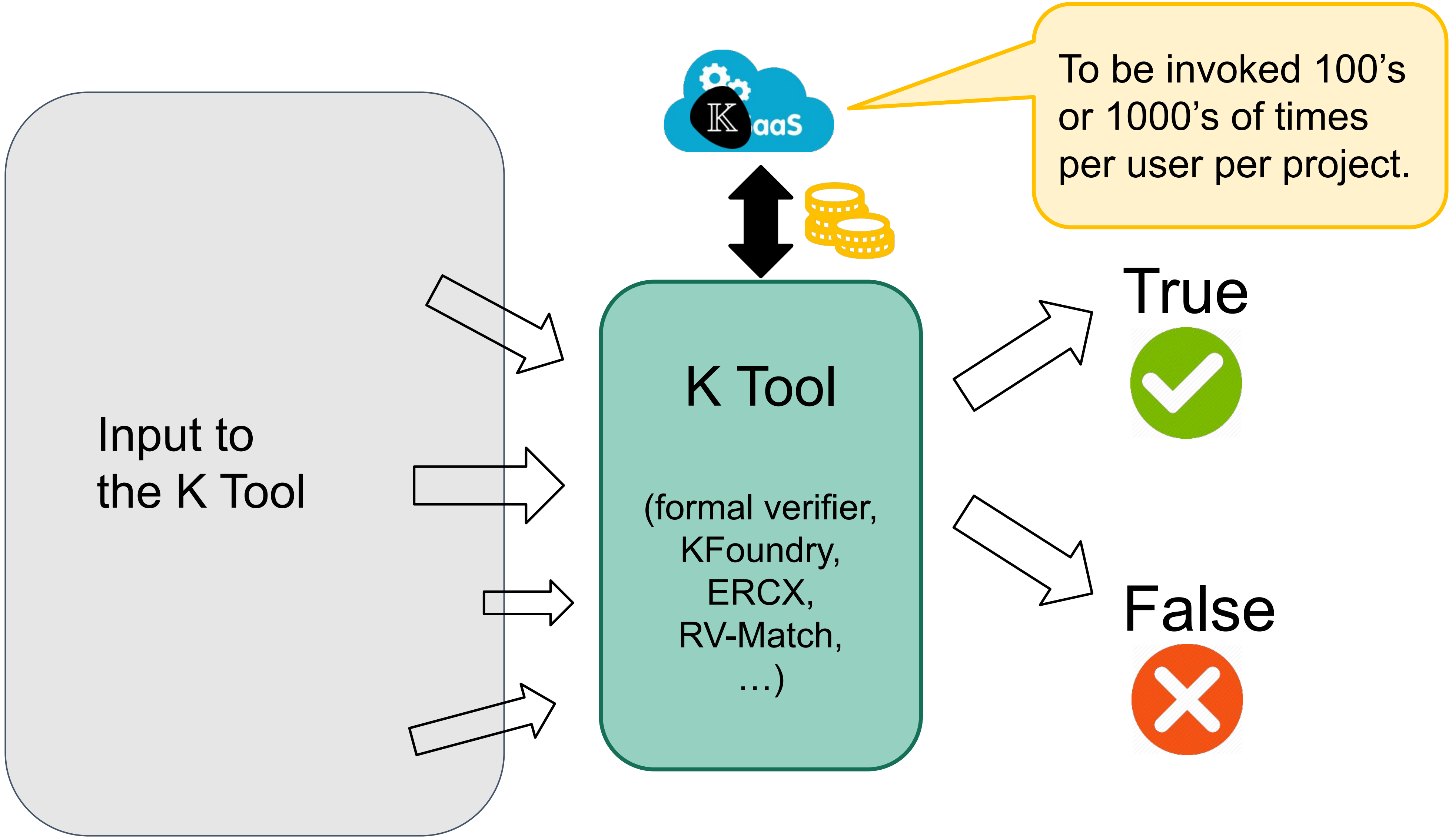
Monitoring services alone, without recovery

- Inform clients of the health status of their protocol
- Inform keepers (arbitrators, liquidators) of opportunities

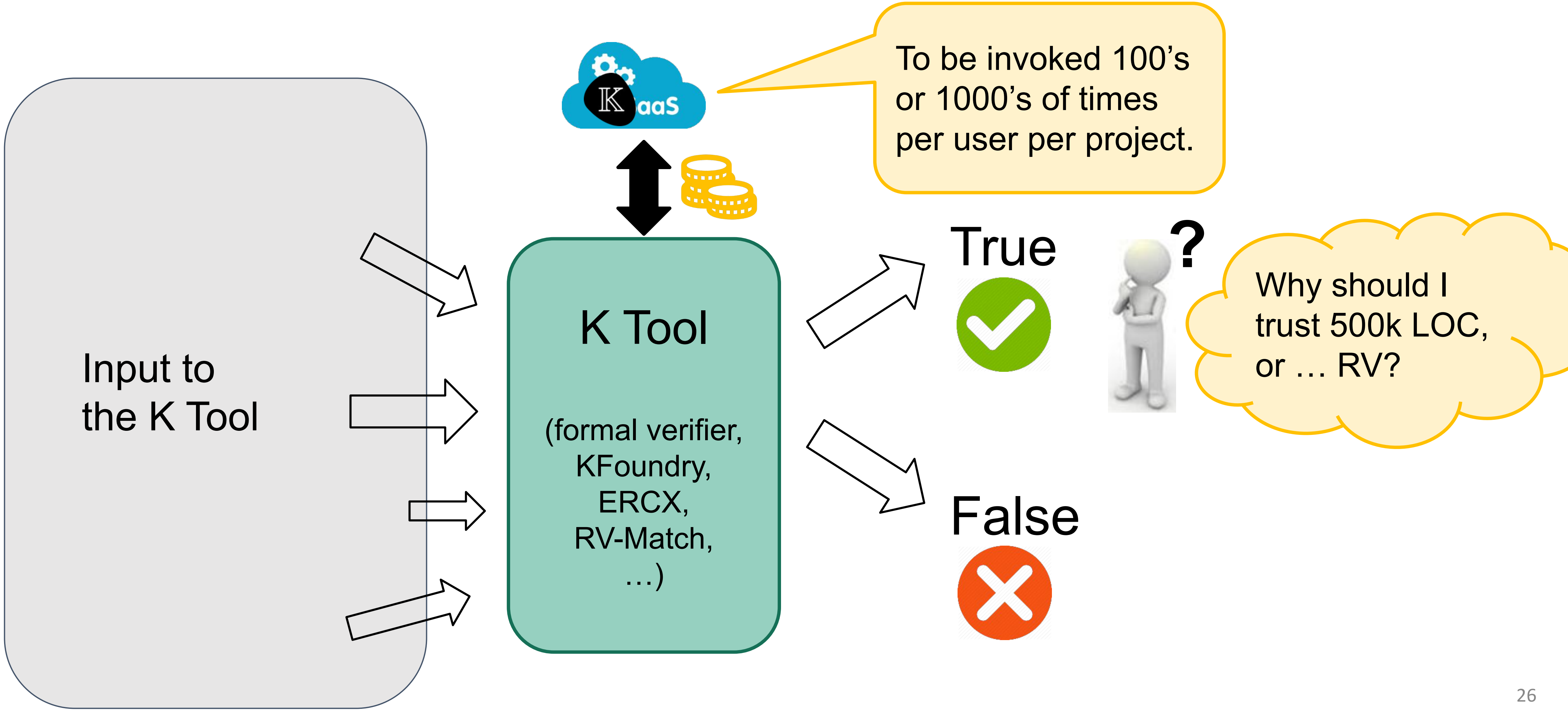
Monitoring services with recovery

- Maintain runtime integrity of protocol
  - E.g. invariant "loans are 80% collateralized" does not hold without liquidations
- Be keepers ourselves

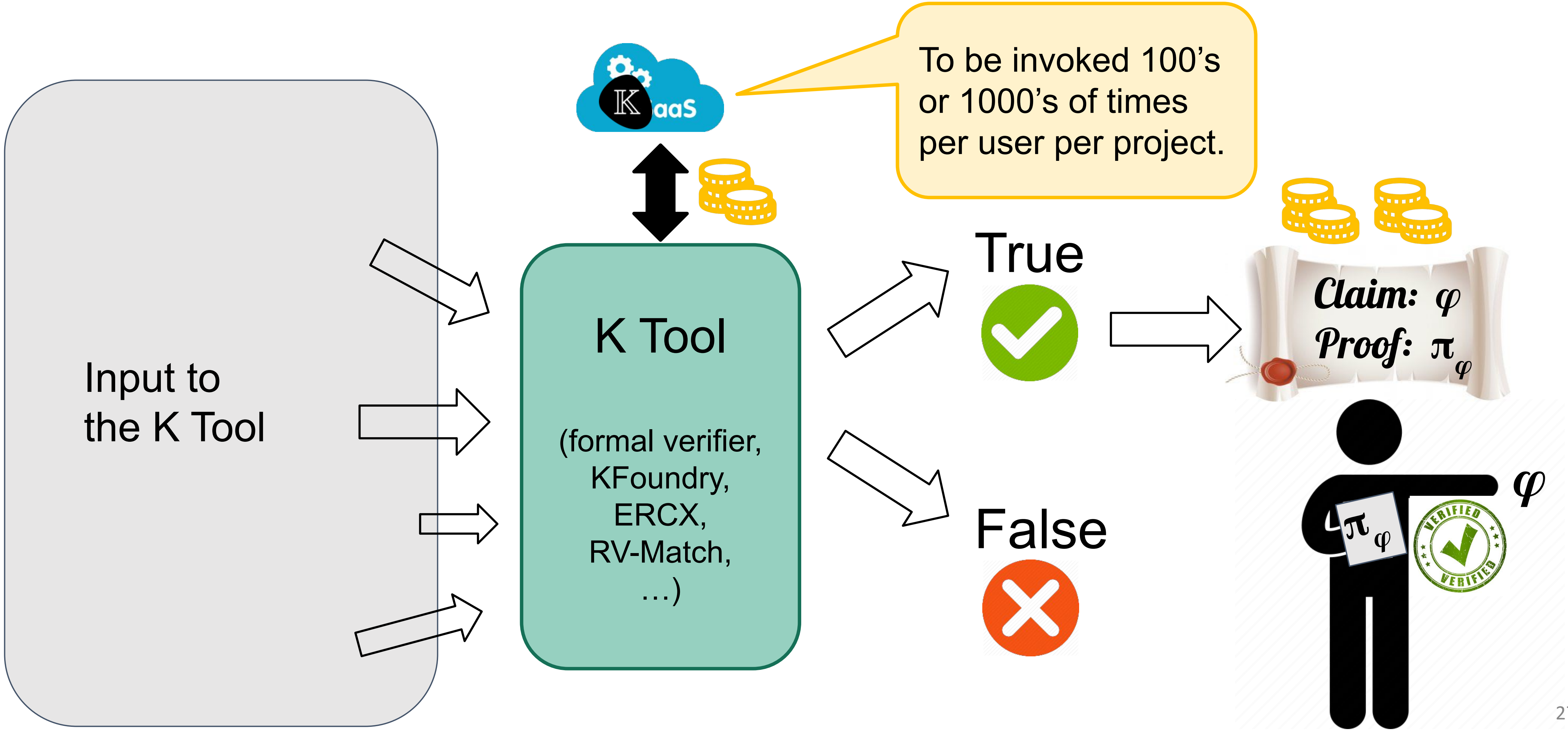
2024 Product: Proof (of Proof) Certificates



2024 Product: Proof (of Proof) Certificates



2024 Product: Proof (of Proof) Certificates



K Advantage: multi-chain, multi-language audits / tools / proofs

RV's core technology platform, dedicated blockchain teams and automated verification tools allow us to quickly and efficiently add new chain and language support based on market demand. So far:

Algorand™
ethereum
CARDANO
Polkadot.
SOLANA
COSMOS
STARKWARE
elrond
Tezos

C
C++
Cairo
Haskell
Ligo
Michelson
Plutus
Reach
Rust
Solidity / EVM
Teal
WASM

runtime verification

Contact

Grigore Rosu

Founder & CEO, RV

Professor of Computer Science, UIUC

grigore.rosu@runtimeverification.com

217-649-8738

runtimeverification.com