runtime verification

Formal Design, Implementation and Verification of Blockchain Languages and Virtual Machines

Grigore Rosu
University of Illinois at Urbana-Champaign, USA
Runtime Verification, Inc.

11 March 2019, Microsoft, Redmond, USA

Cryptocurrency – The future of Money? Built on Blockchain Technology

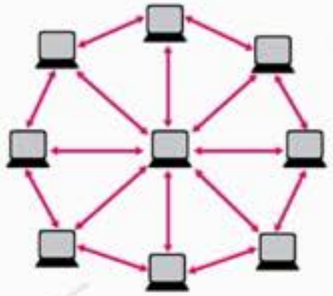
Top 100 Cryptocurrencies By Market Capitalization

Cryptocurrencies ▾ | Watchlist | USD ▾ | Next 100 → | View All

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	$111,849,112,426	$6,541.08	$3,084,420,000	17,099,487 BTC	-0.40%	
2	Ethereum	$50,325,873,111	$502.56	$1,256,840,000	100,139,234 ETH	0.18%	
3	Ripple	$20,841,493,766	$0.531057	$166,338,000	39,245,304,677 XRP *	-1.14%	
4	Bitcoin Cash	$14,658,142,538	$852.76	$286,924,000	17,189,100 BCH	-0.45%	
5	EOS	$9,433,586,474	$10.53	$426,761,000	896,149,492 EOS *	-1.49%	
6	Litecoin	$5,511,430,182	$96.65	$245,677,000	57,024,096 LTC	-1.23%	
7	Stellar	$4,353,972,862	$0.234017	$31,438,600	18,605,369,960 XLM *	0.02%	
8	Cardano	$4,215,300,909	$0.162583	$35,468,400	25,927,070,538 ADA *	-0.55%	
9	IOTA	$3,284,570,935	$1.18	$51,965,500	2,779,530,283 MIOTA *	-2.06%	
10	TRON	$2,819,811,587	$0.042888	$114,788,000	65,748,111,645 TRX *	-1.23%	

Blockchain Technology
Unprecedented Security Challenges

Someone requests a transaction.



The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

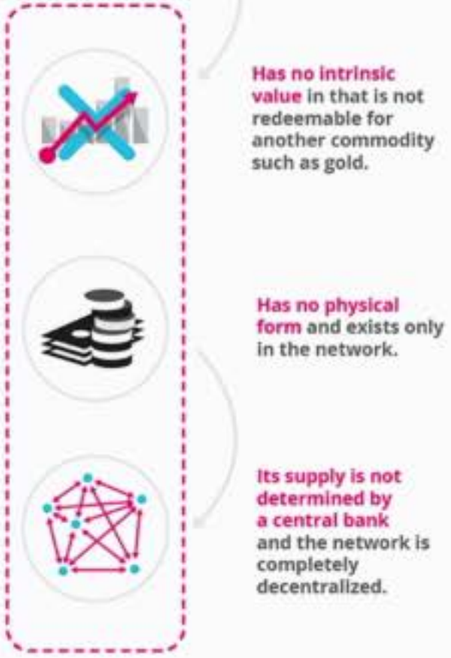
A verified transaction can involve **cryptocurrency,** contracts, records, or other information.

cryptocurrency

Once verified, the transaction is combined with other transactions **to create a new block of data for the ledger.**

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

The transaction is complete.



Has no intrinsic value in that is not redeemable for another commodity such as gold.

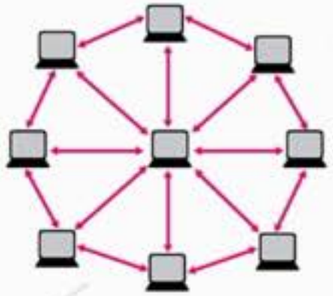
Has no physical form and exists only in the network.

Its supply is not determined by a central bank and the network is completely decentralized.

Think “execute some given, publicly visible code, with shared state”!

n Technology Security Challenges

Someone requests a transaction.



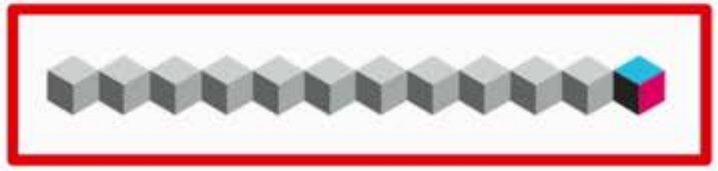
The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

A verified transaction can involve **cryptocurrency,** contracts, records, or other information.

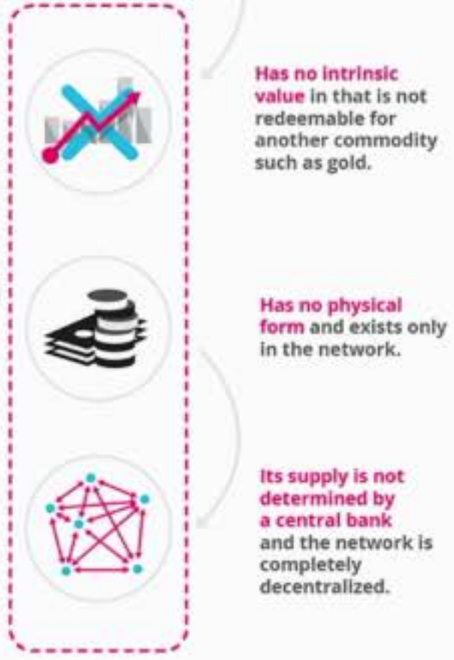
cryptocurrency



Once verified, the transaction is combined with other transactions **to create a new block of data for the ledger.**

The transaction is complete.

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.



Has no intrinsic value in that is not redeemable for another commodity such as gold.

Has no physical form and exists only in the network.

Its supply is not determined by a central bank and the network is completely decentralized.

Bloc
Unprecede

Transaction is broadcast, then "validated" by re-executing it on many "nodes", using agreed upon languages (virtual machines)

Someone requests a transaction.

The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

A verified transaction can involve **cryptocurrency,** contracts, records, or other information.

cryptocurrency

Once verified, the transaction is combined with other transactions **to create a new block of data for the ledger.**

The transaction is complete.

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

Has no intrinsic value in that is not redeemable for another commodity such as gold.

Has no physical form and exists only in the network.

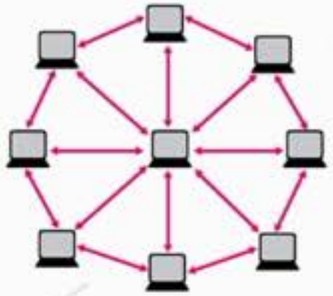
Its supply is not determined by a central bank and the network is completely decentralized.

Blockchain Technology

Unprecedented Securi

Validated transactions are then deployed by all nodes locally…

Someone requests a transaction.



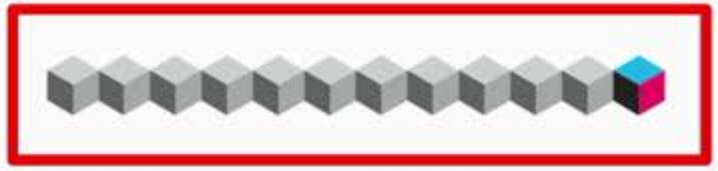
The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

A verified trans
can involve
cryptocur
contract
or oth tion.

The transaction is complete.



The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

Once verified, the transaction is combined with other transactions **to create a new block of data for the ledger.**

cryptocurrency

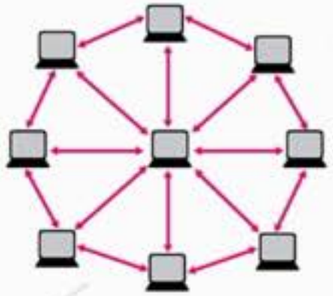
Has no intrinsic value in that is not redeemable for another commodity such as gold.

Has no physical form and exists only in the network.

Its supply is not determined by a central bank and the network is completely decentralized.

Blockchain Technology
Unprecedented Security Challenges

Someone requests a transaction.



The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

A verified transaction

…in blocks, appending each block, irreversibly, to the public “ledger” or “history” or “blockchain”.

On

…ined with other transactions **to create a new block of data for the ledger.**

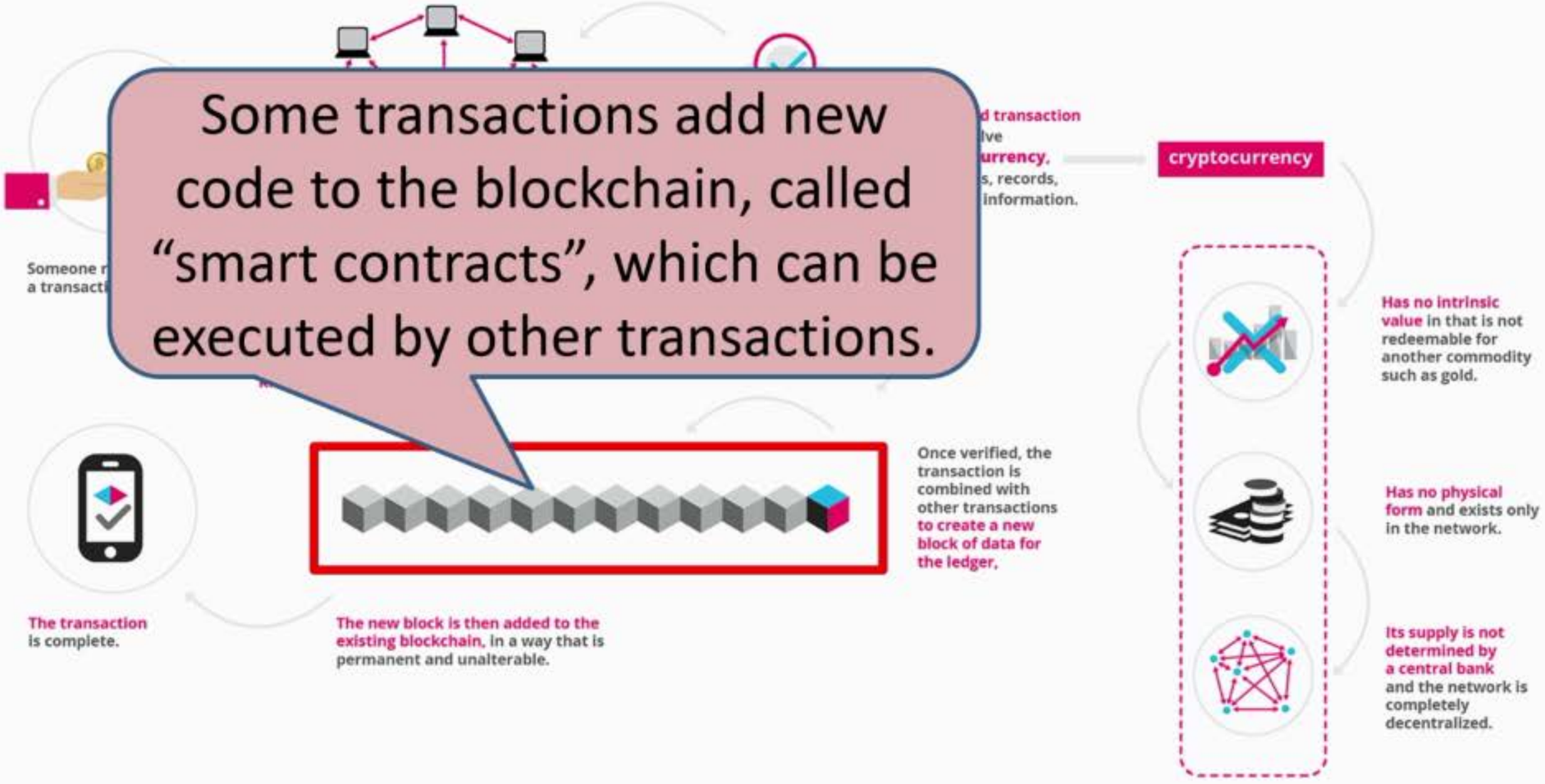
The transaction is complete.

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

Has no physical form and exists only in the network.

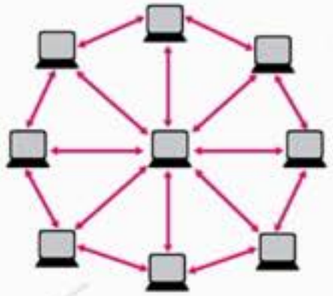
Its supply is not determined by a central bank and the network is completely decentralized.

Blockchain Technology
Unprecedented Security Challenges



Blockchain Technology
Unprecedented Security Challenges

Someone requests a transaction.



The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

A verified transaction can involve **cryptocurrency,** contracts, records, or other information.

cryptocurrency

Has no intrinsic value in that is not redeemable for another commodity such as gold.

Has no physical form and exists only in the network.

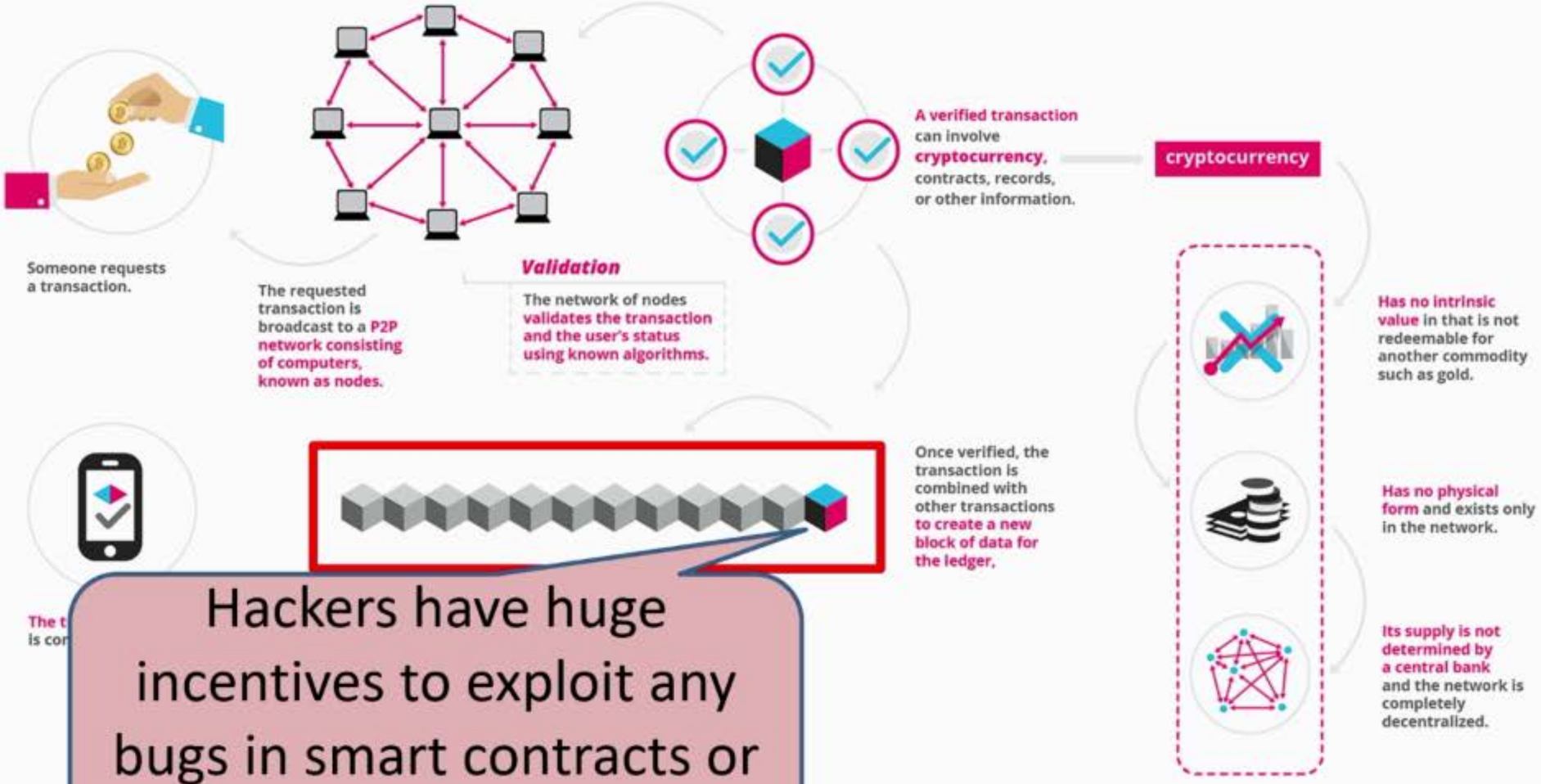
Once verified, the transaction is combined with other transactions **to create a new block of data for**

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

The transaction is complete.

In the end, all code is public, can be invoked by anybody, and can irreversibly change the history (e.g., steal your money).

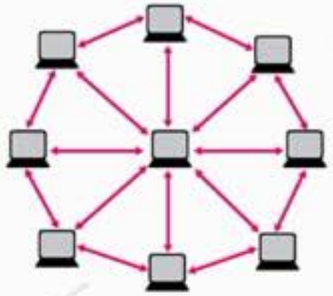
Blockchain Technology
Unprecedented Security Challenges



Hackers have huge incentives to exploit any bugs in smart contracts or underlying infrastructure.

Blockchain Technology
Unprecedented Security Challenges

Someone requests a transaction.



The requested transaction is broadcast to a **P2P network consisting of computers, known as nodes.**

Validation

The network of nodes **validates the transaction and the user's status using known algorithms.**

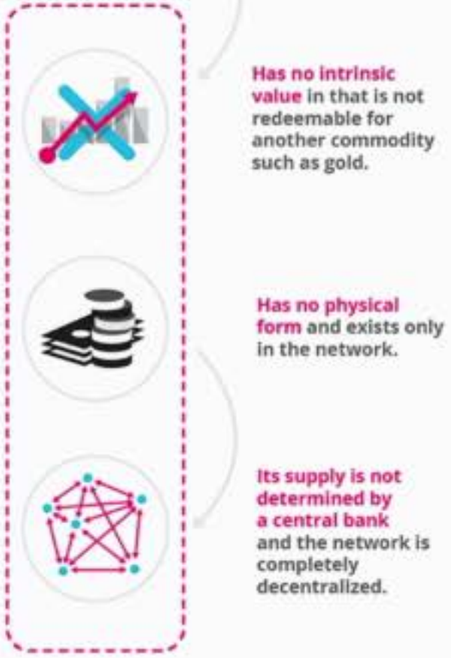
A verified transaction can involve **cryptocurrency,** contracts, records, or other information.

cryptocurrency

Once verified, the transaction is combined with other transactions **to create a new block of data for the ledger.**

The new block is then added to the existing blockchain, in a way that is permanent and unalterable.

The transaction is complete.



Has no intrinsic value in that is not redeemable for another commodity such as gold.

Has no physical form and exists only in the network.

Its supply is not determined by a central bank and the network is completely decentralized.

Smart Contract Snippet (ERC20) (one of the ~40,000 Ethereum ERC20s)

Written in Solidity:

```
function transfer(address _to, uint256 _value) returns (bool success) {
    ...

    if (_value == 0) { return false; }

    uint256 fromBalance = balances[msg.sender];

    bool sufficientFunds = fromBalance >= _value;
    bool overflowed = balances[_to] + _value < balances[_to];

    if (sufficientFunds && !overflowed) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;

        Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; }
}
```

Smart Contract Snippet (ERC20) (one of the ~40,000 Ethereum ERC20s)

Written in Solidity:

```
function transfer(address _to, uint256 _value) returns (bool success) {
    ...

    if (_value == 0) { return false; }

    uint256 fromBalance = balances[msg.sender];

    bool sufficientFunds = fromBalance >= _value;
    bool overflowed = balances[_to] + _value < balances[_to];

    if (sufficientFunds && !overflowed) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;

        Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; }
}
```

ERC20 does not state that...

Smart Contract Snippet (ERC20) (one of the ~40,000 Ethereum ERC20s)

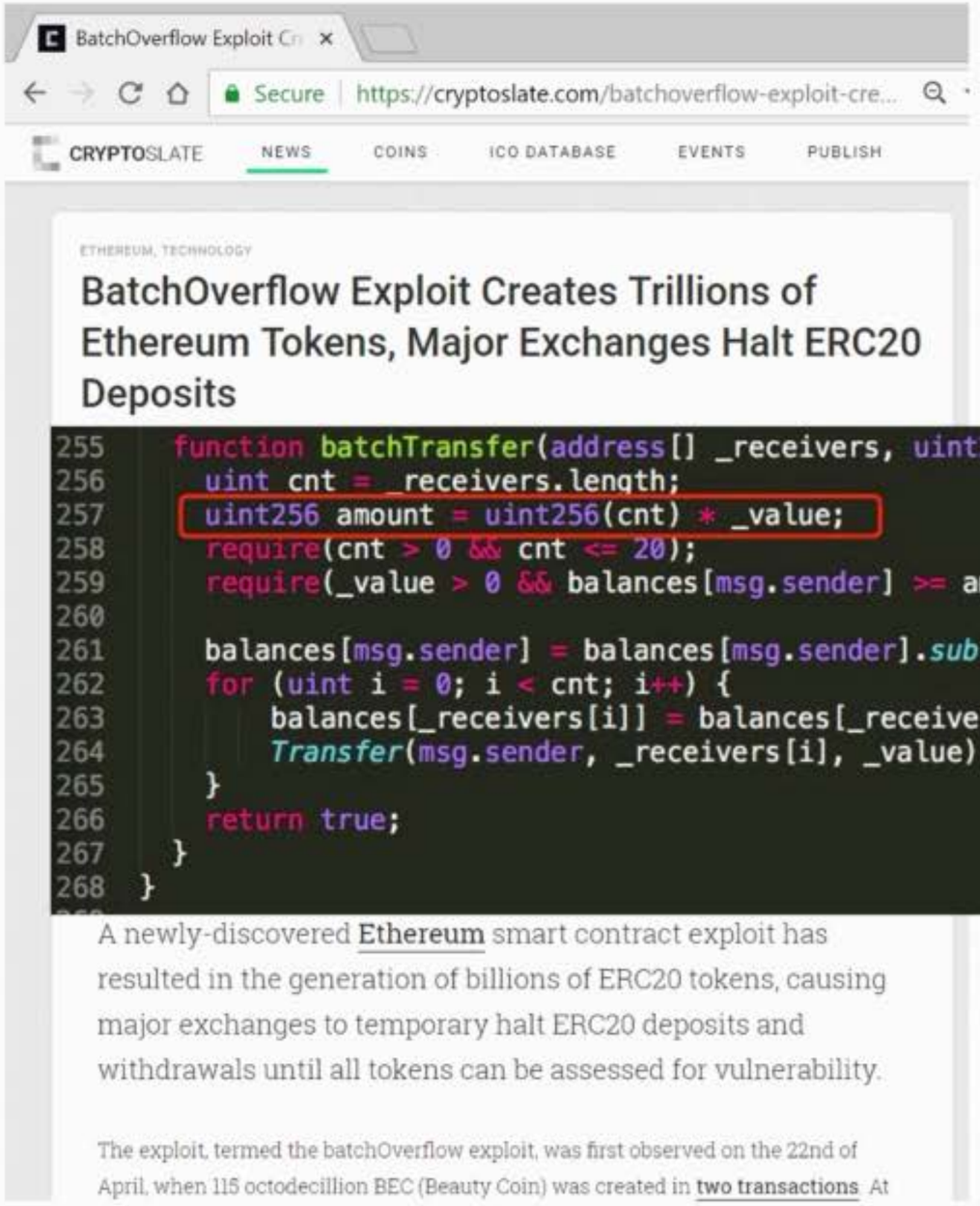
Written in Solidity:

```
function transfer(address _to, uint256 _value) returns (bool success) {
    ...

    if (_value == 0) { return false; }

    uint256 fromBalance = balances[msg.sender];

    bool sufficientFunds = fromBalance >= _value;
    bool overflowed = balances[_to] + _value < balances[_to];

    if (sufficientFunds && !overflowed) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;

        Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; }
}
```

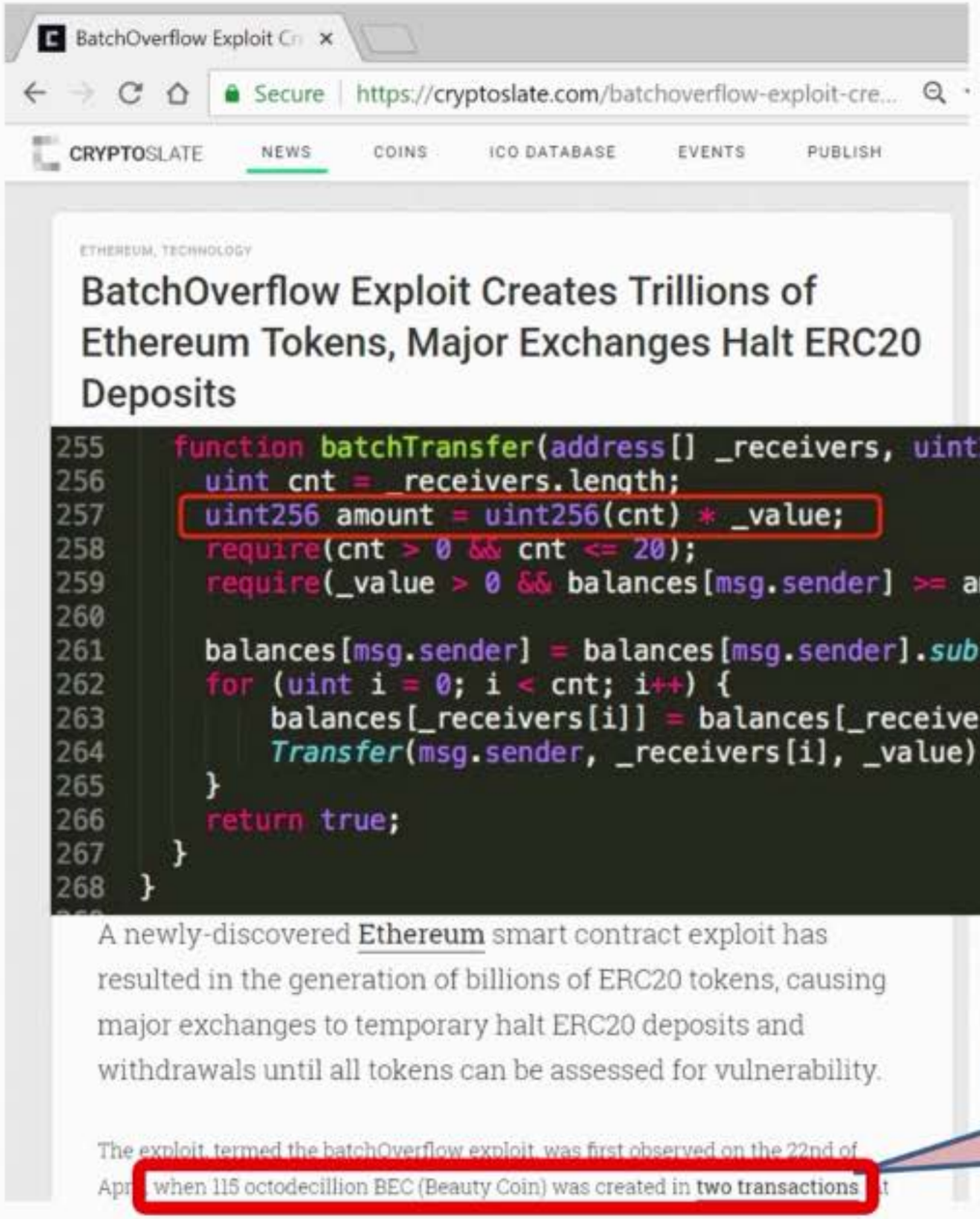
ERC20 does not state that...

There should be no overflow when self-transfer...

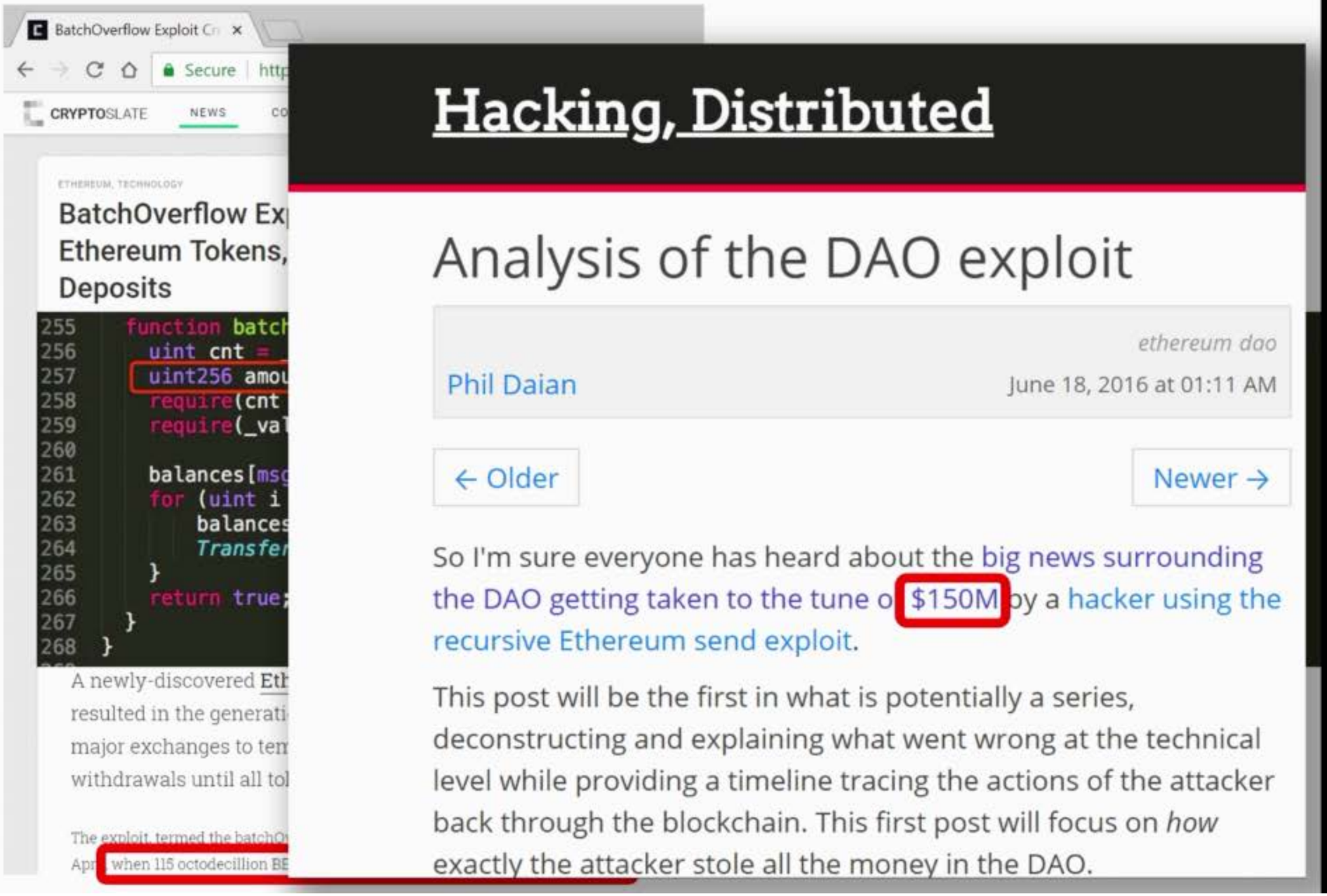
Attacks Happened. Many.



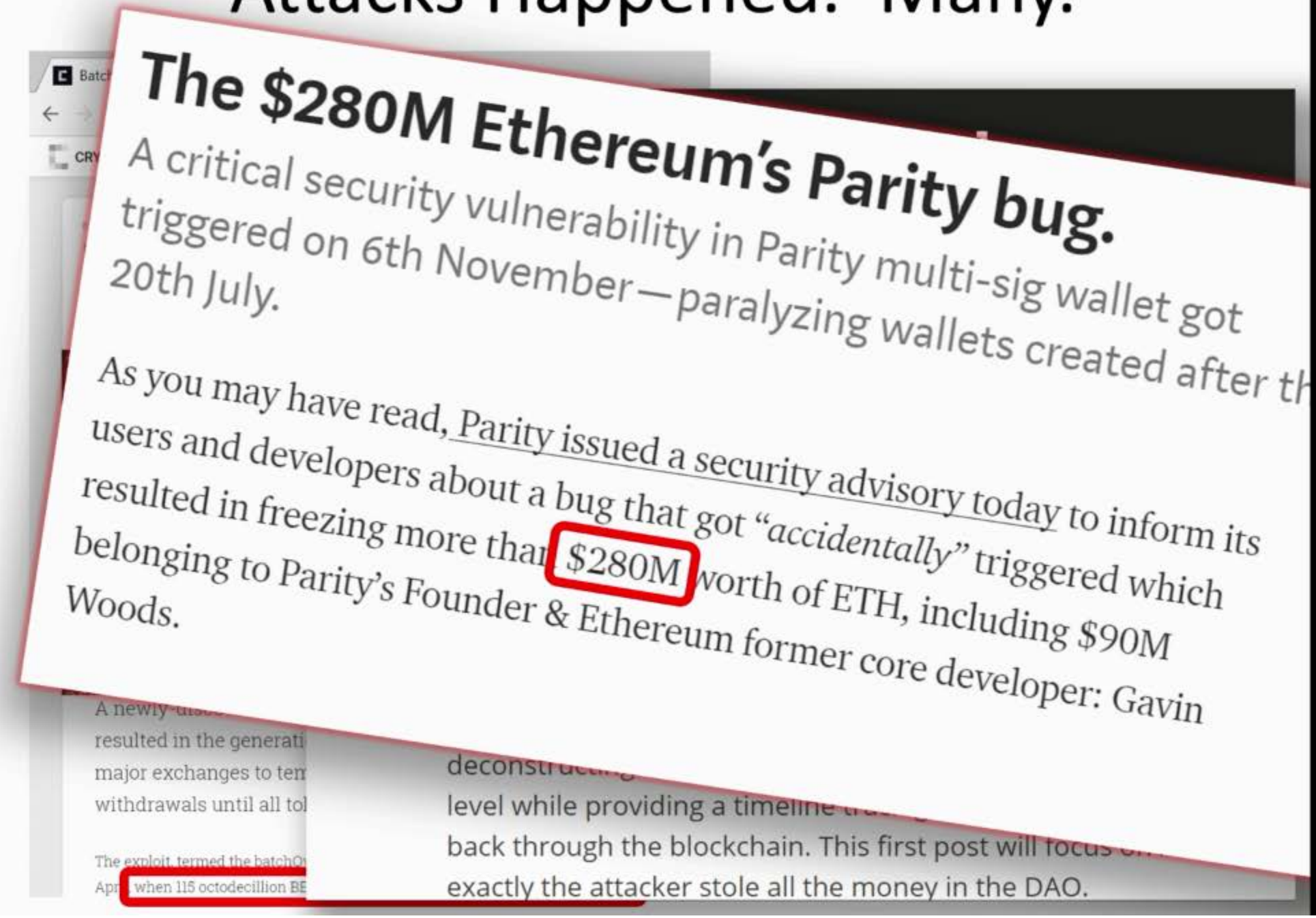
Attacks Happened. Many.



Attacks Happened. Many.



Attacks Happened. Many.



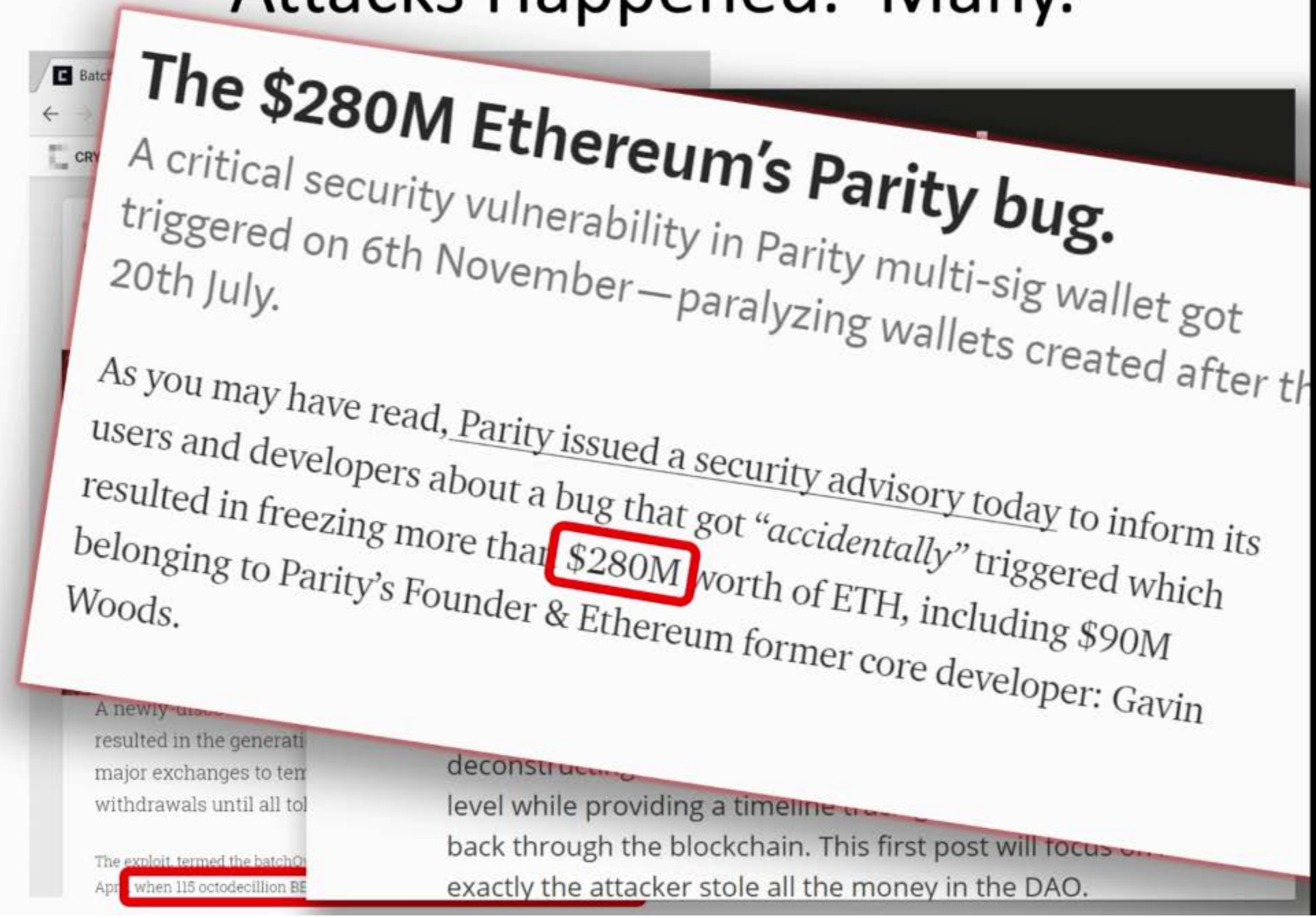
The $280M Ethereum's Parity bug.

A critical security vulnerability in Parity multi-sig wallet got triggered on 6th November—paralyzing wallets created after th 20th July.

As you may have read, Parity issued a security advisory today to inform its users and developers about a bug that got "*accidentally*" triggered which resulted in freezing more than $280M worth of ETH, including $90M belonging to Parity's Founder & Ethereum former core developer: Gavin Woods.


Parity Multisig Hacked. Again
Yesterday, Parity Multisig Wallet was hacked again:
https://paritytech.io/blog/security-alert.html
"This means that currently no funds can be moved out of the [ANY Parity] multi-sig wallets"
A lot of people/companies/ICOs are using Parity-generated multisig wallets.
Abou $300M s frozen and (probably) lost forever.
Disclaimer: I lost little money (about $1000) but my friends lost about $300K.
Who hacked it?
Some guy with a nickname @devops199 (not a member of the Parity team)
and an "empty" github account. His Ethereum address is
0xae7168Deb525862f4FEe37d987A971b385b96952 and he has successfully
verified it.
exactly the attacker stole all the money in the DAO.

Attacks Happened. Many.



The $280M Ethereum's Parity bug.

A critical security vulnerability in Parity multi-sig wallet got triggered on 6th November—paralyzing wallets created after th 20th July.

As you may have read, Parity issued a security advisory today to inform its users and developers about a bug that got "*accidentally*" triggered which resulted in freezing more than $280M worth of ETH, including $90M belonging to Parity's Founder & Ethereum former core developer: Gavin Woods.


Parity Multisig Hacked. Again
Yesterday, Parity Multisig Wallet was hacked again:
https://paritytech.io/blog/security-alert.html
"This means that currently no funds can be moved out of the [ANY Parity] multi-sig wallets"
A lot of people/companies/ICOs are using Parity-generated multisig wallets.
Abou $300M s frozen and (probably) lost forever.
Disclaimer: I lost little money (about $1000) but my friends lost about $300K.
Who hacked it?
Some guy with a nickname @devops199 (not a member of the Parity team)
and an "empty" github account. His Ethereum address is
0xae7168Deb525862f4FEe37d987A971b385b96952 and he has successfully
verified it.
exactly the attacker stole all the money in the DAO.

What Can We Do About This?

- More specifically, what can we do about the execution environment, to increase security?
  - Unacceptable to build this complex and disruptive technology with poorly designed VMs and languages!
- Ideal scenario feasible, stop compromising!
  - Everything must be rigorously designed, using formal methods. Implementations must be provably correct!

What Can We Do About This?

- More specifically, what can we do about the execution environment, to increase security?
  - Unacceptable to build this complex and disruptive technology with poorly designed VMs and languages!
- Ideal scenario feasible, stop compromising!
  - Everything must be rigorously designed, using formal methods. Implementations must be provably correct!
    - Nodes: provably correct VMs or interpreters

What Can We Do About This?

- More specifically, what can we do about the execution environment, to increase security?
  - Unacceptable to build this complex and disruptive technology with poorly designed VMs and languages!
- Ideal scenario feasible, stop compromising!
  - Everything must be rigorously designed, using formal methods. Implementations must be provably correct!
    - Nodes: provably correct VMs or interpreters
    - Smart contracts: use well-designed programming languages, with provably correct compilers or interpreters

What Can We Do About This?

- More specifically, what can we do about the execution environment, to increase security?
  - Unacceptable to build this complex and disruptive technology with poorly designed VMs and languages!
- Ideal scenario feasible, stop compromising!
  - Everything must be rigorously designed, using formal methods. Implementations must be provably correct!
    - Nodes: provably correct VMs or interpreters
    - Smart contracts: use well-designed programming languages, with provably correct compilers or interpreters
    - Verification: Smart contracts provably correct wrt their specs

What Can We Do About This?

- More specifically, what can we do about the execution environment, to increase security?
  - Unacceptable to build this complex and disruptive technology with poorly designed VMs and languages!
- Ideal scenario feasib… …mising!
  - Everything … …sing formal methods … …bly correct!
    - …
    - Smart … …ming languages, with pro… …eters
    - Verification: Si… …vably correct wrt their specs

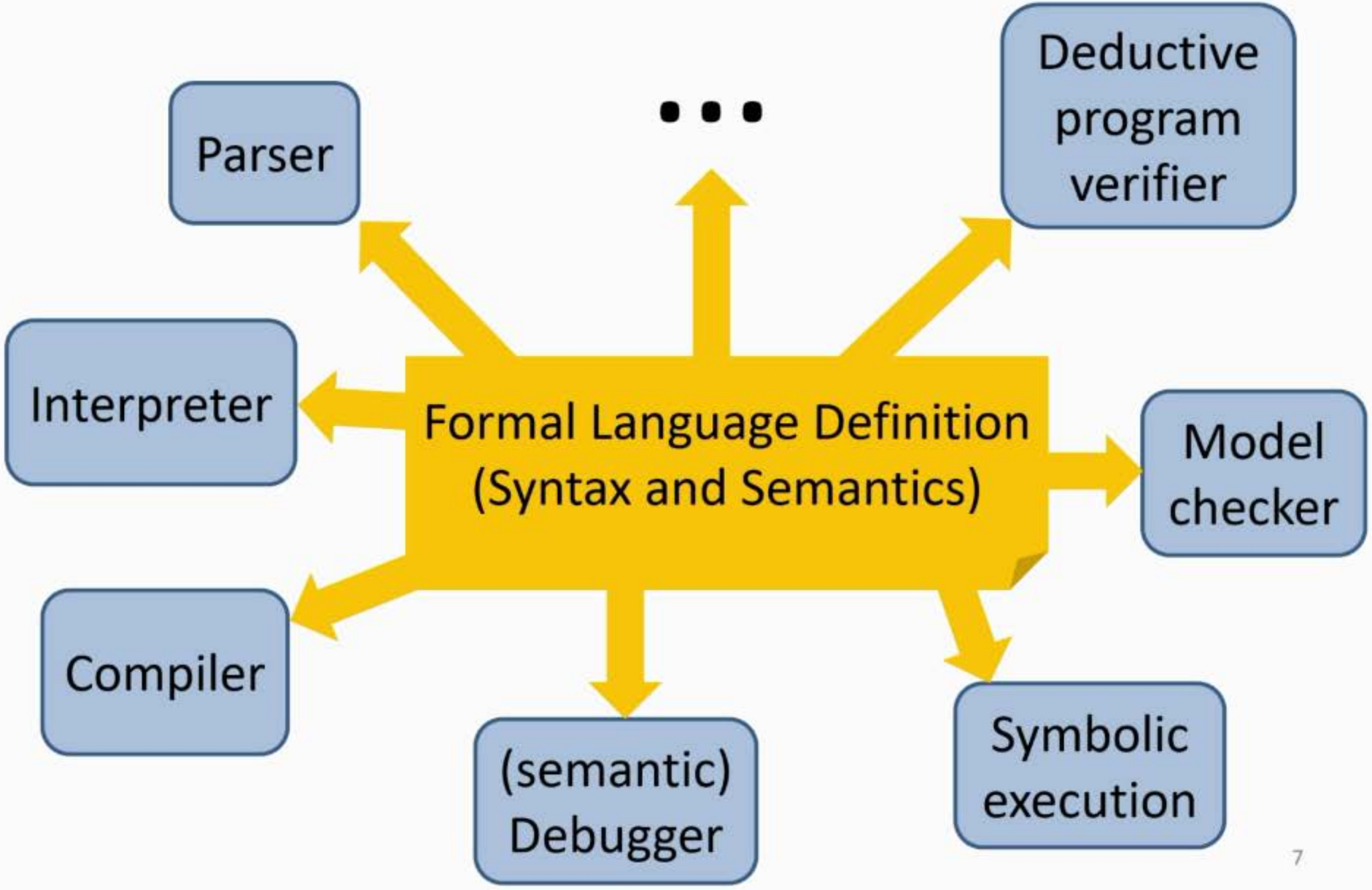
Many languages … +
Provably correct …

Language framework!

Ideal Language Framework Vision

Formal Language Definition
(Syntax and Semantics)

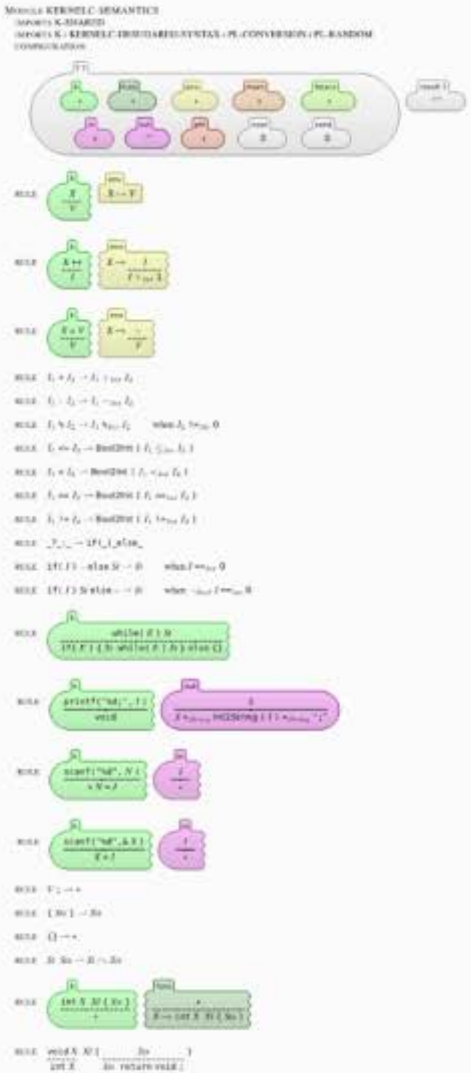
Ideal Language Framework Vision

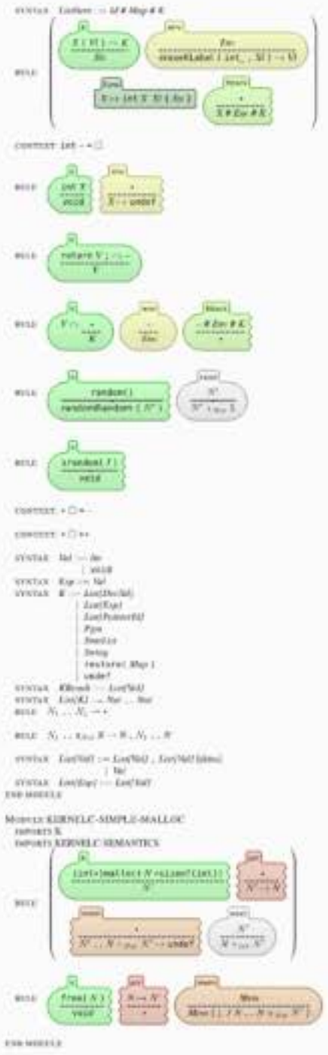


Our Attempt: the K Framework
http://kframework.org

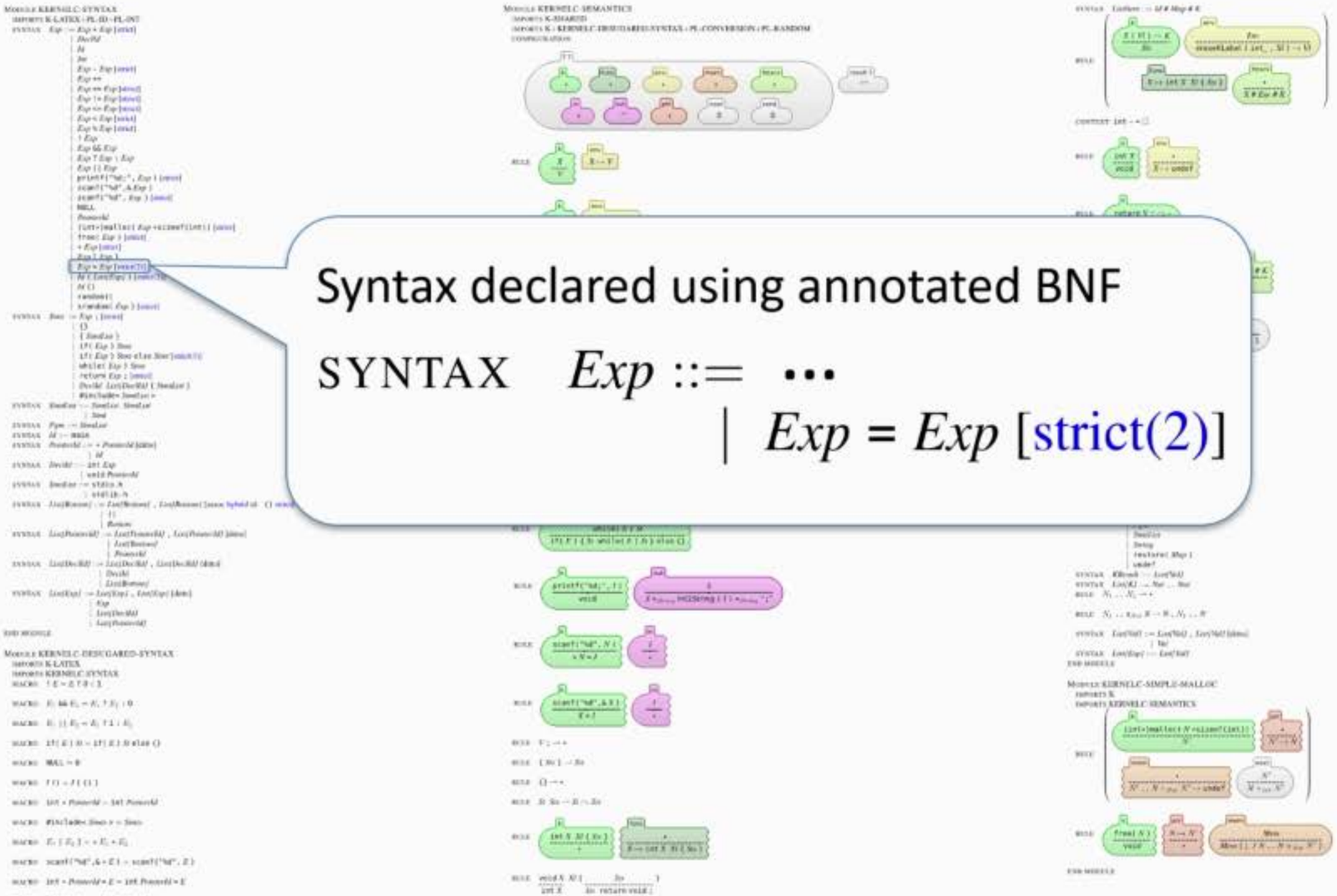
- We tried various semantic styles, for ~15y
  - Small-/big-step SOS; Evaluation contexts; Abstract machines (CC, CK, CEK, SECD, …); Chemical abstract machine; Axiomatic; Continuations; Denotational;…
- But each of the above had limitations
  - Especially related to modularity, notation, verification
- K framework initially *engineered*: keep advantages and avoid limitations of various semantic styles
  - Then theory came

Complete K Definition of KernelC





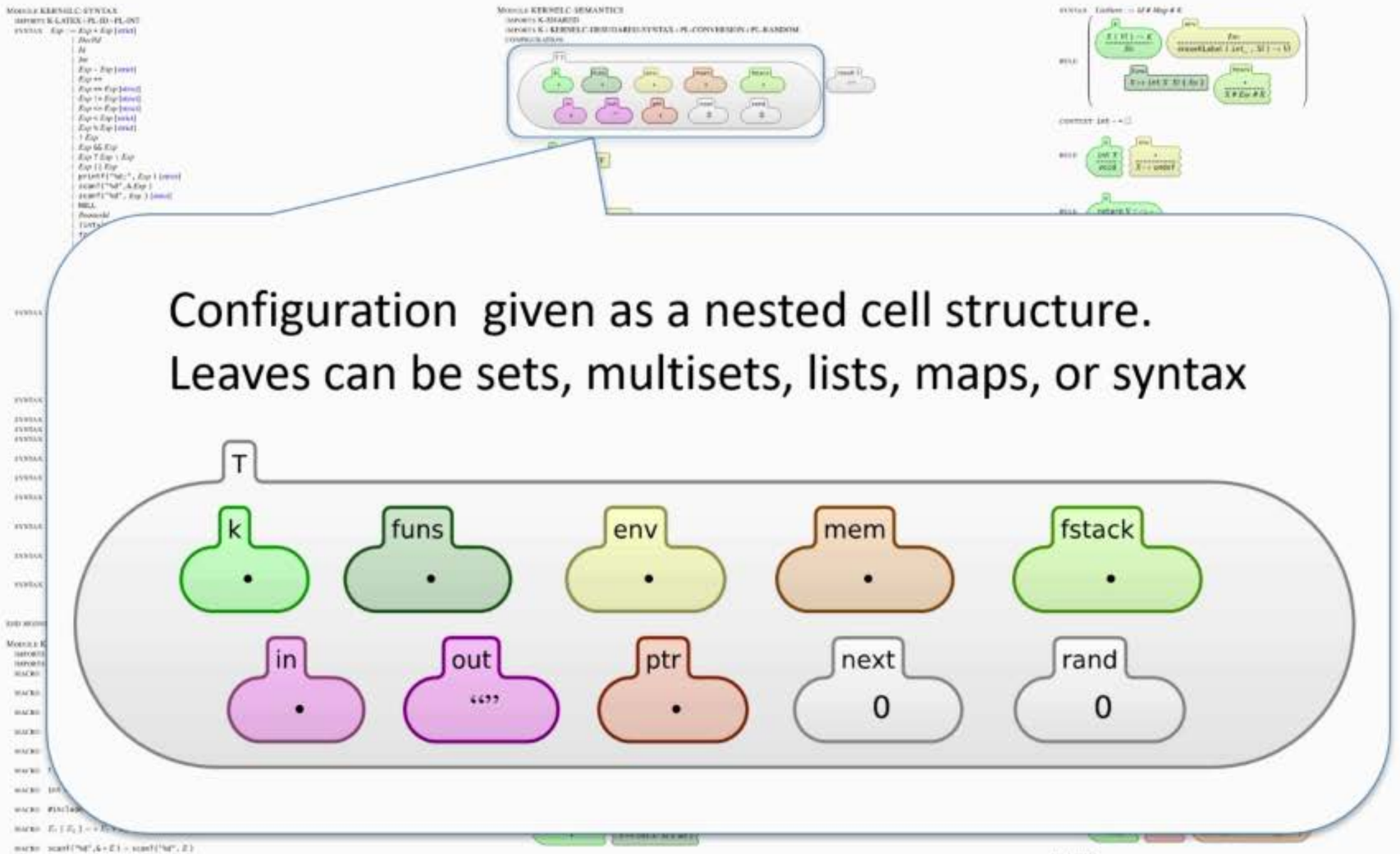
Complete K Definition of KernelC



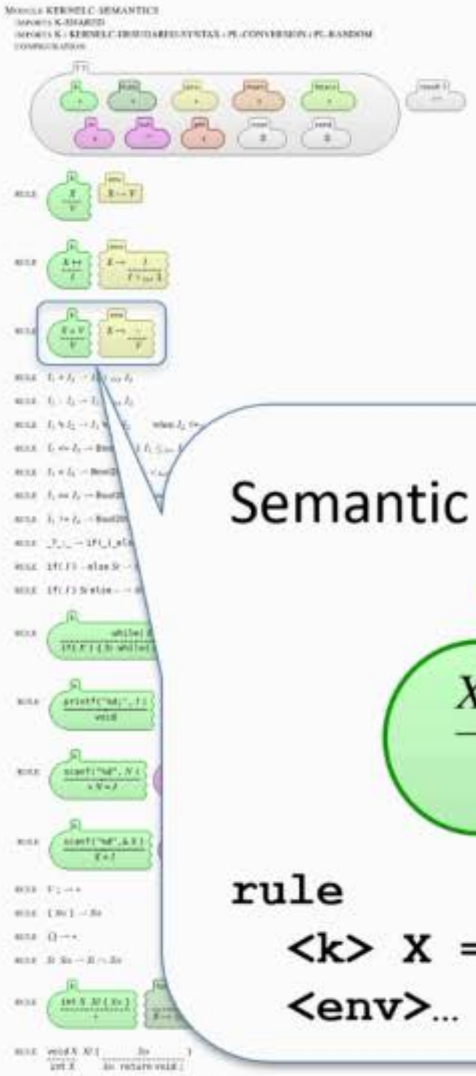
Syntax declared using annotated BNF

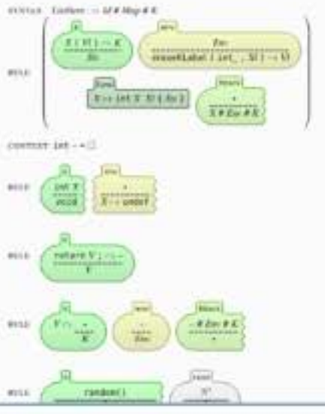
$$\text{SYNTAX} \quad Exp ::= \cdots \mid Exp = Exp \ [\text{strict}(2)]$$

Complete K Definition of KernelC

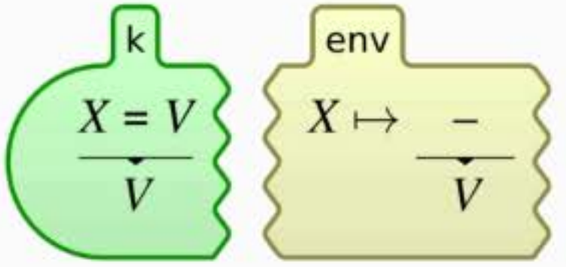


Complete K Definition of KernelC



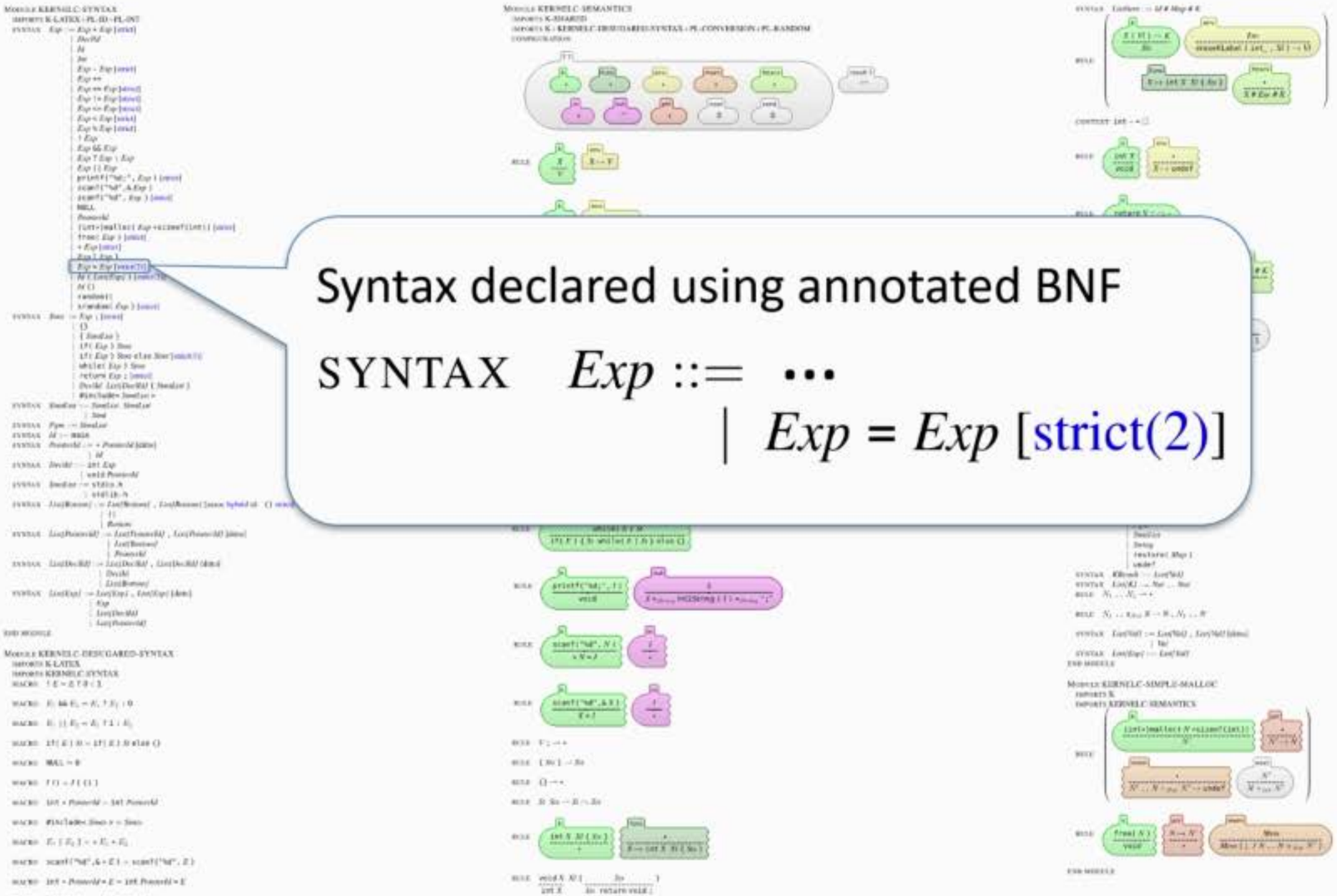


Semantic rules given contextually




```
rule
   <k> X = V => V ...</k>
   <env>... X |-> (_ => V) ...</env>
```

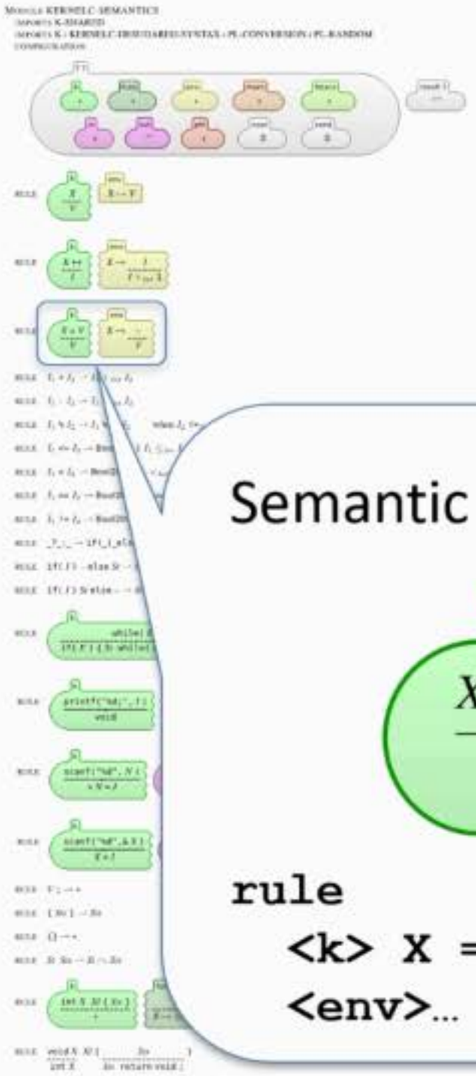
Complete K Definition of KernelC

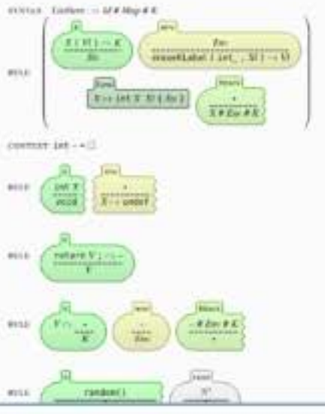


Syntax declared using annotated BNF

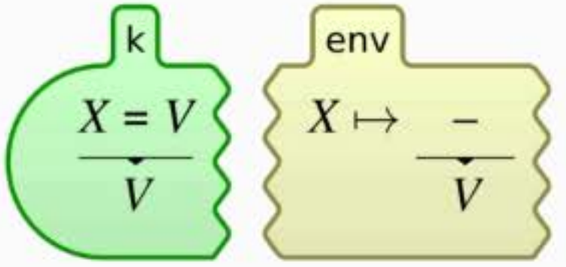
$$\text{SYNTAX} \quad Exp ::= \cdots \mid Exp = Exp \ [\text{strict}(2)]$$

Complete K Definition of KernelC





Semantic rules given contextually




```
rule
   <k> X = V => V ...</k>
   <env>... X |-> (_ => V) ...</env>
```

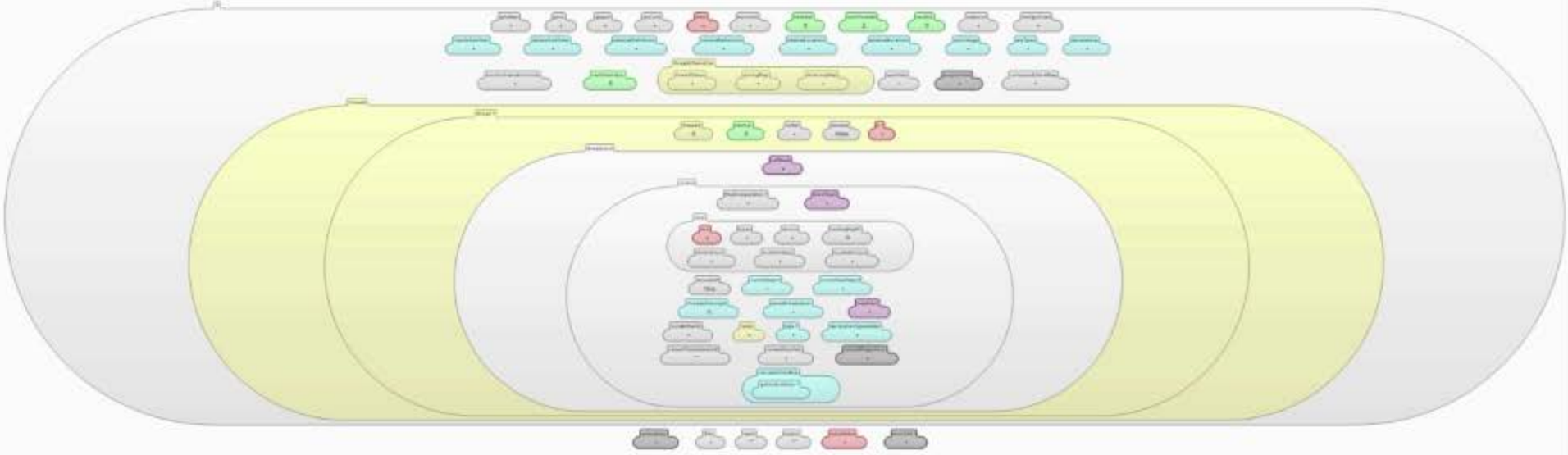
K Scales

Several large languages were recently defined in K:

- JavaScript ES5: by Park etal [PLDI'15]
  - Passes existing conformance test suite (2872 programs)
  - Found (confirmed) bugs in Chrome, IE, Firefox, Safari
- Java 1.4: by Bogdanas etal [POPL'15]
- x86: by Dasgupta etal [PLDI'19]
- C11: Ellison etal [POPL'12, PLDI'15]
  - 192 different types of undefined behavior
  - 10,000+ program tests (gcc torture tests, obfuscated C, …)
  - Commercialized by startup (Runtime Verification, Inc.)

+ EVM [CSF'18], Solidity, IELE, Plutus, Vyper, …

K Configuration and Definition of C

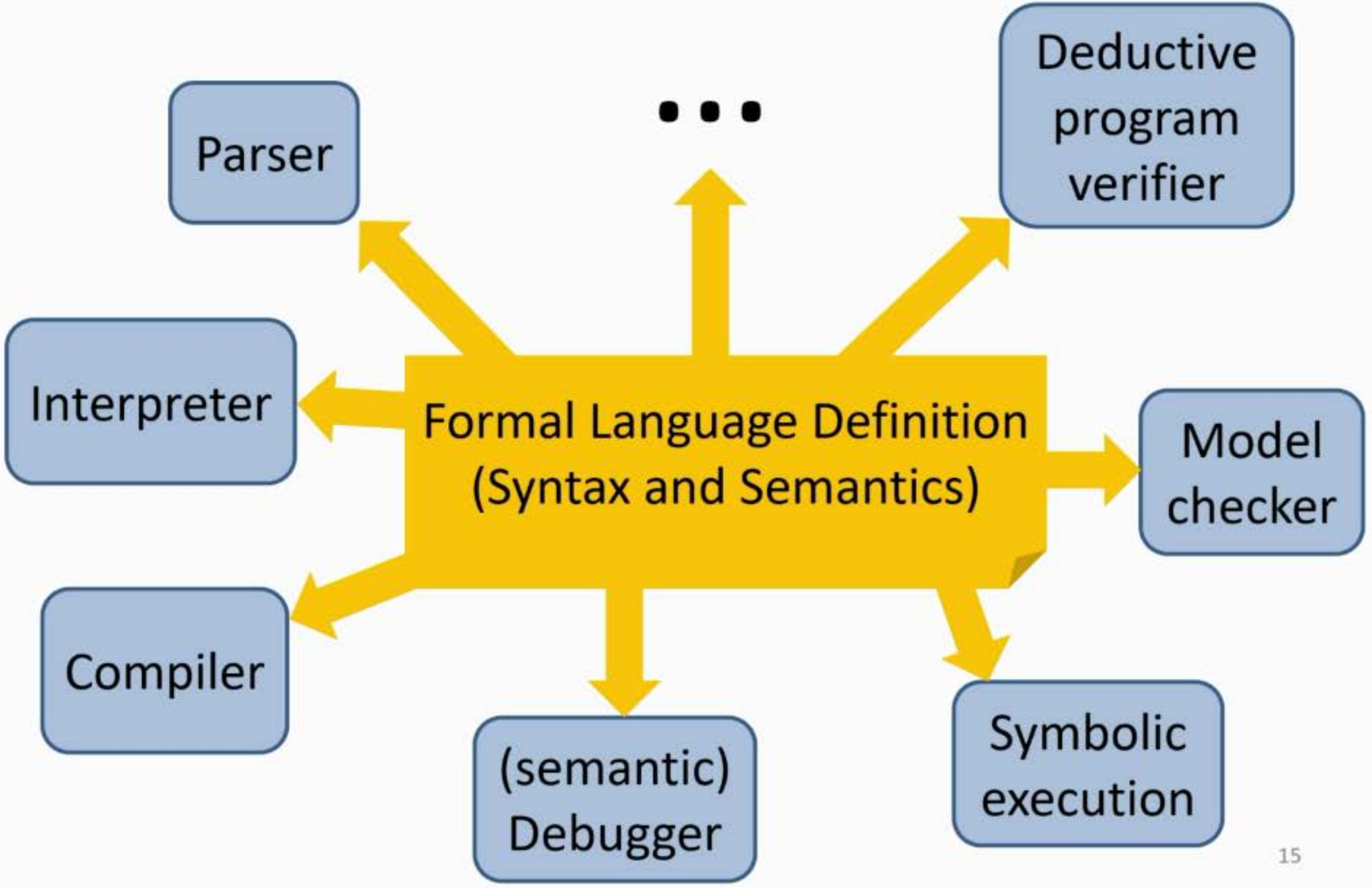


K Configuration and Definition of C

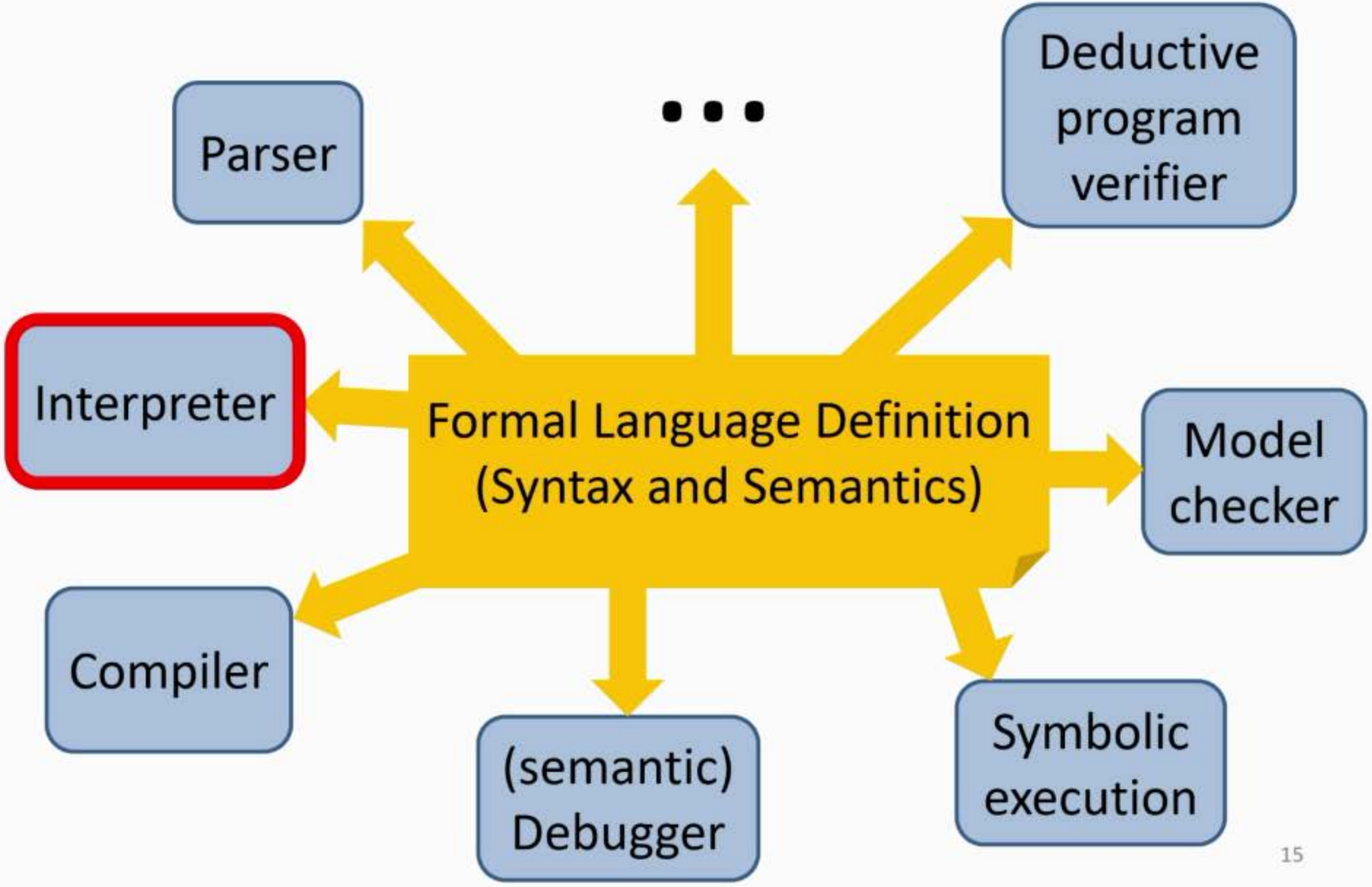
120 Cells!

… plus ~5000 rules …

Ideal Language Framework Vision [K]



Ideal Language Framework Vision [K]



OCAML backend: K -> OCAML

1. Translate K lang def to OCAML
2. Compile OCAML code natively

OCAML backend: K -> OCAML

1. Translate K lang def to OCAML
2. Compile OCAML code natively

runtime verification match

RV-Match: Commercial tool

- Instance of K -> OCAML with ISO C11 language
- an automatic debugger for subtle bugs other tools can't find, with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a platform for analyzing programs, boosting standards compliance and assurance

OCAML backend: K -> OCAML

1. Translate K lang def to OCAML
2. Compile OCAML code natively

runtime verification match

Code (6-int-overflow.c)

```
int main() {
  short int a = 1;
  int i;
  for (i = 0; i < 15; i++) {
    a *= 2;
  }
  return a;
}
```

RV-Match: Commercial tool

- Instance of K -> OCAML with ISO C11 language
- an automatic debugger for subtle bugs other tools can't find, with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a platform for analyzing programs, boosting standards compliance and assurance

OCAML backend: K -> OCAML

1. Translate K lang def to OCAML
2. Compile OCAML code natively

runtime verification match

Code (6-int-overflow.c)

```
int main() {
  short int a = 1;
  int i;
  for (i = 0; i < 15; i++) {
    a *= 2;
  }
  return a;
}
```

RV-Match: Commercial tool

- Instance of K -> OCAML with ISO C11 language
- an automatic debugger for subtle bugs other tools can't find, with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a platform for analyzing programs, boosting standards compliance and assurance

```
$ gcc 6-int-overflow.c
$ ./a.out
$
$ kcc 6-int-overflow.c
$ ./a.out
Error: IMPL-CCV2
Description: Conversion to signed integer outside the range that can be represented.
Type: Implementation defined behavior.
See also: C11 sec. 6.3.1.3:3, J.3.5:1 item 4
  at main(6-int-overflow.c:29)
```

RV-Match on Toyota ITC Benchmark
- Comparison with Static Analysis Tools -

[CAV'16]

Toyota Benchmark ISSRE '15 (1276 tests)	RV-Match			GrammaTech CodeSonar			MathWorks Code Prover			MathWorks Bug Finder			GCC			Clang		
	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM
Static memory	100	100	100	100	100	100	97	100	98	97	100	98	0	100	0	15	100	39
Dynamic memory	94	100	97	89	100	94	92	95	93	90	100	95	0	100	0	0	100	0
Stack-related	100	100	100	0	100	0	60	70	65	15	85	36	0	100	0	0	100	0
Numerical	96	100	98	48	100	69	55	99	74	41	100	64	12	100	35	11	100	33
Resource management	93	100	96	61	100	78	20	90	42	55	100	74	6	100	25	3	100	18
Pointer-related	98	100	99	52	96	71	69	93	80	69	100	83	9	100	30	13	100	36
Concurrency	67	100	82	70	77	73	0	100	0	0	100	0	0	100	0	0	100	0
Inappropriate code	0	100	0	46	99	67	1	97	10	28	94	51	2	100	13	0	100	0
Miscellaneous	63	100	79	69	100	83	83	100	91	69	100	83	11	100	34	11	100	34
AVERAGE (Unweighted)	79	100	89	59	97	76	53	94	71	52	98	71	4	100	20	6	100	24
AVERAGE (Weighted)	82	100	91	68	98	82	53	95	71	62	99	78	5	100	22	7	100	26

DR: Percent of programs with defects where defects are reported

FPR: Percent of programs without defects, with defects incorrectly reported; $\underline{\text{FPR}} = 100 - \text{FPR}$

PM: Productivity metric: $\sqrt{\text{DR} \times (100 - \text{FPR})}$

RV-Match on Toyota ITC Benchmark
- Comparison with Other Analysis Tools -

Toyota Benchmark ISSRE '15 (1276 tests)	RV-Match			Valgrind + Helgrind (GCC)			UBSan + TSan + MSan + ASan (Clang)			Frama-C (Value Analysis Plugin)			Compcert Interpreter		
	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM
Static memory	100	100	100	9	100	30	79	100	89	82	96	89	97	82	89
Dynamic memory	94	100	97	80	95	87	16	95	39	79	27	46	29	80	48
Stack-related	100	100	100	70	80	75	95	75	84	45	65	54	35	70	49
Numerical	96	100	98	22	100	47	59	100	77	79	47	61	48	79	62
Resource management	93	100	96	57	100	76	47	96	67	63	46	54	32	83	52
Pointer-related	98	100	99	60	100	77	58	97	75	81	40	57	87	73	80
Concurrency	67	100	82	72	79	76	67	72	70	7	100	26	58	42	49
Inappropriate code	0	100	0	2	100	13	0	100	0	33	63	45	17	83	38
Miscellaneous	63	100	79	29	100	53	37	100	61	83	49	63	63	71	67
AVERAGE (Unweighted)	79	100	89	44	95	65	51	93	69	61	59	60	52	74	62
AVERAGE (Weighted)	82	100	91	42	97	65	47	95	67	66	55	60	51	76	63

DR: Percent of programs with defects where defects are reported

FPR: Percent of programs without defects, with defects incorrectly reported; $\underline{\text{FPR}} = 100 - \text{FPR}$

PM: Productivity metric: $\sqrt{\text{DR} \times (100 - \text{FPR})}$

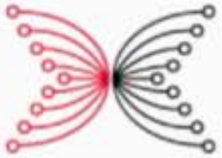
From RV-Match to Blockchain

- RV-Match currently commercialized within

DENSO
Crafting the Core

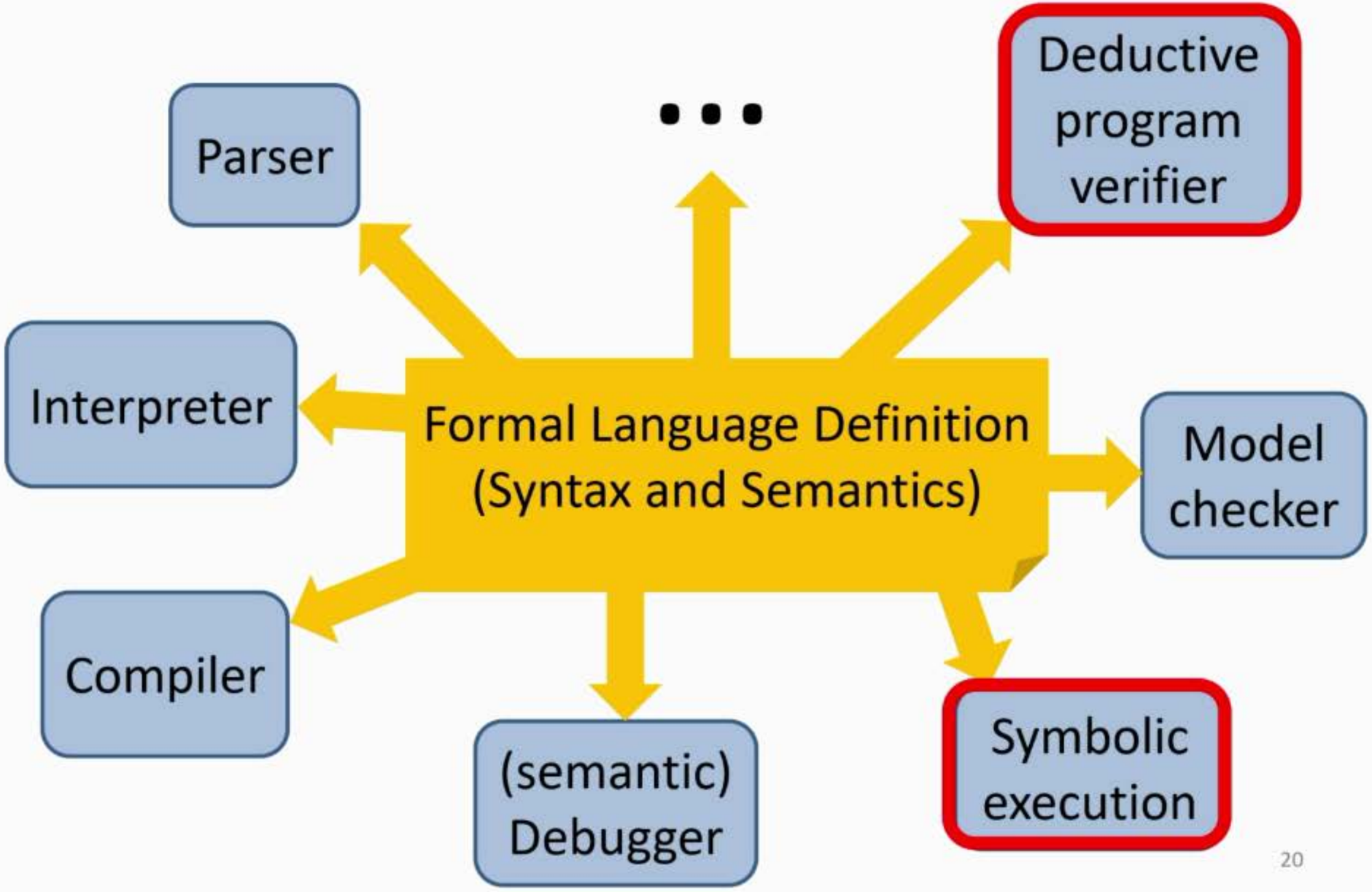
- The same technology, K, used for defining blockchain languages: EVM, IELE, Plutus, …

ethereum



INPUT | OUTPUT

Ideal Language Framework Vision [K]



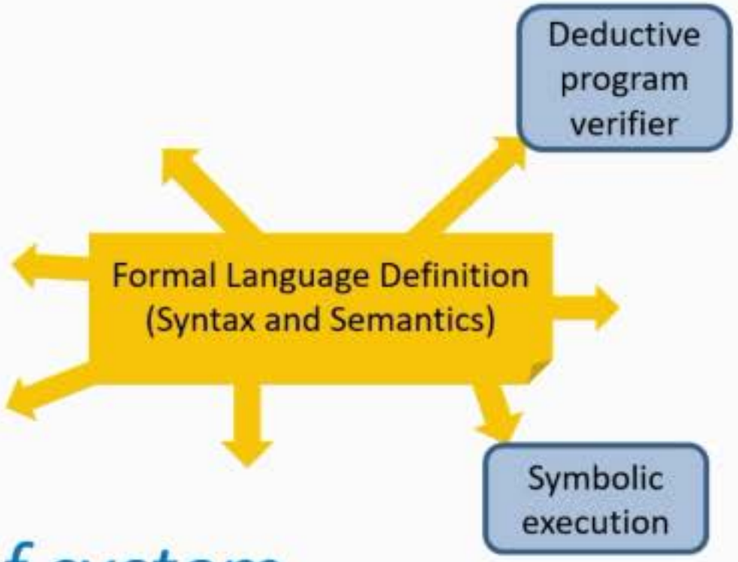
State-of-the-Art

- Redefine the language using a different semantic approach (Hoare/separation/dynamic logic)
- Language specific, non-executable, error-prone

$$\frac{\mathcal{H} \vdash \{\psi \wedge \mathtt{e} \neq 0\}\, \mathtt{s}\, \{\psi\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{while(e)}\, \mathtt{s}\, \{\psi \wedge \mathtt{e} = 0\}}$$

$$\frac{\mathcal{H} \cup \{\psi\}\, \mathtt{proc()}\, \{\psi'\} \vdash \{\psi\}\, \mathtt{body}\, \{\psi'\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{proc()}\, \{\psi'\}}$$

What We Want



- Use directly the trusted executable semantics!
- *Language-independent proof system*
  - Takes operational semantics as axioms
  - Derives reachability properties
  - Sound and relatively complete for all languages!

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

Patterns (of each sort s)

$$
\begin{array}{rcll}
\varphi_s & ::= & x \in Var_s & \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) & with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s & \\
& | & \varphi_s \wedge \varphi_s & \\
& | & \exists x \,.\, \varphi_s\ with\ x \in Var & (of\ any\ sort)
\end{array}
$$

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

Structure

$$
\begin{array}{rcll}
\varphi_s & ::= & x \in Var_s & \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) & with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s & \\
& | & \varphi_s \wedge \varphi_s & \\
& | & \exists x \,.\, \varphi_s\ with\ x \in Var & (of\ any\ sort)
\end{array}
$$

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

$$
\begin{array}{lcll}
\varphi_s & ::= & x \in Var_s & \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) & with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg\varphi_s & \\
& | & \varphi_s \wedge \varphi_s & \\
& | & \exists x \,.\, \varphi_s\ with\ x \in Var & (of\ any\ sort)
\end{array}
$$

Constraints

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

$$
\begin{array}{rcll}
\varphi_s & ::= & x \in Var_s & \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) & with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s & \\
& | & \varphi_s \wedge \varphi_s & \\
& | & \exists x \,.\, \varphi_s & with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

Binders

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

Matching Logic Models

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, ..., \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 ... s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

$$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$$

Matching Logic Proof System

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2) \quad$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi] \quad$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Matching Logic Proof System

First-Order Logic

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2)$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi]$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Matching Logic Proof System

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
(∀)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2)$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi]$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

$C_\sigma \equiv \sigma(\psi_1, \ldots, \psi_{i-1}, \Box, \psi_{i+1}, \ldots, \psi_n)$

Matching Logic Proof System

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2)$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi]$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Local reasoning

Matching Logic Proof System

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2)$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi]$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Technical (completeness)

Matching Logic

[…, LICS'13, RTA'15, OOPSLA'16, FSCD'16, LMCS'17, …]

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

Matching Logic Models

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, ..., \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 ... s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

$$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$$

Matching Logic Proof System

13 Proof rules. Sound and complete

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2) \quad$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi] \quad$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Matching μ-Logic

- Adding support for recursion / induction

$$\varphi_s ::= \ldots \mid X{:}s \in \mathrm{SVAR}_s$$
$$\mid \mu X{:}s.\varphi_s \quad \text{if } \varphi_s \text{ is positive in } X{:}s$$

(PRE-FIXPOINT) $\varphi[\mu X.\,\varphi/X] \rightarrow \mu X.\varphi$

(KNASTER-TARSKI) $\dfrac{\varphi[\psi/X] \rightarrow \psi}{\mu X.\,\varphi \rightarrow \psi}$

Expressiveness

- Important logics for program reasoning can be framed as matching logic theories / notations
  - First-order logic
    - Equality, membership, definedness, partial functions
  - Lambda / mu calculi (least/largest fixed points)
  - Modal logics
  - Hoare logics
  - Dynamic logics
  - LTL, CTL, CTL*
  - Separation logic
  - Reachability logic
  - …

Expressiveness

- Important logics for program reasoning can be framed as matching logic theories / notations
  - First-order logic
    - Equality, membership, definedness, partial functions
  - Lambda / mu calculi (least/largest fixed points)
  - Modal logics
  - Hoare logics
  - Dynamic logics
  - LTL, CTL, CTL*
  - Separation logic
  - Reachability logic
  - …

Expressiveness

- Important logics for program reasoning can be framed as matching logic theories / notations
  - First-order logic
    - Equality, membership, definedness, partial functions
  - Lambda / mu calculi (least/largest fixed points)
  - Modal logics
  - Hoare logics
  - Dynamic logics
  - LTL, CTL, CTL*
  - Separation logic
  - Reachability logic
  - …

Reachability Logic (Semantics of K)

[LICS'13, RTA'14, RTA'15,OOPLSA'16]

- "Rewrite" rules over matching logic patterns:

$$\varphi \Rightarrow \varphi'$$

- Patterns generalize terms, so reachability rules capture rewriting, that is, operational semantics
- Reachability rules capture Hoare triples [FM'12]

$$\{Pre\}\,Code\{Post\} \quad \equiv \quad \widehat{Code} \wedge \widehat{Pre} \Rightarrow \epsilon \wedge \widehat{Post}$$

- Sound & relative complete proof system
  - Now proved as matching logic theorems

Reachability Logic (Semantics of K)

[LICS'13, RTA'14, RTA'15,OOPLSA'16]

- "Rewrite" rules over matching logic patterns:

$$\varphi \Longrightarrow \varphi'$$

Can be expressed in matching logic:
$\varphi \rightarrow \Diamond(\varphi')$ $\Diamond$ is "weak eventually"

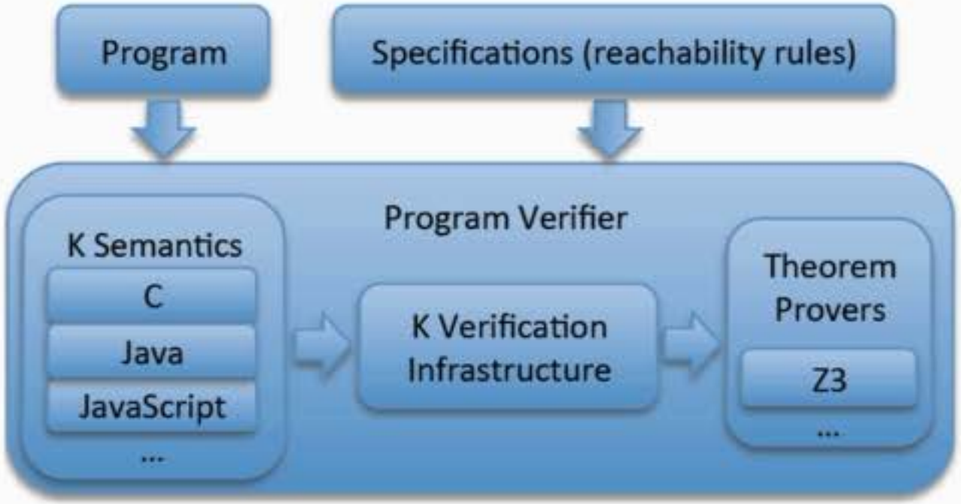
- Patterns generalize terms, so reachability rules capture rewriting, that is, operational semantics
- Reachability rules capture Hoare triples [FM'12]

$$\{Pre\}\,Code\,\{Post\} \quad \equiv \quad \widehat{Code} \wedge \widehat{Pre} \Rightarrow \epsilon \wedge \widehat{Post}$$

- Sound & relative complete proof system
  - Now proved as matching logic theorems

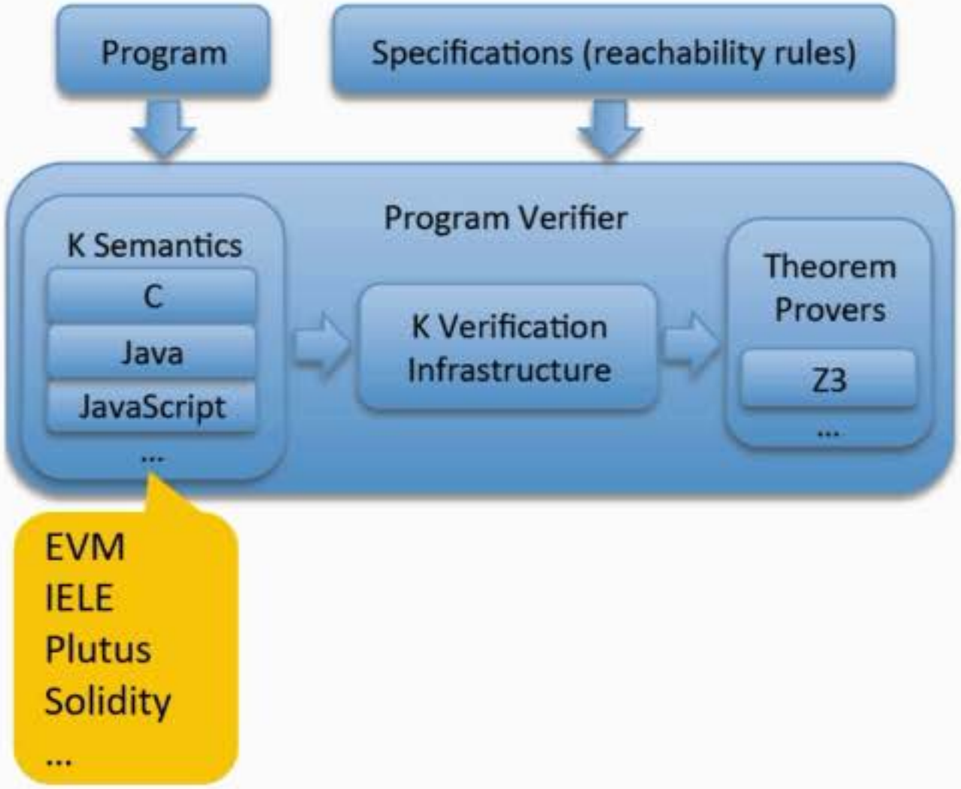
K = (Best Effort) Implementation of RL

- Reachability logic implemented in K, generically



K = (Best Effort) Implementation of RL

- Reachability logic implemented in K, generically



OK Performance [OOPLSA'16]

Time (seconds) spent on applying semantic steps (symbolic execution)

Time (seconds) spent on domain reasoning (matching logic + querying Z3)

Programs	KernelC				C				Java				JavaScript			
	Execution		Reasoning		Execution		Reasoning		Execution		Reasoning		Execution		Reasoning	
	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query
BST find	0.6	192	1.2	95	10.4	1,028	3.6	246	1.9	322	2.8	244	4.5	1,736	1.8	93
BST insert	0.8	336	2.9	160	23.0	2,481	7.2	414	4.1	691	4.5	342	5.4	3,394	2.8	158
BST delete	1.4	582	5.6	420	55.1	4,540	16.6	938	9.8	1,274	15.1	1,125	15.6	5,052	5.6	373
AVL find	0.6	192	1.2	95	9.9	1,028	3.1	214	2.2	322	2.7	244	4.5	1,736	1.9	93
AVL insert	6.2	1,980	42.1	1,133	210.7	12,616	70.6	1,865	42.4	3,753	62.8	2,146	102.5	26,977	32.5	1,221
AVL delete	9.5	2,933	45.4	1,758	514.8	26,003	118.9	3,883	122.2	8,144	149.4	4,866	184.3	38,591	55.3	2,233
RBT find	0.6	192	1.1	95	11.5	1,064	3.0	214	2.1	322	2.9	244	4.9	1,736	1.9	93
RBT insert	7.6	2,331	48.1	1,392	722.0	30,924	181.8	4,394	39.9	4,240	75.7	2,547	84.9	28,082	29.6	1,381
RBT delete	10.6	3,891	33.7	2,033	1593.8	50,389	308.3	15,429	95.8	8,312	75.4	4,460	144.2	51,356	39.4	2,009
Treap find	0.6	200	1.4	118	11.2	1,064	3.2	214	2.0	322	2.9	244	4.6	1,736	1.9	116
Treap insert	1.4	753	4.5	247	52.4	4,954	15.3	724	12.7	1,469	10.4	563	13.7	7,738	5.2	243
Treap delete	2.0	831	9.4	509	73.9	5,512	16.5	656	12.0	1,694	16.4	1,021	24.8	8,333	8.4	460
List reverse	0.4	142	0.3	21	6.6	815	4.8	76	1.5	222	2.6	46	5.0	1,162	0.5	20
List append	0.4	171	0.5	45	7.4	909	7.4	128	1.8	239	5.5	106	4.5	1,392	0.8	46
Bubble sort	0.9	391	26.8	190	28.4	2,401	38.0	357	3.4	589	35.4	345	5.6	2,688	25.7	145
Insertion sort	1.1	468	24.5	300	26.6	2,555	35.3	451	4.1	731	27.0	371	8.3	3,119	36.5	213
Quick sort	1.1	604	31.6	269	31.0	3,601	48.2	518	7.1	958	40.0	413	15.0	5,046	33.1	252
Merge sort	1.7	970	55.0	478	81.6	6,589	89.0	1,070	14.1	1,566	72.9	737	22.8	7,021	43.2	480
Total	47.7	17,159	335.2	9,358	3470.5	158,473	970.6	31,791	379.3	35,170	604.5	20,064	654.9	196,895	326.3	9,629

- Properties very challenging to verify automatically. We only found one such prover for C, based on a separation logic extension of VCC
  - Which takes 260 sec to verify AVL insert (ours takes 280 sec; see above)

OK Performance [OOPLSA'16]

Time (seconds) spent on applying semantic steps (symbolic execution)

Time (seconds) spent on domain reasoning (matching logic + querying Z3)

Programs	KernelC				C				Java				JavaScript			
	Execution		Reasoning		Execution		Reasoning		Execution		Reasoning		Execution		Reasoning	
	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query
BST find	0.6	192	1.2	95	10.4	1,028	3.6	246	1.9	322	2.8	244	4.5	1,736	1.8	93
BST insert	0.8	336	2.9	160	23.0	2,481	7.2	414	4.1	691	4.5	342	5.4	3,394	2.8	158
BST delete	1.4	582	5.6	420	55.1	4,540	16.6	938	9.8	1,274	15.1	1,125	15.6	5,052	5.6	373
AVL find	0.6	192	1.2	95	9.9	1,028	3.1	214	2.2	322	2.7	244	4.5	1,736	1.9	93
AVL insert	6.2	1,980	42.1	1,133	210.7	12,616	70.6	1,865	42.4	3,753	62.8	2,146	102.5	26,977	32.5	1,221
AVL delete	9.5	2,933	45.4	1,758	514.8	26,003	118.9	3,883	122.2	8,144	149.4	4,866	184.3	38,591	55.3	2,233
RBT find	0.6	192	1.1	95	11.5	1,064	3.0	214	2.1	322	2.9	244	4.9	1,736	1.9	93
RBT insert	7.6	2,331	48.1	1,392	722.0	30,924	181.8	4,394	39.9	4,240	75.7	2,547	84.9	28,082	29.6	1,381
RBT delete	10.6	3,891	33.7	2,033	1593.8	50,389	308.3	15,429	95.8	8,312	75.4	4,460	144.2	51,356	39.4	2,009
Treap find	0.6	200	1.4	118	11.2	1,064	3.2	214	2.0	322	2.9	244	4.6	1,736	1.9	116
Treap insert	1.4	753	4.5	247	52.4	4,954	15.3	724	12.7	1,469	10.4	563	13.7	7,738	5.2	243
Treap delete	2.0	831	9.4	509	73.9	5,512	16.5	656	12.0	1,694	16.4	1,021	24.8	8,333	8.4	460
List reverse	0.4	142	0.3	21	6.6	815	4.8	76	1.5	222	2.6	46	5.0	1,162	0.5	20
List append	0.4	171	0.5	45	7.4	909	7.4	128	1.8	239	5.5	106	4.5	1,392	0.8	46
Bubble sort	0.9	391	26.8	190	28.4	2,401	38.0	357	3.4	589	35.4	345	5.6	2,688	25.7	145
Insertion sort	1.1	468	24.5	300	26.6	2,555	35.3	451	4.1	731	27.0	371	8.3	3,119	36.5	213
Quick sort	1.1	604	31.6	269	31.0	3,601	48.2	518	7.1	958	40.0	413	15.0	5,046	33.1	252
Merge sort	1.7	970	55.0	478	81.6	6,589	89.0	1,070	14.1	1,566	72.9	737	22.8	7,021	43.2	480
Total	47.7	17,159	335.2	9,358	3470.5	158,473	970.6	31,791	379.3	35,170	604.5	20,064	654.9	196,895	326.3	9,629

- Properties very challenging to verify automatically. We only found one such prover for C, based on a separation logic extension of VCC
  - Which takes 260 sec to verify AVL insert (ours takes 280 sec; see above)

K for the Blockchain

OK Performance [OOPLSA'16]

Time (seconds) spent on applying semantic steps (symbolic execution)

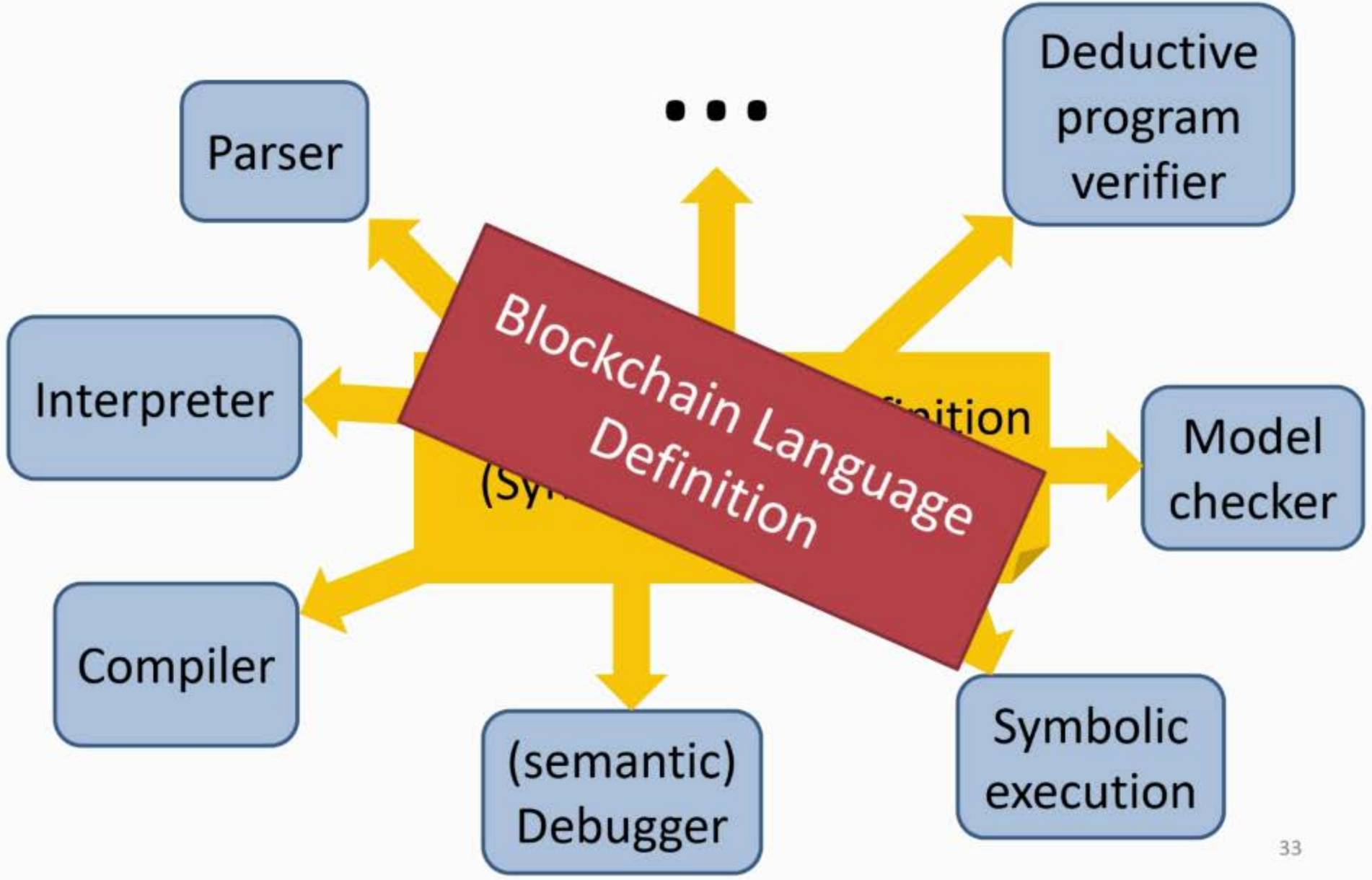
Time (seconds) spent on domain reasoning (matching logic + querying Z3)

Programs	KernelC				C				Java				JavaScript			
	Execution		Reasoning		Execution		Reasoning		Execution		Reasoning		Execution		Reasoning	
	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query
BST find	0.6	192	1.2	95	10.4	1,028	3.6	246	1.9	322	2.8	244	4.5	1,736	1.8	93
BST insert	0.8	336	2.9	160	23.0	2,481	7.2	414	4.1	691	4.5	342	5.4	3,394	2.8	158
BST delete	1.4	582	5.6	420	55.1	4,540	16.6	938	9.8	1,274	15.1	1,125	15.6	5,052	5.6	373
AVL find	0.6	192	1.2	95	9.9	1,028	3.1	214	2.2	322	2.7	244	4.5	1,736	1.9	93
AVL insert	6.2	1,980	42.1	1,133	210.7	12,616	70.6	1,865	42.4	3,753	62.8	2,146	102.5	26,977	32.5	1,221
AVL delete	9.5	2,933	45.4	1,758	514.8	26,003	118.9	3,883	122.2	8,144	149.4	4,866	184.3	38,591	55.3	2,233
RBT find	0.6	192	1.1	95	11.5	1,064	3.0	214	2.1	322	2.9	244	4.9	1,736	1.9	93
RBT insert	7.6	2,331	48.1	1,392	722.0	30,924	181.8	4,394	39.9	4,240	75.7	2,547	84.9	28,082	29.6	1,381
RBT delete	10.6	3,891	33.7	2,033	1593.8	50,389	308.3	15,429	95.8	8,312	75.4	4,460	144.2	51,356	39.4	2,009
Treap find	0.6	200	1.4	118	11.2	1,064	3.2	214	2.0	322	2.9	244	4.6	1,736	1.9	116
Treap insert	1.4	753	4.5	247	52.4	4,954	15.3	724	12.7	1,469	10.4	563	13.7	7,738	5.2	243
Treap delete	2.0	831	9.4	509	73.9	5,512	16.5	656	12.0	1,694	16.4	1,021	24.8	8,333	8.4	460
List reverse	0.4	142	0.3	21	6.6	815	4.8	76	1.5	222	2.6	46	5.0	1,162	0.5	20
List append	0.4	171	0.5	45	7.4	909	7.4	128	1.8	239	5.5	106	4.5	1,392	0.8	46
Bubble sort	0.9	391	26.8	190	28.4	2,401	38.0	357	3.4	589	35.4	345	5.6	2,688	25.7	145
Insertion sort	1.1	468	24.5	300	26.6	2,555	35.3	451	4.1	731	27.0	371	8.3	3,119	36.5	213
Quick sort	1.1	604	31.6	269	31.0	3,601	48.2	518	7.1	958	40.0	413	15.0	5,046	33.1	252
Merge sort	1.7	970	55.0	478	81.6	6,589	89.0	1,070	14.1	1,566	72.9	737	22.8	7,021	43.2	480
Total	47.7	17,159	335.2	9,358	3470.5	158,473	970.6	31,791	379.3	35,170	604.5	20,064	654.9	196,895	326.3	9,629

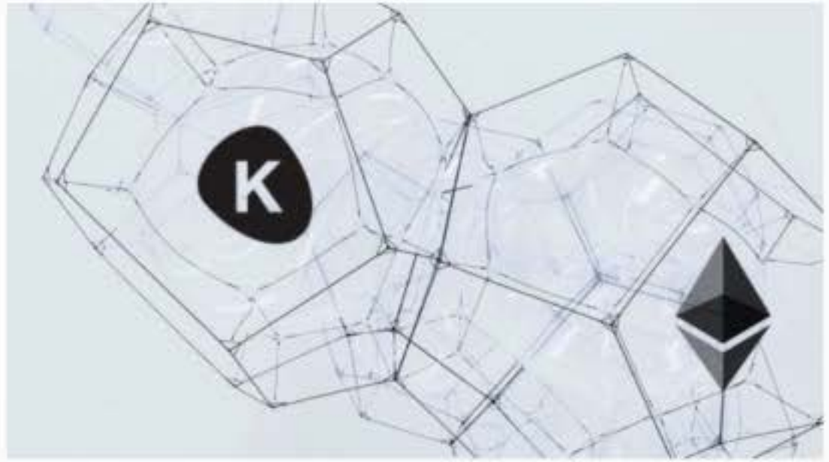
- Properties very challenging to verify automatically. We only found one such prover for C, based on a separation logic extension of VCC
  - Which takes 260 sec to verify AVL insert (ours takes 280 sec; see above)

K for the Blockchain

K framework vision



KEVM: Semantics of the Ethereum Virtual Machine (EVM) in K



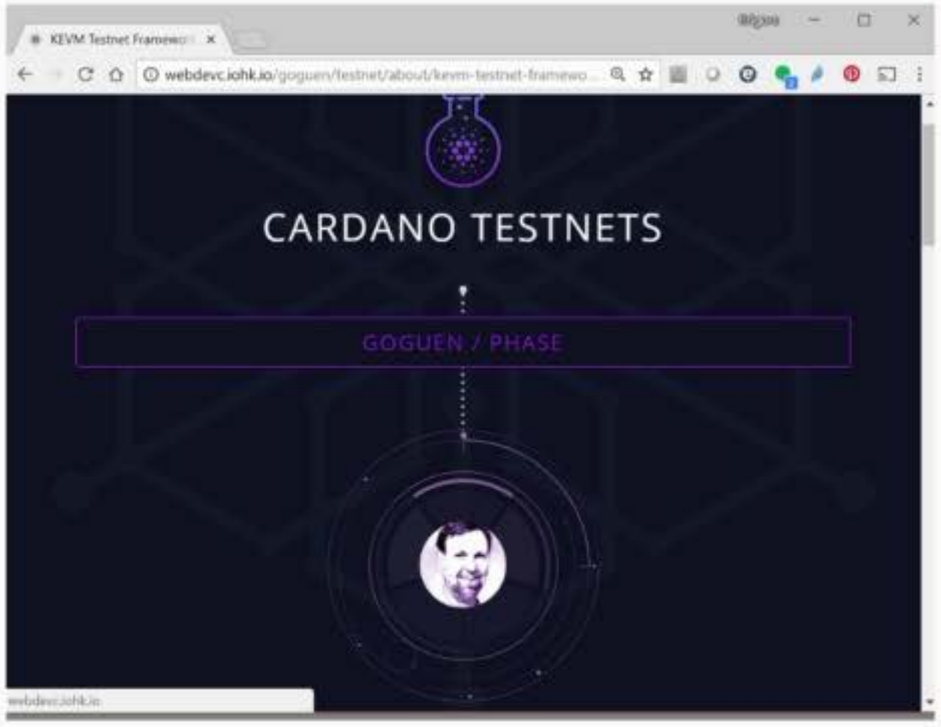
[CSL'18]

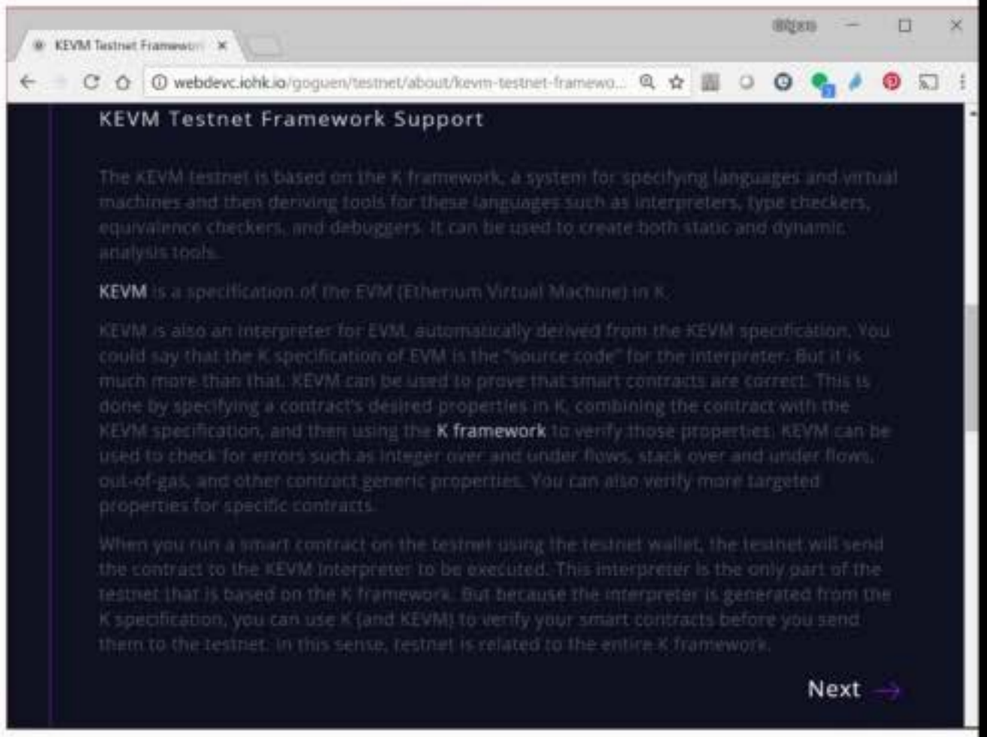
Complete semantics of EVM in K

- https://github.com/kframework/evm-semantics
- Passes 60,000+ tests of C++ reference implementation
- 8x (only!) slower than the C++ implementation
- Adoption by the Ethereum Foundation

What Can We Do with KEVM?

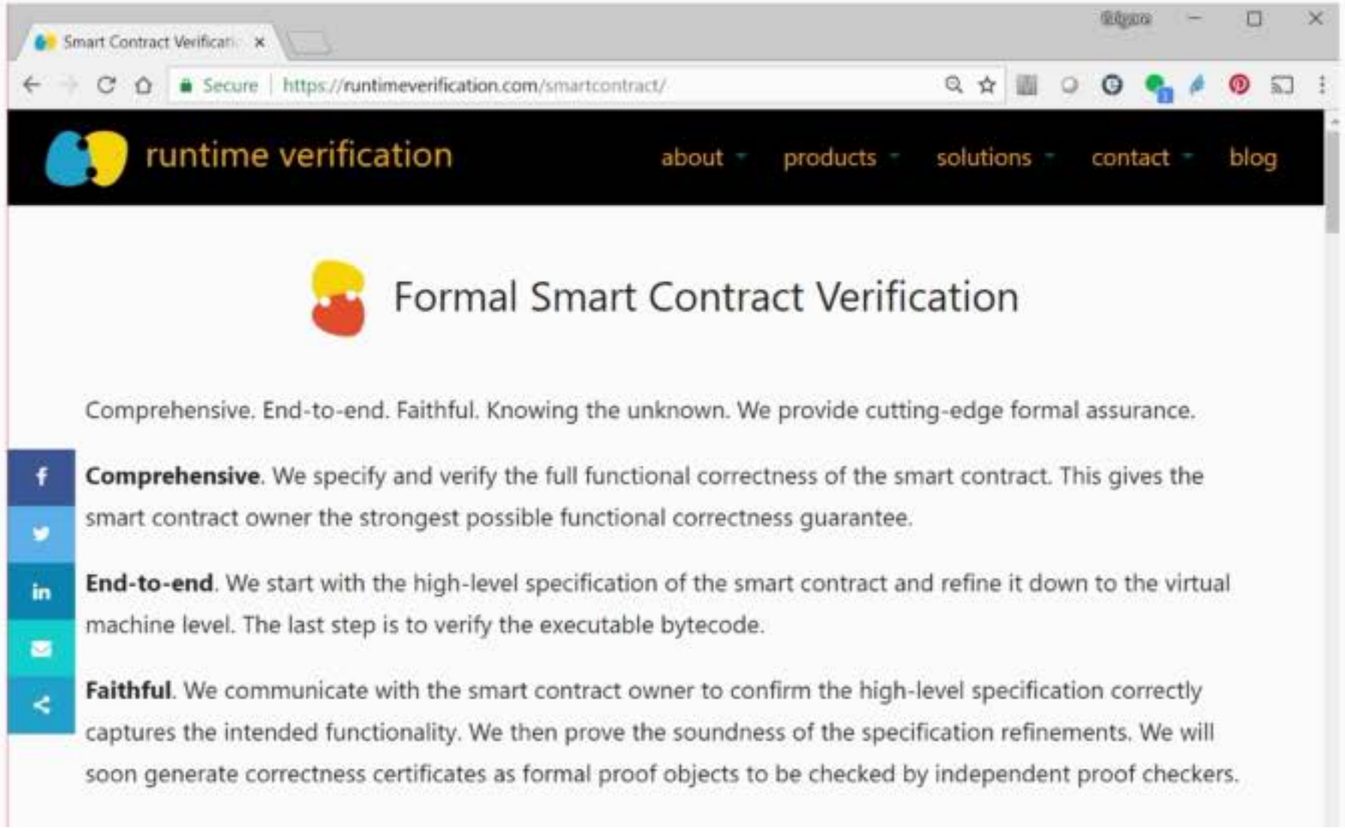
1) Generate and deploy correct-by-construction EVM client! IOHK has just done that, in collaboration with RV, as a Cardano testnet:





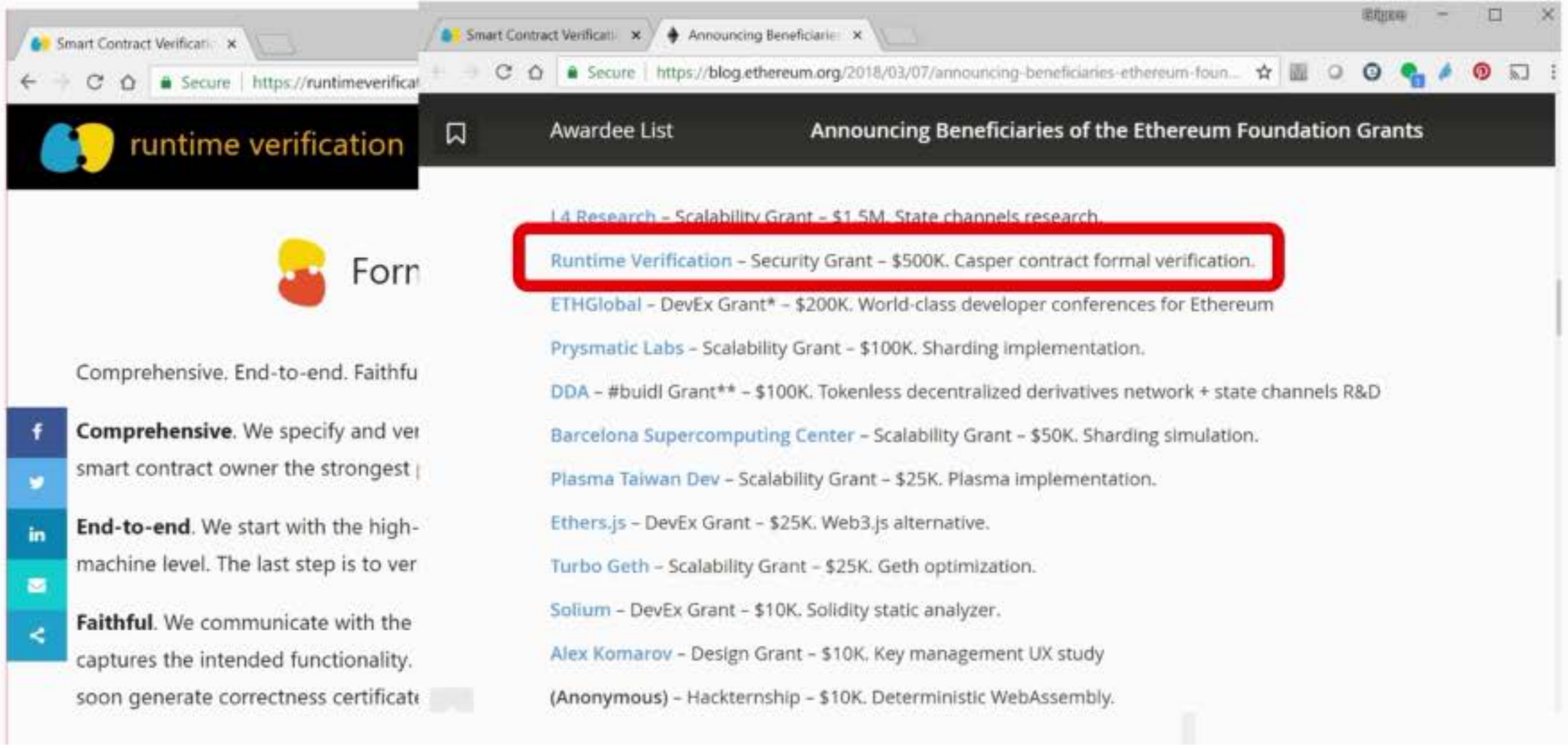
What Can We Do with KEVM?

2) Formally verify Ethereum smart contracts! RV is doing that, commercially. RV also won first Ethereum Security grant to verify Casper.



What Can We Do with KEVM?

2) Formally verify Ethereum smart contracts! RV is doing that, commercially. RV also won first Ethereum Security grant to verify Casper.

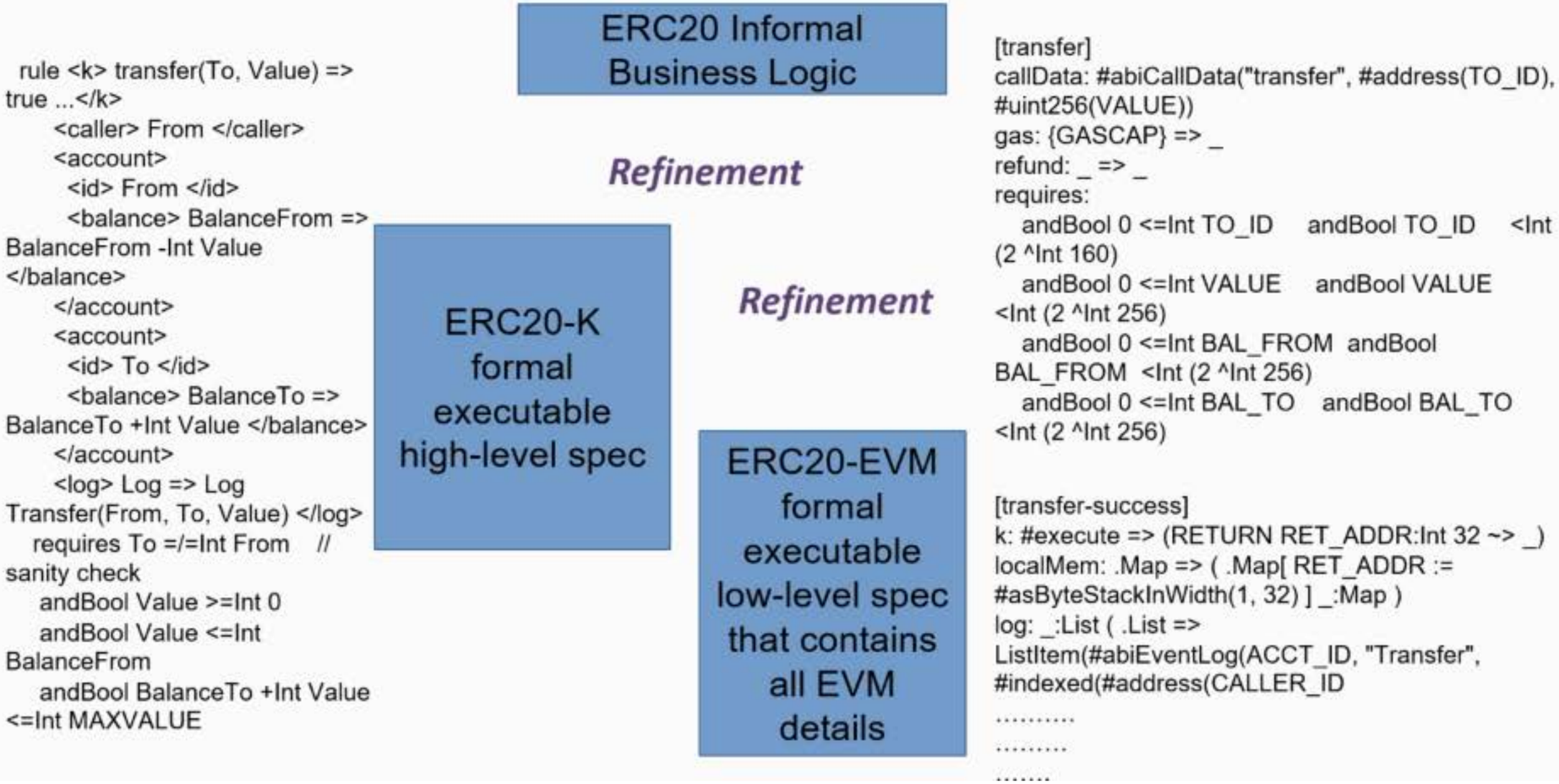


Smart contract verification workflow

Transfers `_value` amount of tokens to address `_to`, and MUST fire the `Transfer` event. The function SHOULD `throw` if the `_from` account balance does not have enough tokens to spend.

Note Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event.

```
function transfer(address _to, uint256 _value) returns (bool success)
```



EVM Not Human Readable (among other nuisences)

```
  PUSH(1, 0)  ; PUSH(1, 0)  ; MSTORE
; PUSH(1, 10) ; PUSH(1, 32) ; MSTORE
; JUMPDEST
; PUSH(1, 0) ; PUSH(1, 32) ; MLOAD ; GT
; ISZERO ; PUSH(1, 43) ; JUMPI
; PUSH(1, 32) ; MLOAD ; PUSH(1, 0)  ; MLOAD ; ADD ; PUSH(1, 0)  ; MSTORE
; PUSH(1, 1)           ; PUSH(1, 32) ; MLOAD ; SUB ; PUSH(1, 32) ; MSTORE
; PUSH(1, 10) ; JUMP
; JUMPDEST
; PUSH(1, 0) ; MLOAD ; PUSH(1, 0) ; SSTORE
```

EVM Not Human Readable (among other nuise

If it must be low-level, then I prefer this:

```
  PUSH(1, 0)   ; PUS
; PUSH(1, 10) ; PUS
; JUMPDEST
; PUSH(1, 0) ; PUSH
; ISZERO ; PUSH(1,
; PUSH(1, 32) ; MLO
; PUSH(1, 1)
; PUSH(1, 10) ; JUM
; JUMPDEST
; PUSH(1, 0) ; MLOA
```

```
define public @sum(%n) {
  %result = 0
condition:
  %cond = cmp le %n, 0
  br %cond, after_loop
  %result = add %result, %n
  %n = sub %n, 1
  br condition
after_loop:
  ret %result
}
```

```
USH(1, 0)   ; MSTORE
USH(1, 32) ; MSTORE
```

A New Virtual Machine (and Language) for the Blockchain

- Incorporates learnings from defining KEVM and from using it to verify smart contracts
- Register-based machine, like LLVM; unbounded*
- IELE was designed and implemented using formal methods and semantics from scratch!
- Until IELE, only existing or toy languages have been given formal semantics in K
  - Not as exciting as designing new languages
  - We should use semantics as an intrinsic, active language design principle, not post-mortem

A New Virtual Machine (and Language) for the Blockchain

- Incorpo… M and from…
- K… ded*
- IELE w… mal m…
- Until IELE, … have been give… ma… ant… K
  - Not as exciting as … gning n… anguages
  - We should use se… ntics as an intrinsic, active language design p… nciple, not post-mortem

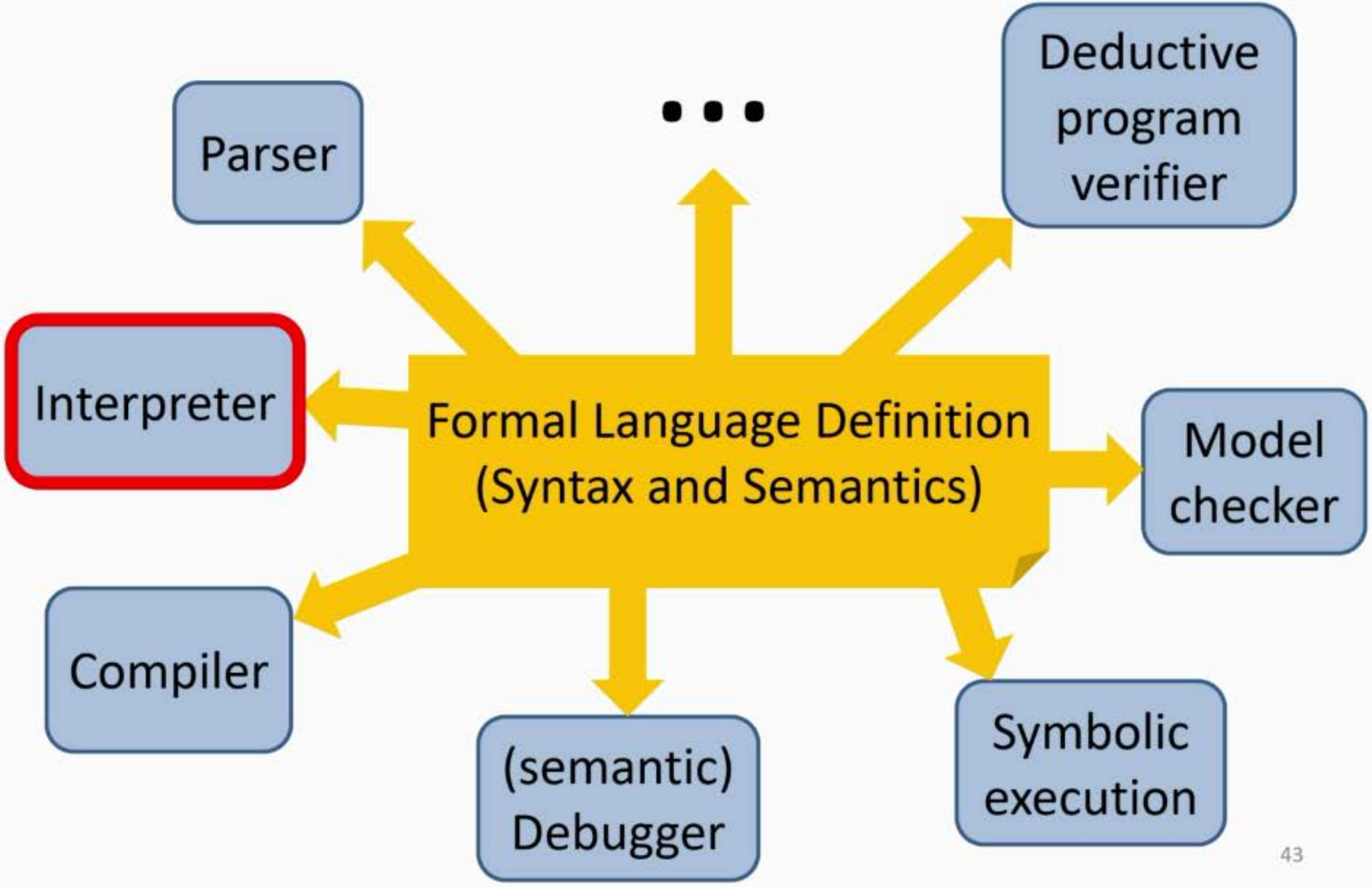
IELE testnet (with IOHK):

- End of July'18
- With tool ecosystem

K Semantics of Other Blockchain Languages

- WASM (web assembly) – in progress, in collaboration with the Ethereum Foundation
- Solidity – in progress, collaboration between RV and Sun Jun's group in Singapore
- Plutus (functional) – in progress, by RV following Phil Wadler's (@IOHK) design of the language
- Vyper – in progress, by RV in collaboration with the Ethereum Foundation
- …

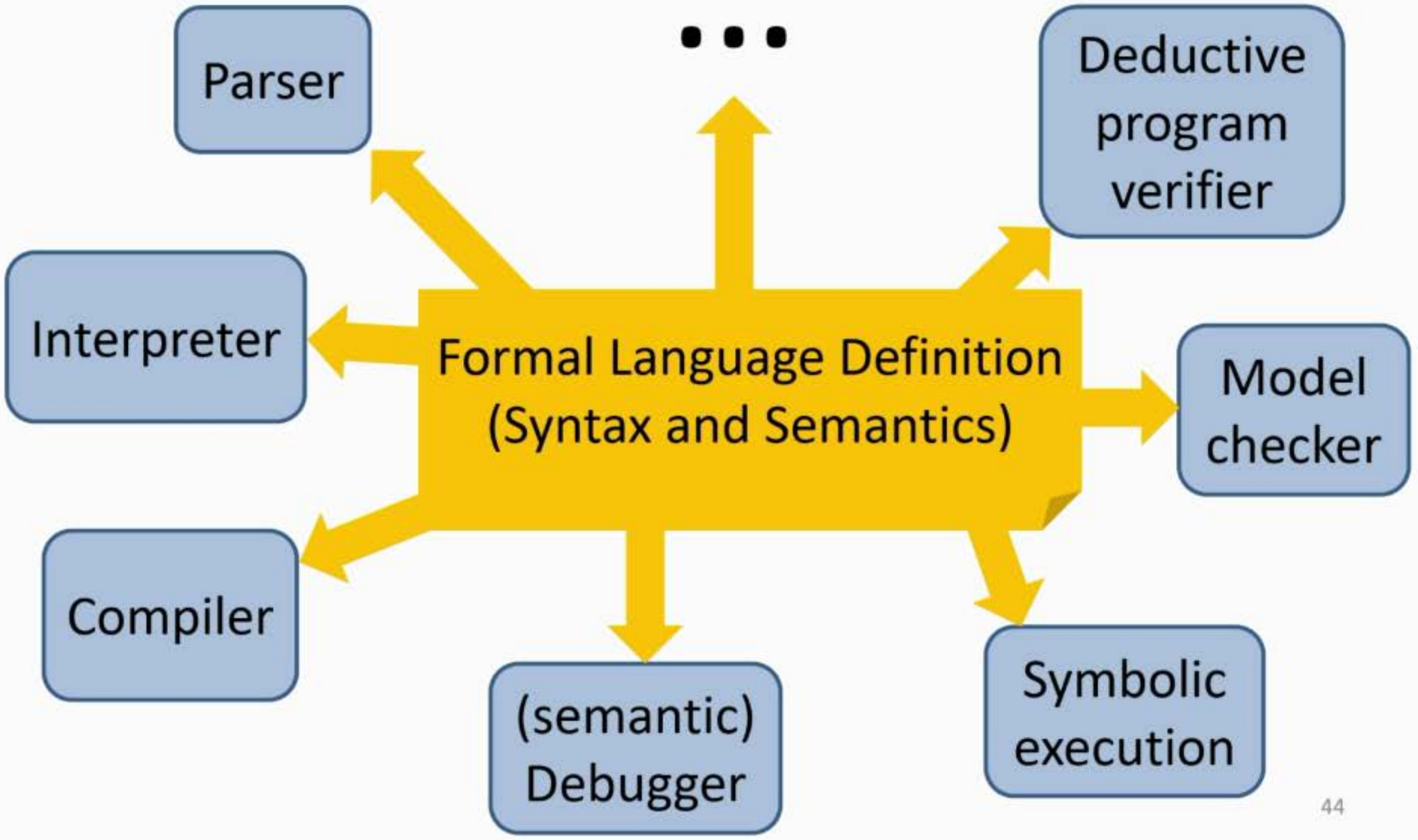
1. Fast LLVM (and IELE) Backend for K



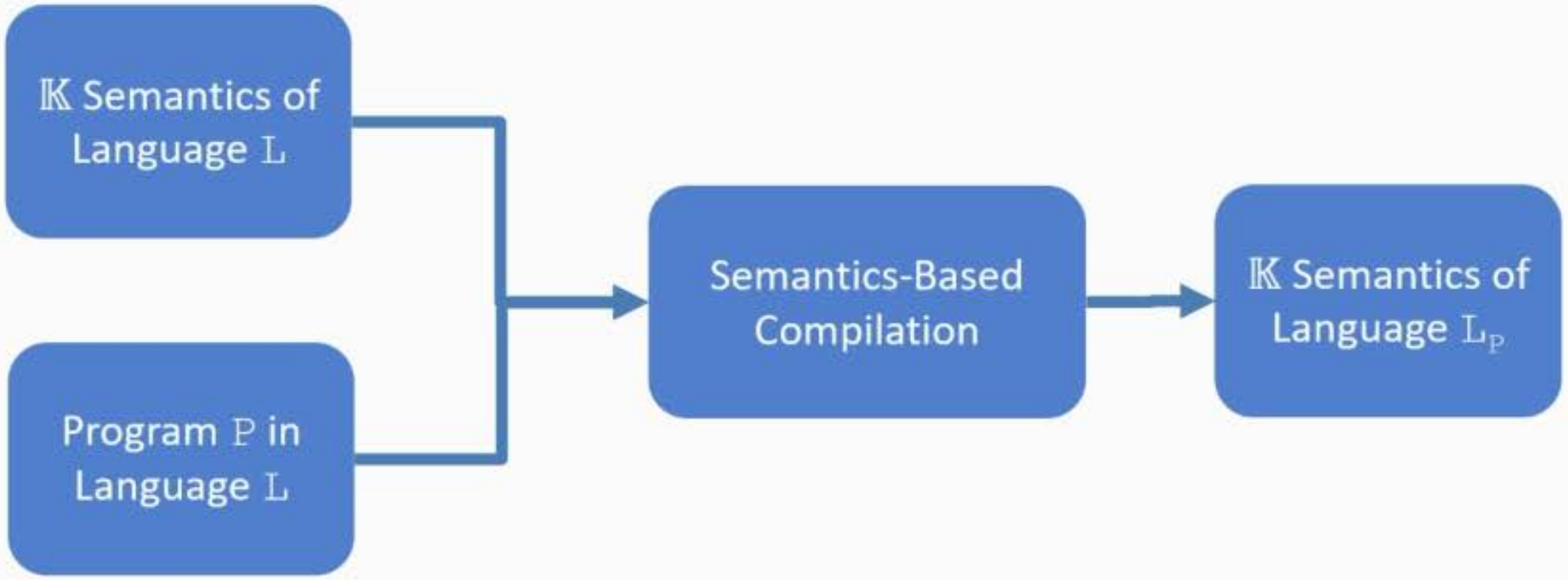
1. Fast LLVM [illegible] IELE) [illegible] for K

- Current OCAML backend of K:
  - Fast enough to power RV-Match product and the KEVM and IELE VMs in testnets
  - But still one or two orders of magnitude slower than hand-crafted interpreters
- LLVM backend for K under development:
  - Take advantage of LLVM's optimizations / pipeline
  - Expected to compete with hand-written interpreters

2. Semantics-Based Compilation



Semantics-Based Compilation (SBC)



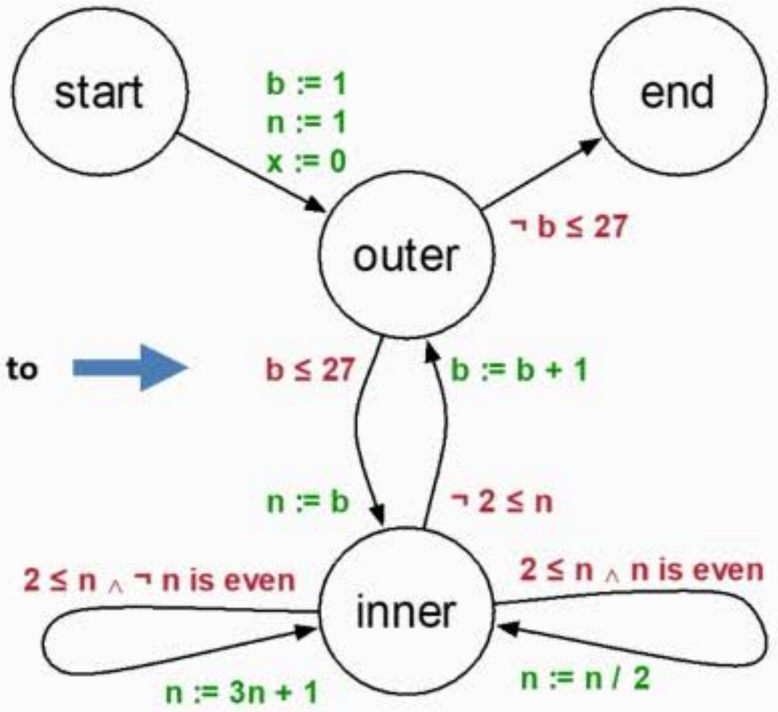
Goals

- Execution of P in L equivalent to executing L_P in a start configuration
- L_P should be "as simple as possible", only capturing exactly the dynamics of L necessary to execute program P

Semantics-Based Compilation (SBC) Experiments with Early Prototype

```
// start
int b , n , x ;
b = 1 ; n = 1 ; x = 0 ;

// outer
while (b <= 27) {
  n = b ;

  // inner
  while (2 <= n) {
    if (n <= ((n / 2) * 2))
{
      n = n / 2 ;
    } else {
      n = (3 * n) + 1 ;
    }
    x = x + 1 ;
  }
  b = b + 1 ;
}
// end
```

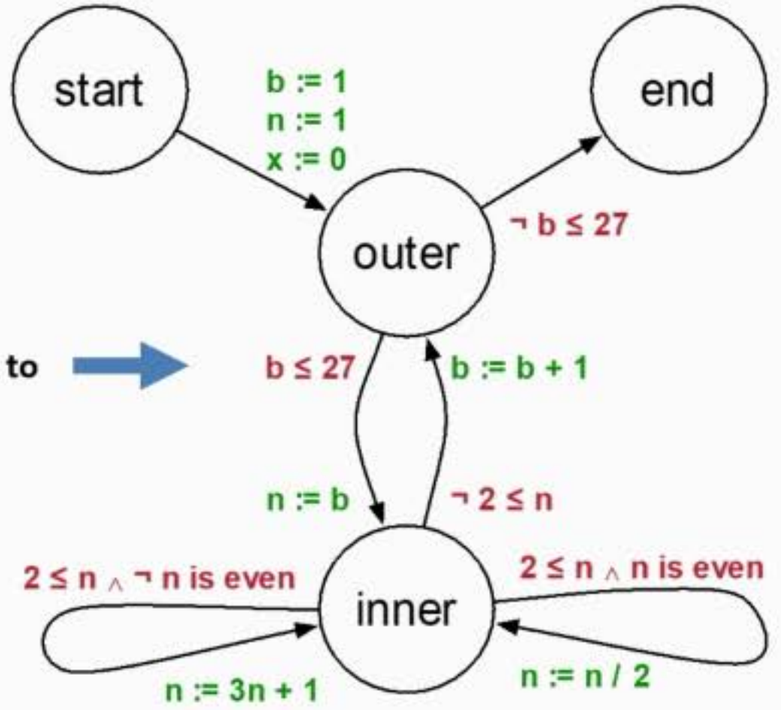
compiles to



Semantics-Based Compilation (SBC) Experiments with Early Prototype

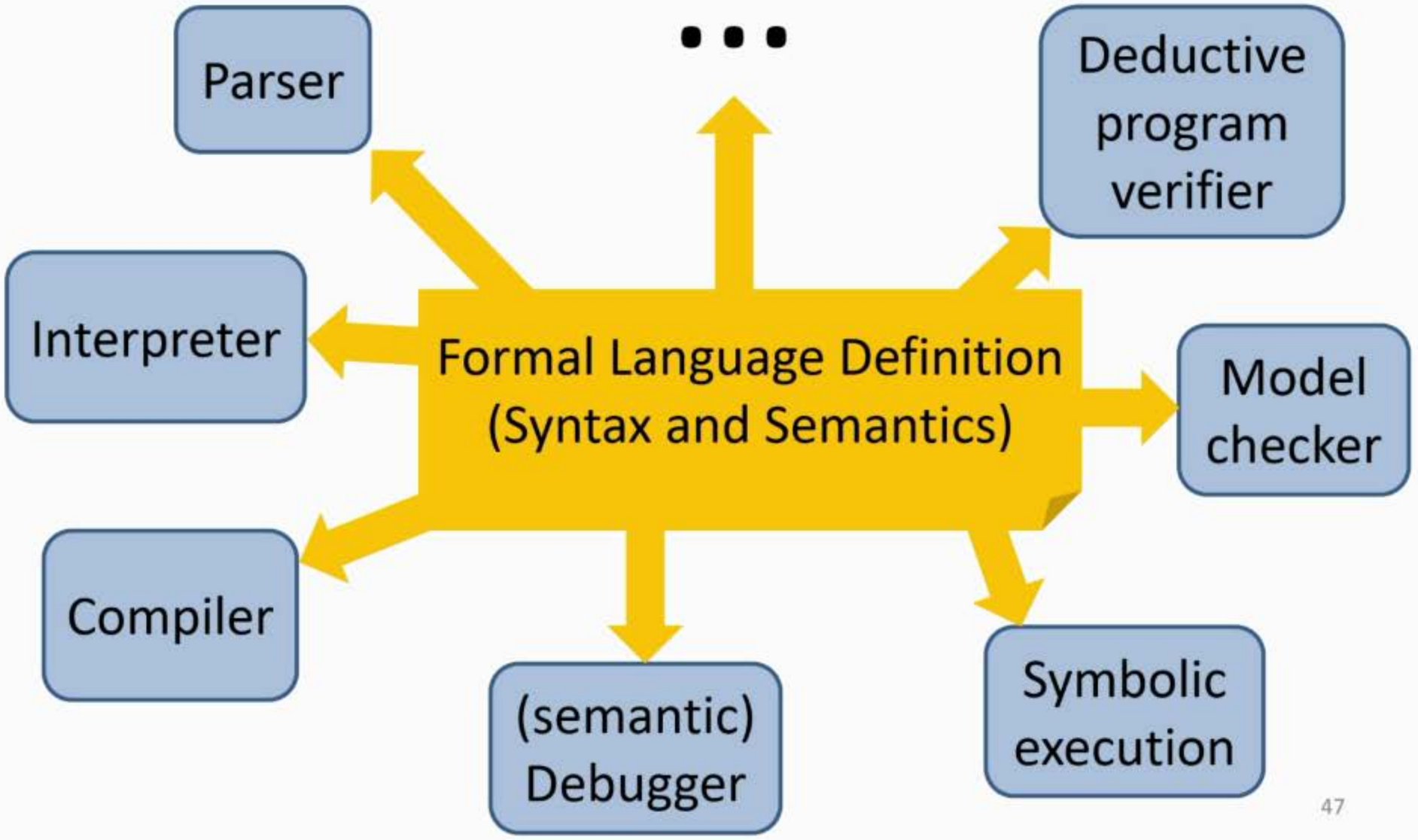
```
// start
int b , n , x ;
b = 1 ; n = 1 ; x = 0 ;

// outer
while (b <= 27) {
  n = b ;

  // inner
  while (2 <= n) {
    if (n <= ((n / 2) * 2))
{
      n = n / 2 ;
    } else {
      n = (3 * n) + 1 ;
    }
    x = x + 1 ;
  }
  b = b + 1 ;
}
// end
```

compiles to



Program	Original (s)	Compiled (s)	Speedup
sum.imp	70.6	7.3	9.7
collatz.imp	34.5	2.7	12.8
collatz-all.imp	77.4	5.7	13.6
krazy-loop.imp	67.6	3.3	20.5

3. Proof Object Generation



Formal Methods for Blockchain

Research Center

ILLINOIS
Computer Science

Formal Methods for Blockchain

Research Center

ILLINOIS
Computer Science

- Vision:

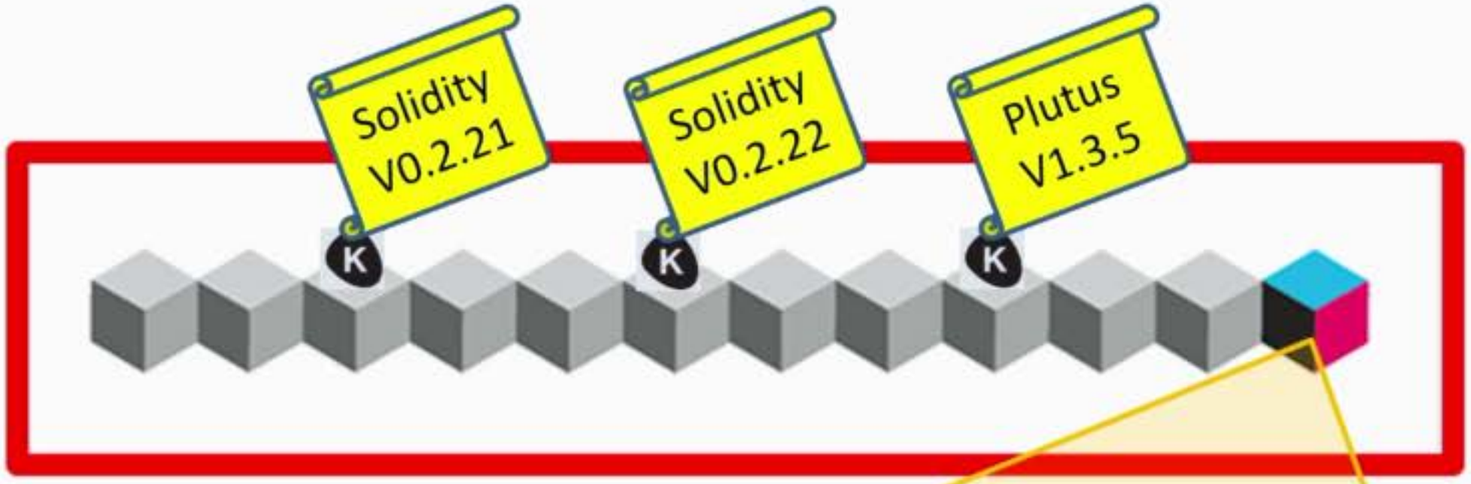
Develop end-to-end technology for future blockchains

- Objective

Provably correct and secure blockchain programs

K – A Universal Blockchain Language

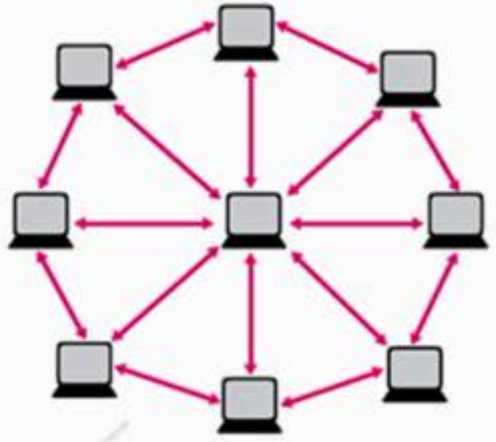
- We want to be able to write (provably correct) smart contracts in *any* programming language!



1. Write contract P in any language, say L (unique address)
2. SBC[L] your P into L_P ; verify P (or L_P) with K prover
3. Generate binary from L_P using LLVM (?) backend

K – A Universal Blockchain Language

- We want to be able to write (provably correct) smart contracts in *any* programming language!



Each node runs a K VM client

Conclusion: It Can Be Done!