The logo for 'The K Approach', featuring a large white 'K' inside a black oval, which is set against a yellow background with a diagonal split.

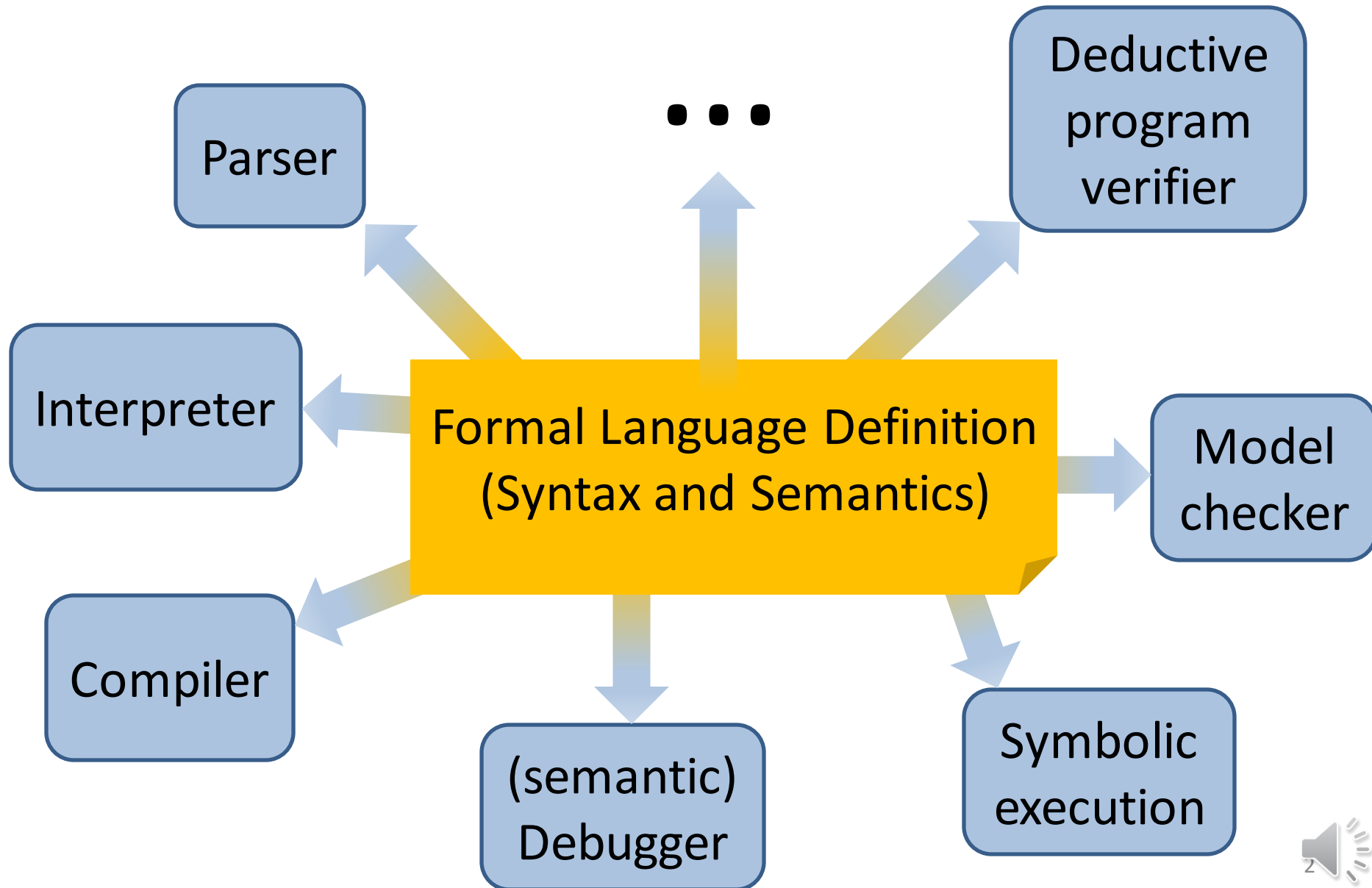
K

The K Approach

Presenter: Sandeep Dasgupta

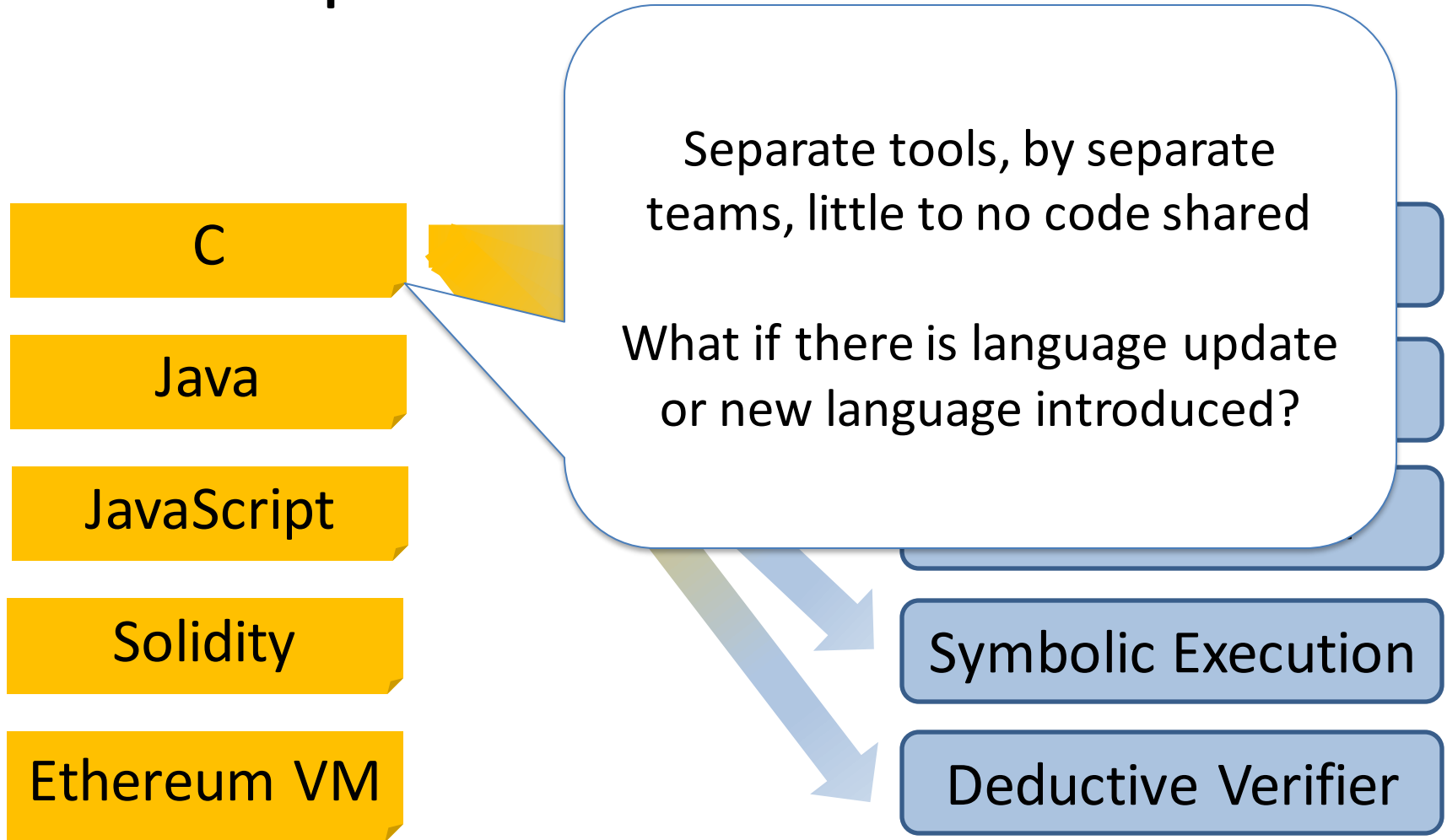
University of Illinois at Urbana

Ideal Language Framework Vision



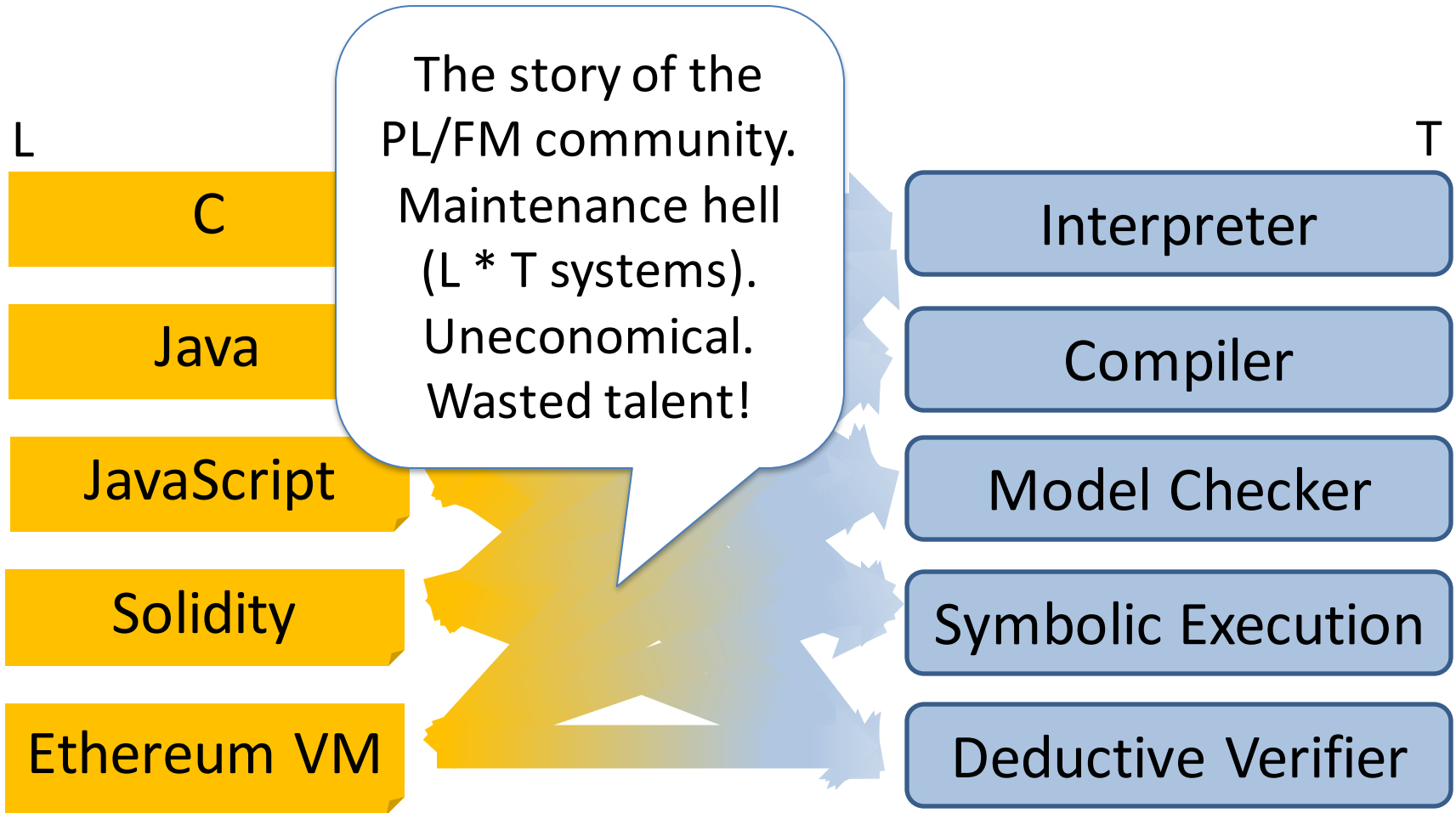
Current State-of-the-Art

- Sharp Contrast to Ideal Vision -

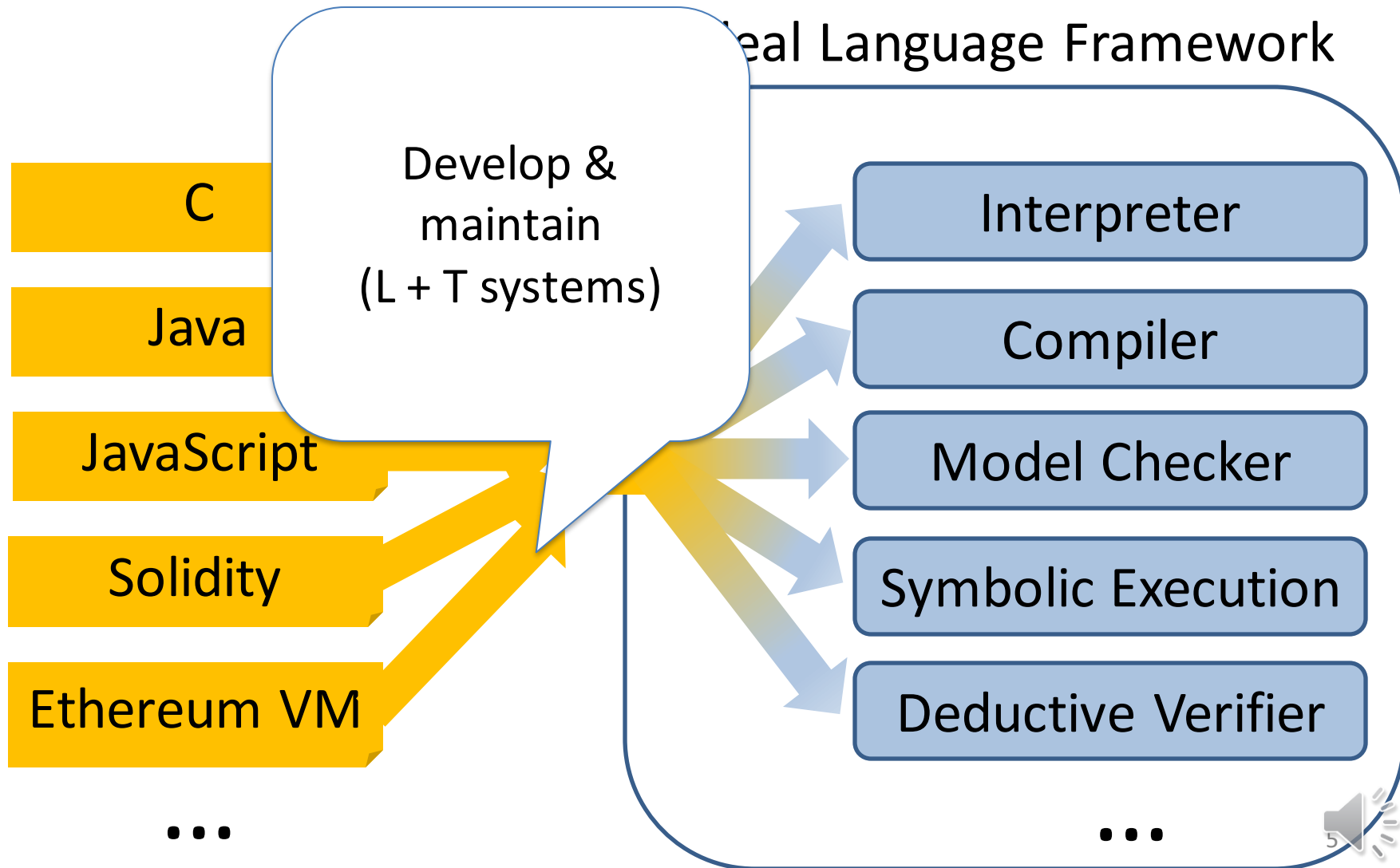


Current State-of-the-Art

- Sharp Contrast to Ideal Vision -



How It Should Be



Our Attempt: the K Framework

<http://kframework.org>

- K Framework started with the “ideal framework” vision
 - Design philosophy: Reusability & Scalability
 - Not just an academic tool, but adapted in industry
- K framework initially *engineered*: keep advantages and avoid limitations of various semantic styles
 - Then theory came
 - Engineering effort is largely focused in providing better experience to language developers
- We tried various semantic styles, for >15y and >100 top-tier conference and journal papers
 - Big emphasis on modularity and expressiveness
 - Simplified language design experience



Complete K Definition of KernelC

```

MODULE KERNELC-SYNTAX
IMPORTS K-LATEX+PL-ID+PL-INT
SYNTAX Exp ::= Exp * Exp [strict]
          DeclId
          Id
          Int
          Exp * Exp [strict]
          Exp ++
          Exp == Exp [strict]
          Exp != Exp [strict]
          Exp < Exp [strict]
          Exp <= Exp [strict]
          Exp % Exp [strict]
          ! Exp
          Exp && Exp
          Exp ? Exp : Exp
          Exp [] Exp
          printf("%d", Exp) [strict]
          scanf("%d", &Exp) [strict]
          scanf("%d", Exp) [strict]
          NULL
          PointerId
          (int*)malloc(Exp * sizeof(int)) [strict]
          free(Exp) [strict]
          * Exp [strict]
          Exp [ Exp ]
          Exp = Exp [strict2]
          Id ( List(Exp) ) [strict2]
          Id ()
          random()
          srand(Exp) [strict]
SYNTAX Stmt ::= Exp ; [strict]
          {}
          { StmtList }
          if(Exp) Stmt
          if(Exp) Stmt else Stmt [strict1]
          while(Exp) Stmt
          return Exp ; [strict]
          DeclId List(DeclId) { StmtList }
          #include< StmtList >
SYNTAX StmtList ::= StmtList Stmt
          {}
SYNTAX Pgm ::= StmtList
SYNTAX Id ::= main
SYNTAX PointerId ::= * PointerId [dimo]
          | Id
SYNTAX DeclId ::= int Exp
          | void PointerId
SYNTAX StmtList ::= stdio.h
          | stdlib.h
SYNTAX List(Bottom) ::= List(Bottom) , List(Bottom) [assoc hybrid id: () strict]
          | {}
          | Bottom
SYNTAX List(PointerId) ::= List(PointerId) , List(PointerId) [dimo]
          | List(Bottom)
          | PointerId
SYNTAX List(DeclId) ::= List(DeclId) , List(DeclId) [dimo]
          | DeclId
          | List(Bottom)
SYNTAX List(Exp) ::= List(Exp) , List(Exp) [dimo]
          | Exp
          | List(DeclId)
          | List(PointerId)
END MODULE

MODULE KERNELC-DESUGARED-SYNTAX
IMPORTS K-LATEX
IMPORTS KERNELC-SYNTAX
MACRO ! E = E ? 0 : 1

MACRO E1 && E2 = E1 ? E2 : 0

MACRO E1 [] E2 = E1 ? 1 : E2

MACRO if( E ) St = if( E ) St else {}

MACRO NULL = 0

MACRO I () = I ( {} )

MACRO int * PointerId = int PointerId

MACRO #include< Stmts > = Stmts

MACRO E1 [ E2 ] = * E1 * E2

MACRO scanf("%d", & * E) = scanf("%d", E)

MACRO int * PointerId = E = int PointerId = E

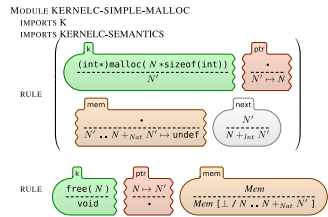
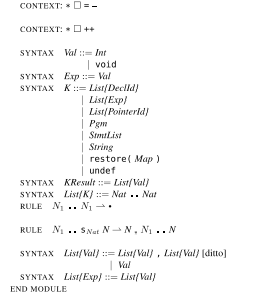
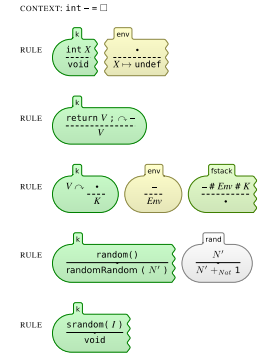
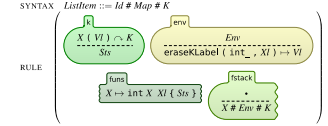
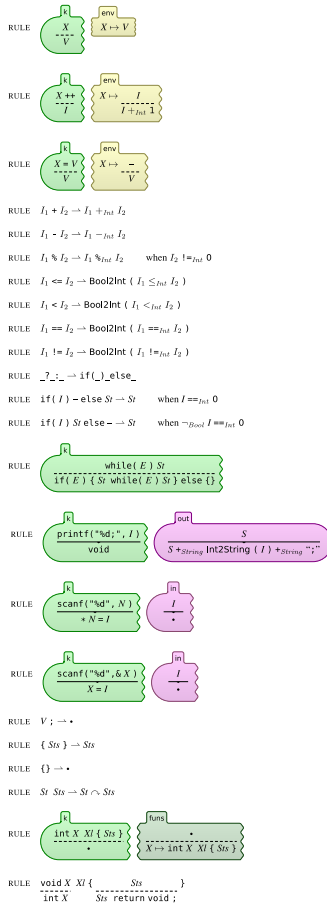
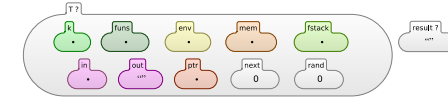
MACRO int X = E ; = int X ; X = E ;

MACRO stdio.h = {}

MACRO stdlib.h = {}
END MODULE

```

MODULE KERNELC-SEMANTICS
IMPORTS K-SHARED
IMPORTS K-KERNELC-DESUGARED-SYNTAX+PL-CONVERSION+PL-RANDOM
CONFIGURATION:



END MODULE



Complete K Definition of KernelC

```

MODULE KERNELC-SYNTAX
IMPORTS K-LATEX+PL-ID+PL-INT
SYNTAX Exp ::= Exp * Exp [strict]
  DeclId
  Id
  Int
  Exp * Exp [strict]
  Exp ++
  Exp == Exp [strict]
  Exp != Exp [strict]
  Exp < Exp [strict]
  Exp <= Exp [strict]
  Exp % Exp [strict]
  ! Exp
  Exp && Exp
  Exp ? Exp : Exp
  Exp || Exp
  printf("%d", Exp) [strict]
  scanf("%d", &Exp) [strict]
  scanf("%d", Exp) [strict]
  NULL
  PointerId
  (int*)malloc(Exp * sizeof(int)) [strict]
  free(Exp) [strict]
  * Exp [strict]
  Exp { Exp }
  Exp * Exp [strict(2)]
  Id { List(Exp) } [strict(2)]
  Id { }
  random()
  srandom(Exp) [strict]
SYNTAX Stmt ::= Exp ; [strict]
  { }
  { StmtList }
  if(Exp) Stmt
  if(Exp) Stmt else Stmt [strict(1)]
  while(Exp) Stmt
  return Exp ; [strict]
  DeclId List(DeclId) { StmtList }
  #include< StmtList >
SYNTAX StmtList ::= StmtList StmtList
  StmtList
SYNTAX Pgm ::= StmtList
SYNTAX Id ::= main
SYNTAX PointerId ::= * PointerId [dimo]
  Id
SYNTAX DeclId ::= int Exp
  void PointerId
SYNTAX StmtList ::= stdio.h
  stdlib.h
SYNTAX List(Bottom) ::= List(Bottom) , List(Bottom) [assoc hybrid id: () strict]
  { }
  Bottom
SYNTAX List(PointerId) ::= List(PointerId) , List(PointerId) [dimo]
  List(Bottom)
  PointerId
SYNTAX List(DeclId) ::= List(DeclId) , List(DeclId) [dimo]
  DeclId
  List(Bottom)
SYNTAX List(Exp) ::= List(Exp) , List(Exp) [dimo]
  Exp
  List(DeclId)
  List(PointerId)
END MODULE

MODULE KERNELC-DESUGARED-SYNTAX
IMPORTS K-LATEX
IMPORTS KERNELC-SYNTAX
MACRO ! E = E ? 0 : 1

MACRO E1 && E2 = E1 ? E2 : 0

MACRO E1 || E2 = E1 ? 1 : E2

MACRO if( E ) St = if( E ) St else { }

MACRO NULL = 0

MACRO f ( ) = f ( { } )

MACRO int * PointerId = int PointerId

MACRO #include< Stmts > = Stmts

MACRO E1 [ E2 ] = * E1 * E2

MACRO scanf("%d",&* E) = scanf("%d", E)

MACRO int * PointerId = E = int PointerId = E

MACRO int X = E ; = int X ; X = E ;

MACRO stdio.h = { }

MACRO stdlib.h = { }

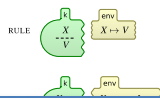
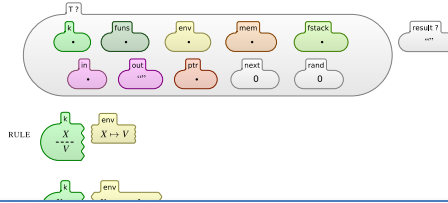
END MODULE

```

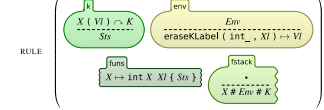
Syntax declared using annotated BNF

SYNTAX $Exp ::= \dots$
 $| Exp = Exp [\text{strict}(2)]$

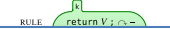
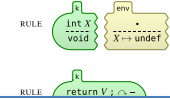
MODULE KERNELC-SEMANTICS
 IMPORTS K-SHARED
 IMPORTS K- KERNELC-DESUGARED-SYNTAX+PL-CONVERSION+PL-RANDOM
 CONFIGURATION:



SYNTAX $ListMem ::= Id \# Map \# K$



CONTEXT: $int = \square$



SYNTAX $Ptr ::= List(Val)$

SYNTAX $List(K) ::= Nat \# Nat$

RULE $N_1 \# N_2 \rightarrow \bullet$

RULE $N_1 \# N_2 \# N_3 \rightarrow N_1 \# N_2 \# N_3$

SYNTAX $List(Val) ::= List(Val) , List(Val) [dimo]$

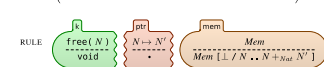
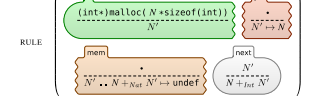
SYNTAX $List(Exp) ::= List(Exp)$

END MODULE

MODULE KERNELC-SIMPLE-MALLOC

IMPORTS K

IMPORTS KERNELC-SEMANTICS



END MODULE



Complete K Definition of KernelC

```

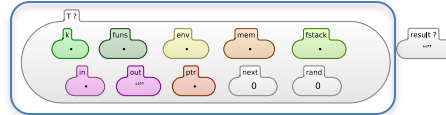
MODULE KERNELC-SYNTAX
IMPORTS K-LATEX+PL-ID+PL-INT
SYNTAX Exp ::= Exp + Exp [strict]
DeclId
Id
Int
Exp - Exp [strict]
Exp ++
Exp == Exp [strict]
Exp != Exp [strict]
Exp < Exp [strict]
Exp <= Exp [strict]
Exp % Exp [strict]
! Exp
Exp && Exp
Exp ? Exp : Exp
Exp [] Exp
printf("%d", Exp) [strict]
scanf("%d", & Exp) [strict]
scanf("%d", Exp) [strict]
NULL
PointerId
(int*)
f

```

```

MODULE KERNELC-SEMANTICS
IMPORTS K-SHARED
IMPORTS K+KERNELC-DESUGARED-SYNTAX+PL-CONVERSION+PL-RANDOM
CONFIGURATION:

```

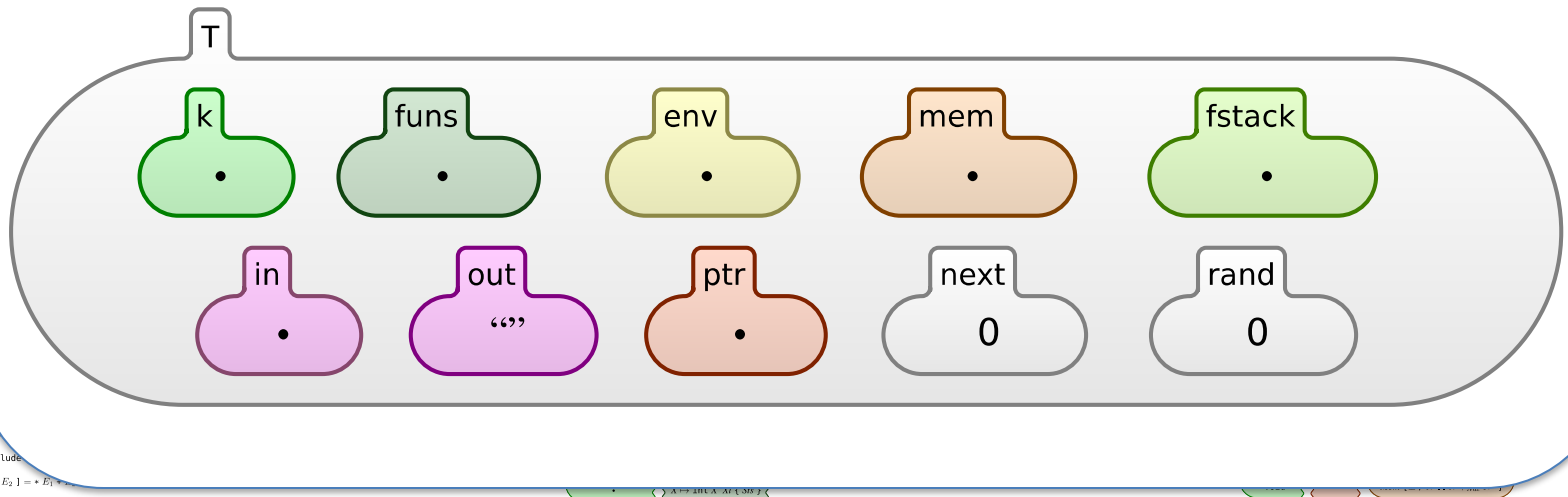


```

SYNTAX ListMem ::= Id # Map # K
RULE
  (
    X (VI) ~ K
    Env
    eraseLabel (int_, X) ~ VI
  )
  (
    X ~ int X M { Sts }
    fstack
    X # Env # K
  )
CONTEXT: int = 0
RULE
  int X
  void
  X ~ undef
RULE
  return V;

```

Configuration given as a nested cell structure.
Leaves can be sets, multisets, lists, maps, or syntax



```

END MODULE
MODULE K
IMPORTS
IMPORTS
MACRO
MACRO
MACRO
MACRO
MACRO /
MACRO int
MACRO #include
MACRO E1 [ E2 ] = E1
MACRO scanf("%d", & * E) = scanf("%d", E)
MACRO int * PointerId = E = int PointerId = E
MACRO int X = E; = int X; X = E;
MACRO stdio.h = {}
MACRO stdlib.h = {}
END MODULE

```

```

RULE void X XI {
  int X
  Sts return void;
}

```

END MODULE

Complete K Definition of KernelC

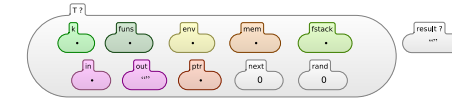
```

MODULE KERNELC-SYNTAX
IMPORTS K-LATEX+PL-ID+PL-INT
SYNTAX Exp ::= Exp * Exp [strict]
          DeclId
          Id
          Int
          Exp * Exp [strict]
          Exp ++
          Exp == Exp [strict]
          Exp != Exp [strict]
          Exp < Exp [strict]
          Exp <= Exp [strict]
          Exp % Exp [strict]
          ! Exp
          Exp && Exp
          Exp ? Exp : Exp
          Exp [] Exp
          printf("%d", Exp) [strict]
          scanf("%d", &Exp) [strict]
          scanf("%d", Exp) [strict]
          NULL
          PointerId
          (int*)malloc(Exp * sizeof(int)) [strict]
          free(Exp) [strict]
          * Exp [strict]
          Exp [ Exp ]
          Exp = Exp [strict2]
          Id ( List(Exp) ) [strict2]
          Id ()
          random()
          srand(Exp) [strict]
SYNTAX Stmt ::= Exp ; [strict]
          {}
          { StmtList }
          if( Exp ) Stmt
          if( Exp ) Stmt else Stmt [strict1]
          while( Exp ) Stmt
          return Exp ; [strict]
          DeclId List(DeclId) { StmtList }
          #include< StmtList >
SYNTAX StmtList ::= StmtList Stmt
          {}
SYNTAX Pgm ::= StmtList
SYNTAX Id ::= main
SYNTAX PointerId ::= PointerId [dimo]
          | Id
SYNTAX DeclId ::= int Exp
          | void PointerId
SYNTAX StmtList ::= stdio.h
          | stdlib.h
SYNTAX List(Bottom) ::= List(Bottom) , List(Bottom) [assoc hybrid id: () strict]
          | {}
          | Bottom
SYNTAX List(PointerId) ::= List(PointerId) , List(PointerId) [dimo]
          | List(Bottom)
          | PointerId
SYNTAX List(DeclId) ::= List(DeclId) , List(DeclId) [dimo]
          | DeclId
          | List(Bottom)
SYNTAX List(Exp) ::= List(Exp) , List(Exp) [dimo]
          | Exp
          | List(DeclId)
          | List(PointerId)
END MODULE

MODULE KERNELC-DESUGARED-SYNTAX
IMPORTS K-LATEX
IMPORTS KERNELC-SYNTAX
MACRO ! E = E ? 0 : 1
MACRO E1 && E2 = E1 ? E2 : 0
MACRO E1 || E2 = E1 ? 1 : E2
MACRO if( E ) St = if( E ) St else {}
MACRO NULL = 0
MACRO f () = f ( () )
MACRO int * PointerId = int PointerId
MACRO #include< Stmts > = Stmts
MACRO E1 [ E2 ] = * E1 * E2
MACRO scanf("%d", &X) = scanf("%d", E)
MACRO int * PointerId = E = int PointerId = E
MACRO int X = E ; = int X ; X = E ;
MACRO stdio.h = {}
MACRO stdlib.h = {}
END MODULE

```

MODULE KERNELC-SEMANTICS
IMPORTS K-SHARED
IMPORTS K-KERNELC-DESUGARED-SYNTAX+PL-CONVERSION+PL-RANDOM
CONFIGURATION:



RULE $I_1 + I_2 \rightarrow I_1 +_{nat} I_2$

RULE $I_1 \cdot I_2 \rightarrow I_1 \cdot I_2$

RULE $I_1 \% I_2 \rightarrow I_1 \% I_2$ when $I_2 \neq 0$

RULE $I_1 \leq I_2 \rightarrow \text{Bool}$ if $I_1 \leq_{nat} I_2$

RULE $I_1 < I_2 \rightarrow \text{Bool2Int}$ if $I_1 <_{nat} I_2$

RULE $I_1 = I_2 \rightarrow \text{Bool2Int}$ if $I_1 =_{nat} I_2$

RULE $I_1 \neq I_2 \rightarrow \text{Bool2Int}$ if $I_1 \neq_{nat} I_2$

RULE $?_i \rightarrow \text{if}(_) \text{else}$

RULE $\text{if}(I) \text{ else } St \rightarrow St$

RULE $\text{if}(I) St \text{ else } \rightarrow St$

RULE $\text{while}(E) \{ St \} \text{ while}(E)$

RULE $\text{printf}("%d", I)$

RULE void

RULE $\text{scanf}("%d", N)$

RULE $\text{scanf}("%d", &X)$

RULE $X = I$

RULE $V ; \rightarrow$

RULE $\{ St \} \rightarrow Sts$

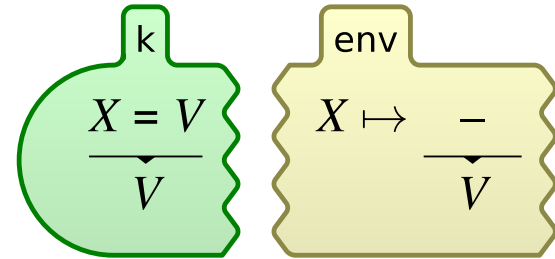
RULE $\{ \} \rightarrow$

RULE $St Sts \rightarrow St \cdot Sts$

RULE $\text{int } X \{ St \}$

RULE $\text{int } X \{ St \} \text{ return void ;}$

Semantic rules given contextually



rule

$\langle k \rangle X = V \Rightarrow V \dots \langle /k \rangle$

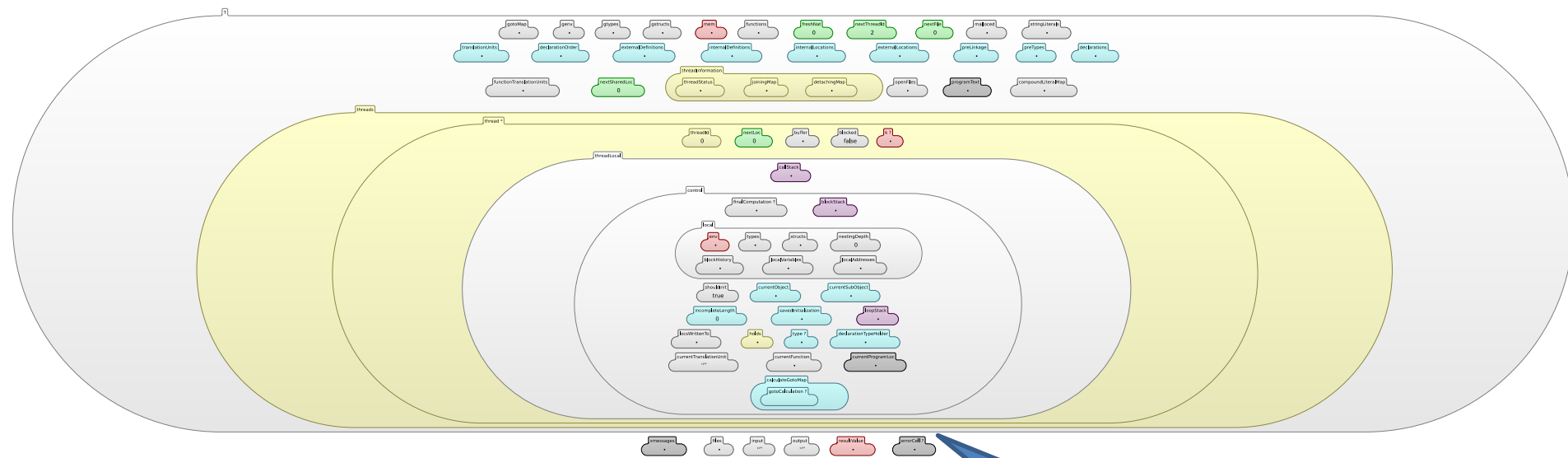
$\langle env \rangle \dots X \mapsto (_ \Rightarrow V) \dots \langle /env \rangle$

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]
- **x86-64 ISA**: 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



... plus ~5000 rules ...

120 Cells!

Definition of x86-64 ISA (PLDI'19)

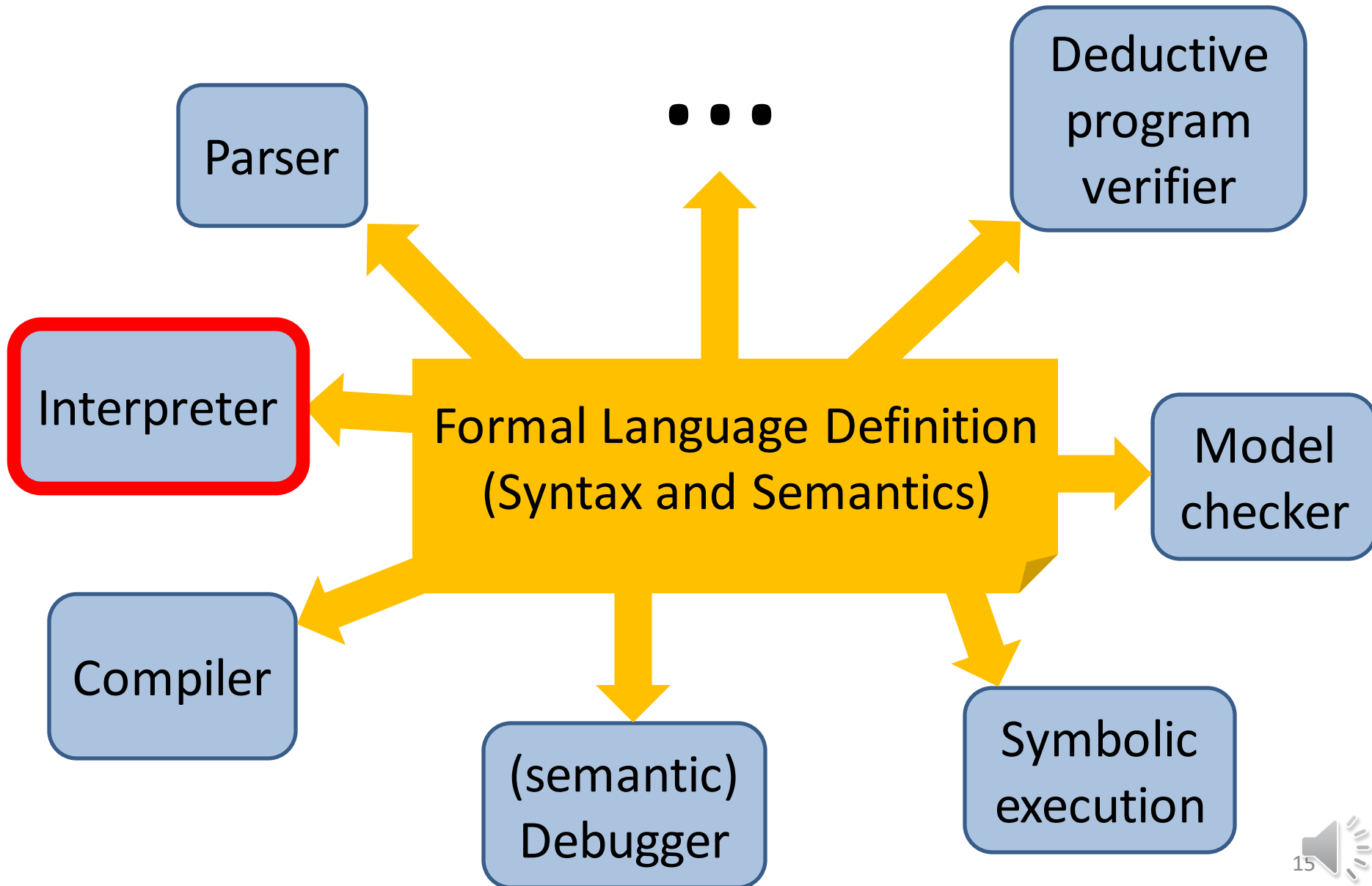
- Complexity of the ISA
 - One of the most complex ISAs widely used in desktop and servers
 - Consists of 3736 instructions with various degree of complexity
 - Previous works formalizes a fraction of instruction semantics

Definition of x86-64 ISA (PLDI'19)

- Few highlights of our work

- The semantic of the defined ISA includes ~5200 semantic rules
- Modeled the IEEE-754 floating point standard in K
- The semantics is validated by co-simulation against hardware
 - Validation includes unit testing with 7000+ input states and executing the entire GCC-c torture test using the semantics
 - Thanks to the K interpreter we leveraged for free
- Demonstrated the applicability of the semantics for program verification, security vulnerability tracking
 - Thanks to K program verifier and symbolic execution engine we leveraged for free

Ideal Language Framework Vision [K]



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

Conventional
compilers do not
detect problem

```
$ gcc 6-int-overflow.c  
$ ./a.out  
$  
$ kcc 6-int-overflow.c  
$ ./a.out
```

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

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			GramaTech 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
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; $FPR = 100 - FPR$

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



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
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; $FPR = 100 - PM$

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

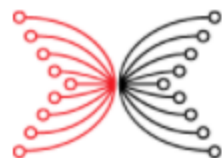


From RV-Match to Blockchain

- RV-Match currently commercialized within

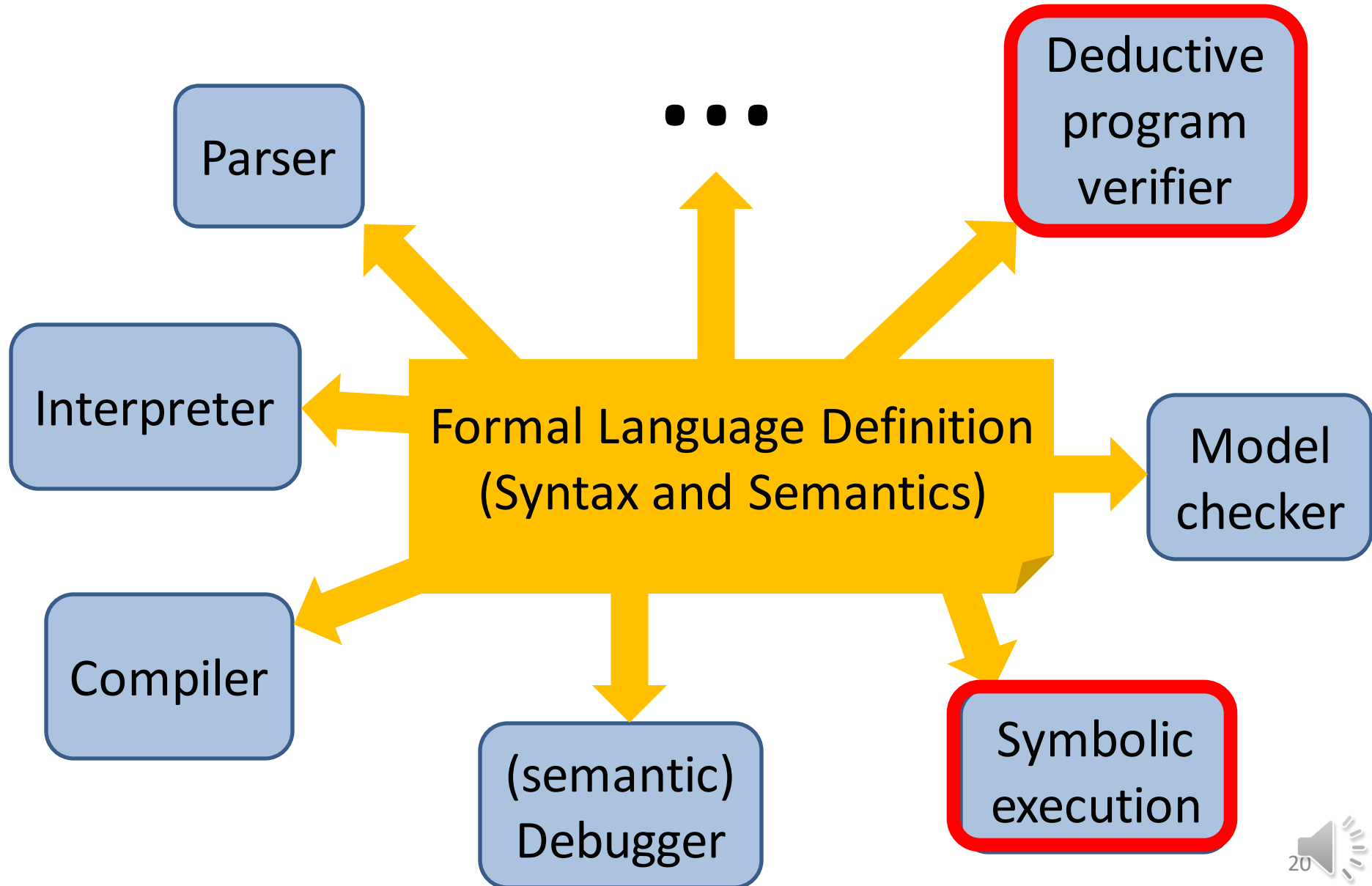


- The same technology, K, used for defining blockchain languages: EVM, IELE, Plutus, ...



INPUT | OUTPUT

Ideal Language Framework Vision [K]



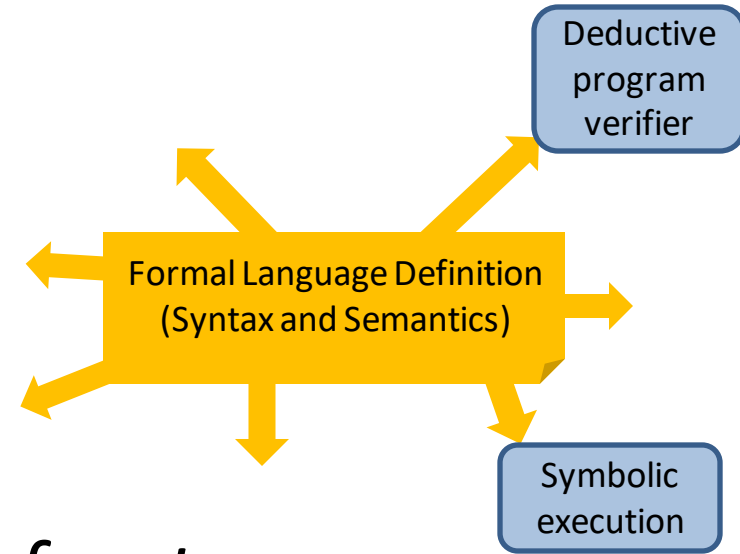
State-of-the-Art

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

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

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!



Reachability Logic (Semantics of K)

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

- “Rewrite” rules over configurations

$$\varphi \Rightarrow \varphi'$$

- Reachability rules capture Hoare triples [FM'12]

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

Sum $1+2+\dots+n$ in IMP: Main

```
rule
  <k>
    int n, sum;
    n = N:Int;
    sum = 0;
    while (!(n <= 0)) {
      sum = sum + n;
      n = n + -1;
    }
  =>
    .K
  </k>
  <state>
    .Map
  =>
    n    |-> 0
    sum  |-> ((N +Int 1) *Int N /Int 2)
  </state>
requires N >=Int 0
```


Sum $1+2+\dots+n$ in IMP: Invariant

```
rule
  <k>
    while (!(n <= 0)) {
      sum = sum + n;
      n = n + -1;
    }
  =>
    .K
  ...</k>
  <state>...
    n    |-> (N:Int => 0)
    sum  |-> (S:Int => S +Int ((N +Int 1) *Int N /Int 2))
  ...</state>
requires N >=Int 0
```

OK Performance

[OOPLSA'16]

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

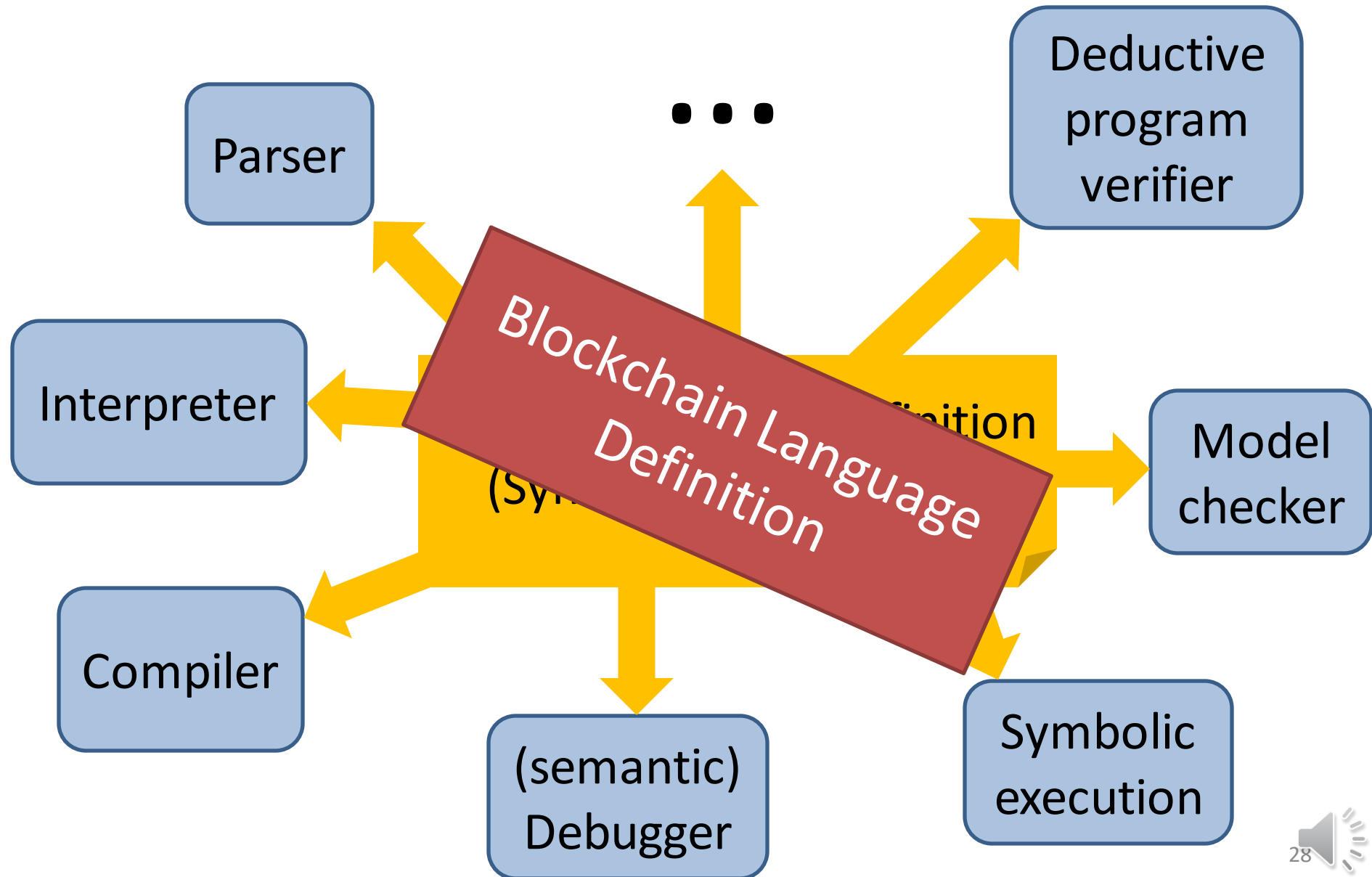
Time (seconds) spent on
domain reasoning (like
querying Z3)

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; now 5x*

What Can We Do with KEVM?

1) *Formal documentation:* <http://jellopaper.org>

The screenshot shows a web browser window with the URL <https://jellopaper.org/evm/>. The page title is "EVM Execution — The EVM Jello". The left sidebar contains a search bar and a list of navigation links: "The EVM Jello 0.1", "Search docs", "KEVM: Semantics of EVM in K", "Resources", "EVM Execution", "EVM Programs", "Ethereum Gas Calculation", "EVM Program Representations", "EVM Integration with Production Client", "Ethereum Simulations", "eDSL High-Level Notations", "Network State", "Resources", and "EVM Words". The main content area is titled "EVM Execution" and includes a sub-section "EVM Opcodes". Under "EVM Opcodes", there is a section "EVM Control Flow" which states: "The `JUMP*` family of operations affect the current program counter." Below this, there is a code block showing the syntax and rules for `JUMPDEST` and `JUMP` opcodes. The code is as follows:

```
rule #blockhash(ListItem(0) _, _, _) => 0
rule #blockhash(ListItem(H) _, N, N, _) => H
rule #blockhash(ListItem(_) L, N, HI, A) => #blockhash(L, N, HI Int 1, A Int 1) [owise]

syntax NullStackOp ::= "JUMPDEST"
// -----
rule <k> JUMPDEST => ... </k>

syntax UnStackOp ::= "JUMP"
// -----
rule <k> JUMP DEST => ... </k>
  <pc> _ => DEST </pc>
  <program> ... DEST |-> JUMPDEST ... </program>

rule <k> JUMP DEST => #end EVMC_BAD_JUMP_DESTINATION ... </k>
  <program> ... DEST |-> OP ... </program>
  requires OP !=K JUMPDEST

rule <k> JUMP DEST => #end EVMC_BAD_JUMP_DESTINATION ... </k>
```

What Can We Do with KEVM?

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

The screenshot shows a web browser with two tabs. The active tab is titled "Announcing Beneficiaries of the Ethereum Foundation Grants" and displays a list of grant recipients. The "Runtime Verification" entry is highlighted with a red rectangle. The browser's address bar shows the URL "https://blog.ethereum.org/2018/03/07/announcing-beneficiaries-ethereum-foun...".

runtime verification

Formal Verification

Comprehensive. End-to-end. Faithful.

- Comprehensive.** We specify and verify smart contract owner the strongest
- End-to-end.** We start with the high-machine level. The last step is to ver
- Faithful.** We communicate with the captures the intended functionality. soon generate correctness certificate

Awardee List

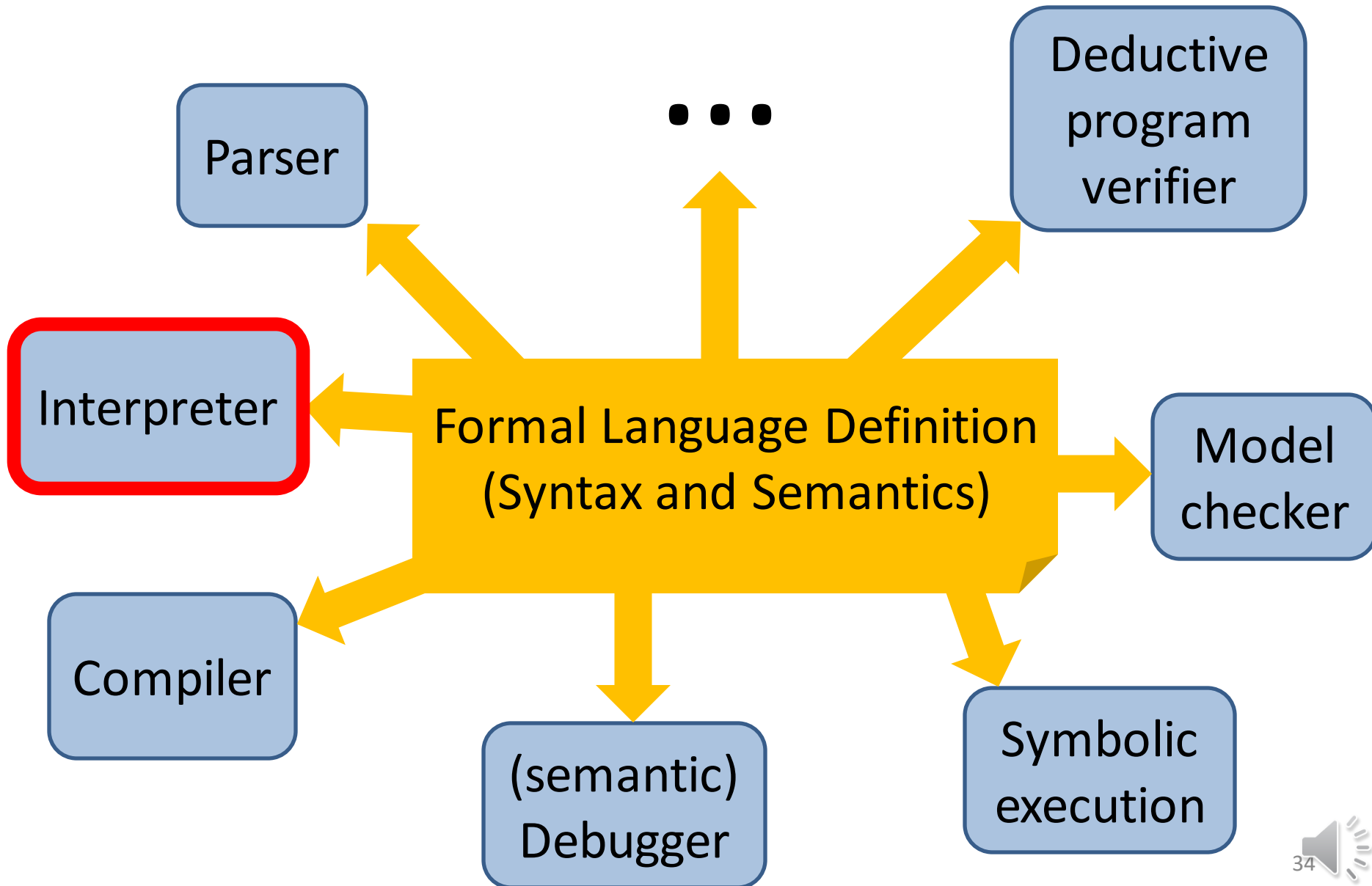
Announcing Beneficiaries of the Ethereum Foundation Grants

- [L4 Research](#) – Scalability Grant – \$1.5M. State channels research.
- Runtime Verification** – Security Grant – \$500K. Casper contract formal verification.
- [ETHGlobal](#) – DevEx Grant* – \$200K. World-class developer conferences for Ethereum
- [Prismatic Labs](#) – Scalability Grant – \$100K. Sharding implementation.
- [DDA](#) – #buidl Grant** – \$100K. Tokenless decentralized derivatives network + state channels R&D
- [Barcelona Supercomputing Center](#) – Scalability Grant – \$50K. Sharding simulation.
- [Plasma Taiwan Dev](#) – Scalability Grant – \$25K. Plasma implementation.
- [Ethers.js](#) – DevEx Grant – \$25K. Web3.js alternative.
- [Turbo Geth](#) – Scalability Grant – \$25K. Geth optimization.
- [Solium](#) – DevEx Grant – \$10K. Solidity static analyzer.
- [Alex Komarov](#) – Design Grant – \$10K. Key management UX study
- [\(Anonymous\)](#) – Hacktarnship – \$10K. Deterministic WebAssembly.

[FSE'18]

Ongoing K Infrastructure Projects

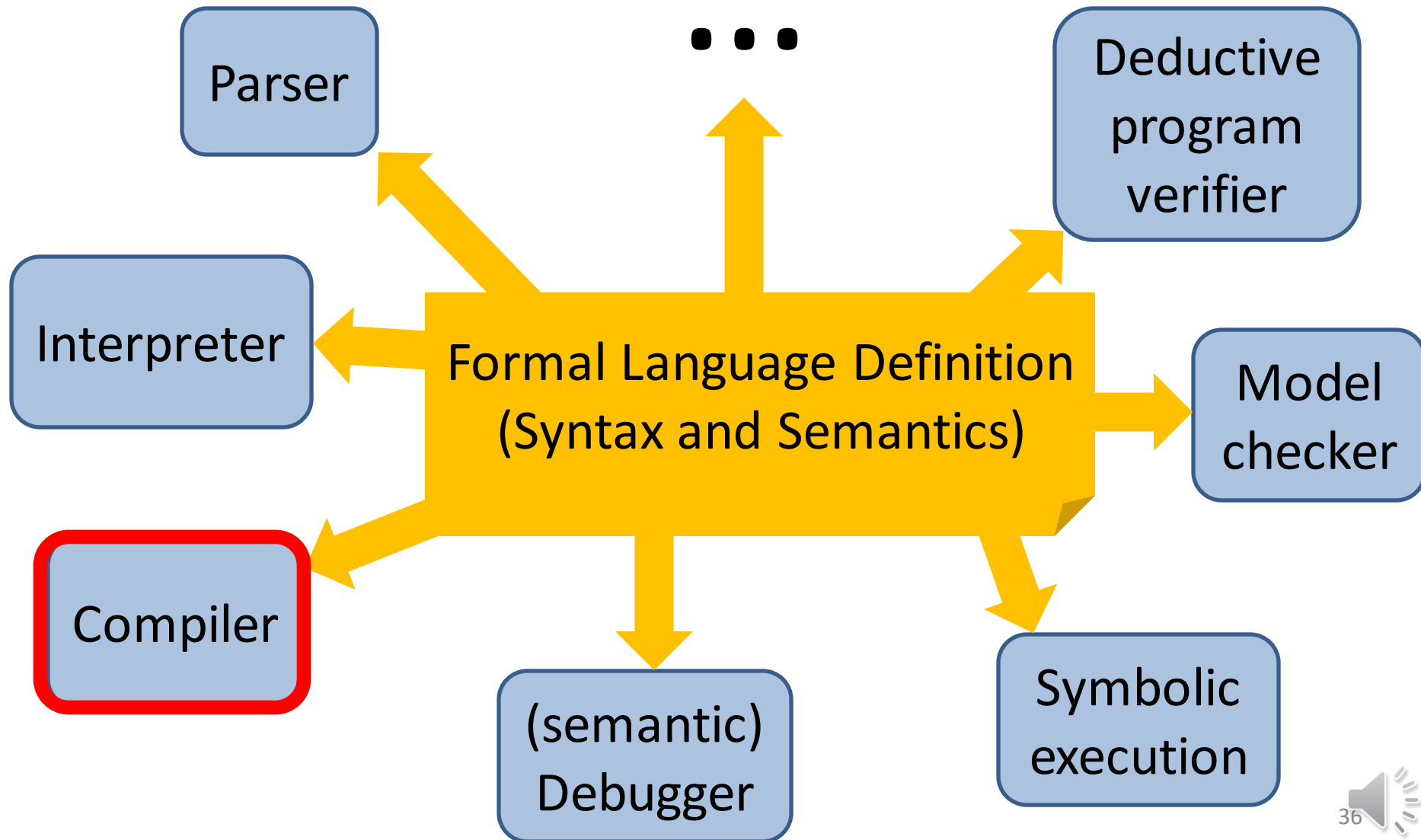
1. Fast LLVM Backend for K



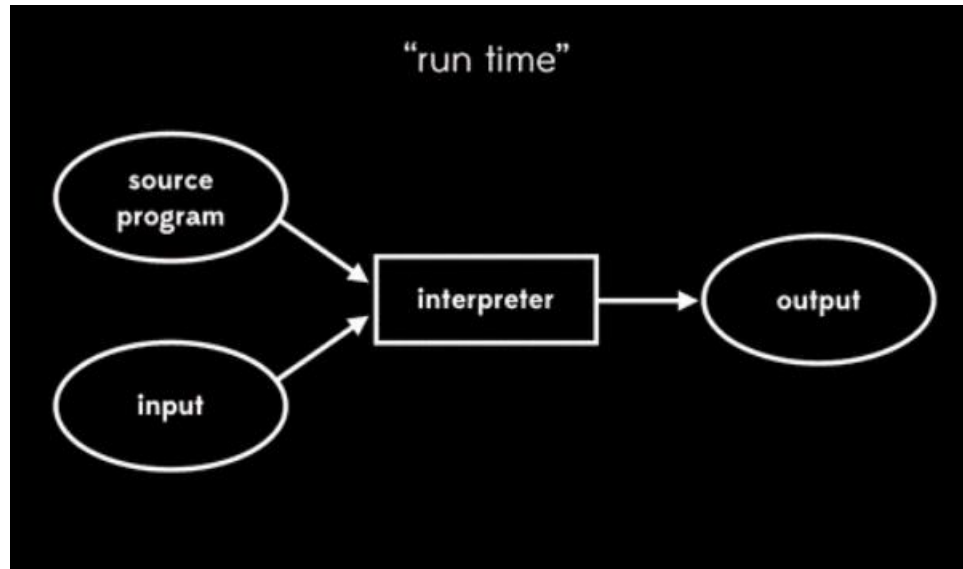
Fast LLVM Backend for K

- Current OCAML backend of K:
 - Fast enough to power RV-Match product and KEVM client
 - But still 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!

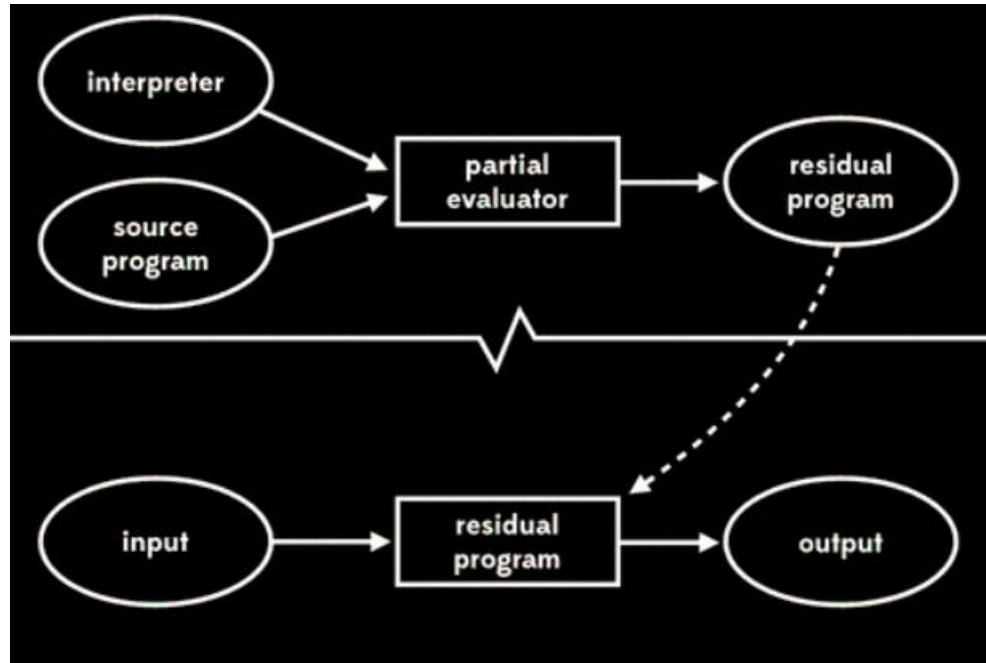
2. Semantics-Based Compilation



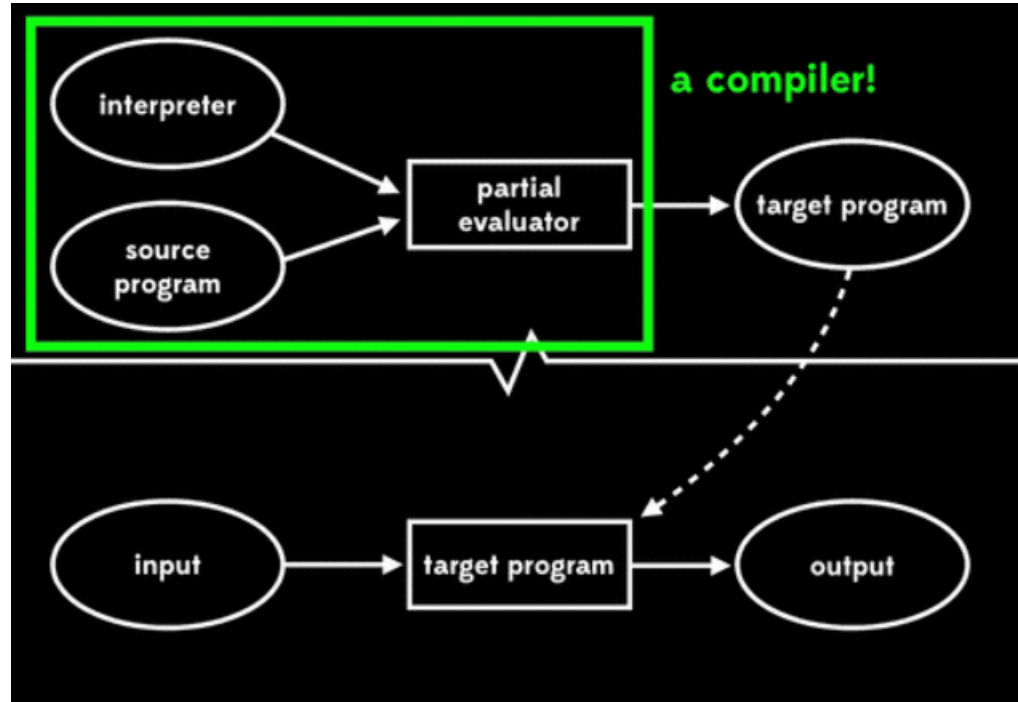
Futamura first projection



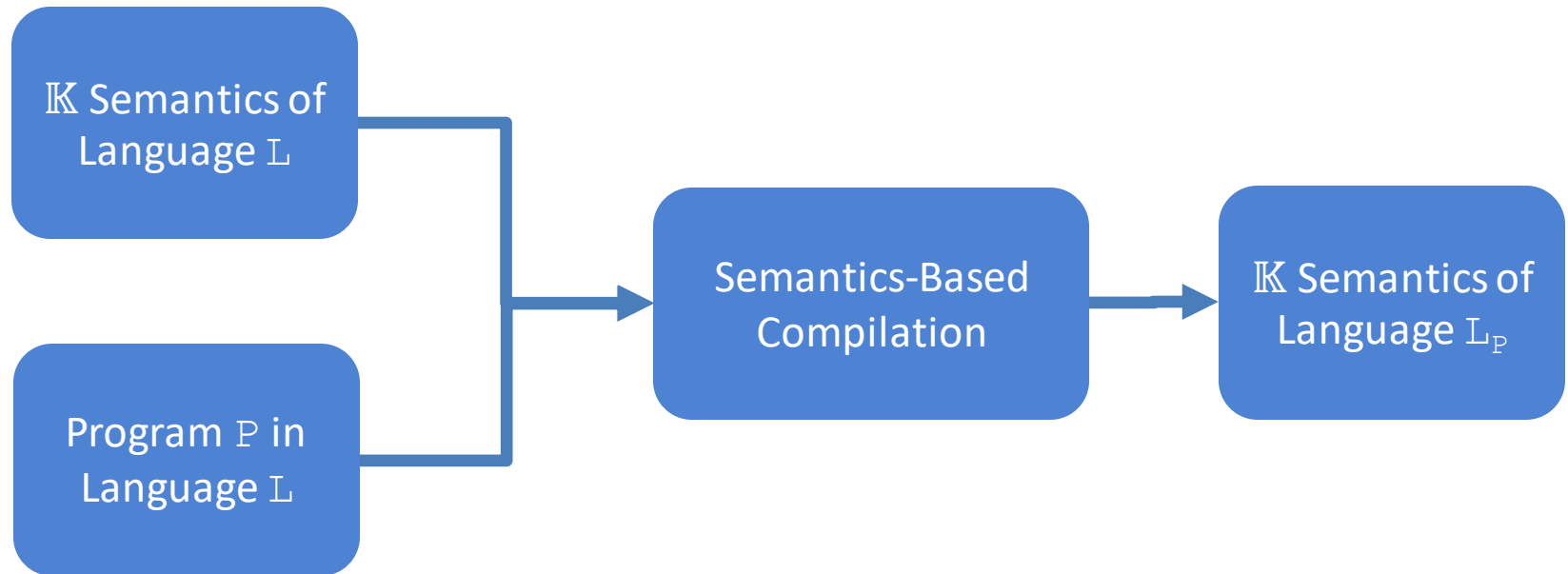
Futamura first projection



Futamura first projection



Semantics-Based Compilation (SBC)



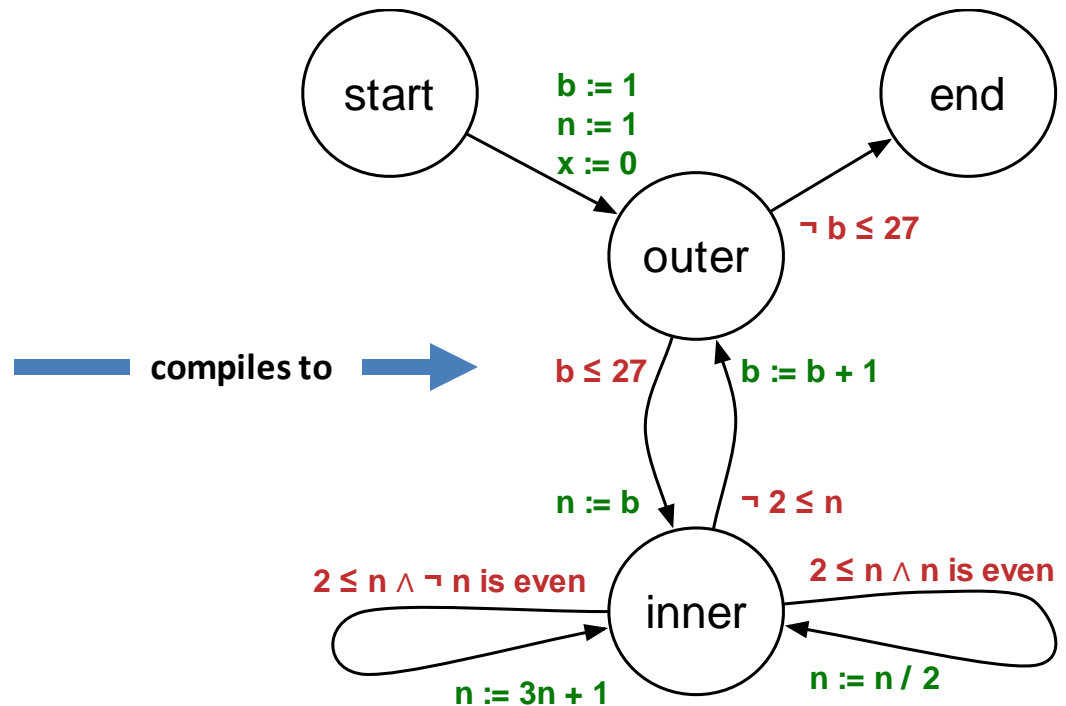
Semantics-Based Compilation (SBC)

Experiments with Early Prototype

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

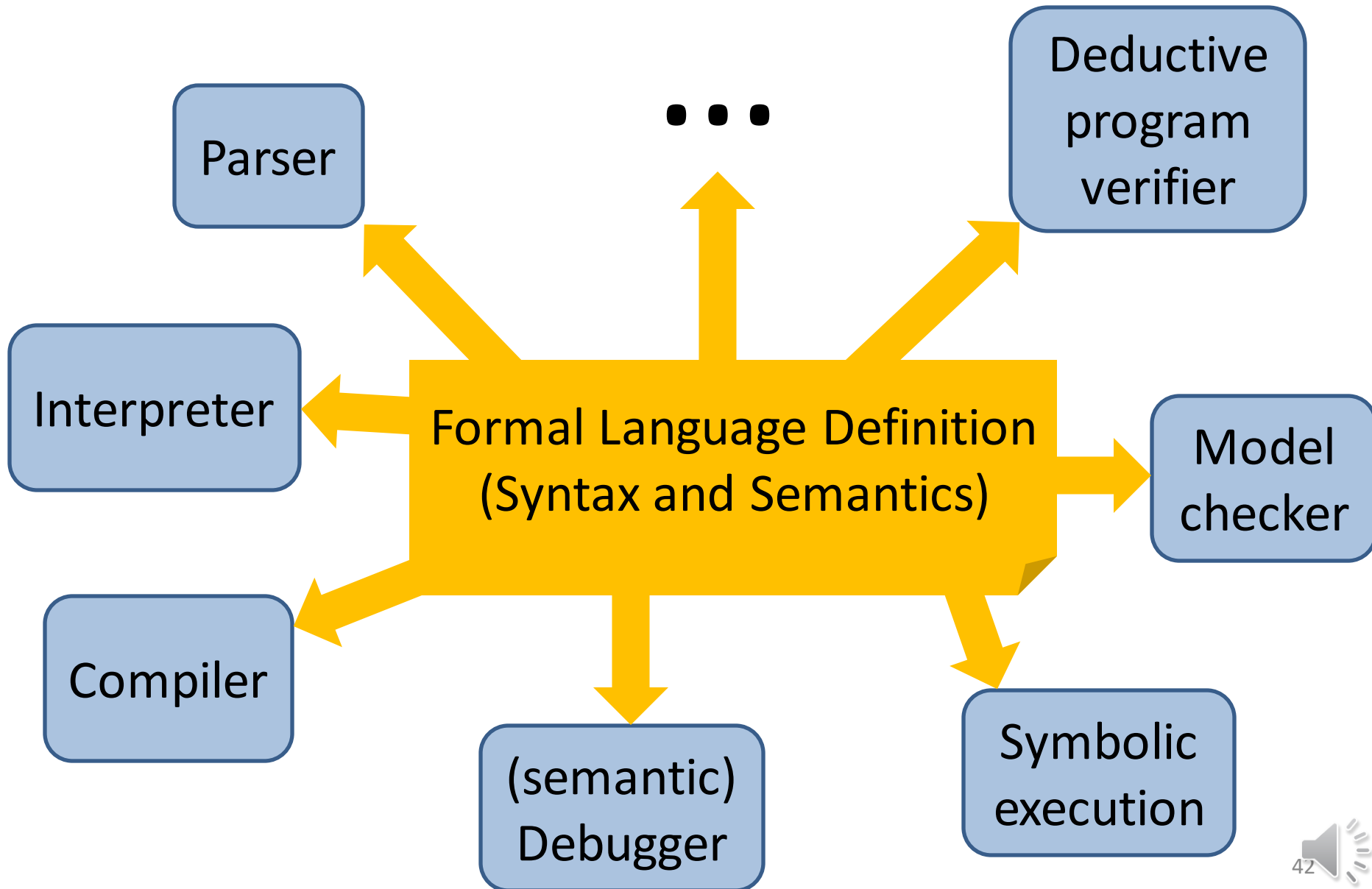
// outer
while (b <= 27) {
  n = b ;

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



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

Conclusion: It Can Be Done!



Backup