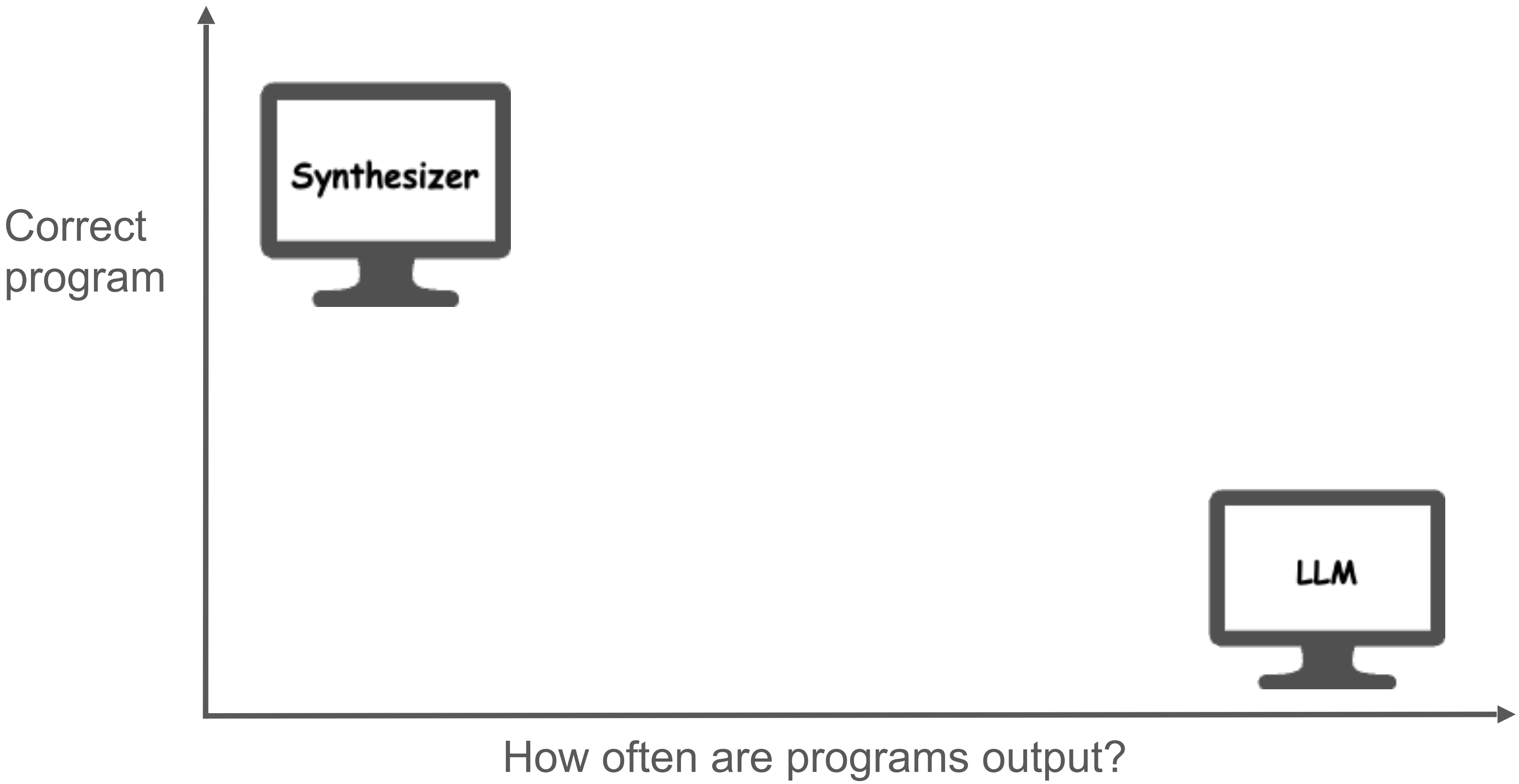
SYNVER: Towards Automated Verification of LLM-Synthesized C Programs

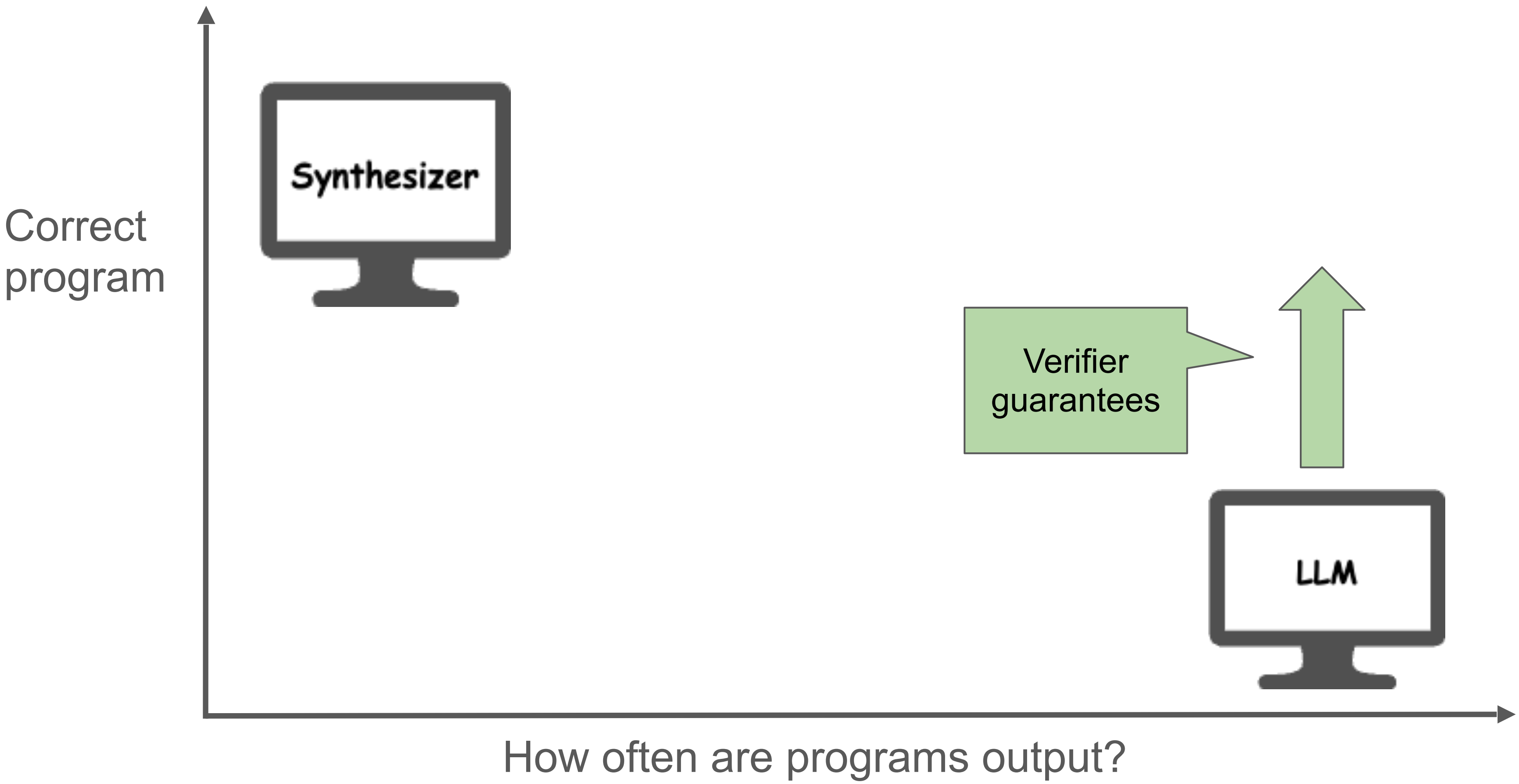
Prasita Mukherjee and Benjamin Delaware

CoqPL 2025

Program Synthesis: Present Scenario



Program Synthesis: Present Scenario



Synthesizing programs: LLMs

$$\forall a, b\ (max(a,b) = a \wedge a \geq b)$$
$$\vee\ (max(a,b) = b \wedge b > a)$$

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ \ * listrep\ (filter\ fi2\ l)\ a$$

....

$$listrep\ [1; l; 2]\ h$$

struct sll *addf(..)
struct sll *addb(..)

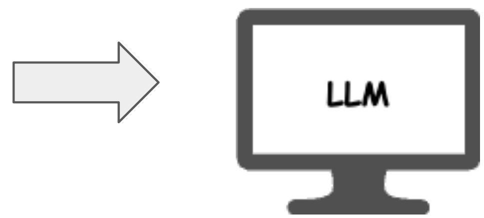
Synthesizing programs: LLMs

$$\forall a, b\ (max(a, b) = a \wedge a \geq b)$$
$$\vee\ (max(a, b) = b \wedge b > a)$$

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ * listrep\ (filter\ fi2\ l)\ a$$

....

$$listrep\ [1; l; 2]\ h$$




```
struct sll *addf(..)
struct sll *addb(..)
```

Synthesizing programs: LLMs

$$\forall a, b\ (max(a,b) = a \land a \geq b) \\ \lor\ (max(a,b) = b \land b > a)$$

$Definition\ fi2 := fun\ x => x =?\ 2$
$listrep\ l\ h\ * listrep\ (filter\ fi2\ l)\ a$

....

$listrep\ [1; l; 2]\ h$

struct sll *addf(..)
struct sll *addb(..)

Synthesizing programs: LLMs

$\forall a, b\ (max(a,b) = a \wedge a \geq b)$
$\vee\ (max(a,b) = b \wedge b > a)$

$Definition\ fi2 := fun\ x => x =?\ 2$
$listrep\ l\ h\ *\ listrep\ (filter\ fi2\ l)\ a$

....

$listrep\ [1; l; 2]\ h$

struct sll *addf(..)
struct sll *addb(..)

Synthesizing programs: LLMs

$\forall a, b\ (max(a,b) = a \wedge a \geq b)$
$\vee\ (max(a,b) = b \wedge b > a)$

$Definition\ fi2 := fun\ x => x =?\ 2$
$listrep\ l\ h\ \ *\ listrep\ (filter\ fi2\ l)\ a$

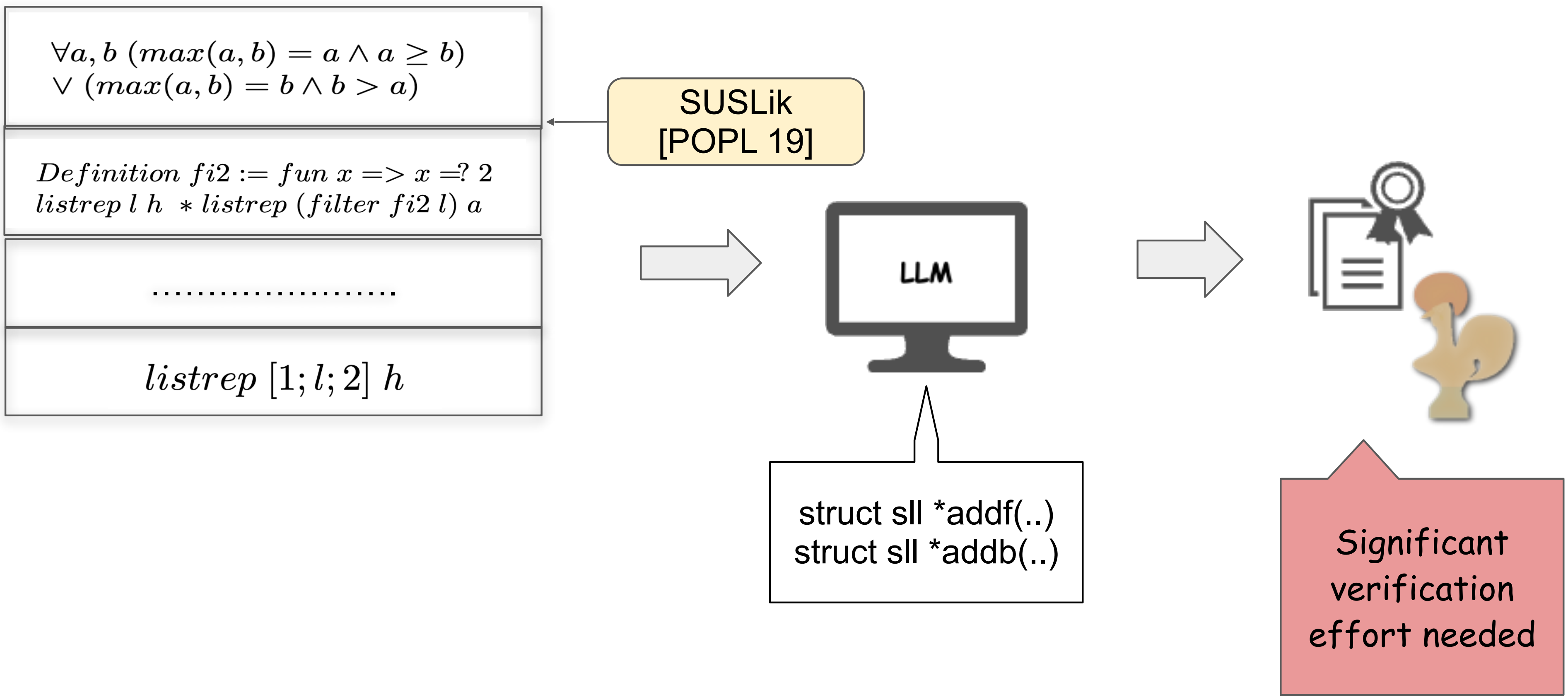
....

$listrep\ [1; l; 2]\ h$

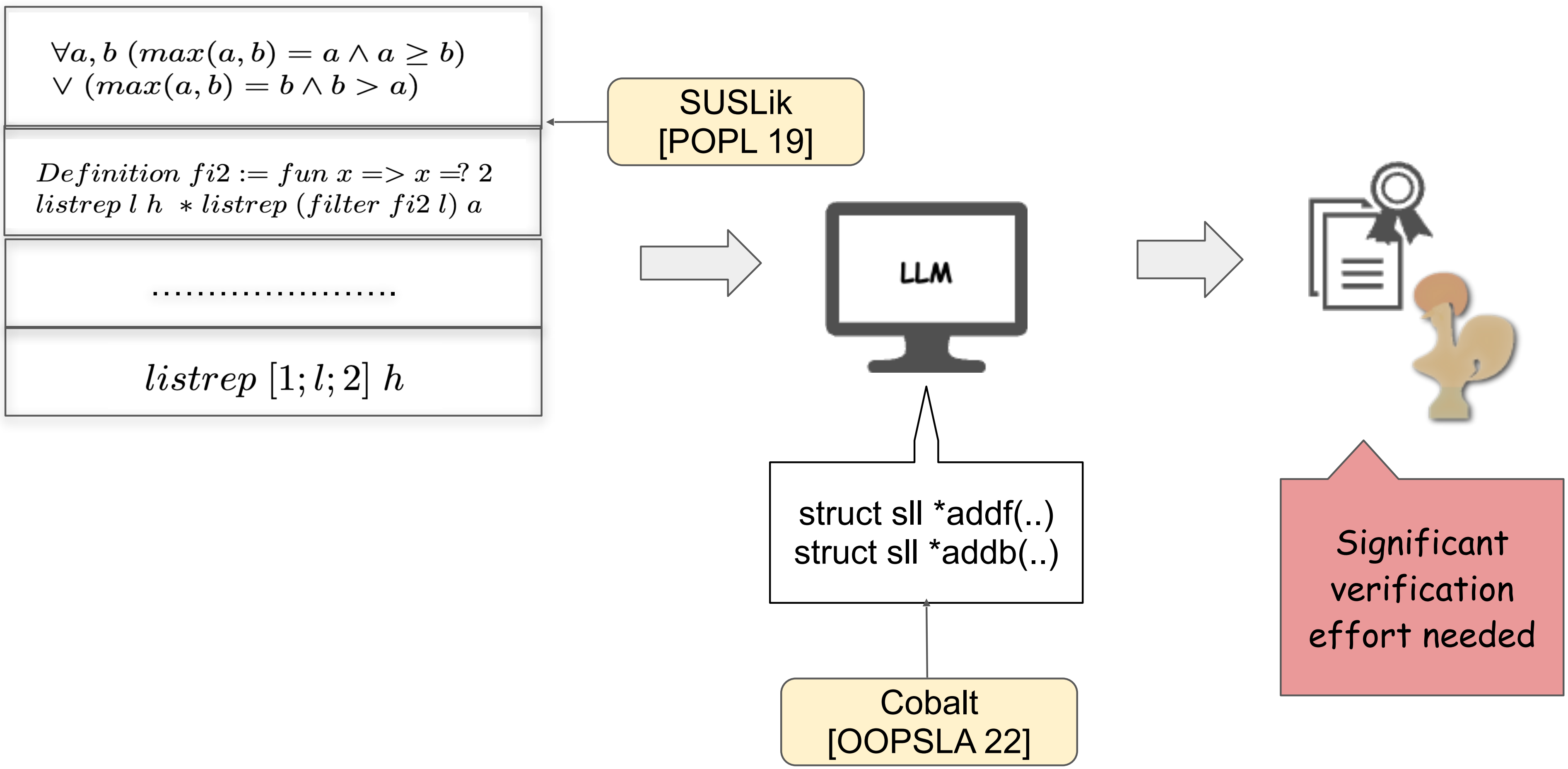
struct sll *addf(..)
struct sll *addb(..)

Significant verification effort needed

Synthesizing programs: LLMs



Synthesizing programs: LLMs



Synthesizing programs: LLMs

$$\forall a, b\ (max(a,b) = a \wedge a \geq b) \vee (max(a,b) = b \wedge b > a)$$

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ * listrep\ (filter\ fi2\ l)\ a$$

....

$$listrep\ [1; l; 2]\ h$$

SUSLik
[POPL 19]

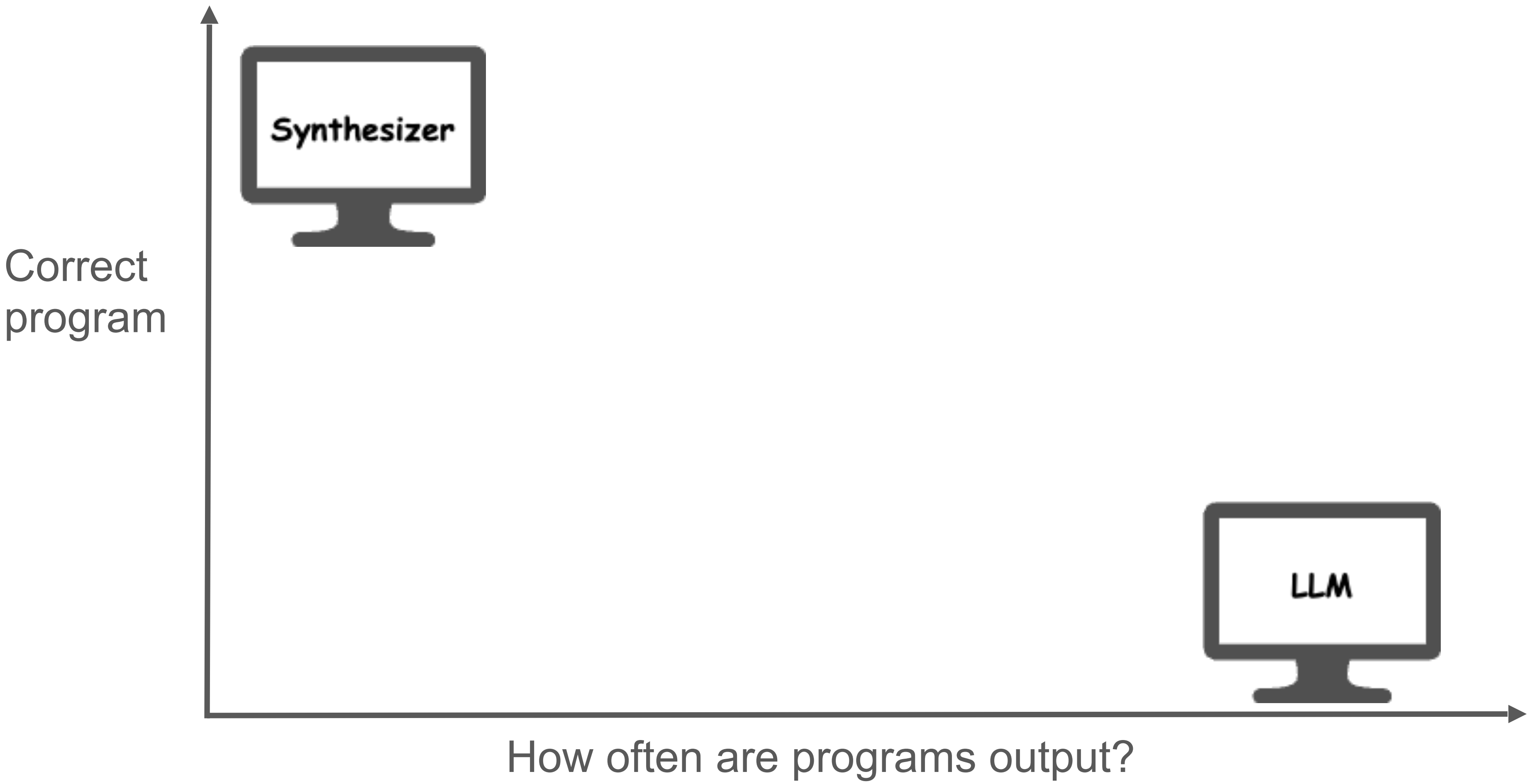
struct sll *addf(..)
struct sll *addb(..)

Cobalt
[OOPSLA 22]

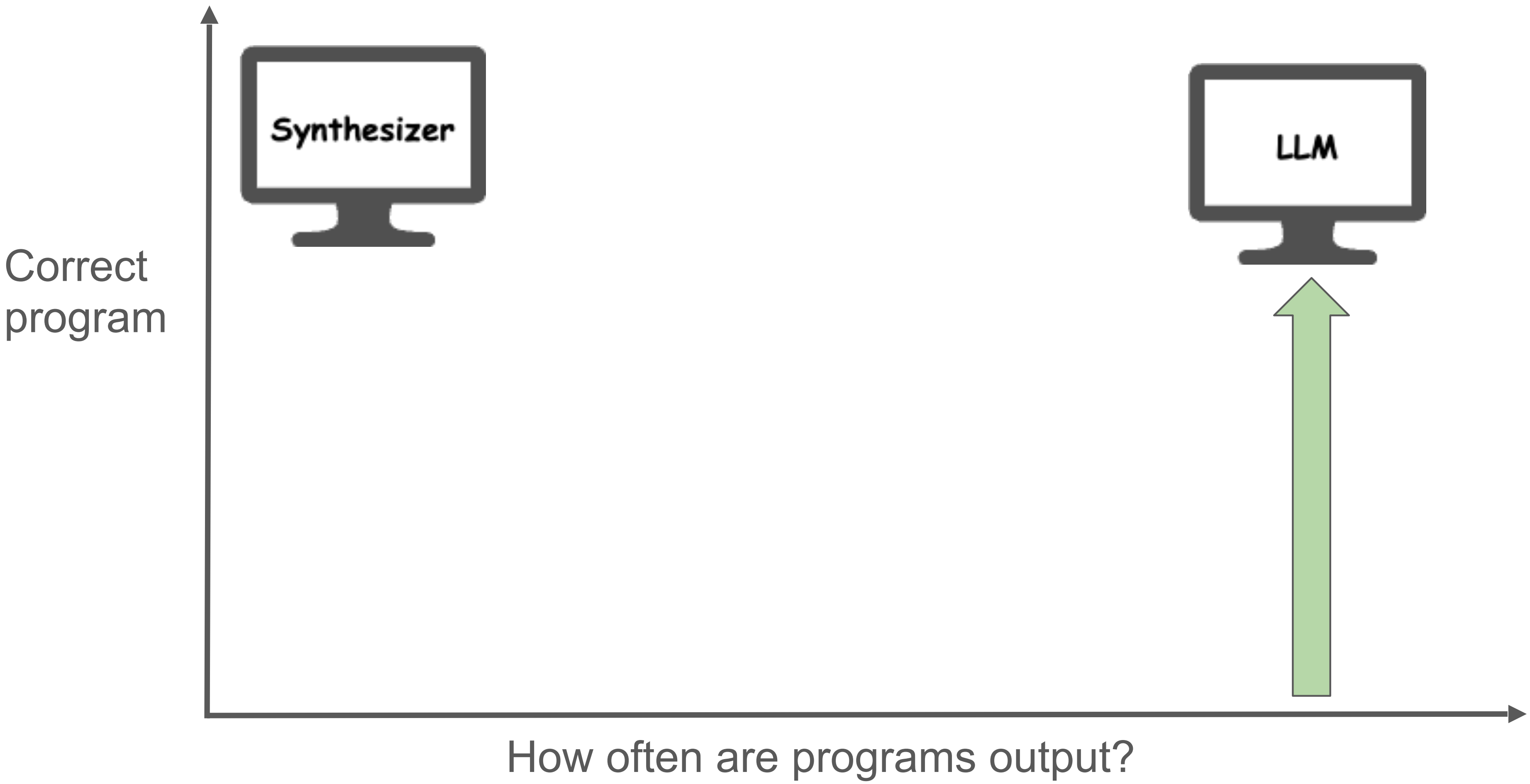
Significant verification effort needed

Large search space.
Will always produce some output

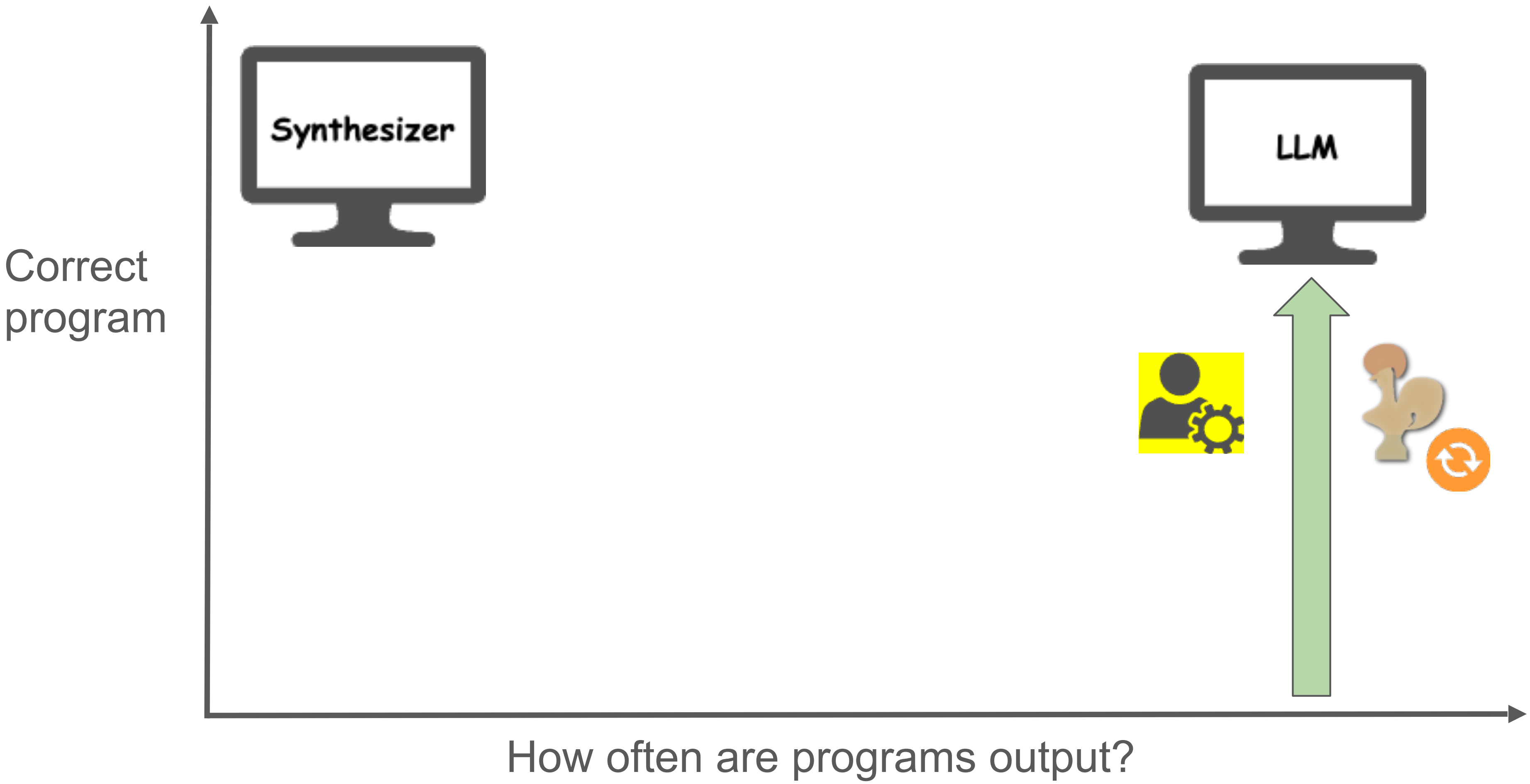
Goal of SYNVER



Goal of SYNVER



Goal of SYNVER



Verification Challenges

{P}

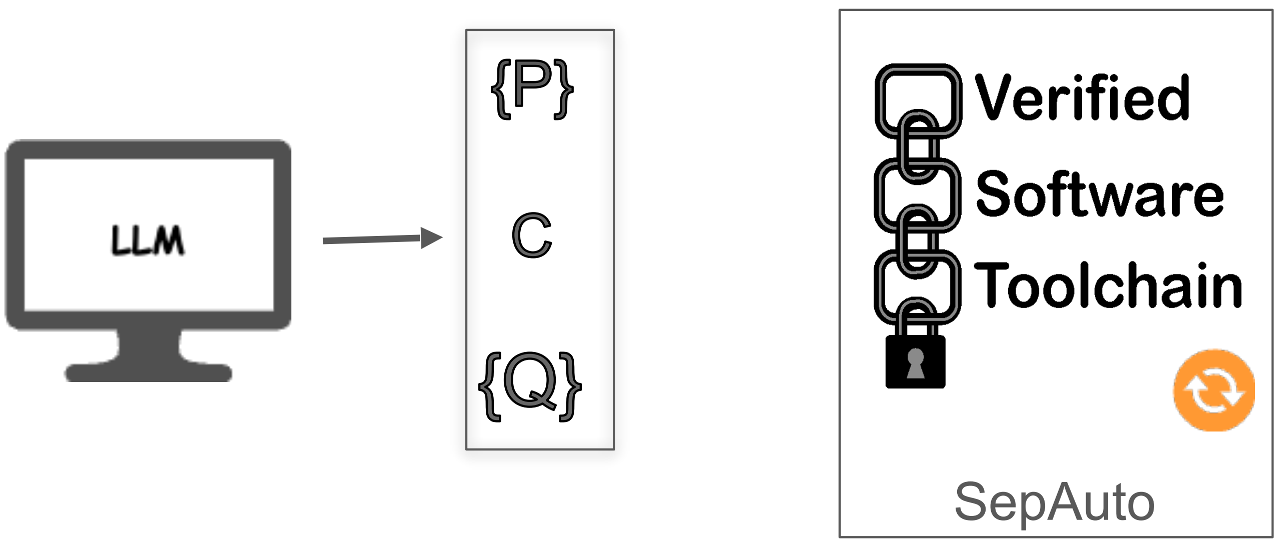
C

{Q}

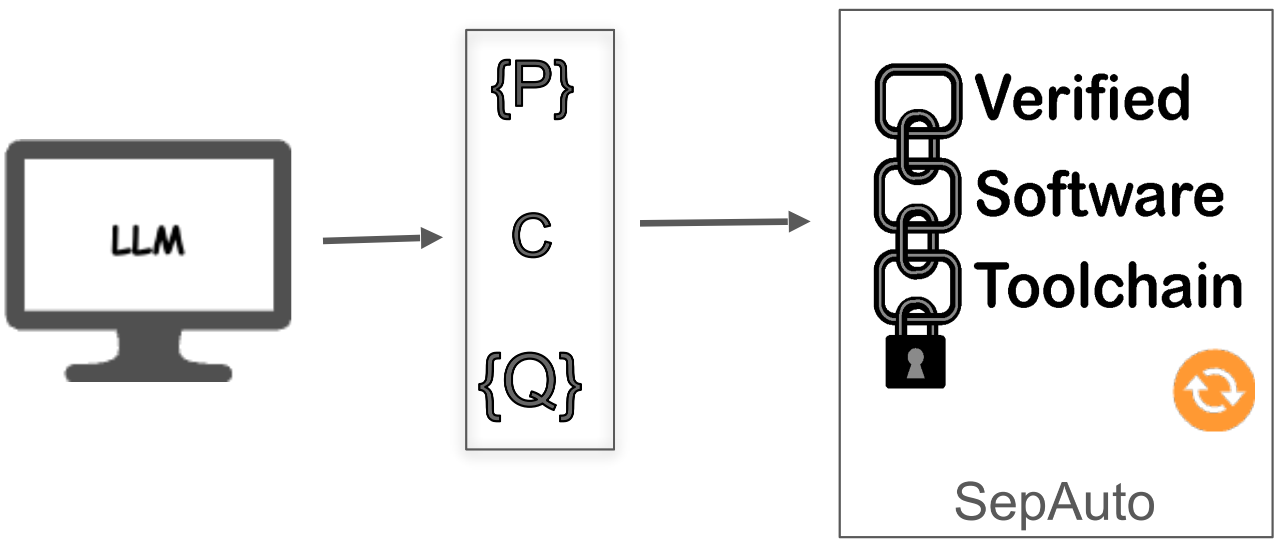
Verified Software Toolchain

SepAuto

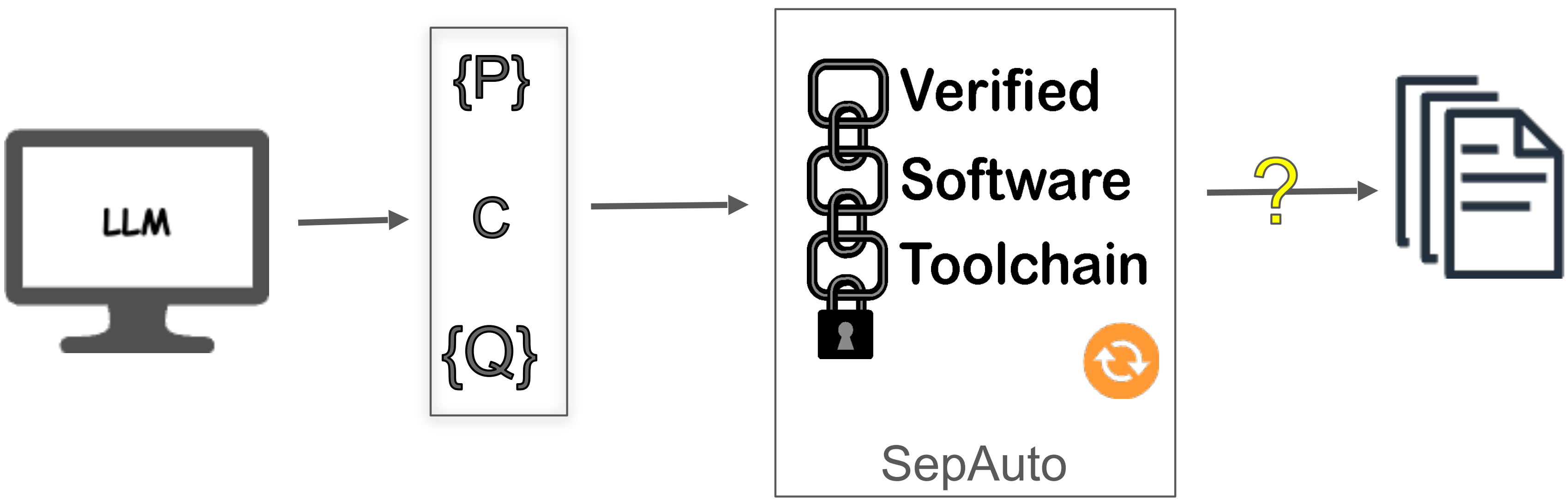
Verification Challenges



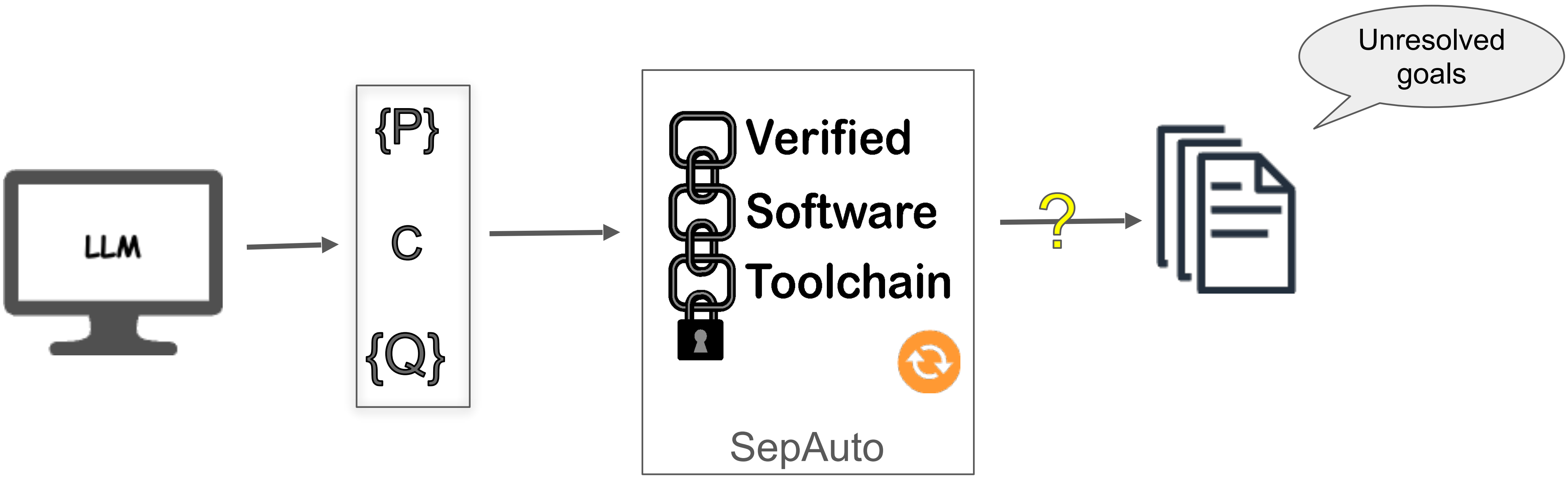
Verification Challenges



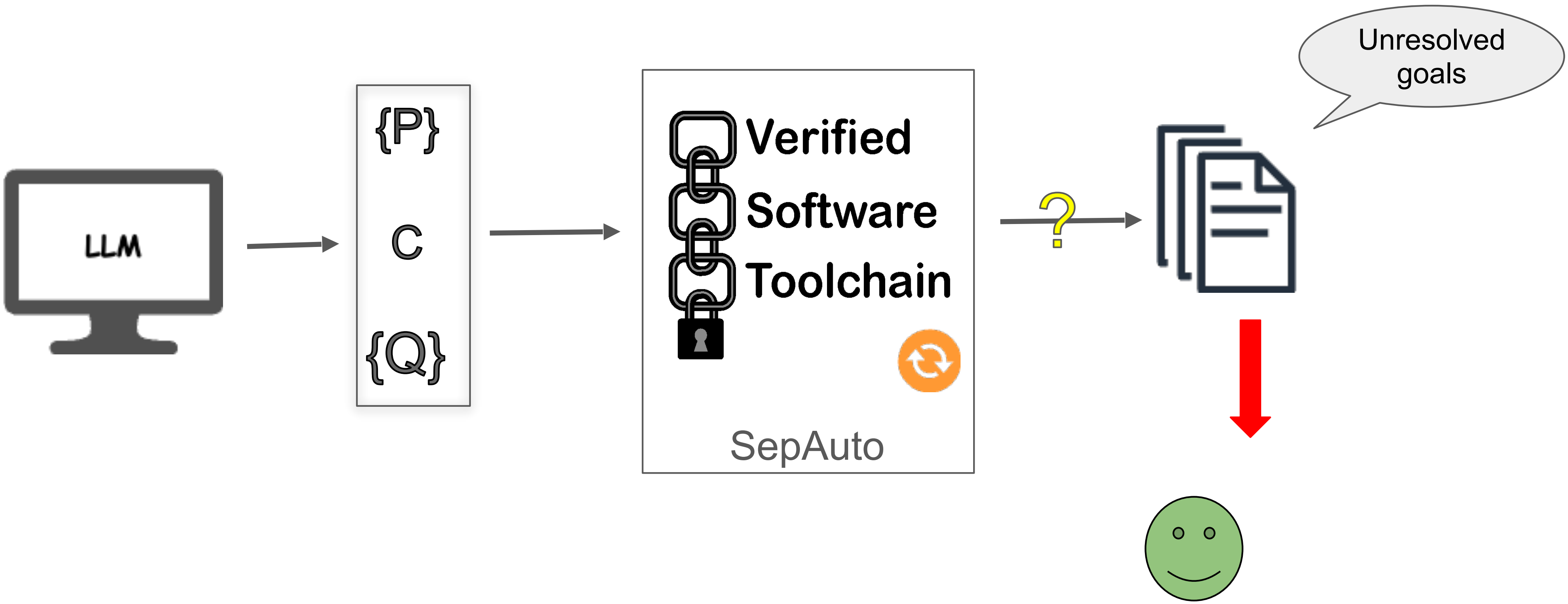
Verification Challenges



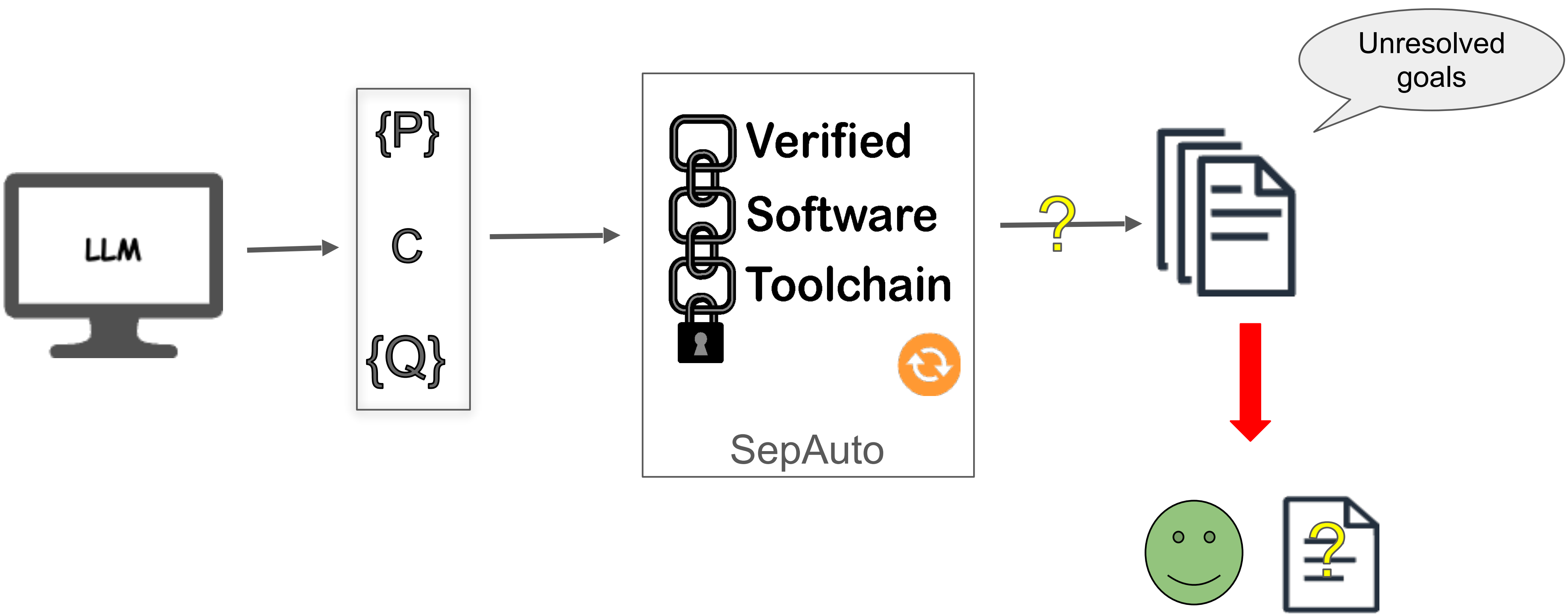
Verification Challenges



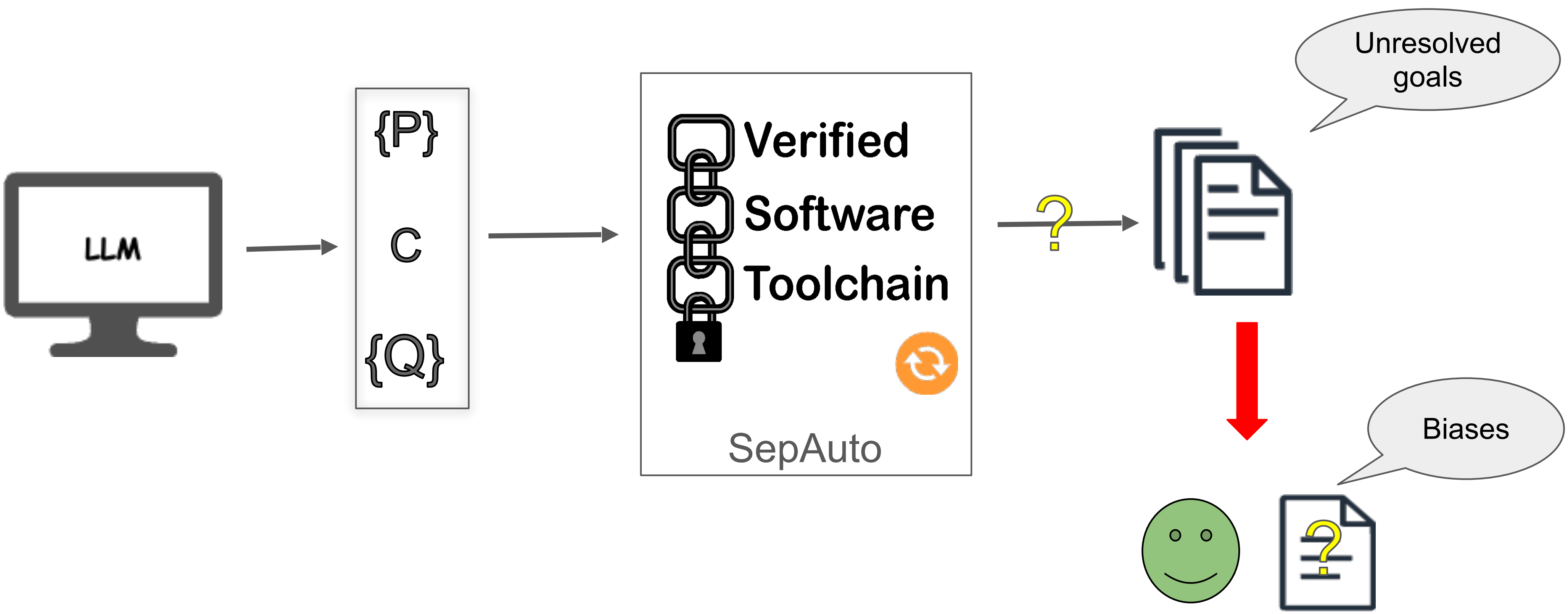
Verification Challenges



Verification Challenges



Verification Challenges

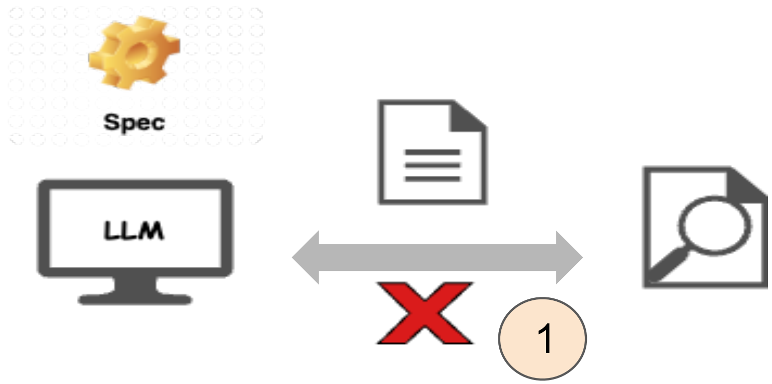


Outline

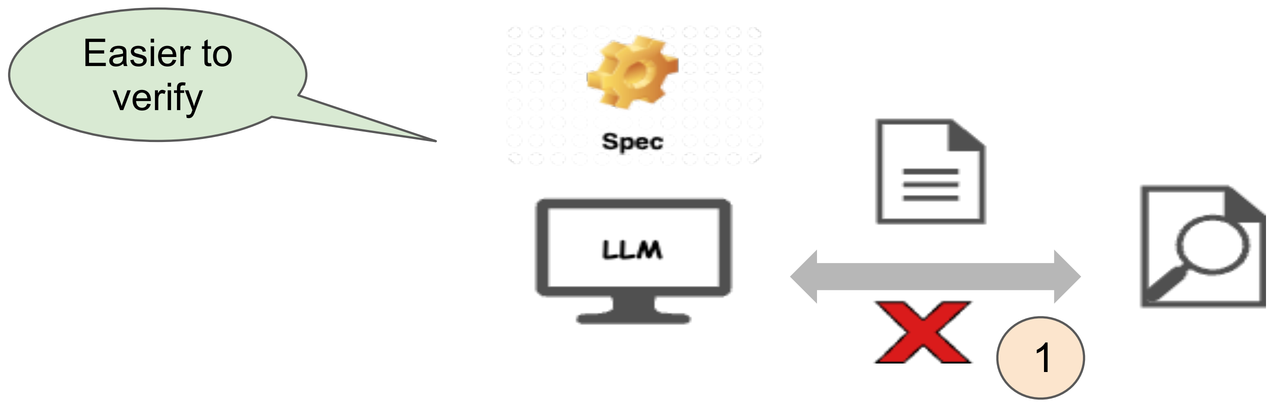
- General purpose synthesizer for C programs
- Generates easily verifiable code through prompt design
- SepAuto - Ltac on top of VST to automate proofs
- Experimental Setup
- Results

SYNVER: Workflow

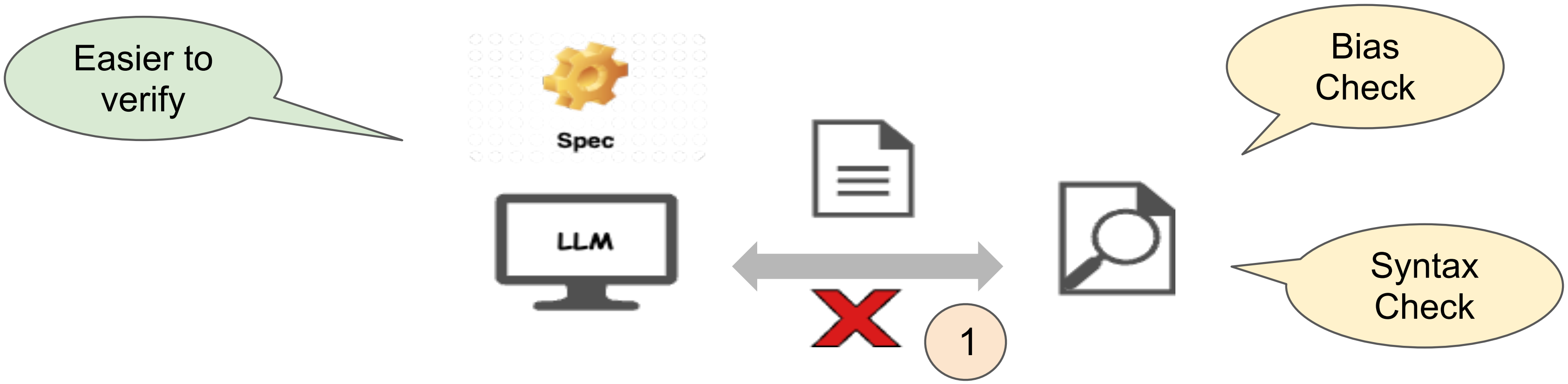
SYNVER: Workflow



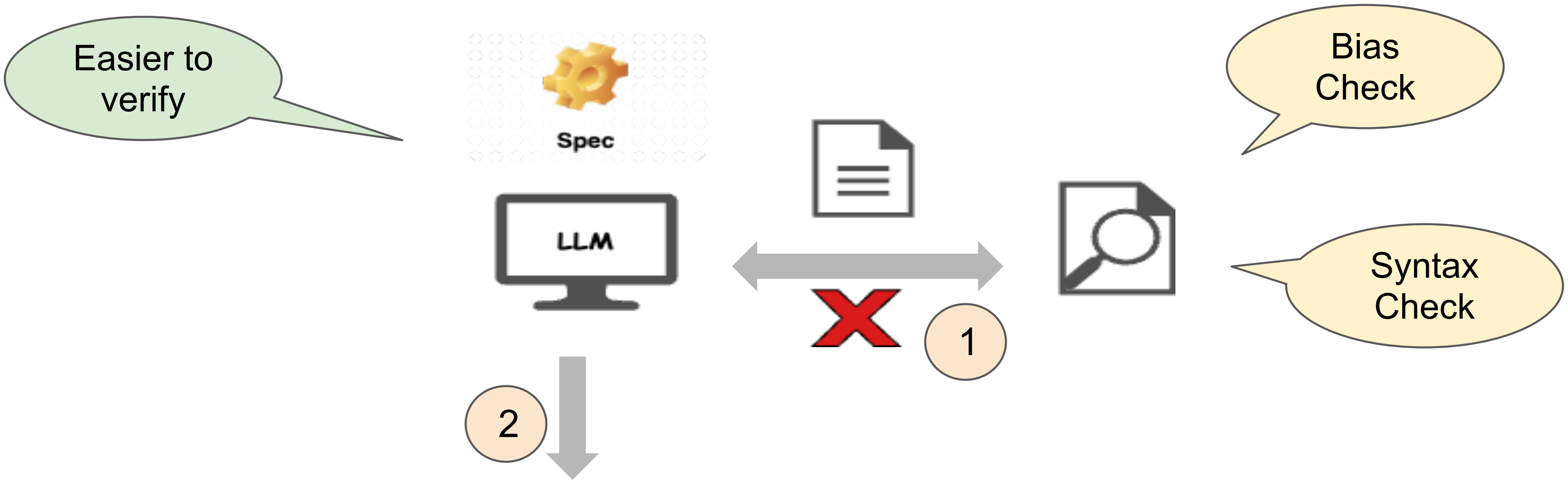
SYNVER: Workflow



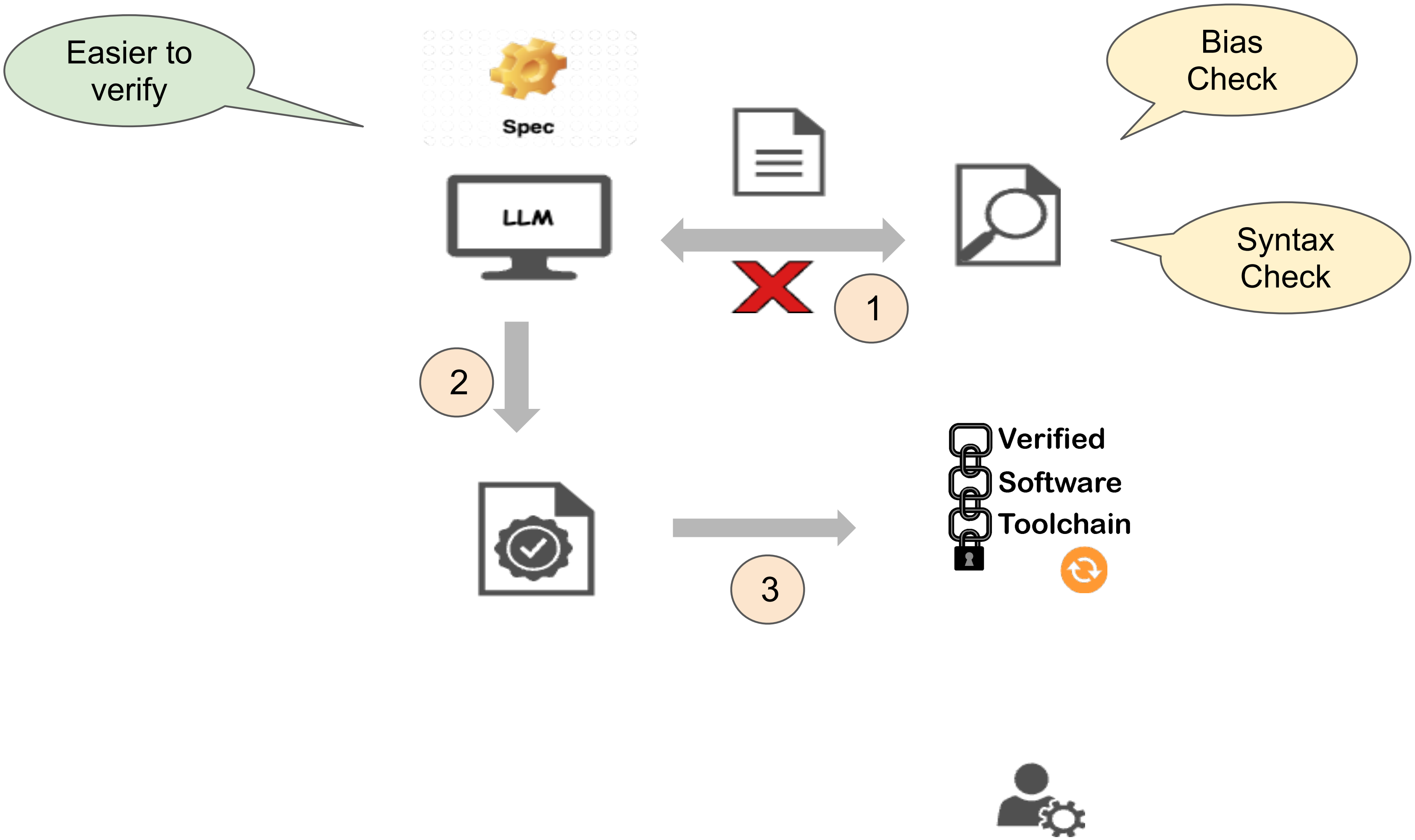
SYNVER: Workflow



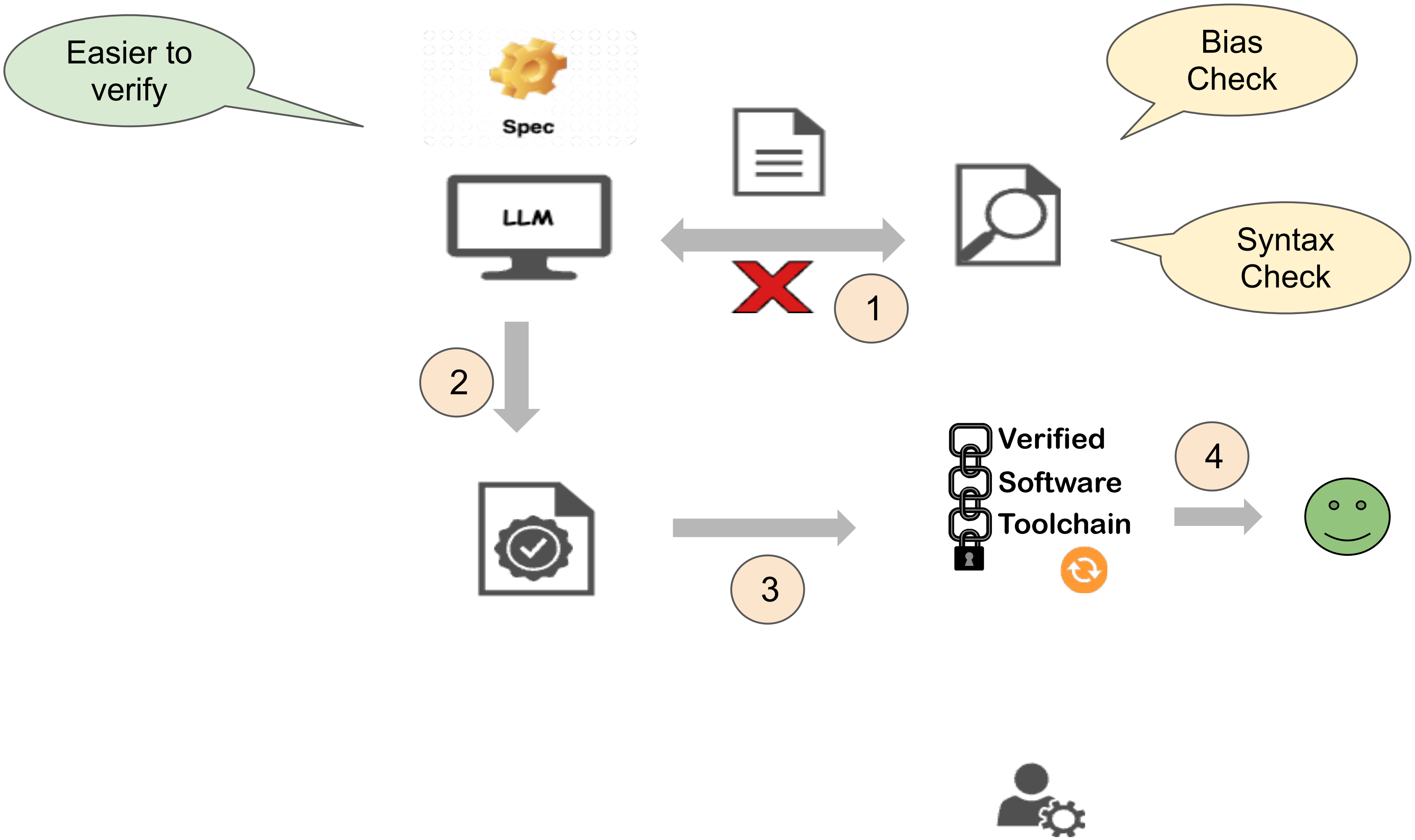
SYNVER: Workflow



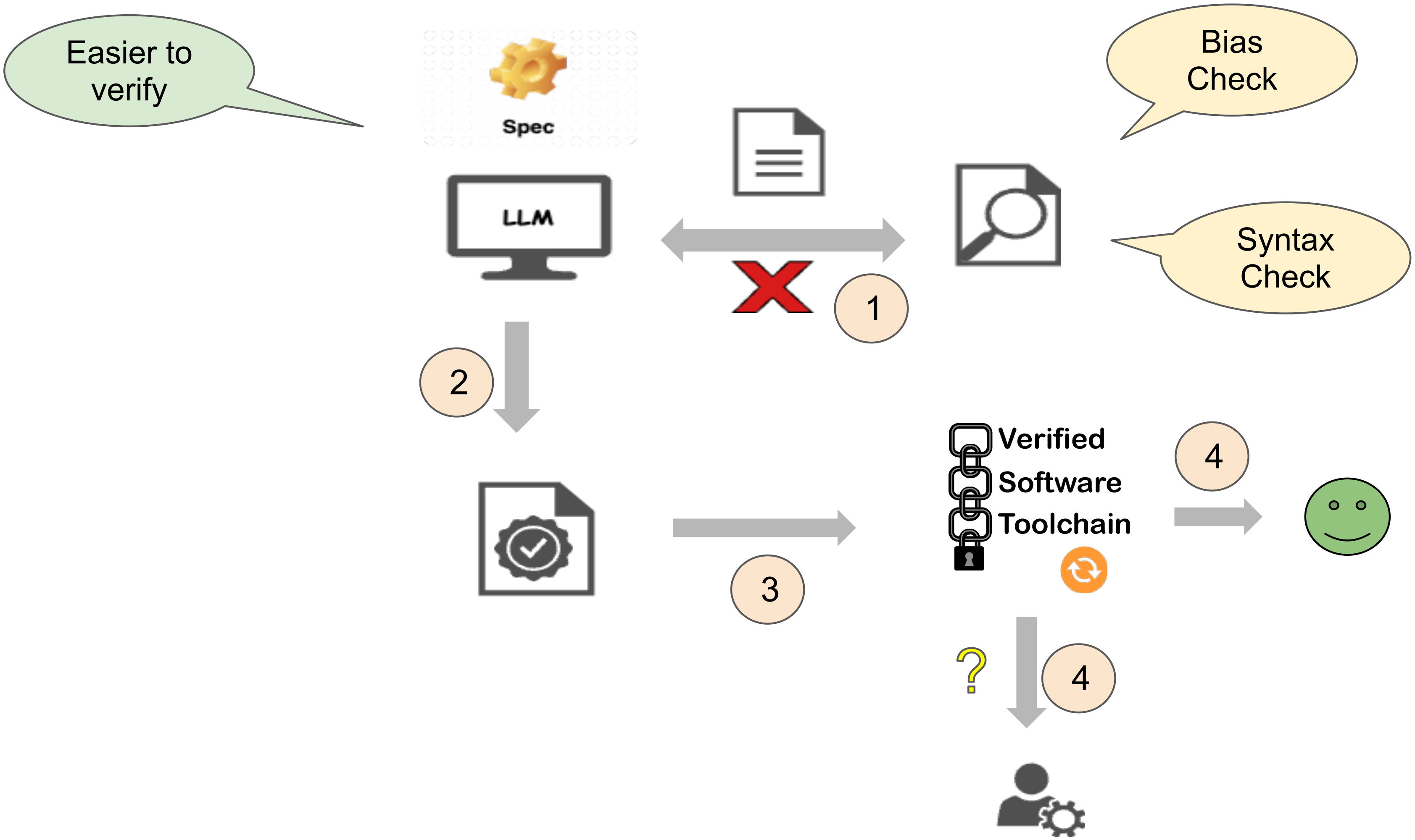
SYNVER: Workflow



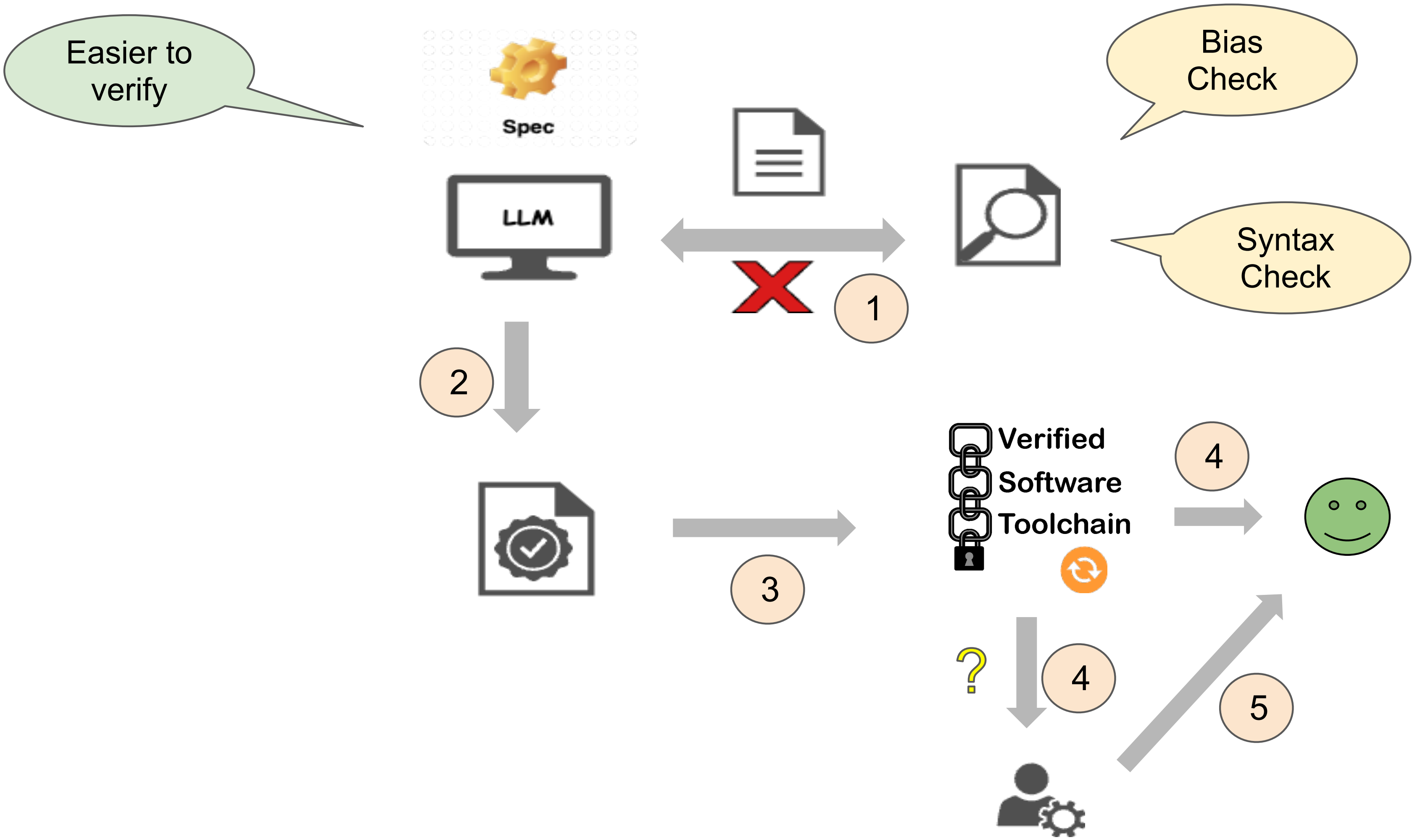
SYNVER: Workflow



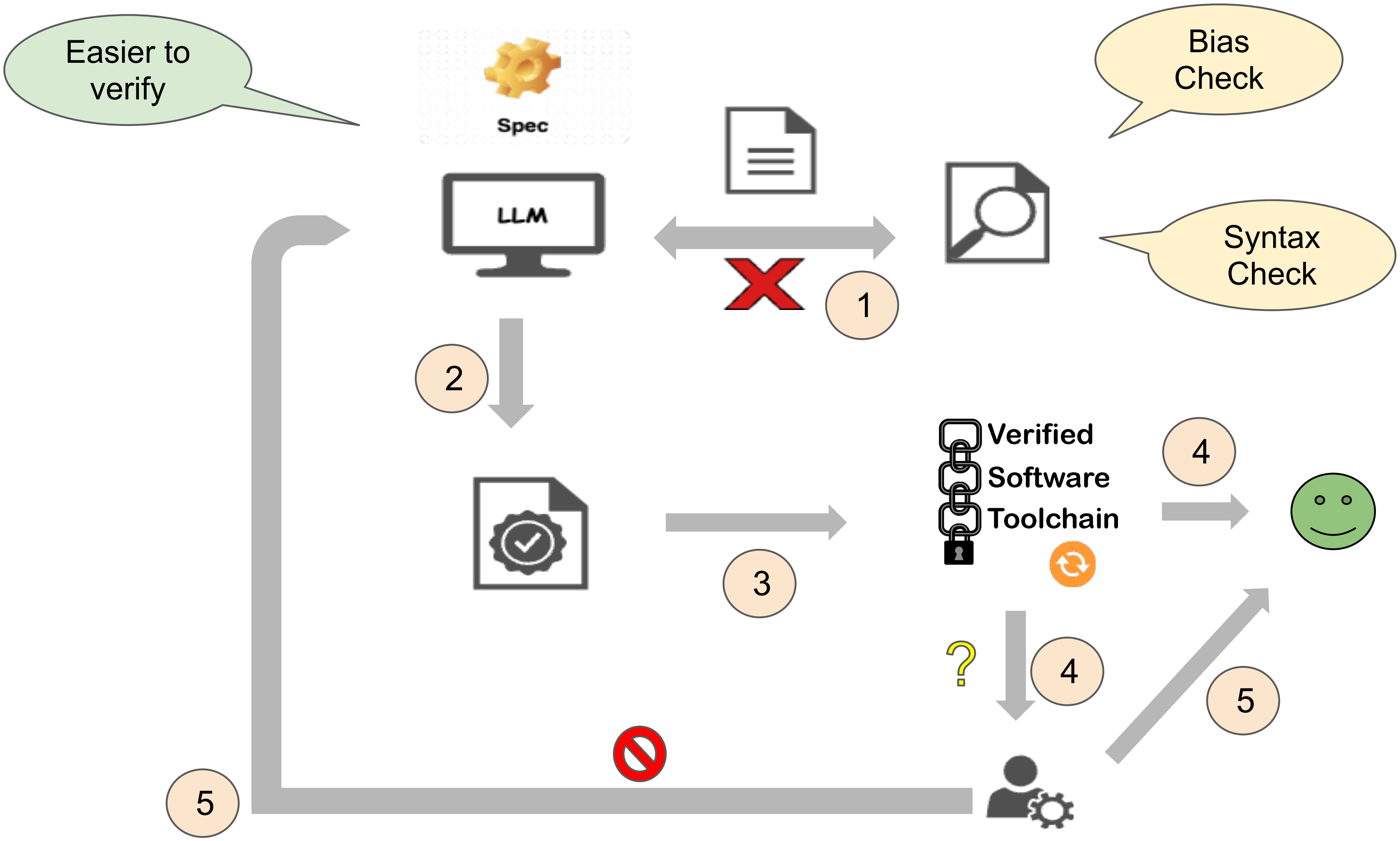
SYNVER: Workflow



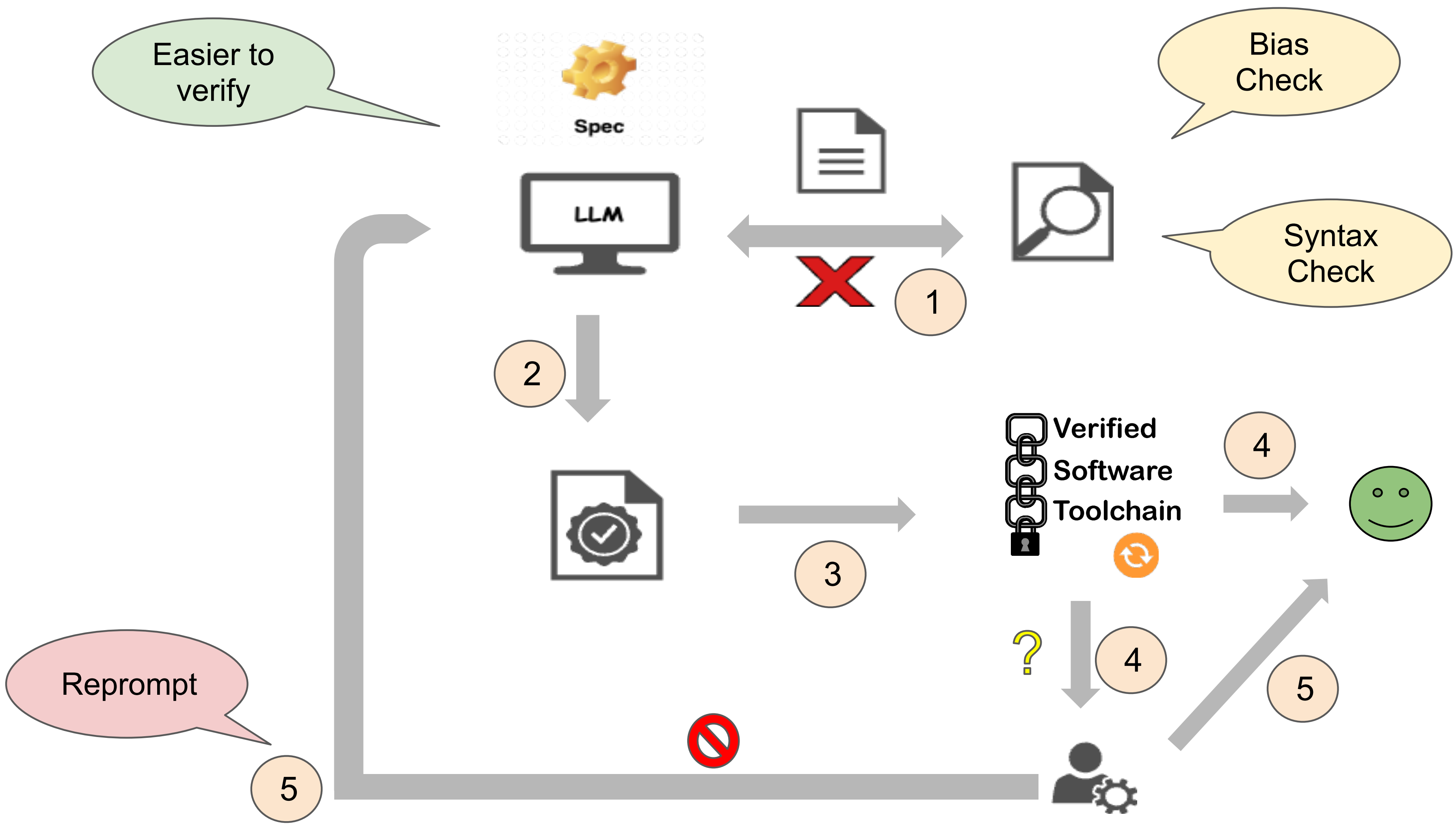
SYNVER: Workflow



SYNVER: Workflow



SYNVER: Workflow



Why incorporate biases?

Why incorporate biases?

↓

Generate programs that are easier to verify

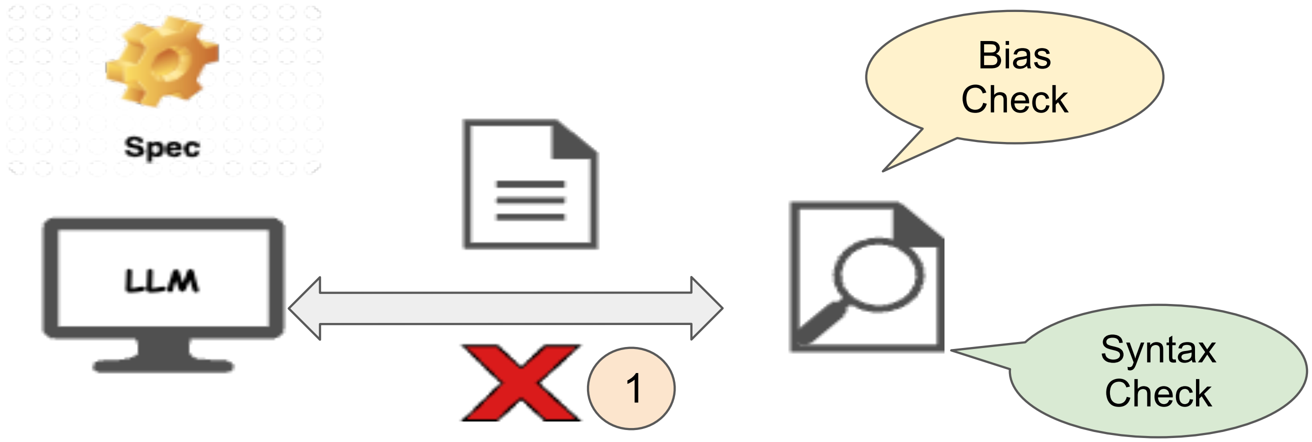
Why incorporate biases?

Generate programs that are easier to verify

✖ Infer novel specifications and verify them

Biases: No loops and novel helper functions

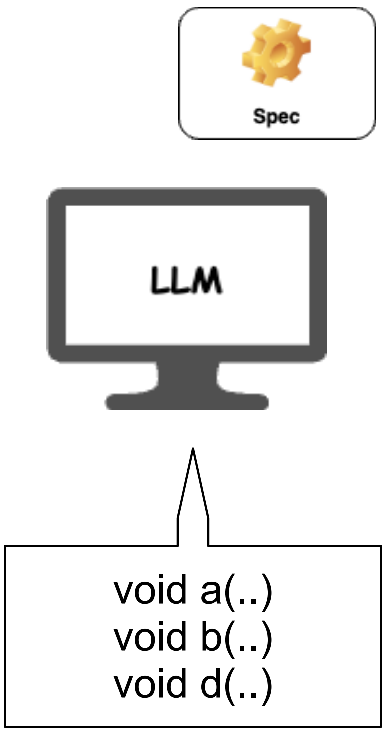
Biases: No loops and novel helper functions



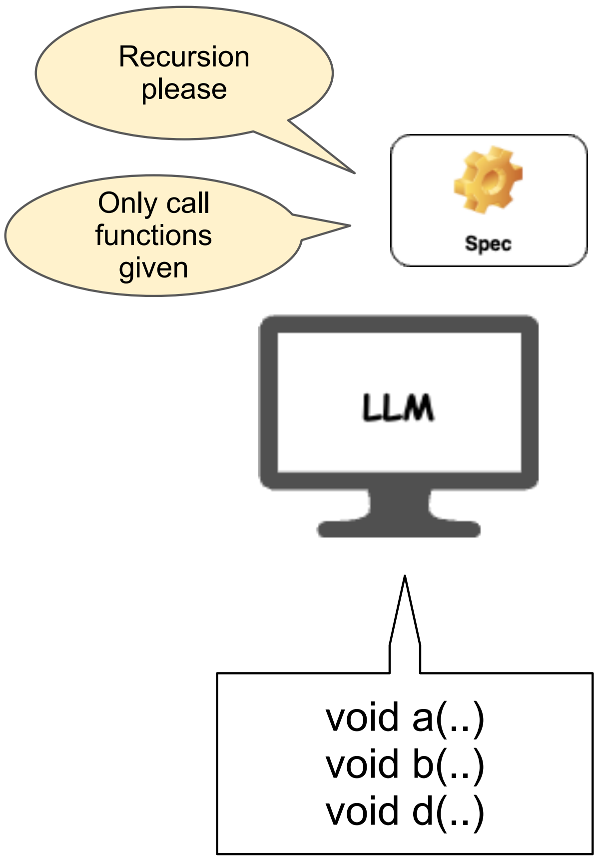
Biases: No loops and novel helper functions

Biases: No loops and novel helper functions

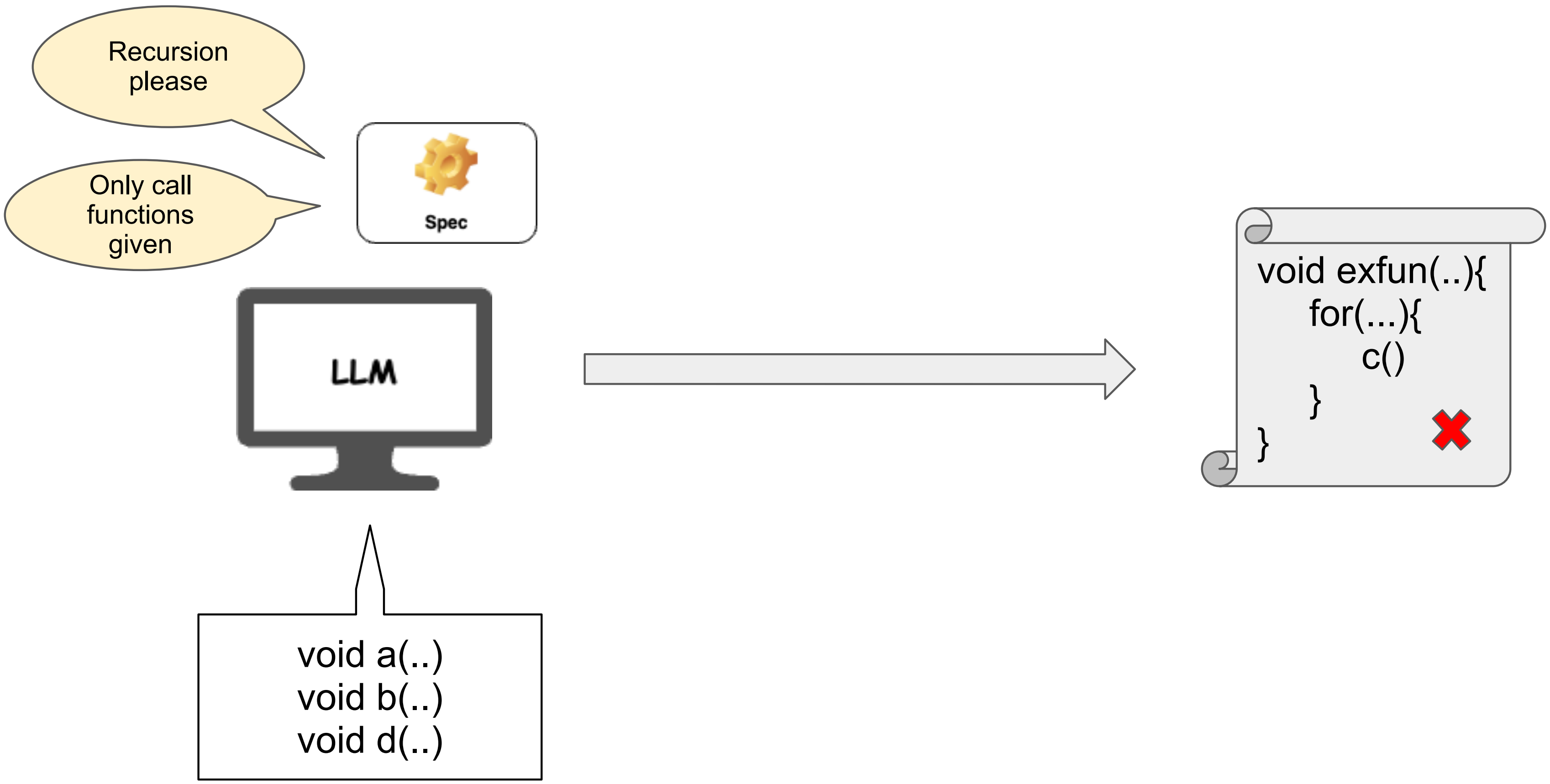
Biases: No loops and novel helper functions



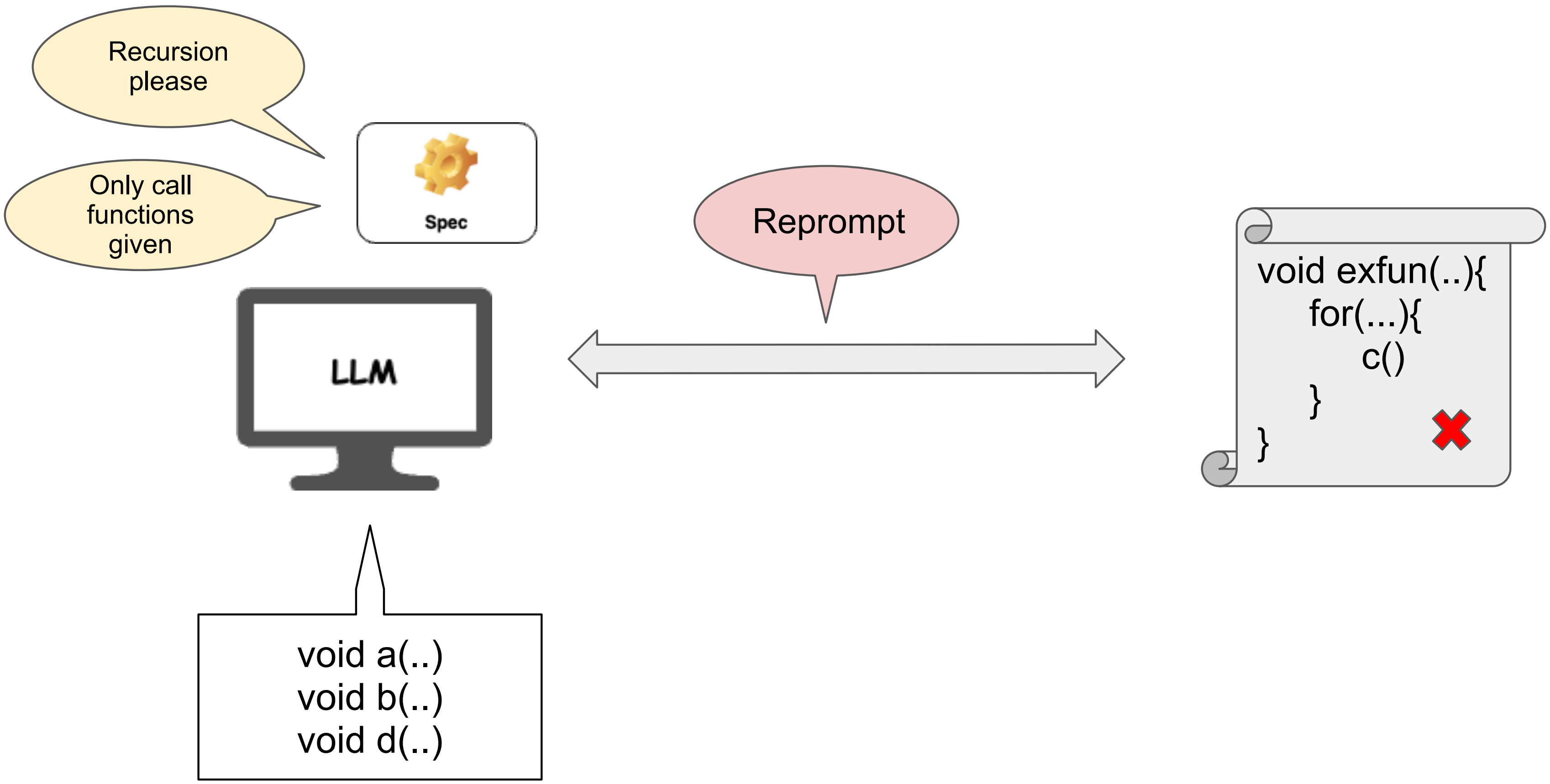
Biases: No loops and novel helper functions



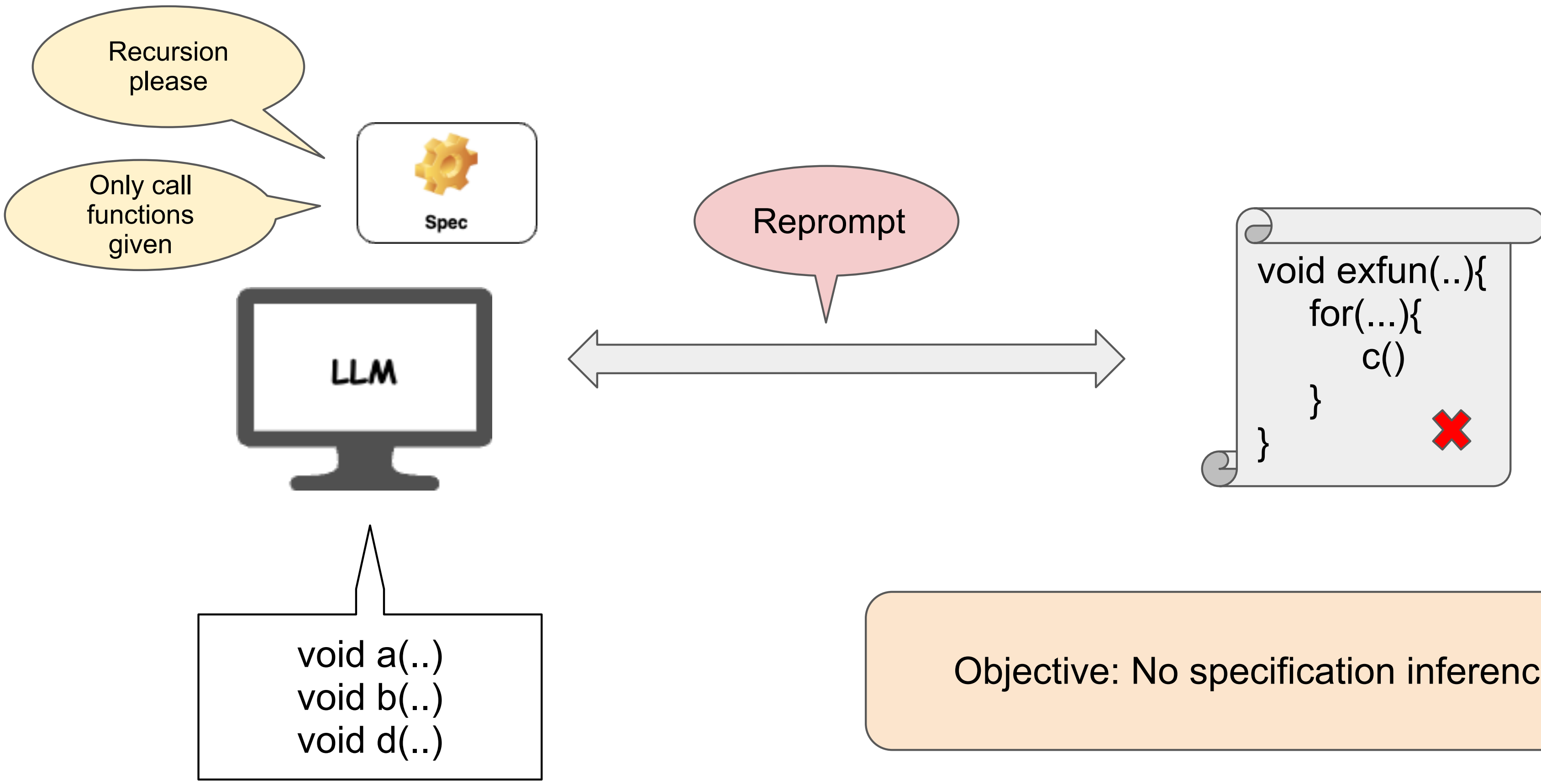
Biases: No loops and novel helper functions



Biases: No loops and novel helper functions



Biases: No loops and novel helper functions



Objective: No specification inference

Writing automation friendly specifications

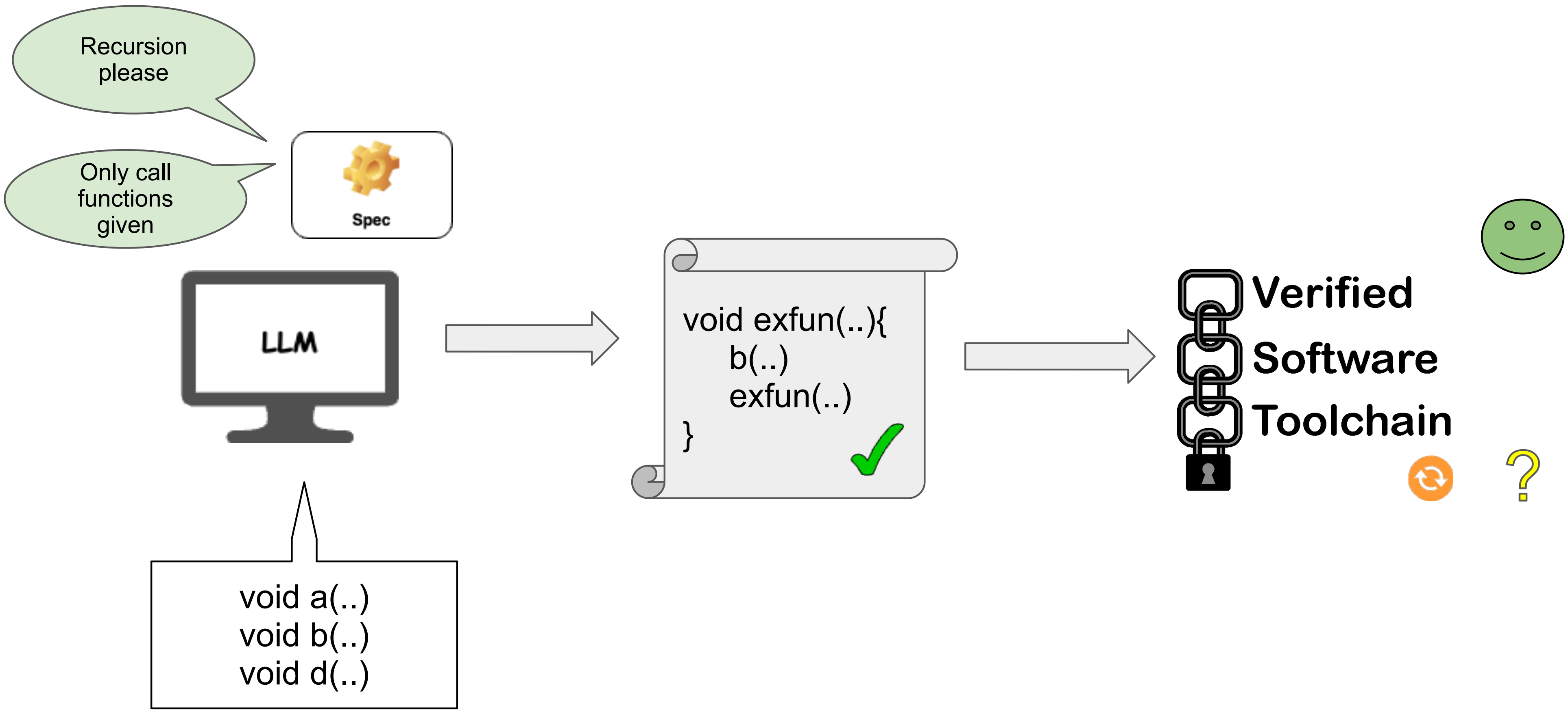
- Equivalent specifications amenable to automated reasoning
- Express with predicates like Forall, Forall2
- Check if an array is sorted in ascending order

$$\forall i, j \ 0 \le i \ < len \rightarrow i < j < len \rightarrow a[i] \le a[j]$$

$$Forall2 \ \le \ (sublist \ 0 \ (len \ - 1) \ a) \ (sublist \ 1 \ len \ a)$$

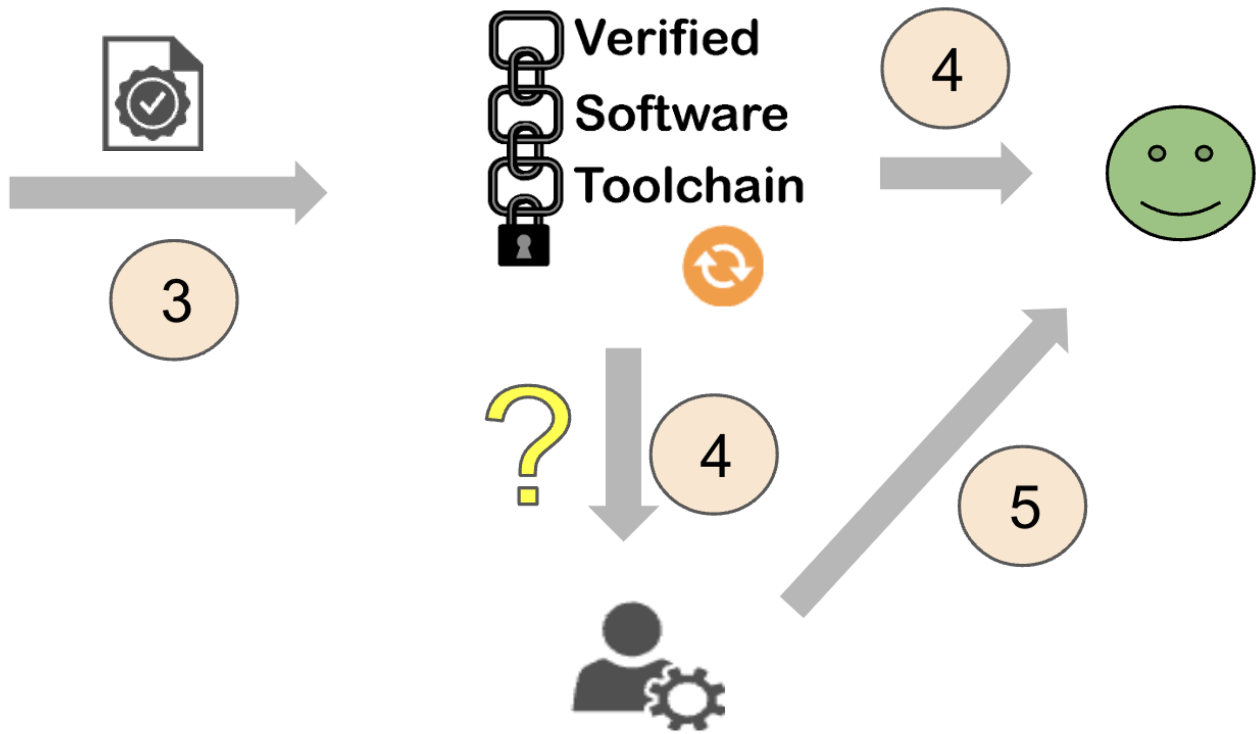
list_solve

Biases checked



SepAuto: Forward symbolic execution

SepAuto: Forward symbolic execution



SepAuto: Forward symbolic execution

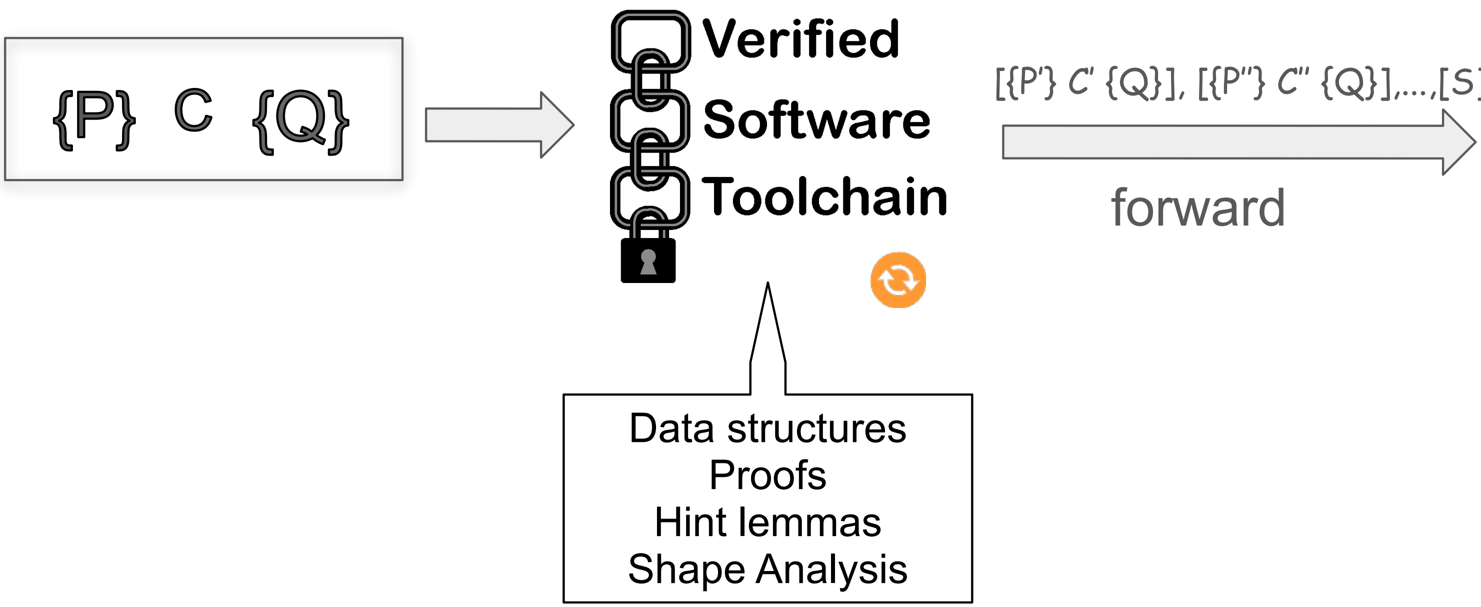
SepAuto: Forward symbolic execution

{P} C {Q}

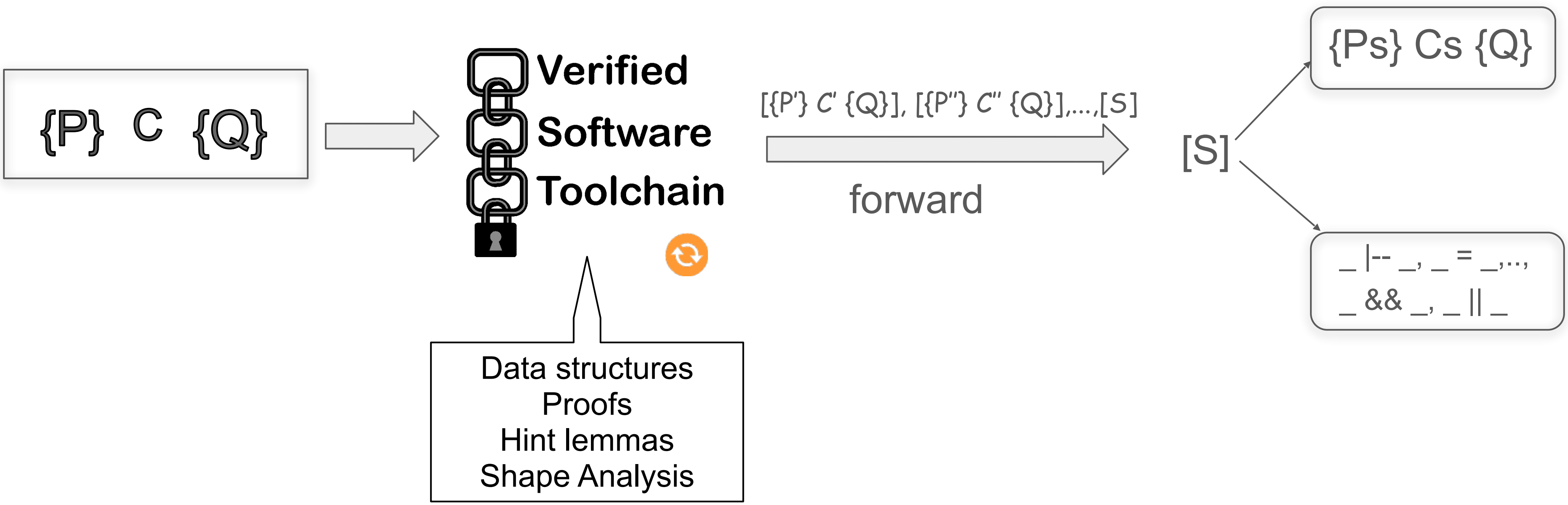
Verified Software Toolchain

Data structures
Proofs
Hint lemmas
Shape Analysis

SepAuto: Forward symbolic execution



SepAuto: Forward symbolic execution



SepAuto: if statements

SepAuto: if statements

{Ps} if b then c else d; e {Q}

SepAuto: if statements

forward_if ✖

{Ps} if b then c else d; e {Q}

SepAuto: if statements

forward_if ✖

{Ps} if b then c else d; e {Q}

Requires a joint post-condition

SepAuto: if statements

forward_if ✖

{Ps} if b then c else d; e {Q}

Requires a joint post-condition

No specification inference

SepAuto: if statements

forward_if ✖

{Ps} if b then c else d; e {Q}

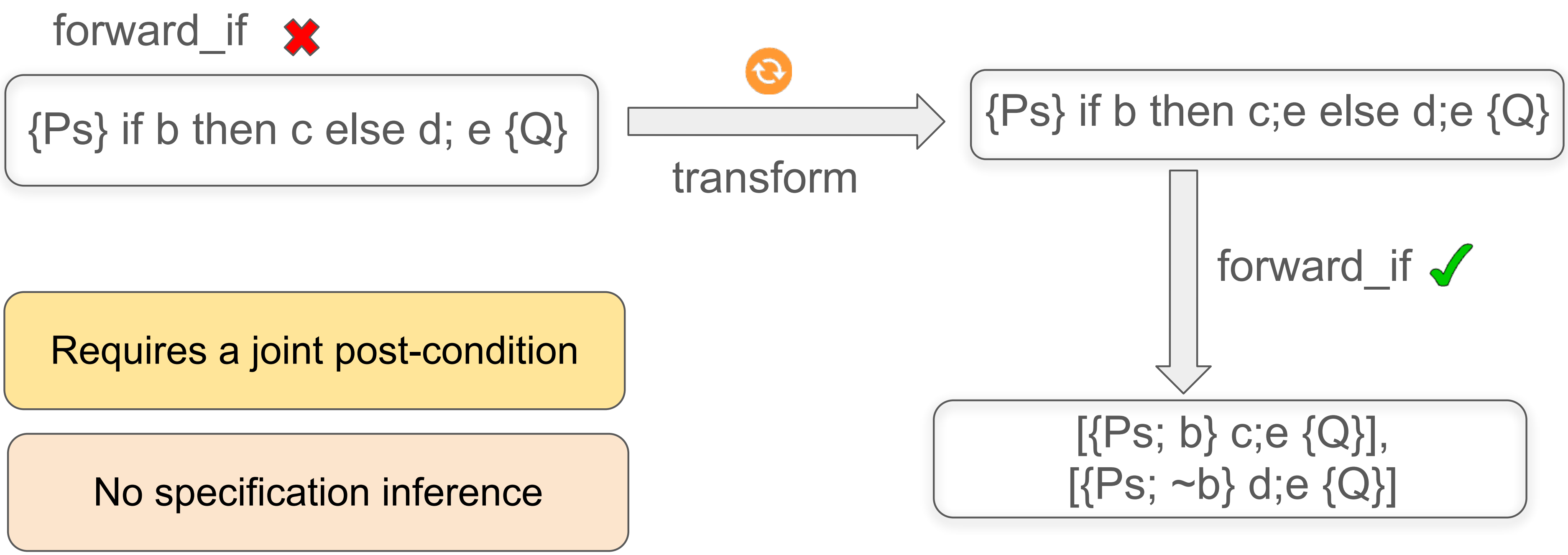
transform

{Ps} if b then c;e else d;e {Q}

Requires a joint post-condition

No specification inference

SepAuto: if statements



SepAuto: Function calls

SepAuto: Function calls

{Ps} f(....); e {Q}

SepAuto: Function calls

forward_call(....)

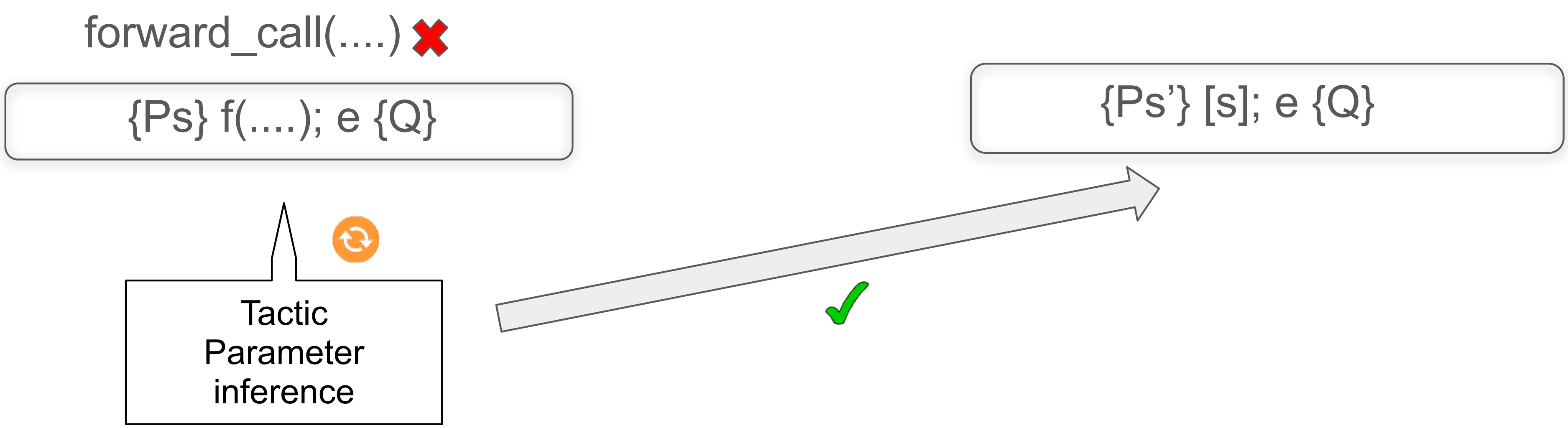
{Ps} f(....); e {Q}

SepAuto: Function calls

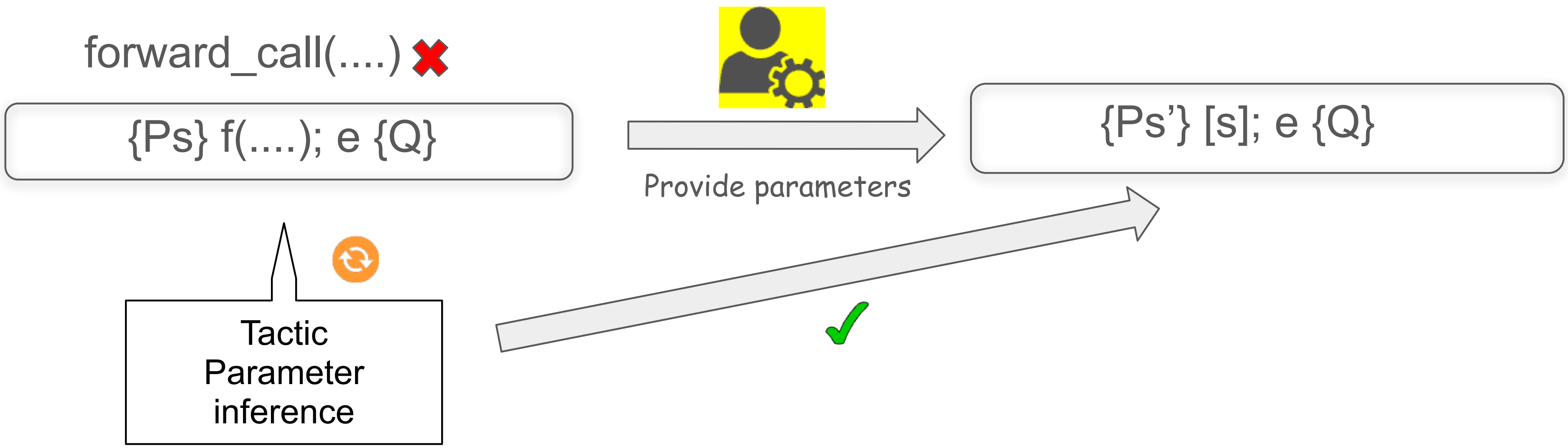
forward_call(....) ✖

{Ps} f(....); e {Q}

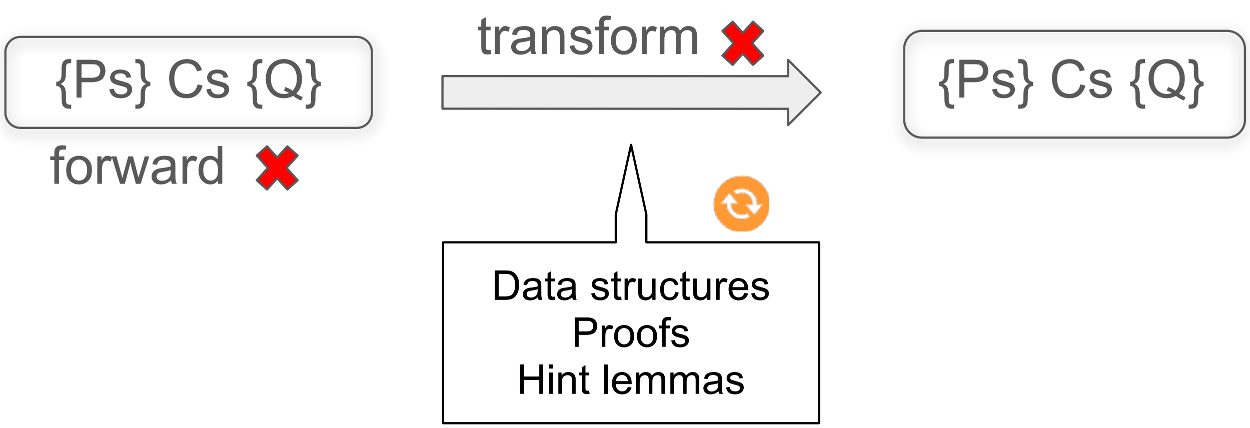
SepAuto: Function calls



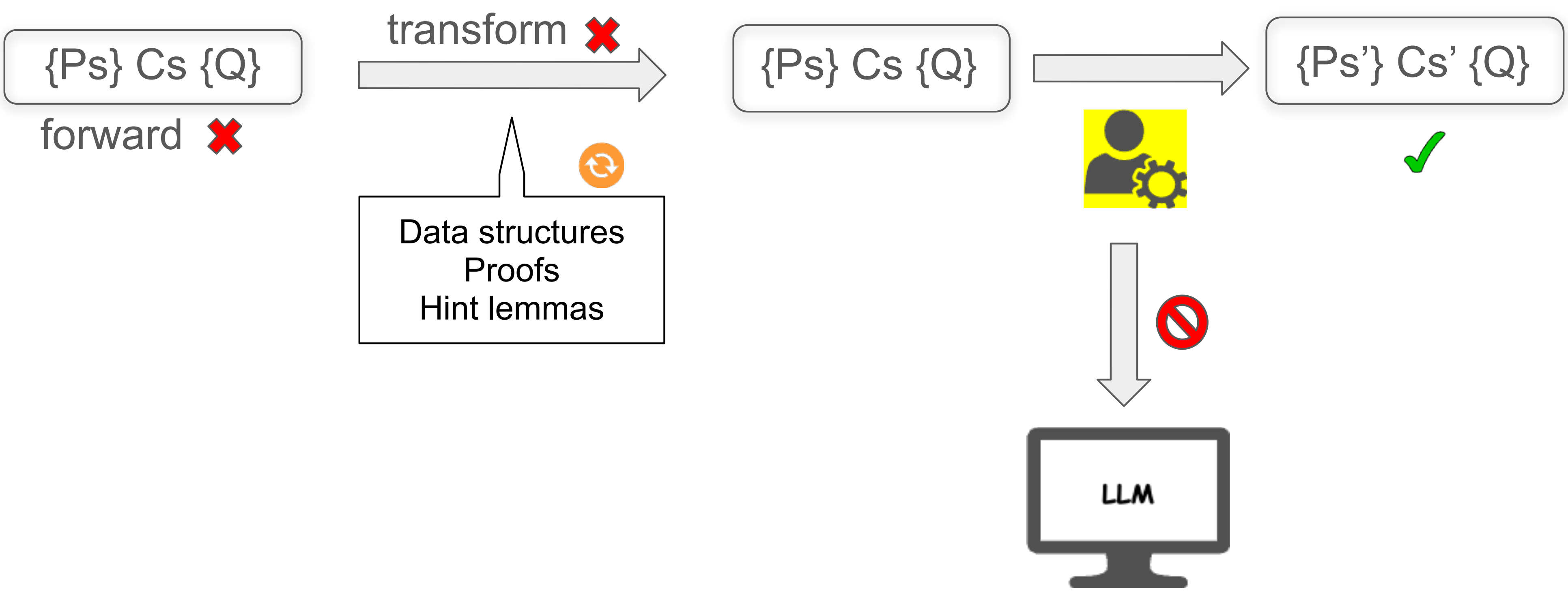
SepAuto: Function calls



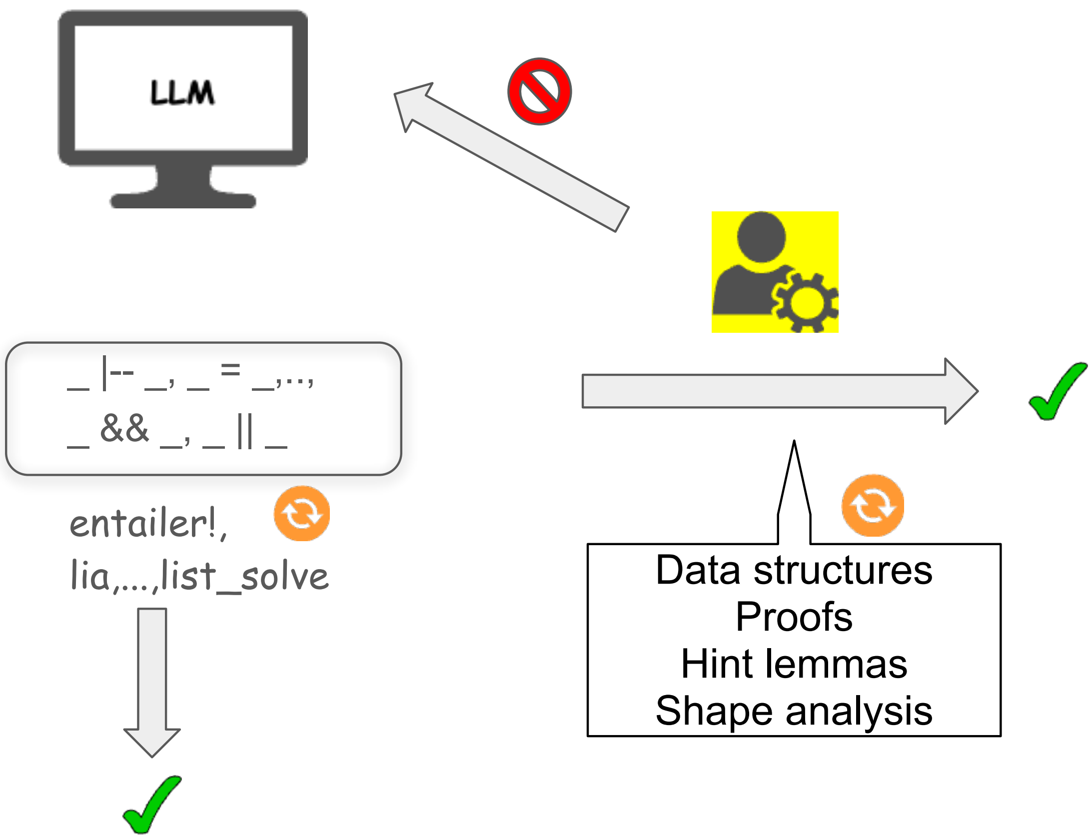
SepAuto: When forward fails



SepAuto: When forward fails



SepAuto: Non-Hoare Triples



Outline

- General purpose synthesizer for C programs
- Generates easily verifiable code through prompt design
- SepAuto - tactic on top of VST to automate proofs
- Experimental Setup
- Results

Outline

- General purpose synthesizer for C programs
- Generates easily verifiable code through prompt design
- SepAuto - tactic on top of VST to automate proofs
- Experimental Setup
- Results

Outline

- General purpose synthesizer for C programs
- Generates easily verifiable code through prompt design
- SepAuto - tactic on top of VST to automate proofs
- Experimental Setup
- Results

Experimental Setup

- Benchmark of 50 programs across basic [1], SL [2] and API [3]
- Multiple types supported for programs
- GPT-4o as the LLM
- Prompted with five components

[1] Md Rakib Hossain Misu, Cristina V. Lopes, Iris Ma, and James Noble. FSE (2024), 812–835. https://doi.org/10.1145/3643763

[2] Nadia Polikarpova and Ilya Sergey. 2019. Structuring the synthesis of heap-manipulating programs. POPL (2019), 72:1–72:30. https://doi.org/10.1145/3290385

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)

Results

- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm

Benchmark	**Bias**	**Types**	**Recursion**	**SC**	**Attempts**
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm

Benchmark	Bias	Types	Recursion	SC	Attempts
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm

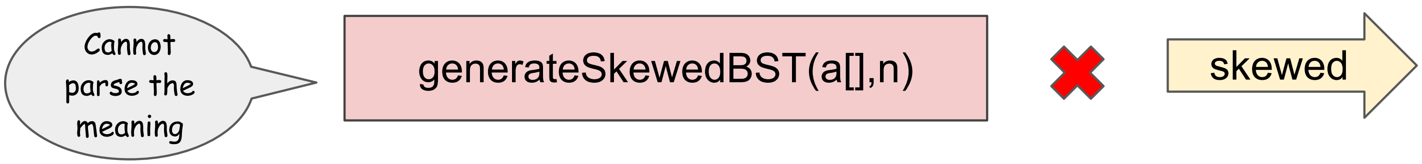
Cannot parse the meaning

generateSkewedBST(a[],n) ✖

Benchmark	Bias	Types	Recursion	SC	Attempts
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

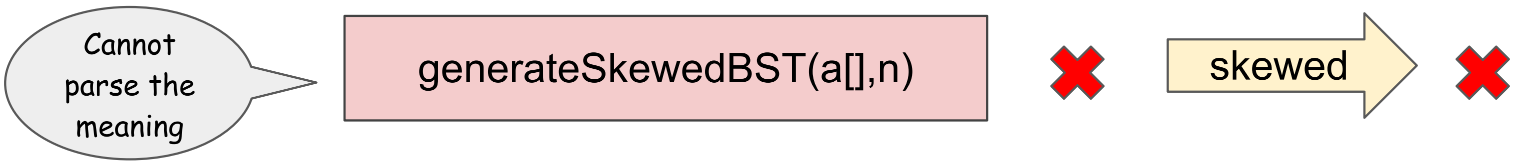
- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm



Benchmark	**Bias**	**Types**	**Recursion**	**SC**	**Attempts**
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

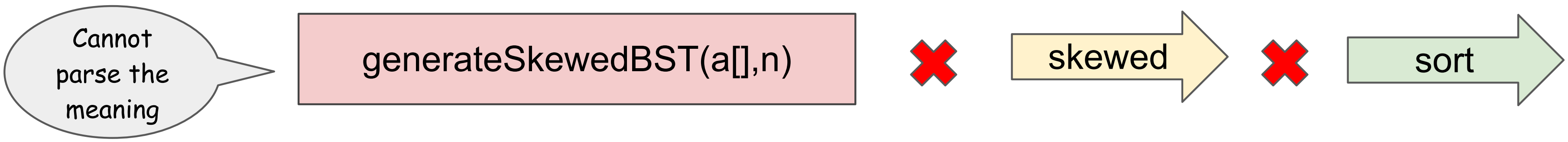
- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm



Benchmark	Bias	Types	Recursion	SC	Attempts
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

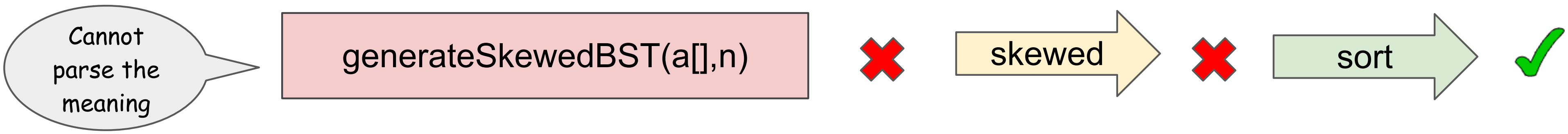
- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm



Benchmark	Bias	Types	Recursion	SC	Attempts
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

- GPT-4o produced correct code in the first try for 49/50 programs
- Incorrectly generated program(s) required detailed information about the algorithm



Benchmark	Bias	Types	Recursion	SC	Attempts
arrayLookUpTable	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
completegraphAdjMatrix	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
deleteBST	✔	$\{\mathbb{Z}\}$	✖	0	1
generateSkewedBST	✔	$\{\mathbb{Z}\}$	✖	2	3
addLastVoid	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	0	1
stackAPILinkedList	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✖	1	1

Results

- Verifies mostly automatically
- Side conditions primarily generated for complex specifications

SL

listDelEnd	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	1	1
listLookup	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
listCopy	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
listInsertEnd	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
listFilter	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
listAndArraySame	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
listAdd1	✔	$\{\mathbb{Z}\}$	✔	0	1
bstInit	✔	$\{\mathbb{Z}\}$	✖	0	1
bstFree	✔	$\{\mathbb{Z}\}$	✔	0	1
bstLookup	✔	$\{\mathbb{Z}\}$	✔	0	1
bstInsert	✔	$\{\mathbb{Z}\}$	✔	0	1
bstMinNode	✔	$\{\mathbb{Z}\}$	✔	2	1

Basic

checkZ	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	1	1
consecNumbers	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
nIsGreater	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
firstodd	✔	$\{\mathbb{Z}\}$	✔	3	1
arrayMemberValue	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
isOddAtIndexOdd	✔	$\{\mathbb{Z}\}$	✔	3	1
lastelempos	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	2	1
arrayModifyWithOneValue	✔	$\{\mathbb{Z}, \mathbb{R}, \mathbb{S}\}$	✔	0	1
compareTwoArrays	✔	$\{\mathbb{Z}, \mathbb{S}\}$	✔	0	1
addArrayElementsBy1	✔	$\{\mathbb{Z}\}$	✔	0	1

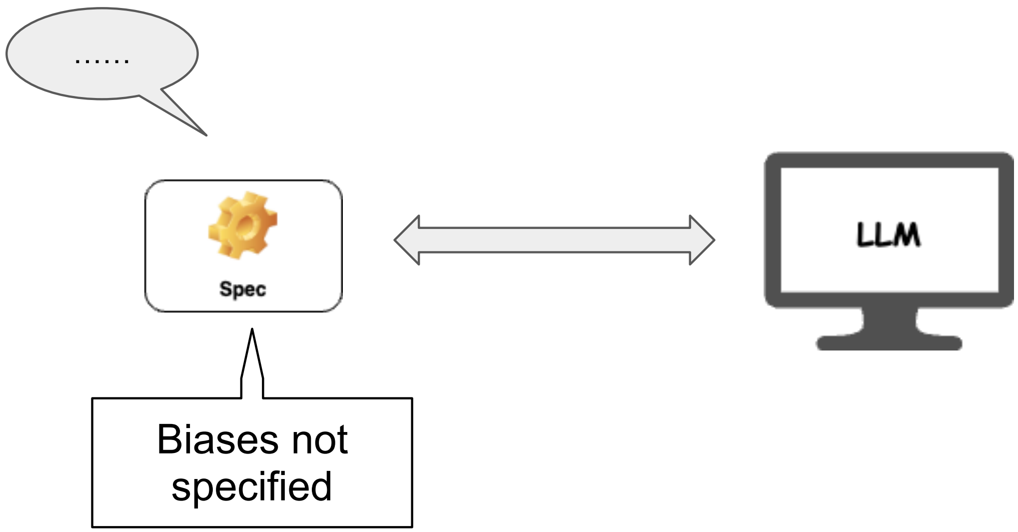
RQ1: Are the biases necessary?

RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?

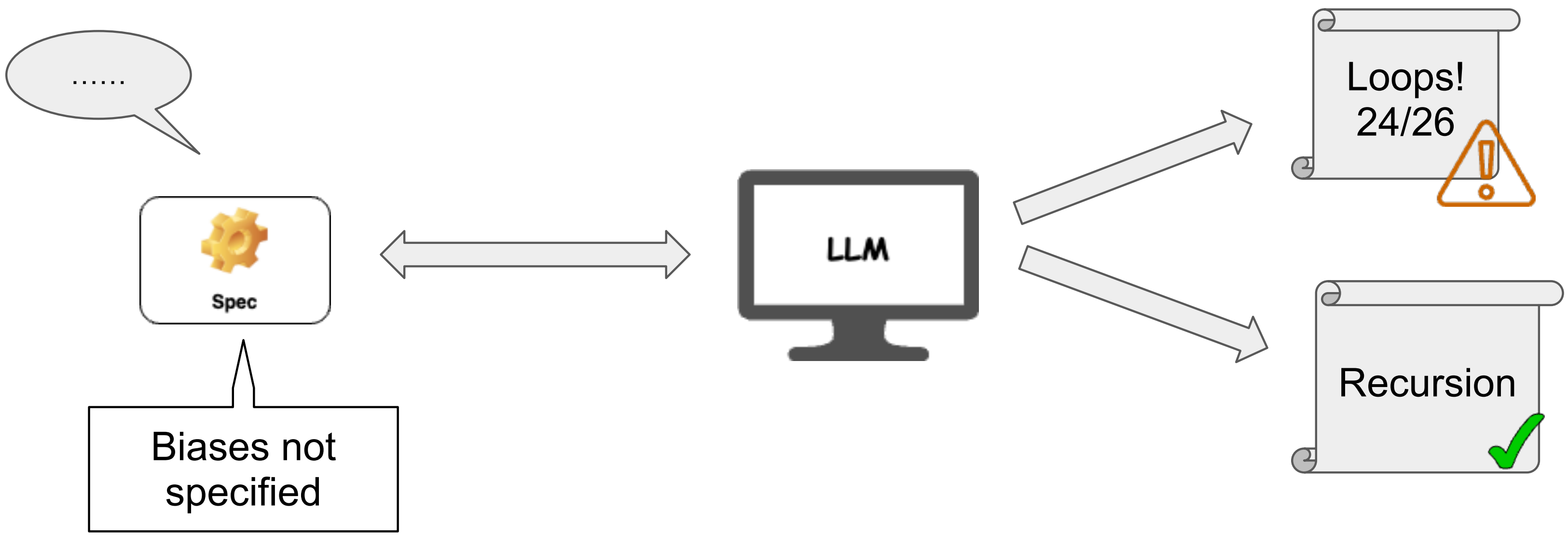
RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?



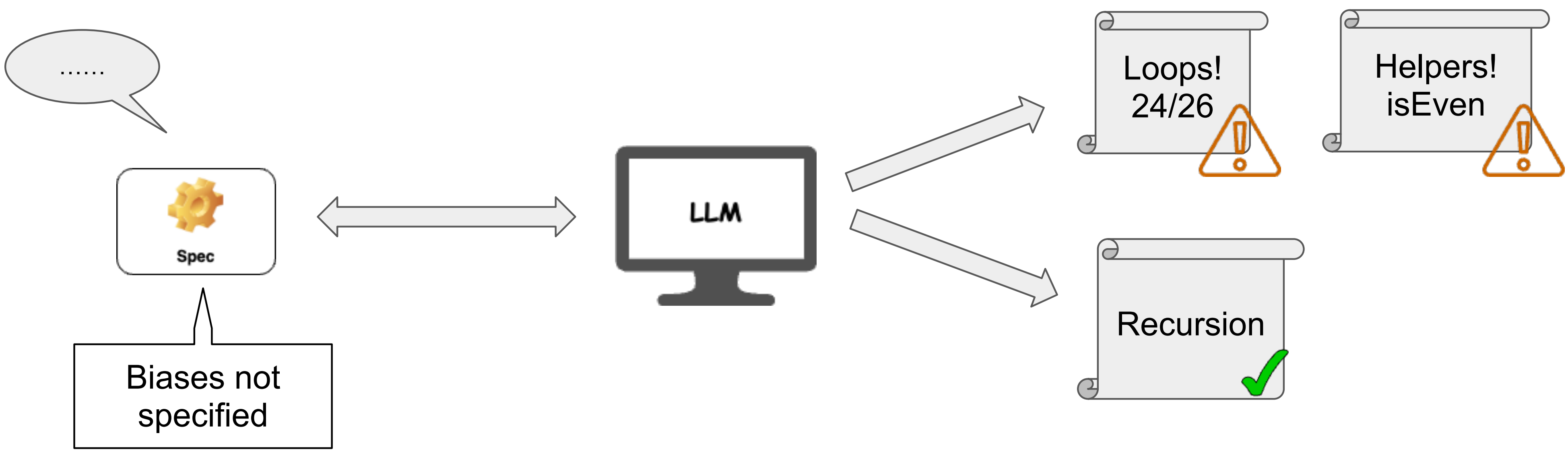
RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?



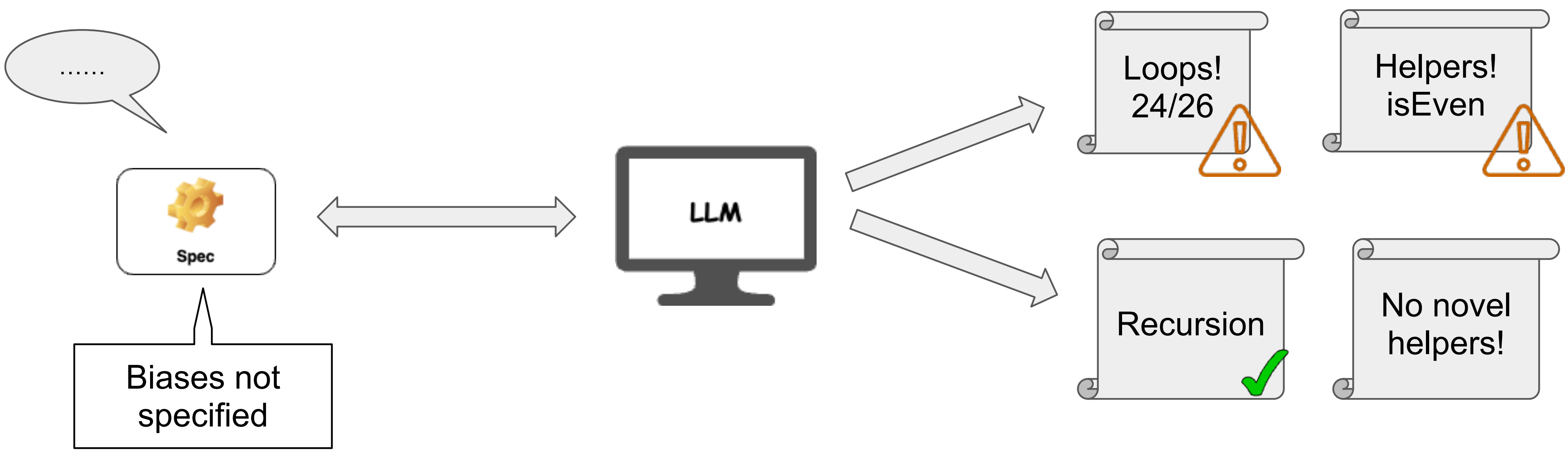
RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?



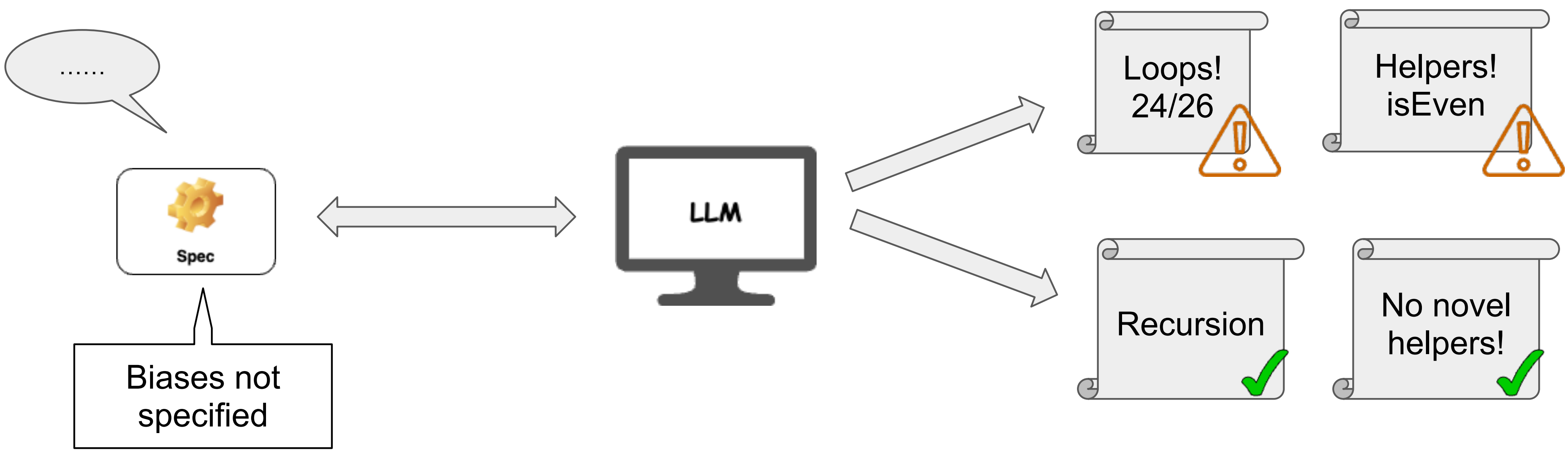
RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?



RQ1: Are the biases necessary?

Without additional biases, what programs do LLMs tend to generate?



RQ2: Where does SepAuto Fail?

Unable to infer relation between two variables

```
H8 :   ind >= 0
H6 :   ind < Zlength contents
H9 :   Znth ind contents <> ele
Hform :  Znth vret contents = ele /\
         Forall (fun x : Z => x <> ele)
           (sublist (vret + 1) (ind - 1 + 1) contents)
```

```
(1/1)
Forall (fun x : Z => x <> ele)
  (sublist (vret + 1) (ind + 1) contents)
```

RQ2: Where does SepAuto Fail?

Unable to infer relation between two variables

```
H8 :   ind >= 0
H6 :   ind < Zlength contents
H9 :   Znth ind contents <> ele
Hform :  Znth vret contents = ele /\
         Forall (fun x : Z => x <> ele)
           (sublist (vret + 1) (ind - 1 + 1) contents)
```

```
(1/1)
Forall (fun x : Z => x <> ele)
  (sublist (vret + 1) (ind + 1) contents)
```

RQ2: Where does SepAuto Fail?

Unable to infer relation between two variables

```
H8 :   ind >= 0
H6 :   ind < Zlength contents
H9 :   Znth ind contents <> ele
Hform :  Znth vret contents = ele /\
         Forall (fun x : Z => x <> ele)
           (sublist (vret + 1) (ind - 1 + 1) contents)
```

```
(1/1)
Forall (fun x : Z => x <> ele)
  (sublist (vret + 1) (ind + 1) contents)
```

destruct vret
>=? ind

RQ2: Where does SepAuto Fail?

Unable to infer relation between two variables

```
H8 :   ind >= 0
H6 :   ind < Zlength contents
H9 :   Znth ind contents <> ele
Hform :  Znth vret contents = ele /\
        Forall (fun x : Z => x <> ele)
          (sublist (vret + 1) (ind - 1 + 1) contents)
```

```
(1/1)
Forall (fun x : Z => x <> ele)
  (sublist (vret + 1) (ind + 1) contents)
```

destruct vret
>=? ind

RQ2: Where does SepAuto Fail?

Unable to infer relation between two variables

```
H8 :   ind >= 0
H6 :   ind < Zlength contents
H9 :   Znth ind contents <> ele
Hform :  Znth vret contents = ele /\
         Forall (fun x : Z => x <> ele)
           (sublist (vret + 1) (ind - 1 + 1) contents)
```

```
(1/1)
Forall (fun x : Z => x <> ele)
  (sublist (vret + 1) (ind + 1) contents)
```

destruct vret >=? ind

sublist_split

RQ2: Where does SepAuto Fail?

Unable to pose and prove non-trivial premises

```
H :  p <> nullval
H4 :  y = nullval <-> hs = []
```

```
(1/1)
listrep hs y
|-- listrep (sublist 1 (Zlength (h0 :: hs)) (h0 :: hs)) y
```

RQ2: Where does SepAuto Fail?

Unable to pose and prove non-trivial premises

```
H :  p <> nullval
H4 :  y = nullval <-> hs = []
```

```
(1/1)
listrep hs y
|-- listrep (sublist 1 (Zlength (h0 :: hs)) (h0 :: hs)) y
```

RQ2: Where does SepAuto Fail?

Automation is unsupported for the wand operator

```
H :  p <> nullval
H6 :  field_compatible t_struct_tree [] p
H7 :  value_fits t_struct_tree
       (Vint (Int.repr k),
        (Vint (Int.repr v), (nullval, pb)))
PNpb :  is_pointer_or_null pb
```

```
(1/1)
data_at Tsh t_struct_tree
  (Vint (Int.repr k),
   (Vint (Int.repr v), (nullval, pb))) p
  * tree_rep b pb
|-- EX a0 : val,
    !! (a0 <> nullval /\ p = a0) &&
    (tree_rep (inorderSuccessor (T E k v b)) a0
       * (tree_rep (inorderSuccessor (T E k v b)) a0 -*
          tree_rep (T E k v b) p))
```

RQ2.5: SepAuto versus other verifiers?

RQ2.5: SepAuto versus other verifiers?

Where do other verifiers fail in comparison to SepAuto?

RQ2.5: SepAuto versus other verifiers?

Where do other verifiers fail in comparison to SepAuto?

Frama C	VeriFast	RefinedC
Basic ☺	Expressive ☺	Expressive ☺
Dynamic allocation ⚠	Annotation inference ⚠	SL implication ☺
Separation Logic ⚠	Precise messaging ⚠	Precise messaging ☺
		Hint Lemmas ⚠

RQ2.5: SepAuto versus other verifiers?

Where do other verifiers fail in comparison to SepAuto?

Frama C	VeriFast	RefinedC
Basic	Expressive	Expressive
Dynamic allocation	Annotation inference	SL implication
Separation Logic	Precise messaging	Precise messaging
		Hint Lemmas

```
void addHelper(struct node* n, int x)
  //@ requires n!= 0 &*& lseg(n, 0, ?vs);
  //@ ensures lseg(n, 0, append(vs, cons(x, nil)));
{
  //@ open lseg(n, 0, vs);
  if(n->next == 0) {
    // @ open lseg(0, 0, _);
    struct node* nn = create_node(0, x);
    n->next = nn;
  } else {
    addHelper(n->next, x);
  }
}
```

RQ2.5: SepAuto versus other verifiers?

Where do other verifiers fail in comparison to SepAuto?

Frama C	VeriFast	RefinedC
Basic	Expressive	Expressive
Dynamic allocation	Annotation inference	SL implication
Separation Logic	Precise messaging	Precise messaging
		Hint Lemmas

```
void addHelper(struct node* n, int x)
  //@ requires n!= 0 &*& lseg(n, 0, ?vs);
  //@ ensures lseg(n, 0, append(vs, cons(x, nil)));
{
  //@ open lseg(n, 0, vs);
  if(n->next == 0) {
    // @ open lseg(0, 0, _);
    struct node* nn = create_node(0, x);
    n->next = nn;
  } else {
    addHelper(n->next, x);
  }
}
```

RQ2.5: SepAuto versus other verifiers?

Where do other verifiers fail in comparison to SepAuto?

Frama C	VeriFast	RefinedC
Basic	Expressive	Expressive
Dynamic allocation	Annotation inference	SL implication
Separation Logic	Precise messaging	Precise messaging
		Hint Lemmas

```
void addHelper(struct node* n, int x)
  //@ requires n!= 0 &*& lseg(n, 0, ?vs);
  //@ ensures lseg(n, 0, append(vs, cons(x, nil)));
{
  //@ open lseg(n, 0, vs);
  if(n->next == 0) {
    // @ open lseg(0, 0, _);
    struct node* nn = create_node(0, x);
    n->next = nn;
  } else {
    addHelper(n->next, x);
  }
}
```

Postcondition ✗

RQ2.5: SepAuto versus other verifiers?

```
[[rc::parameters("l : {list Z}", "p : loc", "n : Z" )]]
[[rc::args("p @ &own<l @ list_t>", "n @ int<i32>")]]
[[rc::exists("b : bool")]]
[[rc::returns("b @ builtin_boolean")]]
[[rc::ensures("own p : l @ list_t", "{b ↔ n ∈ l}")]]
//[[rc::tactics("all: try set_solver.")]]
bool member_rec (list_t *p, int k) {
  if (*p == NULL) {
    return false;
  }
  int head = (*p)->val;
  if (head == k) {
    return true;
  }
  return member_rec(&(*p)->next, k);
}
```

RQ2.5: SepAuto versus other verifiers?

```
[[rc::parameters("l : {list Z}", "p : loc", "n : Z" )]]
[[rc::args("p @ &own<l @ list_t>", "n @ int<i32>")]]
[[rc::exists("b : bool")]]
[[rc::returns("b @ builtin_boolean")]]
[[rc::ensures("own p : l @ list_t", "{b ↔ n ∈ l}")]]
//[[rc::tactics("all: try set_solver.")]]
bool member_rec (list_t *p, int k) {
  if (*p == NULL) {
    return false;
  }
  int head = (*p)->val;
  if (head == k) {
    return true;
  }
  return member_rec(&(*p)->next, k);
}
```

RQ2.5: SepAuto versus other verifiers?

```
[[rc::parameters("l : {list Z}", "p : loc", "n : Z" )]]
[[rc::args("p @ &own<l @ list_t>", "n @ int<i32>")]]
[[rc::exists("b : bool")]]
[[rc::returns("b @ builtin_boolean")]]
[[rc::ensures("own p : l @ list_t", "{b ↔ n ∈ l}")]]
//[[rc::tactics("all: try set_solver.")]]
bool member_rec (list_t *p, int k) {
  if (*p == NULL) {
    return false;
  }
  int head = (*p)->val;
  if (head == k) {
    return true;
  }
  return member_rec(&(*p)->next, k);
}
```

RQ2.5: SepAuto versus other verifiers?

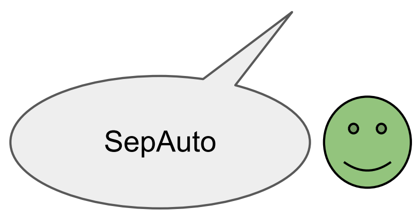
```
[[rc::parameters("l : {list Z}", "p : loc", "n : Z" )]]
[[rc::args("p @ &own<l @ list_t>", "n @ int<i32>")]]
[[rc::exists("b : bool")]]
[[rc::returns("b @ builtin_boolean")]]
[[rc::ensures("own p : l @ list_t", "{b ↔ n ∈ l}")]]
//[[rc::tactics("all: try set_solver.")]]
bool member_rec (list_t *p, int k) {
  if (*p == NULL) {
    return false;
  }
  int head = (*p)->val;
  if (head == k) {
    return true;
  }
  return member_rec(&(*p)->next, k);
}
```

```
HCASE :  CASE_DISTINCTION_INFO
          (optional == ... : n :: x1 ≠ [])
          [loc_38; loc_38]
HCASE0 :  CASE_DISTINCTION_INFO
           (if false, false) [loc_38]
H1 :  n ≤ max_int i32
H0, H2 :  min_int i32 ≤ n
H3 :  n ≤ max_int i32
HCASE1 :  CASE_DISTINCTION_INFO
           (if bool_decide (n = n), true)
           [loc_21]
________________________________________
(1/3)
True ↔ n ∈ n :: x1
________________________________________
(2/3)
x' ↔ n ∈ x0 :: x1
________________________________________
(3/3)
False ↔ n ∈ []
```

RQ2.5: SepAuto versus other verifiers?

```
[[rc::parameters("l : {list Z}", "p : loc", "n : Z" )]]
[[rc::args("p @ &own<l @ list_t>", "n @ int<i32>")]]
[[rc::exists("b : bool")]]
[[rc::returns("b @ builtin_boolean")]]
[[rc::ensures("own p : l @ list_t", "{b ↔ n ∈ l}")]]
//[[rc::tactics("all: try set_solver.")]]
bool member_rec (list_t *p, int k) {
  if (*p == NULL) {
    return false;
  }
  int head = (*p)->val;
  if (head == k) {
    return true;
  }
  return member_rec(&(*p)->next, k);
}
```

```
HCASE :  CASE_DISTINCTION_INFO
          (optional == ... : n :: x1 ≠ [])
          [loc_38; loc_38]
HCASE0 :  CASE_DISTINCTION_INFO
           (if false, false) [loc_38]
H1 :  n ≤ max_int i32
H0, H2 :  min_int i32 ≤ n
H3 :  n ≤ max_int i32
HCASE1 :  CASE_DISTINCTION_INFO
           (if bool_decide (n = n), true)
           [loc_21]
______________________________________
(1/3)
True ↔ n ∈ n :: x1
______________________________________
(2/3)
x' ↔ n ∈ x0 :: x1
______________________________________
(3/3)
False ↔ n ∈ []
```



RQ3: Do the components of the prompt matter?

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

Biases

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

Biases

English Description

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

- Biases
- English Description
- Meaningful Function Name

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

- Biases
- English Description
- Meaningful Function Name
- Formal Specification

RQ3: Do the components of the prompt matter?

How much does each component of the prompt contribute to generating the correct, desired programs?

- Biases
- English Description
- Meaningful Function Name
- Formal Specification
- I/O Examples

RQ3: Prompt Components

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ * listrep\ (filter\ fi2\ l)\ a$$

Biases

English

Name

Formal Spec.

I/O Examples

RQ3: Prompt Components

Biases ✓

English ✗

Name ✗

Formal Spec. ✓

I/O Examples ⊘

$$\begin{array}{l} Definition\ fi2 := fun\ x => x =? 2 \\ listrep\ l\ h\ \ * listrep\ (filter\ fi2\ l)\ a \end{array}$$

```
#include "treelistdef.c"
struct sll* abc(struct sll* h) {
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2) {
        h->next = abc(h->next);
        return h;
    }
    else {
        return abc(h->next);
    }
}
```

RQ3: Prompt Components

Biases ✓

English ✗

Name ✗

Formal Spec. ✓

I/O Examples

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ \ *\ listrep\ (filter\ fi2\ l)\ a$$

```c
#include "treelistdef.c"
struct sll* abc(struct sll* h) {
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2) {
        h->next = abc(h->next);
        return h;
    }
    else {
        return abc(h->next);
    }
}
```

$$listrep\ (filter\ fi2\ l)\ a$$ ✗

RQ3: Prompt Components

Biases	✓
English	✗
Name	✗
Formal Spec.	✓
I/O Examples	⊘

Basic: 17/19

SL: 12/23

$$Definition\ fi2 := fun\ x \Rightarrow x =?\ 2$$
$$listrep\ l\ h\ \ *\ listrep\ (filter\ fi2\ l)\ a$$

```c
#include "treelistdef.c"
struct sll* abc(struct sll* h) {
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2) {
        h->next = abc(h->next);
        return h;
    }
    else {
        return abc(h->next);
    }
}
```

$listrep\ (filter\ fi2\ l)\ a$ ✗

RQ3: Prompt Components

Biases	✓
English	✗
Name	✓
Formal Spec.	✓
I/O Examples	⊘

$$\mathit{Definition}\ \mathit{fi2} := \mathit{fun}\ x => x =?\ 2$$
$$\mathit{listrep}\ l\ h\ * \mathit{listrep}\ (\mathit{filter}\ \mathit{fi2}\ l)\ a$$

RQ3: Prompt Components

Biases ✓

English ✗

Name ✓

Formal Spec. ✓

I/O Examples ⃠

```c
#include "treelistdef.c"
struct sll* copyfilteredListWhereElementis2AndReturnItsHead(struct sll *h)
{
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2)
    {
        struct sll* new_node = (struct sll*)malloc(sizeof(struct sll));
        new_node->key = 2;
        new_node->next = copyfilteredListWhereElementis2AndReturnItsHead(h->next);
        return new_node;
    }
    else
    {
        return copyfilteredListWhereElementis2AndReturnItsHead(h->next);
    }
}
```

$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ *\ listrep\ (filter\ fi2\ l)\ a$$

RQ3: Prompt Components

Biases	✓
English	✗
Name	✓
Formal Spec.	✓
I/O Examples	⊘

```c
#include "treelistdef.c"
struct sll* copyfilteredListWhereElementis2AndReturnItsHead(struct sll *h)
{
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2)
    {
        struct sll* new_node = (struct sll*)malloc(sizeof(struct sll));
        new_node->key = 2;
        new_node->next = copyfilteredListWhereElementis2AndReturnItsHead(h->next);
        return new_node;
    }
    else
    {
        return copyfilteredListWhereElementis2AndReturnItsHead(h->next);
    }
}
```

$$Definition\ fi2 := fun\ x \Rightarrow x =?\ 2$$
$$listrep\ l\ h\ *\ listrep\ (filter\ fi2\ l)\ a$$

✓

RQ3: Prompt Components

Biases ✓

English ✗

Name ✓

Formal Spec. ✓

I/O Examples ⃠

Basic: 17/19

SL: 20/23

```c
#include "treelistdef.c"
struct sll* copyfilteredListWhereElementis2AndReturnItsHead(struct sll *h)
{
    if (h == NULL) {
        return NULL;
    }
    if (h->key == 2)
    {
        struct sll* new_node = (struct sll*)malloc(sizeof(struct sll));
        new_node->key = 2;
        new_node->next = copyfilteredListWhereElementis2AndReturnItsHead(h->next);
        return new_node;
    }
    else
    {
        return copyfilteredListWhereElementis2AndReturnItsHead(h->next);
    }
}
```

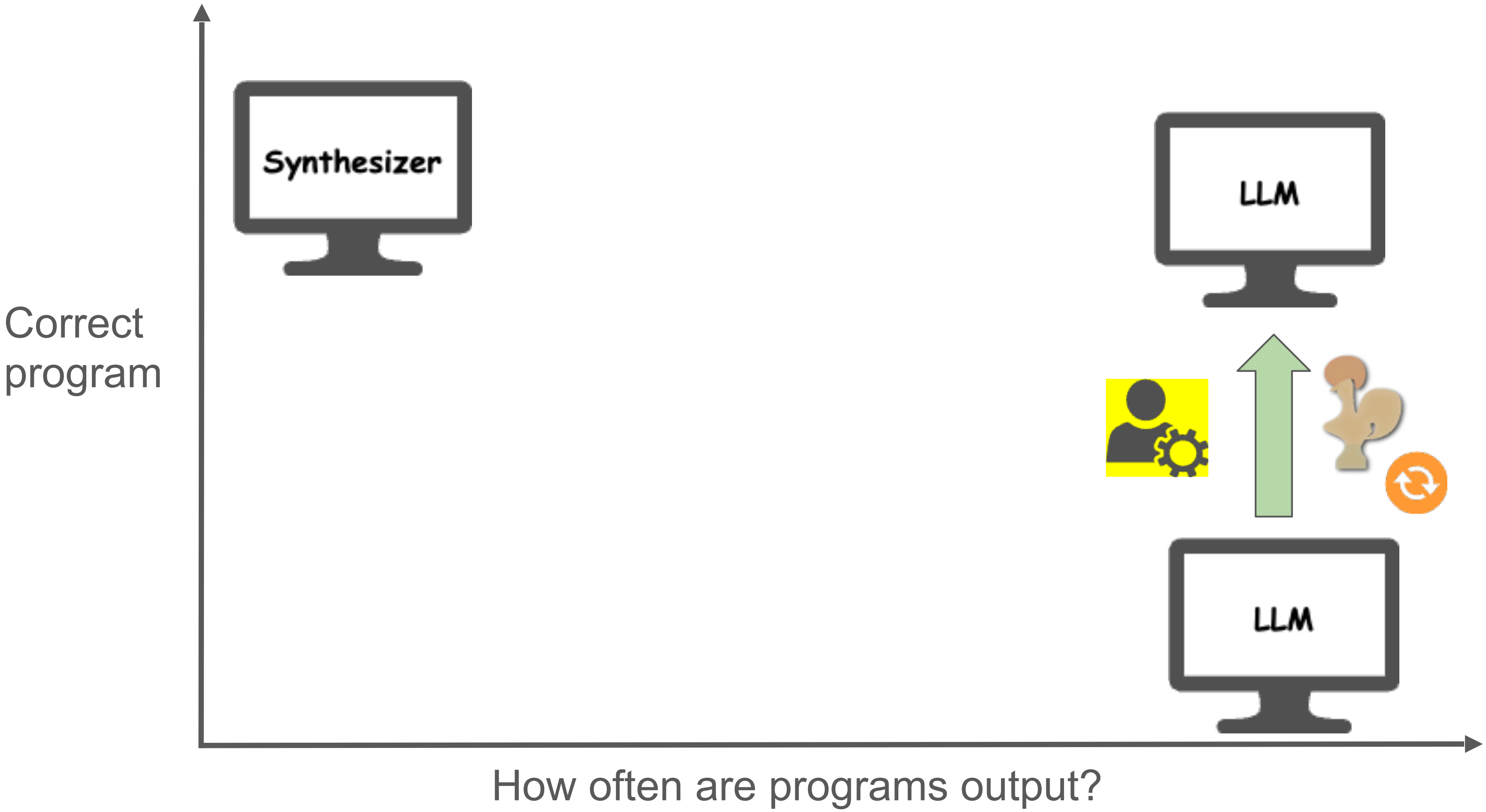
$$Definition\ fi2 := fun\ x => x =?\ 2$$
$$listrep\ l\ h\ *\ listrep\ (filter\ fi2\ l)\ a$$

✓

Conclusions

- LLMs with foundational verifiers as general purpose synthesizers
- Large search space well suited for a popular, mainstream language like C
- LLMs respond positively to biases and natural language
- Automation is well suited to the biases
- SepAuto is not complete
- Data leakage may have required less re prompting

Current State



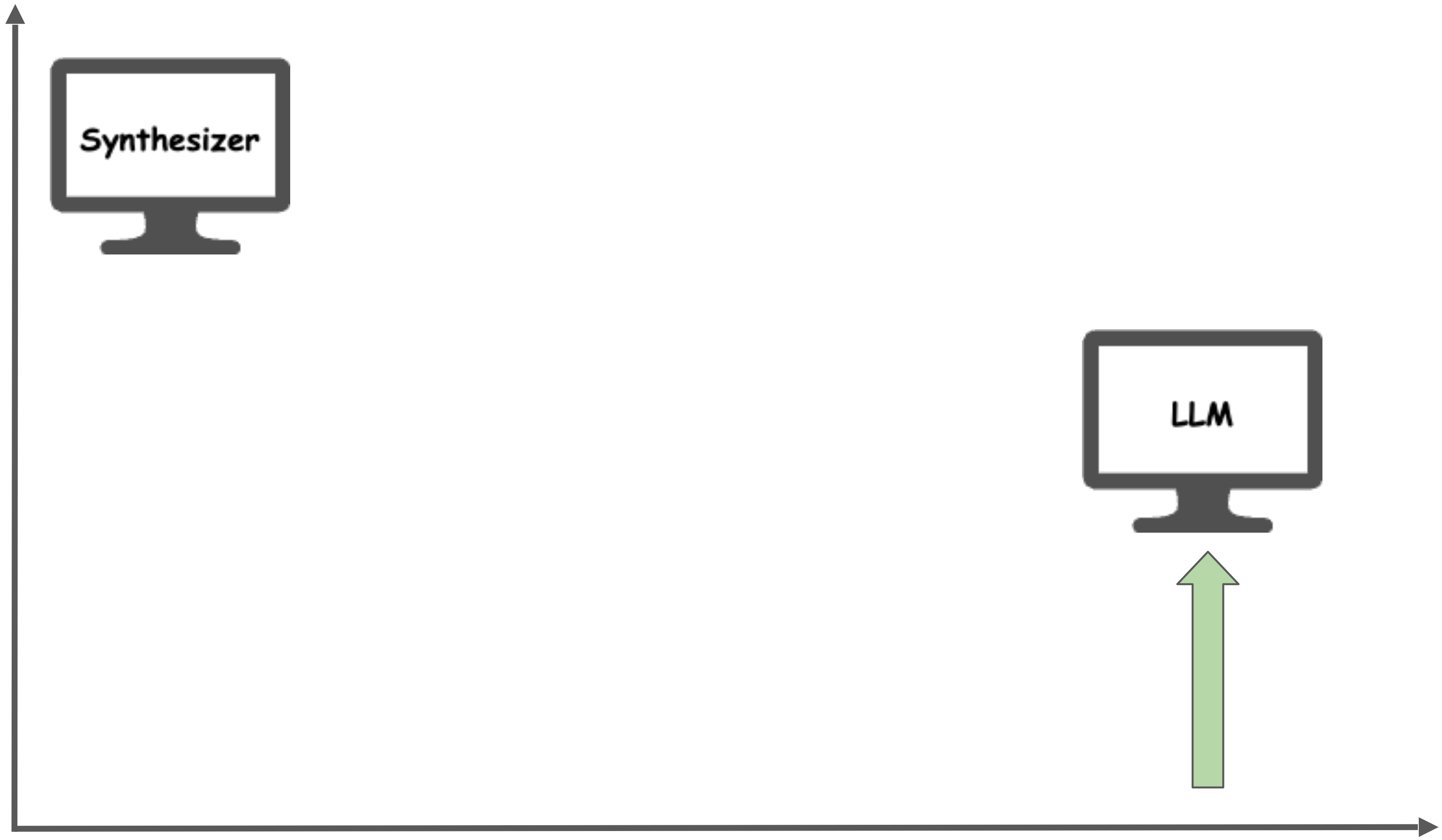
What’s next?

Correct program

SYNVER

How often are programs output?

What’s next?

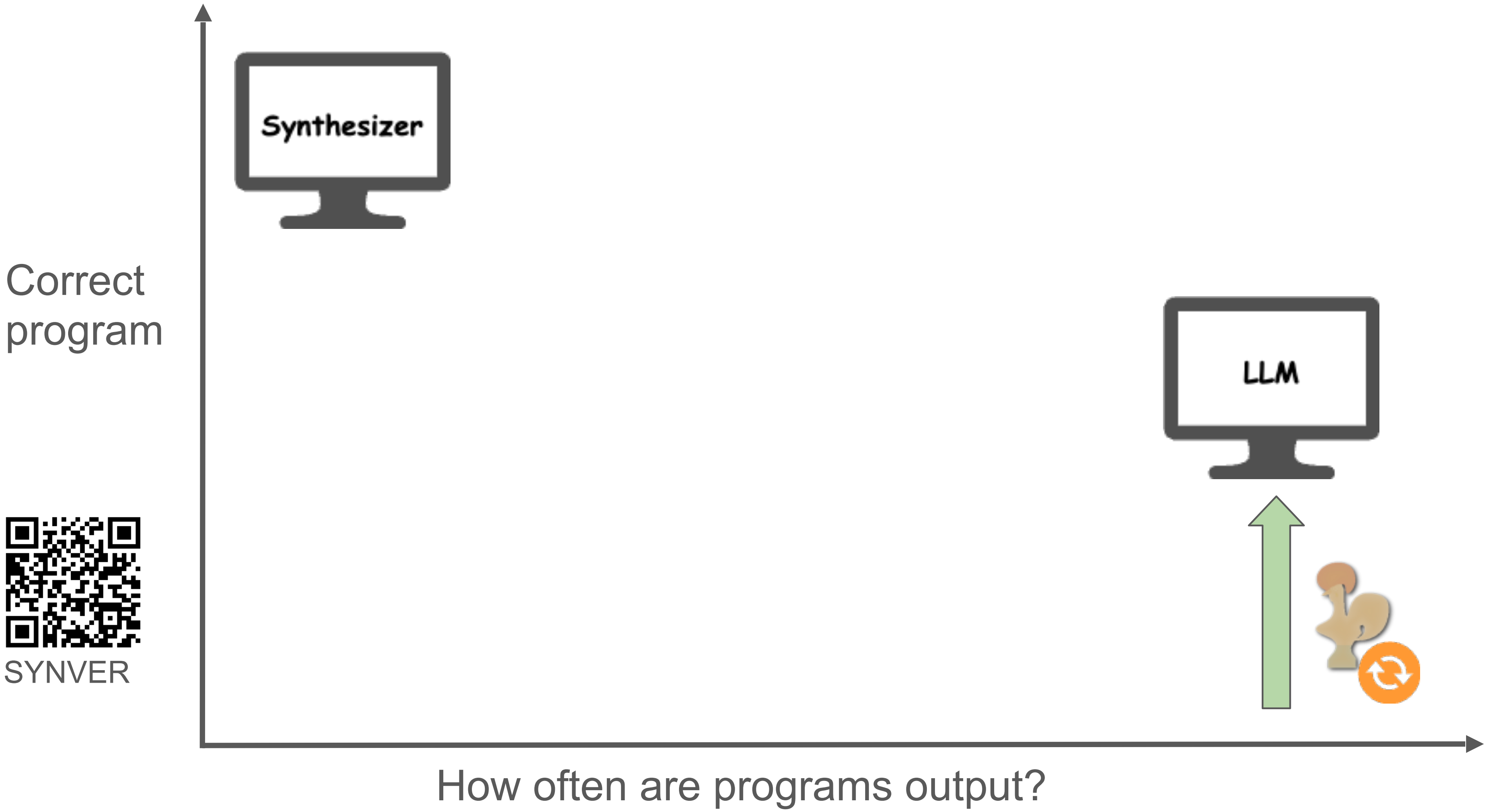


Correct program

How often are programs output?

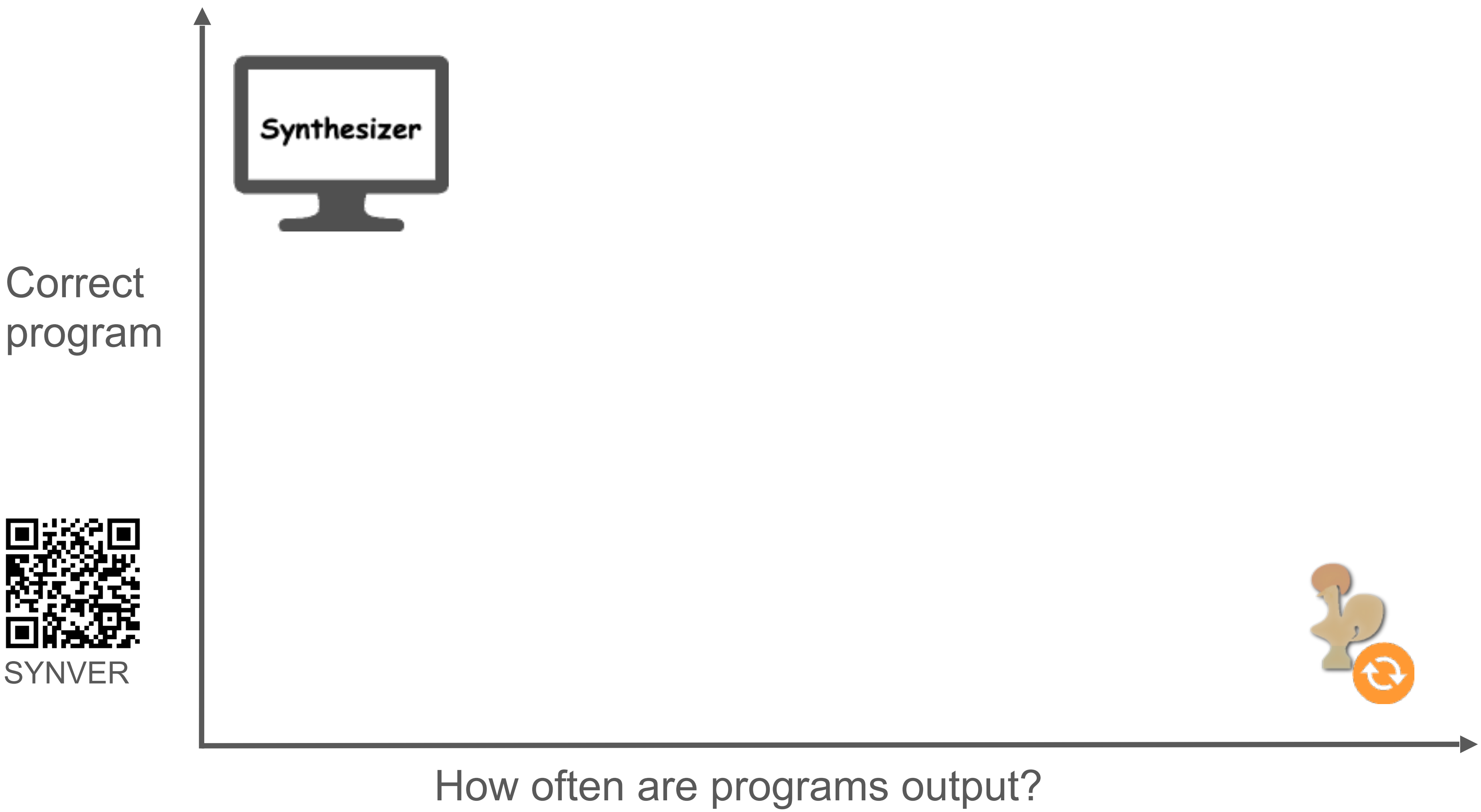
SYNVER

What's next?

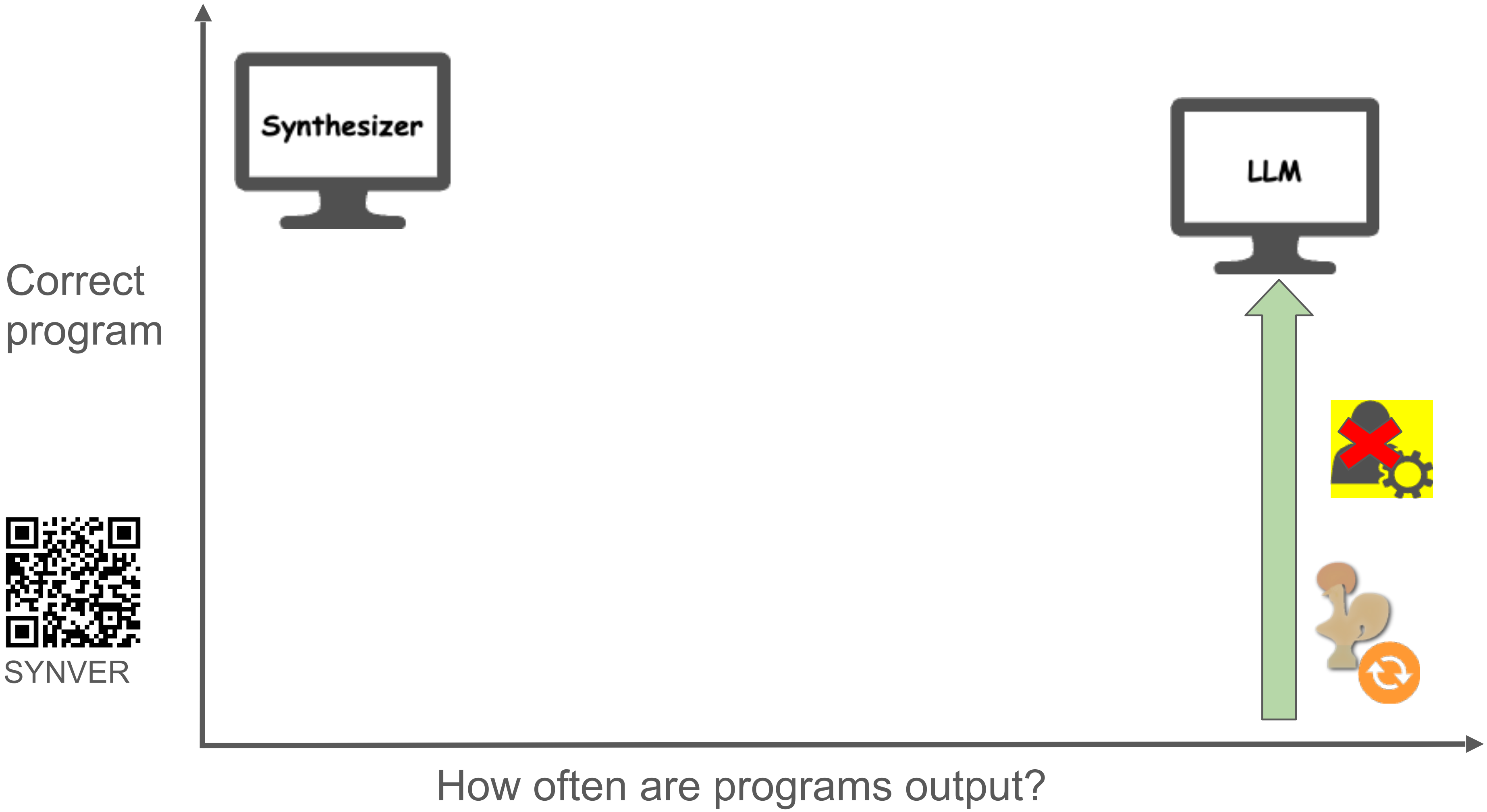


Correct program
SYNVER
Synthesizer
LLM
How often are programs output?

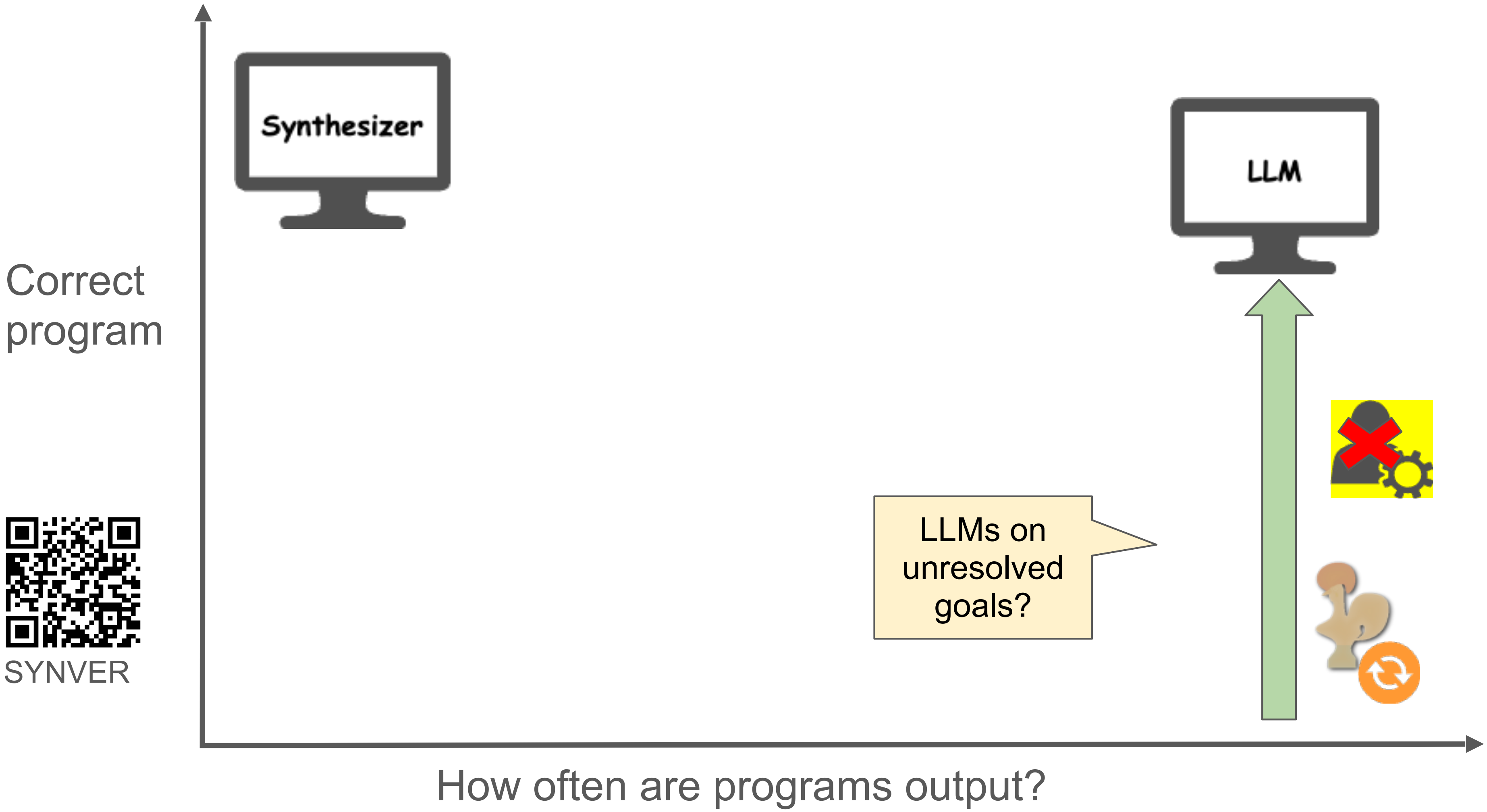
What’s next?


Synthesizer
Correct program
SYNVER
How often are programs output?

What’s next?


Synthesizer
LLM
Correct
program
SYNVER
How often are programs output?

What's next?


Synthesizer
LLM
Correct program
SYNVER
LLMs on unresolved goals?
How often are programs output?