

The Art of ~~Writing~~ Righting Code

Aalok Thakkar, Ashoka University

IIIT Delhi, CSE Seminar Series
February 14, 2025

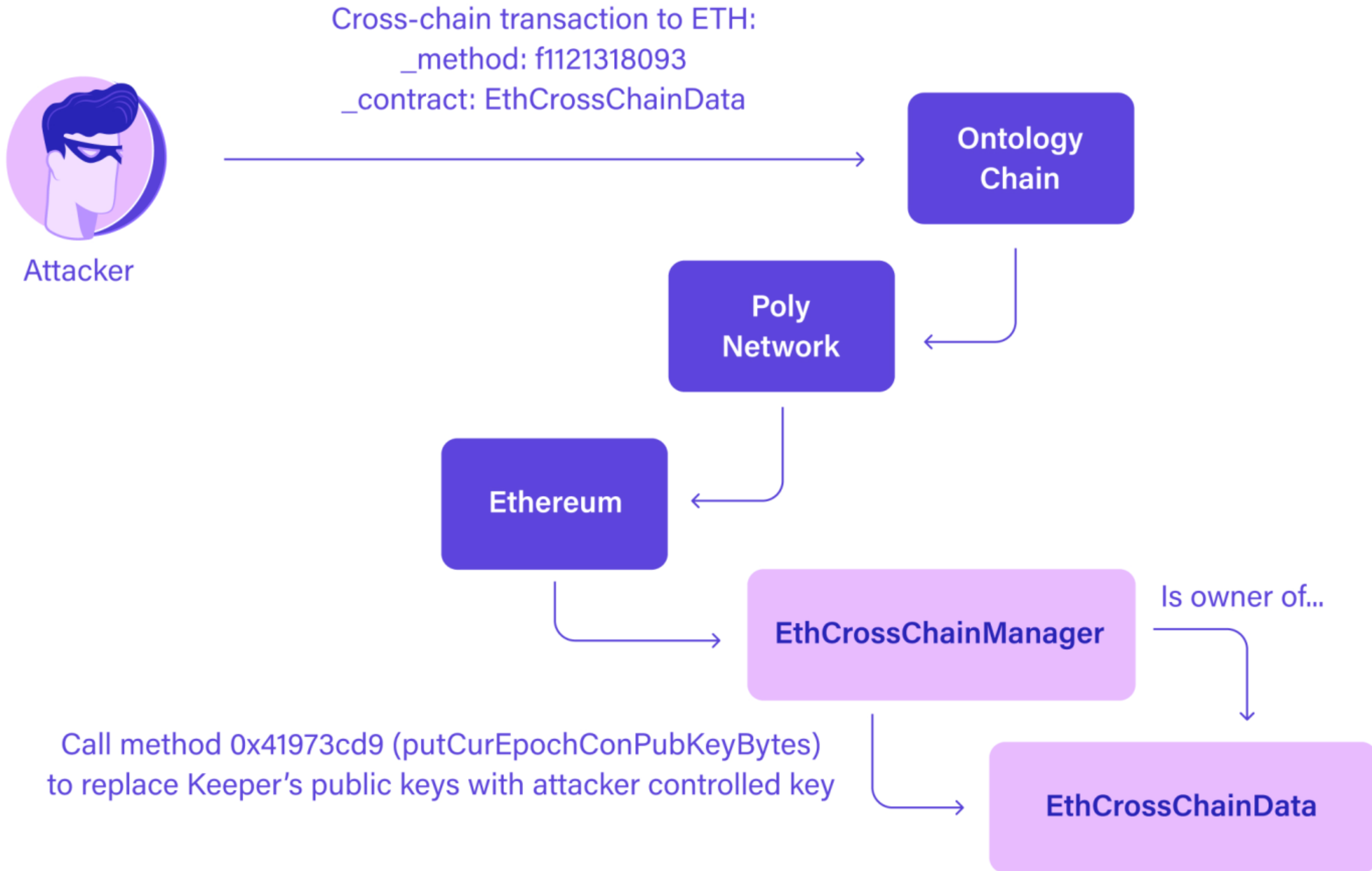
Poly Network Hack

August 10, 2021



PolyNetwork

611 million USD or 4600 crores INR



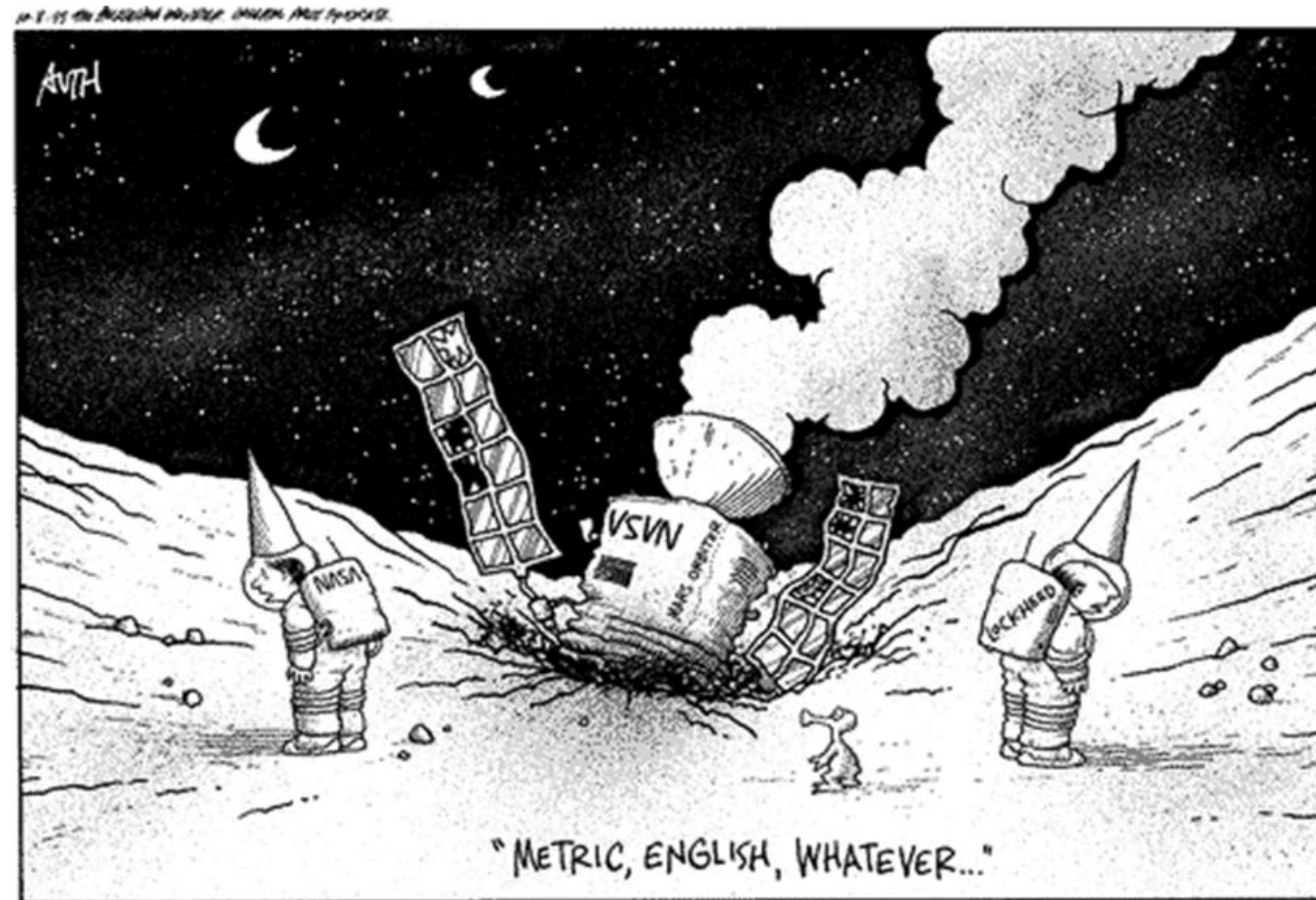
Therac-25 Radiation Therapy Machine

Administered radiation doses hundreds of times higher than intended.

Race Conditions!



Mars Climate Orbiter (1999)



Remember the Mars Climate Orbiter incident from 1999?



Boeing 737 MAX

Direct Costs: US\$20 billion

Indirect Costs: US\$60 billion

Deaths: 346



Toyota Unintended Acceleration

5.2 million vehicles
US\$2.2 billion in lawsuits
37 deaths

How do you *detect* bugs?

How do *avoid* vulnerabilities?

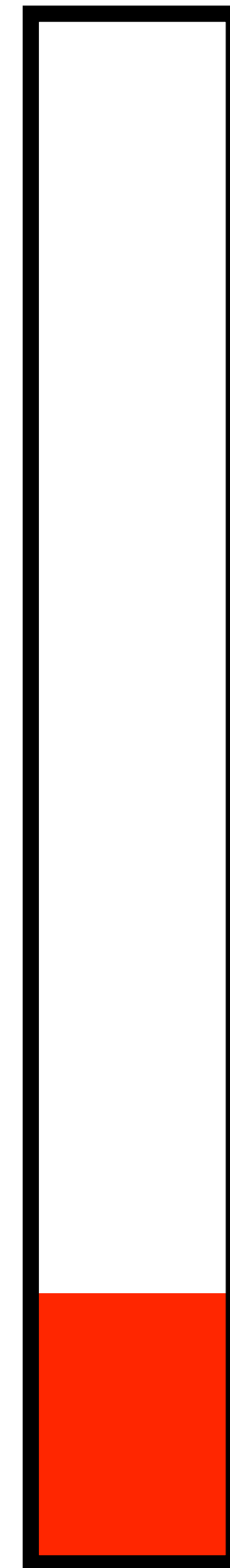
How do *ensure* security?

Read Your Code Carefully!

Read Your Code Carefully!



Read Your Code Carefully!



Insecure!

Testing?

Testing?

[26, 1, 2, 300, - 3]



Array Sorting



[-3, 1, 2, 26, 300]

Testing?

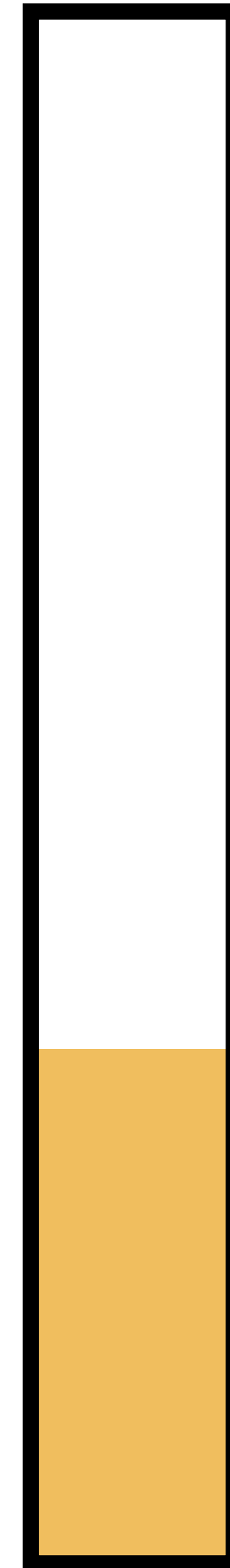
[26, 1, 2, 300, - 3]



Array Sorting



[-3, 1, 2, 26, 300]



*Slightly
Better*

Fuzz Testing

[1,2] [26, 1, 2, 300, - 3] [0, 0, - 7, - 8, 7]



Array Sorting



[1,2] [-3, 1, 2, 26, 300] [-7, - 8, 0, 0, 7]

Fuzz Testing

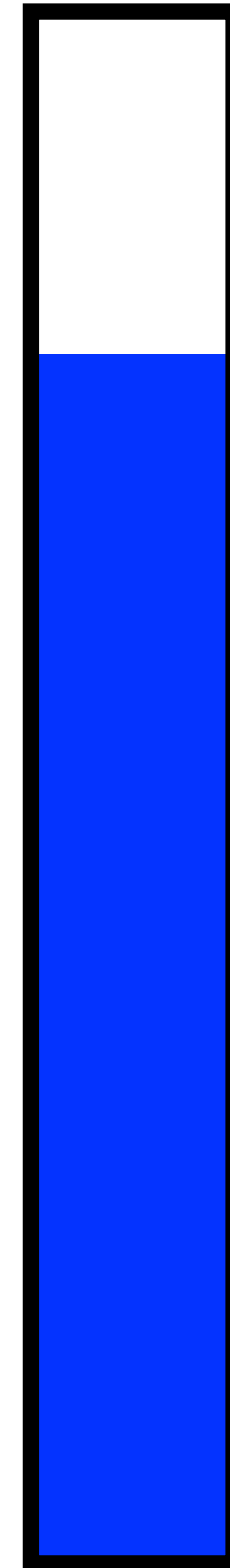
[1,2] [26, 1, 2, 300, - 3] [0, 0, - 7, - 8, 7]



Array Sorting



[1,2] [-3, 1, 2, 26, 300] [-7, - 8, 0, 0, 7]



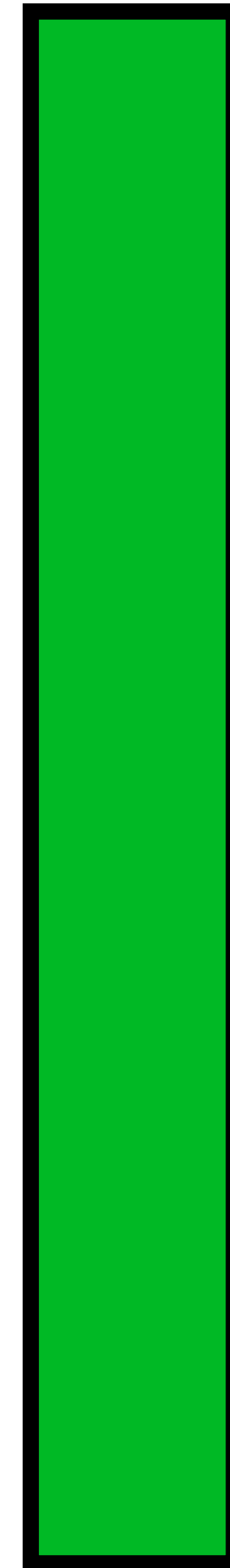
Much
Better!



“Program testing can be used to
show the presence of bugs, but
never to show their absence!”

— Edsger W. Dijkstra

How to prove the
absence of bugs?



Correctness
Guarantees!

How can one check a routine in the sense of making sure that it is right? In order that the man who checks may not have too difficult a task, the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows.

- Alan Turing, 1949

[1,2] [26, 1, 2, 300, − 3] [0, 0, − 7, − 8, 7]



Array Sorting



[1,2] [−3, 1, 2, 26, 300] [−7, − 8, 0, 0, 7]

8

5

3

2

Array Sorting

2

3

5

8

Array Sorting

```
selectionSort(int A[], n) {  
    i = 0;  
    while (i < n - 1) {  
        v = i;  
        j = i + 1;  
        while (j < n) {  
            if (A[j] < A[v])  
                v = j;  
            j++;  
        }  
        swap(A[i], A[v]);  
        i++;  
    }  
    return A;  
}
```


Pre-condition: I have an array!

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

```
    return A;
```

```
}
```

Post-condition: I want a sorted array!

Pre-condition: I have an array!

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

```
    return A;
```

```
}
```

Post-condition: I want a sorted array!

Specification.

Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

```
    return A;
```

```
}
```

Post-condition: $\forall i \in [n]. \forall j \in [n]. \left((i < j) \rightarrow (A[i] < A[j]) \right)$

Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

```
    return A;
```

```
}
```

Post-condition: $\forall i \in [n]. \forall j \in [n]. \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$

Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

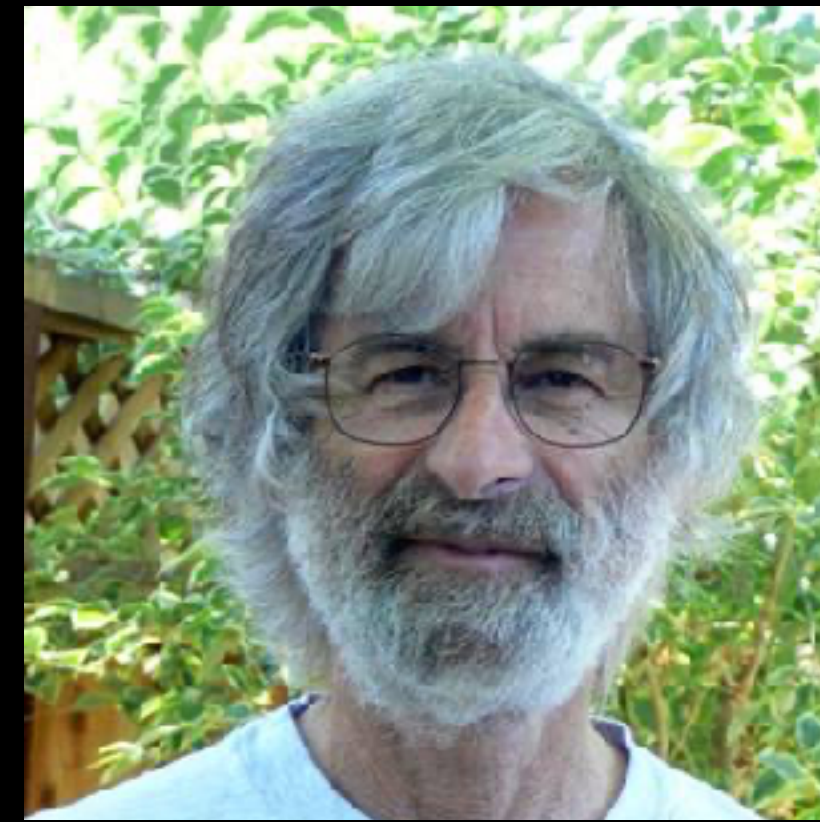
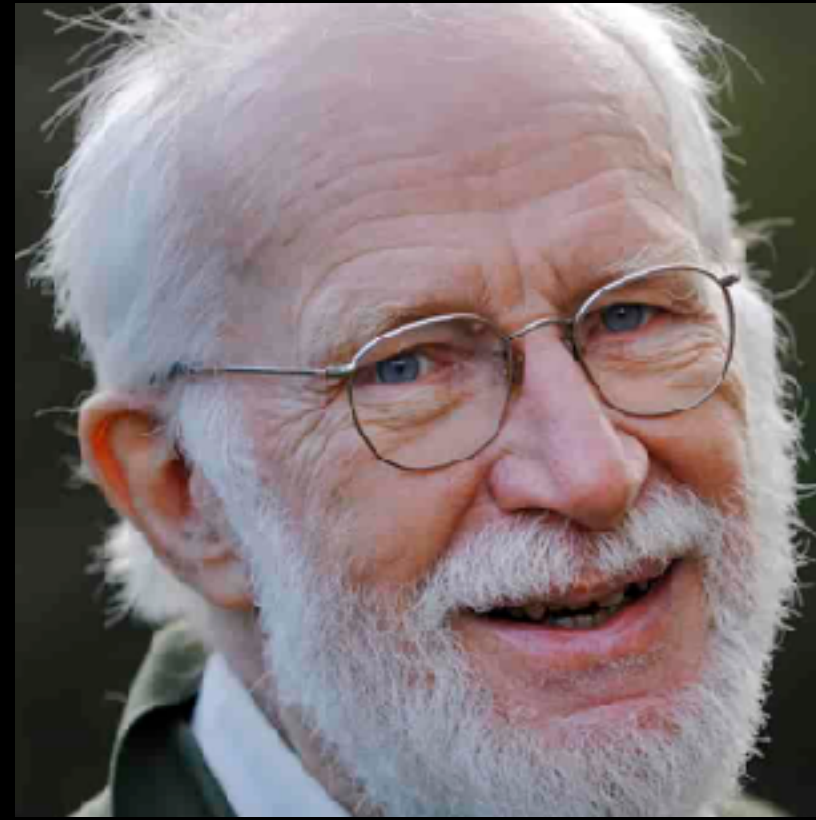
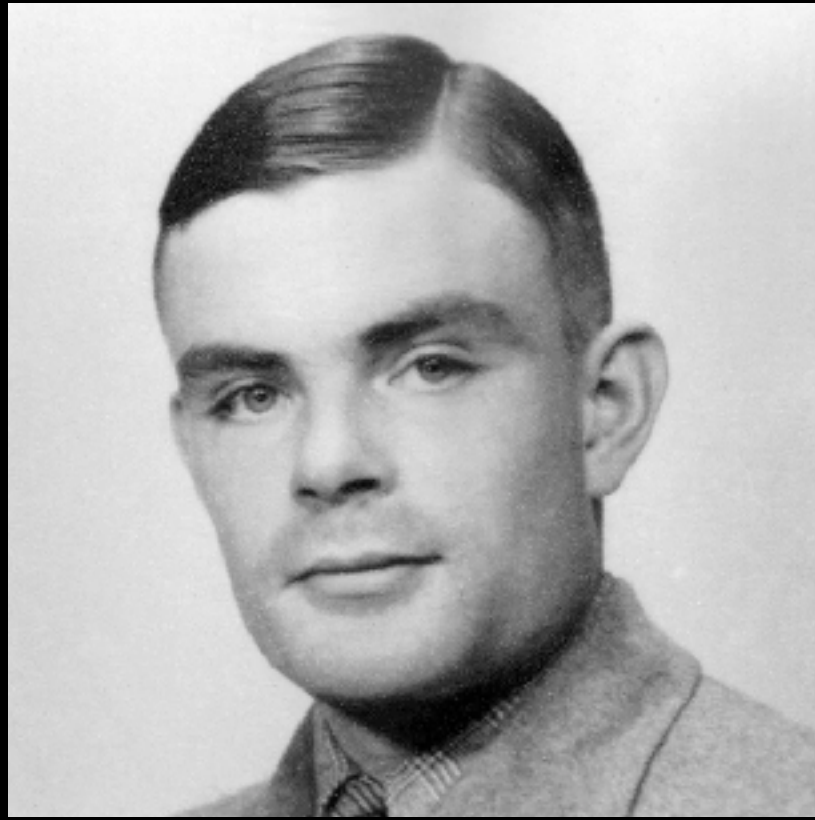
```
    }
```

```
    return A;
```

```
}
```

Post-condition: $\forall i \in [n]. \forall j \in [n]. \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$

Theorem: For any input array A of size n , if the pre-condition holds before running the code, the post-condition holds after running `selectionSort(int A[], n)`.



Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
                j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

```
    return A;
```

```
}
```

Post-condition: $\forall i \in [n]. \forall j \in [n]. \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$

Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

```
    }
```

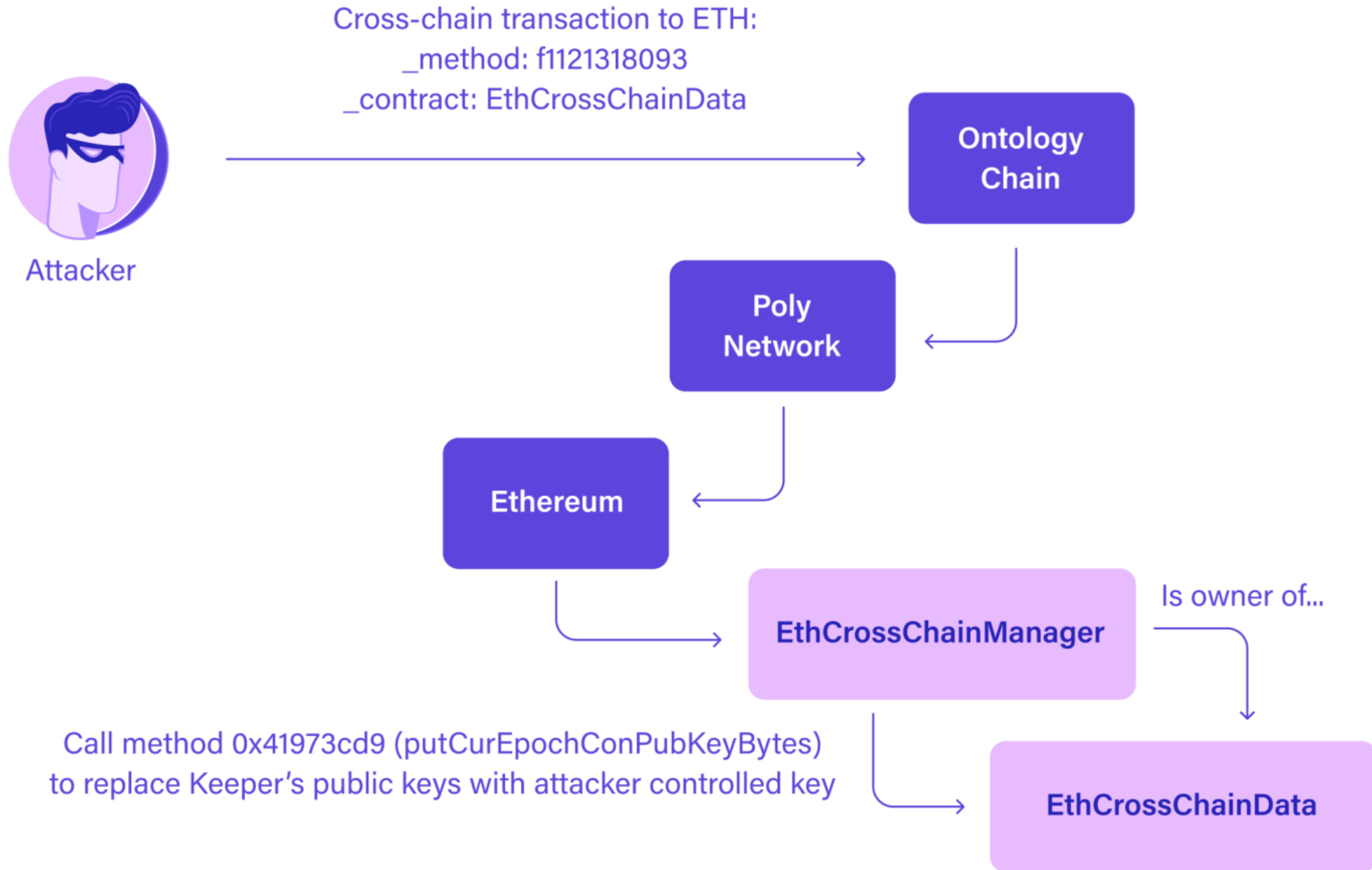
```
    return A;
```

```
}
```

Invariant: $\forall k_1 \forall k_2 . \left((0 \leq k_1 < k_2 < n) \wedge (k_1 < i) \right) \rightarrow (A[k_1] \leq A[k_2])$

Invariant: $\forall k . (i \leq k \leq j) \rightarrow (A[i] \leq A[k])$

Post-condition: $\forall i \in [n] . \forall j \in [n] . \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$



Social Processes and Proofs of Theorems and Programs

Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis

It is argued that formal verifications of programs, no matter how obtained, **will not play the same key role** in the development of computer science and software engineering **as proofs do in mathematics**. Furthermore the absence of continuity, the inevitability of change, and the complexity of specification of significantly many real programs make the formal verification process difficult to justify and manage. It is felt that ease of formal verification should not dominate program language design.

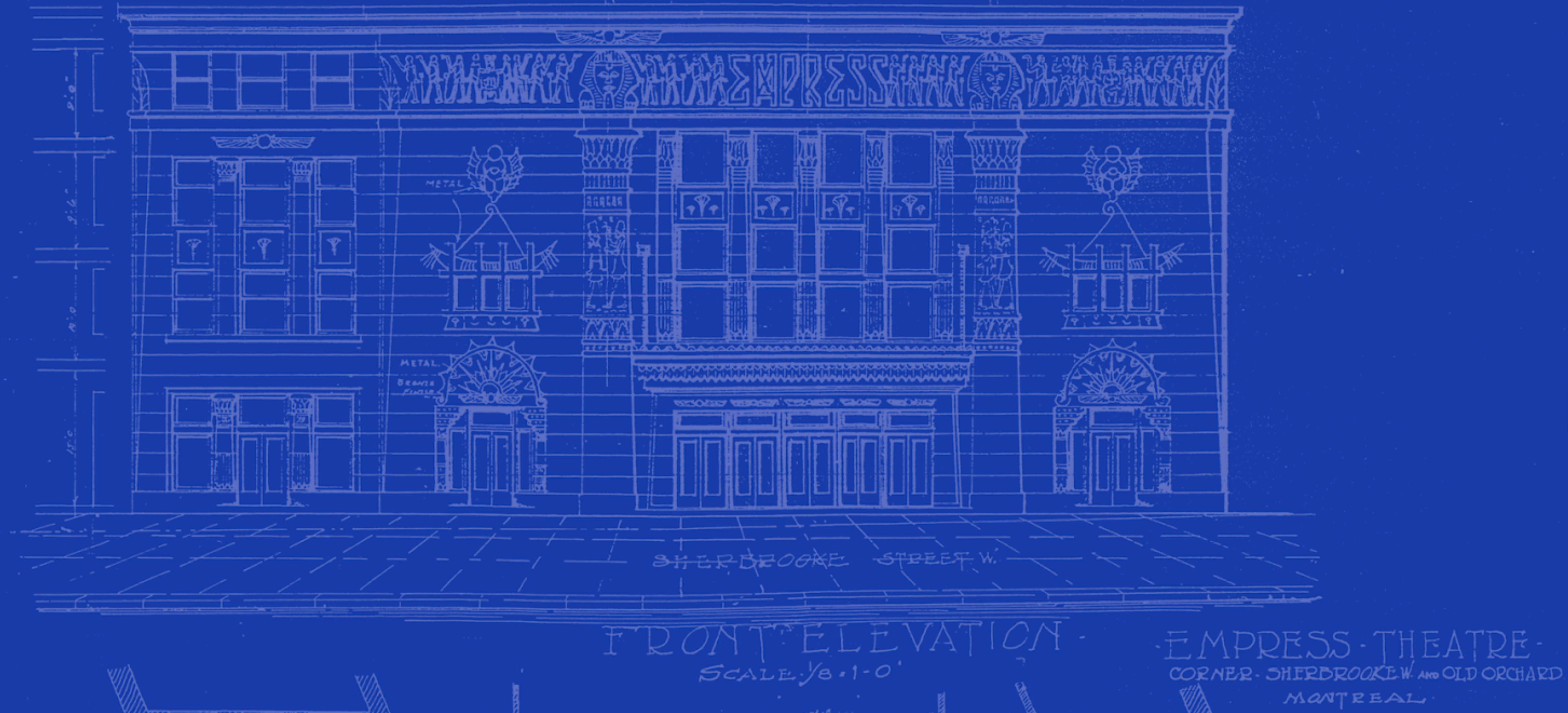
Program Verification is Declared Dead!

Social Processes and Proofs of Theorems and Programs

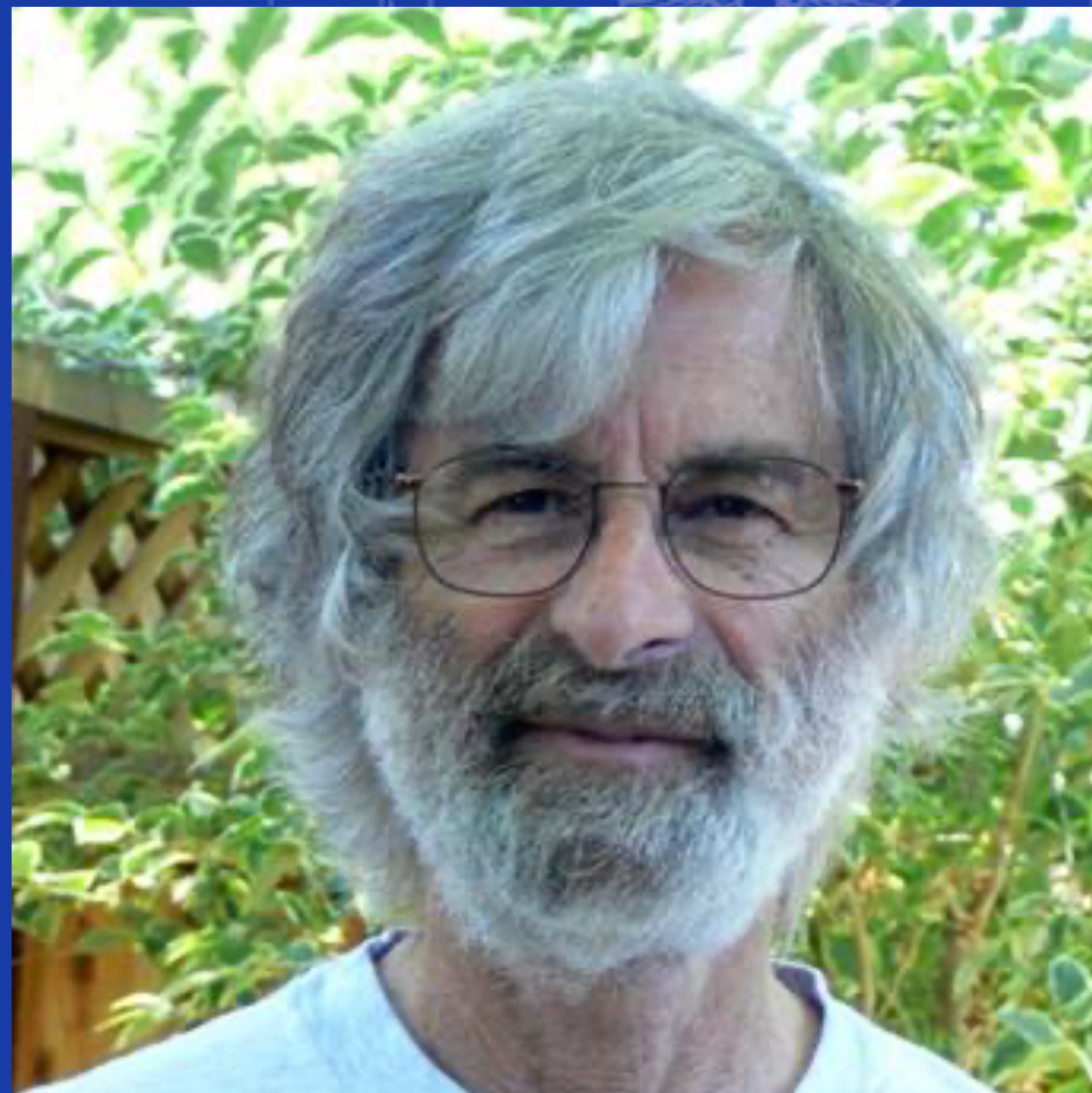
Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis

It is argued that formal verifications of programs, no matter how obtained, **will not play the same key role** in the development of computer science and software engineering **as proofs do in mathematics**. Furthermore the absence of continuity, the inevitability of change, and the complexity of specification of significantly many real programs make the formal verification process difficult to justify and manage. It is felt that ease of formal verification should not dominate program language design.

Who Builds a House Without Drawing Blueprints?



Who Builds a House Without Drawing Blueprints?

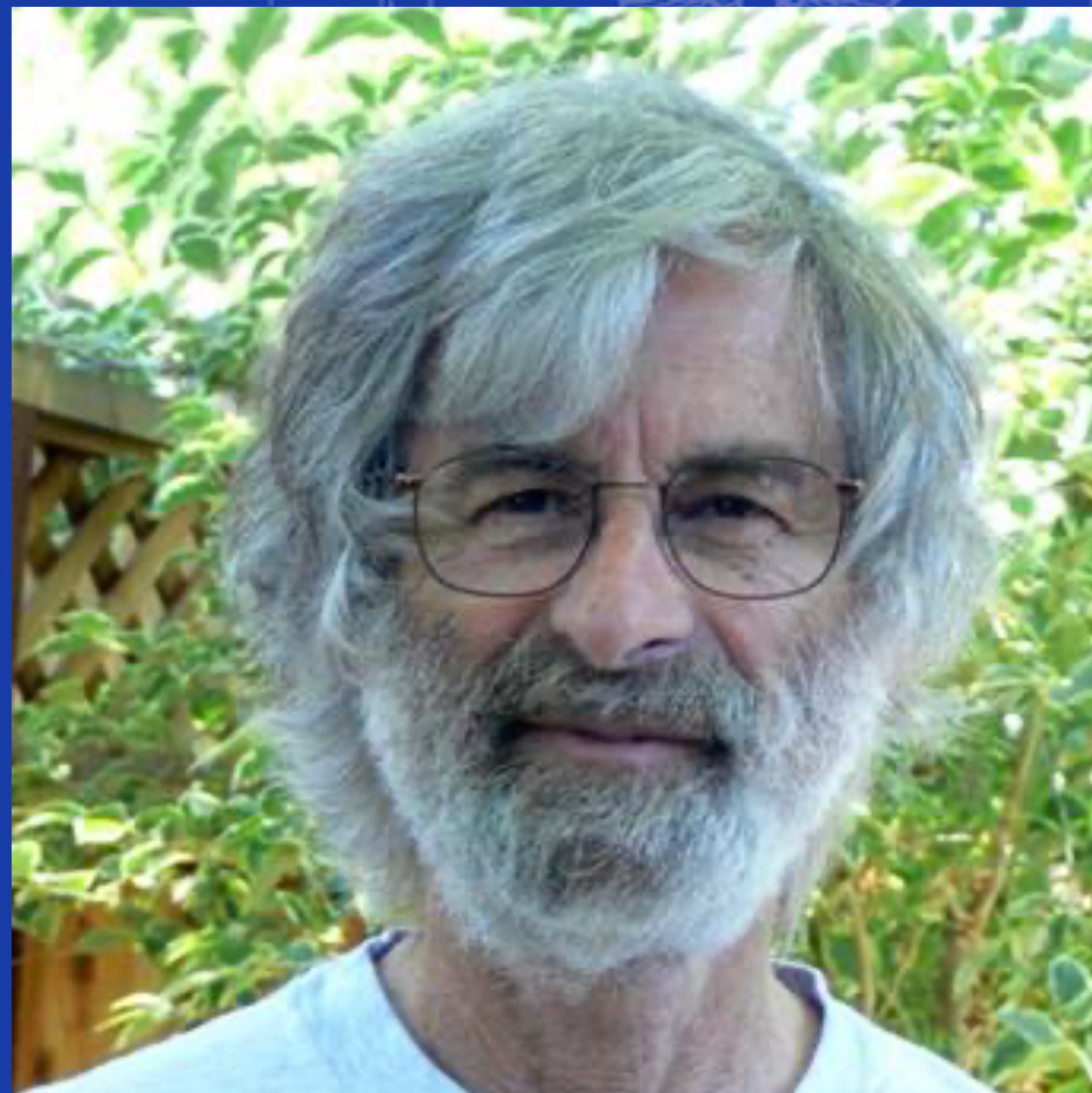


Architects draw detailed plans before a brick is laid or a nail is hammered. But few programmers write even a rough sketch of what their programs will do before they start coding. We can learn from architects.

FRONT ELEVATION
SCALE 1/8" = 1'-0"

EMPRESS THEATRE
CORNER SHERBROOKE W. AND OLD ORCHARD
MONTREAL

Who Builds a House Without Drawing Blueprints?



The main reason for writing a formal spec is to apply tools to check it. Tools cannot find design errors in informal specifications.

FRONT ELEVATION
SCALE 1/8" = 1'-0"

EMPRESS THEATRE
CORNER SHERBROOKE W. AND OLD ORCHARD
MONTREAL

Pre-condition: $A : t^n, t : (\leq)$

```
selectionSort(int A[], n) {
```

```
    i = 0;
```

```
    while (i < n - 1) {
```

```
        v = i;
```

```
        j = i + 1;
```

```
        while (j < n) {
```

```
            if (A[j] < A[v])
```

```
                v = j;
```

```
            j++
```

```
        }
```

```
        swap(A[i], A[v]);
```

```
        i++;
```

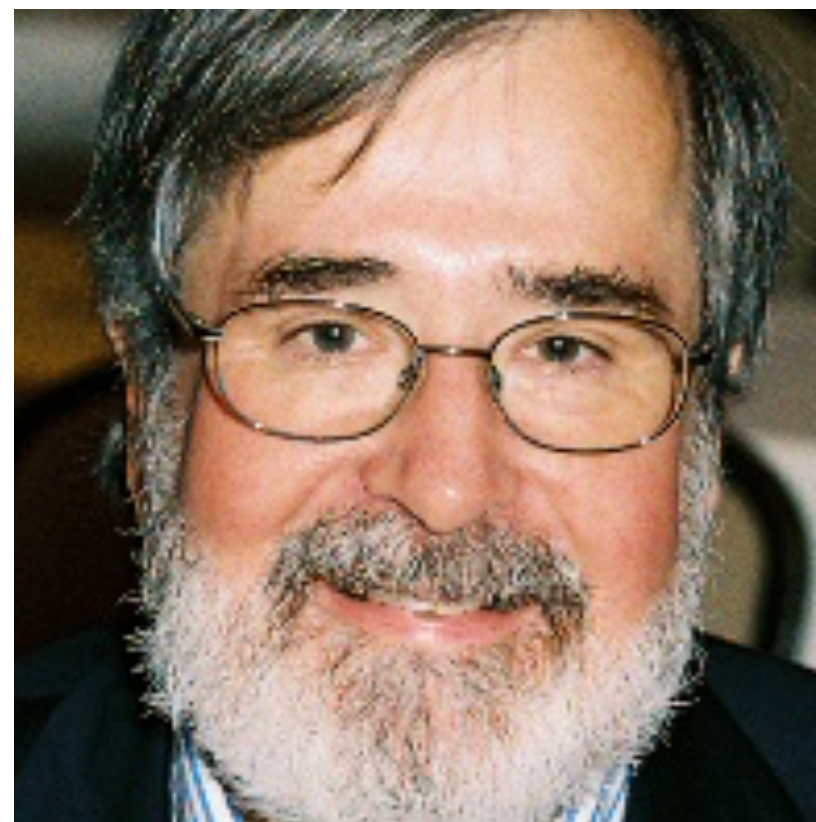
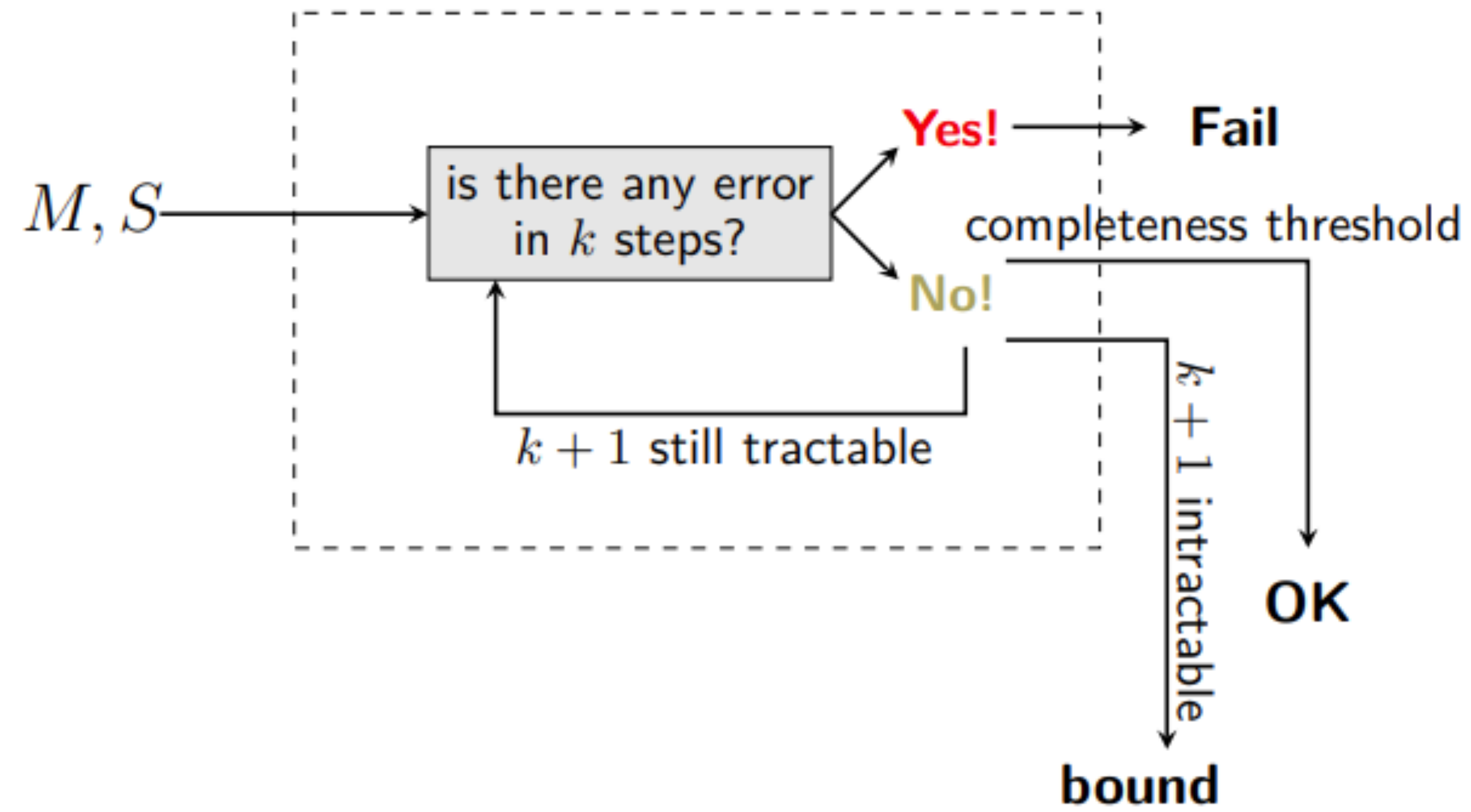
```
    }
```

```
    return A;
```

```
}
```

Post-condition: $\forall i \in [n]. \forall j \in [n]. \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$

Bounded Model Checking:



SLAM

```
if (node->id != node->id) { i ++ vis[procs, end()*node]{
```


A problem has been detected and windows has been shut down to prevent damage to your computer.

A driver has overrun a stack-based buffer. This overrun could potentially allow a malicious user to gain control of this machine.

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

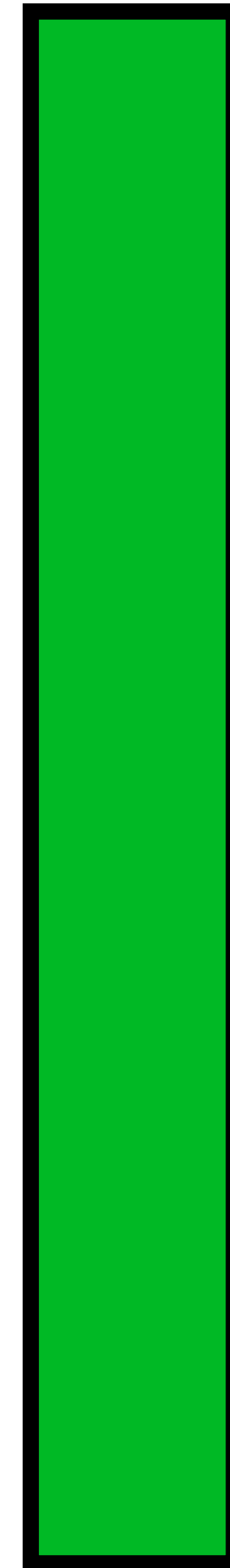
Technical information:

*** STOP: 0x000000F7 (0x00000280029089B0, 0x000029C1DC791FF4, 0xFFFFD63E2386E00B, 0x0000000000000000)

collecting data for crash dump ...

initializing disk for crash dump ...

How to prove the
absence of bugs?



Correctness
Guarantees!


```

selectionSort(int A[], n) {
    i = 0;
    while (i < n - 1) {
        v = i;
        j = i + 1;
        while (j < n) {
            if (A[j] < A[v])
                v = j;
            j++;
        }
        swap(A[i], A[v]);
        i++;
    }
    return A;
}

```

Invariant: $\forall k_1 \forall k_2 . \left((0 \leq k_1 < k_2 < n) \wedge (k_1 < i) \right) \rightarrow (A[k_1] \leq A[k_2])$

Invariant: $\forall k . (i \leq k \leq j) \rightarrow (A[v] \leq A[k])$

Post-condition: $\forall i \in [n] . \forall j \in [n] . \left((i < j) \rightarrow (A[i] \leq A[j]) \right)$

Part II: Automated Theorem Proving

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

What are \vee , \leftarrow , \rightarrow , \neg , \oplus , \leftrightarrow ?

What are variables (atoms)?

What are assignments (models)?

What does satisfaction mean?

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

Is there a satisfying assignment?

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

Is there a satisfying assignment?

Check all assignments!

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

Is there a satisfying assignment?

Check all assignments!

Can we do better?

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

Is there a satisfying assignment?

Check all assignments!

Can we do something smarter?

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$p = c_1 \wedge c_2 \wedge \dots \wedge c_m$$

$$c_i = l_{i,1} \vee l_{i,2} \vee \dots \vee l_{i,n_i}$$

$$l_{i,j} = A \text{ or } l_{i,j} = \neg A$$

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$p = c_1 \wedge c_2 \wedge \dots \wedge c_m$$

$$c_i = l_{i,1} \vee l_{i,2} \vee \dots \vee l_{i,n_i}$$

$$l_{i,j} = A \text{ or } l_{i,j} = \neg A$$

NORMALISATION THEOREM: For every propositional formula, there exists an *equivalent* formula in conjunctive normal form.

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$p = c_1 \wedge c_2 \wedge \dots \wedge c_m$$

$$c_i = l_{i,1} \vee l_{i,2} \vee \dots \vee l_{i,n_i}$$

$$l_{i,j} = A \text{ or } l_{i,j} = \neg A$$

TSEITIN'S THEOREM: For every propositional formula, there exists a polynomial size *equisatisfiable* formula in conjunctive normal form.

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$\{(A \vee B), (A \vee C), (A \vee \neg D), (\neg A \vee D), (\neg D \vee A), (B \vee \neg D), (D \vee \neg B)\}$$

Can find a satisfying assignment in polynomial time?

Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$\{(A \vee B), (A \vee C) \wedge (A \vee \neg D), (\neg A \vee D) \wedge (\neg D \vee A), (B \vee \neg D) \wedge (D \vee \neg B)\}$$

$$\{(A \vee B), (A \vee C), (A \vee \neg D), (\neg A \vee D), (\neg D \vee A), (B \vee \neg D), (D \vee \neg B)\}$$

Can find a satisfying assignment in polynomial time?

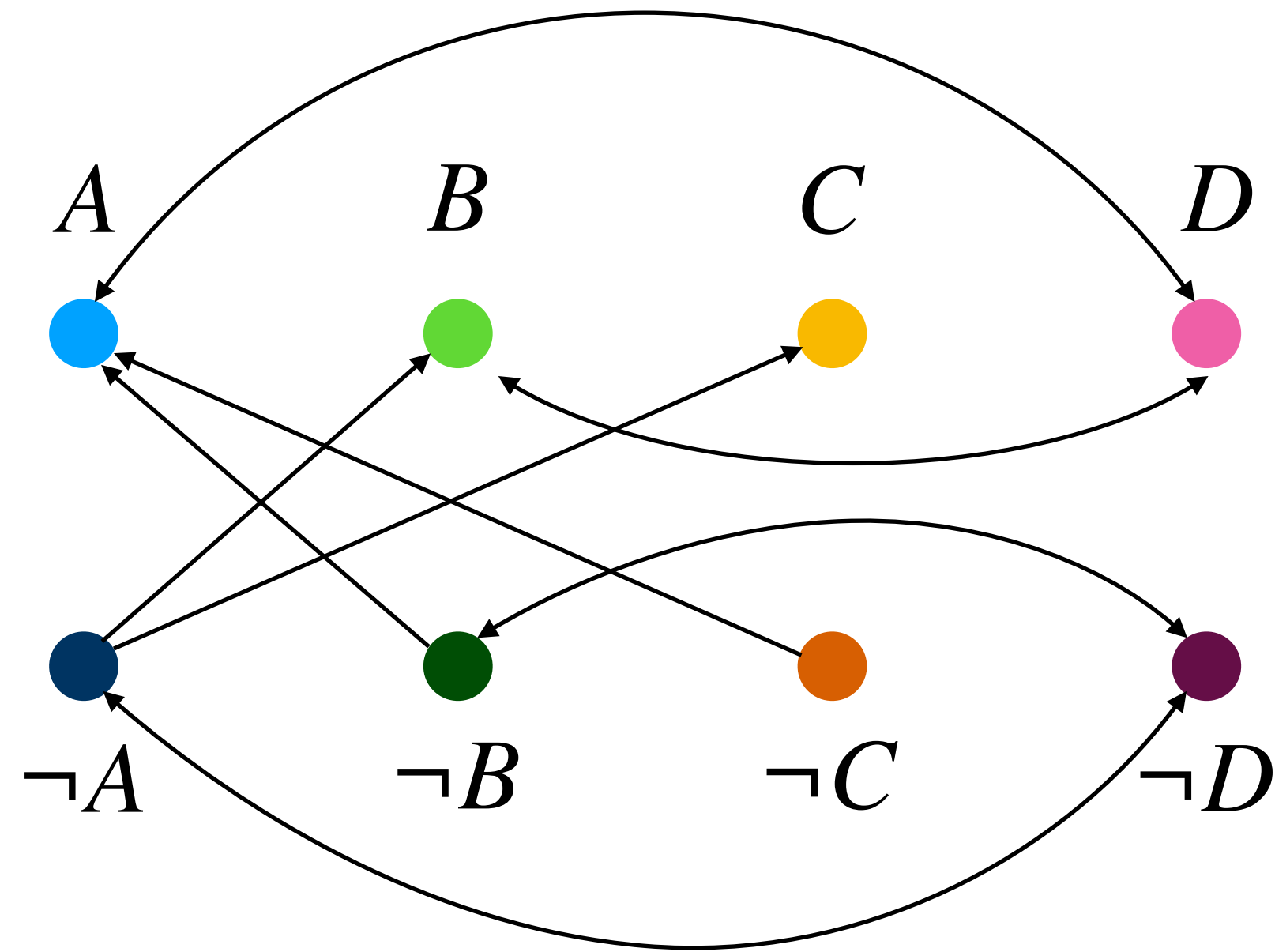
Conjunctive Normal Form (CNF)

$$\{A \vee B, A \leftarrow (C \rightarrow D), \neg(D \oplus A), B \leftrightarrow D\}$$

$$\{A \vee B, A \vee \neg(\neg C \vee D), \neg((D \vee A) \wedge \neg(D \wedge A)), (B \wedge D) \vee (\neg B \wedge \neg D)\}$$

$$\{(A \vee B), (A \vee C) \wedge (A \vee \neg D), (\neg A \vee D) \wedge (\neg D \vee A), (B \vee \neg D) \wedge (D \vee \neg B)\}$$

$$\{(A \vee B), (A \vee C), (A \vee \neg D), (\neg A \vee D), (\neg D \vee A), (B \vee \neg D), (D \vee \neg B)\}$$



ASPVALL, PLASS, TARJAN (1979): For any variable X , the vertices for X and $\neg X$ exist in a strongly connected component of the implication graph if and only if the set is not satisfiable.

DAVIS–PUTNAM–LOGEMANN–LOVELAND (DPLL) ALGORITHM

Input: CNF f , and partial assignment m

If f is true under m , return m .

If f is false under m , return \perp .

If \exists unit literal p under m , then return $DPLL(f, m[p \rightarrow 1])$.

If \exists unit literal $\neg p$ under m , then return $DPLL(f, m[p \rightarrow 0])$.

Chose an unassigned variable a , and assign it $b \in \{0,1\}$.

If $DPLL(f, m[a \rightarrow b]) = SAT$, return $m[a \rightarrow b]$

Else, return $DPLL(f, m[a \rightarrow 1 - b])$

$$c_1 = (\neg x_1 \vee x_2)$$

$$c_2 = (\neg x_1 \vee x_3 \vee x_5)$$

$$c_3 = (\neg x_2 \vee x_4)$$

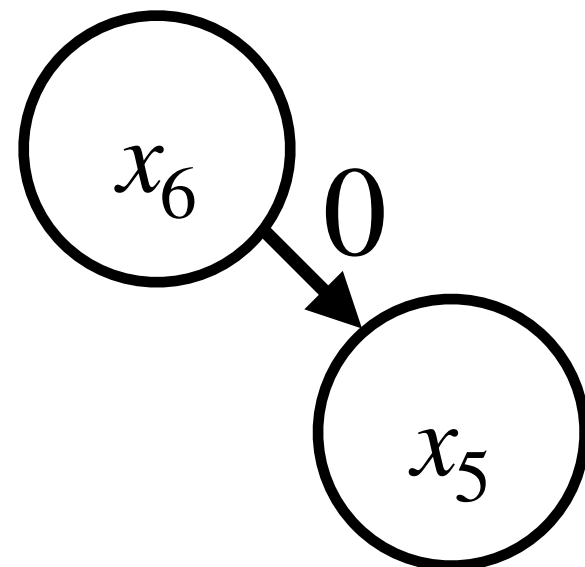
$$c_4 = (\neg x_3 \vee \neg x_4)$$

$$c_5 = (x_1 \vee x_5 \vee \neg x_2)$$

$$c_6 = (x_2 \vee x_3)$$

$$c_7 = (x_2 \vee \neg x_3 \vee x_7)$$

$$c_8 = (x_6 \vee \neg x_5)$$



$$c_1 = (\neg x_1 \vee x_2)$$

$$c_2 = (\neg x_1 \vee x_3 \vee x_5)$$

$$c_3 = (\neg x_2 \vee x_4)$$

$$c_4 = (\neg x_3 \vee \neg x_4)$$

$$c_5 = (x_1 \vee x_5 \vee \neg x_2)$$

$$c_6 = (x_2 \vee x_3)$$

$$c_7 = (x_2 \vee \neg x_3 \vee x_7)$$

$$c_8 = (x_6 \vee \neg x_5)$$

DAVIS–PUTNAM–LOGEMANN–LOVELAND (DPLL) ALGORITHM

Input: CNF f , and partial assignment m

If f is true under m , return m .

If f is false under m , return \perp .

If \exists unit literal p under m , then return $DPLL(f, m[p \rightarrow 1])$.

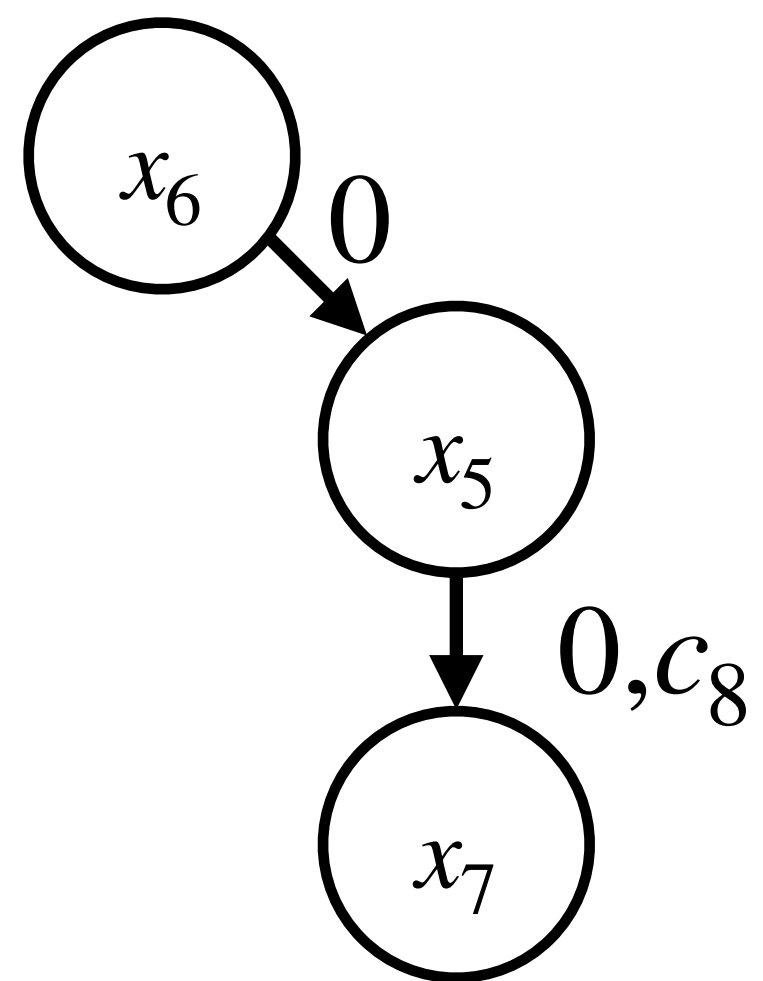
If \exists unit literal $\neg p$ under m , then return $DPLL(f, m[p \rightarrow 0])$.

Chose an unassigned variable a , and assign it $b \in \{0,1\}$.

If $DPLL(f, m[a \rightarrow b]) = SAT$, return $m[a \rightarrow b]$

Else, return $DPLL(f, m[a \rightarrow 1 - b])$

$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



DAVIS–PUTNAM–LOGEMANN–LOVELAND (DPLL) ALGORITHM

Input: CNF f , and partial assignment m

If f is true under m , return m .

If f is false under m , return \perp .

If \exists unit literal p under m , then return $DPLL(f, m[p \rightarrow 1])$.

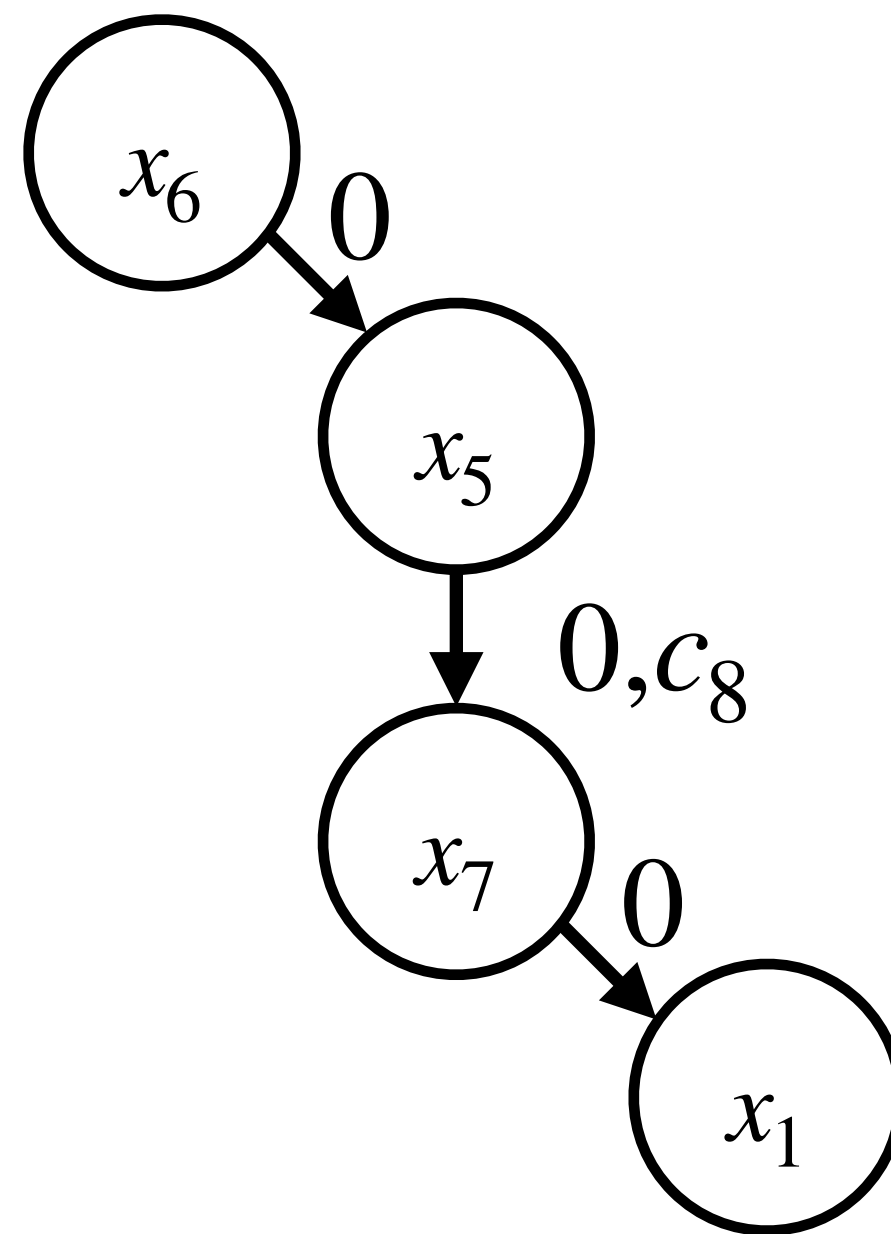
If \exists unit literal $\neg p$ under m , then return $DPLL(f, m[p \rightarrow 0])$.

Chose an unassigned variable a , and assign it $b \in \{0,1\}$.

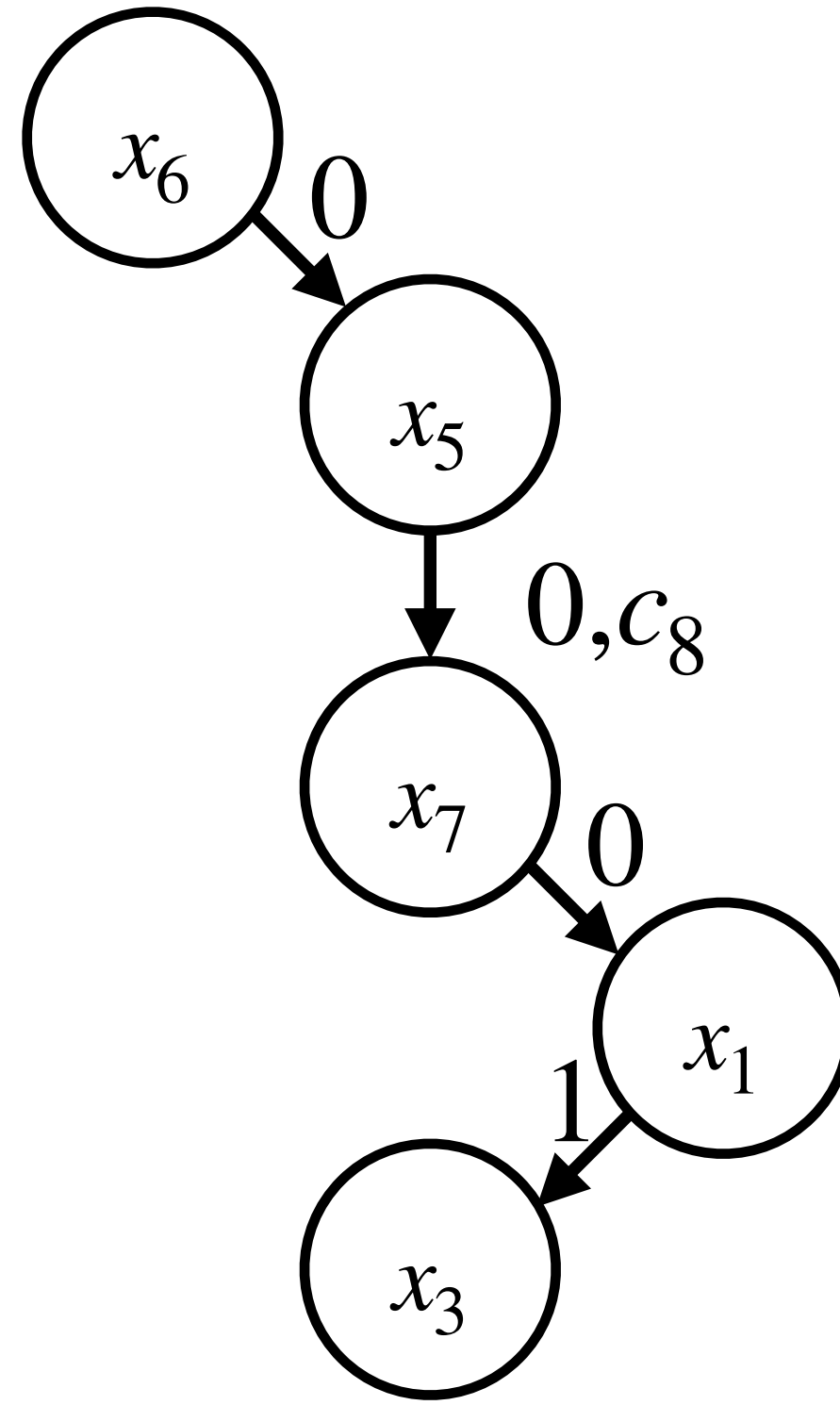
If $DPLL(f, m[a \rightarrow b]) = SAT$, return $m[a \rightarrow b]$

Else, return $DPLL(f, m[a \rightarrow 1 - b])$

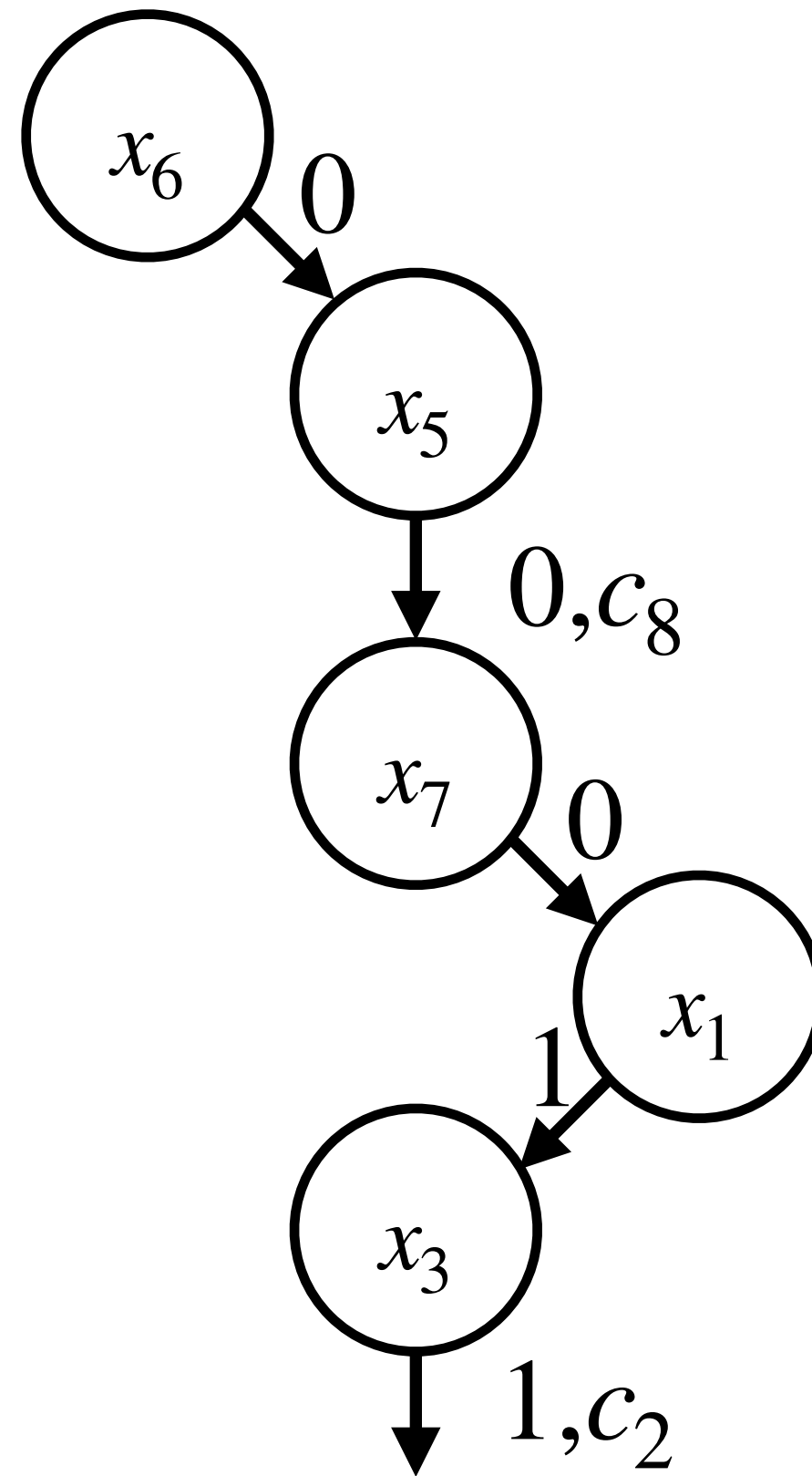
$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



$$c_1 = (\neg x_1 \vee x_2)$$

$$c_2 = (\neg x_1 \vee x_3 \vee x_5)$$

$$c_3 = (\neg x_2 \vee x_4)$$

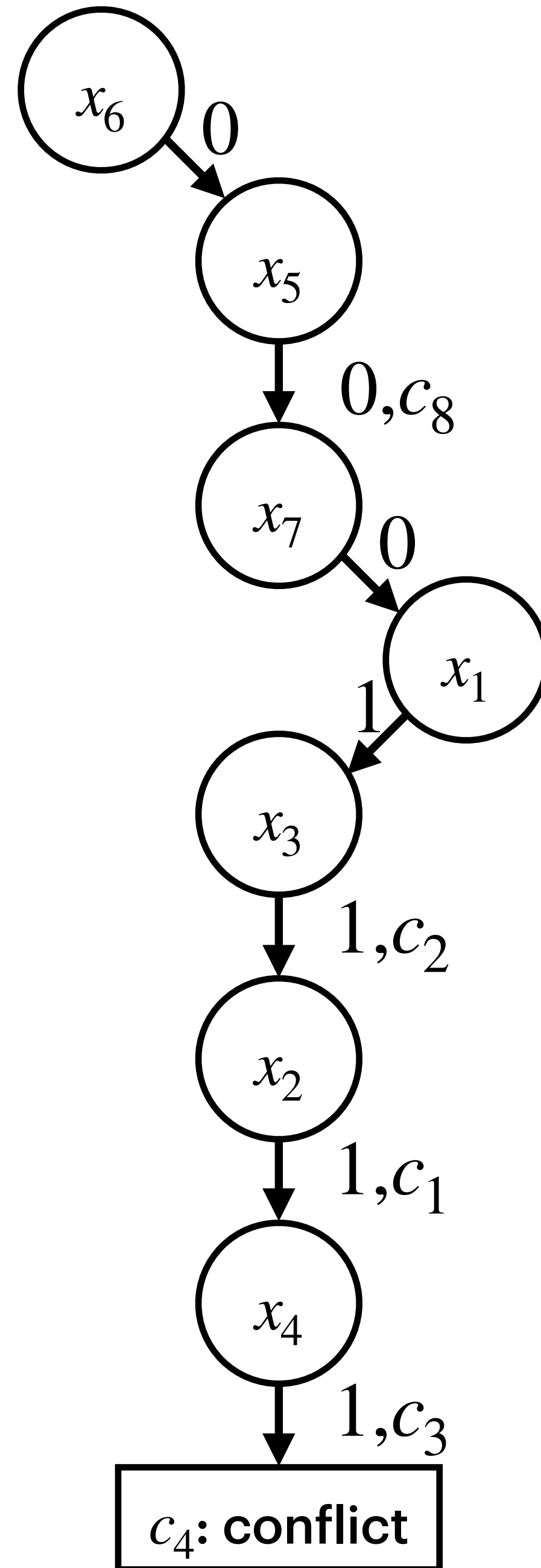
$$c_4 = (\neg x_3 \vee \neg x_4)$$

$$c_5 = (x_1 \vee x_5 \vee \neg x_2)$$

$$c_6 = (x_2 \vee x_3)$$

$$c_7 = (x_2 \vee \neg x_3 \vee x_7)$$

$$c_8 = (x_6 \vee \neg x_5)$$



DAVIS–PUTNAM–LOGEMANN–LOVELAND (DPLL) ALGORITHM

Input: CNF f , and partial assignment m

If f is true under m , return m .

If f is false under m , return \perp .

If \exists unit literal p under m , then return $DPLL(f, m[p \rightarrow 1])$.

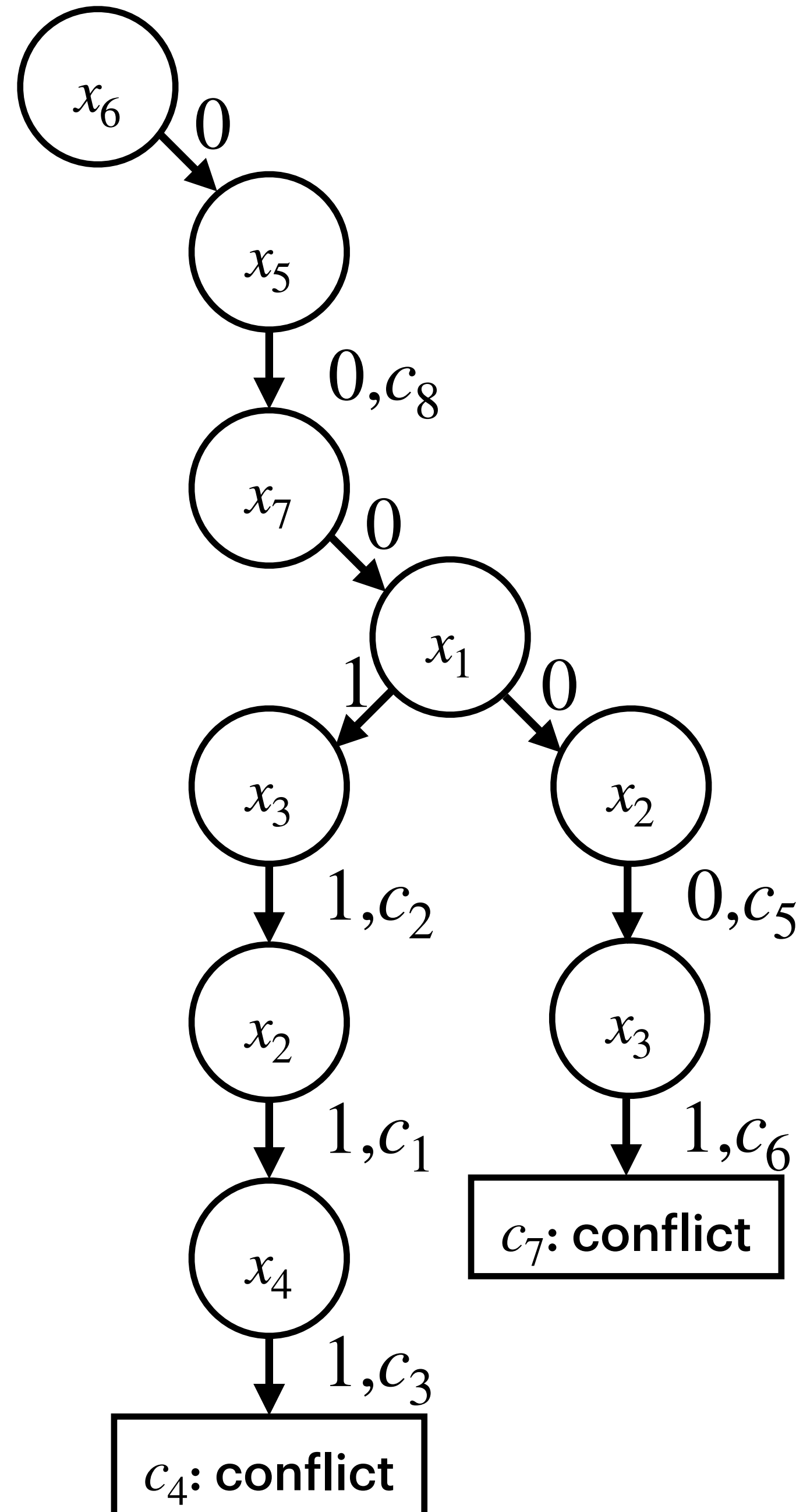
If \exists unit literal $\neg p$ under m , then return $DPLL(f, m[p \rightarrow 0])$.

Chose an unassigned variable a , and assign it $b \in \{0,1\}$.

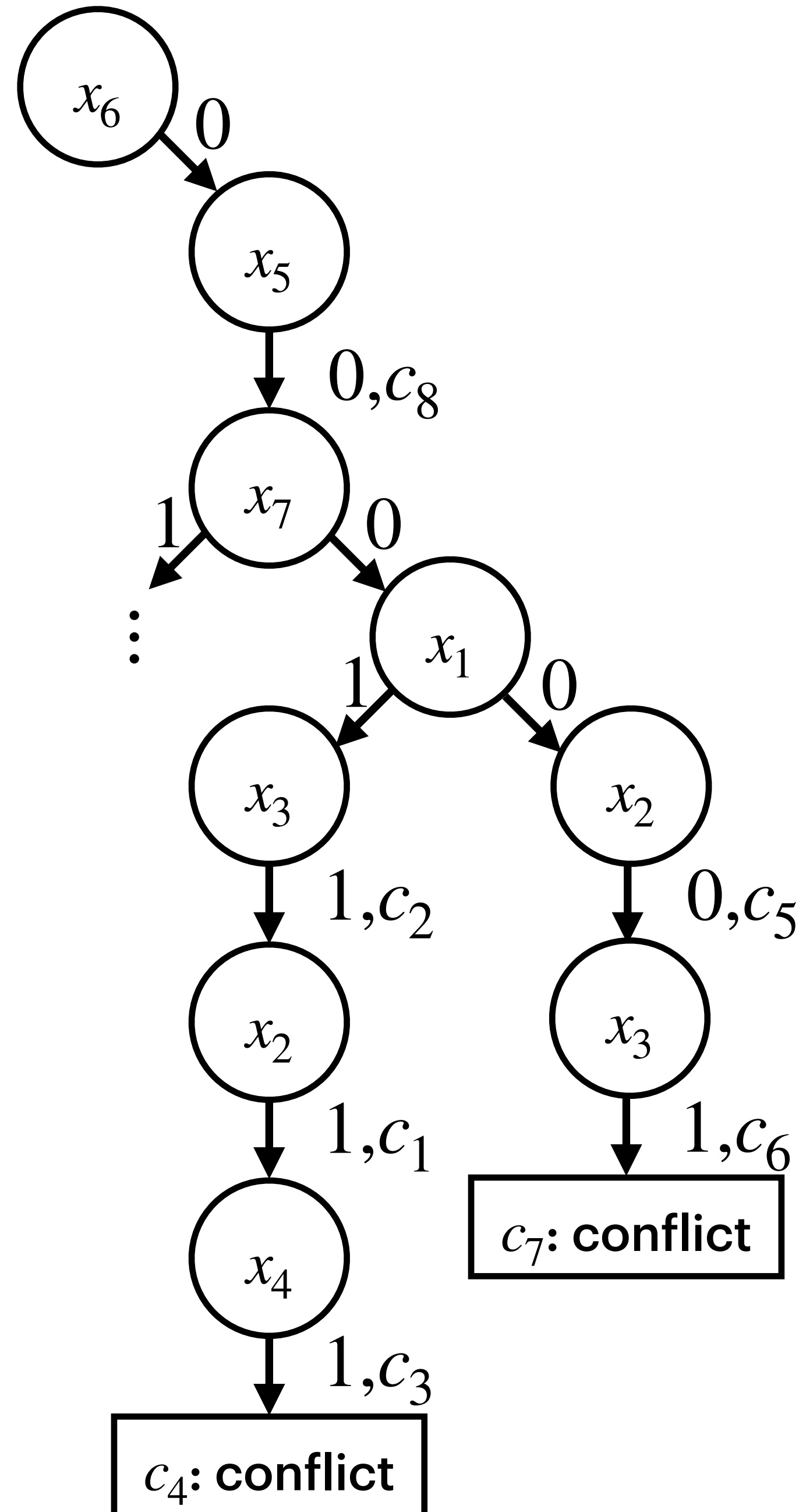
If $DPLL(f, m[a \rightarrow b]) = SAT$, return $m[a \rightarrow b]$

Else, return $DPLL(f, m[a \rightarrow 1 - b])$

$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



$$\begin{aligned}
c_1 &= (\neg x_1 \vee x_2) \\
c_2 &= (\neg x_1 \vee x_3 \vee x_5) \\
c_3 &= (\neg x_2 \vee x_4) \\
c_4 &= (\neg x_3 \vee \neg x_4) \\
c_5 &= (x_1 \vee x_5 \vee \neg x_2) \\
c_6 &= (x_2 \vee x_3) \\
c_7 &= (x_2 \vee \neg x_3 \vee x_7) \\
c_8 &= (x_6 \vee \neg x_5)
\end{aligned}$$



An Extension:

$$p := A \mid p \wedge p \mid p \vee p \mid \neg p$$

An Extension:

$$p := A \mid p \wedge p \mid p \vee p \mid \neg p$$

$$A := (e = e)$$

An Extension:

$$p := A \mid p \wedge p \mid p \vee p \mid \neg p$$

$$A := (e = e)$$

$$e \in \mathbb{R} \cup V \qquad e := e \smile e$$

$$\smile := + \mid -$$

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

Are there (x, y) such that p can be satisfied?

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

A

B

C

Are there (x, y) such that p can be satisfied?

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{A : 0, B : 1, C : 0\}$$

$$\{A : 0, B : 0, C : 1\}$$

$$\{A : 0, B : 1, C : 1\}$$

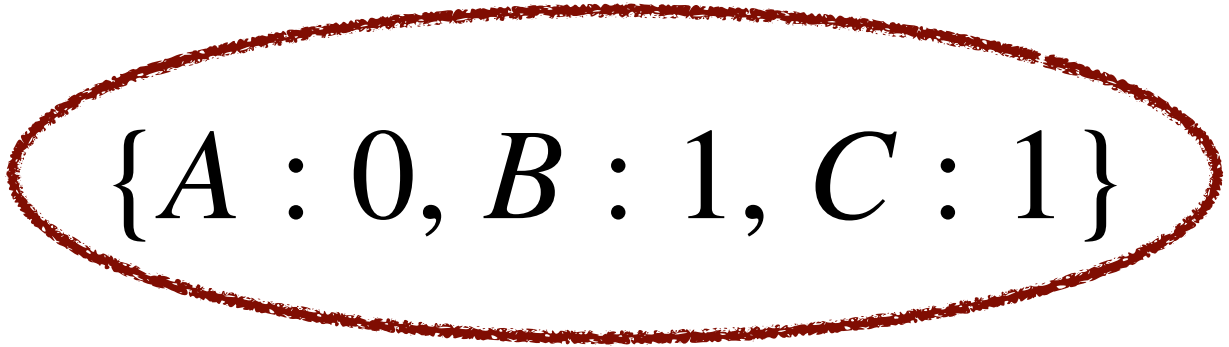
An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{(x, y) \mid x \neq 0, x + y = 3.5, y - x = 2\}$$


$$\{A : 0, B : 1, C : 1\}$$

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{(x, y) \mid x \neq 0, x + y = 3.5, y - x = 2\}$$

$$\{(0.75, 2.75)\}$$

$$\{A : 0, B : 1, C : 1\}$$

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y + x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{(x, y) \mid x \neq 0, x + y = 3.5, y + x = 2\}$$

$$\{A : 0, B : 1, C : 1\}$$

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y + x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{A : 0, B : 1, C : 0\}$$

$$\{A : 0, B : 0, C : 1\}$$

$$\{A : 0, B : 1, C : 1\}$$

An Example:

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y + x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{A : 0, B : 1, C : 0\}$$

$$\{A : 0, B : 0, C : 1\}$$

$$\{A : 0, B : 1, C : 1\}$$

Why does this work?

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{(x, y) \mid x \neq 0, x + y = 3.5, y - x = 2\}$$

$$\{(0.75, 2.75)\}$$

$$\{A : 0, B : 1, C : 1\}$$

Why does this work?

$$p : \neg(x = 0) \wedge ((x + y = 3.5) \vee (y - x = 2))$$

$$p' : \neg A \wedge (B \vee C)$$

Are there (x, y) such that p can be satisfied?

$$\{(x, y) \mid x \neq 0, x + y = 3.5, y - x = 2\} \quad \text{Decidability of SLE}$$

$$\{(0.75, 2.75)\}$$

$$\{A : 0, B : 1, C : 1\}$$



SRI-CSL/**yices2**

The Yices SMT Solver



diffblue
AI for Code

diffblue/cbmc

C Bounded Model Checker



Astrée



Polyspace
Static Code Analysis

Semmlé™

Part III: Automated Program Synthesis

Synthesis: Dreams \Rightarrow Programs

ZOHAR MANNA AND RICHARD WALDINGER

Abstract—Deductive techniques are presented for deriving programs systematically from given specifications. The specifications express the purpose of the desired program without giving any hint of the algorithm to be employed. The basic approach is to transform the specifications repeatedly according to certain rules, until a satisfactory program is produced. The rules are guided by a number of strategic controls. These techniques have been incorporated in a running program-synthesis system, called DEDALUS.

Many of the transformation rules represent knowledge about the program's subject domain (e.g., numbers, lists, sets); some represent the meaning of the constructs of the specification language and the target programming language; and a few rules represent basic programming principles. Two of these principles, the *conditional-formation rule* and the *recursion-formation rule*, account for the introduction of conditional expressions and of recursive calls into the synthesized program. The termination of the program is ensured as new recursive calls are formed.

Two extensions of the recursion-formation rule are discussed: a *procedure-formation rule*, which admits the introduction of auxiliary subroutines in the course of the synthesis process, and a *generalization rule*, which causes the specifications to be altered to represent a more general problem that is nevertheless easier to solve. Special techniques are introduced for the formation of programs with side effects.

The techniques of this paper are illustrated with a sequence of examples of increasing complexity; programs are constructed for list processing, numerical calculation, and array computation.

The methods of program synthesis can be applied to various aspects of programming methodology—program transformation, data abstrac-

INTRODUCTION

IN RECENT years there has been increasing activity in the field of program verification. The goal of these efforts is to construct computer systems for determining whether a given program is correct, in the sense of satisfying given specifications. These attempts have met with increasing success; while automatic proofs of the correctness of large programs may be a long way off, it seems evident that the techniques being developed will be useful in practice, to find the bugs in faulty programs and to give us confidence in correct ones.

The general scenario of the verification system is that a programmer will present his completed computer program, along with its specifications and associated documentation, to a system which will then prove or disprove its correctness. It has been pointed out, most notably by the advocates of structured programming, that this is "putting the cart before the horse." Once we have techniques for proving program correctness, why should we wait to apply them until after the program is complete? Instead, why not ensure the correctness of the program while it is being constructed, thereby developing the program and its correctness proof "hand in hand"?

The point is particularly well-taken when we consider that

Dream



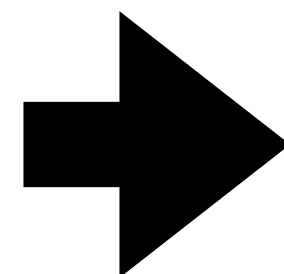
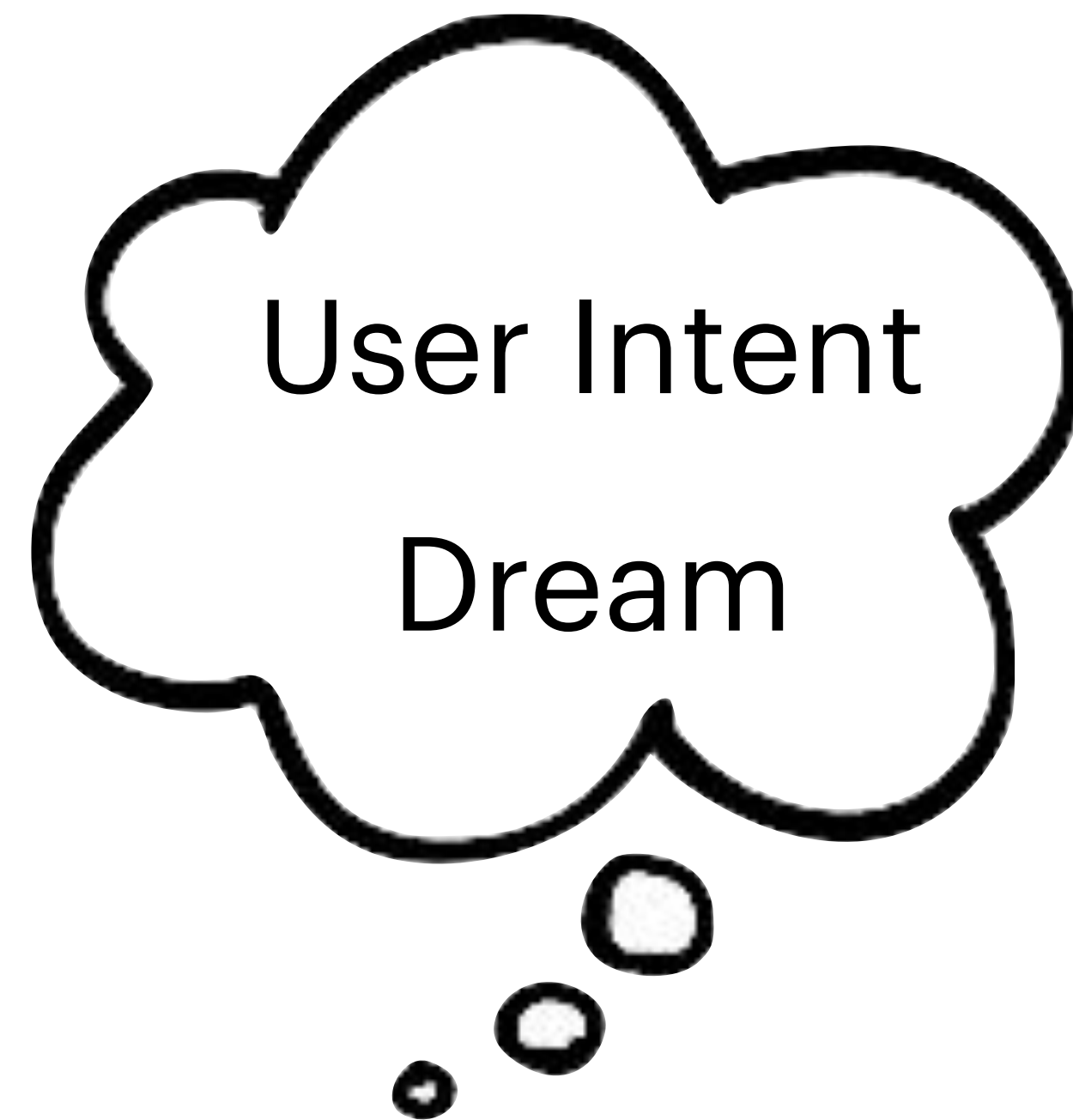
User Intent
Dream



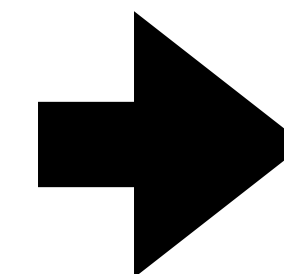
User Intent
Dream

Program
Synthesis





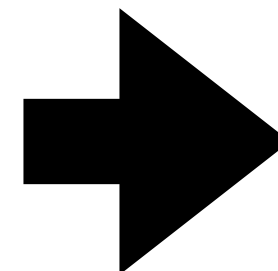
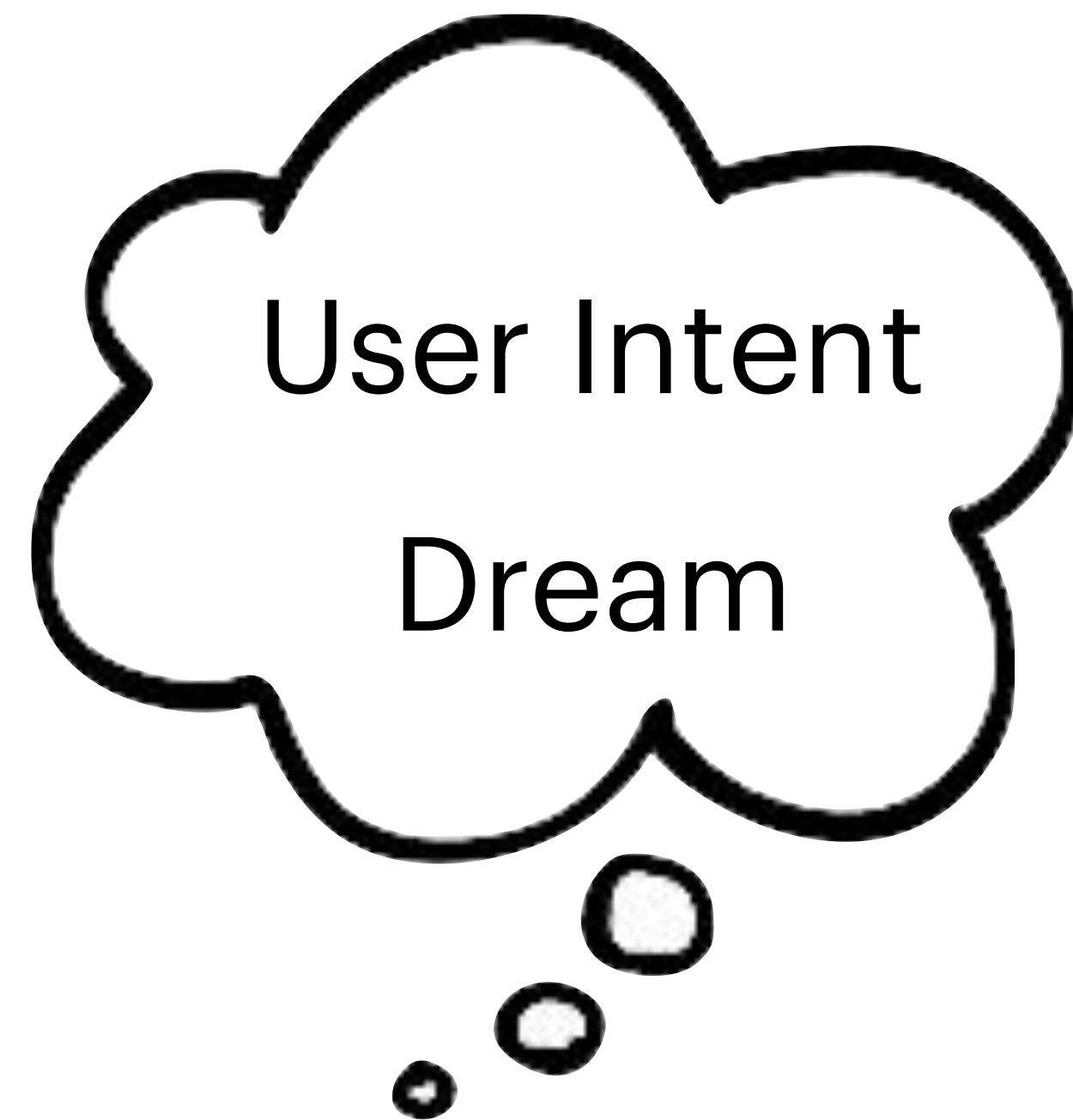
Program
Synthesis



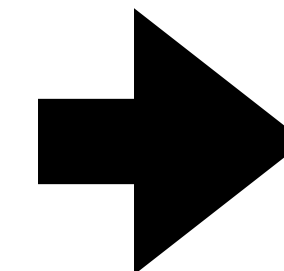
```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



How to describe?



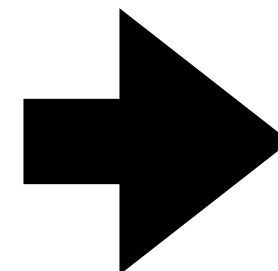
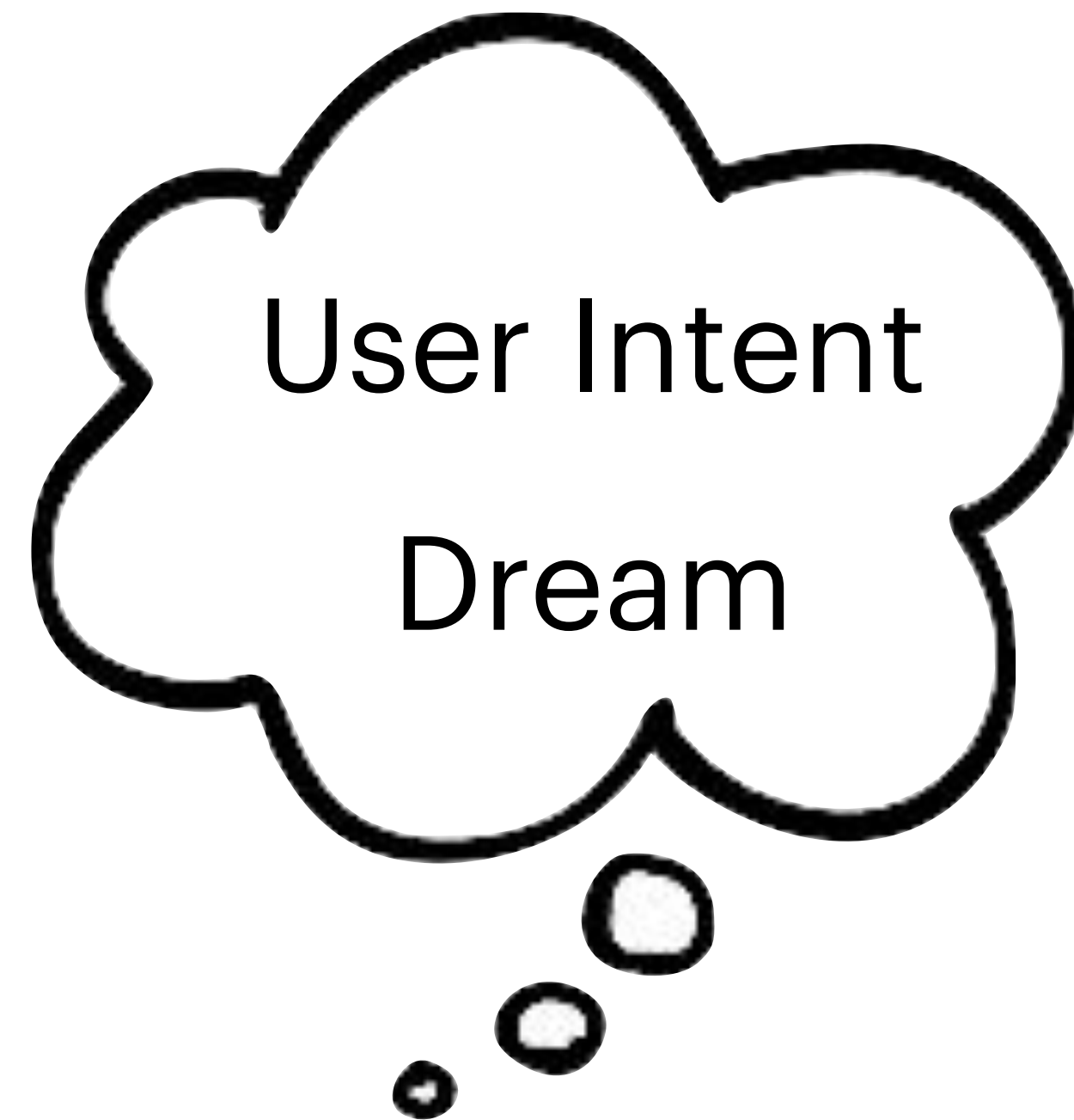
Program
Synthesis



```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```

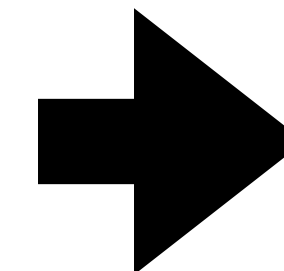


How to describe?



How to design?

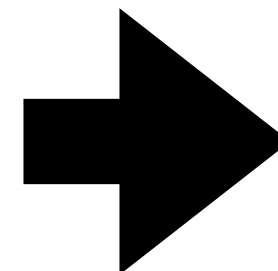
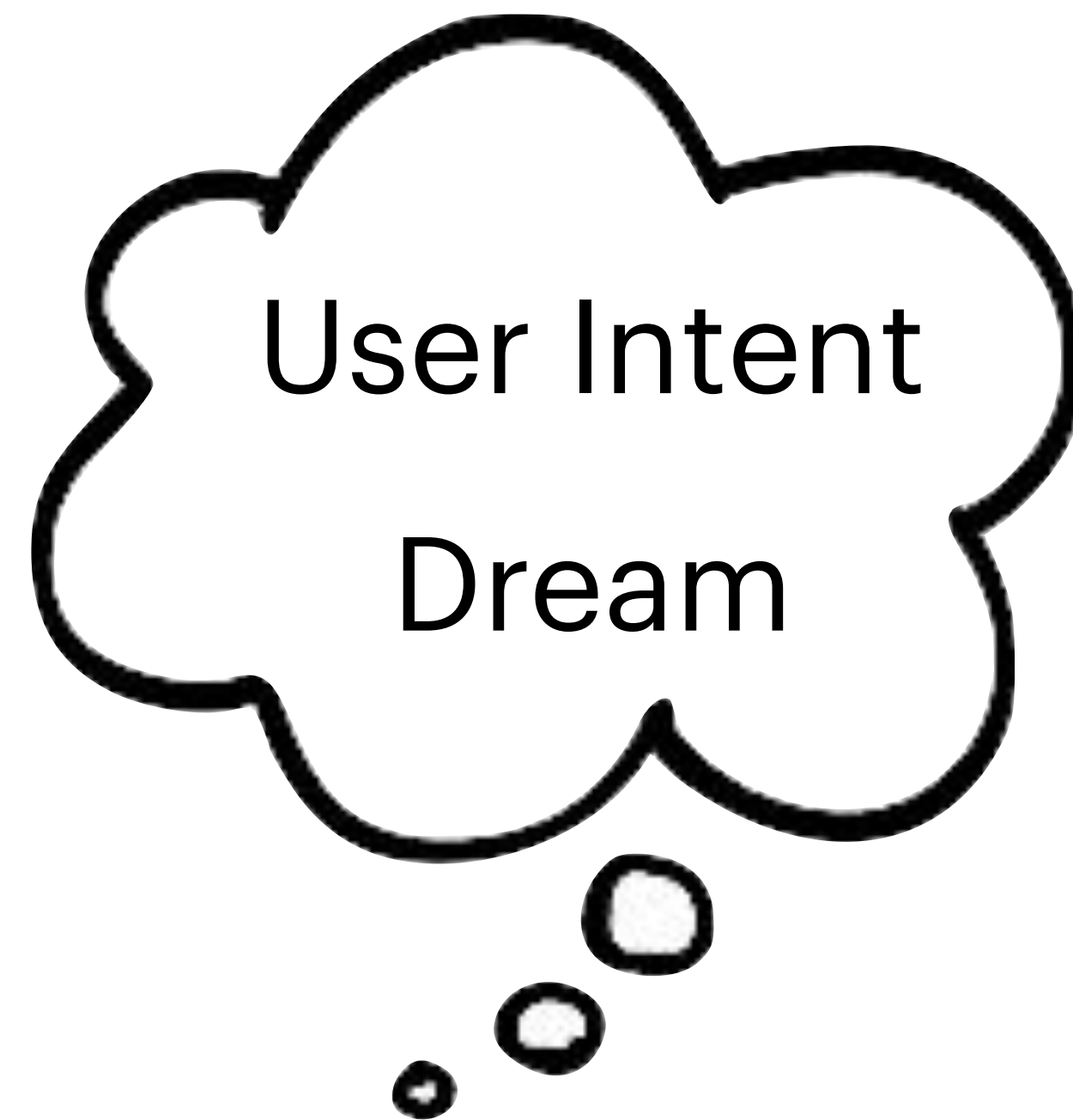
Program
Synthesis



```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```

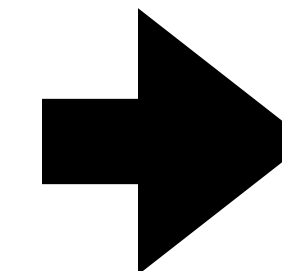


How to describe?



How to design?

Program
Synthesis



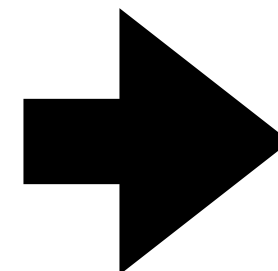
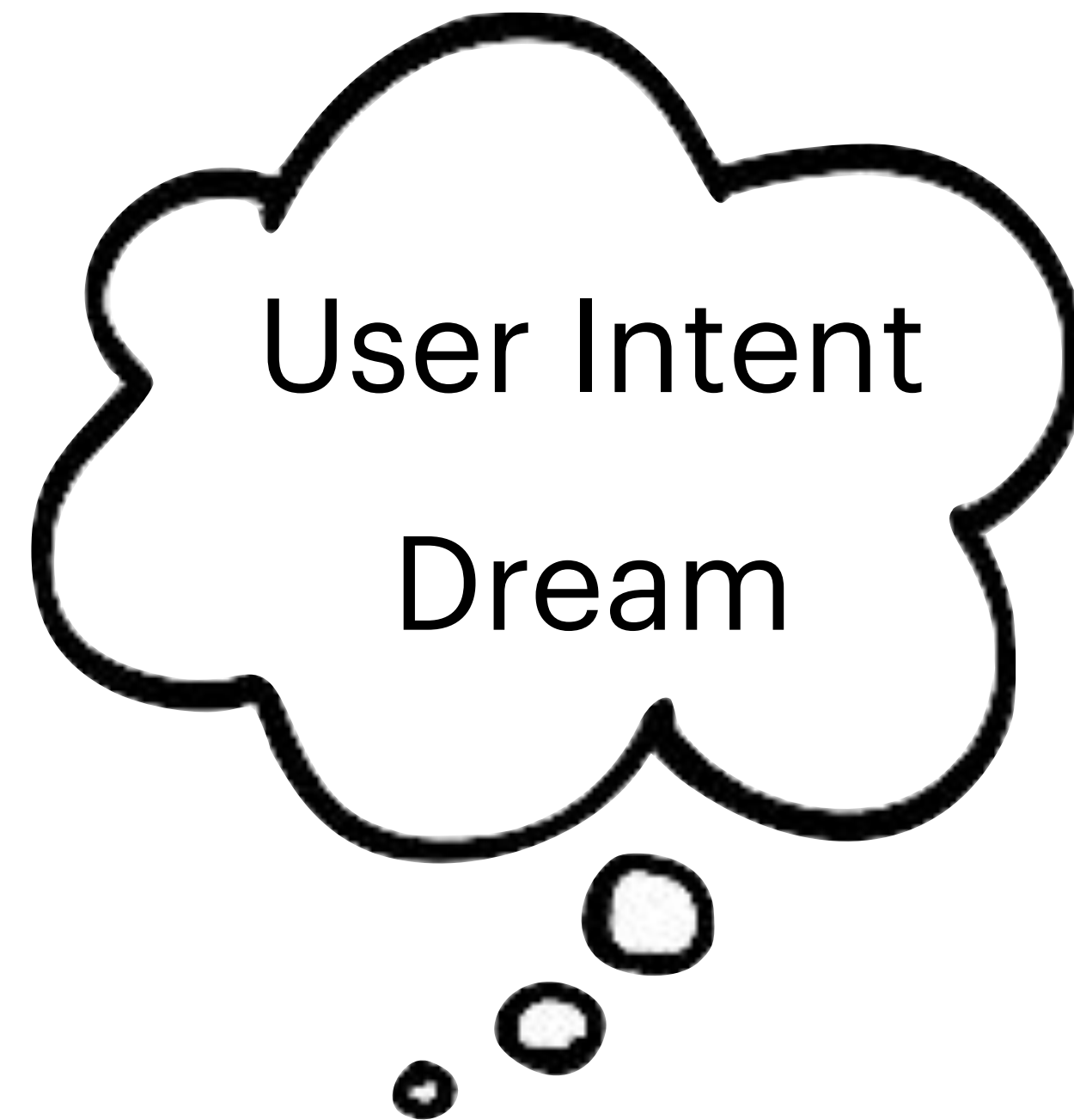
How to ensure?

```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



SPECIFICATION

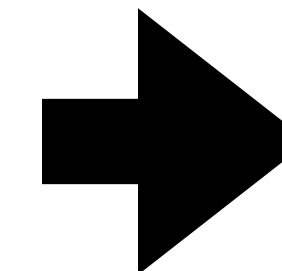
How to describe?



SYNTHESIS

How to design?

Program
Synthesis



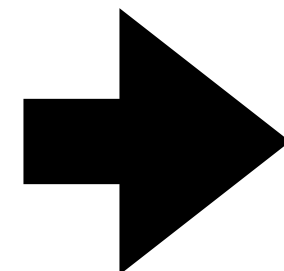
VERIFICATION

How to ensure?

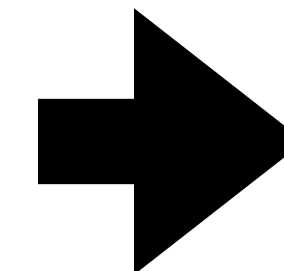
```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



Complete
Mathematical
Specification



Program
Synthesis



```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



```
1  def max(a, b, c):  
2      m = a  
3      if b > a: m = b  
4      if c > m: m = c  
5      return m
```

```
1  def max(a, b, c):  
2      m = a  
3      if b > a: m = b  
4      if c > m: m = c  
5      return m
```

$$\forall x, y, z. (max(x, y, z) \geq x) \wedge (max(x, y, z) \geq y) \wedge (max(x, y, z) \geq z) \\ \wedge \left((max(x, y, z) = x) \vee (max(x, y, z) = y) \vee (max(x, y, z) = z) \right)$$

Verification

```
1  def max(a, b, c):  
2      m = a  
3      if b > a: m = b  
4      if c > m: m = c  
5      return m
```

$$\forall x, y, z. (max(x, y, z) \geq x) \wedge (max(x, y, z) \geq y) \wedge (max(x, y, z) \geq z) \\ \wedge \left((max(x, y, z) = x) \vee (max(x, y, z) = y) \vee (max(x, y, z) = z) \right)$$

Verification

```
1  def max(a, b, c):  
2      m = a  
3      if b > a: m = b  
4      if c > m: m = c  
5      return m
```


Function of three numbers that returns their max.

Write python code for a function of three numbers that returns their max.



Here's the Python code for a function that returns the maximum of three numbers using comparisons:

python

 Copy code

```
def max_of_three(a, b, c):  
    if a >= b and a >= c:  
        return a  
    elif b >= a and b >= c:  
        return b  
    else:  
        return c  
  
# Example usage  
result = max_of_three(5, 10, 3)  
print(f"The maximum value is: {result}")
```

Mathematical Specification

Natural Language

Mathematical Specification

Input-Output Examples

Natural Language

Input-Output Examples

```
1  def max(a, b, c):  
2      m = a  
3      if b > a: m = b  
4      if c > m: m = c  
5      return m
```

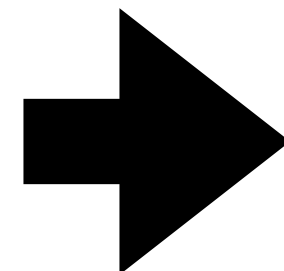
$$\text{max}(1,2,4) = 4$$

$$\text{max}(8,3, -4) = 8$$

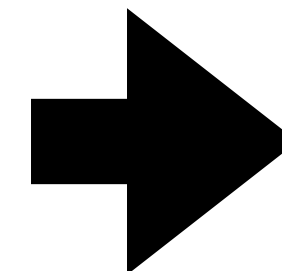
$$\text{max}(10,10,1) = 10$$

$$\text{max}(1,6,4) = 6$$

$\max(1,2,4) = 4$
 $\max(8,3,-4) = 8$
 $\max(10,10,1) = 10$
 $\max(1,6,4) = 6$



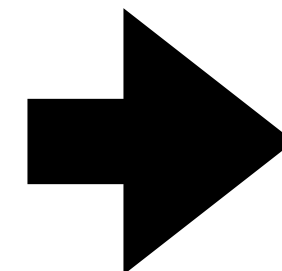
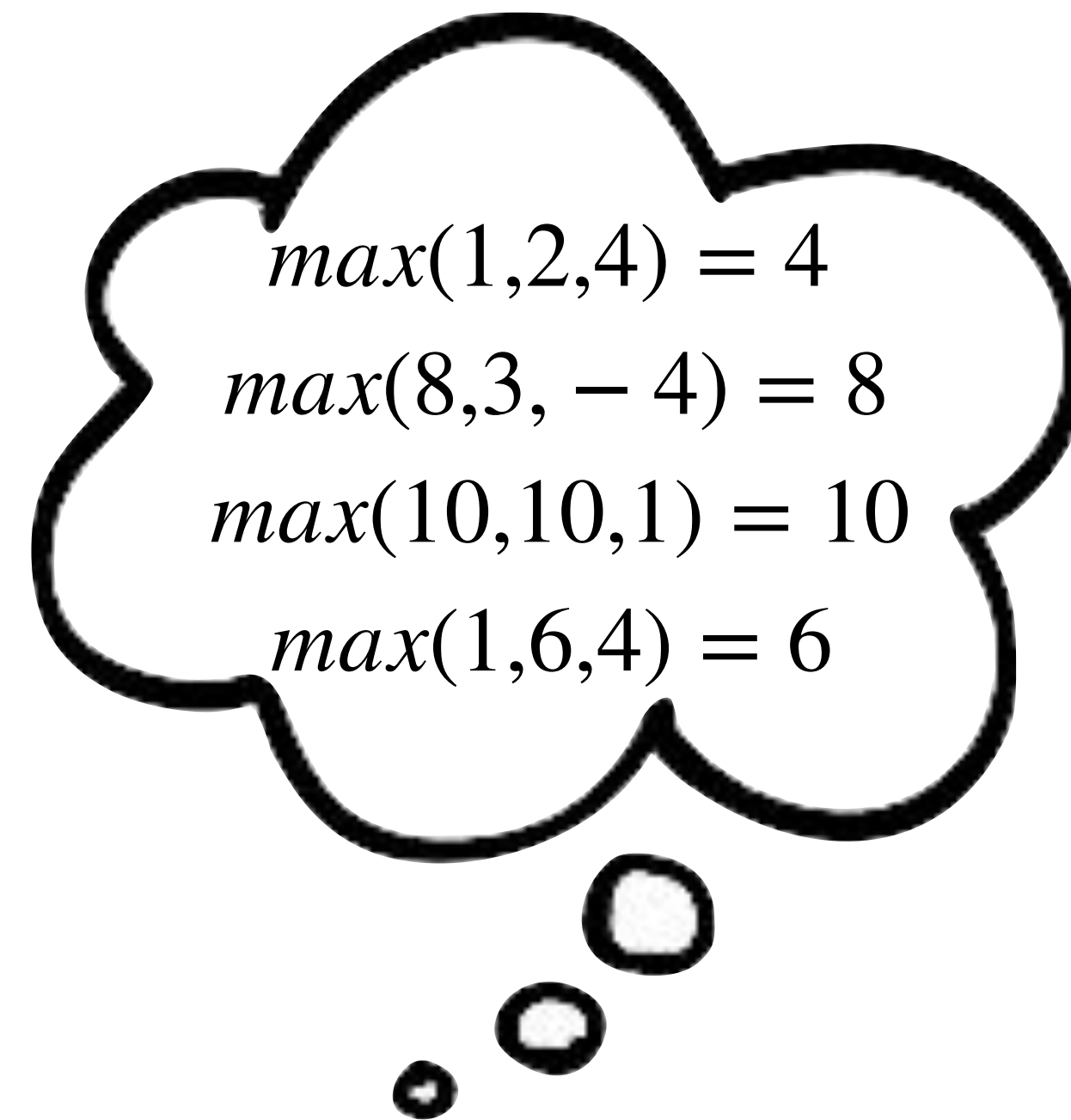
Program
Synthesis



```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```

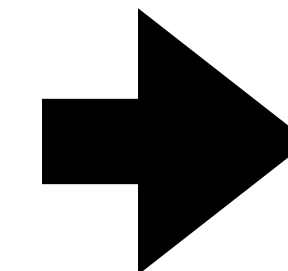


Easy to describe



Search Problem
Easy to automate and scale

Program
Synthesis

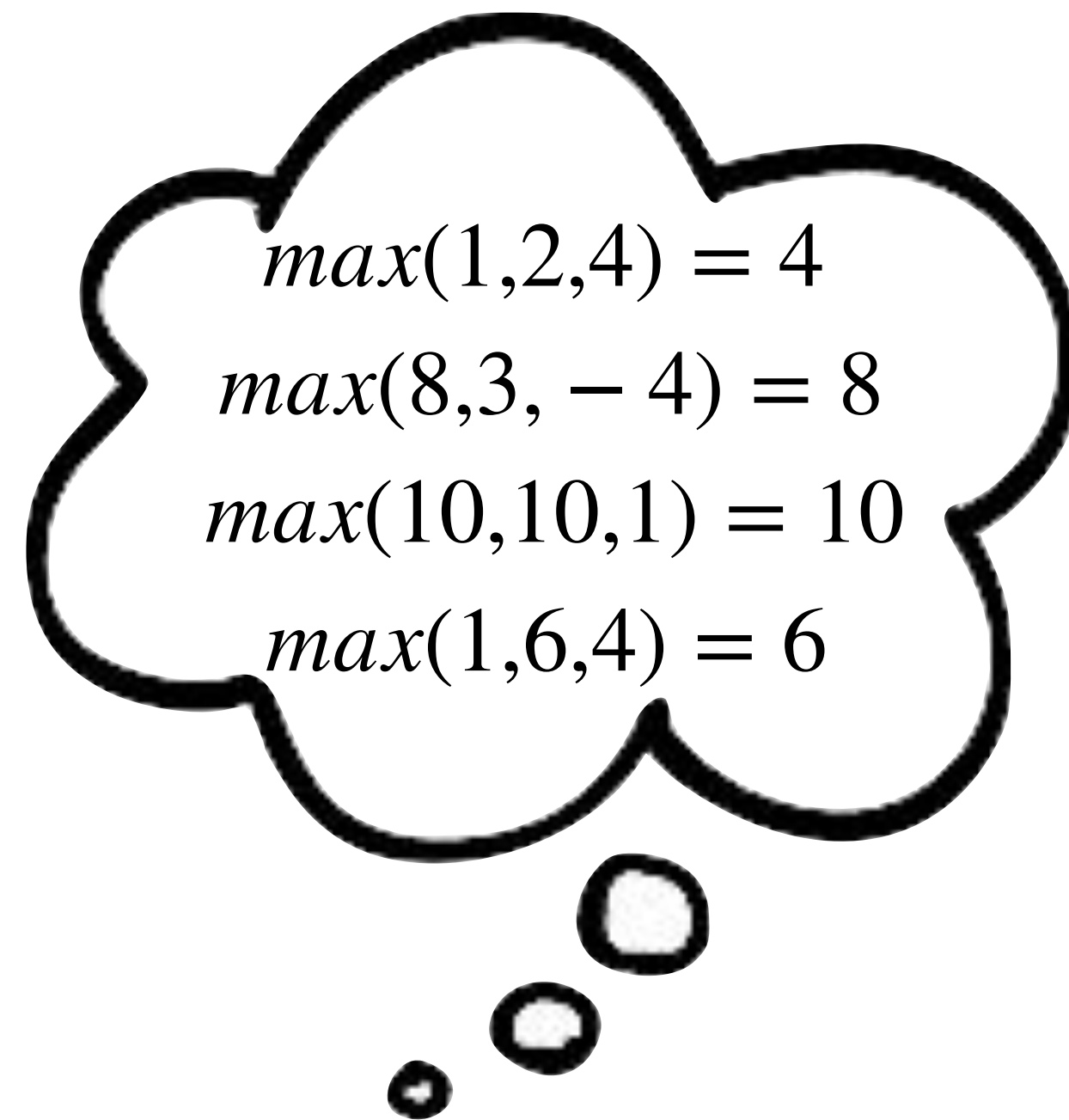


Easy to ensure!

```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



Easy to describe



Search Problem
Easy to automate and scale

Program
Synthesis

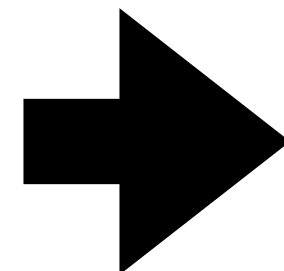
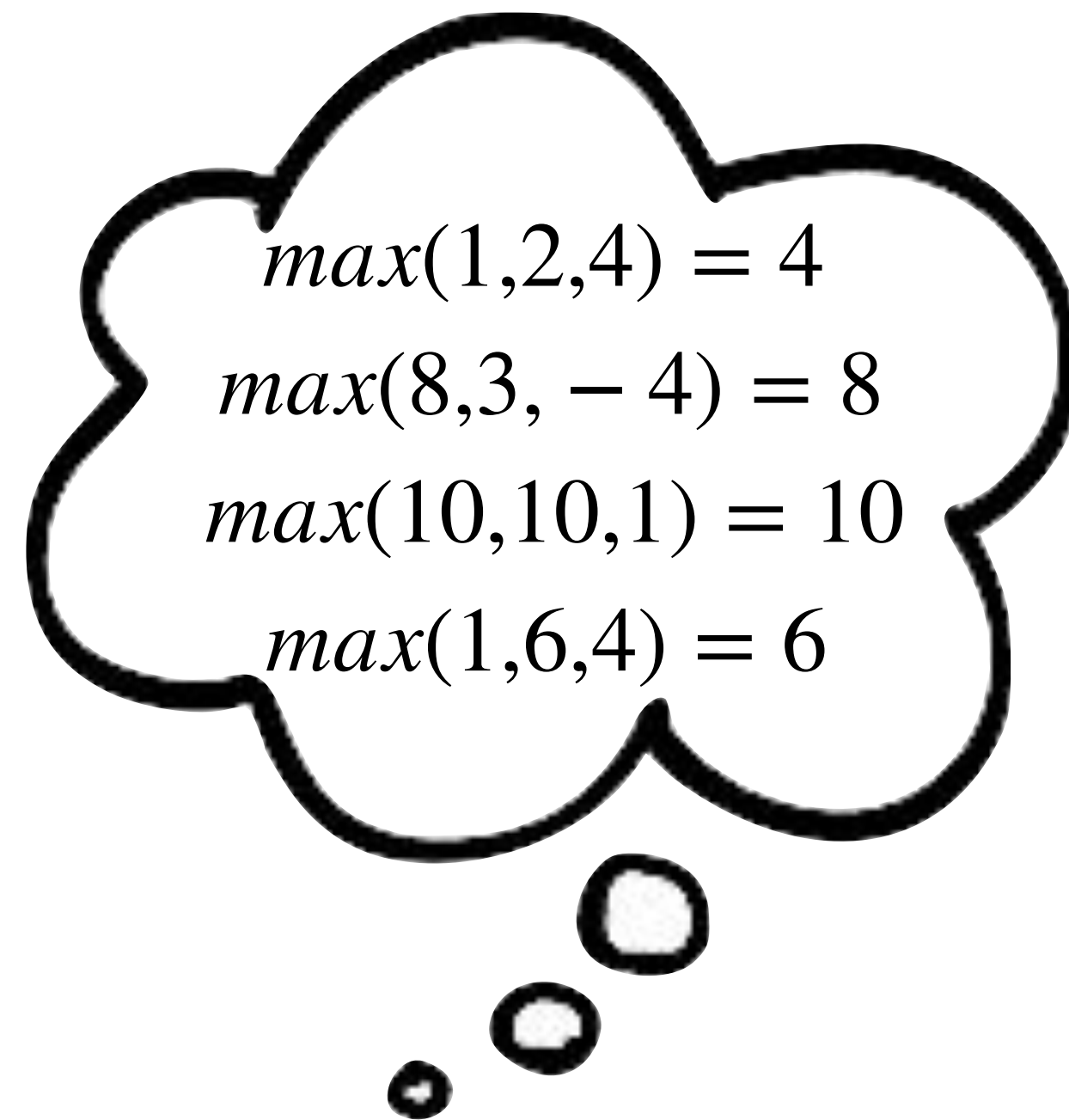
Easy to ensure!

```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



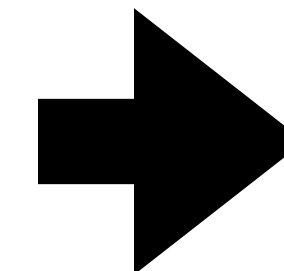
Correctness?

Easy to describe



Search Problem
Easy to automate and scale

Program
Synthesis



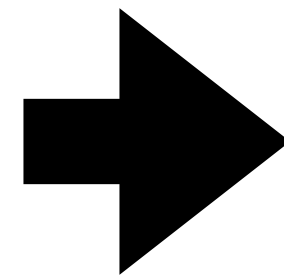
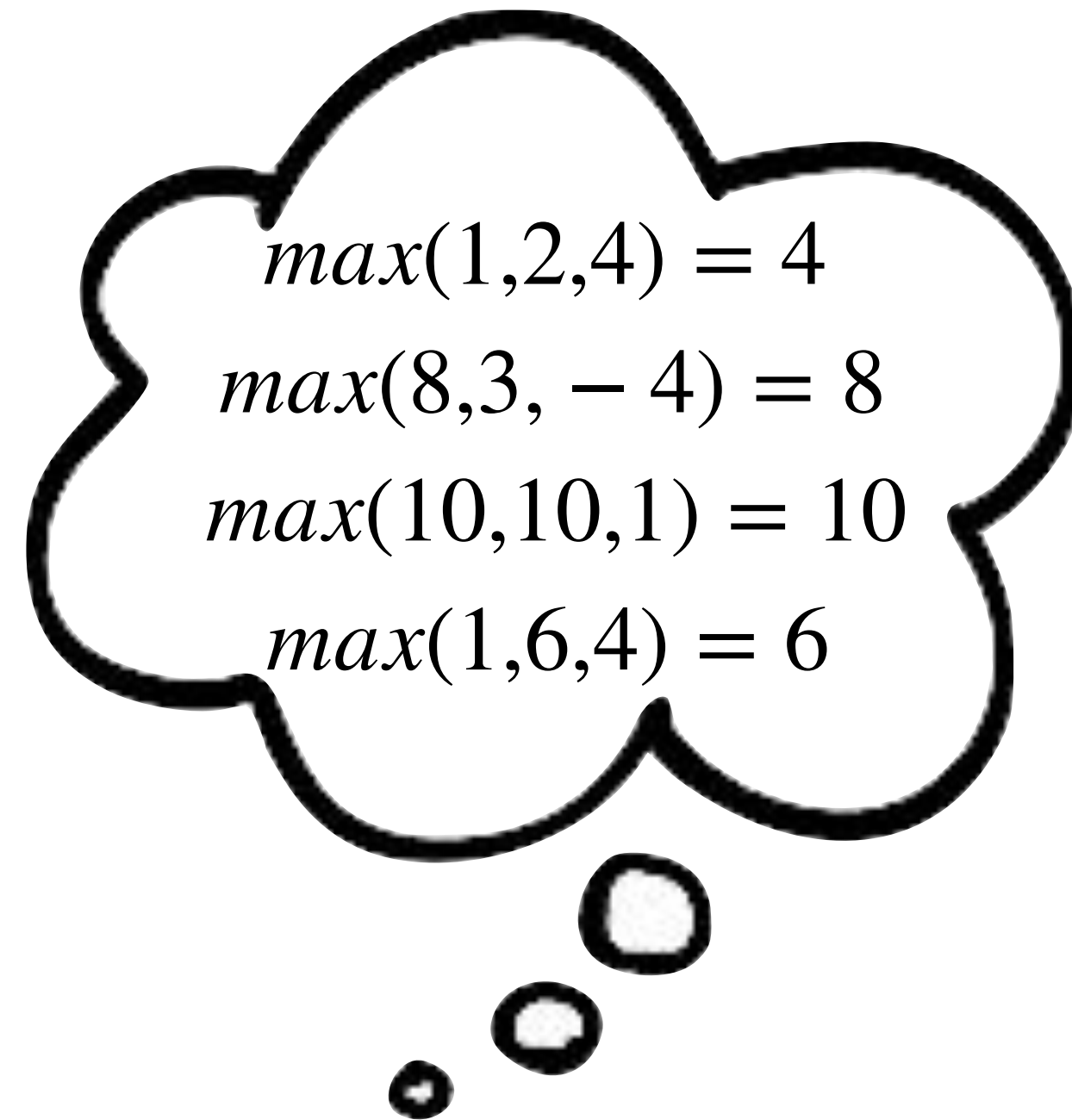
Easy to ensure!

```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```



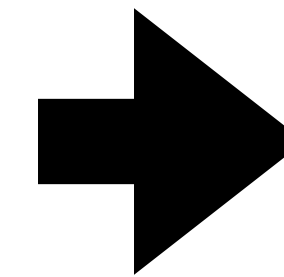
Ambiguity?
Correctness?
Overfitting?

Easy to describe



Search Problem
Easy to automate and scale

Program
Synthesis



Easy to ensure!

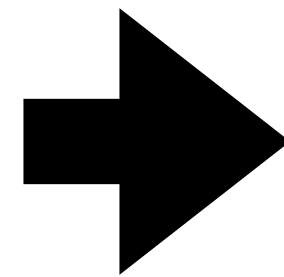
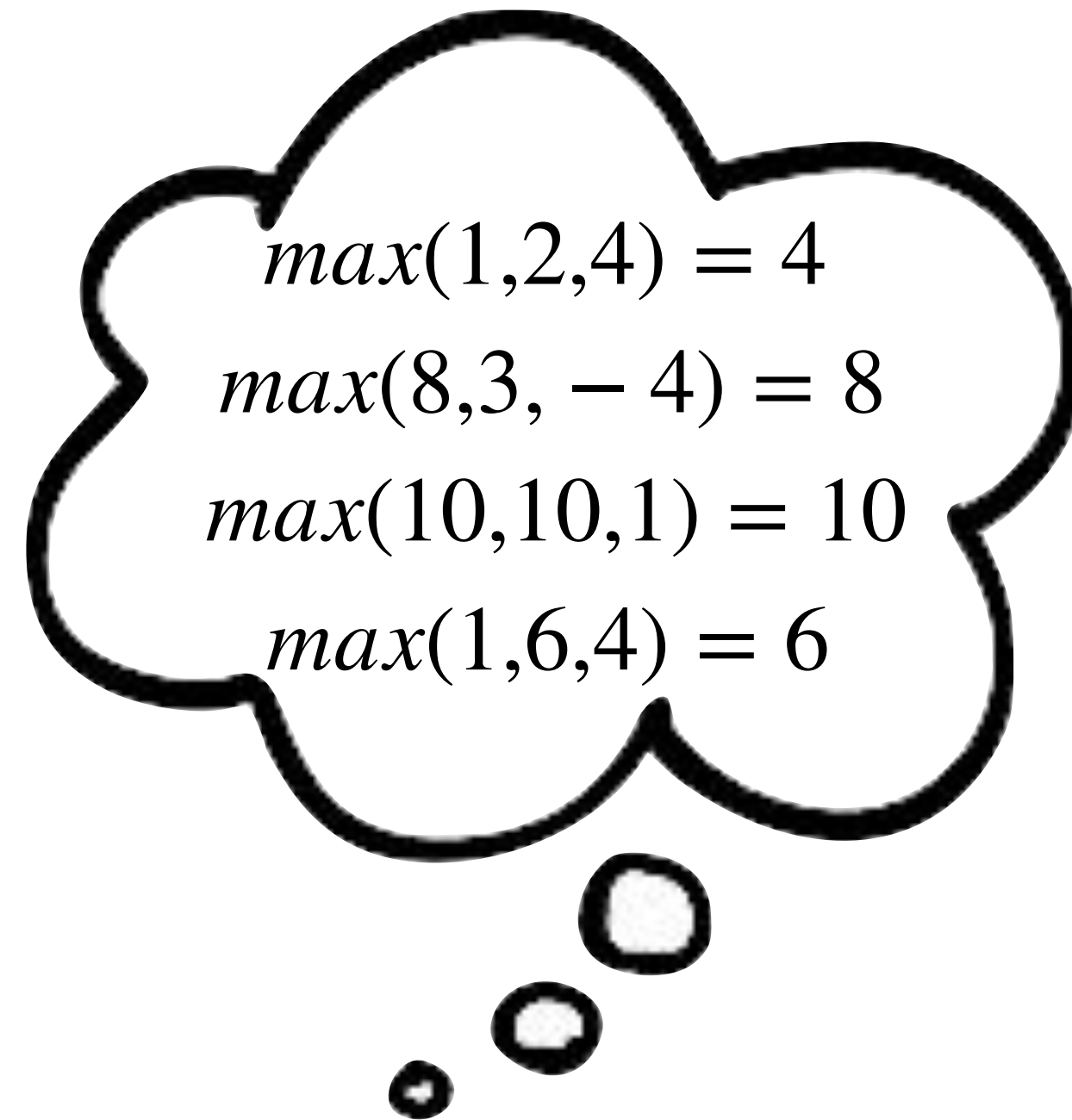
```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```

```
def max(a, b, c):  
    m = c  
    if b > m: m = b  
    if a > b: m = a  
    return m
```



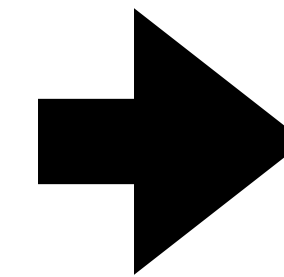
Ambiguity!

Easy to describe



Search Problem
Easy to automate and scale

Program
Synthesis



Easy to ensure!

```
def max(a, b, c):  
    m = a  
    if b > a: m = b  
    if c > m: m = c  
    return m
```

```
def max(a, b, c):  
    m = c  
    if b > m: m = b  
    if a > b: m = a  
    return m
```



Overfitting!

Example-guided Synthesis of Relational Queries

Example-guided Synthesis of Relational Queries

declarative logic programs

SQL

Datalog

Cypher

SPARQL

Example-guided Synthesis of Relational Queries

declarative logic programs

PQL

Prolog

LogiQL

CodeQL

Program Analysis



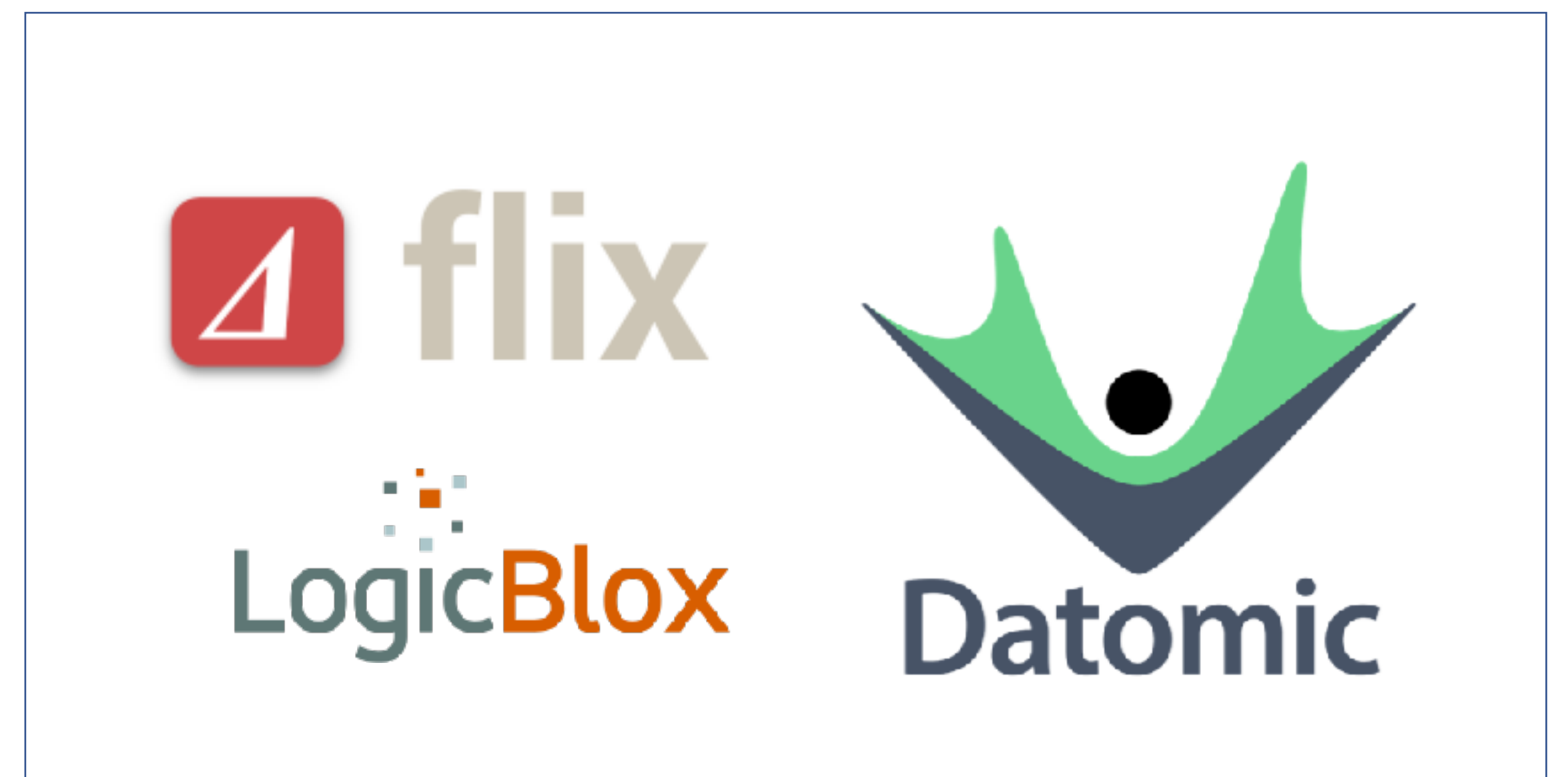
Network Analysis



Knowledge Discovery

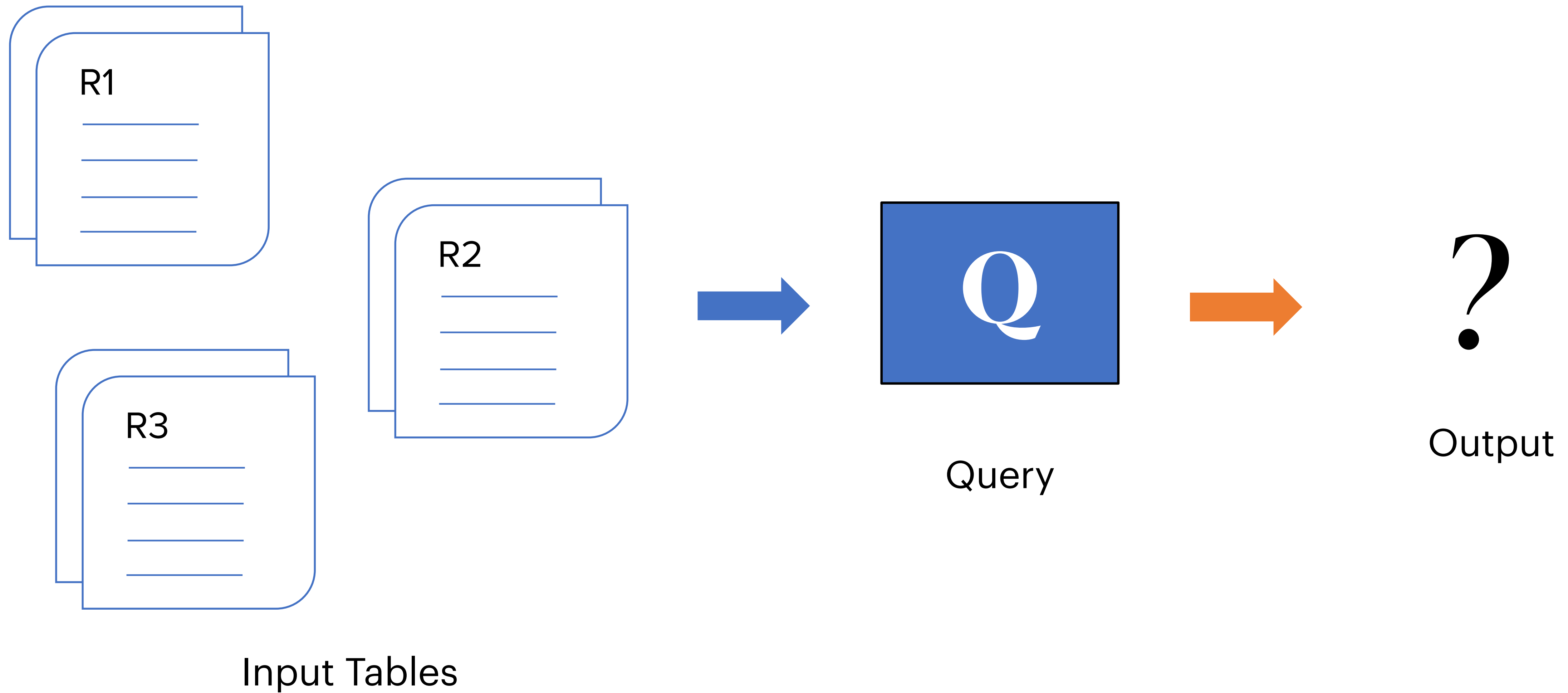


Database Querying



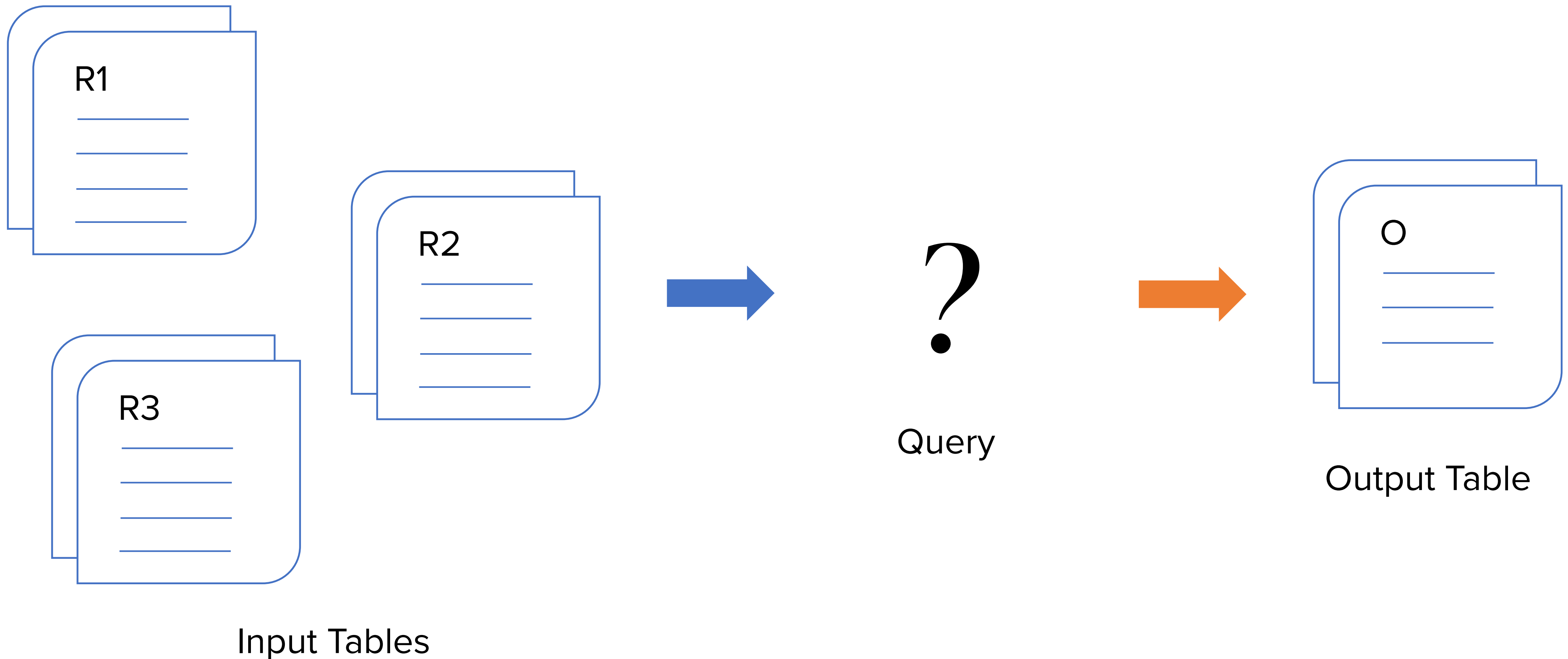
Industrial Applications

Querying



Example-guided Synthesis of Relational Queries

Query Synthesis



End User



Select
rows with
maximum value
for each user.

Find rows with
duplicate values.

Calculate
running average.



Select first row in each GROUP BY group?

As the title suggests, I'd like to select the first row of each set of rows grouped with a GROUP BY. Specifically, if I've got a purchases table that looks like this:

id	customer	total
1	Joe	5
2	Sally	3
3	Joe	2
4	Sally	1

I'd like to query for the (id of the largest purchase (total)) made by each (customer). Something like this:

```
SELECT FIRST(id), customer, FIRST(total)
FROM purchases
GROUP BY customer
ORDER BY total DESC;
```

FIRST(id)	customer	FIRST(total)
1	Joe	5
2	Sally	3

Finding duplicate values in a SQL table

It's easy to find duplicates with one field:

```
SELECT name, COUNT(email)
FROM users
GROUP BY email
HAVING COUNT(email) > 1;
```

So if we have a table

ID	NAME	EMAIL
1	John	john@domain.com
2	Sarah	sarah@domain.com
3	Tom	tom@domain.com
4	Bob	bob@domain.com
5	Tom	tom@domain.com

This query will give us John, Sarah, Tom, because they all have the same email. However, what I want is to get duplicates with the same email AND name. That is, I want to get "Tom", "Tom". The reason I need this: I made a mistake, and allowed to insert duplicate (name and email) values. Now I need to remove/change the duplicates, so I need to find them first.

Calculate a Running Total in SQL Server

Fast, reliable MySQL hosting

Imagine the following table (called 'TrackTable'):

id	timestamp	scoreValue
45	41/Jan/09	3
73	42/Jan/09	5
12	42/Feb/09	8
77	14/Feb/09	2
88	28/Feb/09	34
33	42/Mar/09	5

I would like a query that returns a running total in date order, like:

id	timestamp	scoreValue	runningTotal
45	41/Jan/09	3	3
73	42/Jan/09	5	8
12	42/Feb/09	8	16
77	14/Feb/09	2	18
88	28/Feb/09	34	52
33	42/Mar/09	5	57

Know there are various ways of doing this in SQL Server 2000 / 2005 / 2008. I am particularly interested in this sort of method that uses the aggregating set statement trick:

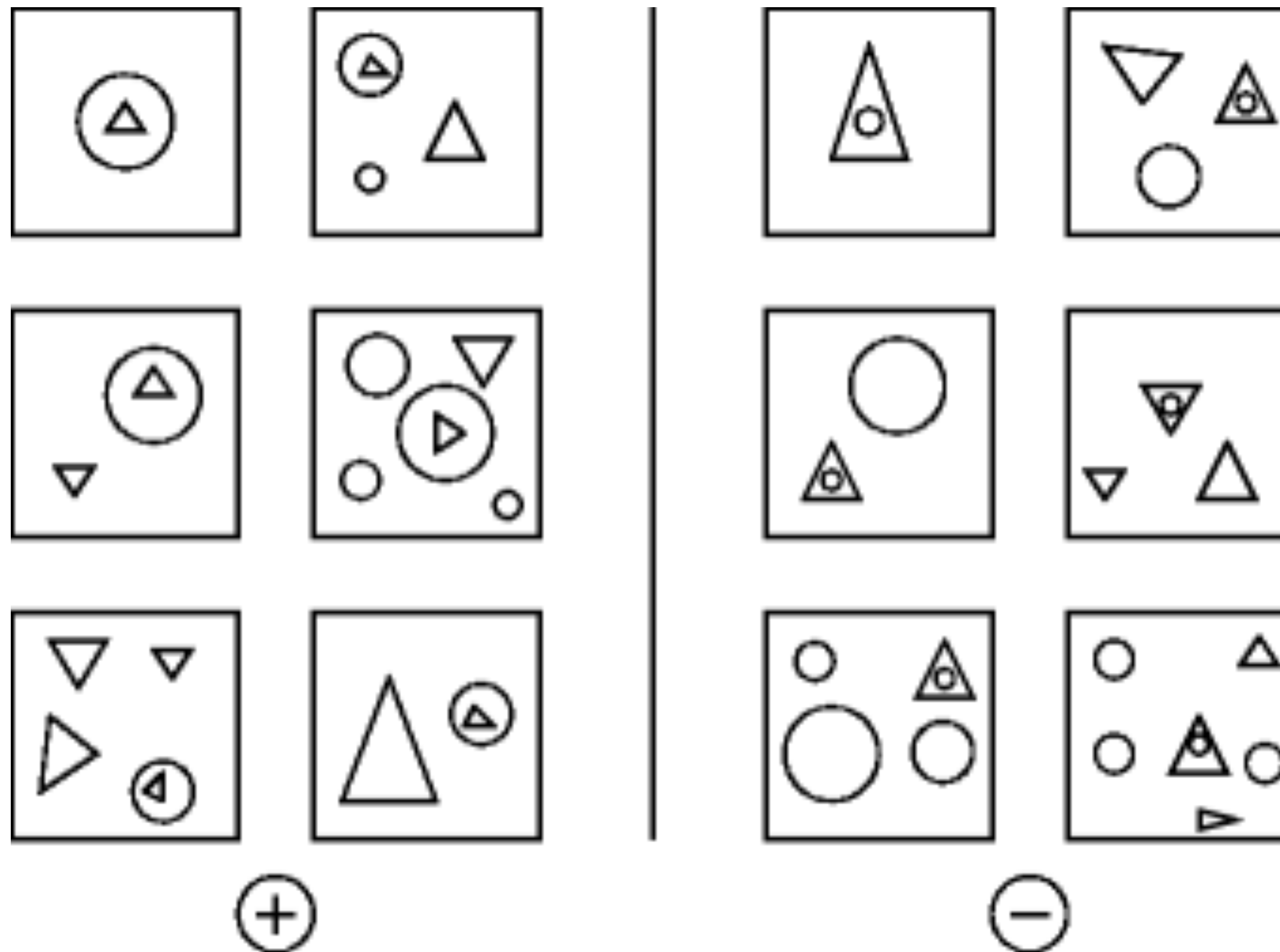
SQL

```
Select x.id, x.customer, x.total
From PURCHASES x
Join (Select p.customer,
          Max(total)
        From PURCHASES p
       Group By p.customer) y
On y.customer = x.customer
And y.max_total = x.total
```

```
Select *
From Users a
Where Exists
(Select *
 From Users b
 Where (a.name = b.name
        Or a.email = b.email)
 And a.ID <> b.id)
```

```
Select a.ord, a.val, Avg(b.val)
From t As a Join t As b
Where b.ord <= a.ord
Group By a.ord,a.val
Order By a.ord
```

Bongard problem 47



An Example

GreenSignal

Broadway

Liberty St

William St

Whitehall St

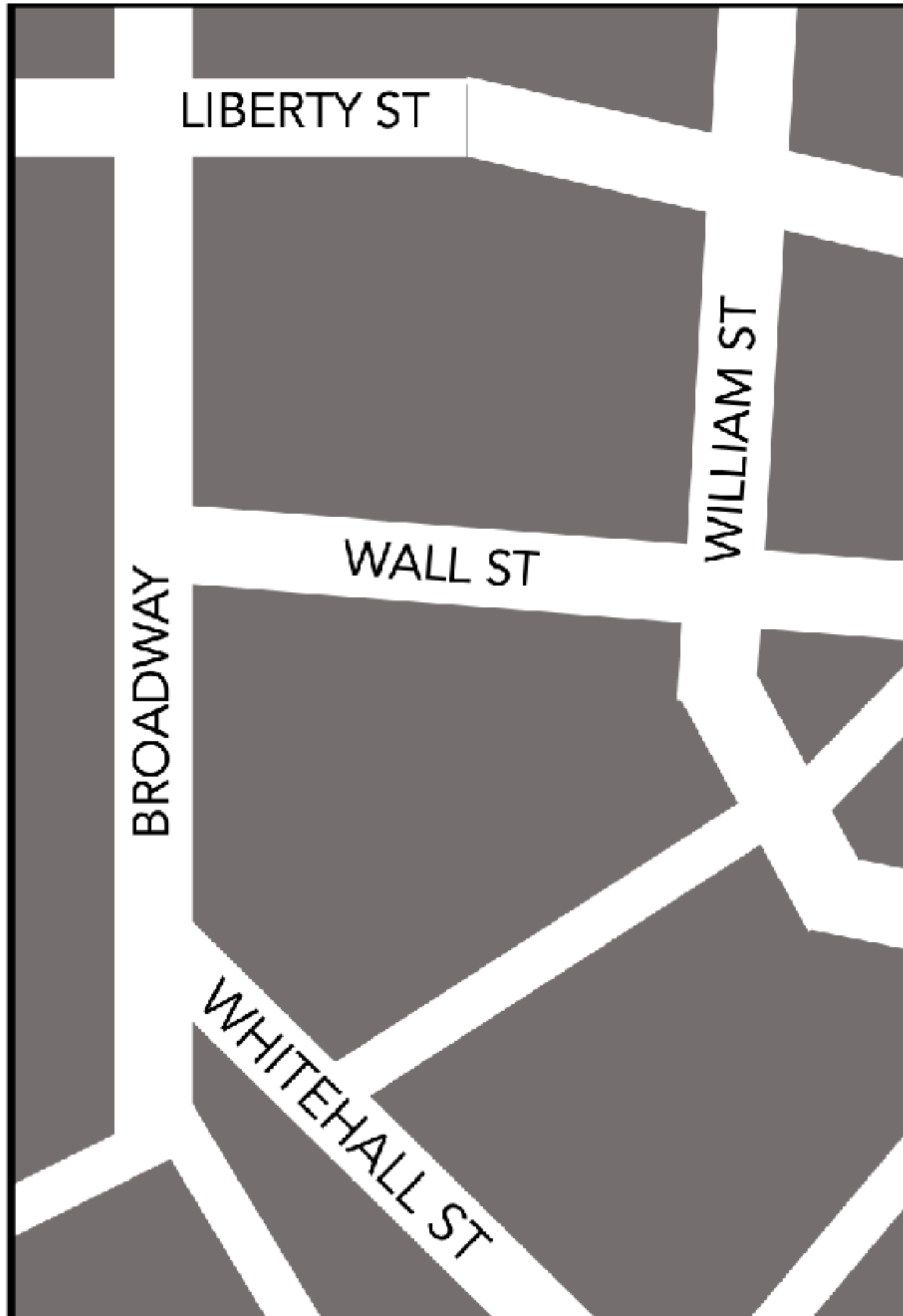
HasTraffic

Broadway

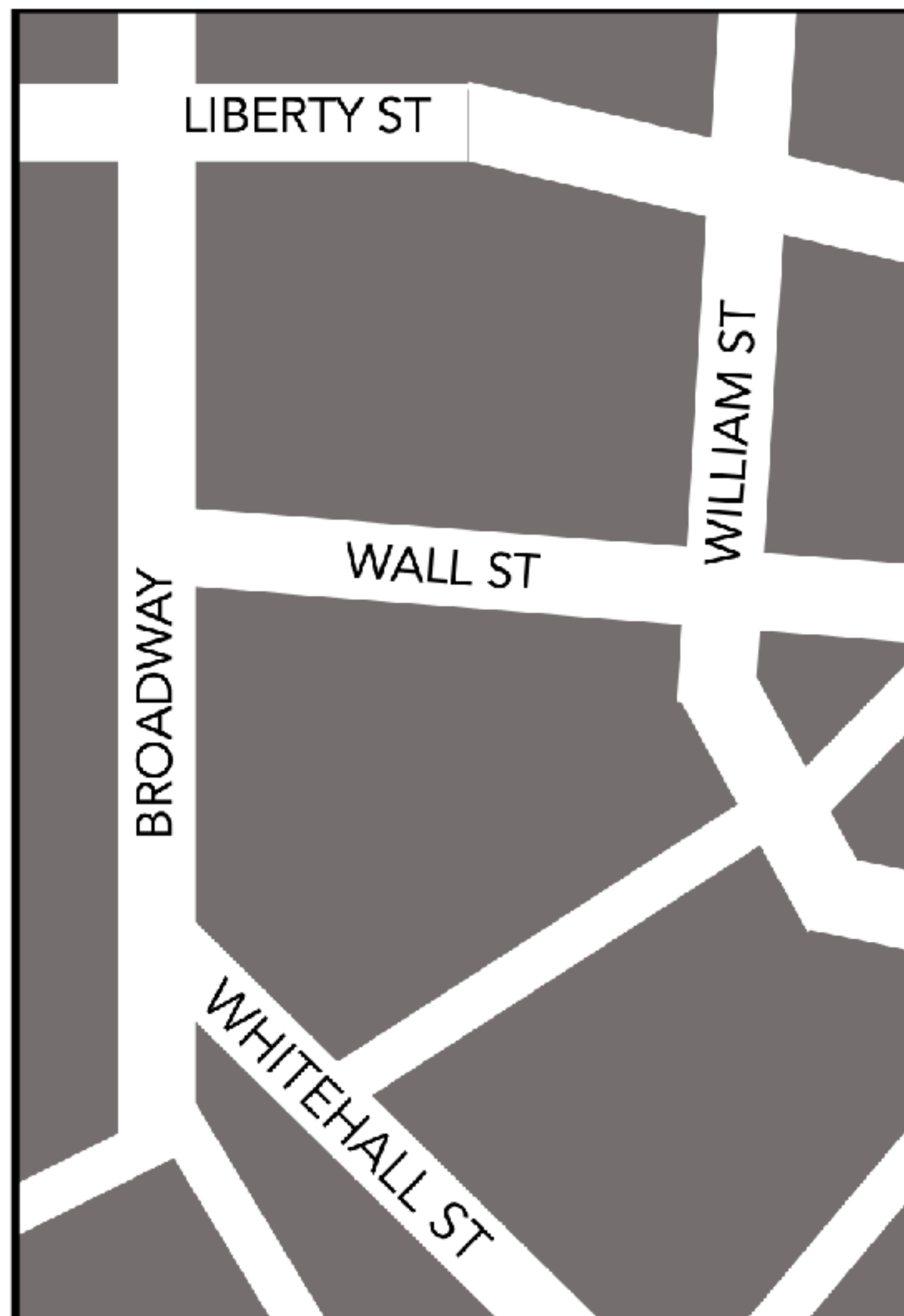
Wall St

William St

Whitehall St



An Example



GreenSignal

Broadway
Liberty St
William St
Whitehall St

HasTraffic

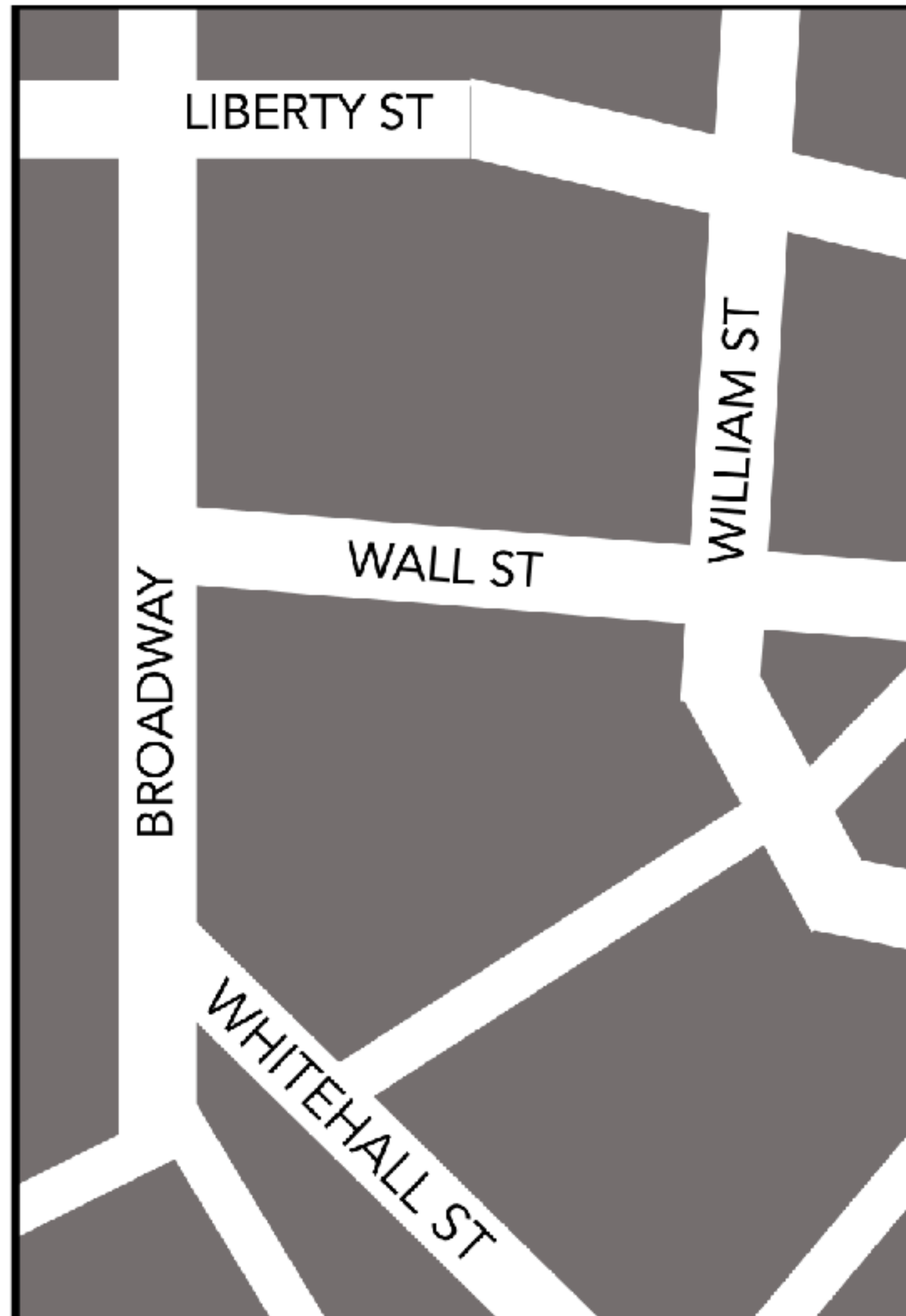
Broadway
Wall St
William St
Whitehall St



Crashes

Broadway
Whitehall St

An Example



GreenSignal

Broadway
Liberty St
William St
Whitehall St

HasTraffic

Broadway
Wall St
William St
Whitehall St



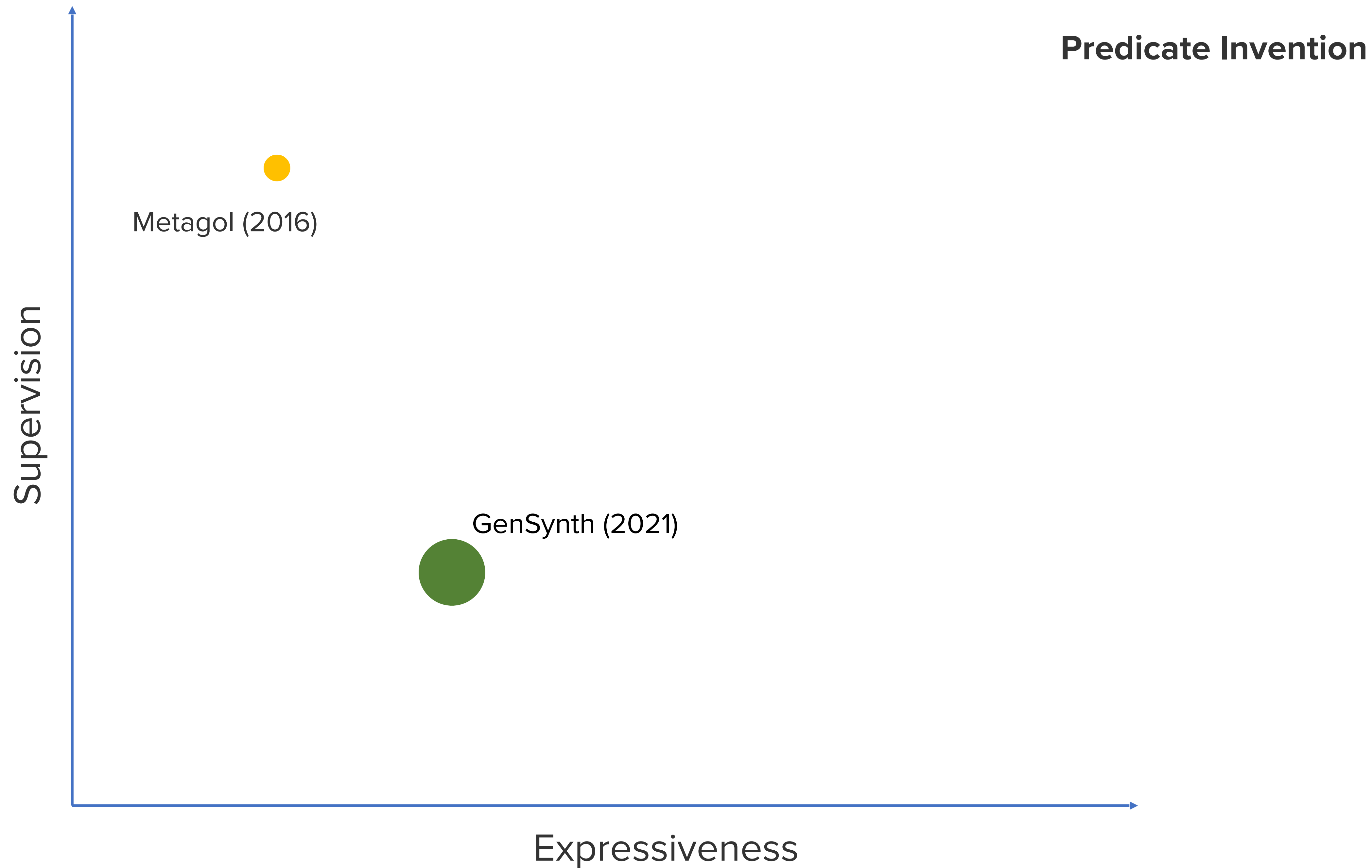
Crashes

Broadway
Whitehall St

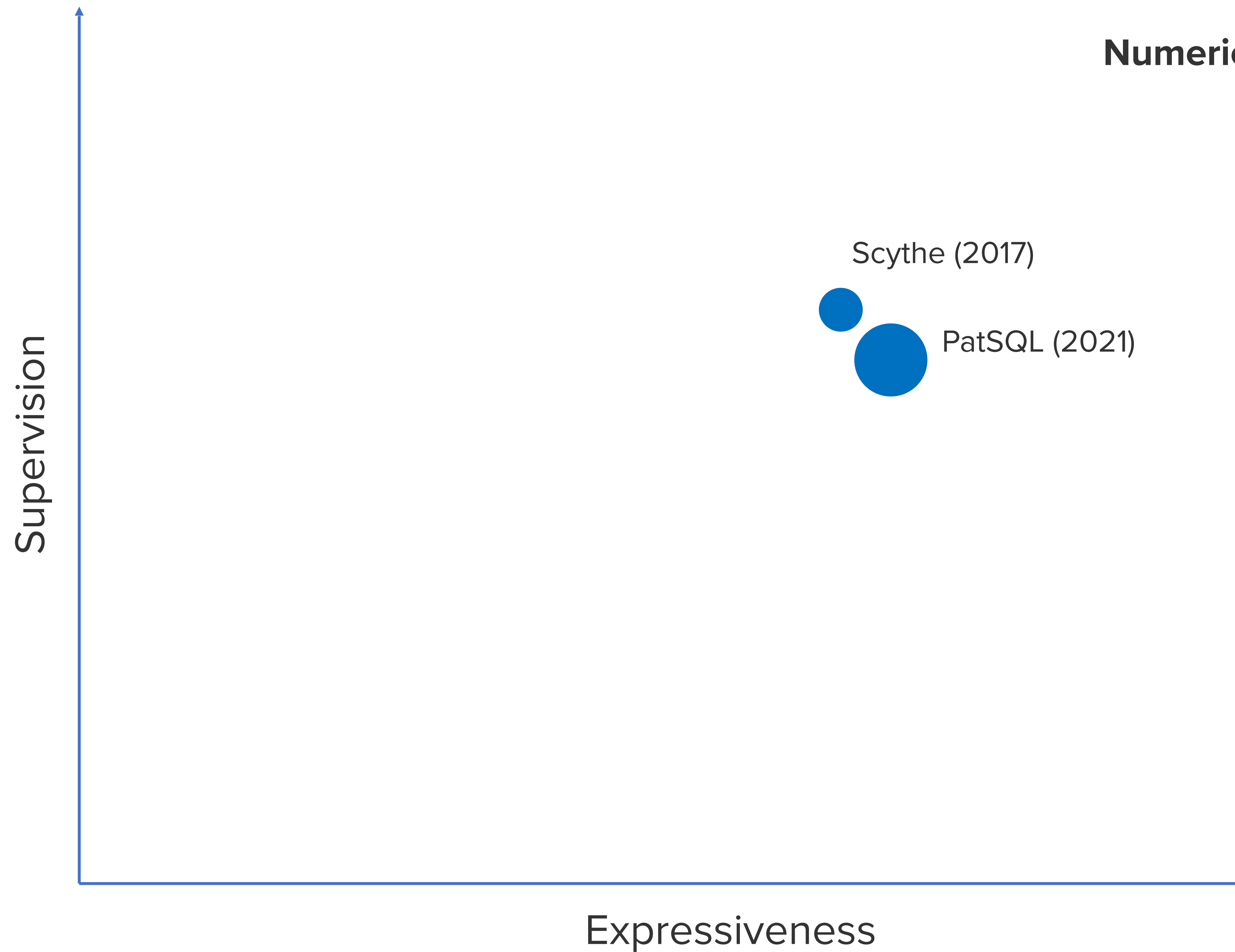
Crashes(x) : — HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).

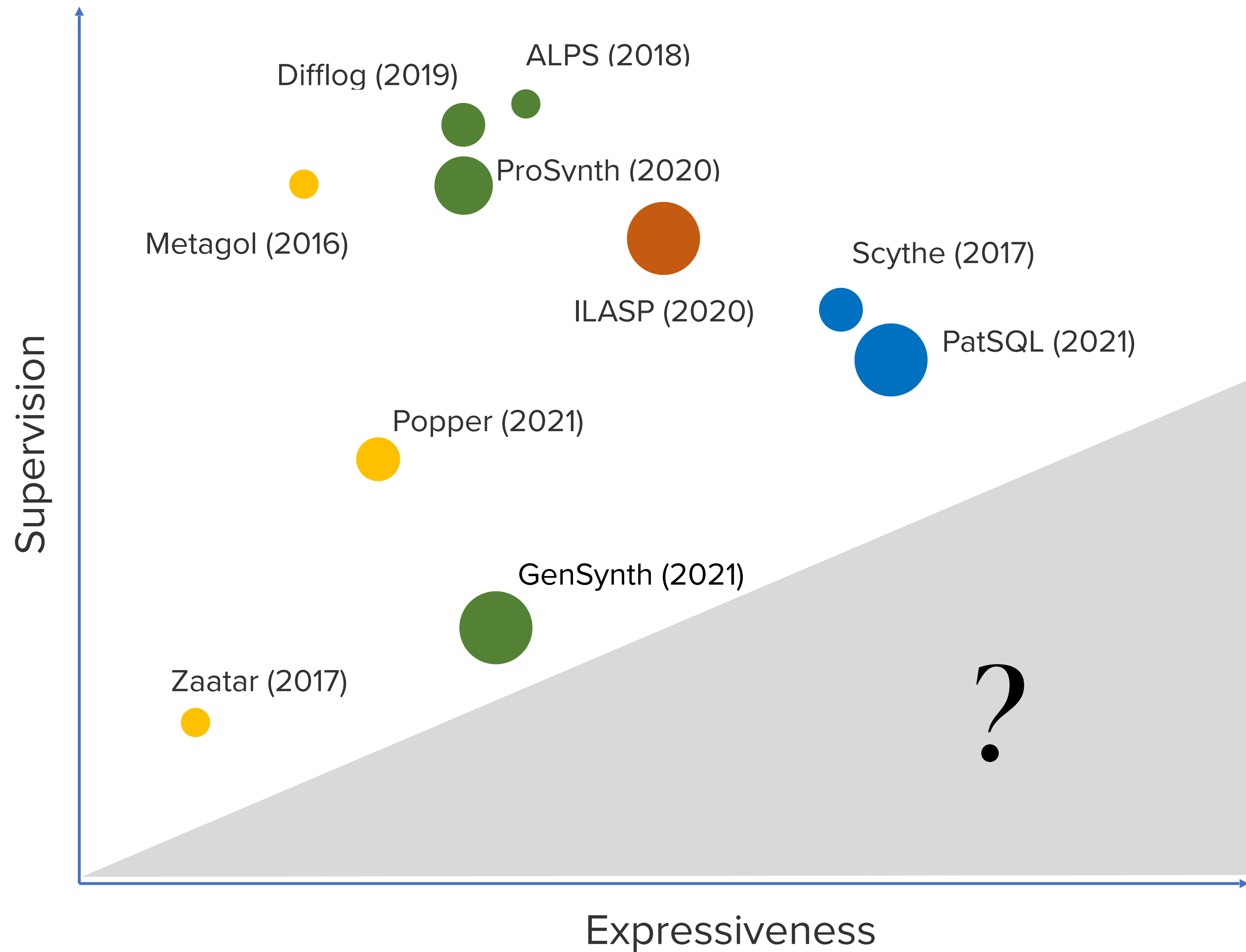


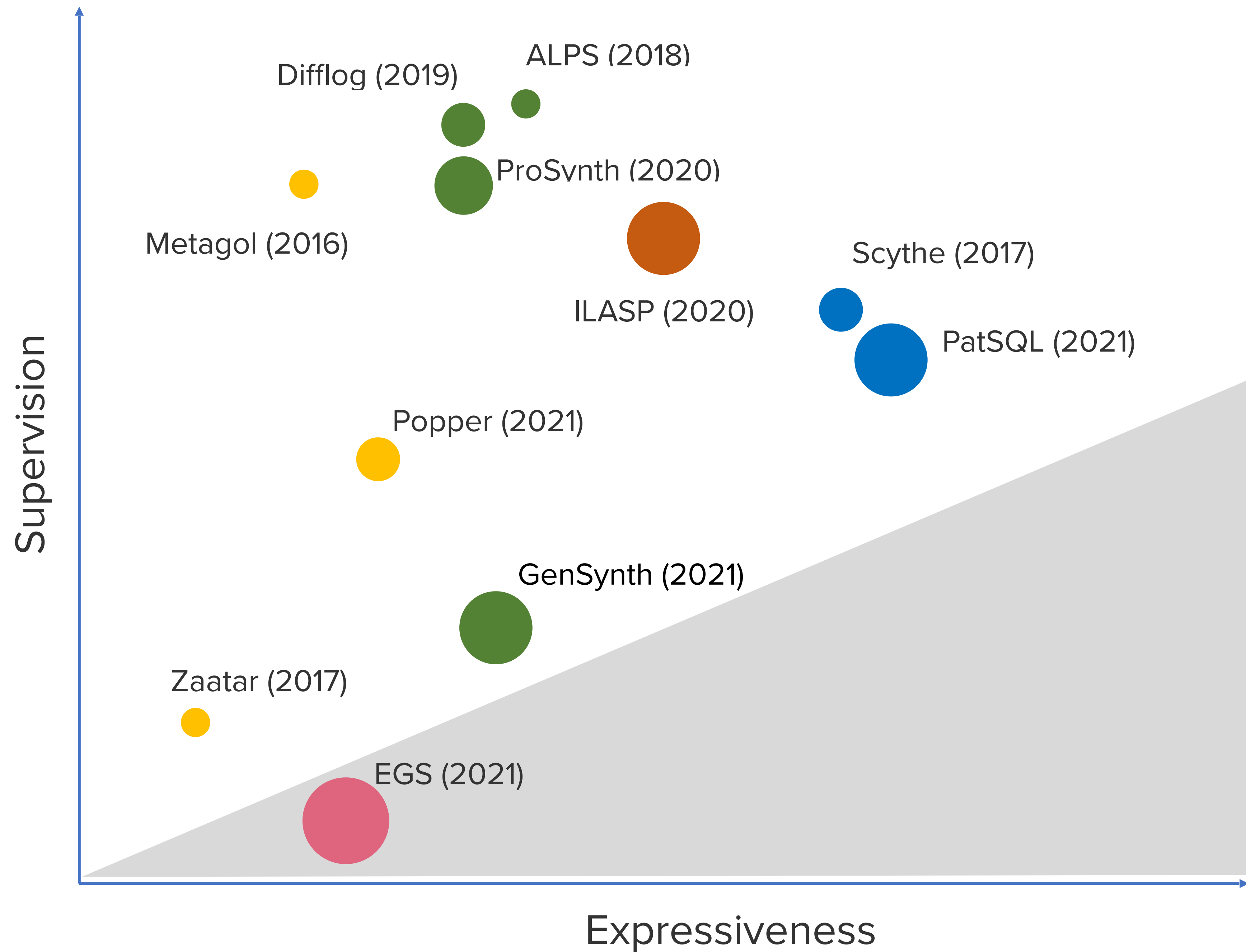




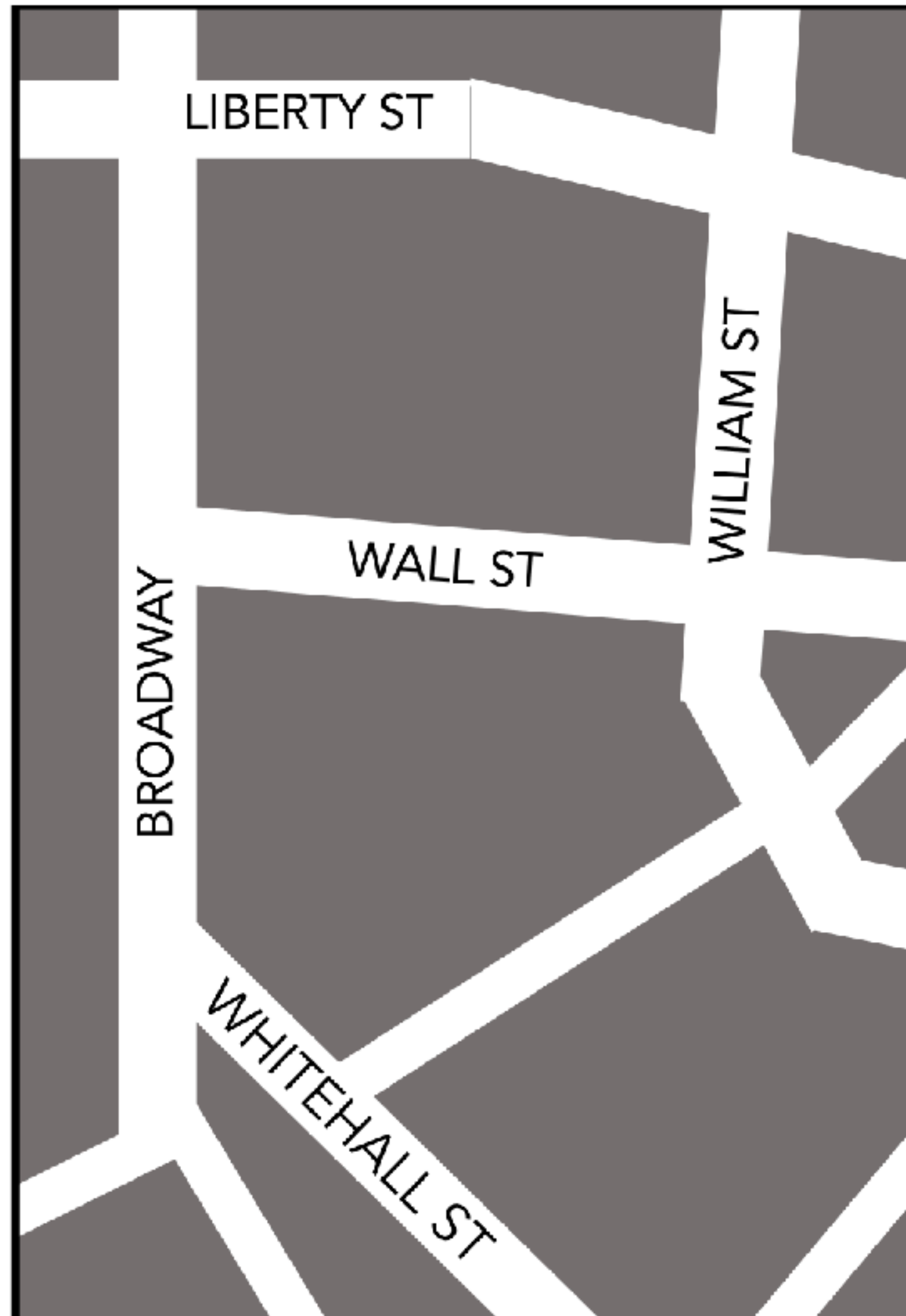
Numerical Comparison







Example-guided Synthesis



GreenSignal

Broadway
Liberty St
William St
Whitehall St

HasTraffic

Broadway
Wall St
William St
Whitehall St

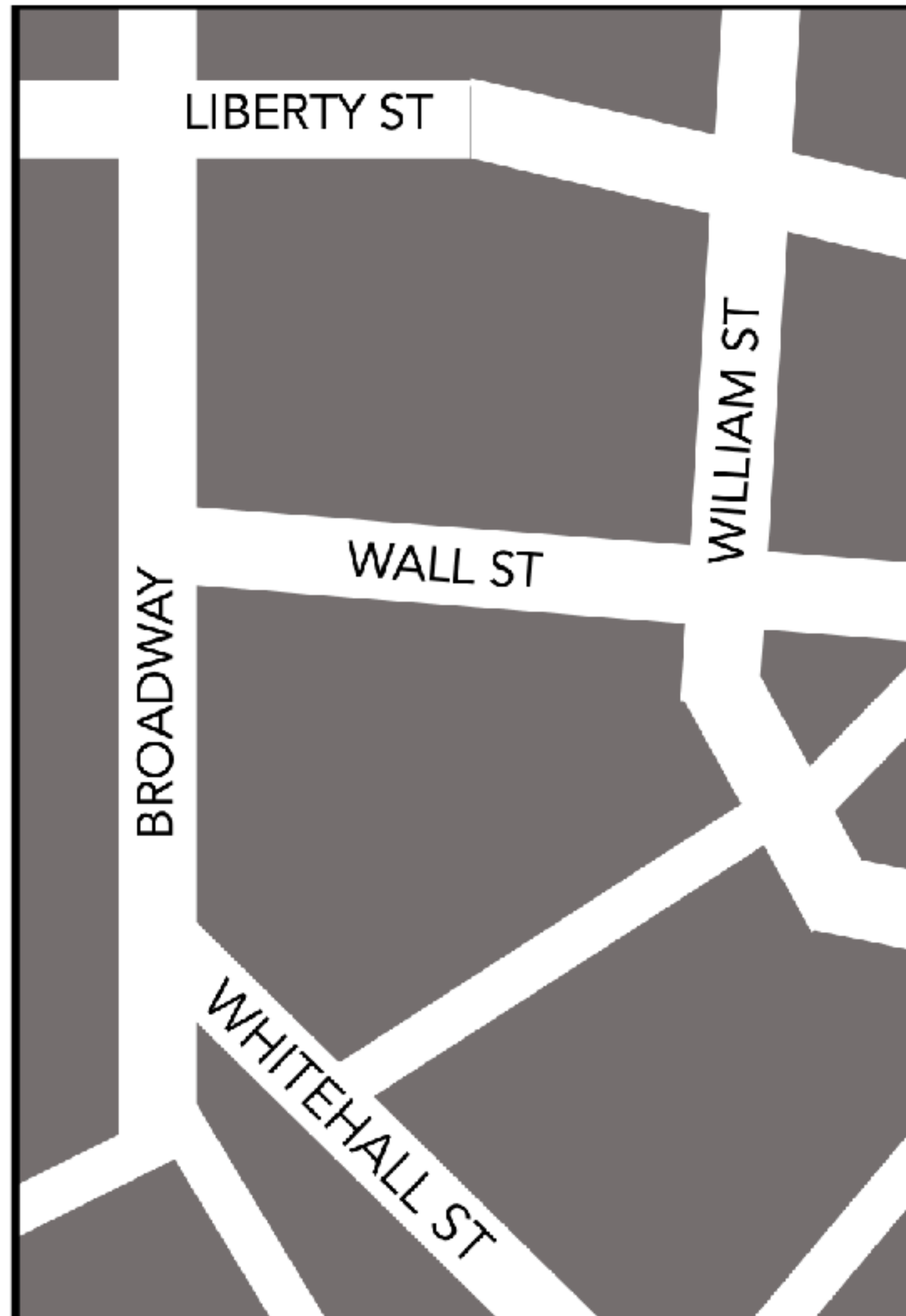


Crashes

Broadway
Whitehall St

Crashes(x) : — HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).

Example-guided Synthesis

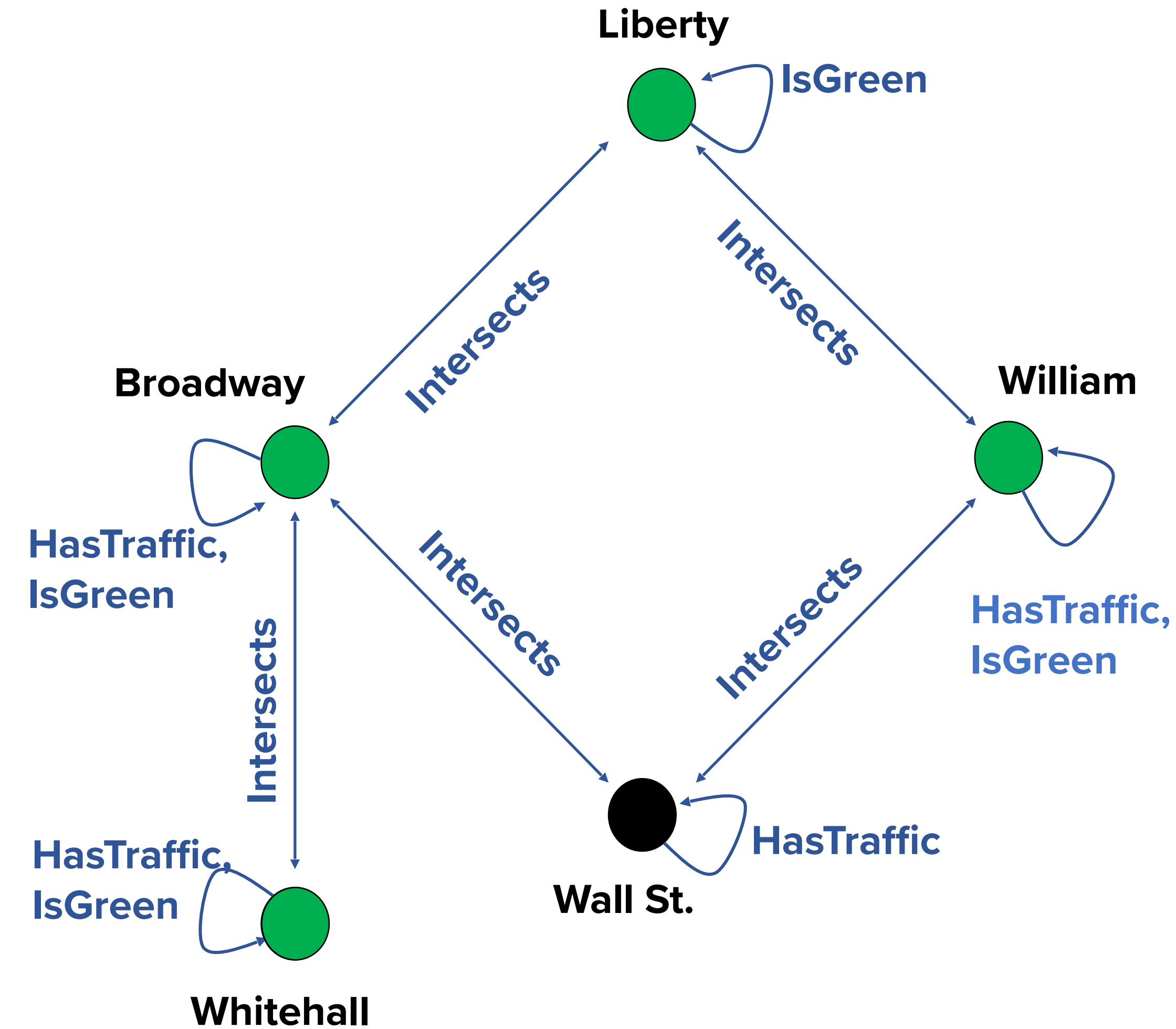


GreenSignal

Broadway
Liberty St
William St
Whitehall St

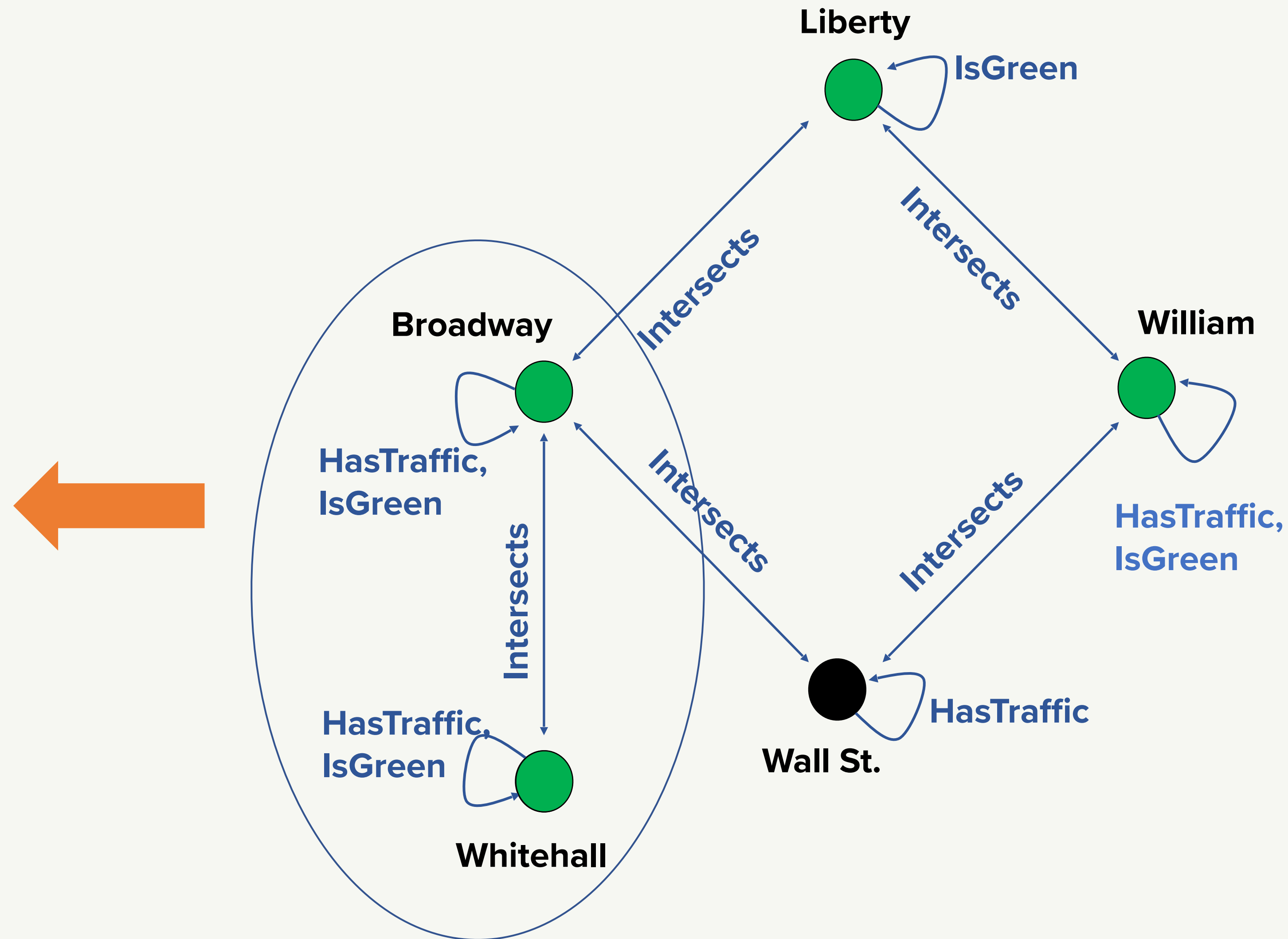
HasTraffic

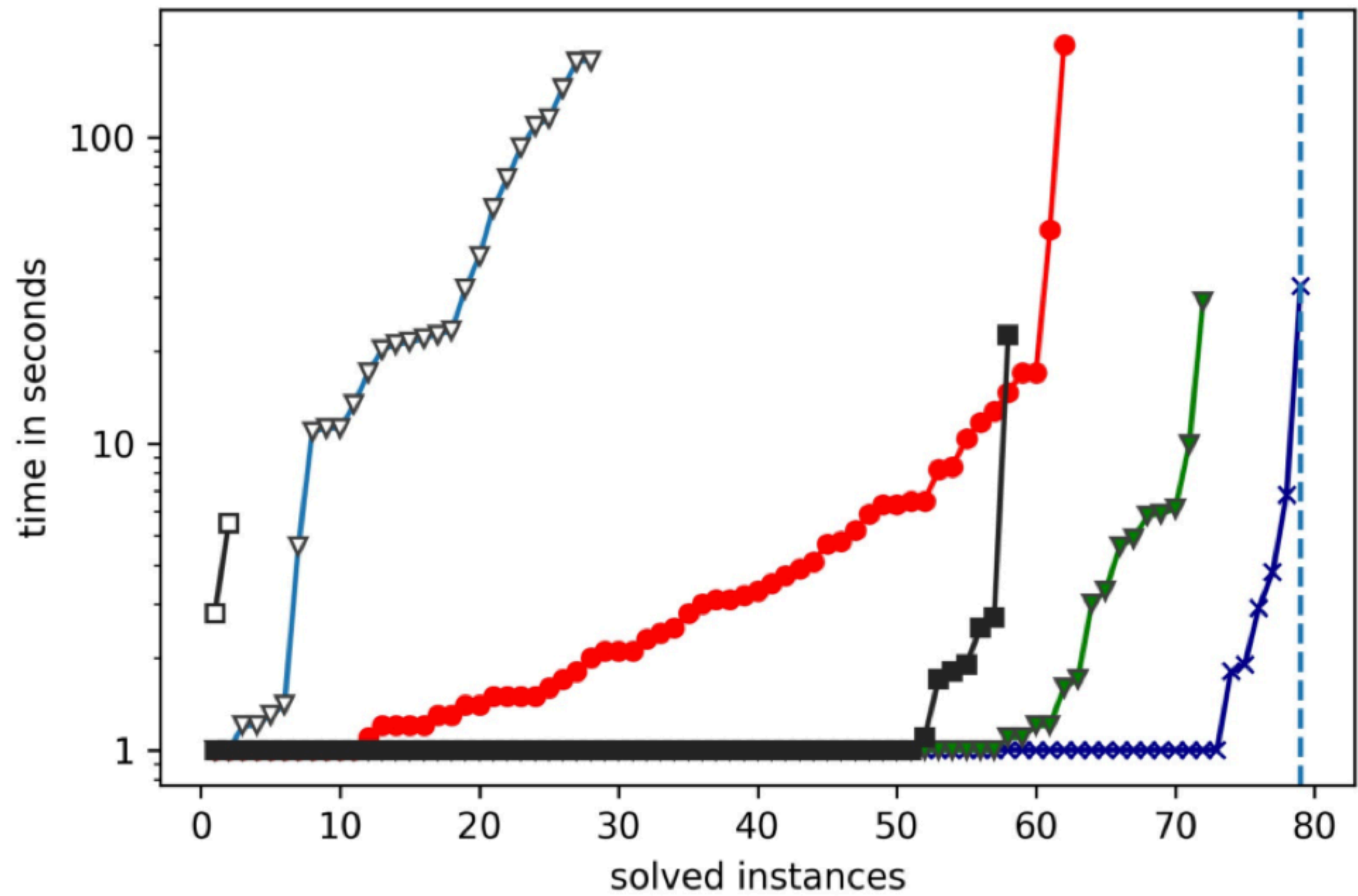
Broadway
Wall St
William St
Whitehall St

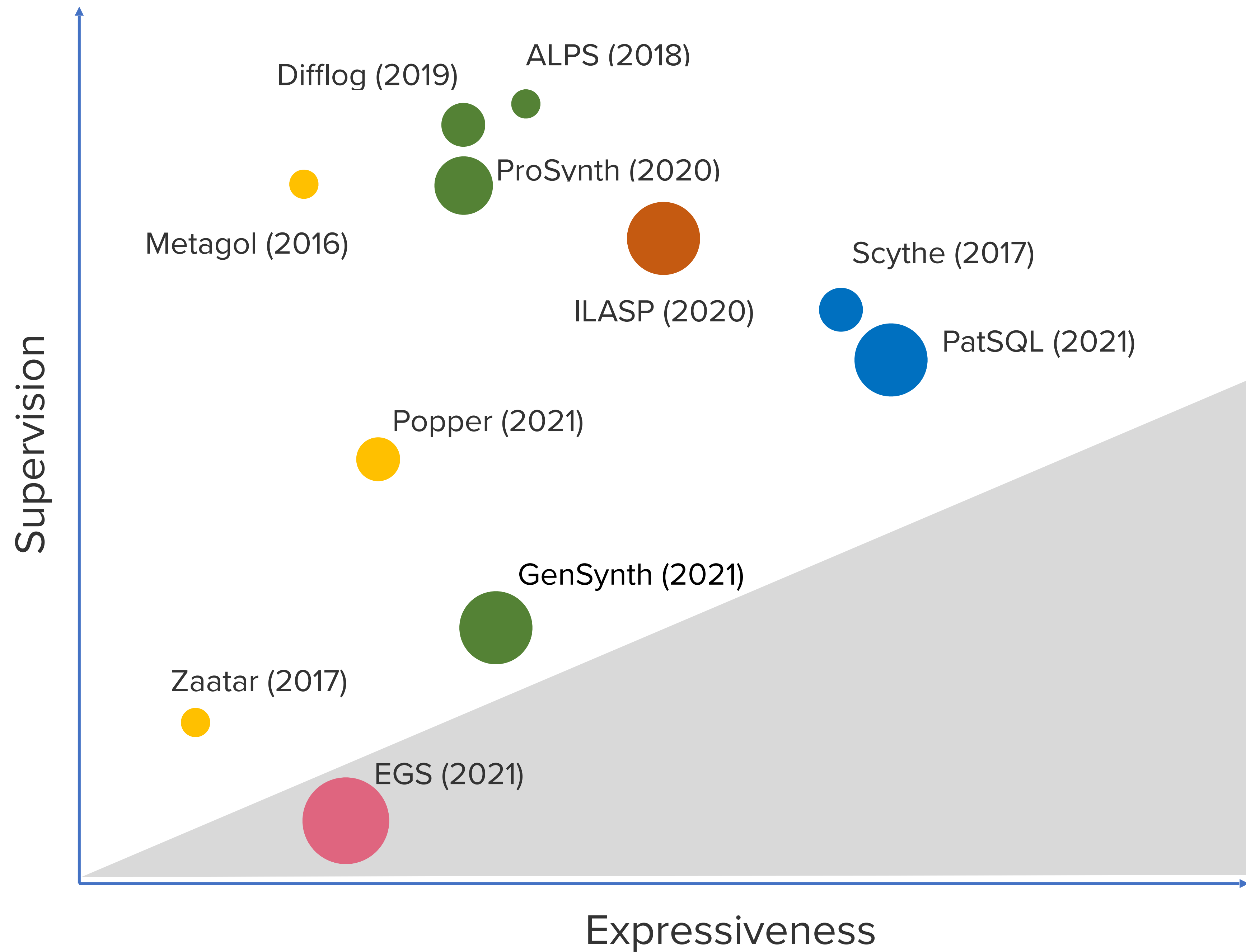


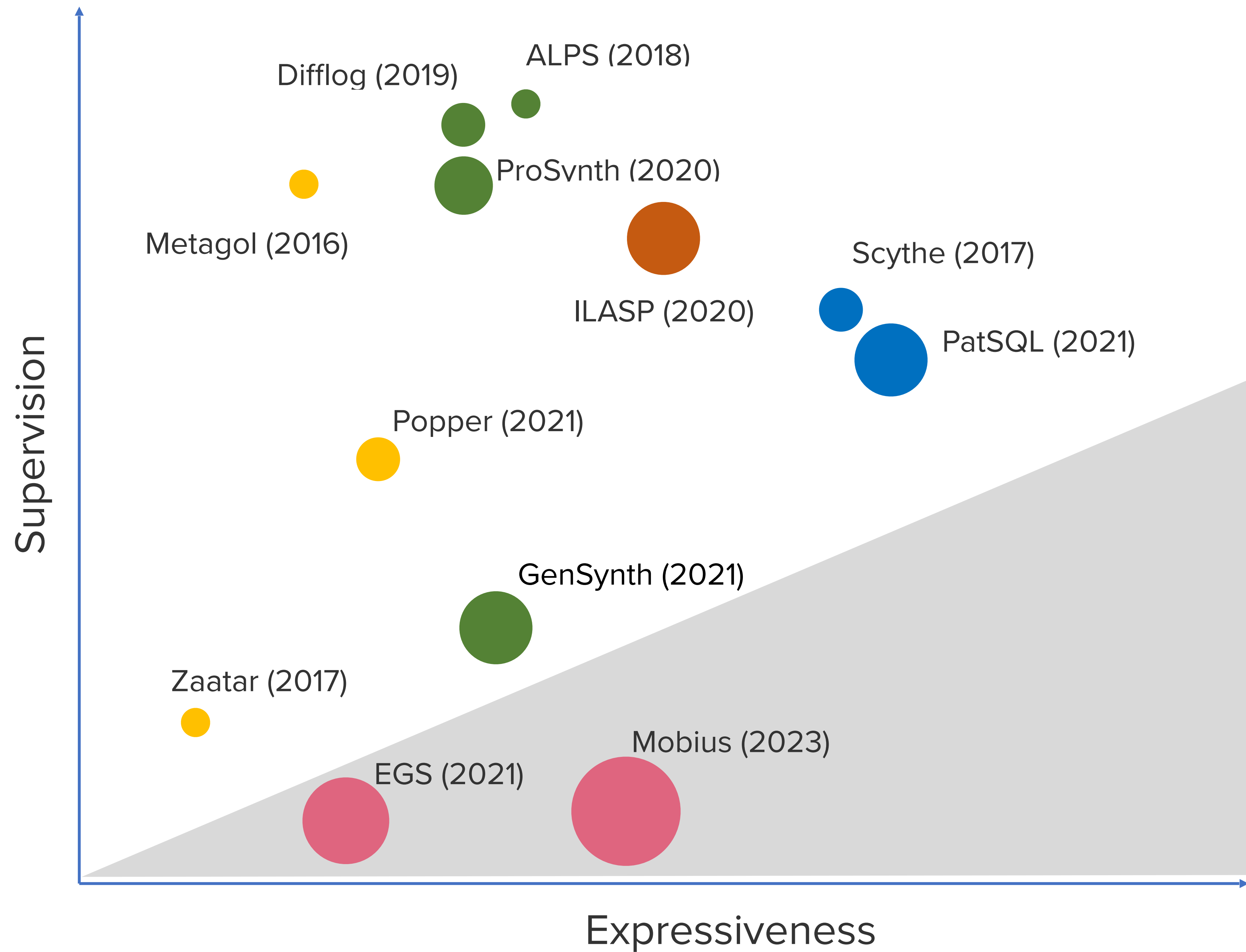
Example-guided Synthesis

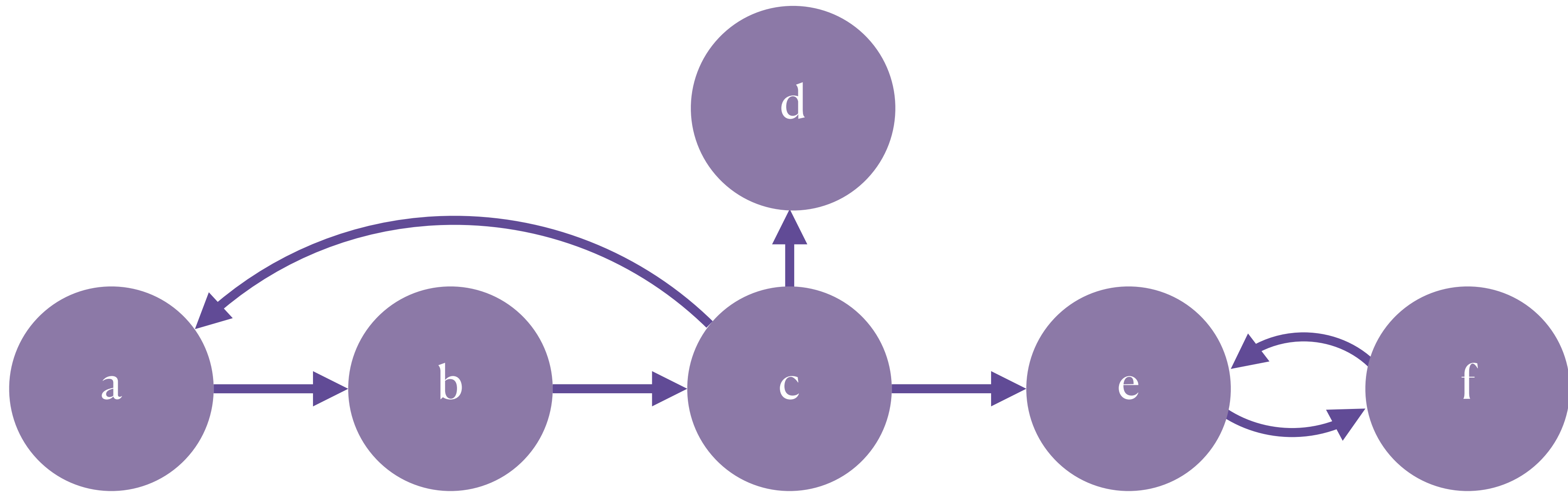
Crashes(x) : — HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).



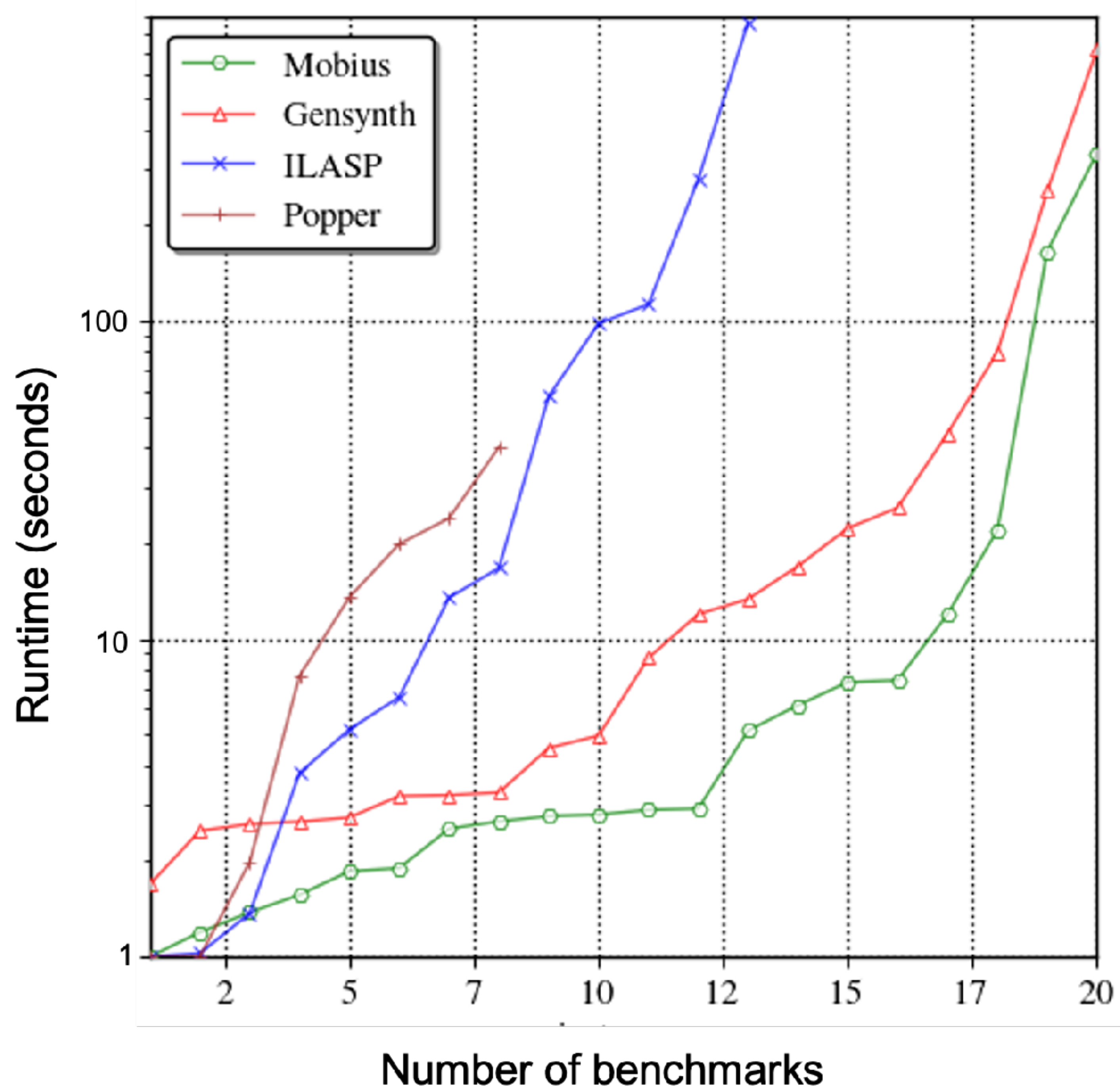


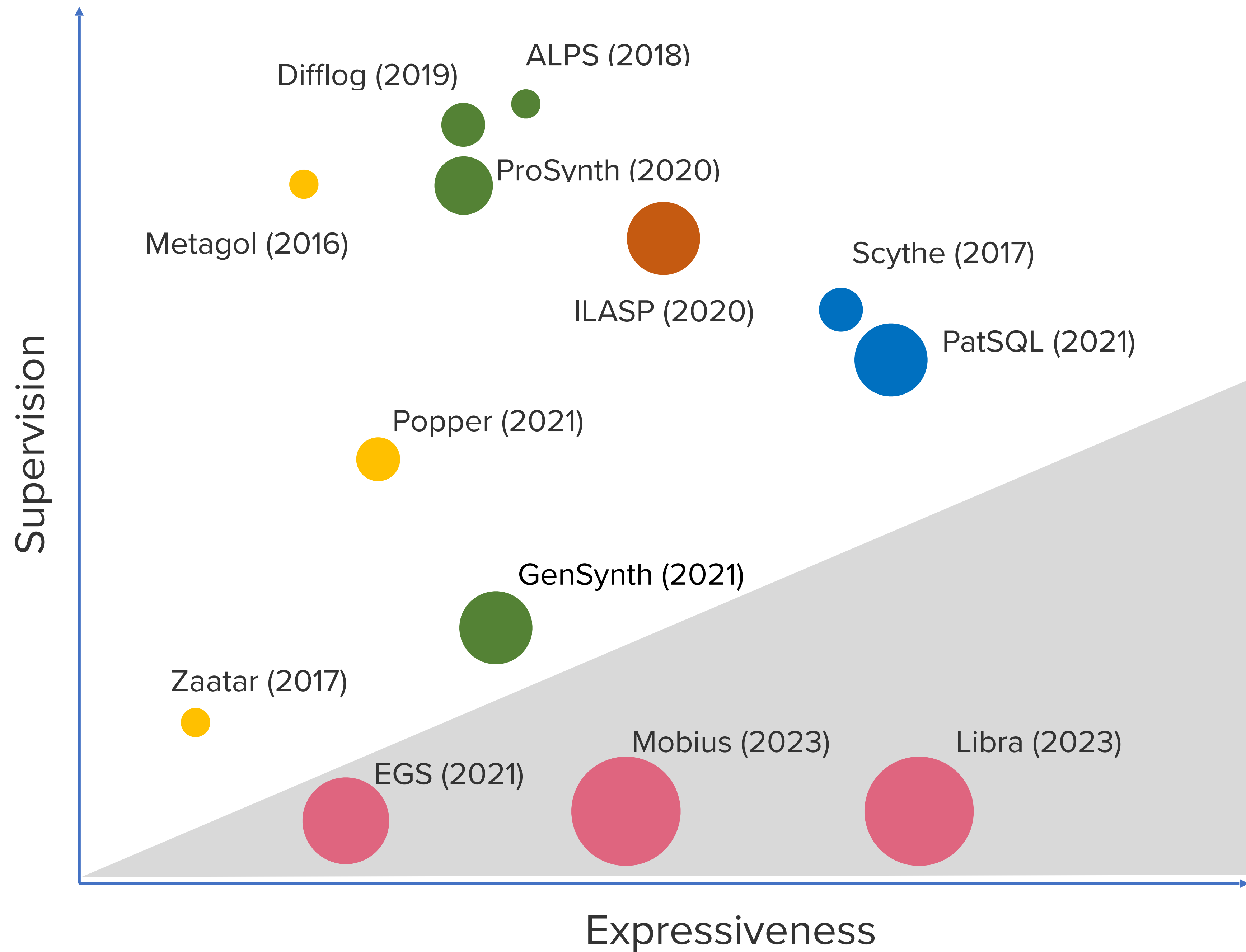






$\text{scc}(x, y) : - \text{path}(x, y), \text{path}(y, x).$
 $\text{path}(x, y) : - \text{edge}(x, y).$
 $\text{path}(x, y) : - \text{path}(x, z), \text{path}(z, y).$





```
SELECT registration.studentID
      FROM registration JOIN department
            ON registration.deptCode = department.deptCode
     WHERE registration.courseID < 500
            AND department.school = "Engineering"
```

```
SELECT registration.studentID
      FROM registration JOIN department
            ON registration.deptCode = department.deptCode
      WHERE registration.courseID < 500
            AND department.school = "Engineering"
```

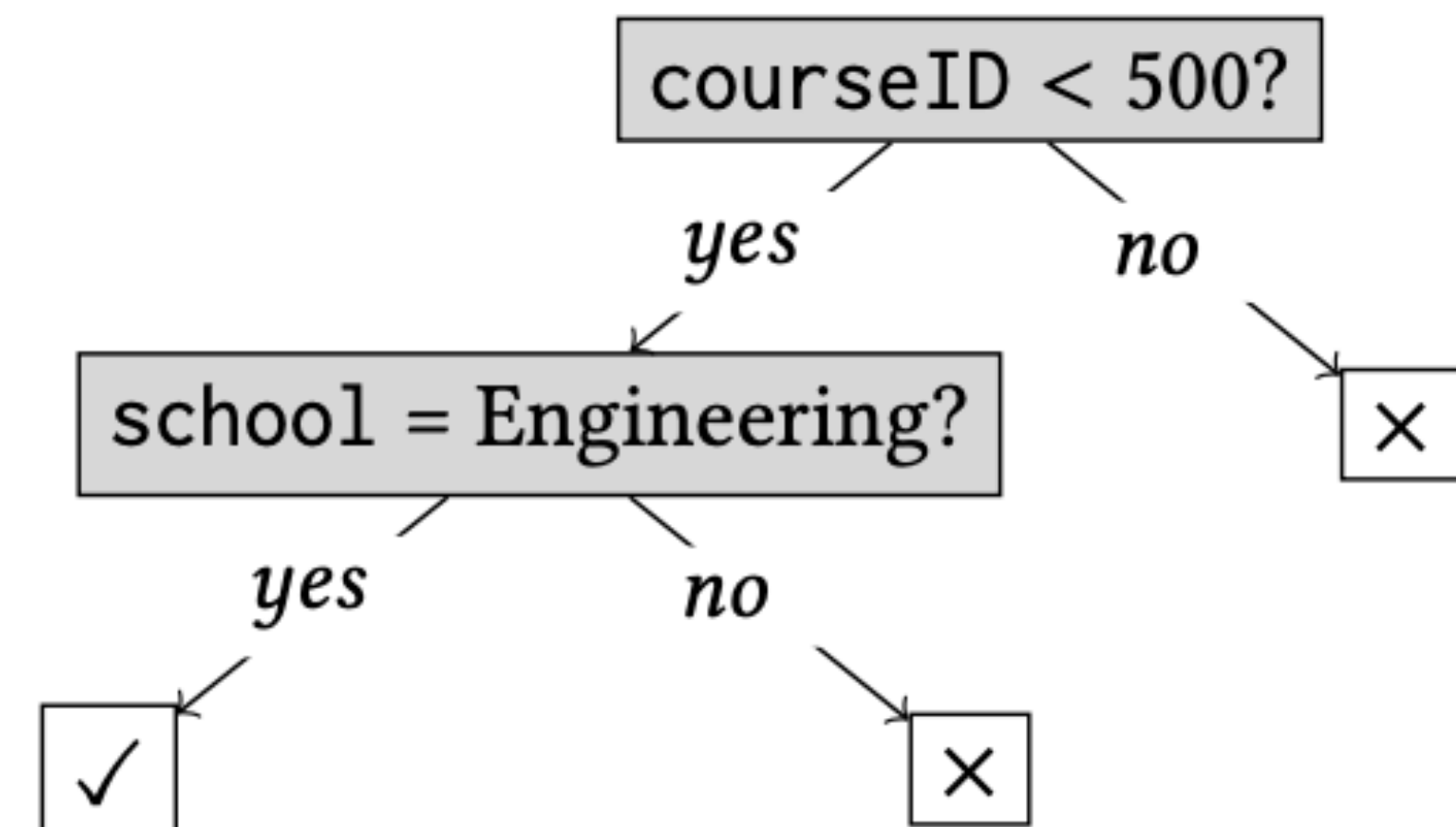


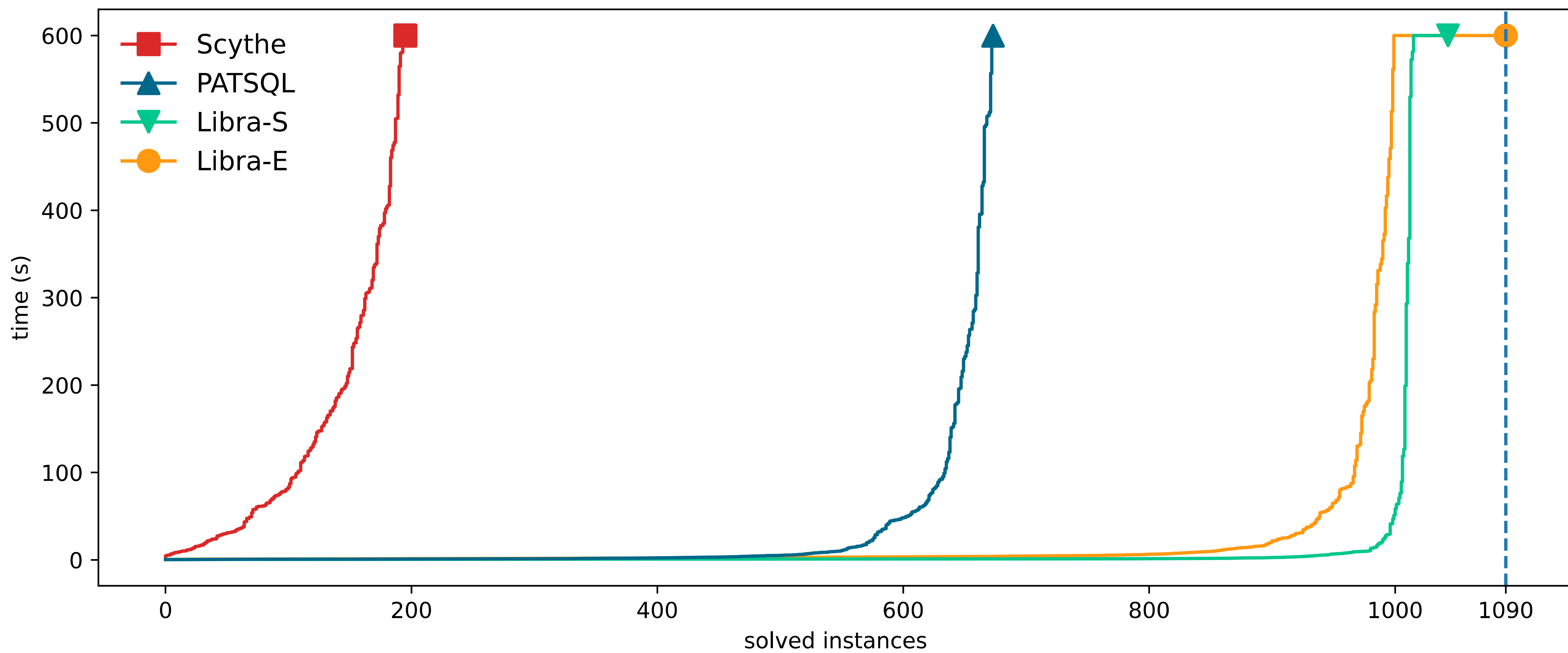
```

SELECT registration.studentID
FROM registration JOIN department
ON registration.deptCode = department.deptCode
WHERE registration.courseID < 500
AND department.school = "Engineering"

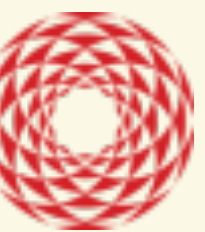
```

studentID	deptCode	courseID	school
Alice	Comp.	201	Engineering
Alice	Chem.	310	Arts and Science
Alice	Mech.	550	Engineering
Bob	Mech.	320	Engineering
Bob	Mech.	550	Engineering
Charlie	Chem.	310	Arts and Science
David	Comp.	500	Engineering
David	Mech.	502	Engineering
Erin	Chem.	310	Arts and Science







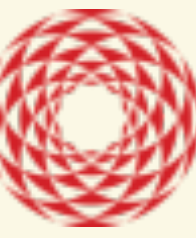


CENTRE FOR
Data Sciences and
Analytics

SAFEXPRESS CENTRE FOR
Data, Learning, and
Decision Sciences

CENTER FOR
Digitalisation, AI, and
Society

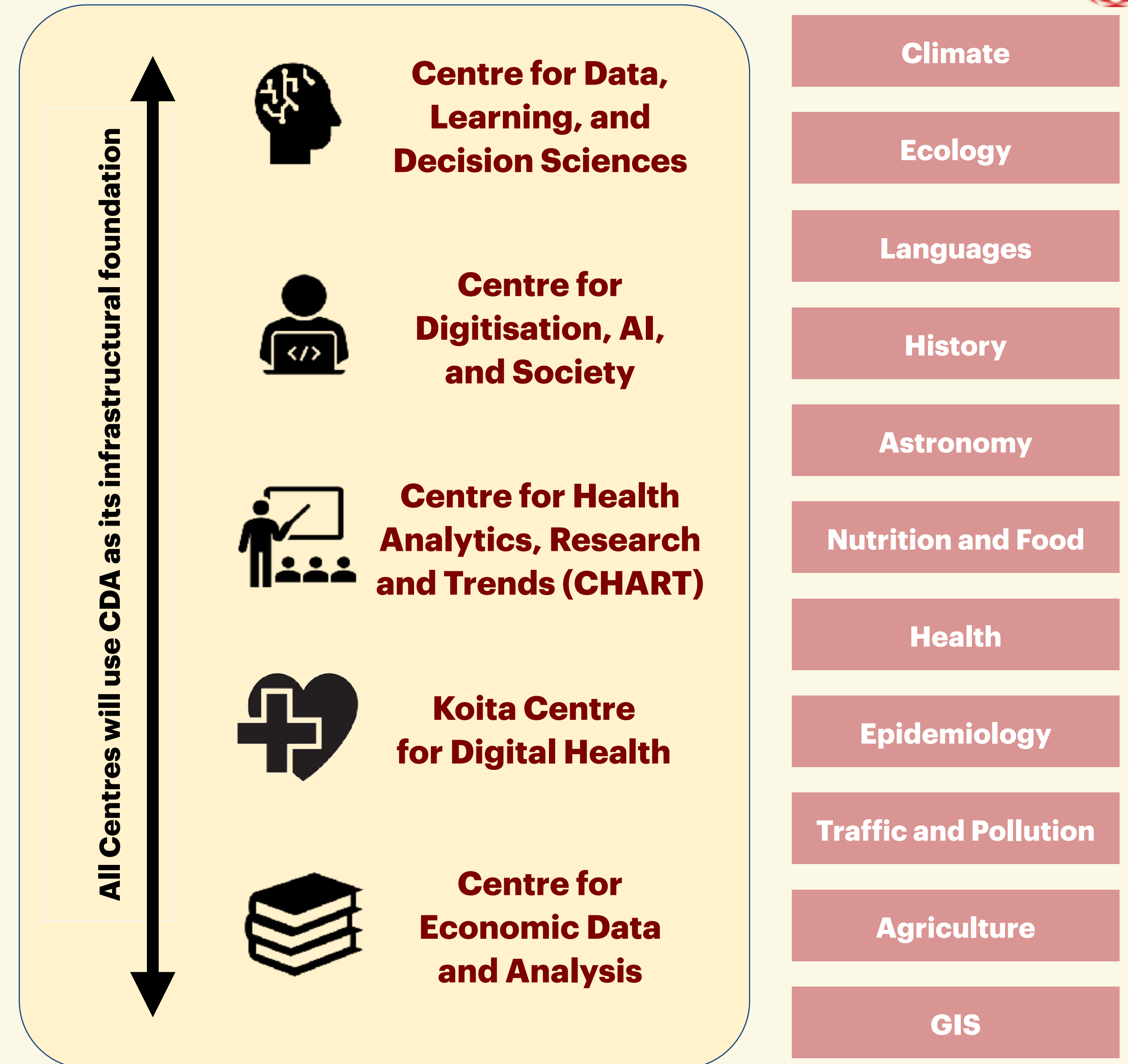
Koita Centre for
Digital Health



CENTRE FOR Data Sciences and Analytics

Comprehensive Data Lake Framework:

1. Repository of multimodal across interdisciplinary fields
2. Metadata of open source/public data
3. Unified access and integration
4. Inference, versioning, and provenance

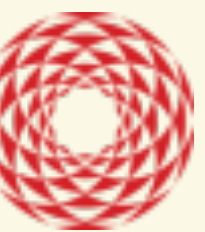




Building AI (with guarantees) as a tool

SAFEXPRESS CENTRE FOR Data, Learning, and Decision Sciences

- Data-driven quantitative modelling (weather, epidemiology, cultural behaviour)
- Financial Mathematics (risk, pricing, optimisation)
- Reinforcement Learning
- **Automated Reasoning**



AI as an agent, and its interaction with society

Brazilian Artificial Intelligence Strategy (EBIA)

Russia: National AI Strategy

IndiaAI Mission, Responsible AI (2021)

China: New Generation AI Development Plan

South Africa: National AI Plan

Voting Protocols and Their Properties

Privacy and Integrity of Electoral Rolls

Electronic Voting

Applications of Blockchains

Digitalisation in Healthcare

Cryptocurrency Regulation

Computational Techniques for Census

AI for Social Good

Robust, Fair, and Explainable AI

Ethics of Computing

CENTER FOR
Digitalisation, AI, and
Society



120 crore
biometric records



25 crore
linked health records



36 crore
daily transactions



Personal Health & Wellness

- Generation and use of personalised health data to identify risks, promote wellness, and reinforce healthy behaviour
- Genetic disease screening
- Use of wearables & healthcare apps

Precision Public Health

- Integrating multi-modal information for multi-scale precision health
- Population cohorts, convenience cohorts, biobanks
- Precision Medicine and Precision Public Health

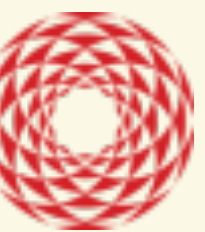
Intersections

- Assessing impact of food choices on health
- Promoting appropriate choices in foods
- Learning from history of medicine for digital health/ AI policy

AI + Health Data

- Developing a health data & analytics ecosystem for preventive and personalised medicine
- Ethical, purpose based, privacy preserving health data architectures that promote appropriate uses, while minimising risks to individuals
- Use of LLMs to empower citizens & public institutions with fit-for-purpose information

Koita Centre for
Digital Health



CENTRE FOR
Data Sciences and
Analytics

SAFEXPRESS CENTRE FOR
Data, Learning, and
Decision Sciences

CENTER FOR
Digitalisation, AI, and
Society

Koita Centre for
Digital Health

Aalok Thakkar

aalok.thakkar@ashoka.edu.in

aalok-thakkar.github.io