Analysis/Bug-finding/Verification for Security

VIJAY GANESH
University of Waterloo
Winter 2013

Analysis/Test/Verify for Security

- Instrument code for testing
  - Heap memory: Purify
  - Perl tainting (information flow)
  - Java race condition checking

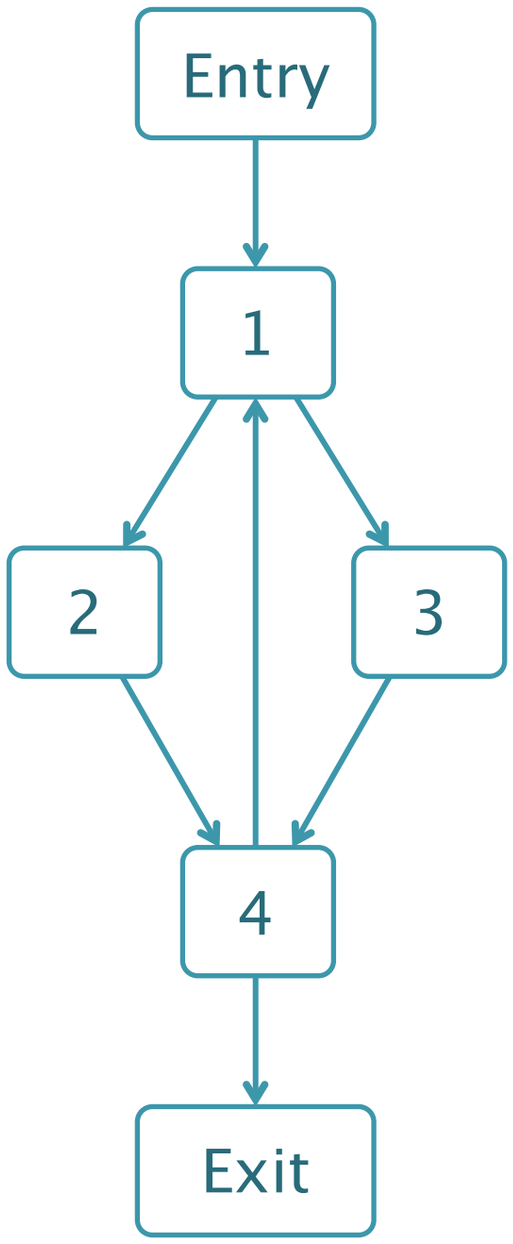
- Black-box testing
  - Fuzzing and penetration testing
  - Black-box web application security analysis

- Static code analysis
  - FindBugs, Fortify, Coverity, MS tools, …

- Model Checking tools
  - NuSMV to verify program properties

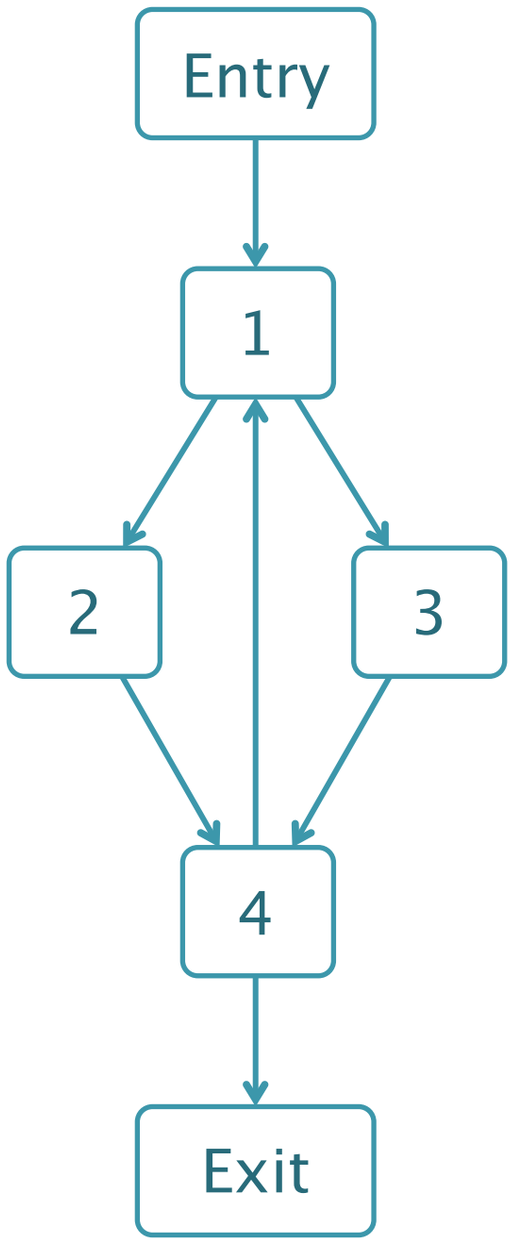
Slide courtesy John Mitchell

Manual Testing Doesn't Scale



Software

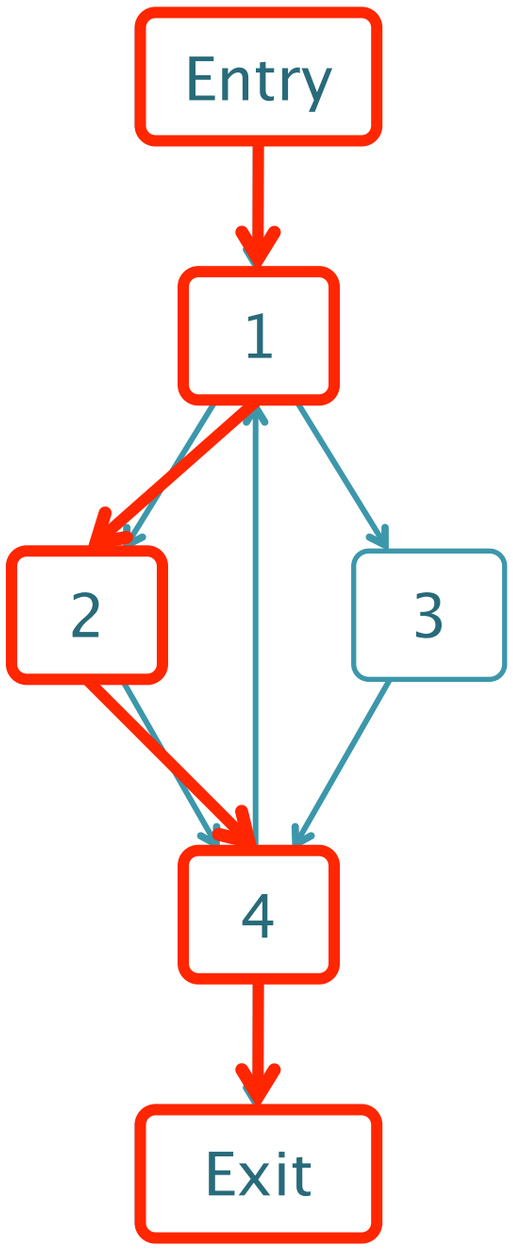
Manual Testing Doesn't Scale



Software

Behaviors

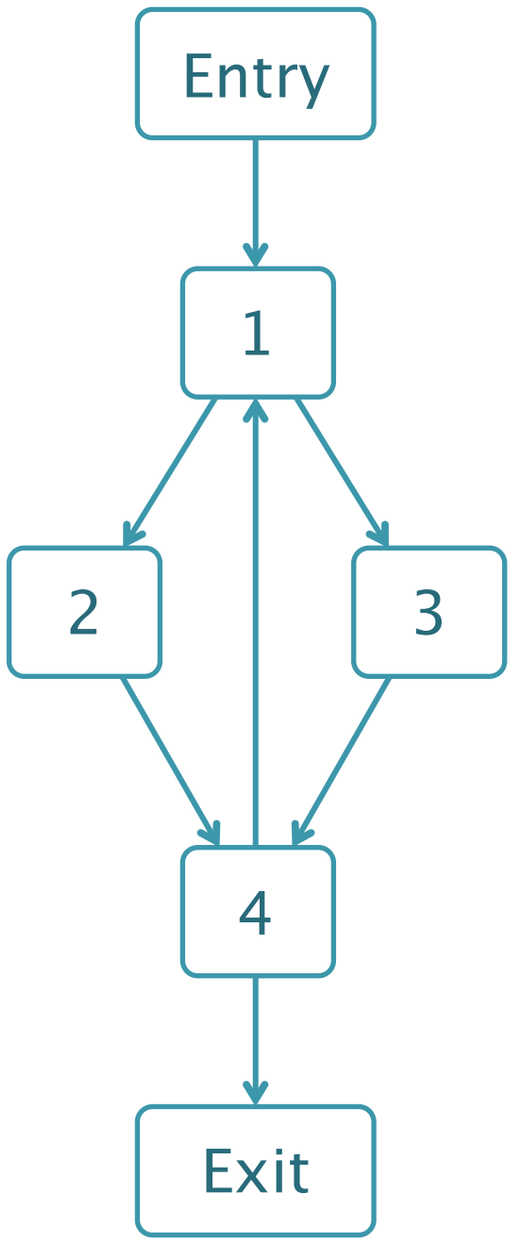
Manual Testing Doesn't Scale



Software

Behaviors

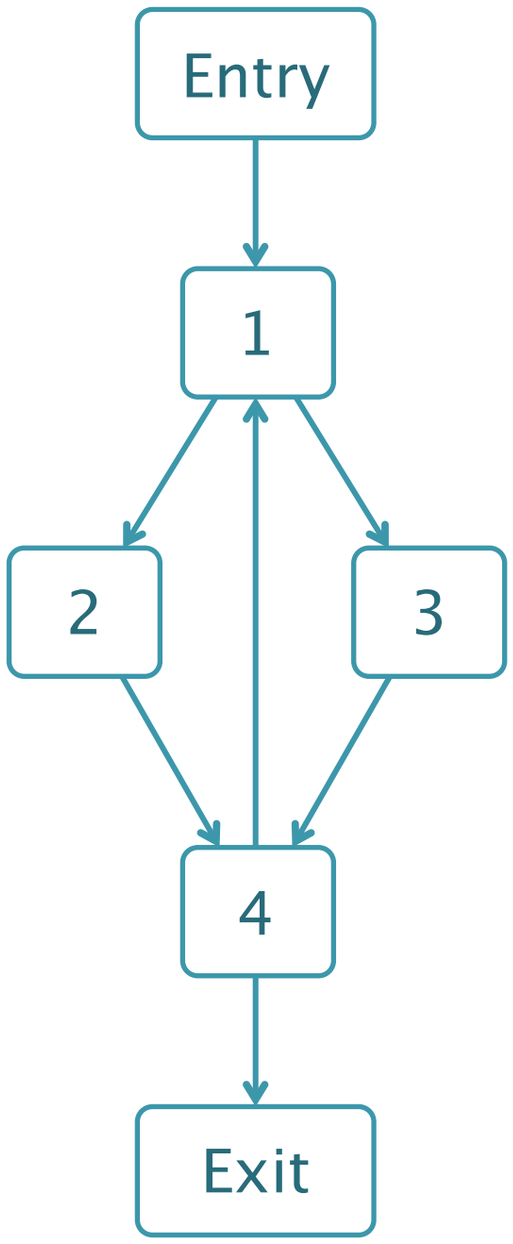
Manual Testing Doesn't Scale



Software

Behaviors

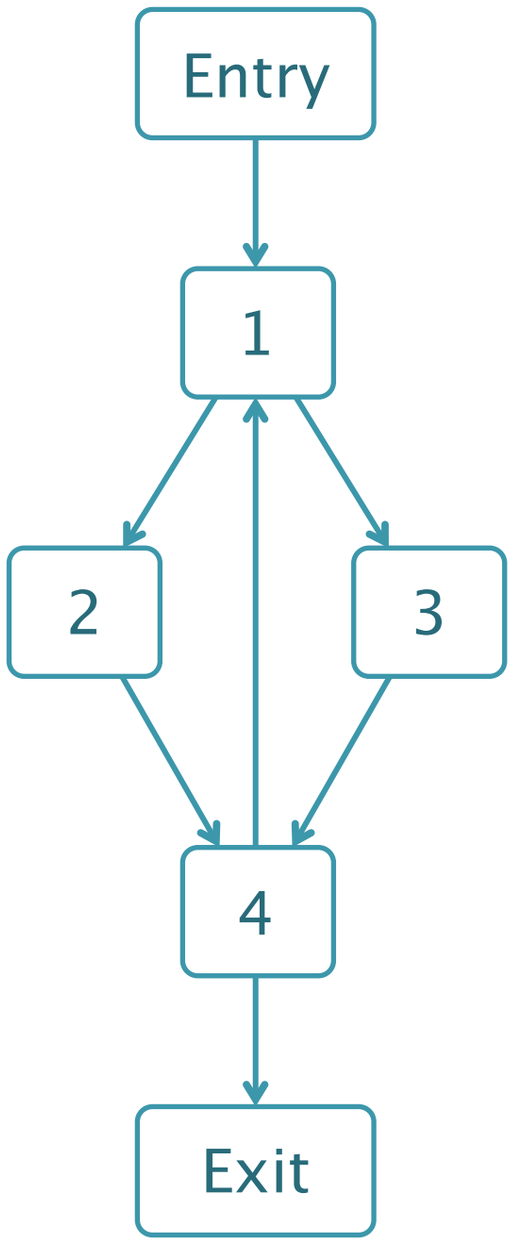
Manual Testing Doesn't Scale



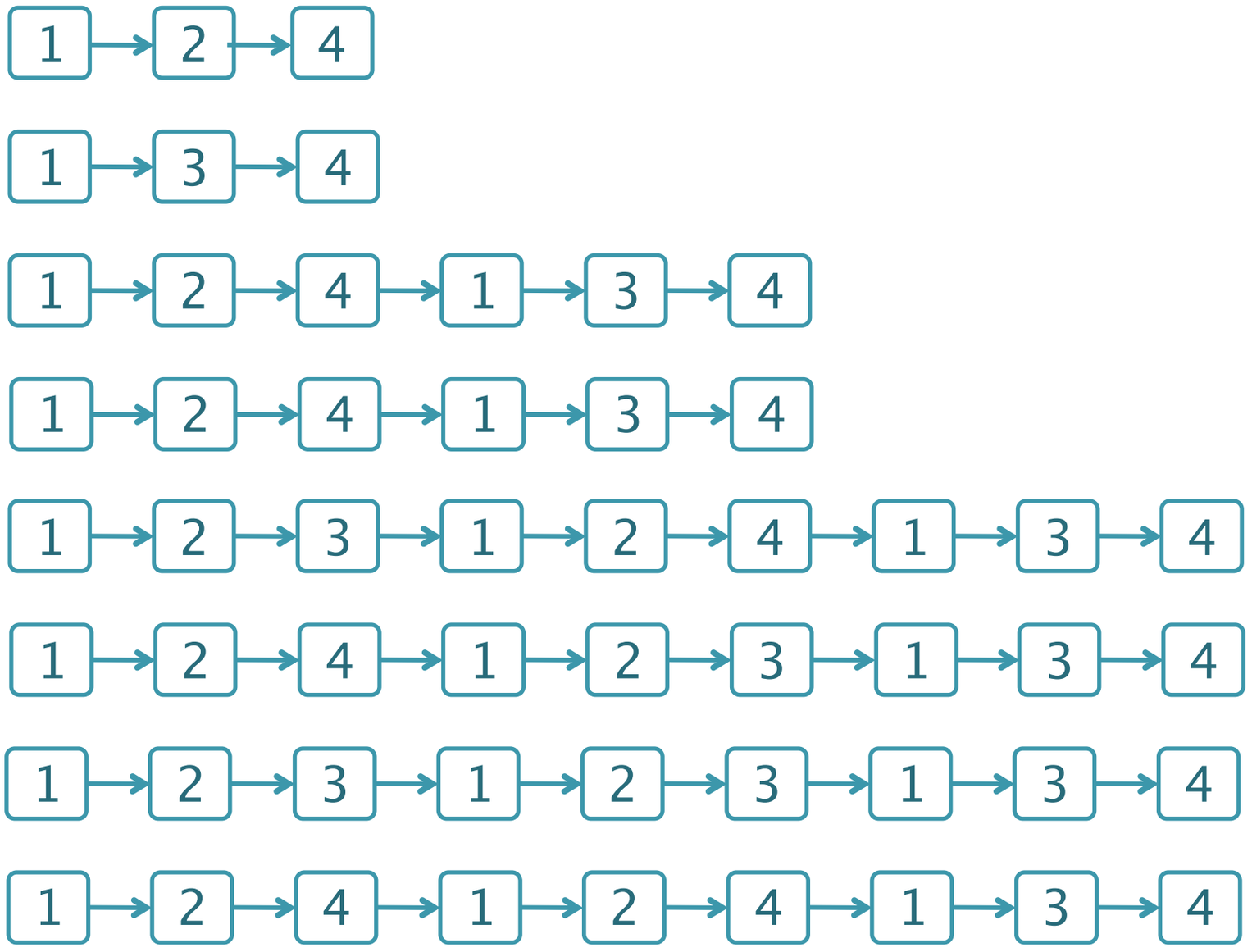
Software

Behaviors

Manual Testing Doesn't Scale

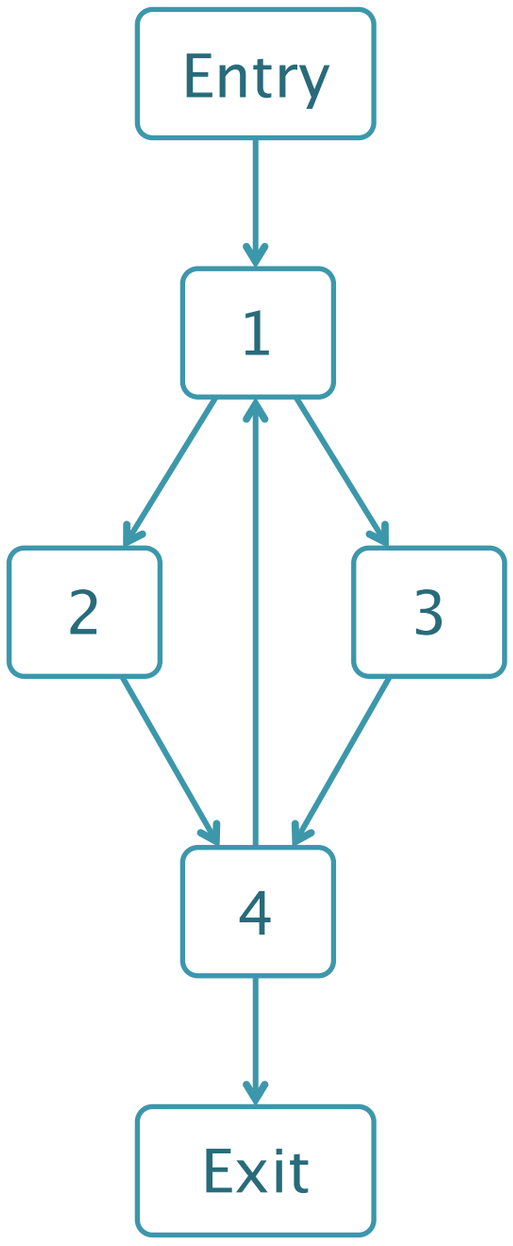


Software

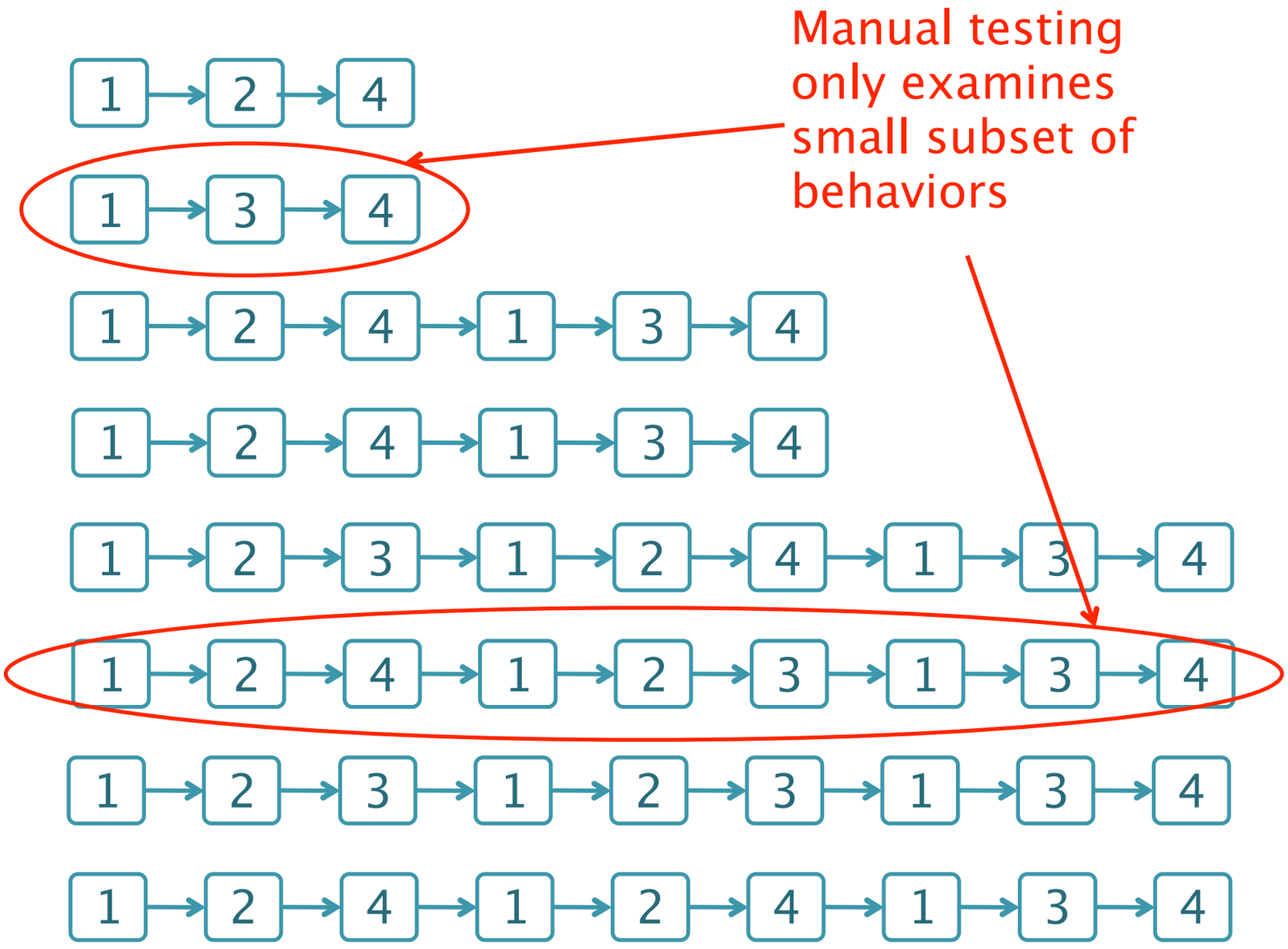


Behaviors

Manual Testing Doesn't Scale



Software



Behaviors

Bug in Swfdec Adobe Flash Movie Player

```
clipconv8x8_u8_s16_c(ptr...){
    ...
     for (i = 0; i < 8; i++) {
       for (j = 0; j < 8; j++) {

          x = BLOCK8x8_S16 (src,sstr,i,j);
          if (x < 0) x = 0;
          if (x > 255) x = 255;
          (*((uint8_t *)((void *) ptr + stride*row) + column)) = x;
       }
     }
}
```

ptr comes from caller

ptr references unallocated memory

Code From LibOIL Library version 0.3.x (GNOME Windowing System)

Bug in Swfdec Adobe Flash Movie Player

```
jpeg_decoder(JpegDecoder* dec) {
  dec->width_blocks = (dec->width + 8*max_h_sample - 1)/(8*max_h_sample);
  dec->height_blocks = (dec->height + 8*max_v_sample - 1)/(8*max_v_sample);
  int rowstride, image_size;
  ...
  rowstride  =   dec->width_blocks * 8*max_h_sample / dec->comps[i].h_subsample;
  imagesize =  rowstride * (dec->height_blocks * 8*max_v_sample/dec->comps[i].v_subsample);
```

malloc OK
But
imagesize 0

```
dec->c[i].image=malloc(imagesize);
...

//LibOIL API function call
clipconv8x8_u8_s16_c(dec->c[i].image...);
...
```

Code from Swfdec Shockwave Flash Movie Player

Bug in Swfdec Adobe Flash Movie Player

from input movie

```
jpeg_decoder(JpegDecoder* dec) {
  dec->width_blocks = (dec->width + 8*max_h_sample - 1)/(8*max_h_sample);
  dec->height_blocks = (dec->height + 8*max_v_sample - 1)/(8*max_v_sample);
  int rowstride, image_size;
  ...
  rowstride  =   dec->width_blocks * 8*max_h_sample / dec->comps[i].h_subsample;
  imagesize =  rowstride * (dec->height_blocks * 8*max_v_sample/dec->comps[i].v_subsample);
```

malloc OK
But
imagesize 0

```
dec->c[i].image=malloc(imagesize);
...

//LibOIL API function call
clipconv8x8_u8_s16_c(dec->c[i].image...);
...
```

Code from Swfdec Shockwave Flash Movie Player

Essence of this Bug

Overflow in imagesize computation

```
jpegdecode(image) {
  …
  imagesize = f(image->height)*g(image->width);
  ptr = malloc(imagesize);
  LibraryCall(ptr,…);
}
```

Difficulty of finding this Bug

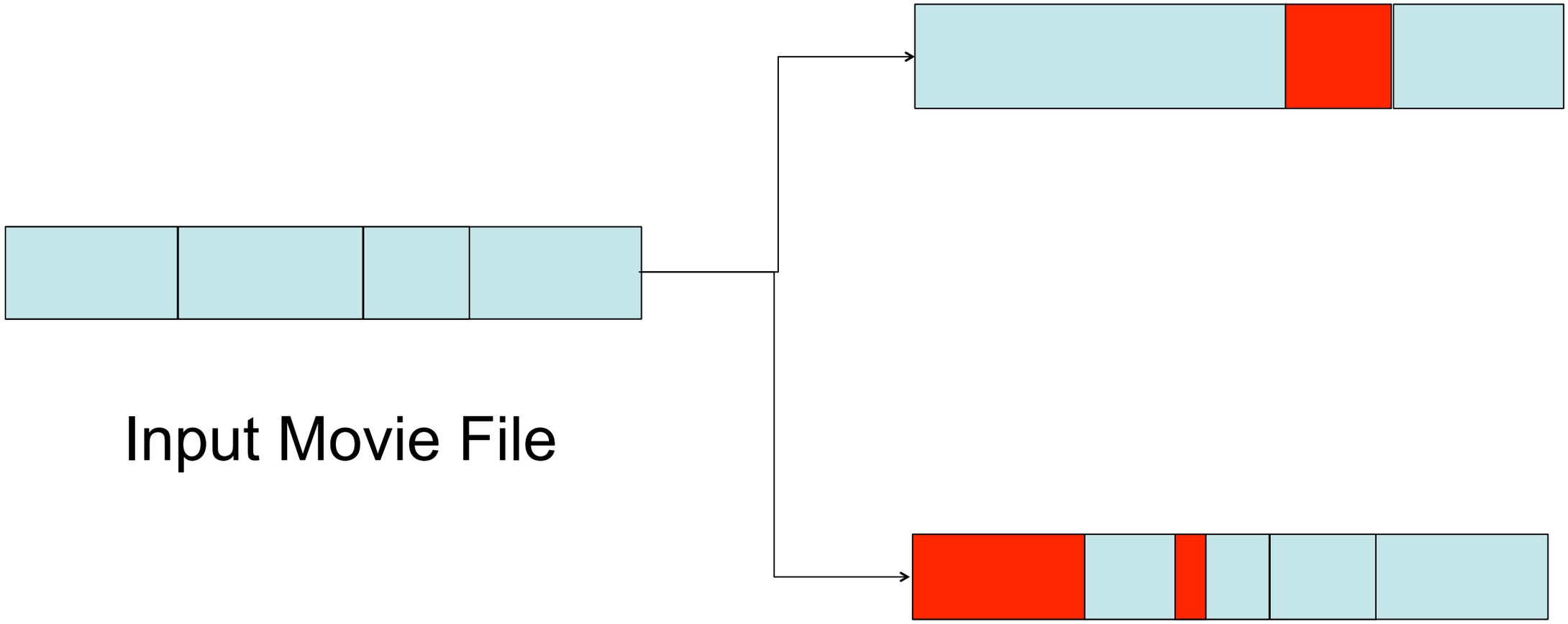
- Deep in the program
  - Stack depth 50
  - Number of instructions in path: ~ 7 million
  - Program source (excluding libraries) ~70 KLOC

- Complex input format
  - Movie file, arbitrarily large and complex

- Few regions of input (height, width)

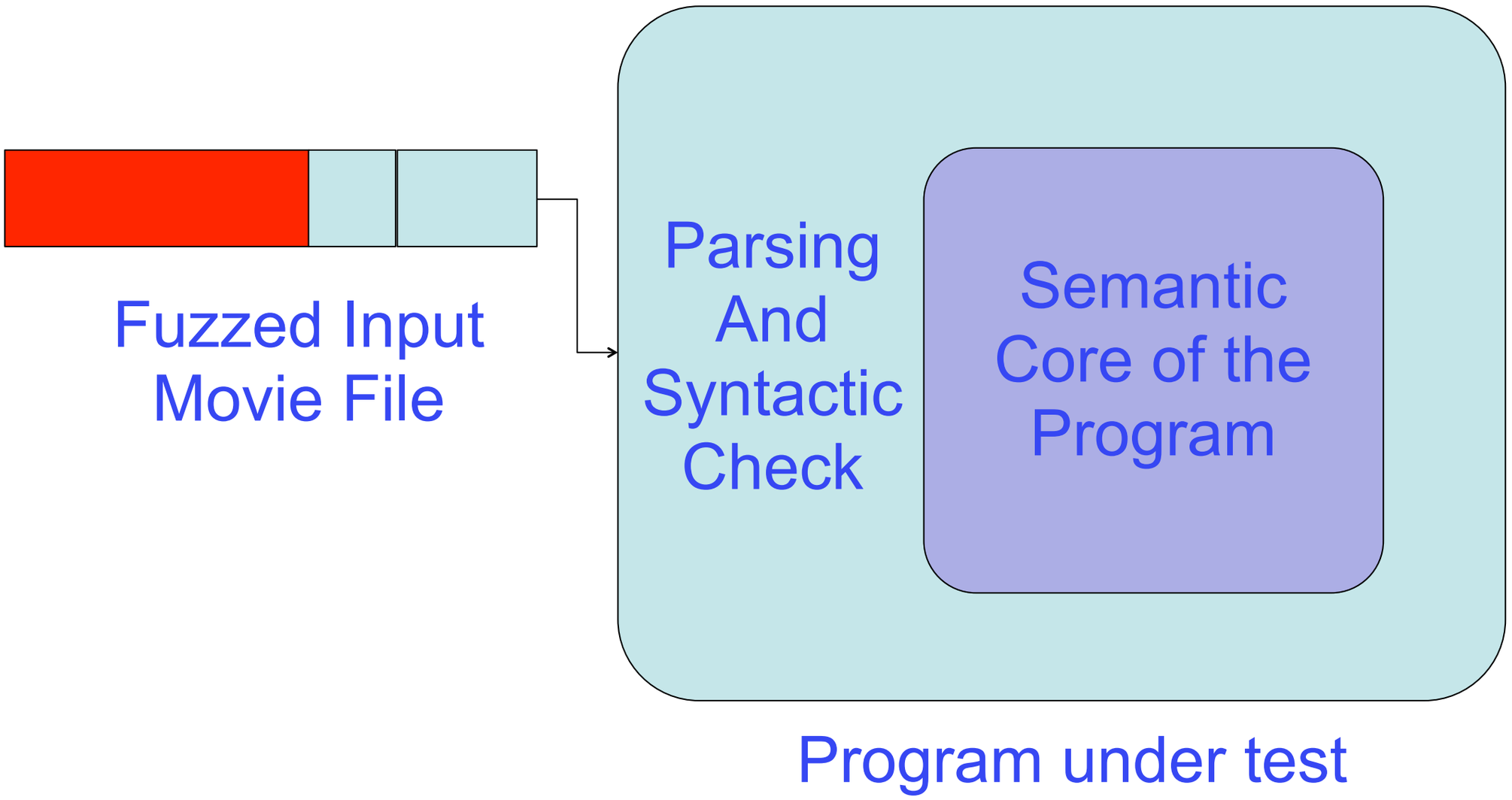
- Construct a test input to find bug automatically

Random Fuzzing

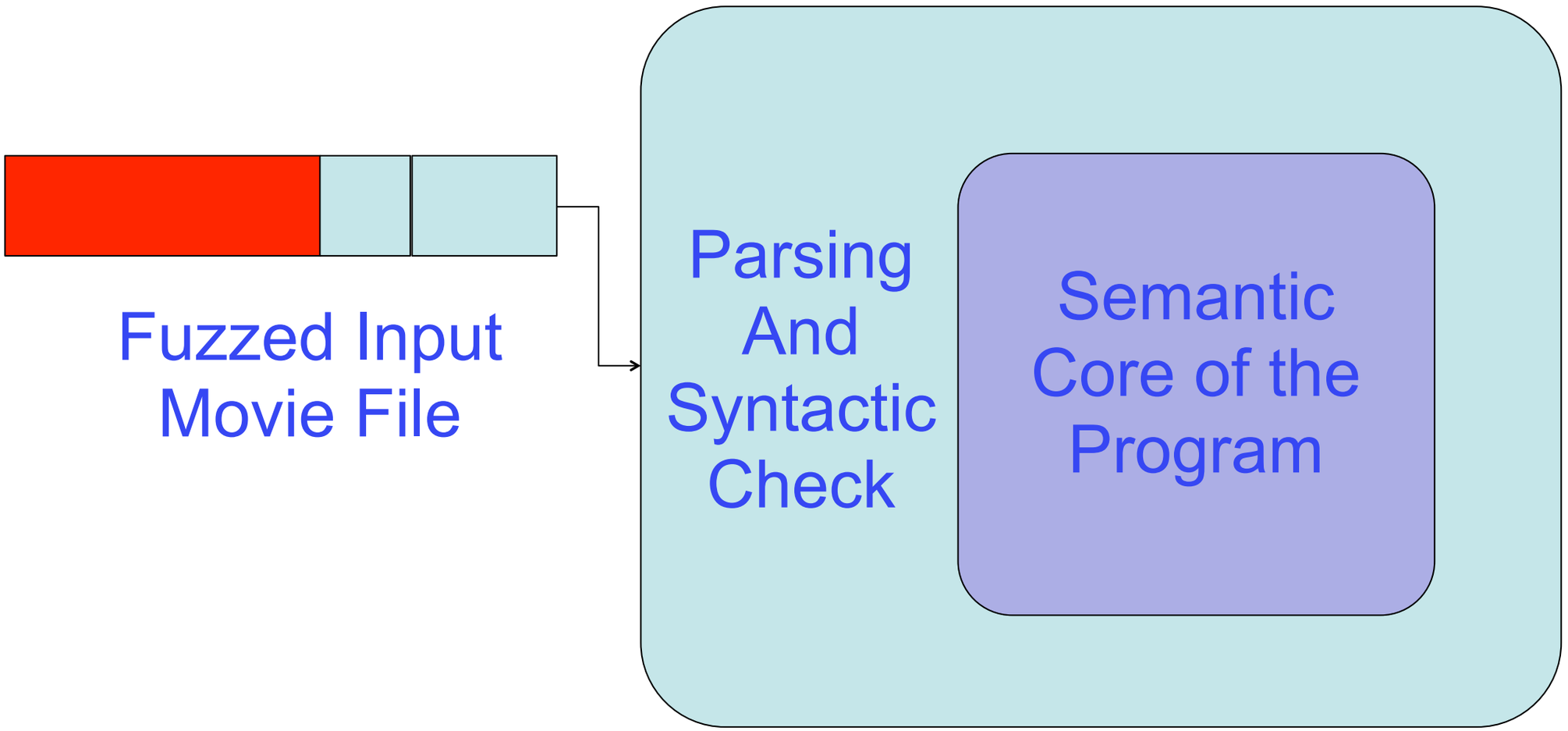


- Multiple Fuzzed Movie Files
- The Fuzzer randomly mutates various fields in the file
- It can work sometimes
- However, often produces mal-formed inputs rejected by parser

Problem with Random Fuzzing

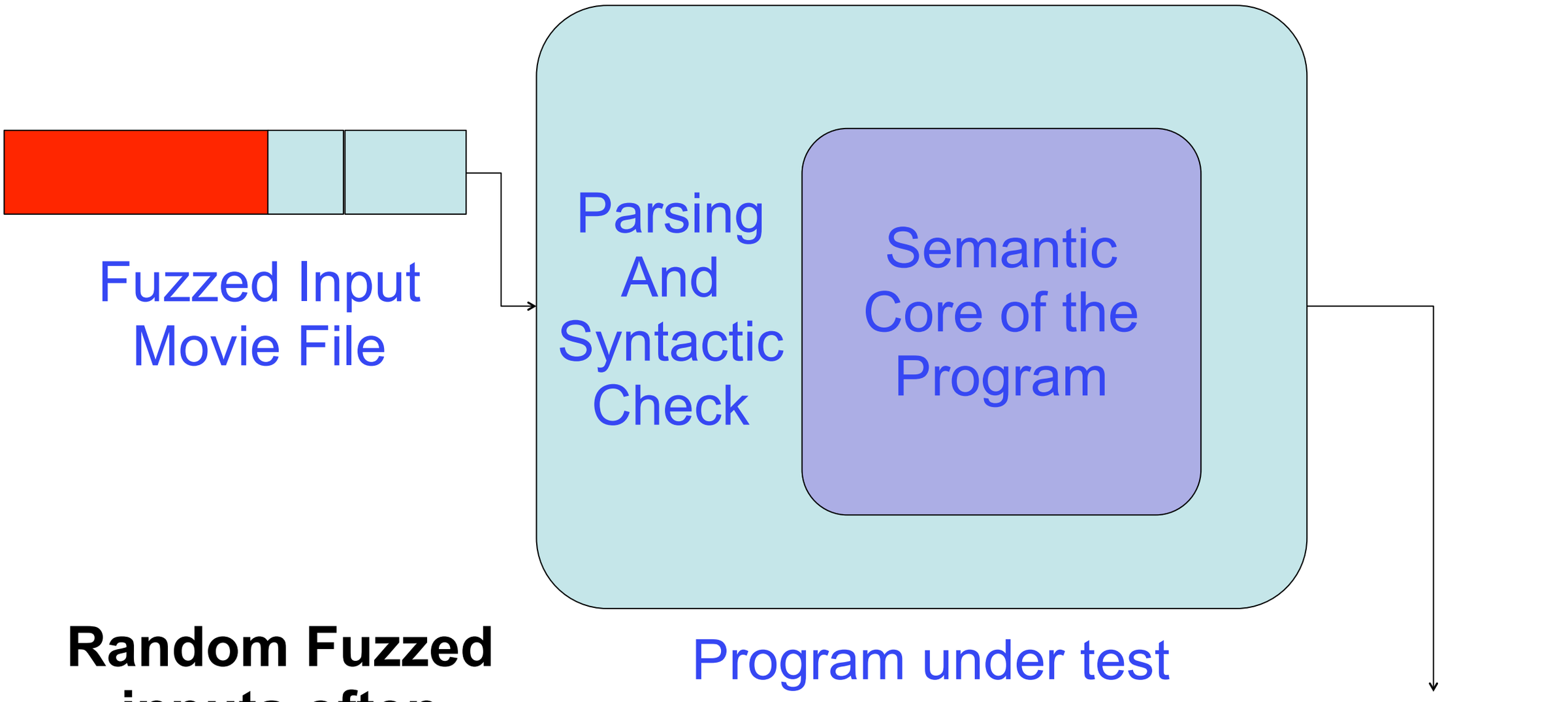


Problem with Random Fuzzing



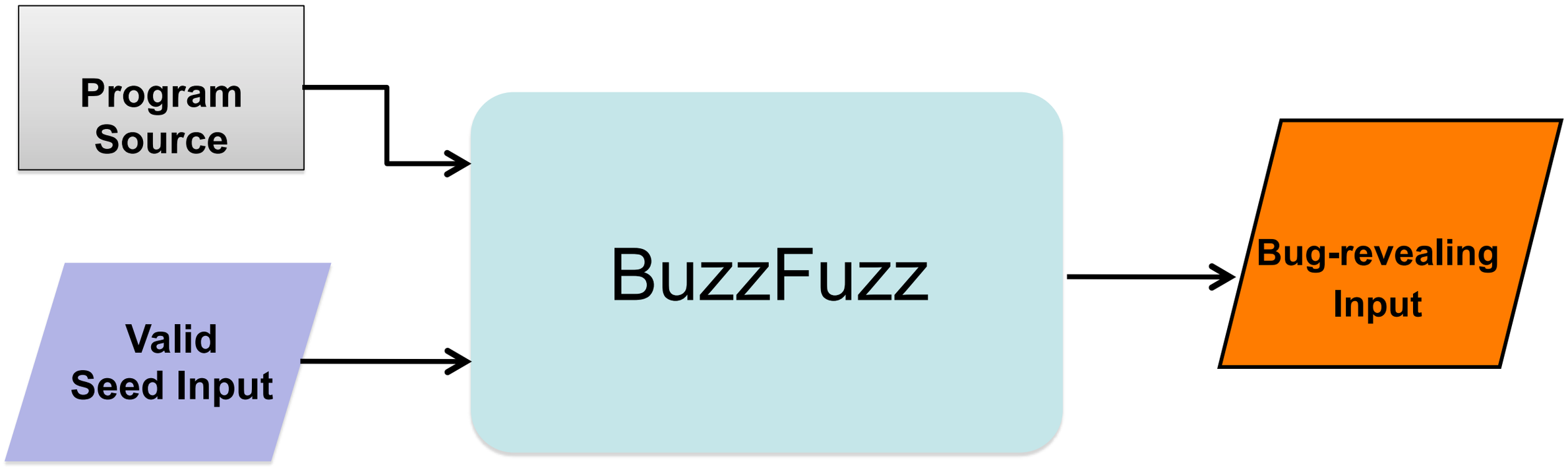
Random Fuzzed inputs often Rejected by Parser

Problem with Random Fuzzing



Random Fuzzed inputs often Rejected by Parser

Information-flow based Fuzzer



Information-flow based Execution

- Instrument source for information-flow analysis (taint-tracking)

- Track input regions to sinks

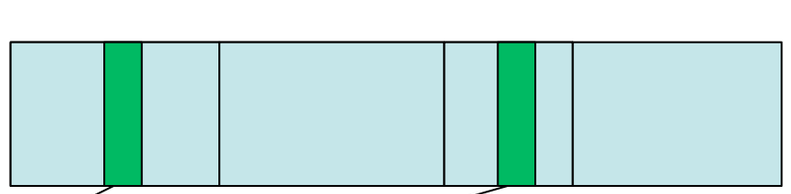
- Dynamic data dependency analysis

- Map from values to set of input bytes

Information-flow based Execution

```
jpeg_decoder(Jpeg* dec) {

dec->width_blocks = (dec->width +
8*max_h_sample - 1)/(8*max_h_sample);

dec->height_blocks = (dec->height +
8*max_h_sample - 1)/(8*max_h_sample);
...
   rowstride = dec->width_blocks *
8*max_h_sample / dec->comps[i].h_subsample;
   image_size =
    rowstride * (dec->height_blocks *
8*max_v_sample/dec->comps[i].v_subsample);

   dec->c[i].image=malloc(image_size);
   ...
   //LibOIL API function call
   clipconv8x8_u8_s16_c(dec->c[i].image...);
   ...
  } //End of Movie Player Code
```

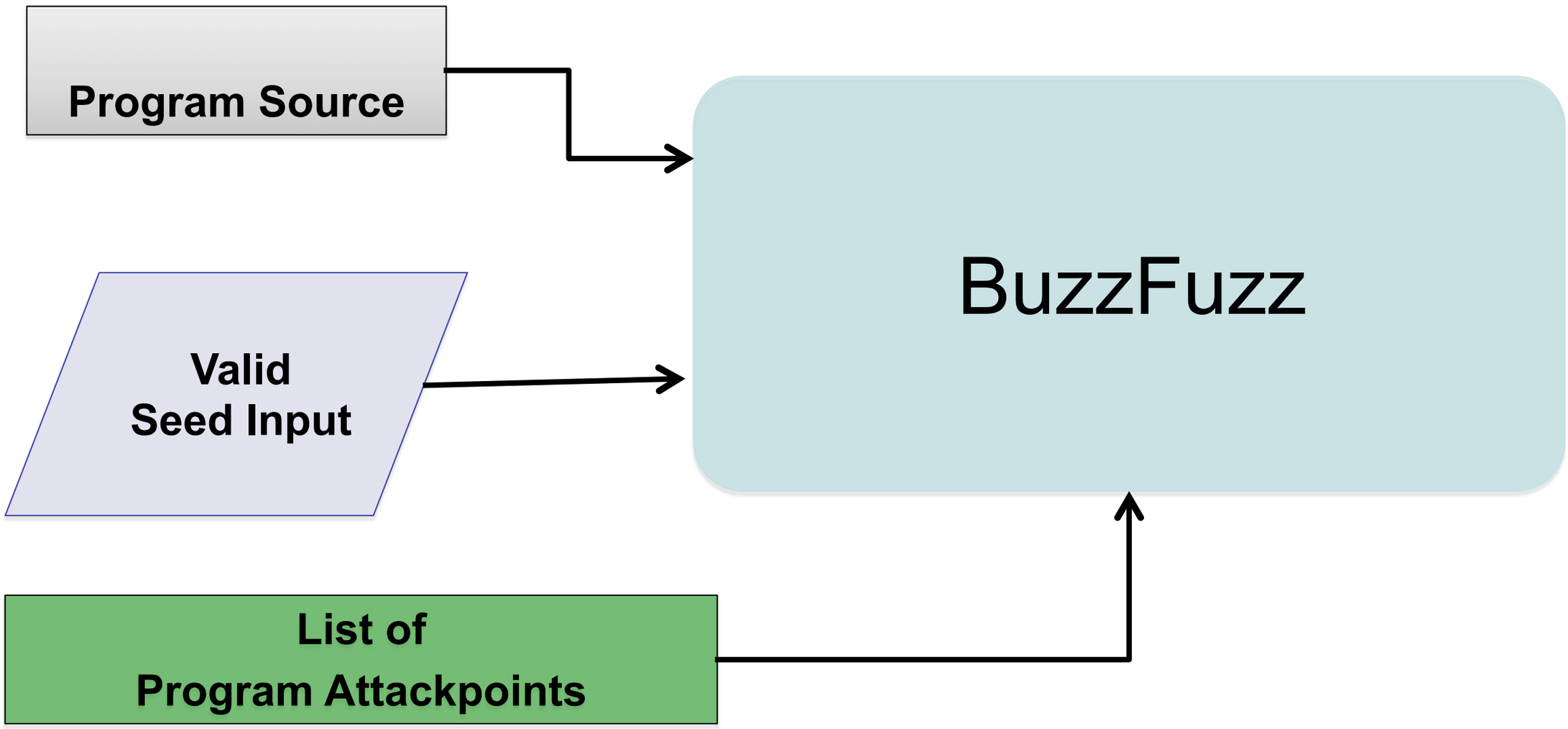
Input
Movie File



Data Dependency from input bytes to computed values

Information-flow based Fuzzer
Selecting Attack Points

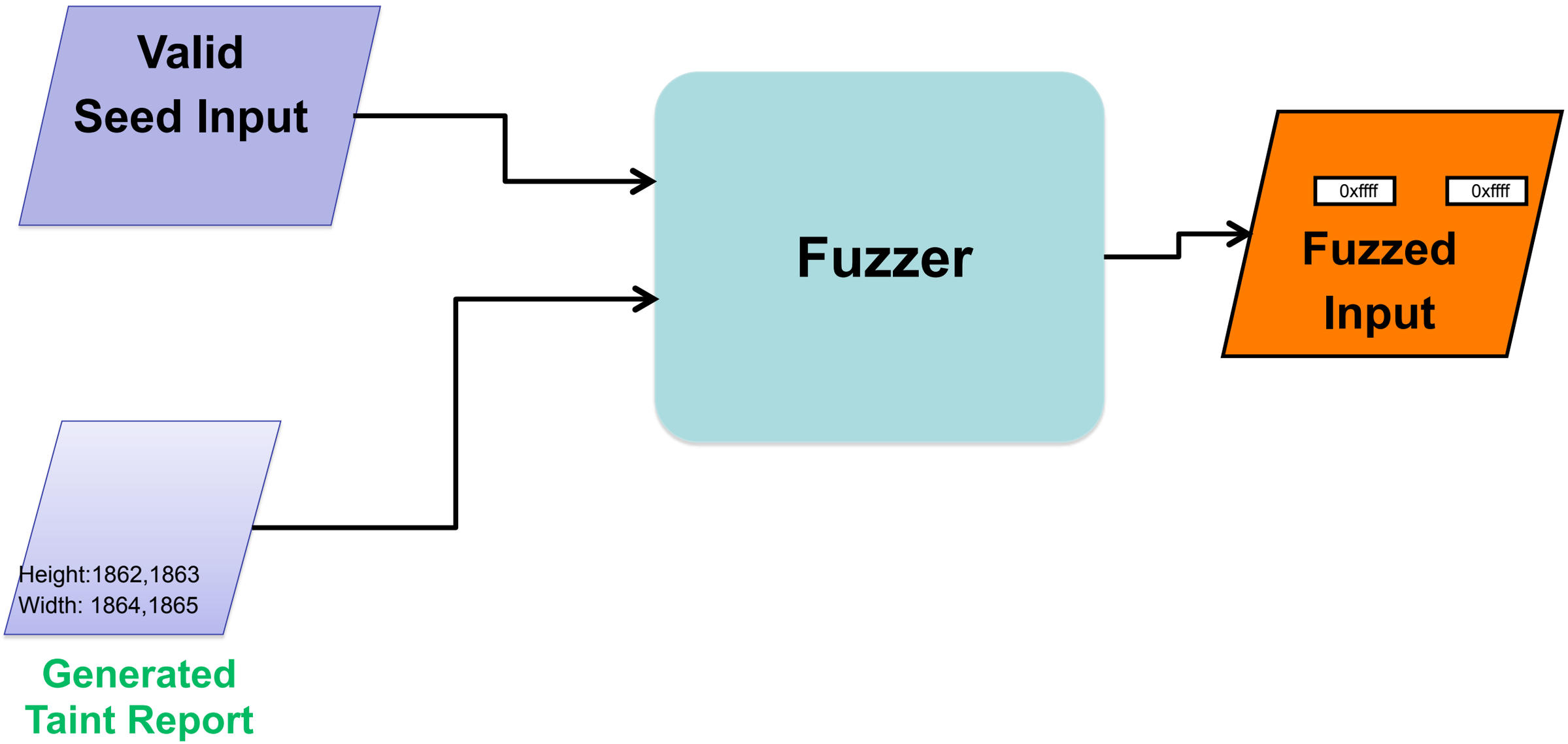
- Library API as Sinks or Attack Points



Information-flow based Fuzzer

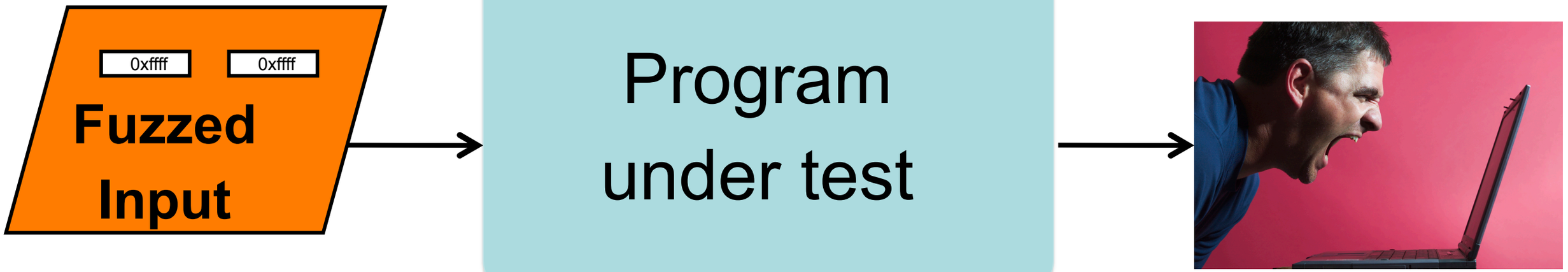
Fuzzing with Extremal Values

Extremal values for integers: 0,-1,int_max

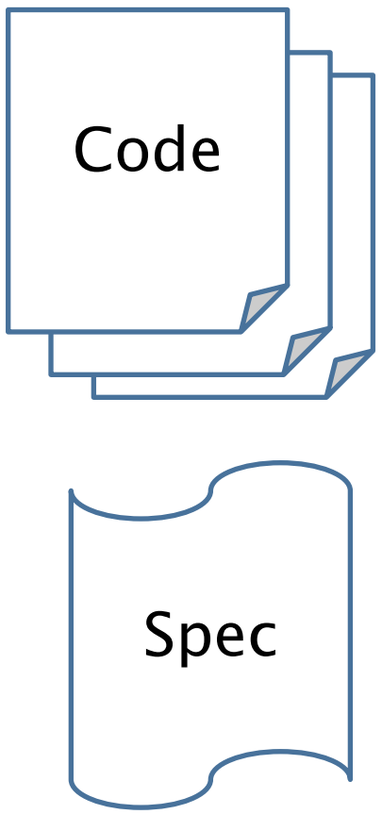


Information-flow based Fuzzer

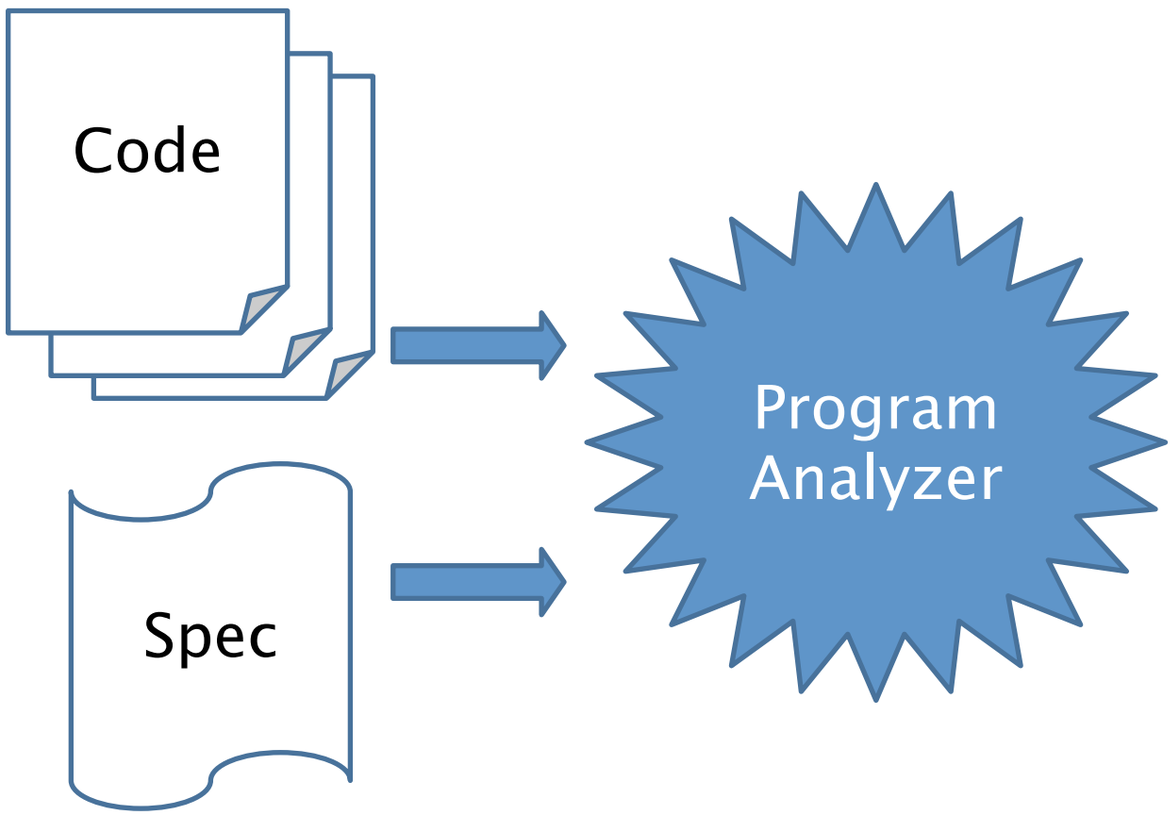
Smarter way to Fuzz



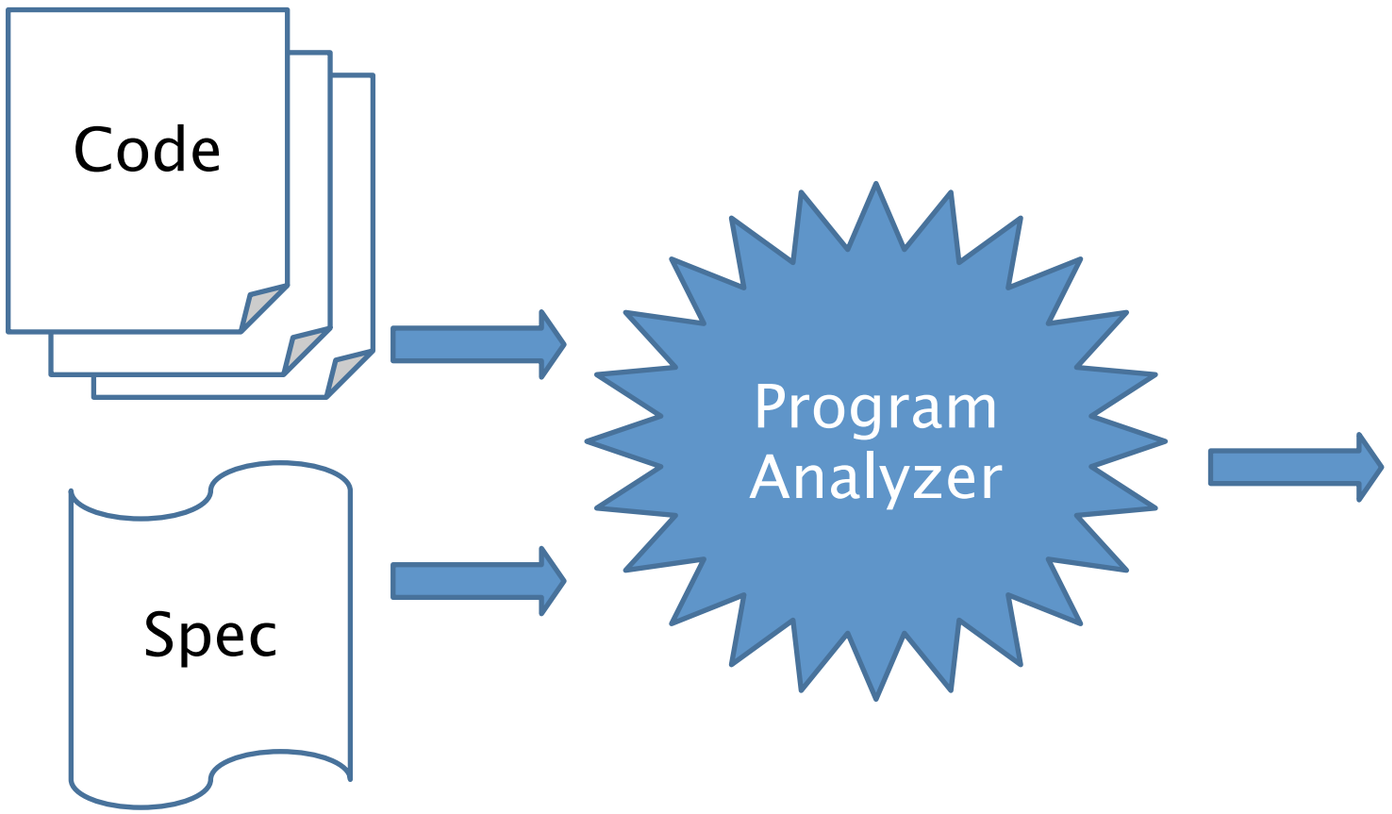
Program Analyzers



Program Analyzers

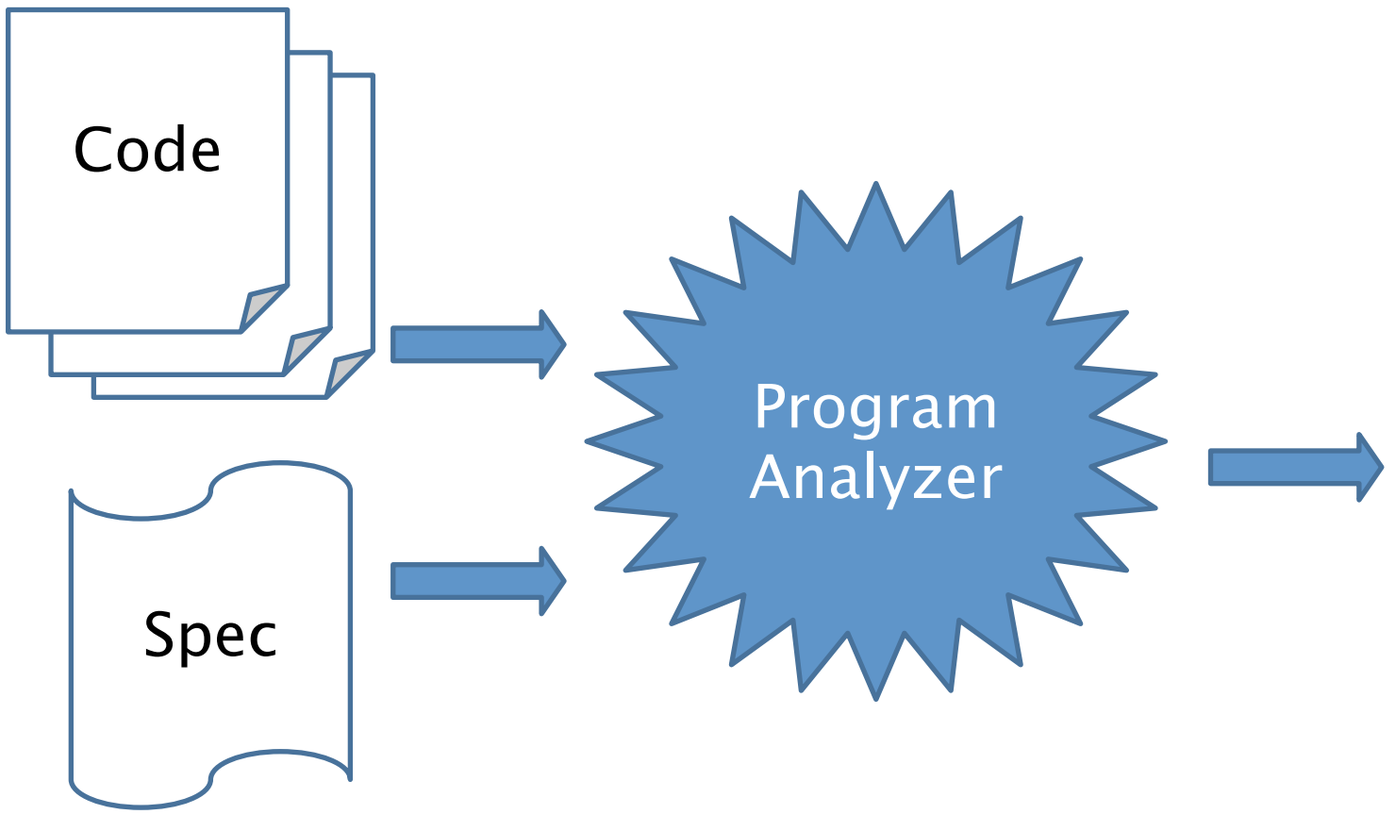


Program Analyzers



Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

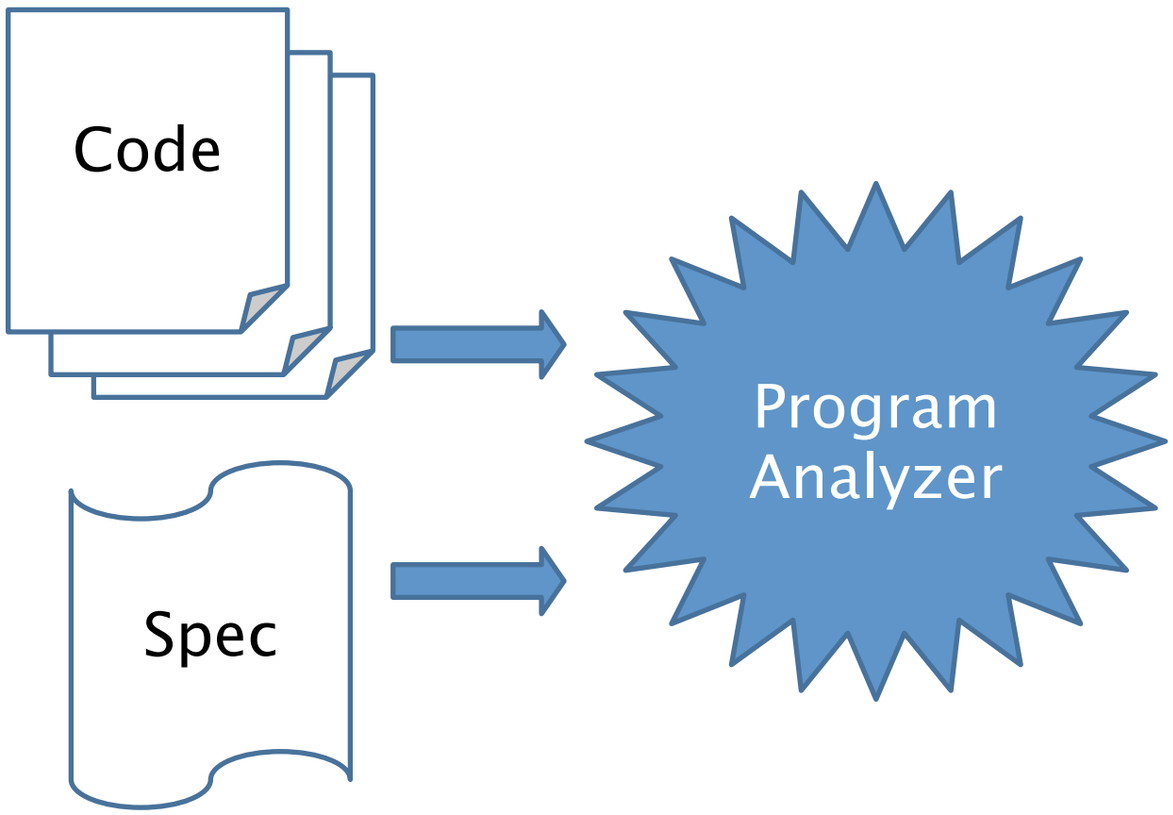
Program Analyzers



analyze large code bases

Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

Program Analyzers

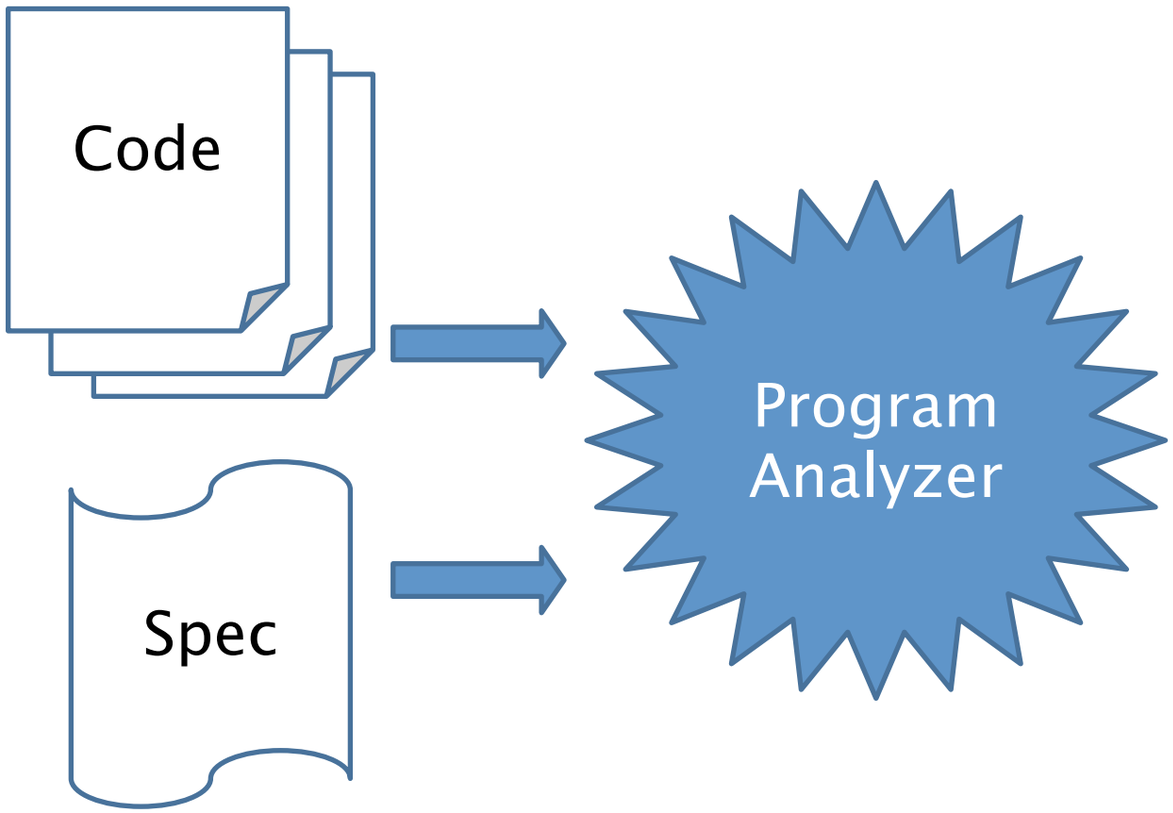


Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

analyze large code bases

potentially reports many warnings

Program Analyzers



analyze large code bases

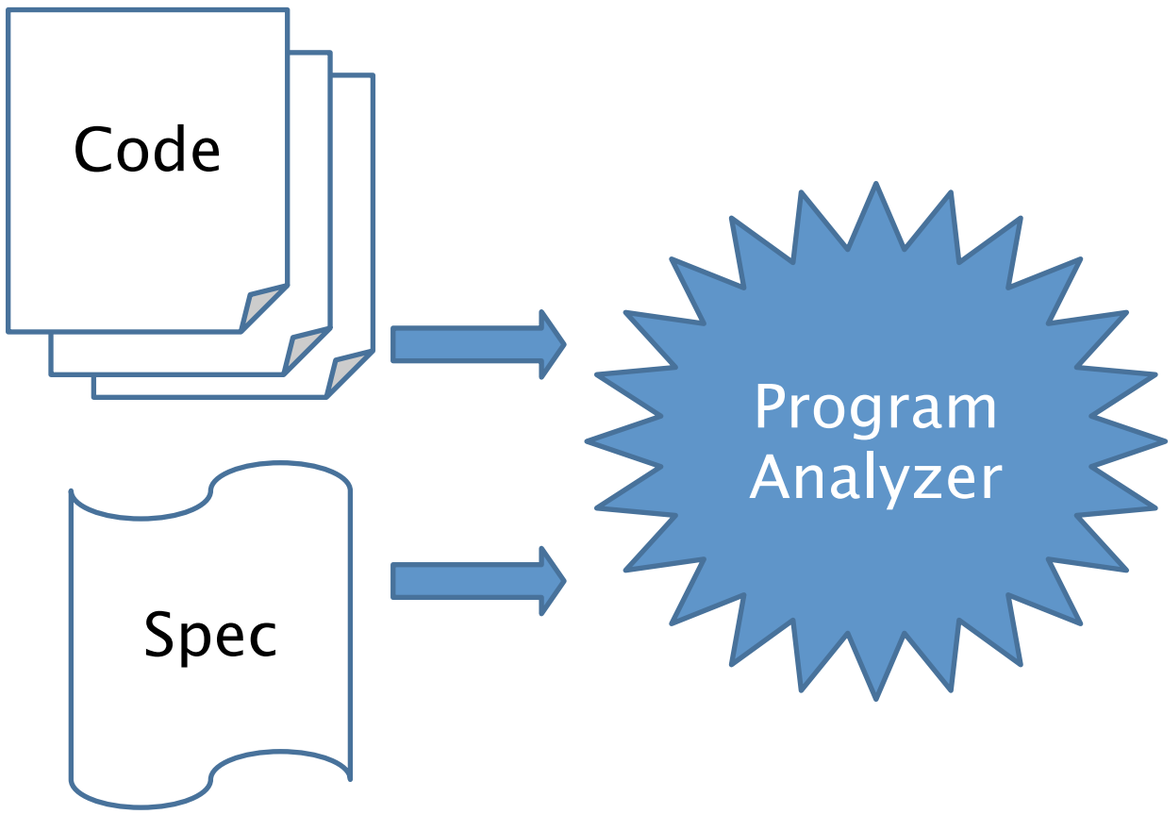
Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

false alarm

false alarm

potentially reports many warnings

Program Analyzers



analyze large code bases

Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

false alarm

false alarm

potentially reports many warnings

may emit false alarms

Static Analysis Goals

- Bug finding
  - Identify code that the programmer wishes to modify or improve

- Correctness
  - Verify the absence of certain classes of errors

Note: some fundamental limitations…

Slides courtesy John Mitchell

Soundness and Completeness

Property	Definition
Soundness	If the program contains an error, the analysis will report a warning. "Sound for reporting correctness"
Completeness	If the analysis reports an error, the program will contain an error. "Complete for reporting correctness"

Slides courtesy John Mitchell

	Complete	Incomplete
Sound		
Unsound		

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms	
Unsound		

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	
Unsound		

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms
Unsound		

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound		

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms	

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms

Slides courtesy John Mitchell

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms **Decidable**

Slides courtesy John Mitchell

Static Analysis

Slides courtesy Andy Chou

Static Analysis

Source Code

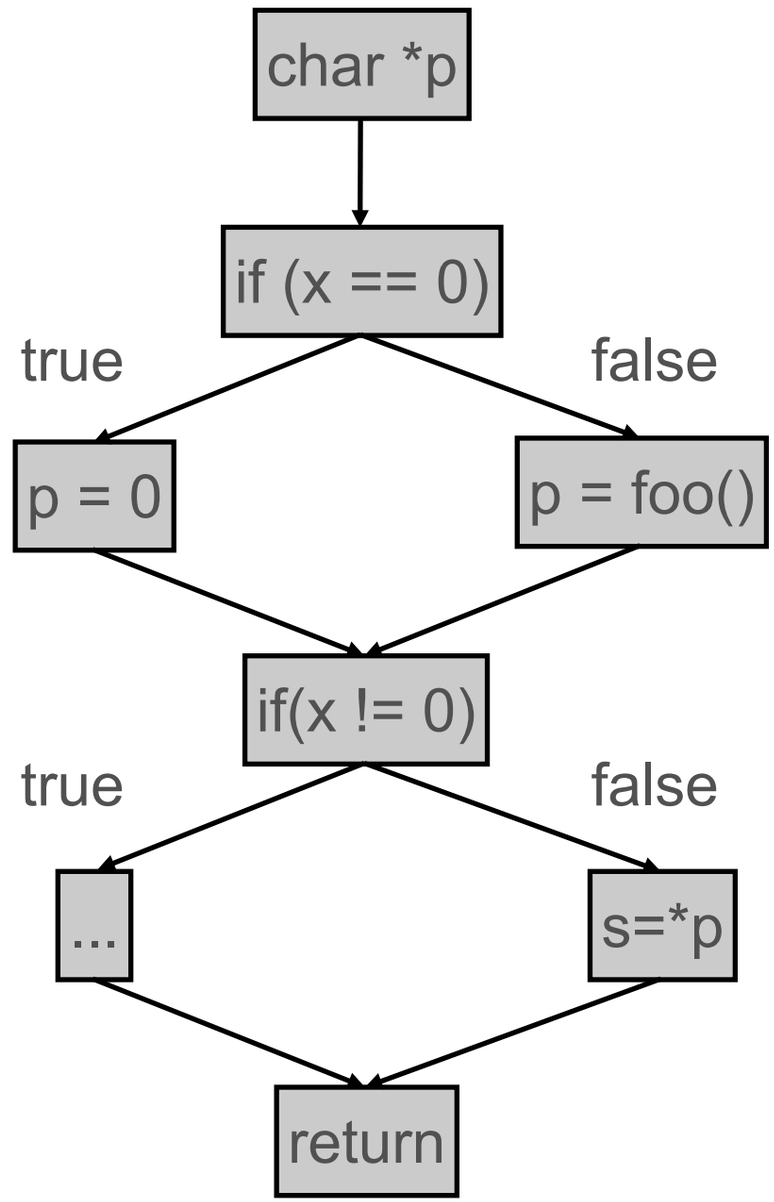
```
  char *p;
if(x == 0)
 p = foo();
    else
   p = 0;

if(x != 0)
    s=*p;
    else
     ...;
  return;
```

Slides courtesy Andy Chou

Static Analysis

Source Code

```
  char *p;
if(x == 0)
 p = foo();
   else
   p = 0;

if(x != 0)
   s=*p;
   else
    ...;
  return;
```



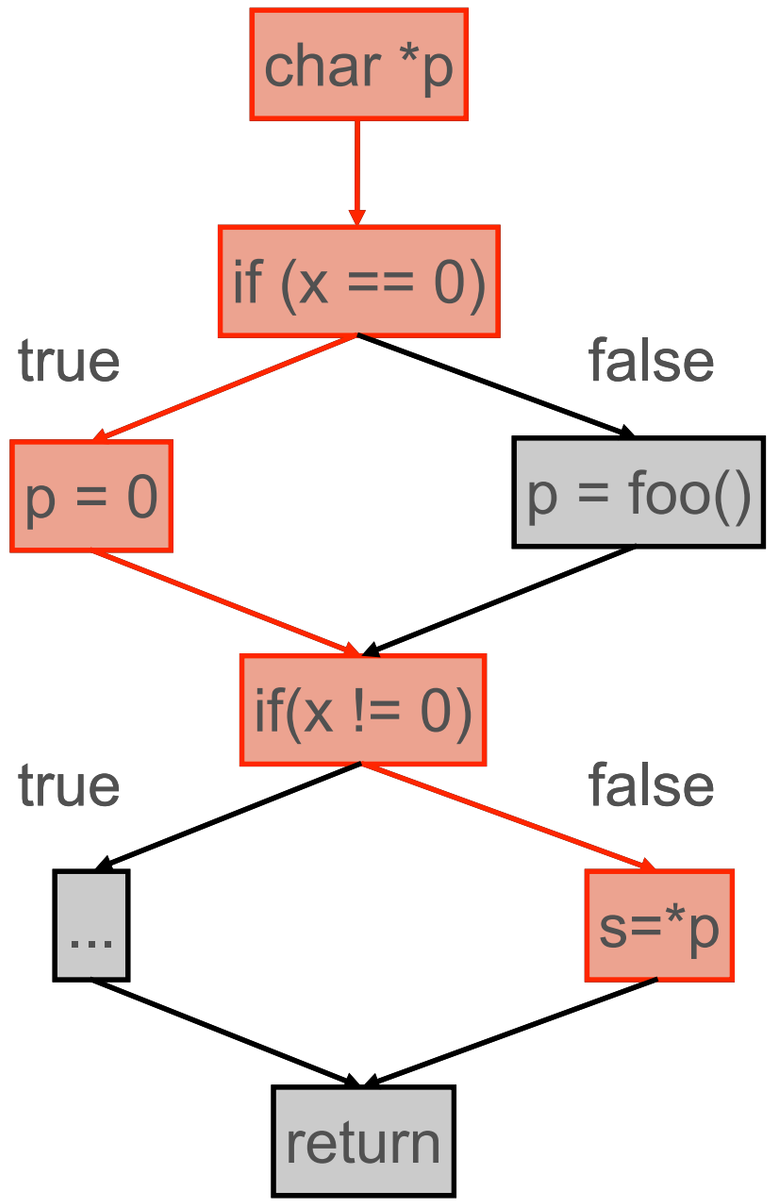
Slides courtesy Andy Chou

Static Analysis

Source Code

```
char *p;
if(x == 0)
 p = foo();
   else
  p = 0;

if(x != 0)
   s=*p;
   else
    ...;
 return;
```

Symbolic CFG Analysis



Defects detected

Assigning: p=0

x!=0 taking true branch

Dereferencing null pointer p

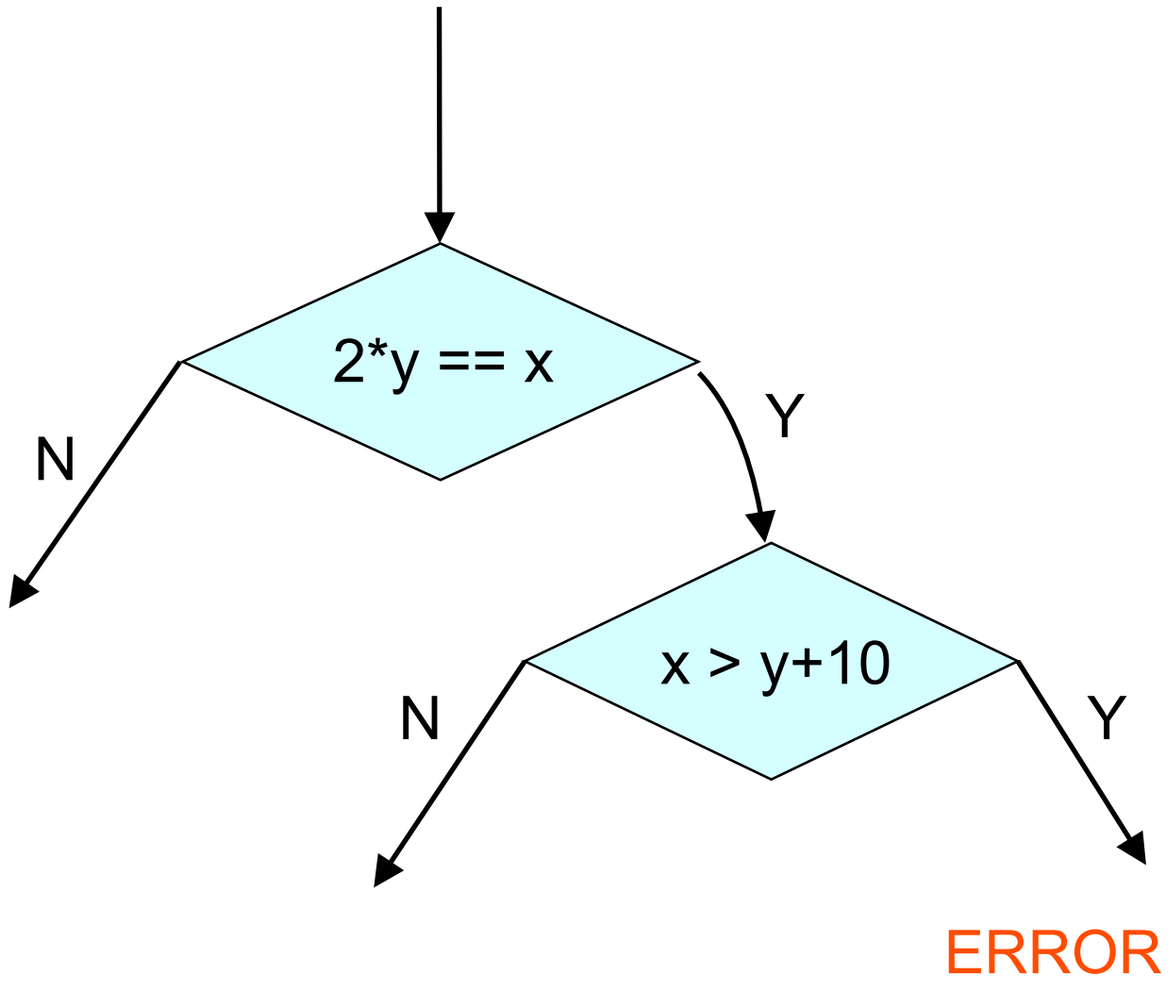
Slides courtesy Andy Chou

Example

```
int double (int v) {
    return 2*v;
}


void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

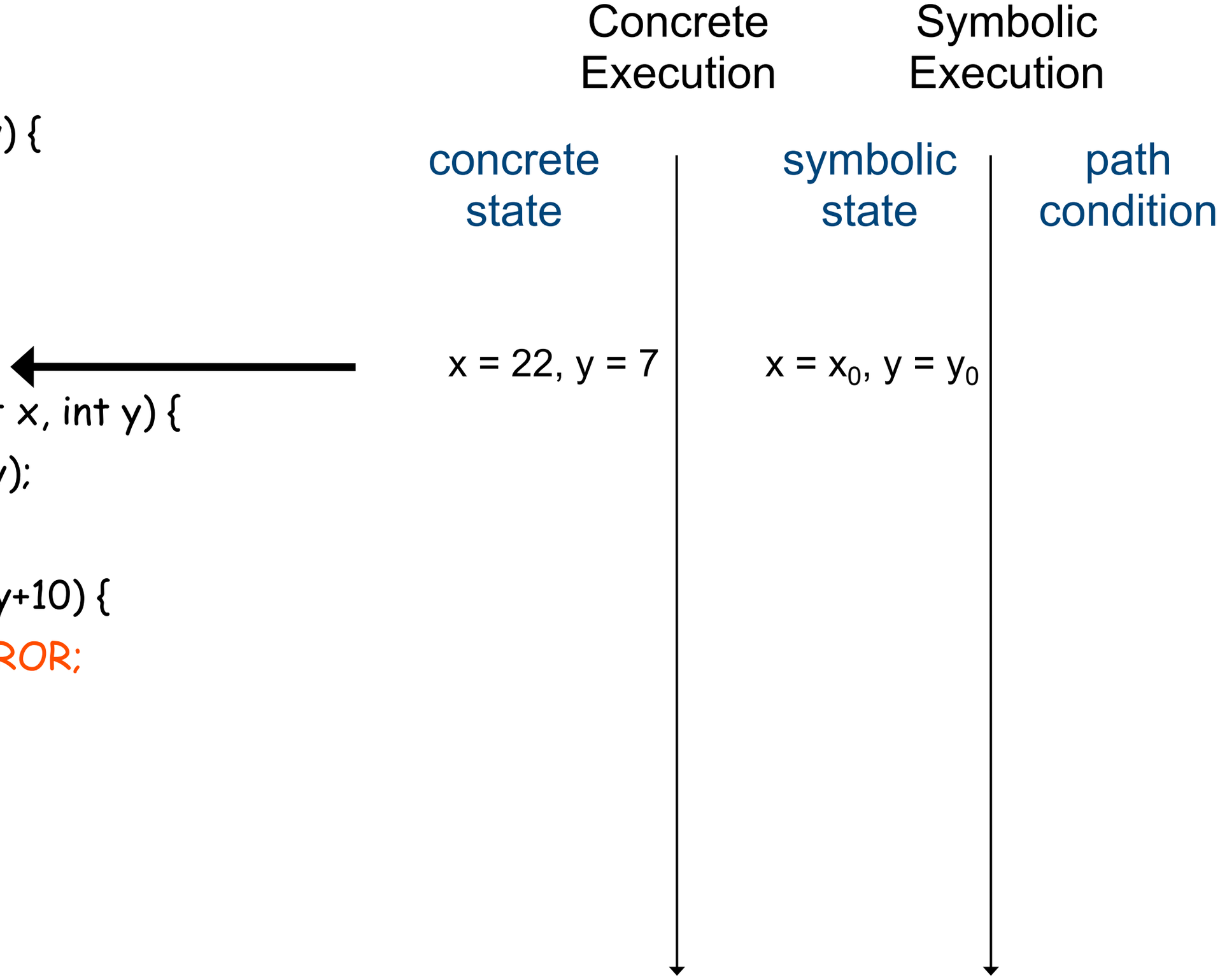
Example

```
int double (int v) {
      return 2*v;
}


void testme (int x, int y) {
      z = double (y);
      if (z == x) {
            if (x > y+10) {
                  ERROR;
            }
      }
}
```



Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```


Concrete Execution: concrete state	Symbolic Execution: symbolic state	Symbolic Execution: path condition
x = 22, y = 7, z = 14	x = x_0, y = y_0, z = 2*y_0	

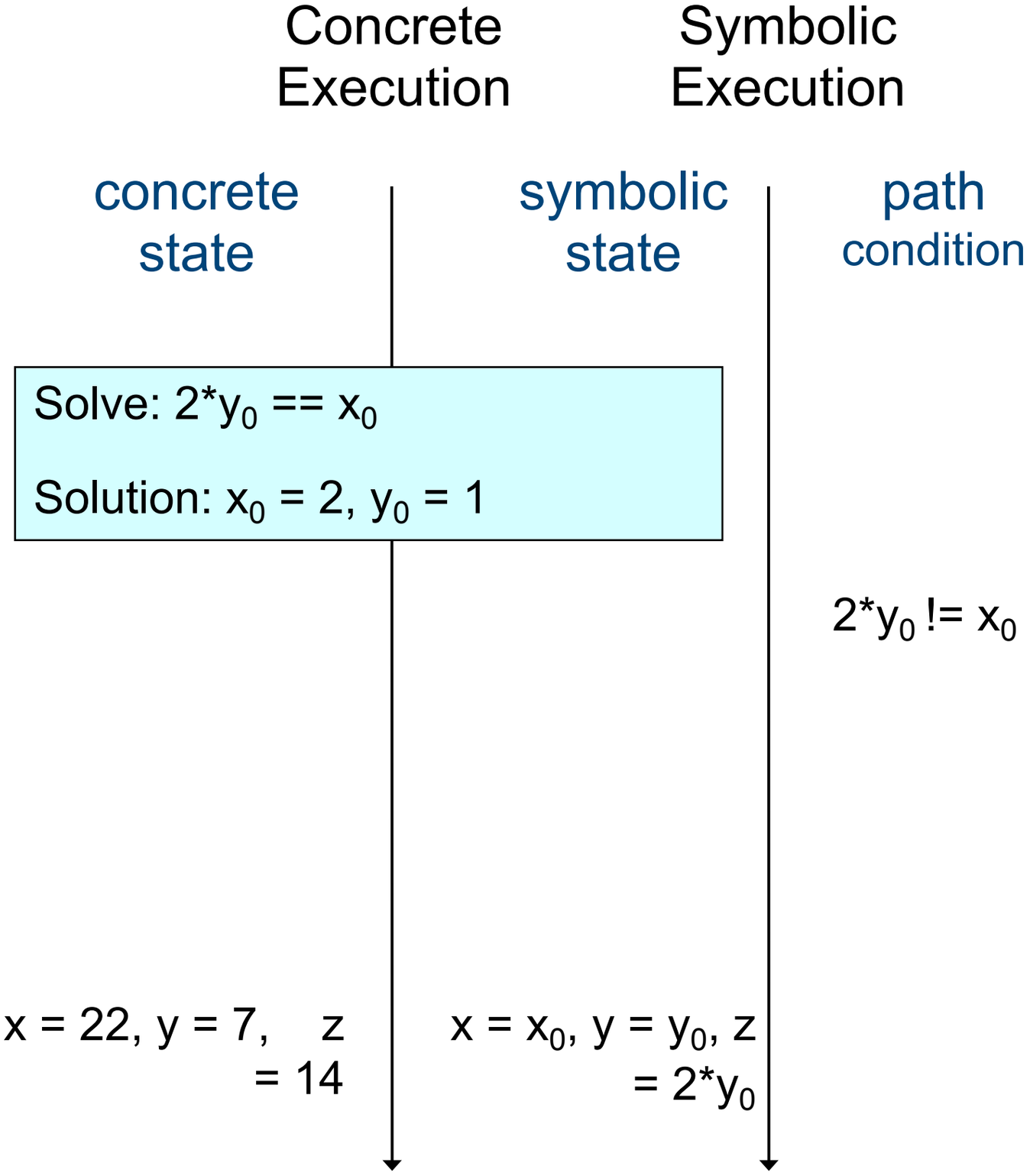
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution		Symbolic Execution	
concrete state		symbolic state	path condition
x = 22, y = 7, z = 14	⟵	x = x_0, y = y_0, z = $2*y_0$	$2*y_0 != x_0$

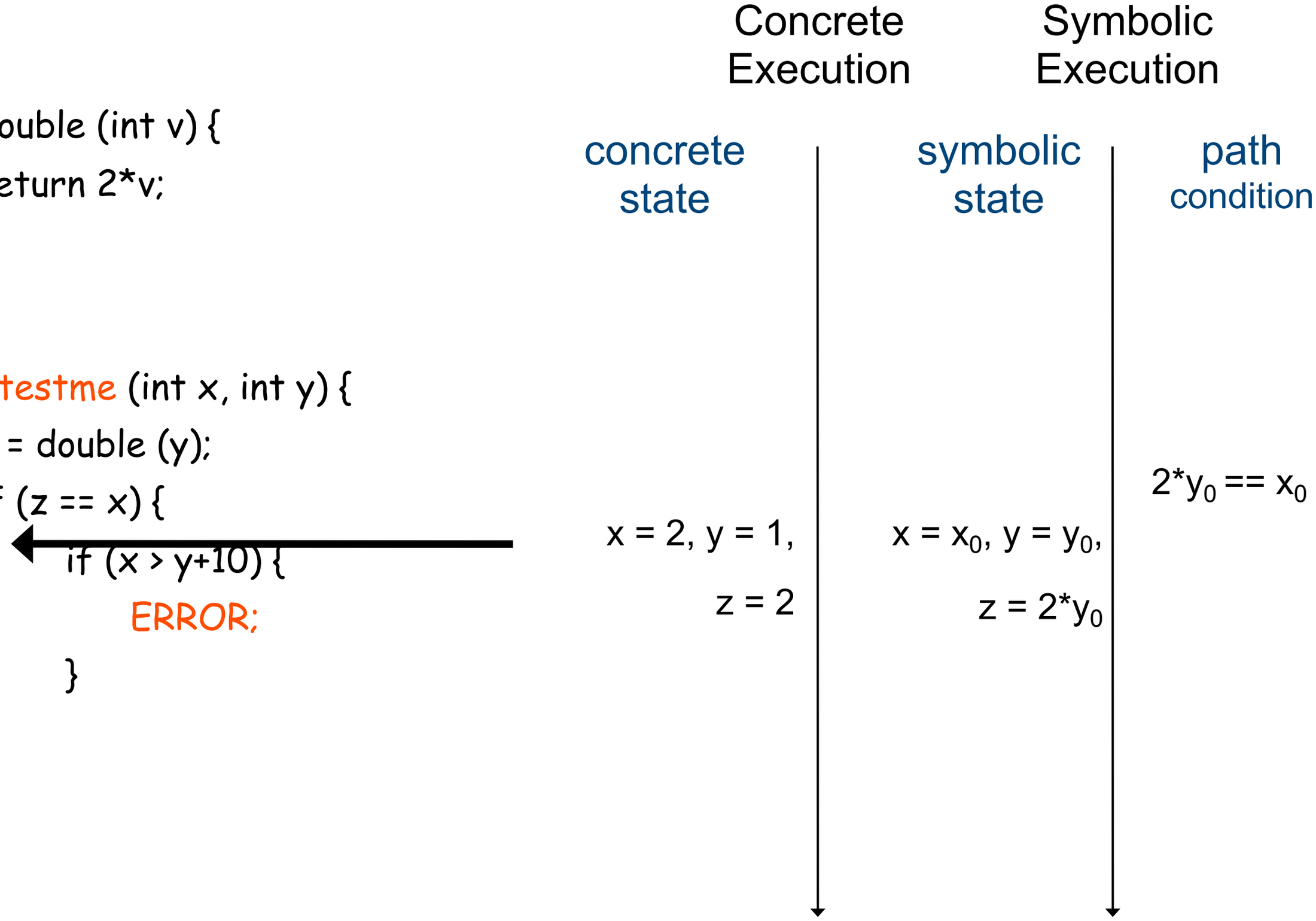
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



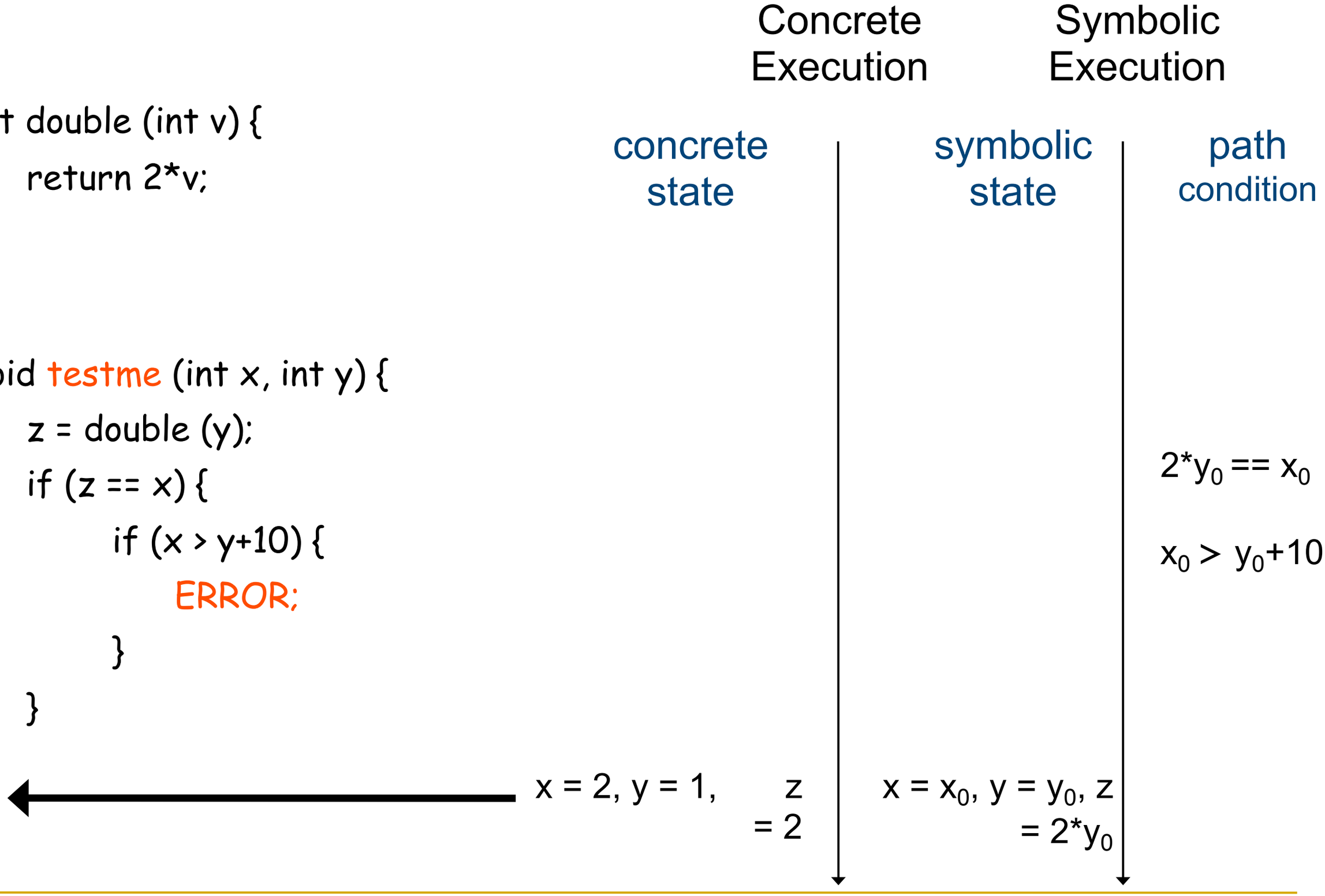
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



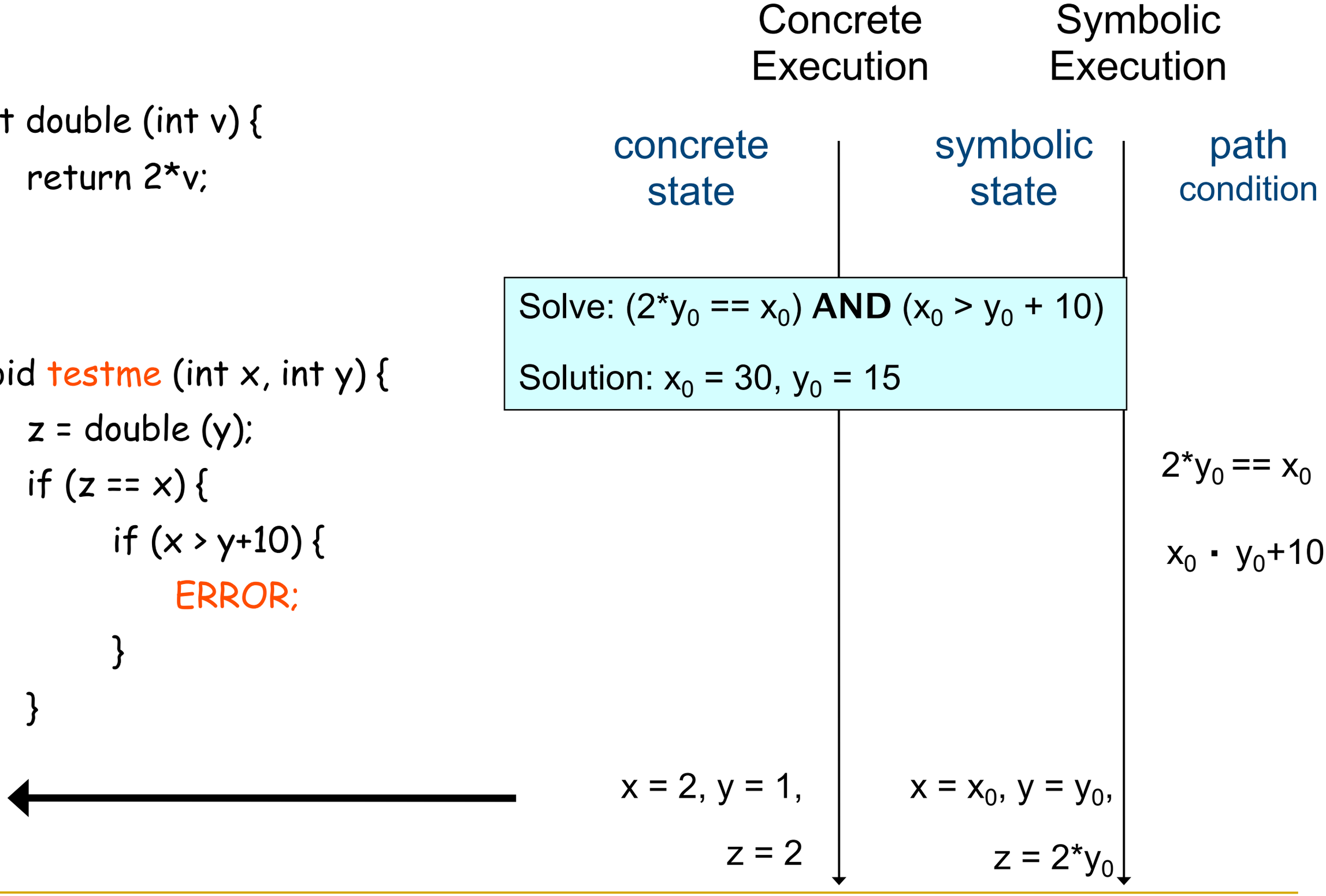
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



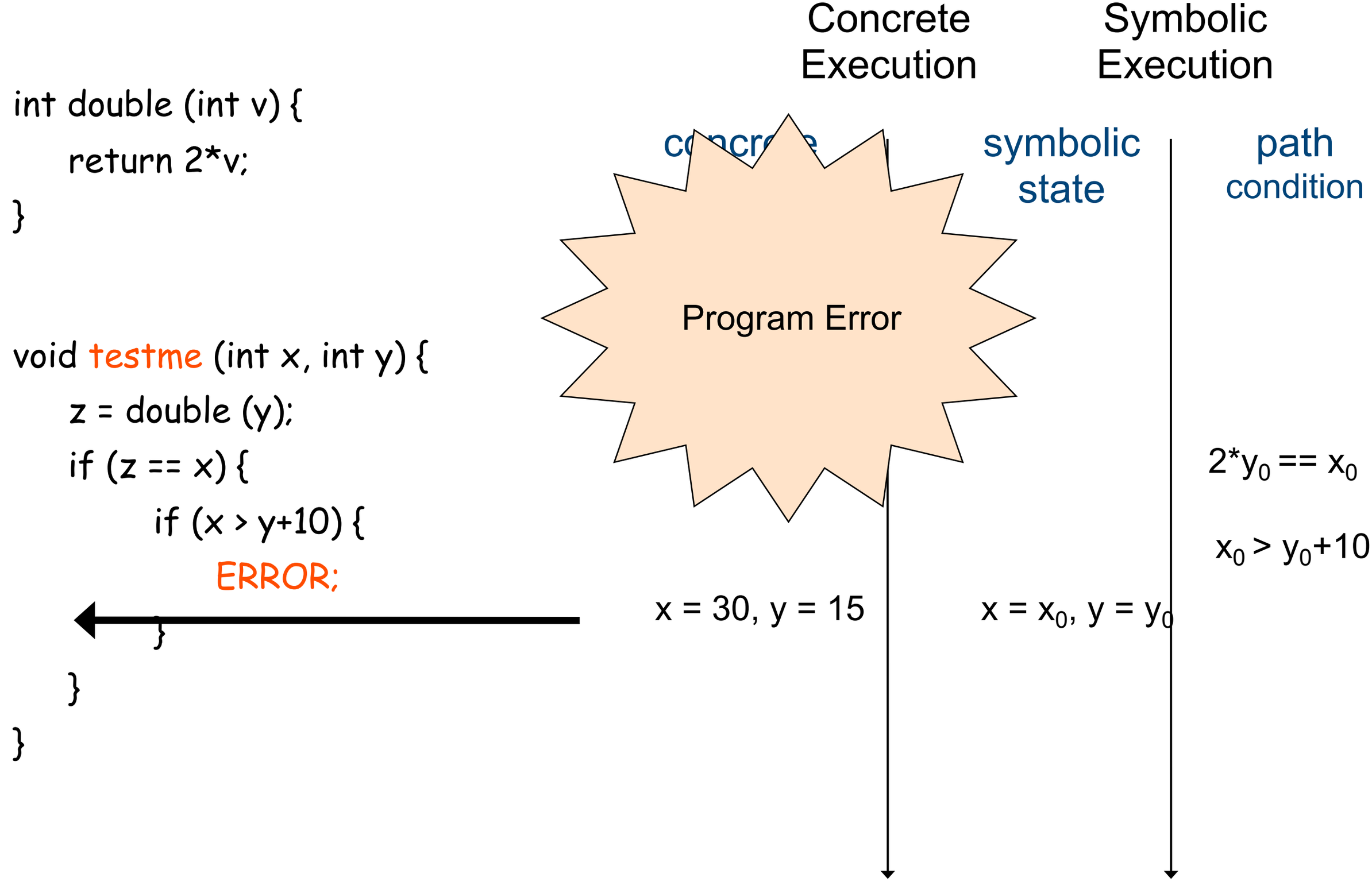
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 30, y = 15	$x = x_0, y = y_0$	

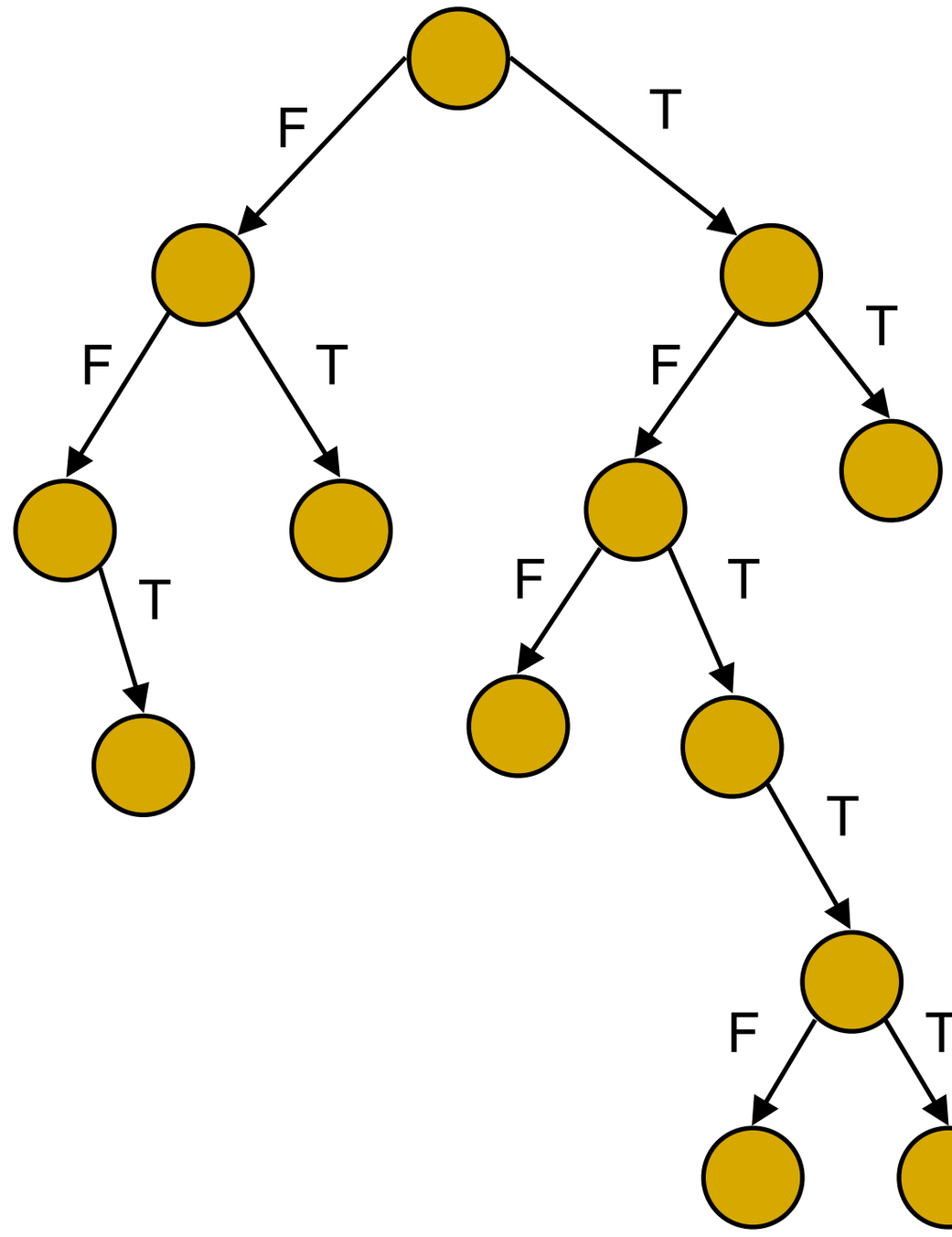
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



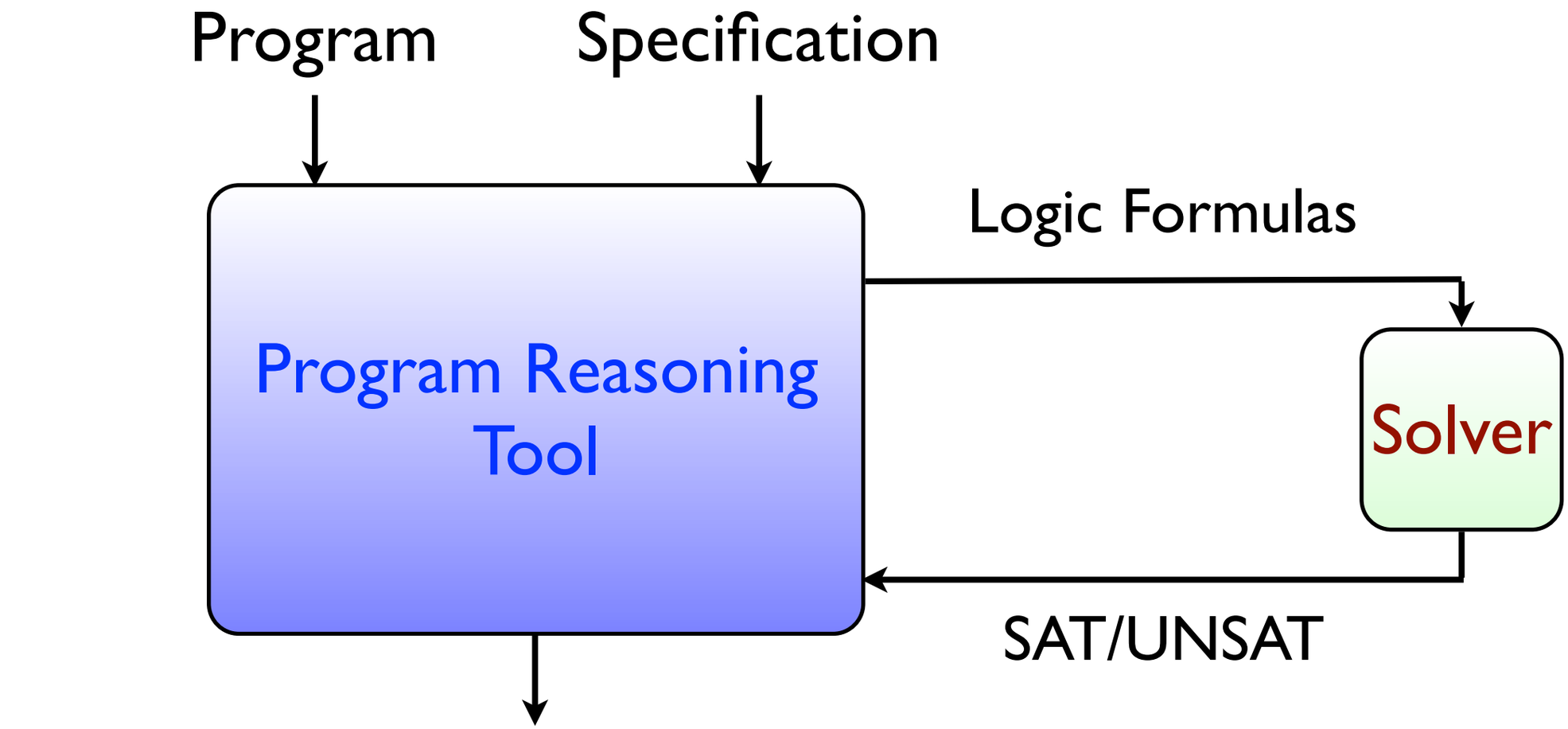
Explicit Path (not State) Model

- Traverse all execution paths one by one to detect errors
  - assertion violations
  - program crash
  - uncaught exceptions
- combine with valgrind to discover memory errors



Reliability through Logical Reasoning

Engineering, Usability, Novelty



What is at the Core?

The SAT/SMT Problem

- Rich logics (Modular arithmetic, Arrays, Strings,...)
- NP-complete, PSPACE-complete,...
- Practical, scalable, usable, automatic
- Enable novel software reliability approaches

Programs Reasoning & STP

Why Bit-vectors and Arrays

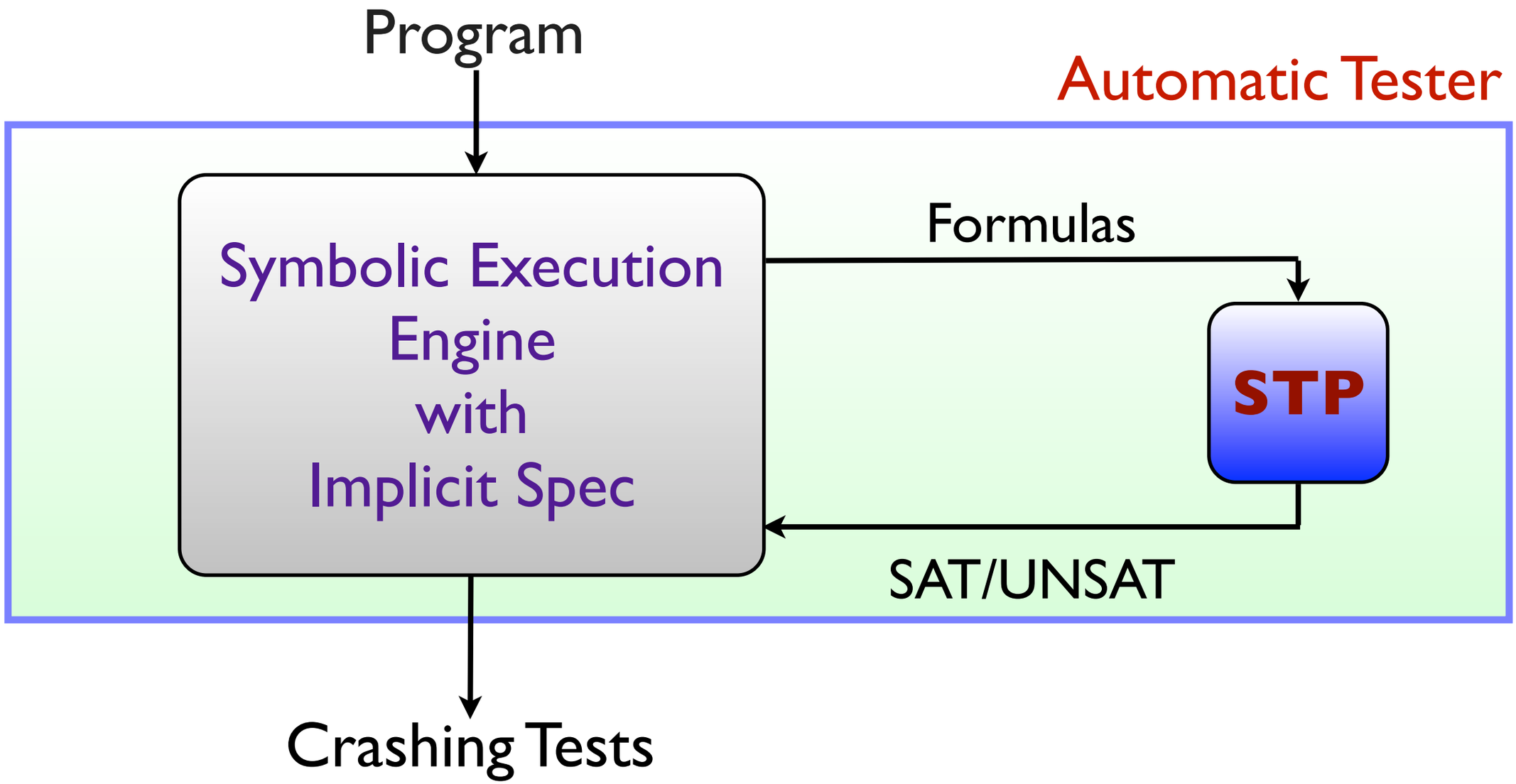
- STP logic tailored for software reliability applications
- Support symbolic execution/program analysis

C/C++/Java/...	Bit-vectors and Arrays
Int Var Char Var	32 bit variable 8 bit variable
Arithmetic operation (x+y, x-y, x*y, x/y,...)	Arithmetic function (x+y,x-y,x*y,x/y,...)
assignments x = expr;	equality x = expr;
if conditional if(cond) x = $expr^1$ else x = $expr^2$	if-then-else construct x = if(cond) $expr^1$ else $expr^2$
inequality	inequality predicate
Memory read/write x = *ptr + i;	Array read/write ptr[]; x = Read(ptr,i);
Structure/Class	Serialized bit-vector expressions
Function	Symbolic execution
Loops	Bounding

How to Automatically Crash Programs?

Concolic Execution & STP

Problem: Automatically generate crashing tests given only the code



How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

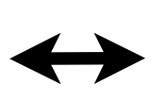
- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```



Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
//INTEGER OVERFLOW QUERY
0 <= j <= process(len_field);
ptr + i + j = 0?
```

- Formula captures computation
- Tester attaches formula to capture spec