Finding vulnerabilities

CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

rist at cs dot wisc dot edu

Administrative

- Do people have access to running the HW1 VM?
- Are people still on wait list?
  - send me your name and ID

Finding vulnerabilities

Manual analysis

Simple example: double free

Fuzzing tools

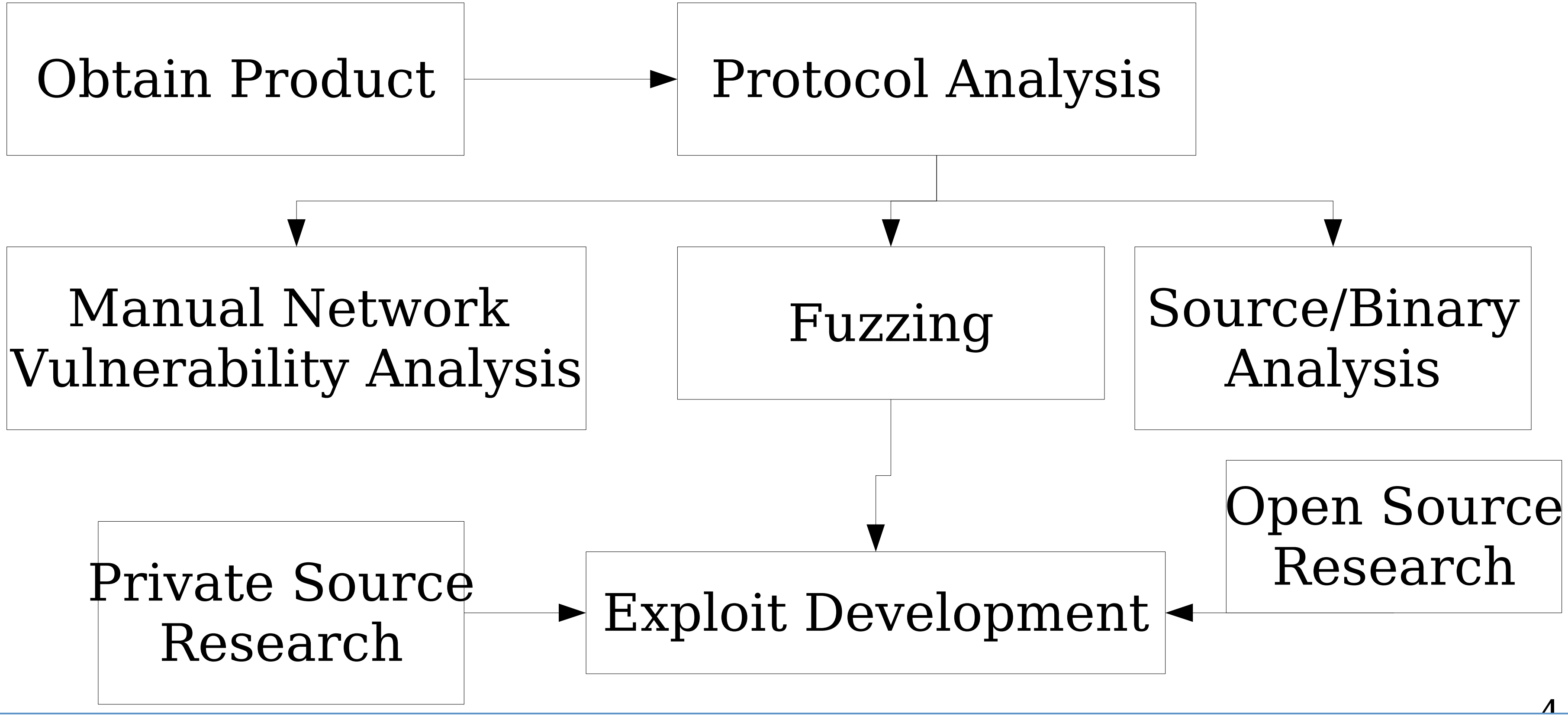
Static analysis, dynamic analysis

...

Hackers use People, Processes and Technology to obtain a singular goal: Information dominance

From “How Hackers Look for Bugs”, Dave Aitel

Take a sample product X and attack it remotely

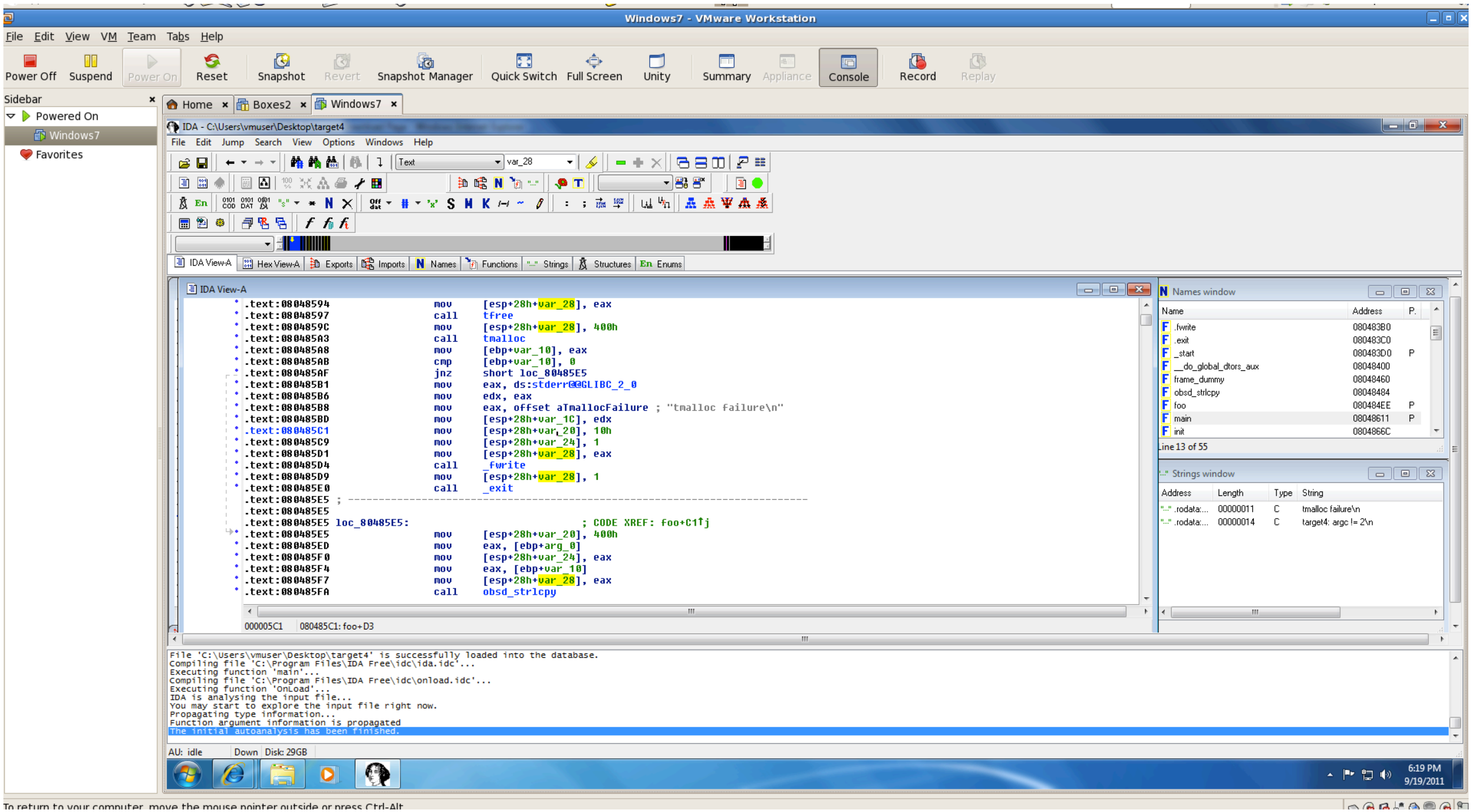


From “How Hackers Look for Bugs”, Dave Aitel

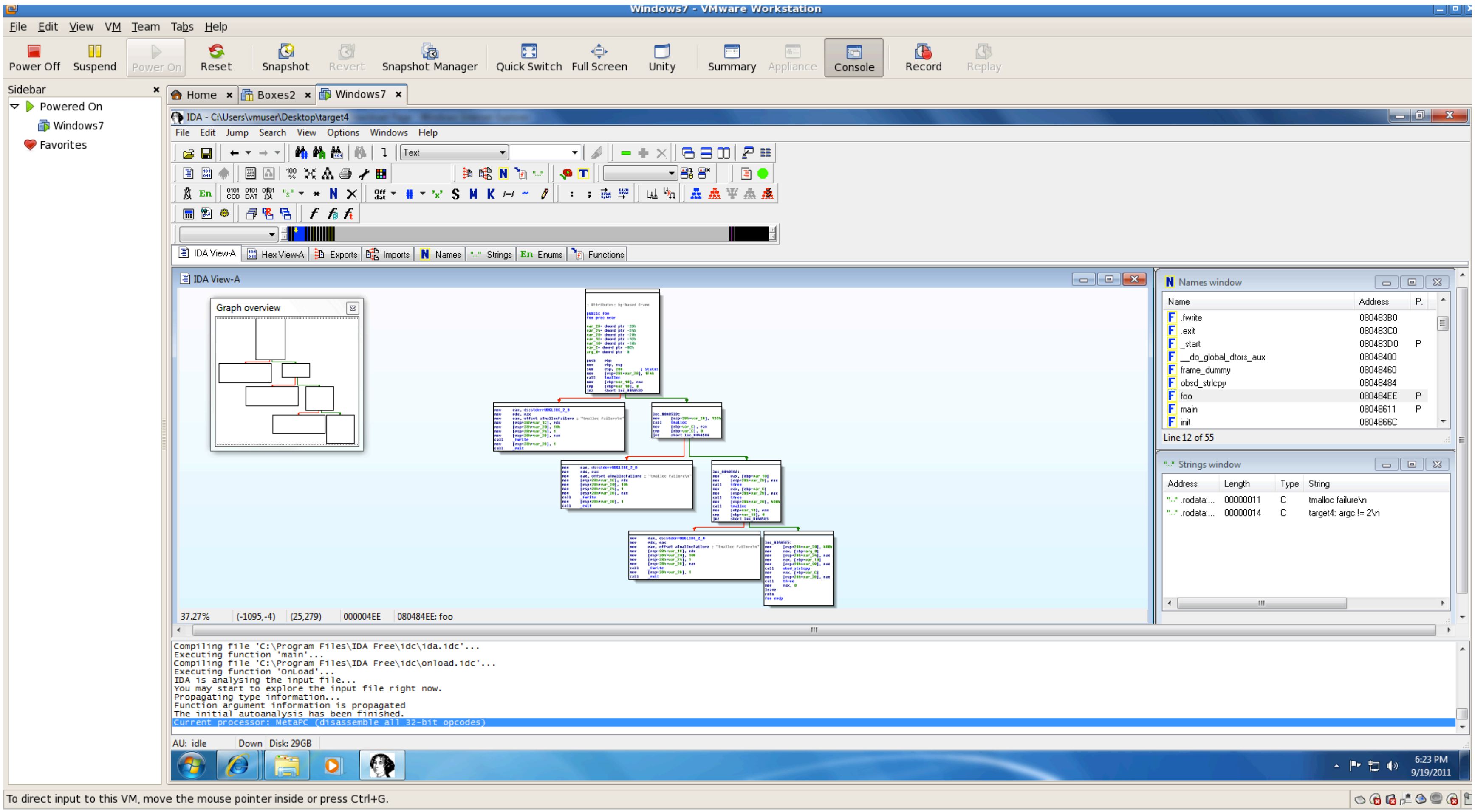
Manual analysis

- You get a binary or the source code
- You find vulnerabilities

IDA Pro



IDA Pro



Windows7 - VMware Workstation
File Edit View VM Team Tabs Help
Power Off
Suspend
Power On
Reset
Snapshot
Revert
Snapshot Manager
Quick Switch
Full Screen
Unity
Summary
Appliance
Console
Record
Replay
Sidebar
Powered On
Windows7
Favorites
Home
Boxes2
Windows7
IDA - C:\Users\vmuser\Desktop\target4
File Edit Jump Search View Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Strings
Enums
Functions
Graph overview
Names window
Name
Address
.fwrite 080483B0
.exit 080483C0
_start 080483D0 P
__do_global_dtors_aux 08048400
frame_dummy 08048460
obsd_strlcpy 08048484
foo 080484EE P
main 08048611 P
init 0804866C
Line 12 of 55
Strings window
Address
Length
Type
String
.rodata:... 00000011 C tmalloc failure\n
.rodata:... 00000014 C target4: argc != 2\n
37.27% (-1095,-4) (25,279) 000004EE 080484EE: foo
Compiling file 'C:\Program Files\IDA Free\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
Current processor: MetaPC (disassemble all 32-bit opcodes)
AU: idle Down Disk: 29GB
6:23 PM
9/19/2011
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```
movl   $0xf8,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x14(%esp)
movl   $0xf8,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x18(%esp)
mov    0x14(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
mov    0x18(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
movl   $0x200,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x1c(%esp)
mov    0xc(%ebp),%eax
add    $0x4,%eax
mov    (%eax),%eax
movl   $0x1ff,0x8(%esp)
mov    %eax,0x4(%esp)
mov    0x1c(%esp),%eax
mov    %eax,(%esp)
call   0x8048334 <strncpy@plt>
mov    0x18(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
mov    0x1c(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
leave
ret
```

What type of vulnerability might this be?

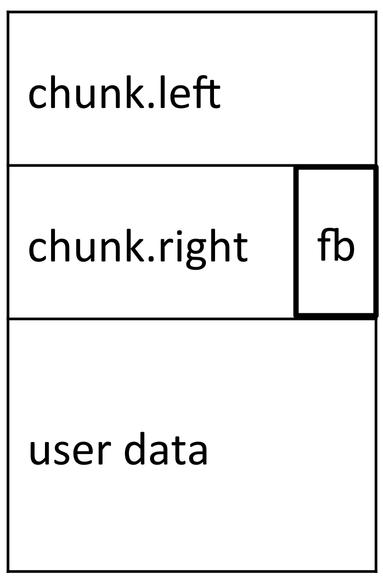
```
main(  int argc, char* argv[] ) {
 char* b1;
 char* b2;
 char* b3;

 b1 = (char*)malloc(248);
 b2 = (char*)malloc(248);
 free(b1);
 free(b2);
 b3 = (char*)malloc(512);
 strncpy( b3, argv[1], 511 );
 free(b2);
 free(b3);
}
```

Double-free vulnerability

Double-free vulnerabilities

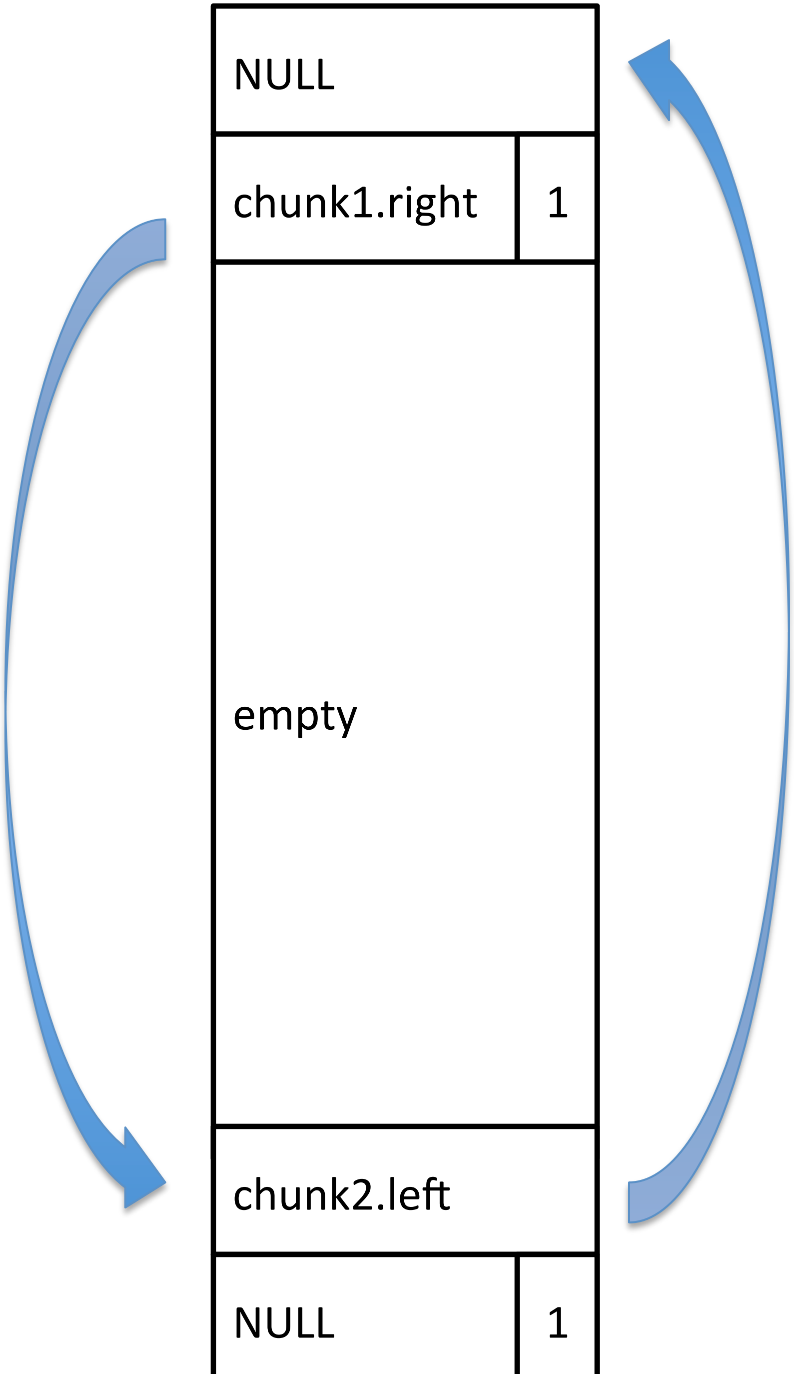
Can corrupt the state of the heap management

Say we use a simple doubly-linked list malloc implementation with control information stored alongside data

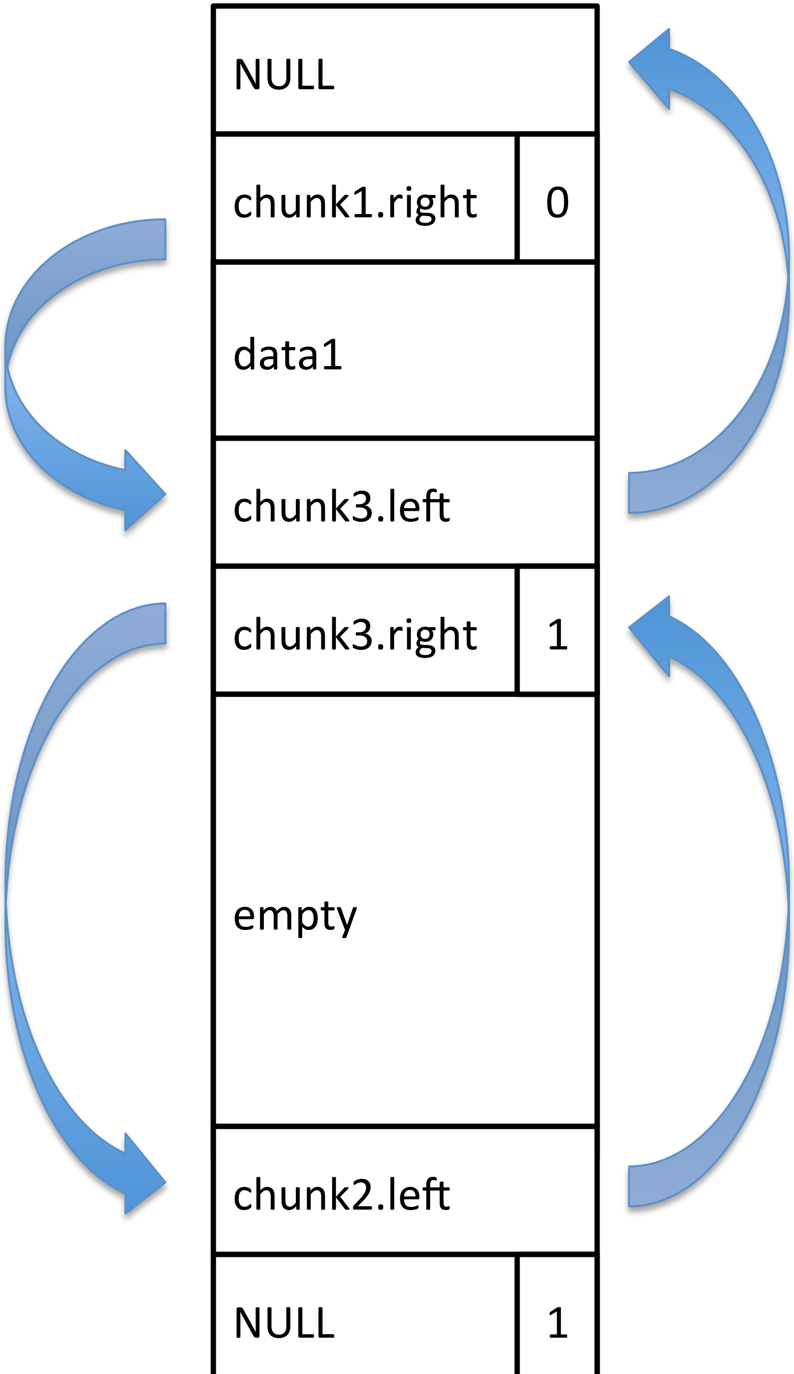


Chunk has:

1) left ptr (to previous chunk)
2) right ptr (to next chunk)
3) free bit which denotes if chunk
   this reuses low bit of right ptr
   because we will align chunks
4) user data

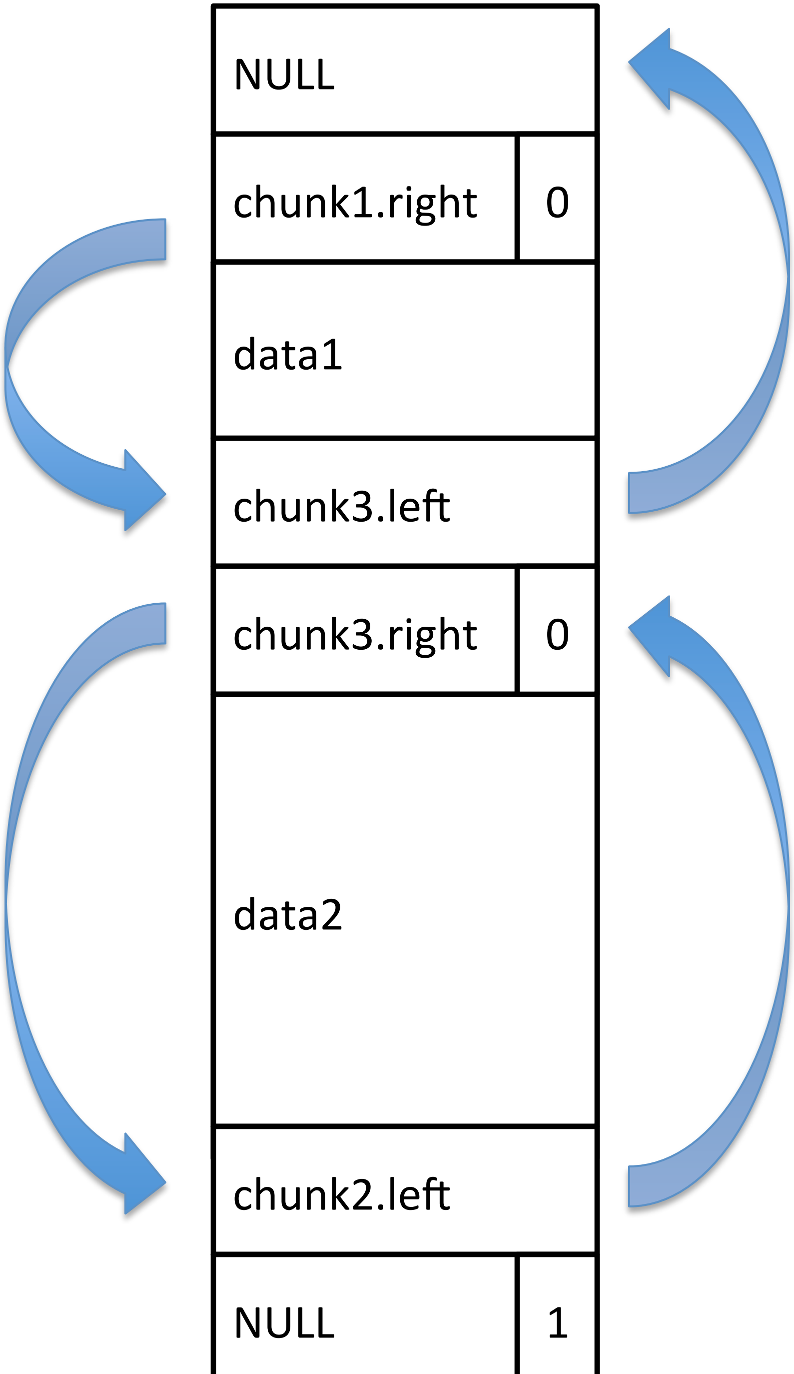


malloc()
- search left-to-right for free chunk
- modify pointers



malloc()
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1);



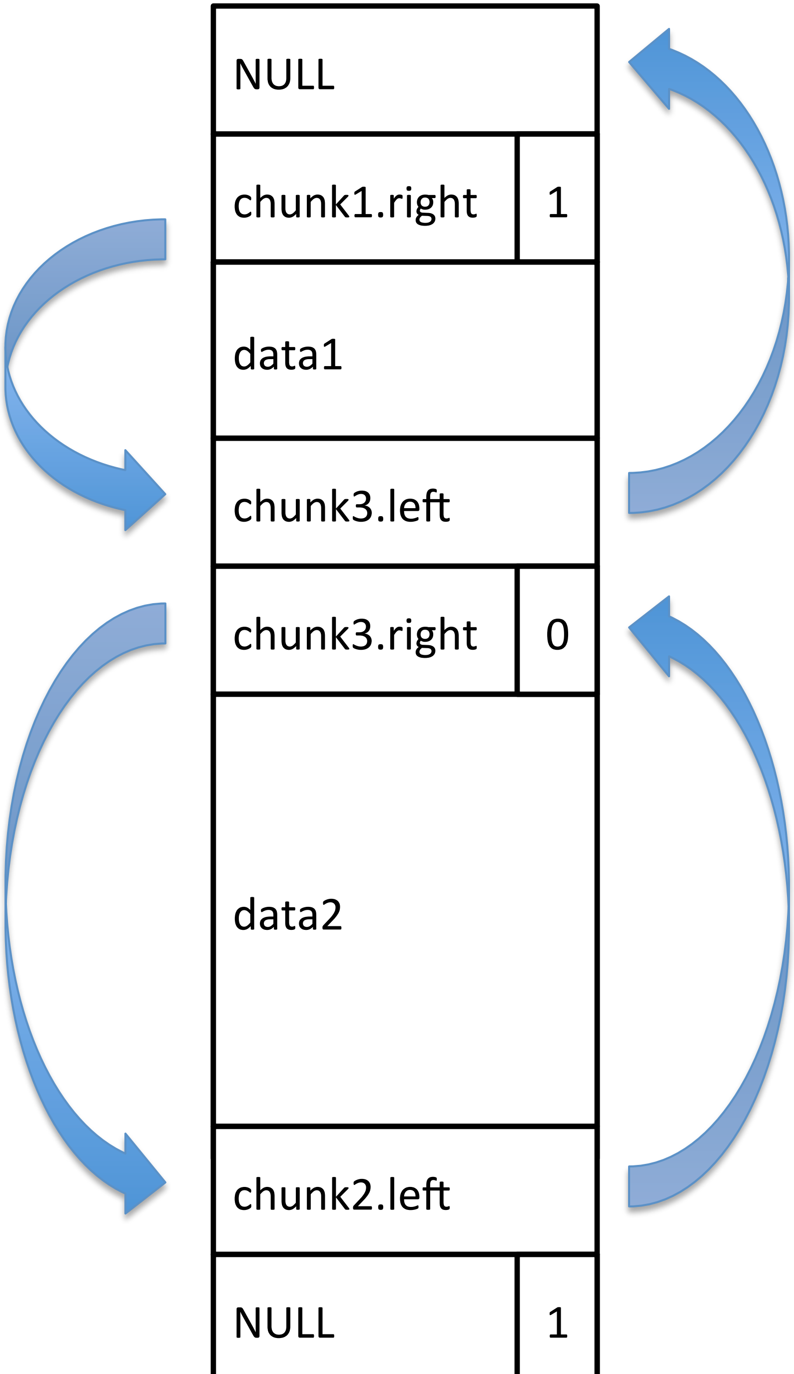
malloc()

- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()

- Consolidate with free neighbors



malloc()

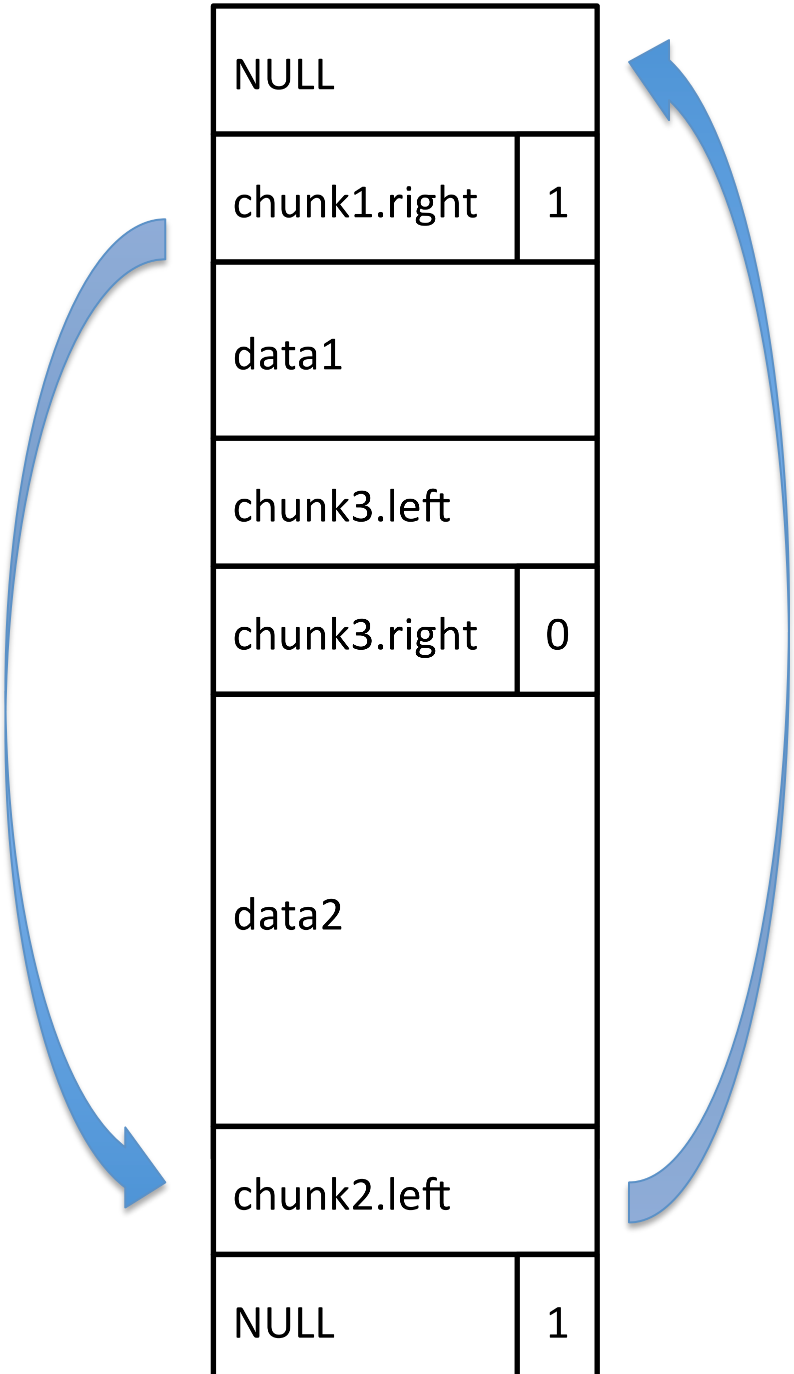
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()

- Consolidate with free neighbors

free(b1)

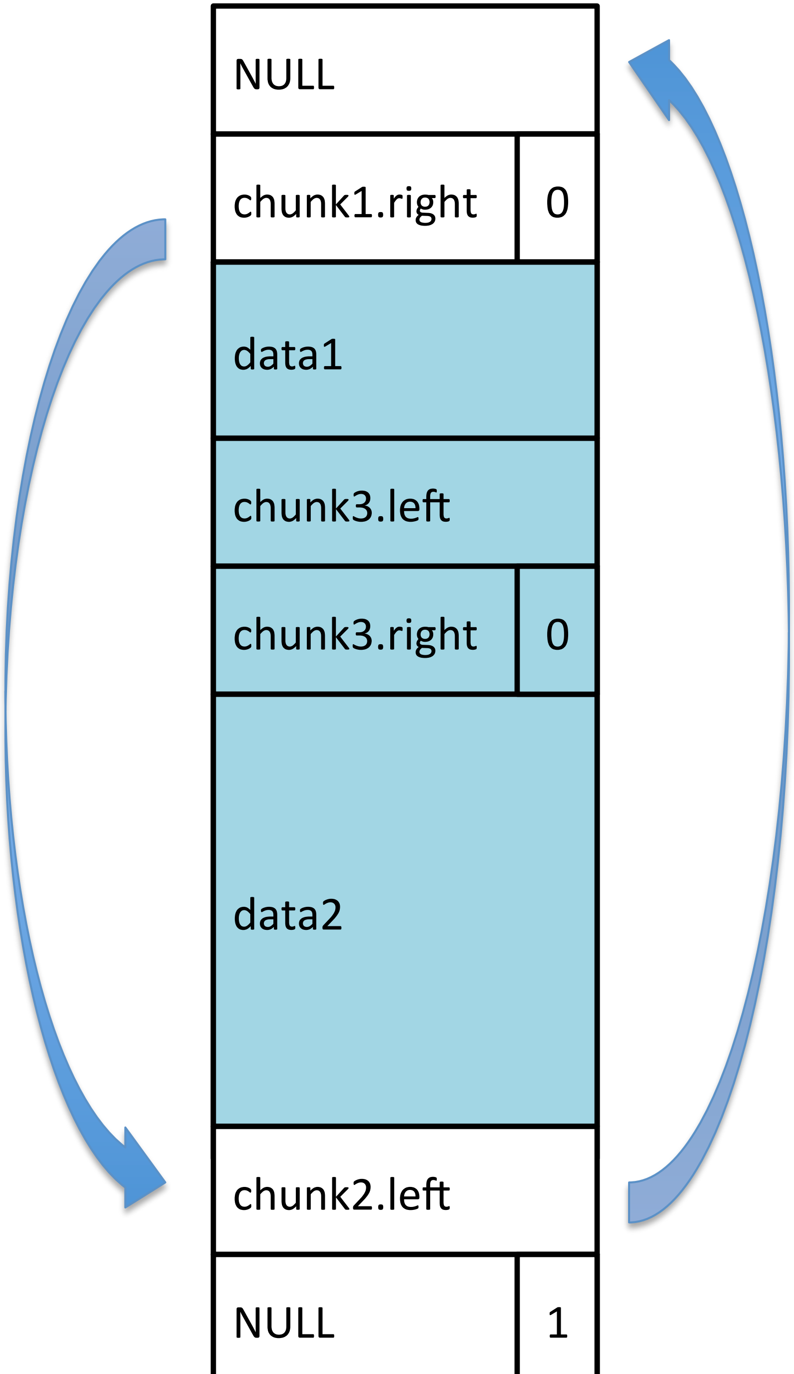


malloc()
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()
- Consolidate with free neighbors

free(b1)
free(b2)

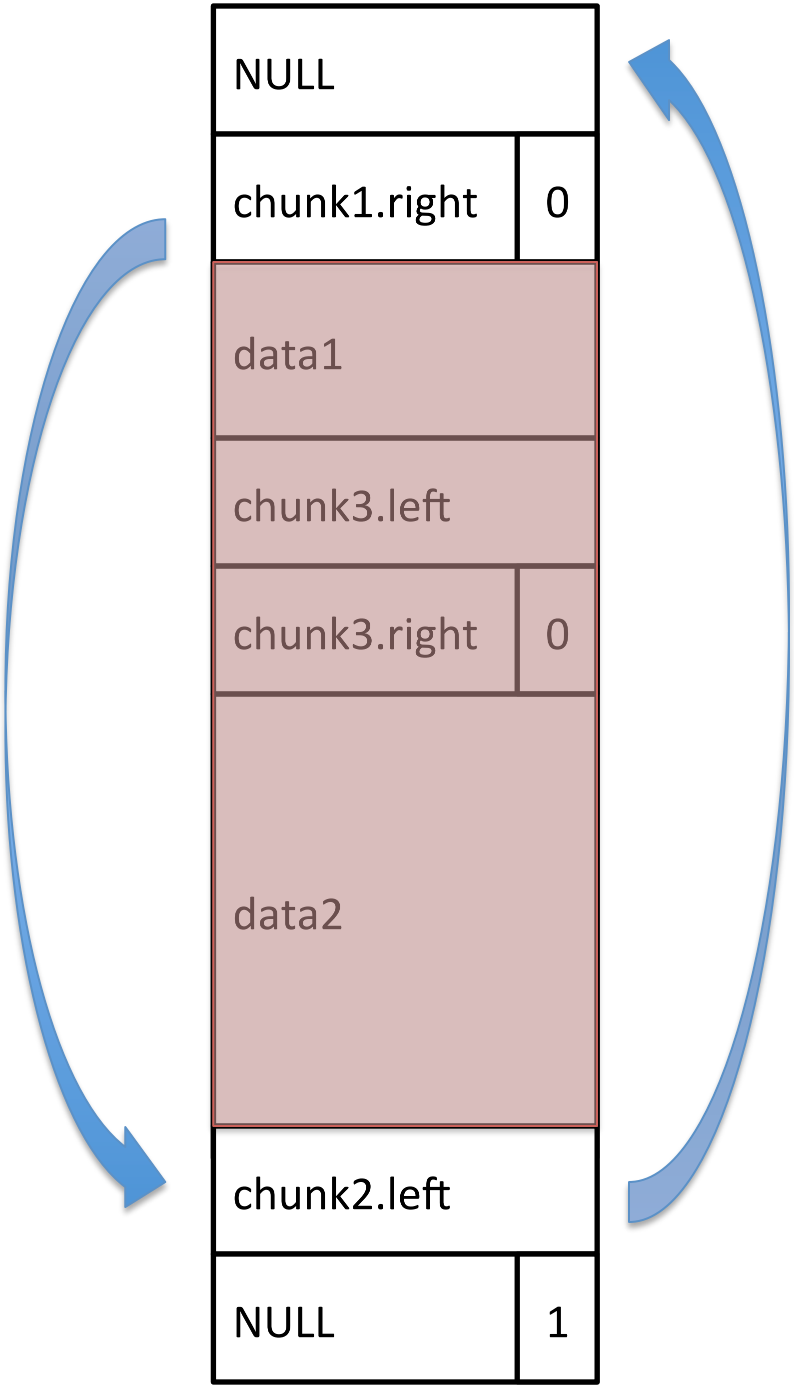


malloc()
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()
- Consolidate with free neighbors

free(b1)
free(b2)
b3 = malloc(BUF_SIZE1 + BUF_SIZE2)

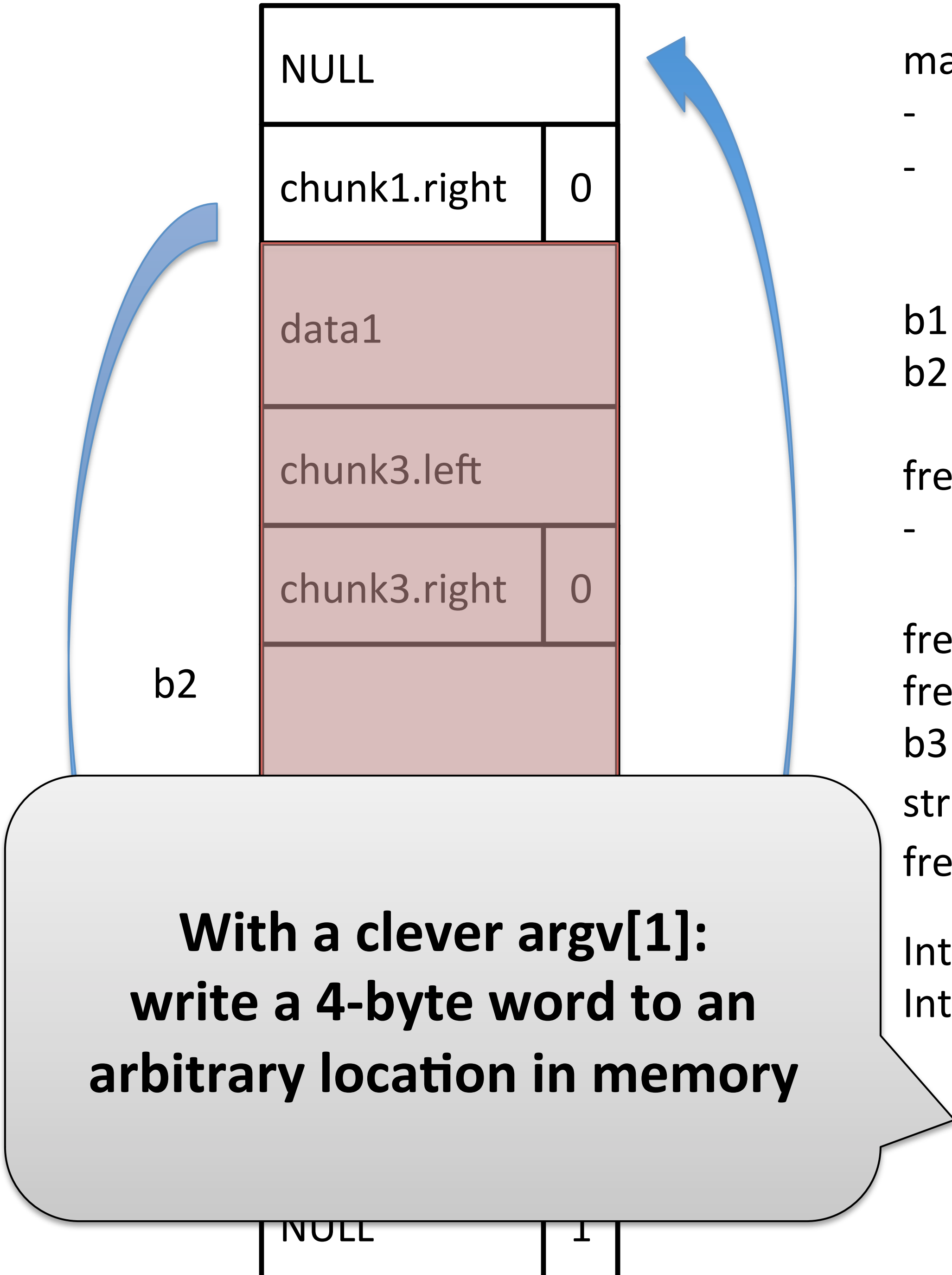


malloc()
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()
- Consolidate with free neighbors

free(b1)
free(b2)
b3 = malloc(BUF_SIZE1 + BUF_SIZE2)
strncpy(b3, argv[1], BUF_SIZE1+BUF_SIZE2-1)



malloc()
- search left-to-right for free chunk
- modify pointers

b1 = malloc(BUF_SIZE1)
b2 = malloc(BUF_SIZE2)

free()
- Consolidate with free neighbors

free(b1)
free(b2)
b3 = malloc(BUF_SIZE1 + BUF_SIZE2)
strncpy(b3, argv[1], BUF_SIZE1+BUF_SIZE2-1)
free(b2)

Interprets b2-8 as a chunk3.left
Interprets b2-4 as a chunk3.right

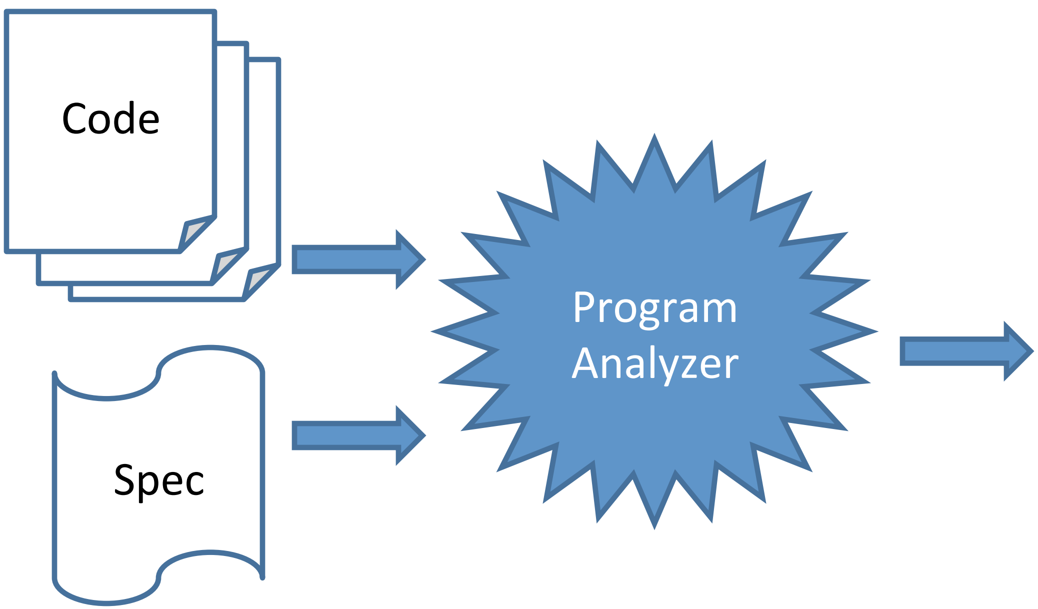
(b2 - 8)->left->right = (b2-8)->right
(b2 - 8)->right->left = (b2-8)->left

```
movl   $0xf8,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x14(%esp)
movl   $0xf8,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x18(%esp)
mov    0x14(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
mov    0x18(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
movl   $0x200,(%esp)
call   0x8048364 <malloc@plt>
mov    %eax,0x1c(%esp)
mov    0xc(%ebp),%eax
add    $0x4,%eax
mov    (%eax),%eax
movl   $0x1ff,0x8(%esp)
mov    %eax,0x4(%esp)
mov    0x1c(%esp),%eax
mov    %eax,(%esp)
call   0x8048334 <strncpy@plt>
mov    0x18(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
mov    0x1c(%esp),%eax
mov    %eax,(%esp)
call   0x8048354 <free@plt>
leave
ret
```

What type of vulnerability might this be?

This is ridiculously simple example. Manual analysis is very time consuming.

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

false alarm (row 2)

false alarm (row 5)

analyze large code bases

potentially reports many warnings

may emit false alarms

Slide credit: Prof Mitchell Stanford's CS 155

Program analyzers

- Static analysis
  – Do not execute program

- Dynamic analysis
  – Execute program on test cases

Soundness, 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"

Slide credit: Prof Mitchell Stanford's CS 155

	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**

Slide credit: Prof Mitchell Stanford's CS 155

Source code scanners

Look at source code, flag suspicious constructs

```
…
strcpy( ptr1, ptr2 );
…
```

Warning: Don't use strcpy

Simplest example: grep

Lint is early example

RATS (Rough auditing tool for security)

ITS4 (It's the Software Stupid Security Scanner)

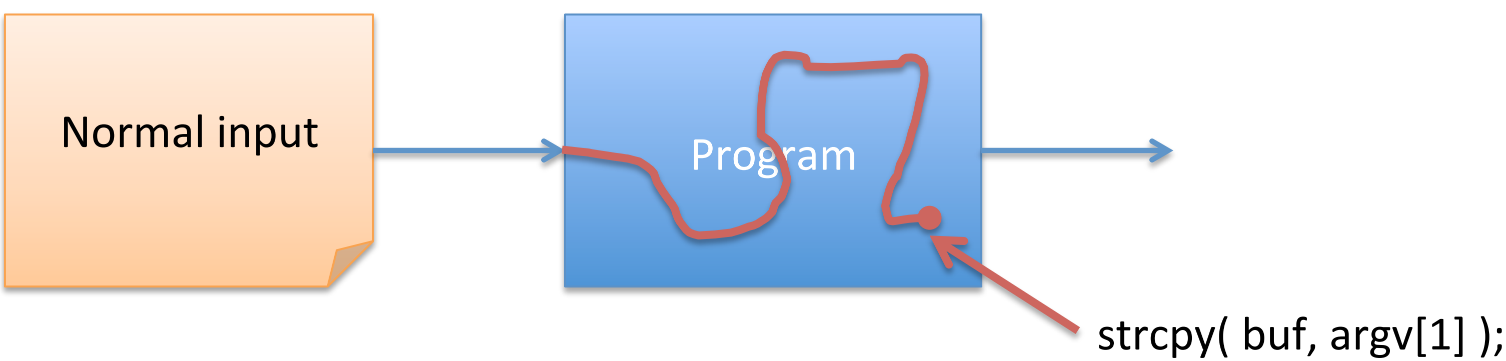
Often look for buffer overflows, race conditions

Taint tracking

Track information flow from user input to it's use

Can be either static or dynamic

Useful to augment manual testing

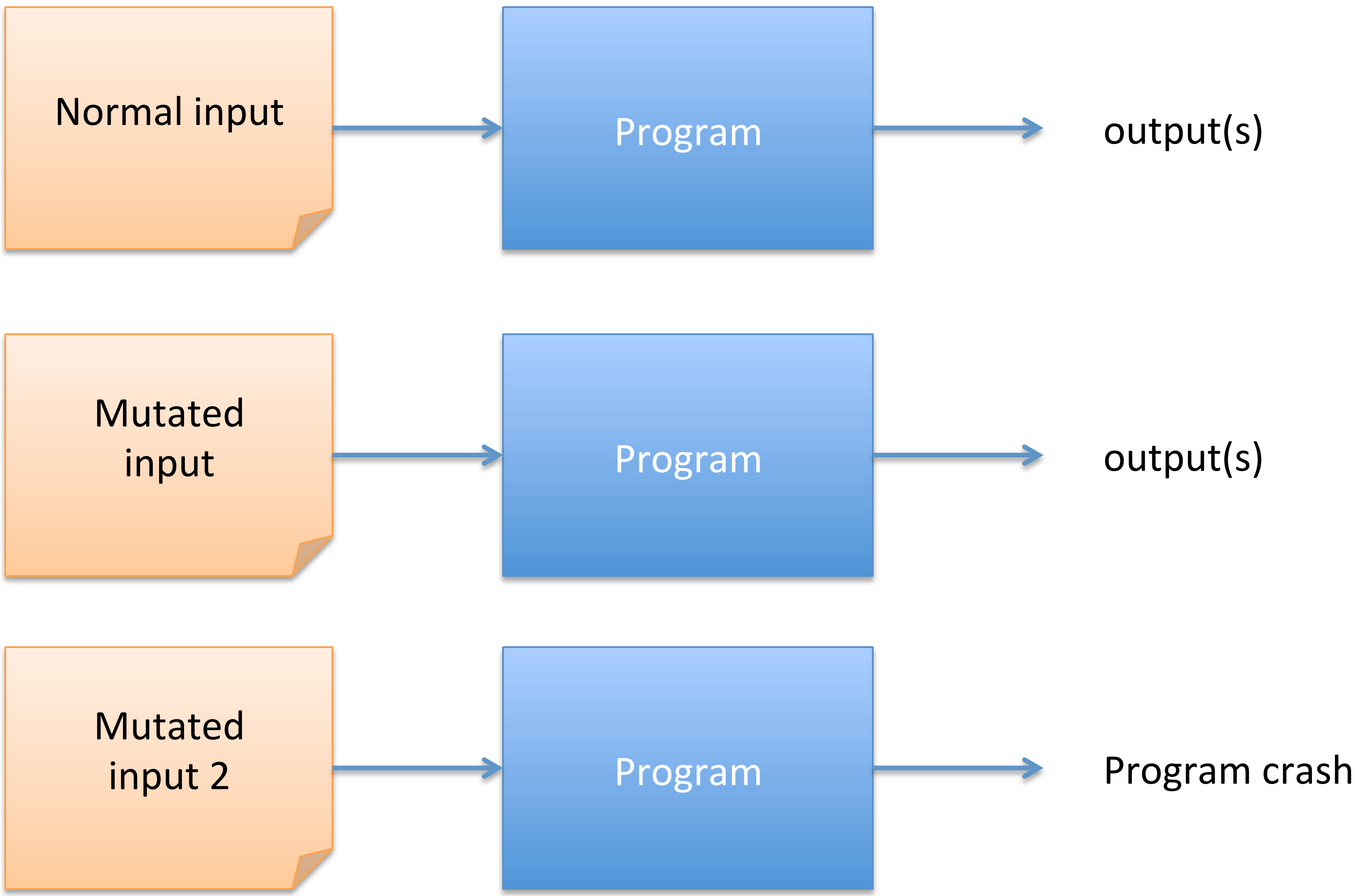


Fuzzing

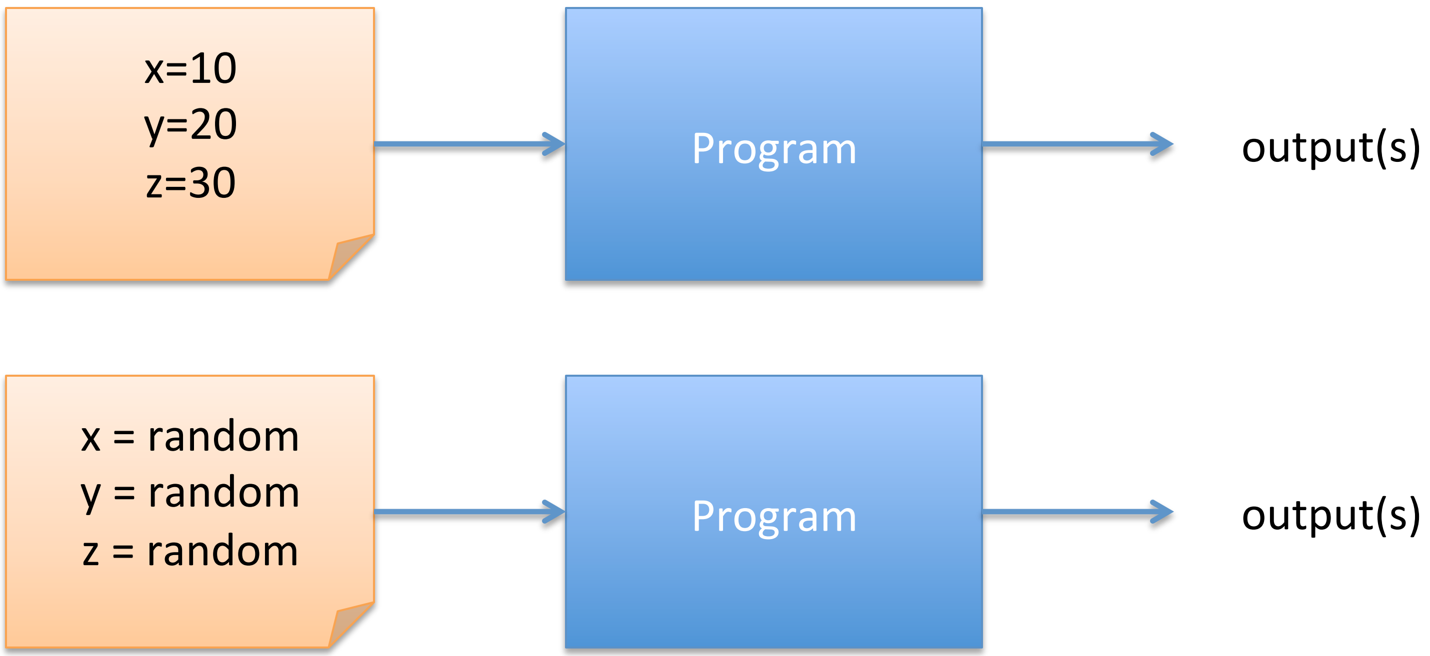
"The term first originates from a class project at the University of Wisconsin 1988 although similar techniques have been used in the field of quality assurance, where they are referred to as robustness testing, syntax testing or negative testing."
Wikipedia
http://en.wikipedia.org/wiki/Fuzz_testing

Choose a bunch of inputs
See if they cause program to misbehave
Example of dynamic analysis

Black-box fuzz testing

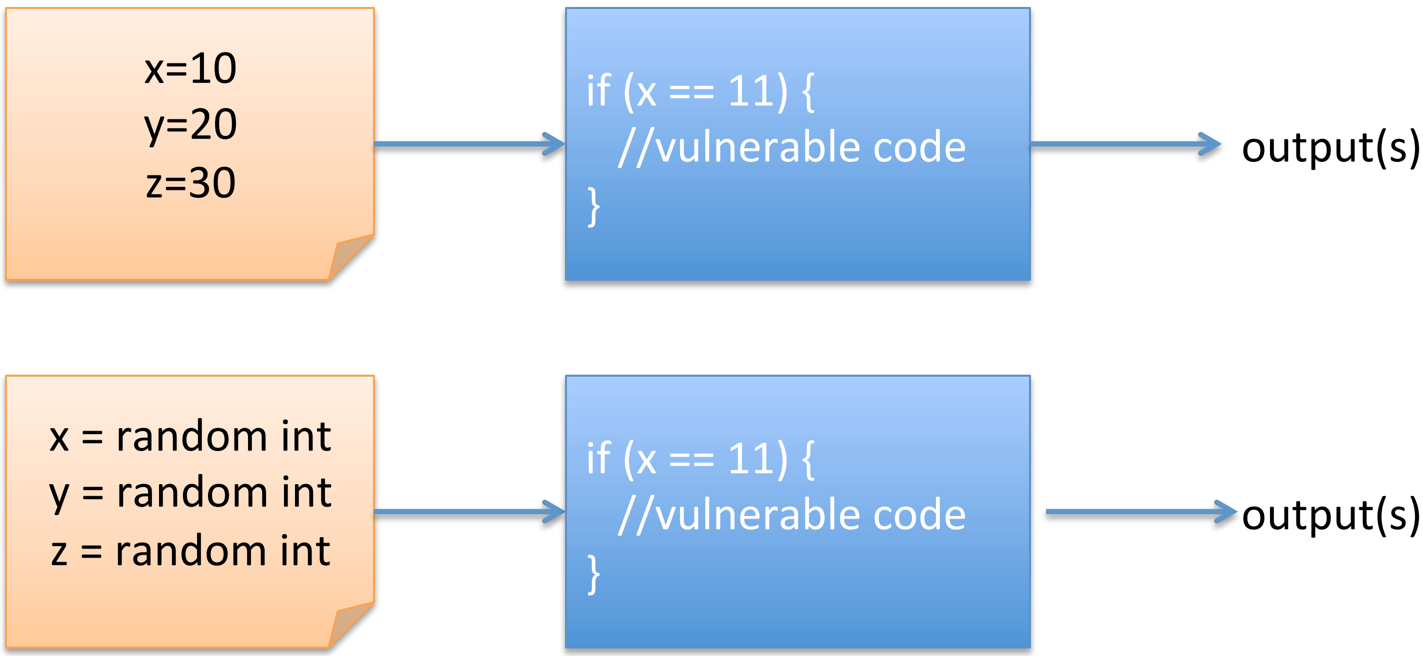


Black-box fuzz testing



Achieving code coverage can be very difficult

Black-box fuzz testing



Achieving code coverage can be very difficult

If x is 32 bits, then probability of crashing is **at most**?

Fuzzing is a lot about code coverage

- How many code paths are exercised
- Mutation based
  - Start with known-good examples
  - Mutate them to new test cases
    - heuristics: increase string lengths (AAAAAAAAA…)
    - randomly change items
- Generative
  - Start with specification of protocol, file format
  - Build test case files from it
    - Rarely used parts of spec

White-box fuzz testing

- Start with real input
  - Symbolic execution of program
  - Gather constraints (control flow) along way
  - Systematically negate constraints backwards
  - Eventually this yields a new input
- Repeat

Godefroid, Levin, Molnar. “Automated Whitebox Fuzz Testing”

Symbolic execution

```
void top(char input[4]) {
  int cnt=0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 3) abort(); // error
}
```

Example from Godefroid et al.

Say input = "good"

i0 != 'b'
i1 != 'a'
i2 != 'd'
i3 != '!'

i0,i1,i2,i3 are all symbolic variables

This gives set of constraints on input
Negate them one at a time.

Example:
i0 != 'b' and i1 != 'a' and i2 != 'd' and i3 = '!'

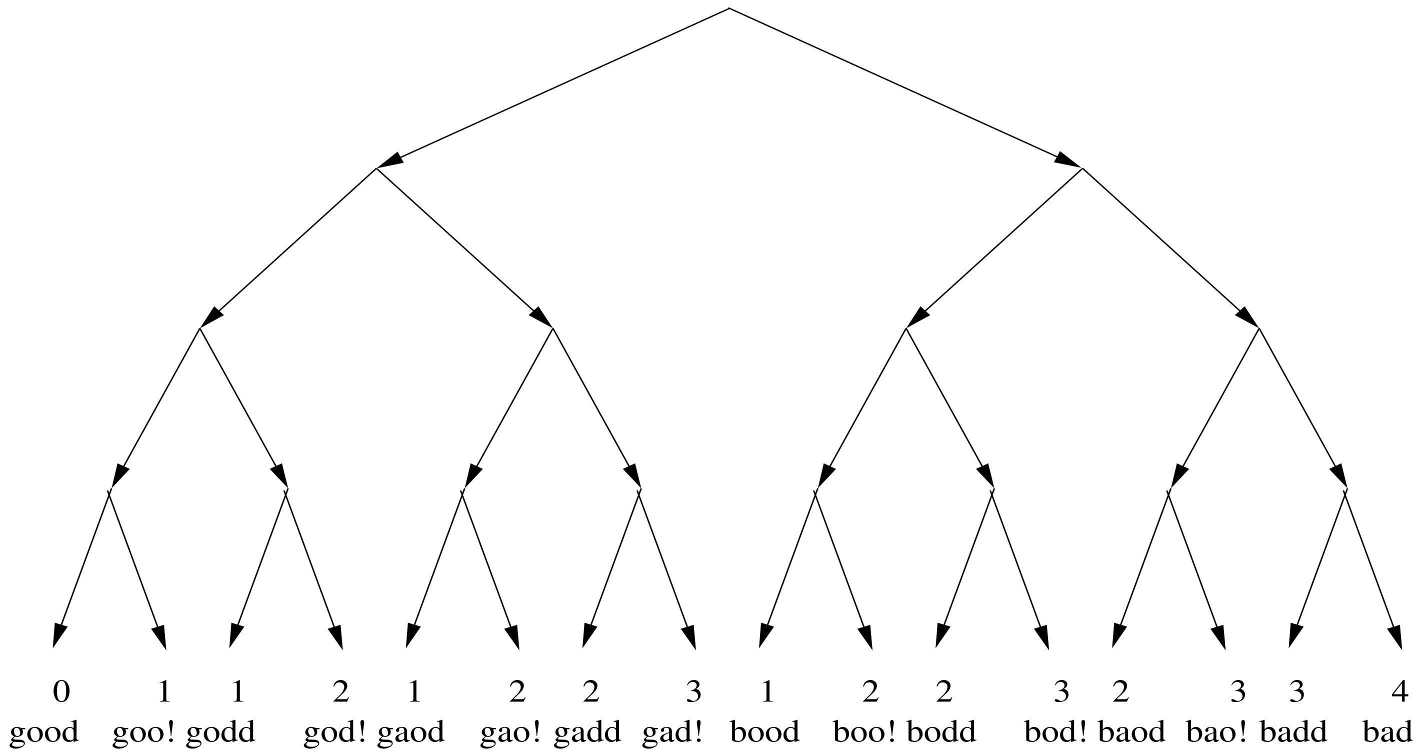


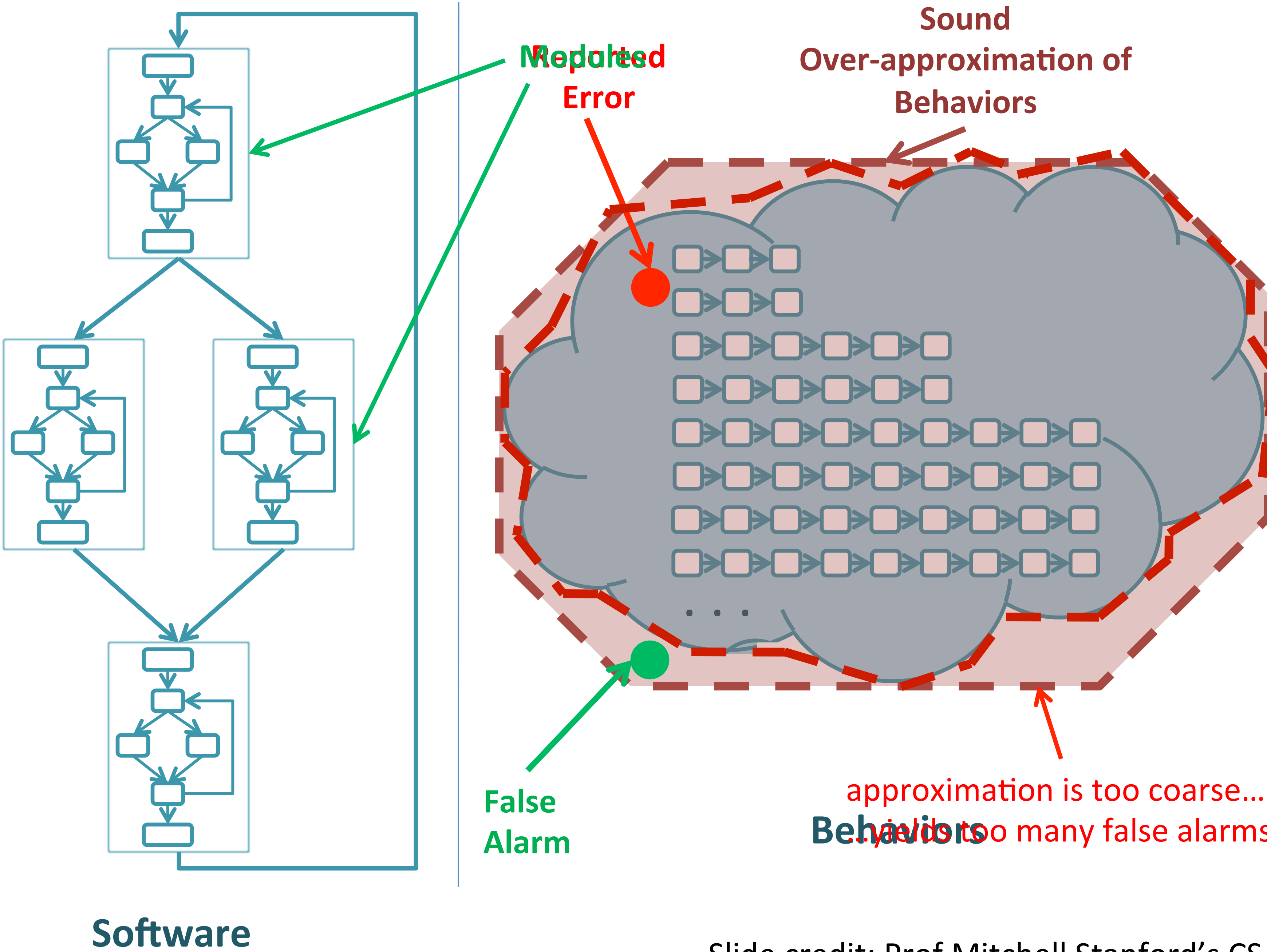
Figure 2. Search space for the example of Figure 1 with the value of the variable `cnt` at the end of each run and the corresponding input string.

Example from Godefroid et al.

Fuzz testing

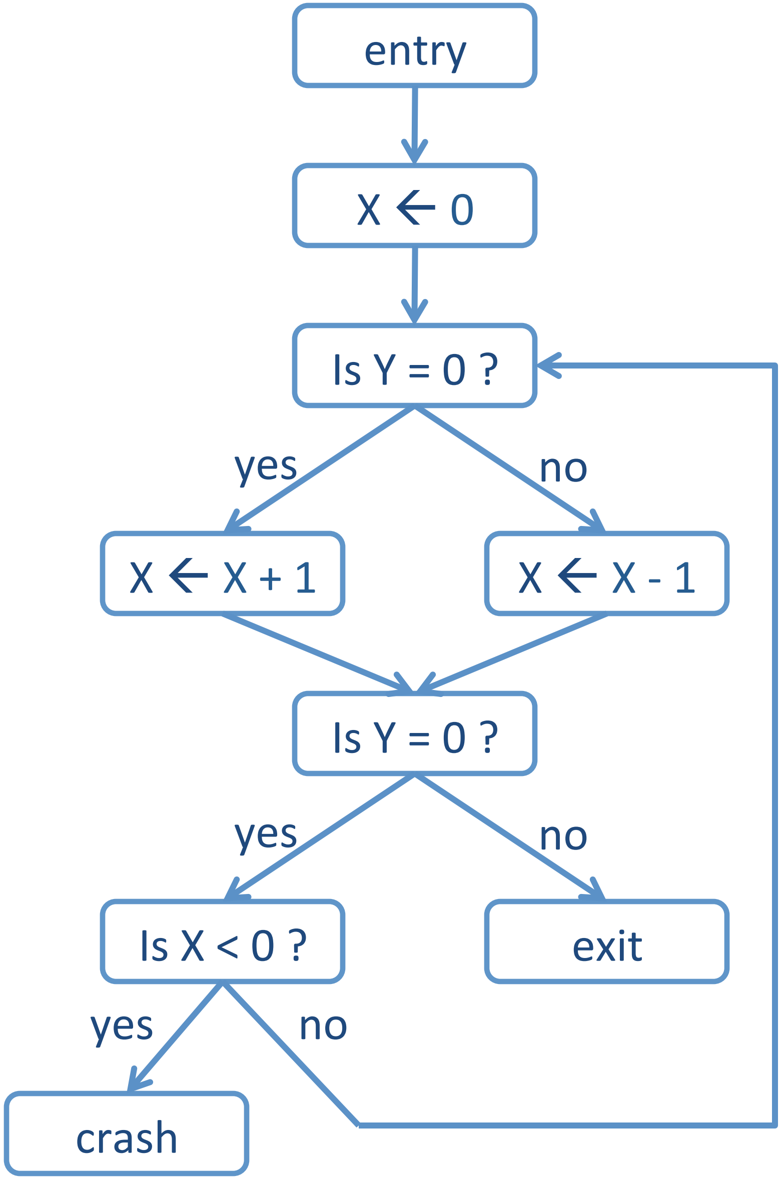
- Black-box is dynamic
- Whitebox is dynamic + static analysis

- Neither sound nor complete



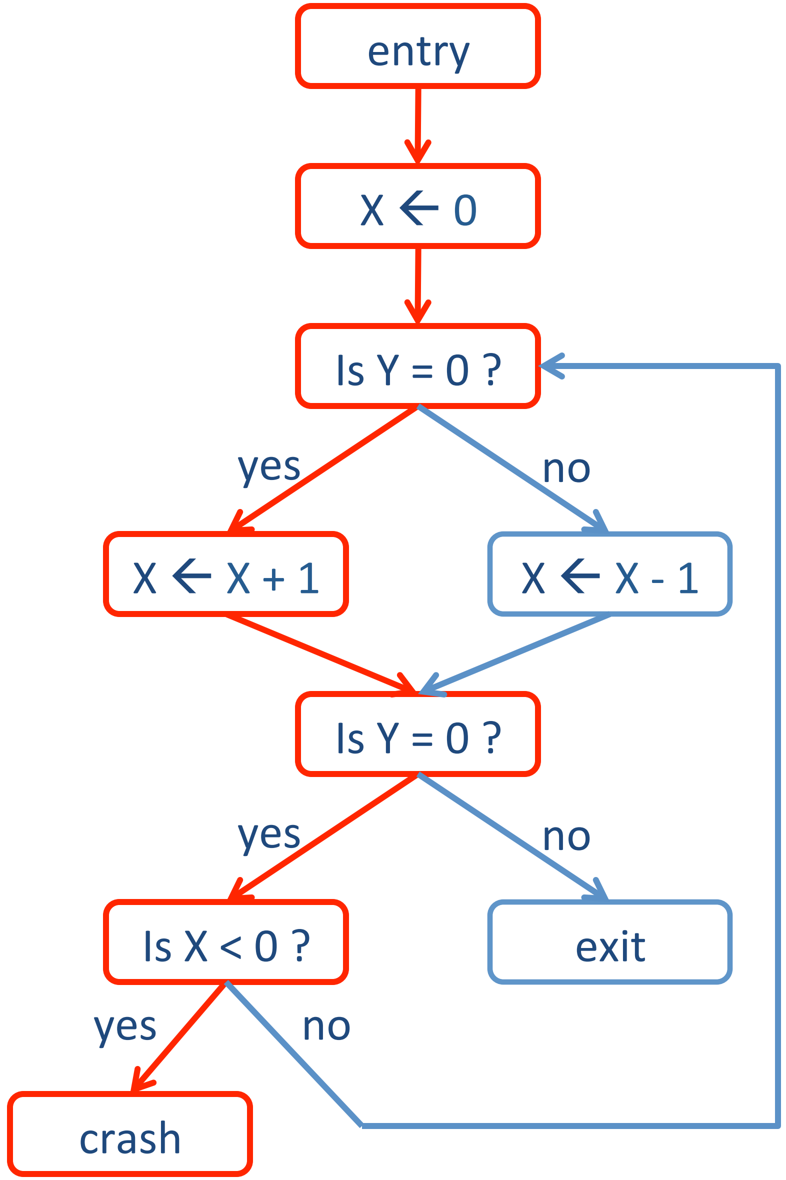
Slide credit: Prof Mitchell Stanford’s CS 155

Does this program ever crash?



Slide credit: Prof Mitchell Stanford's CS 155

Does this program ever crash?

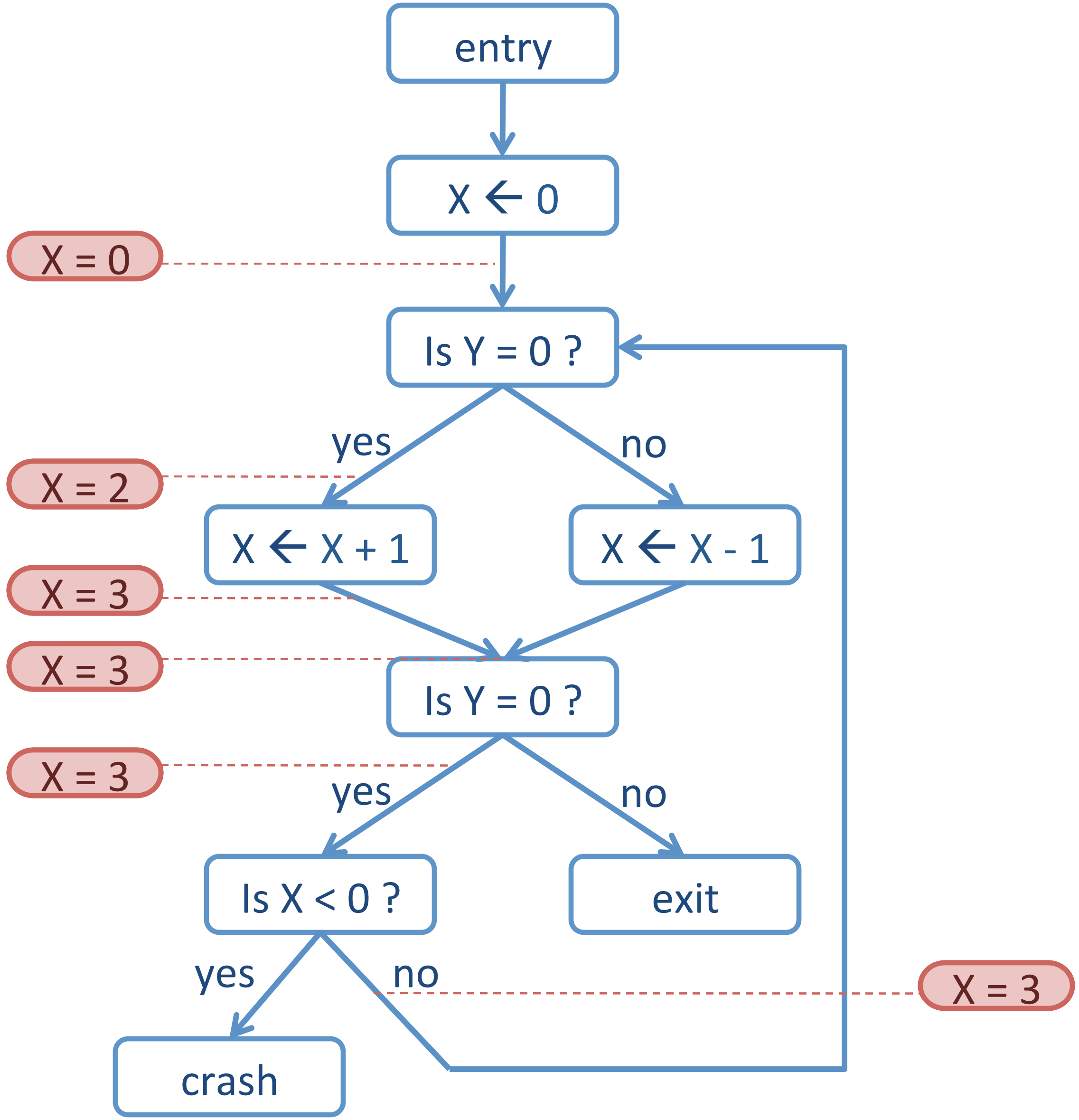


infeasible path!

… program will never crash

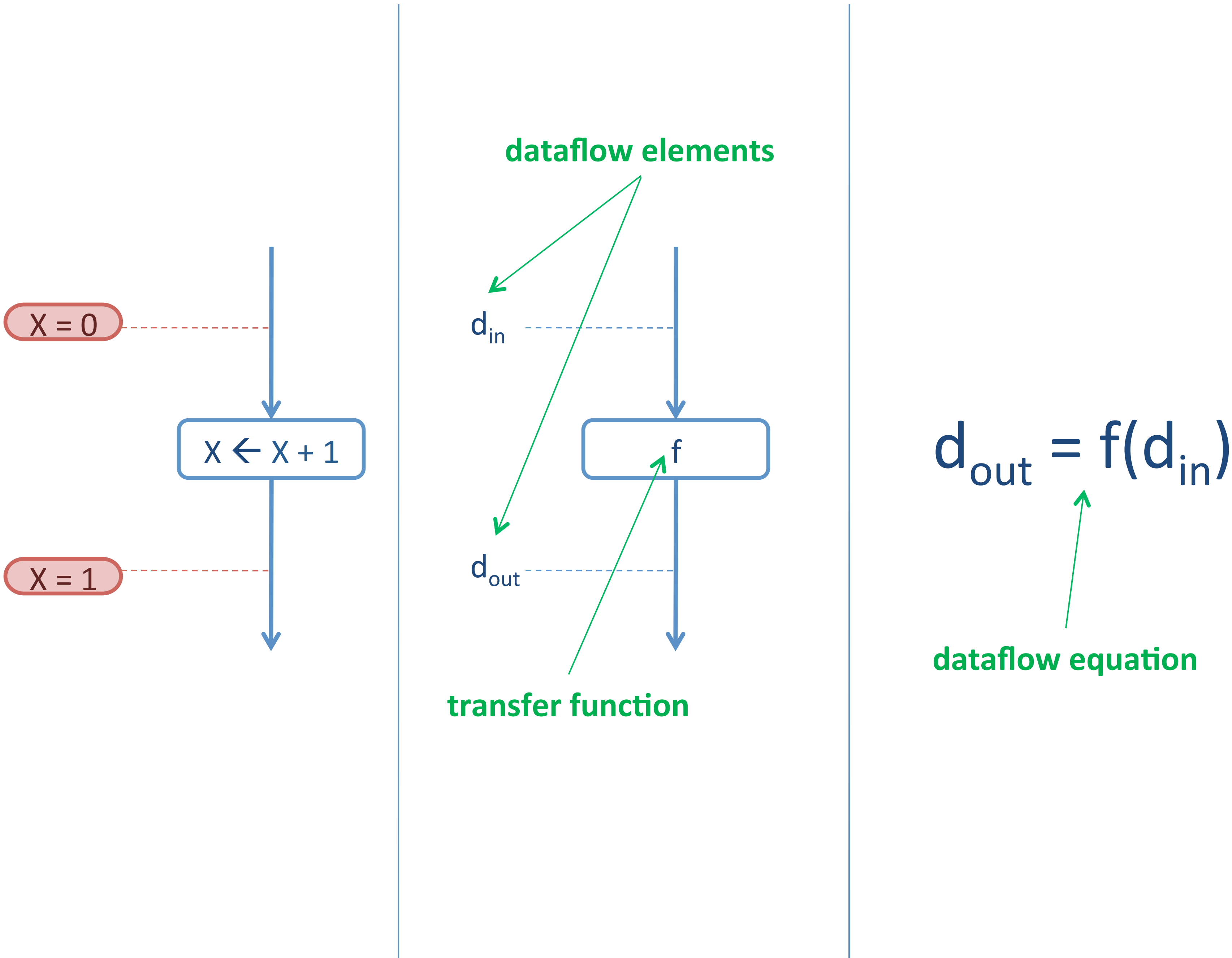
Slide credit: Prof Mitchell Stanford's CS 155

Try analyzing without approximating…

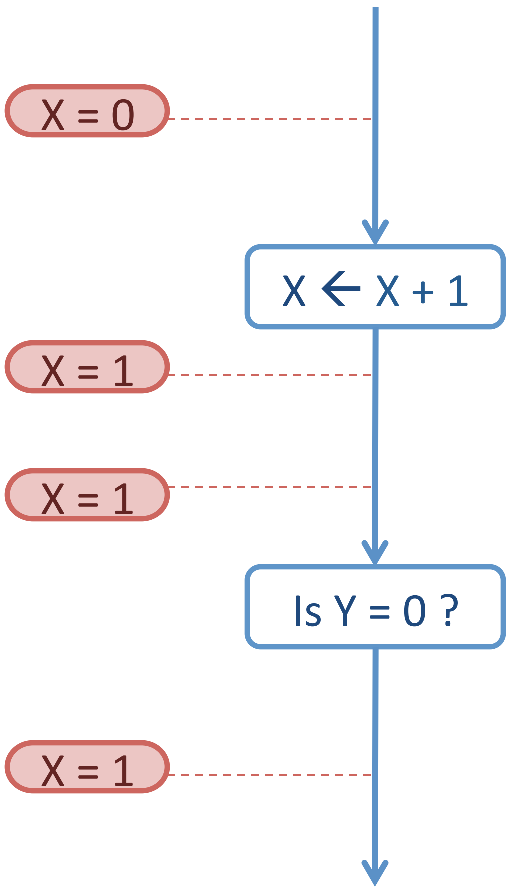


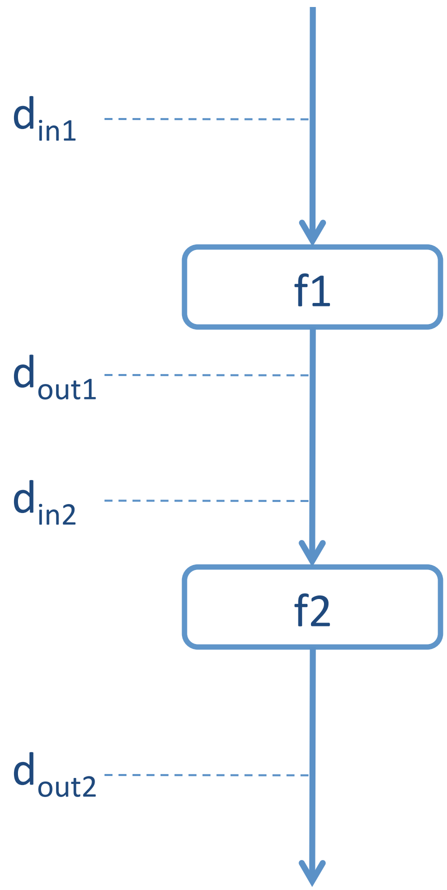
non-termination!
… therefore, need to approximate

Slide credit: Prof Mitchell Stanford's CS 155



Slide credit: Prof Mitchell Stanford's CS 155



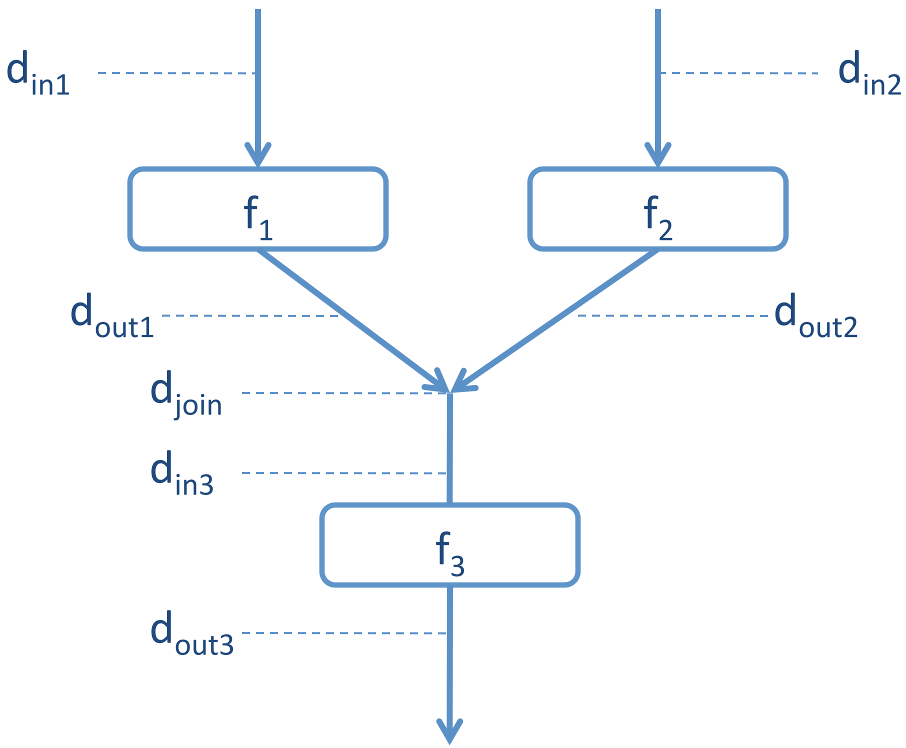


$$d_{out1} = f_1(d_{in1})$$

$$d_{out1} = d_{in2}$$

$$d_{out2} = f_2(d_{in2})$$

Slide credit: Prof Mitchell Stanford's CS 155



$$d_{out1} = f_1(d_{in1})$$

$$d_{out2} = f_2(d_{in2})$$

$$d_{join} = d_{out1} \sqcup d_{out2}$$

$$d_{join} = d_{in3}$$

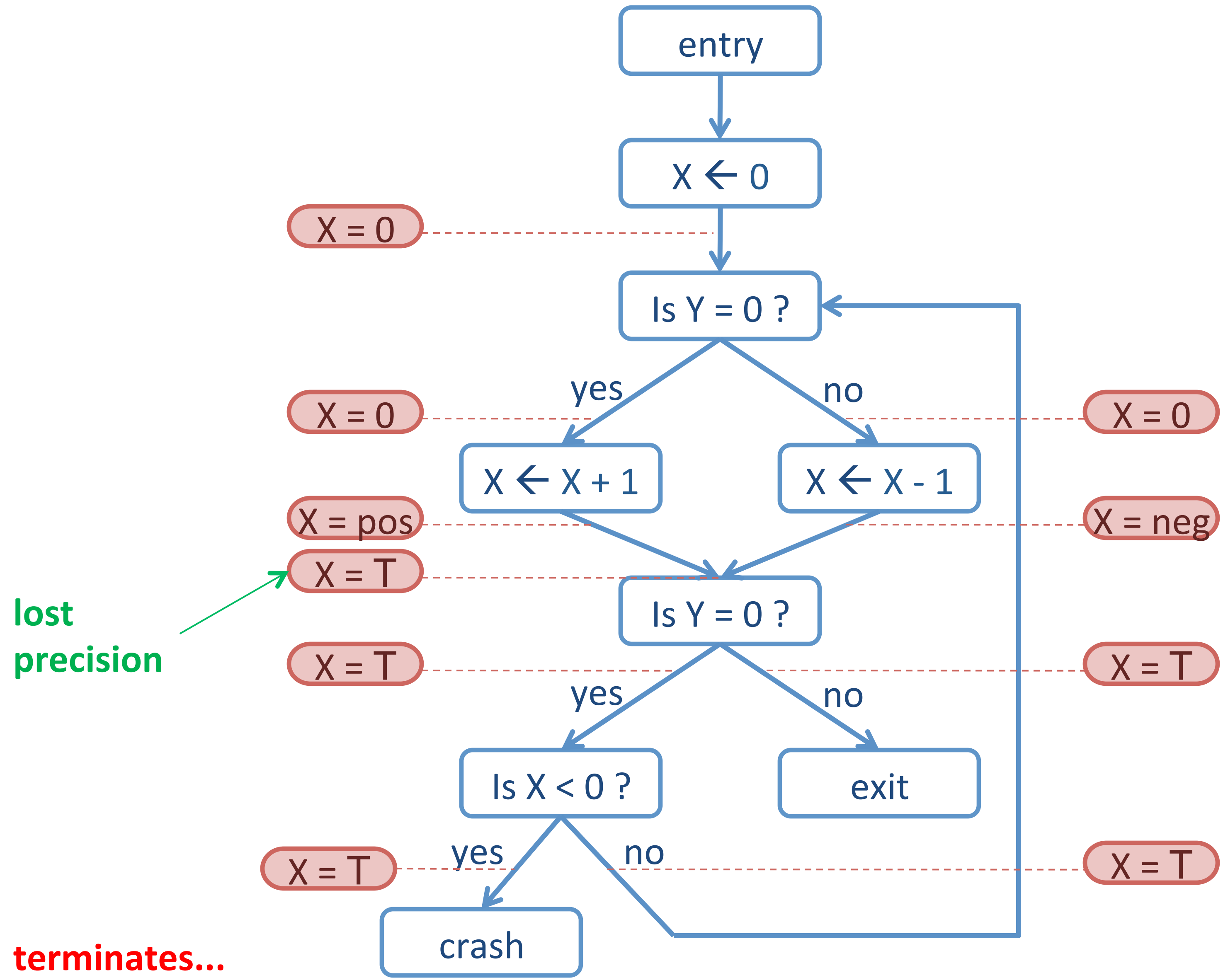
$$d_{out3} = f_3(d_{in3})$$

least upper bound operator
Example: union of possible values

What is the space of dataflow elements, Δ?
What is the least upper bound operator, $\sqcup$?

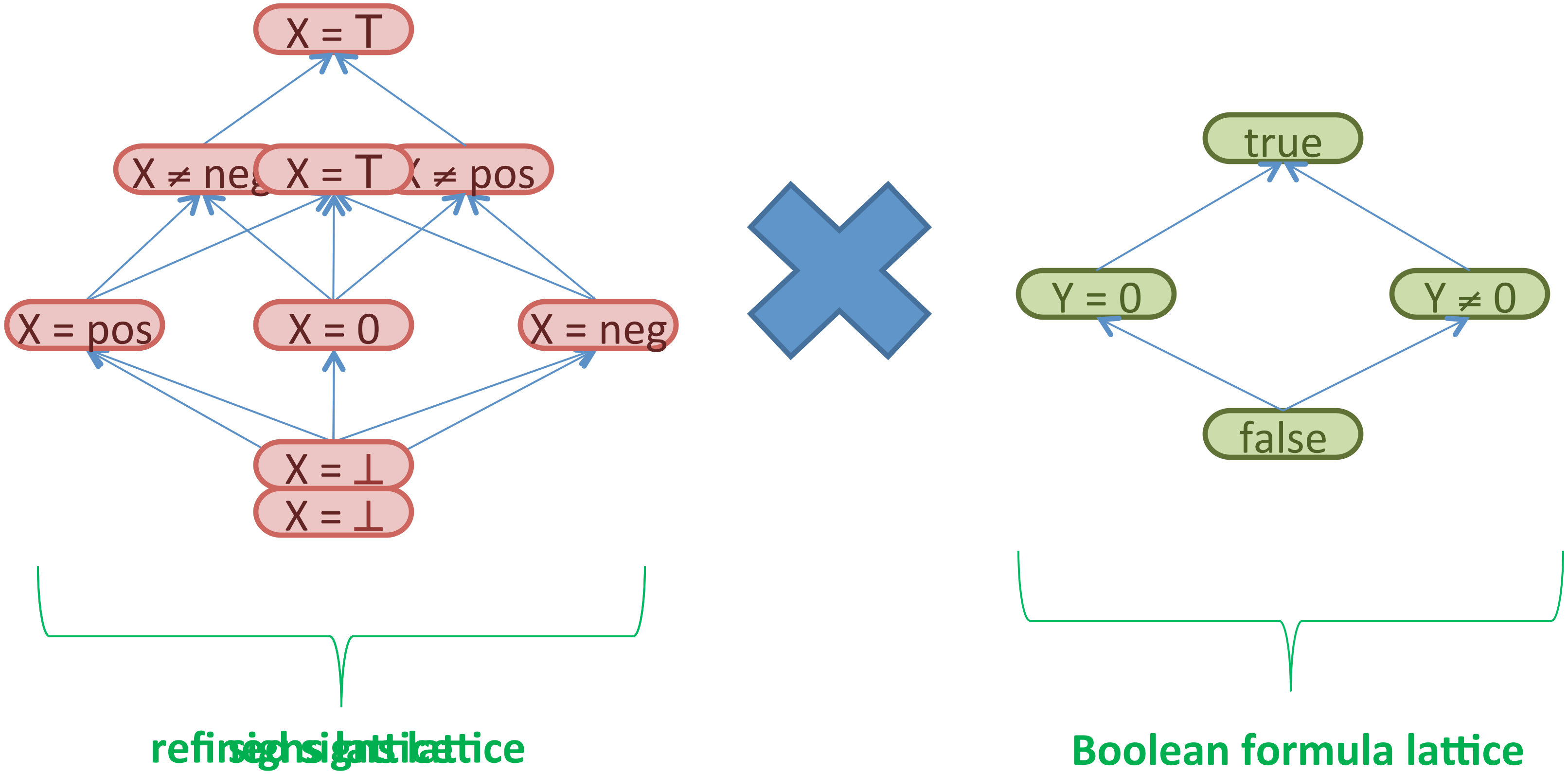
Slide credit: Prof Mitchell Stanford's CS 155

Try analyzing with "signs" approximation…



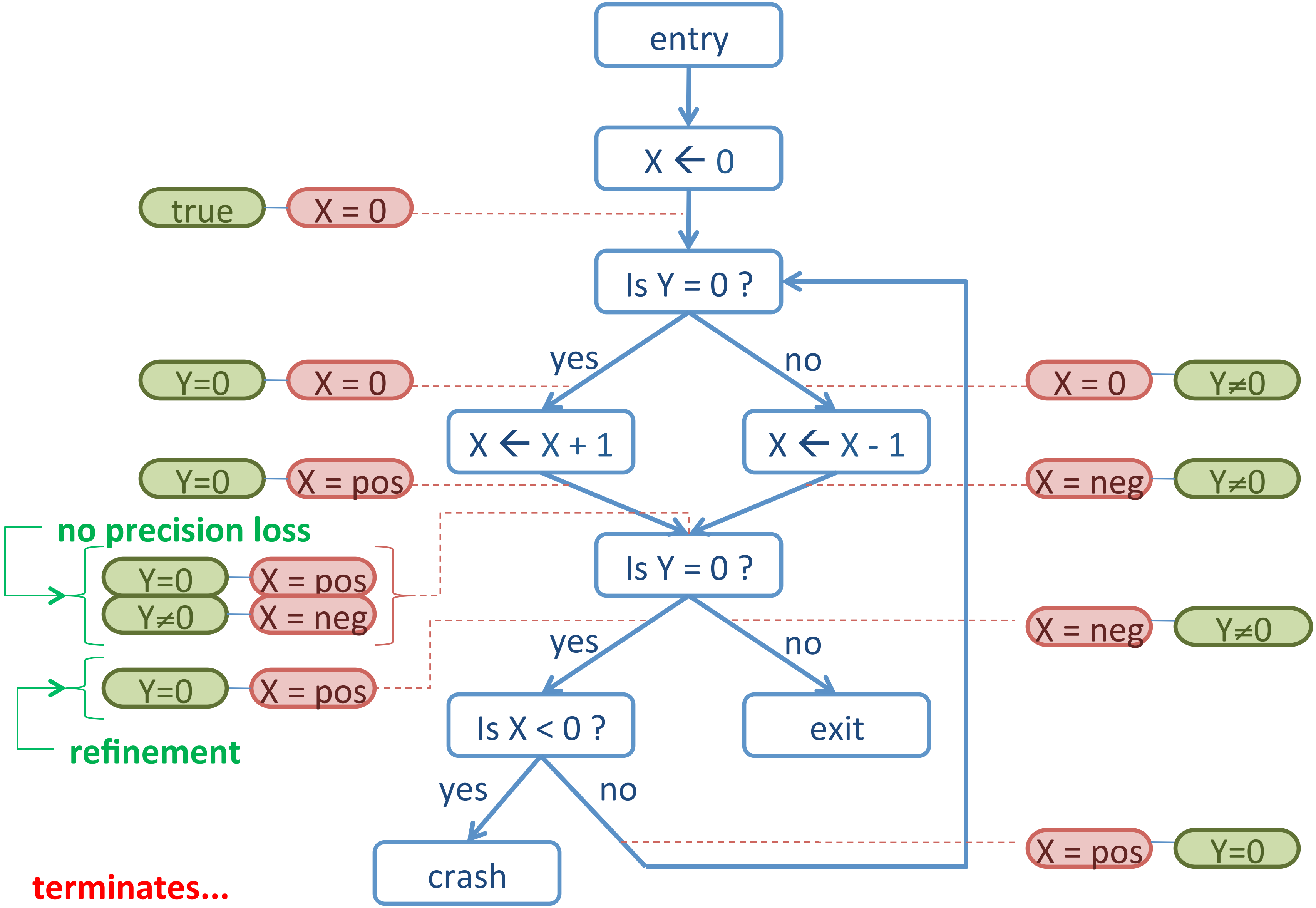
terminates...
… but reports false alarm
… therefore, need more precision

Slide credit: Prof Mitchell Stanford's CS 155



Slide credit: Prof Mitchell Stanford’s CS 155

Try analyzing with "path-sensitive signs" approximation...



terminates...

... no false alarm

... soundly proved never crashes

Slide credit: Prof Mitchell Stanford's CS 155

Coverity

- One of the big names nowadays in static analysis
- Lots of customers (including ones here at Wisconsin)
- One key insight:
  - Statistical bug finding

How Bug Bounties Are Like Rat Farming

Posted by **timothy** on Tuesday September 20, @11:00AM
from the first-we-hypothesize-a-problem dept.

Gunkerty Jeb writes

> "In a keynote speech at the United Security Summit, Stephen Dubner, co-author of *Freakonomics*, drew parallels between the increasingly popular (and successful) practice of software vendors offering bug bounties and a new industry springing up in Johannesburg, South Africa, where the population has recently found itself beset with a growing rat problem. In order to help mitigate their rodent problem, officials in Johannesburg began offering a small monetary rewards for each dead rat turned in. It was wildly successful, and it didn't take long for fresh batch of entrepreneurs to pop up and exploit the situation. Of course, I'm talking about rat farming. Evidently, business minded individuals have taken to breeding rats, only to kill them and turn them in for rewards. Obviously, rat farming is somewhat unscrupulous, but security researchers are doing the same thing: breeding bugs in the lab, then leading them to the slaughter for a nice payday. And it's a good thing."

Read the **104** comments

security software incentives

Tales in insecurity…

"The most critical servers contain malicious software that can normally be detected by anti-virus software," it says. "The separation of critical components was not functioning or was not in place. We have strong indications that the CA-servers, although physically very securely placed in a tempest proof environment, were accessible over the network from the management LAN."

All CA servers were members of one Windows domain and all accessible with one user/ password combination. Moreover, the used password was simple and susceptible to brute-force attacks.

http://www.net-security.org/secworld.php?id=11570

DigiNotar