Program Analysis for Security

Original slides created by Prof. John Mitchell


Facebook missed a single security check…
Man Finds Easy Hack to Delete Any Facebook Photo Album
Facebook awards him a $12,500 "bug bounty" for his discovery
[PopPhoto.com Feb 10]

App stores

Apps for whatever you're up for.

Stay on top of the news. Stay on top of your finances. Or plan your dream vacation. No matter what you want to do with your iPhone, there's probably an app to help you do it.

Business

iPhone is ready for work. Manage projects, track stocks, monitor finances, and more with these 9-to-5 apps.

View business apps in the App Store ›

Education

Keep up with your studies using intelligent education apps like King of Math and NatureTap.

View education apps in the App Store ›

Entertainment

Kick back and enjoy the show. Or find countless other ways to entertain yourself. These apps offer hours of viewing pleasure.

View entertainment apps in the App Store ›

Family & Kids

Turn every night into family night with interactive apps that are fun for the whole house.

View family and kids apps in the App Store ›

Finance

Create budgets, pay bills, and more with financial apps that take everything into account.

View finance apps in the App Store ›

Food & Drink

Hungry? Thirsty? A little of both? Learn new recipes, drinks, and the secrets behind what makes a great meal.

View food and drink apps in the App Store ›

How can you tell whether software you

- Develop
- Buy

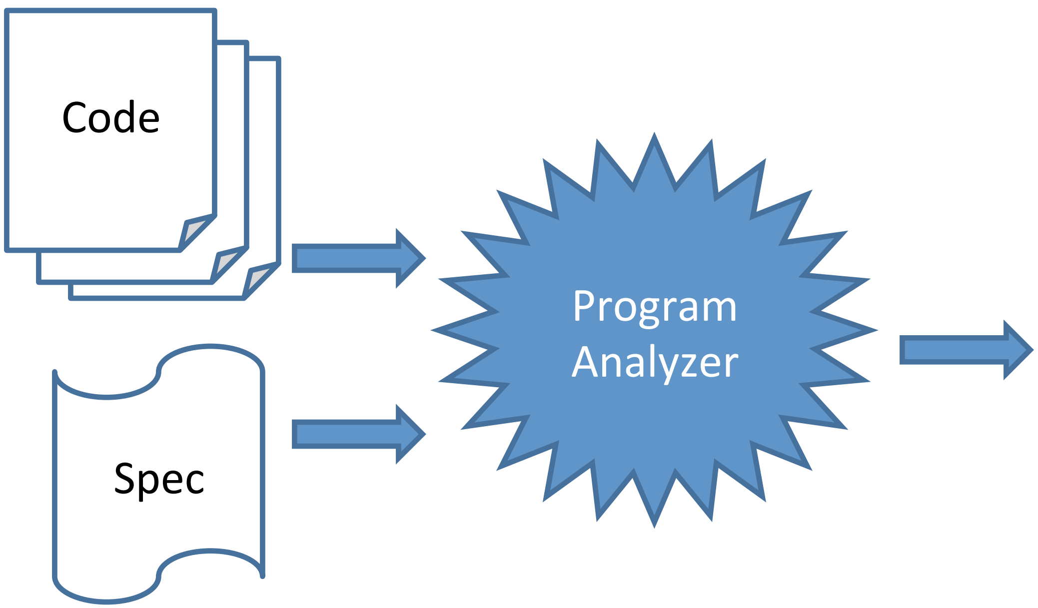
is safe to install and run?

Two options

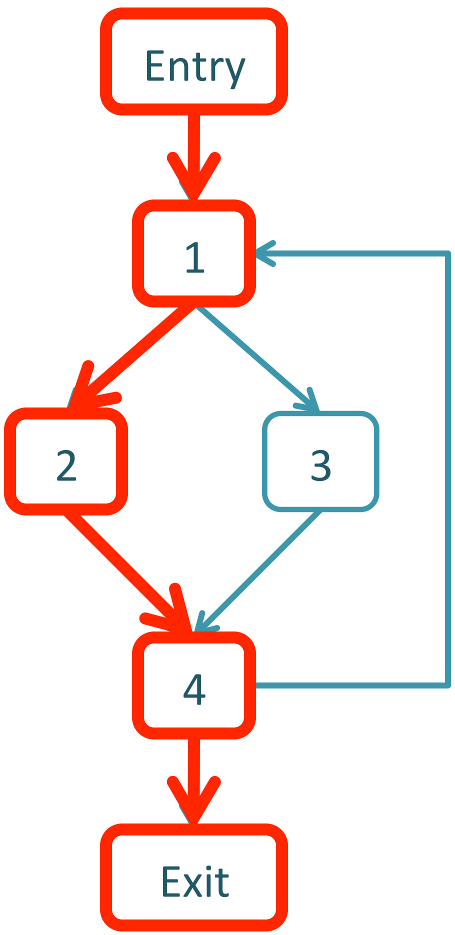
- Static analysis
  – Inspect code or run automated method to find errors or gain confidence about their absence

- Dynamic analysis
  – Run code, possibly under instrumented conditions, to see if there are likely problems

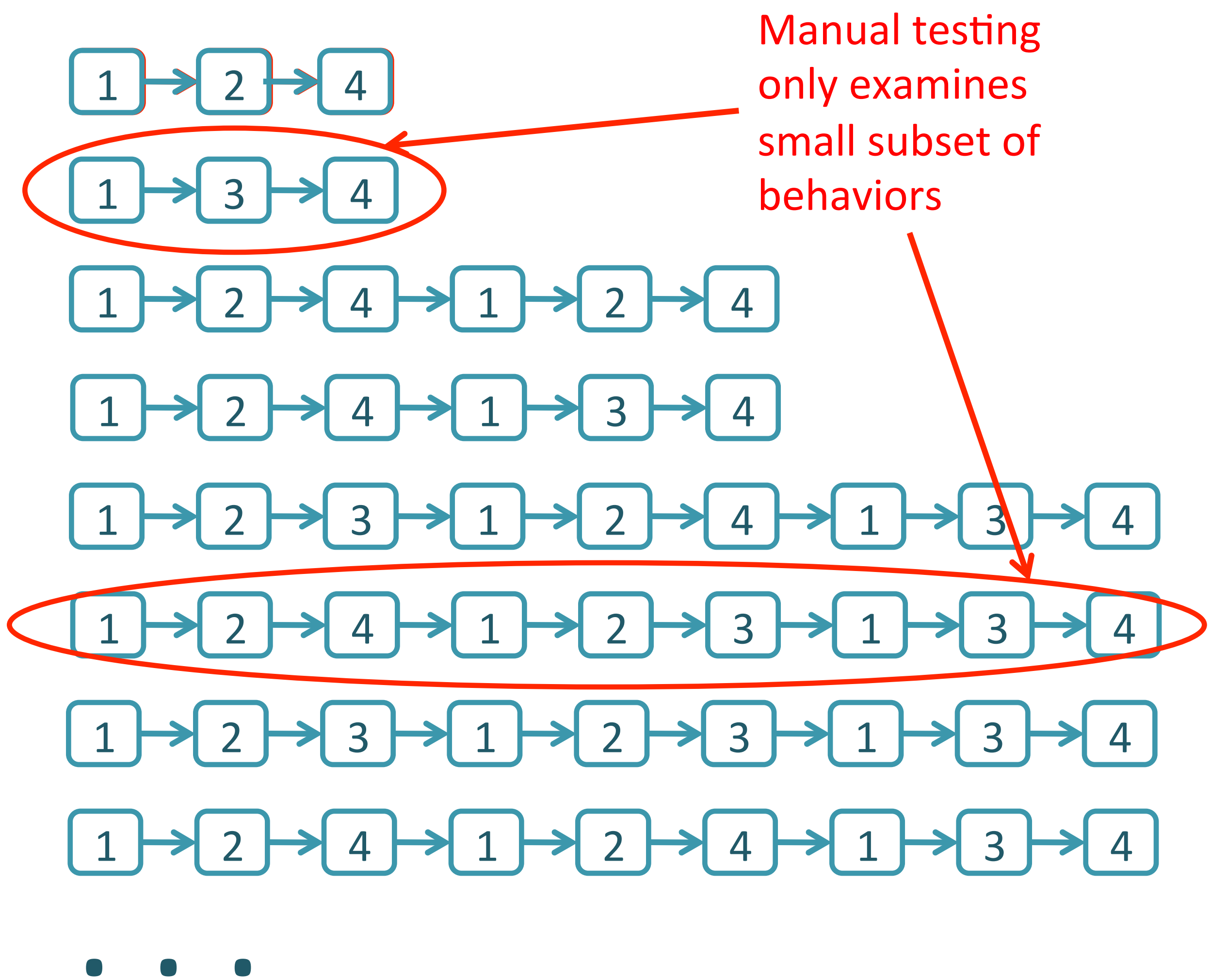
Program Analyzers



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



Software

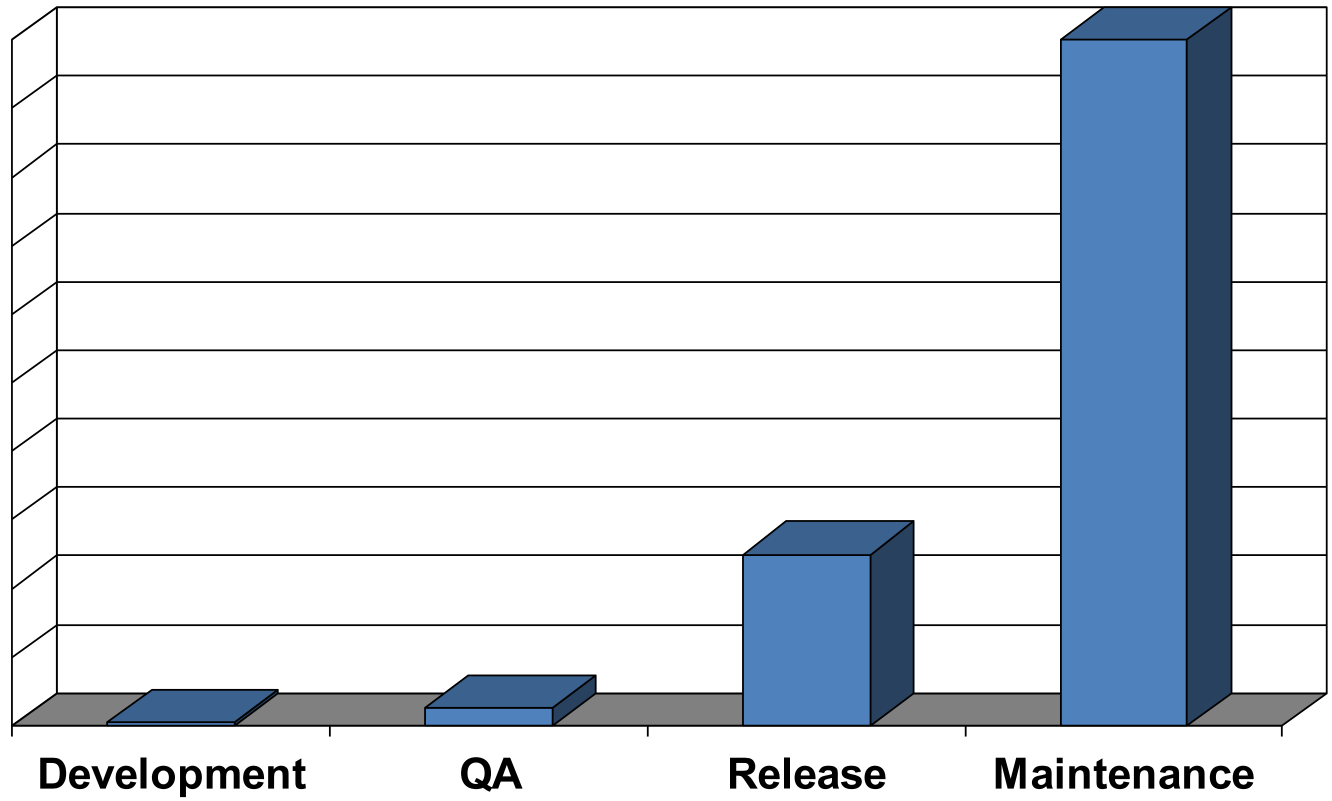


Behaviors

Static vs Dynamic Analysis

- Static
  - Can consider all possible inputs
  - Find bugs and vulnerabilities
  - Can prove absence of bugs, in some cases
- Dynamic
  - Need to choose sample test input
  - Can find bugs vulnerabilities
  - Cannot prove their absence

Cost of Fixing a Defect



Credit: Andy Chou, Coverity

Cost of security or data privacy vulnerability?

Dynamic analysis

- 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 Analysis

- Long research history
- Decade of commercial products
  - FindBugs, Fortify, Coverity, MS tools, …

Static Analysis: Outline

- General discussion of static analysis tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security checkers results
- Static analysis for of Android apps

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

Static analysis goals

- Bug finding
  - Identify code that the programmer wishes to modify or improve
- Correctness
  - Verify the absence of certain classes of errors

Soundness, Completeness

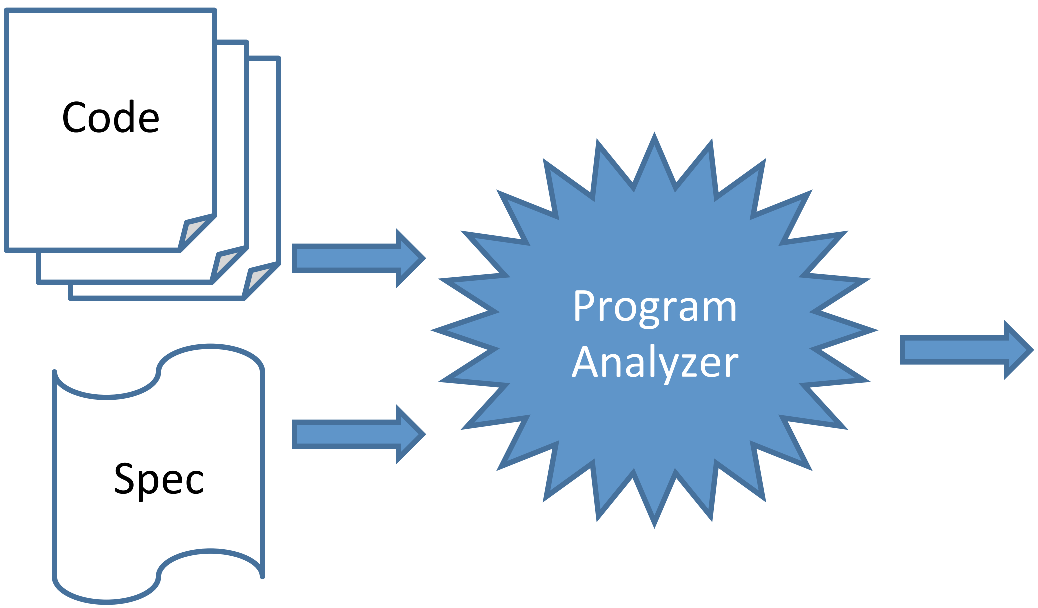
Property	Definition
Soundness	"Sound for reporting correctness" Analysis says no bugs $\rightarrow$ No bugs or equivalently There is a bug $\rightarrow$ Analysis finds a bug
Completeness	"Complete for reporting correctness" No bugs $\rightarrow$ Analysis says no bugs

Recall: $A \rightarrow B$ is equivalent to $(\neg B) \rightarrow (\neg A)$

	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**

Sound Program Analyzer

Analyze large code bases



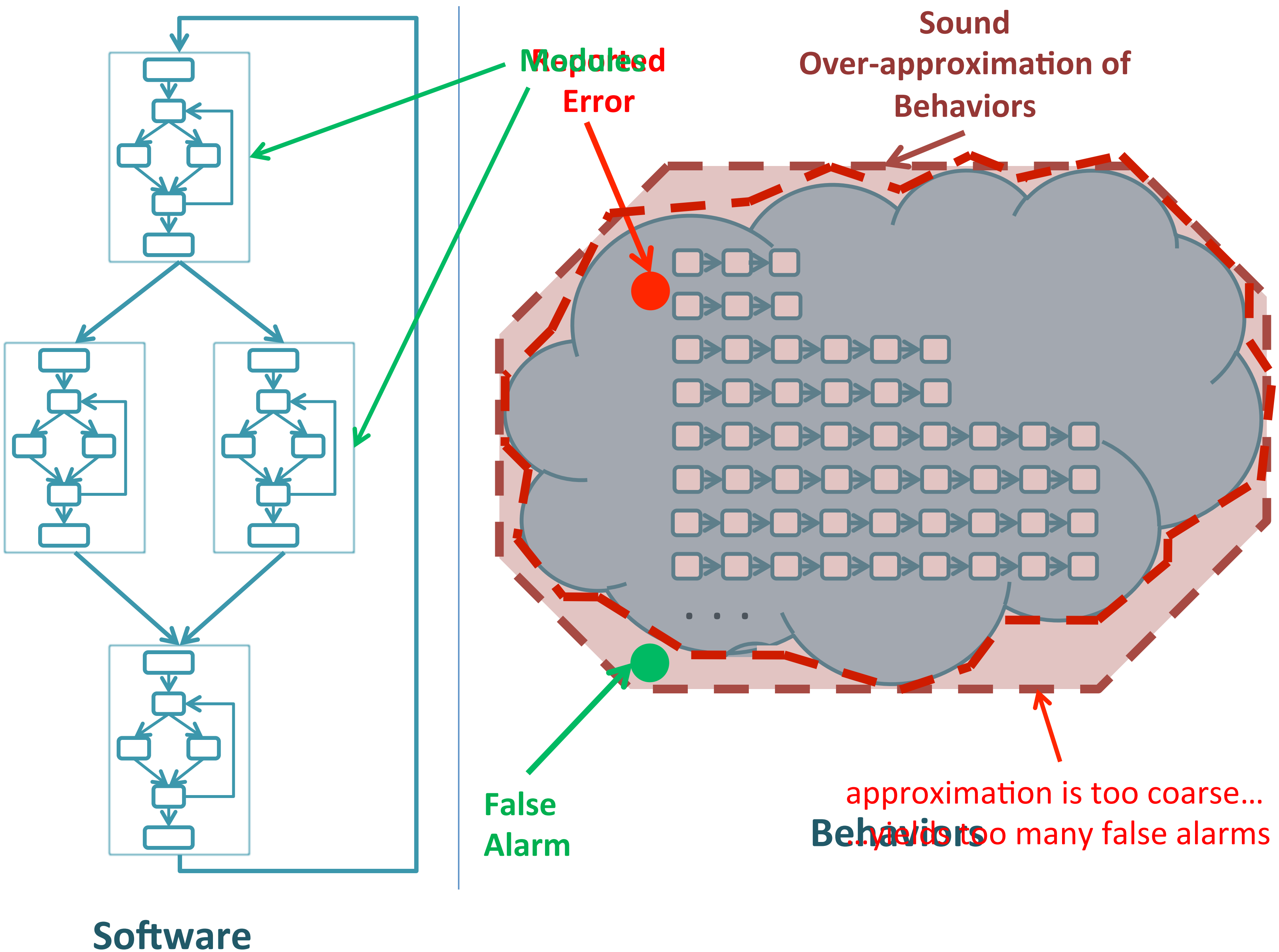
Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

false alarm

false alarm

Sound: may report many warnings

May emit false alarms

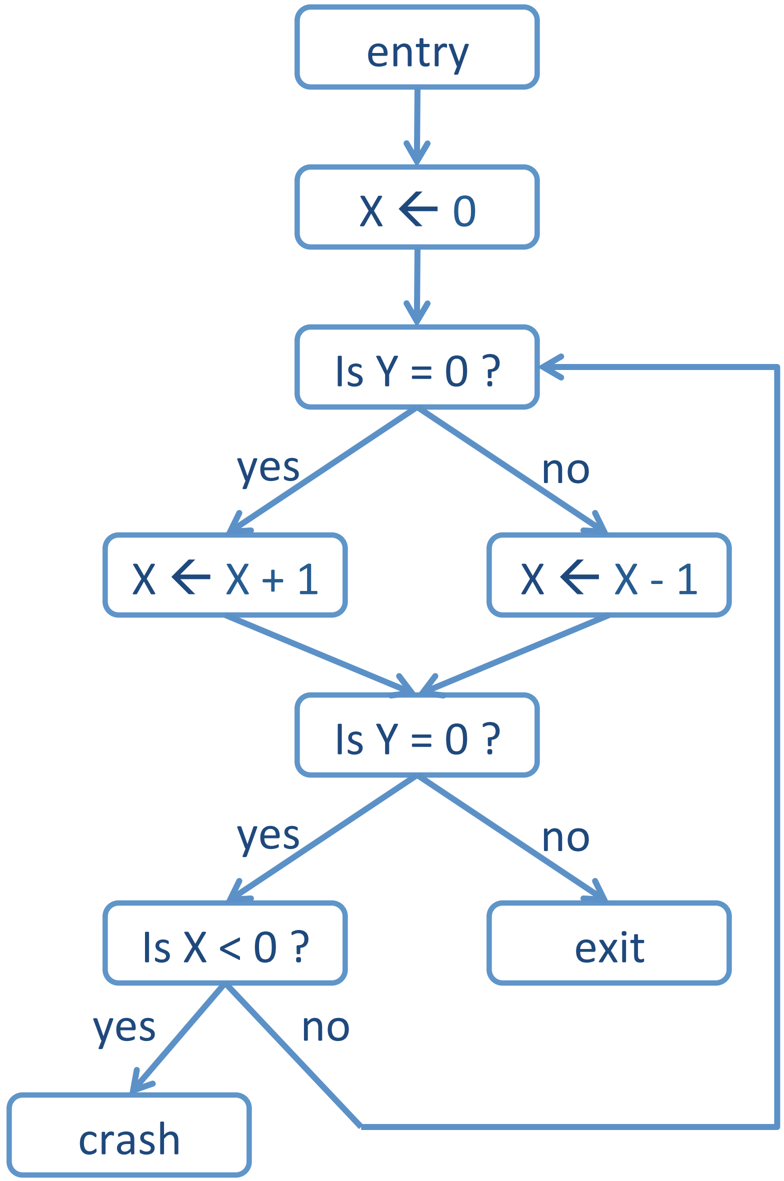

Module
Reported
Error
Sound
Over-approximation of
Behaviors
False
Alarm
approximation is too coarse…
…yields too many false alarms
Behaviors
Software

Outline

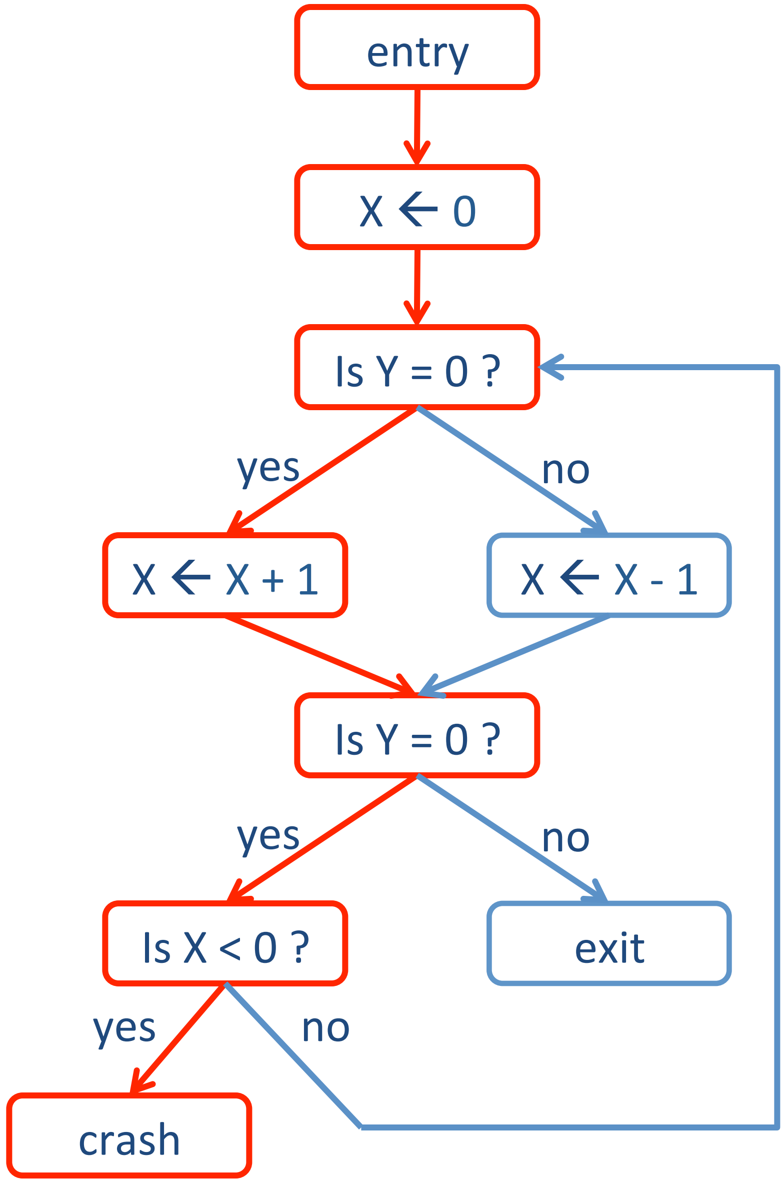
- General discussion of tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for Android malware
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

Does this program ever crash?

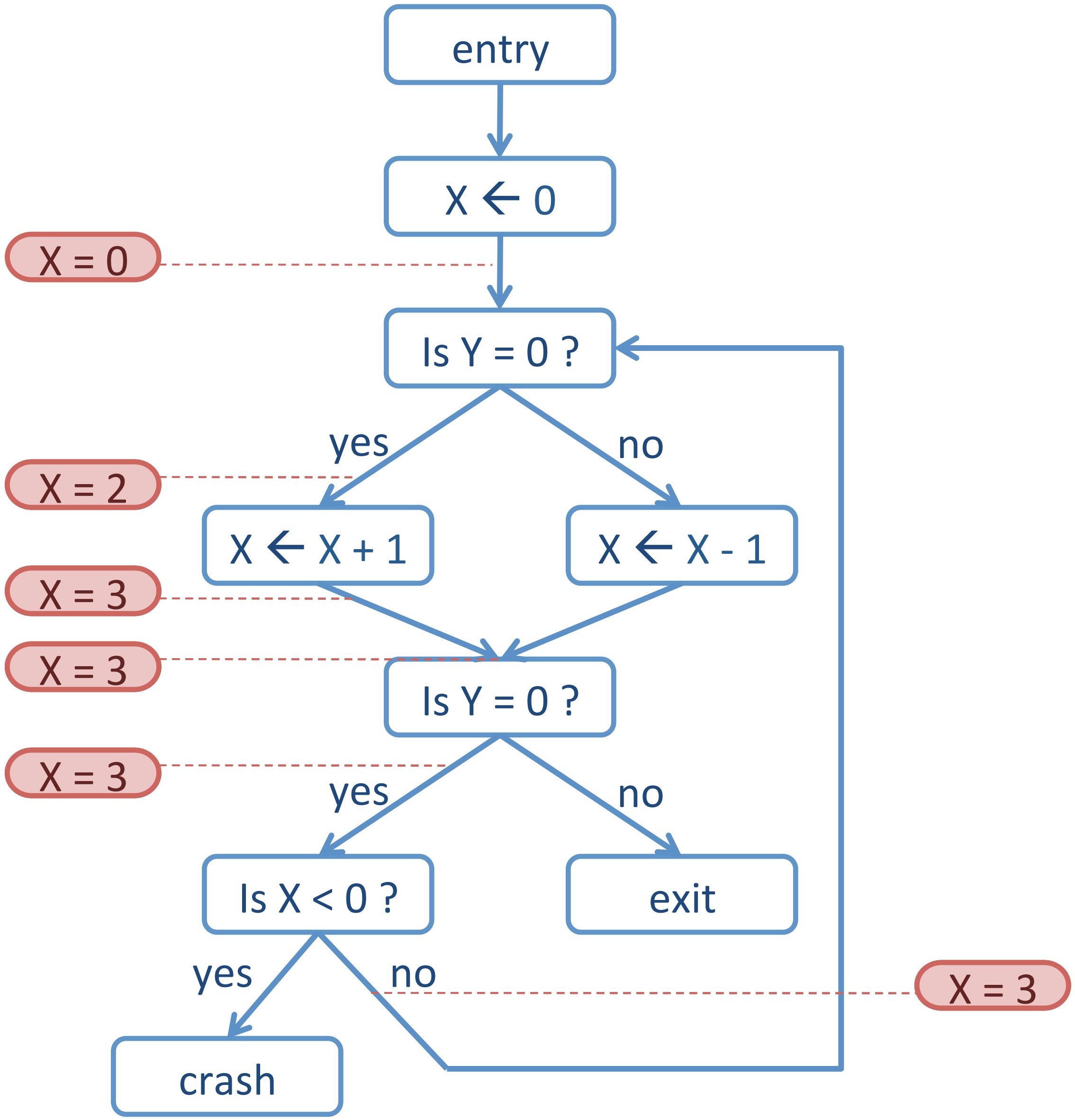


Does this program ever crash?



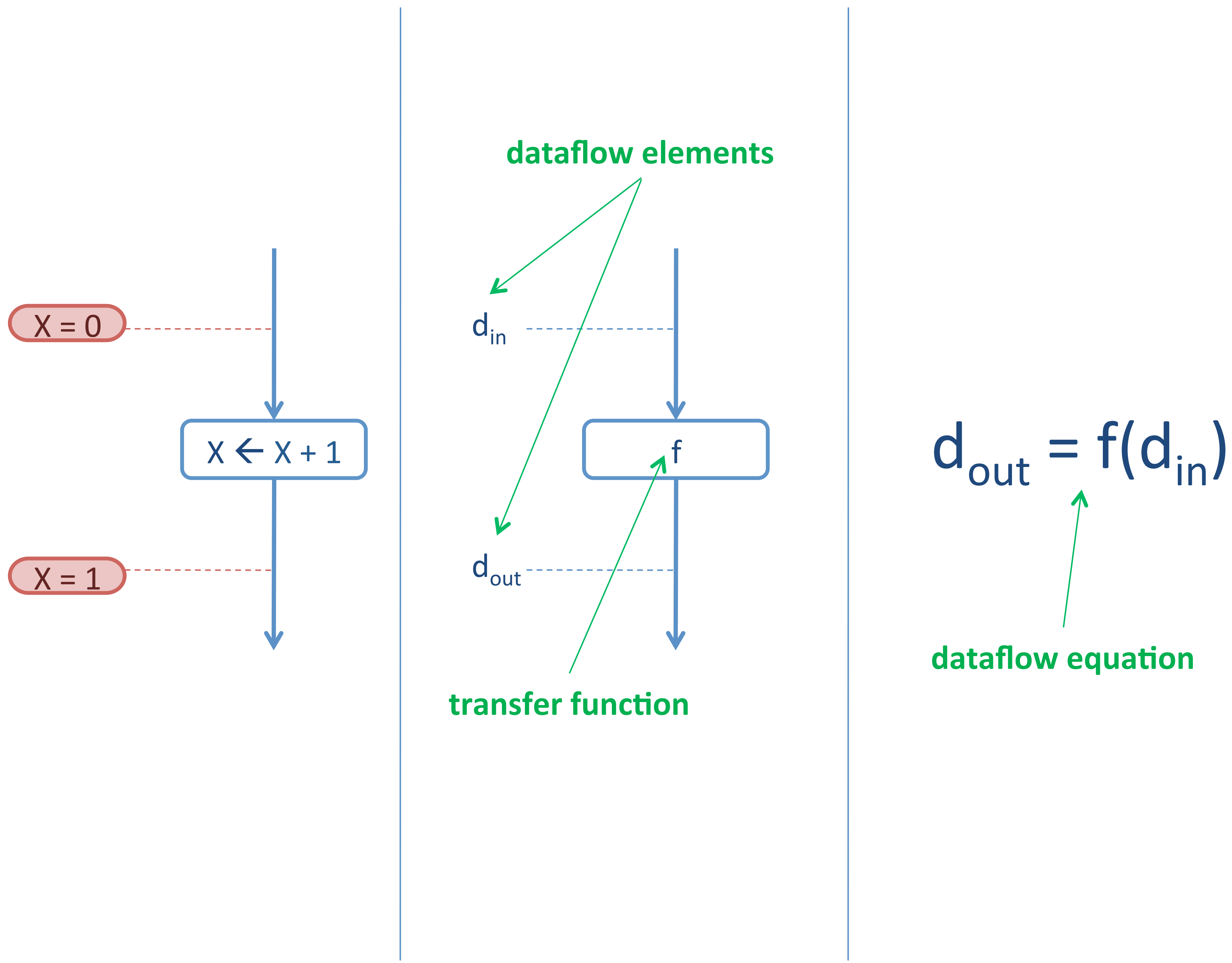
infeasible path!
… program will never crash

Try analyzing without approximating…

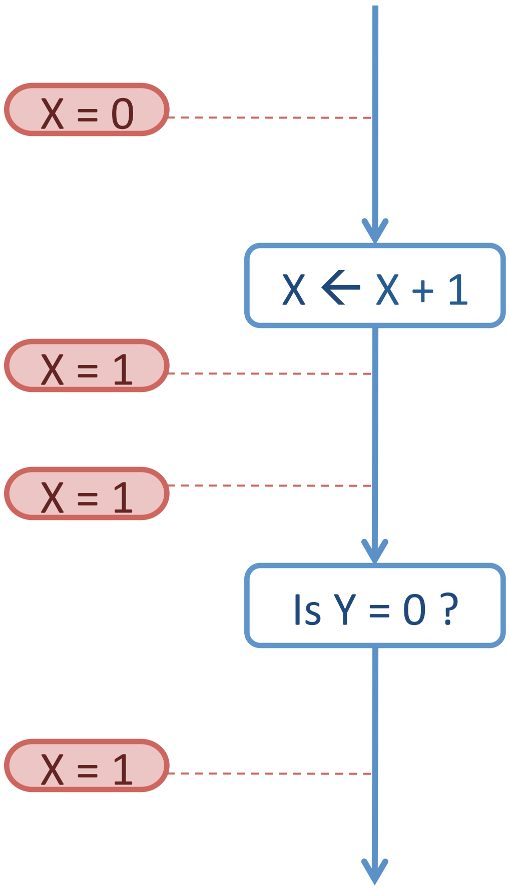


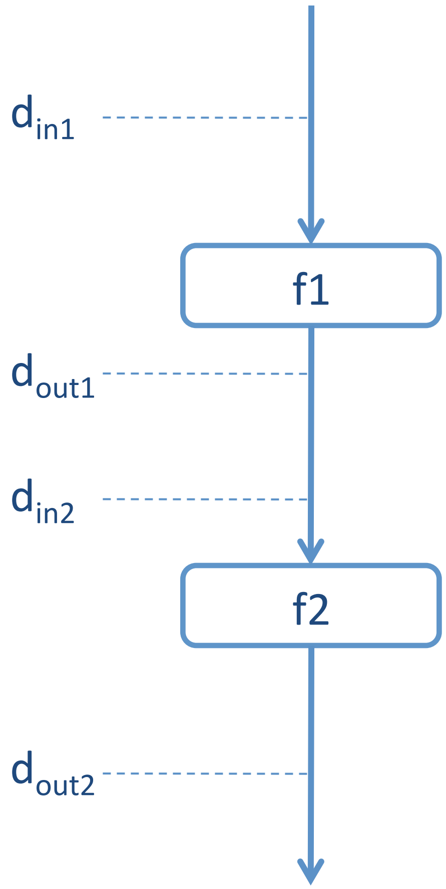
non-termination!

… therefore, need to approximate



X = 0
X ← X + 1
X = 1
dataflow elements
d_in
f
d_out
transfer function
d_out = f(d_in)
dataflow equation

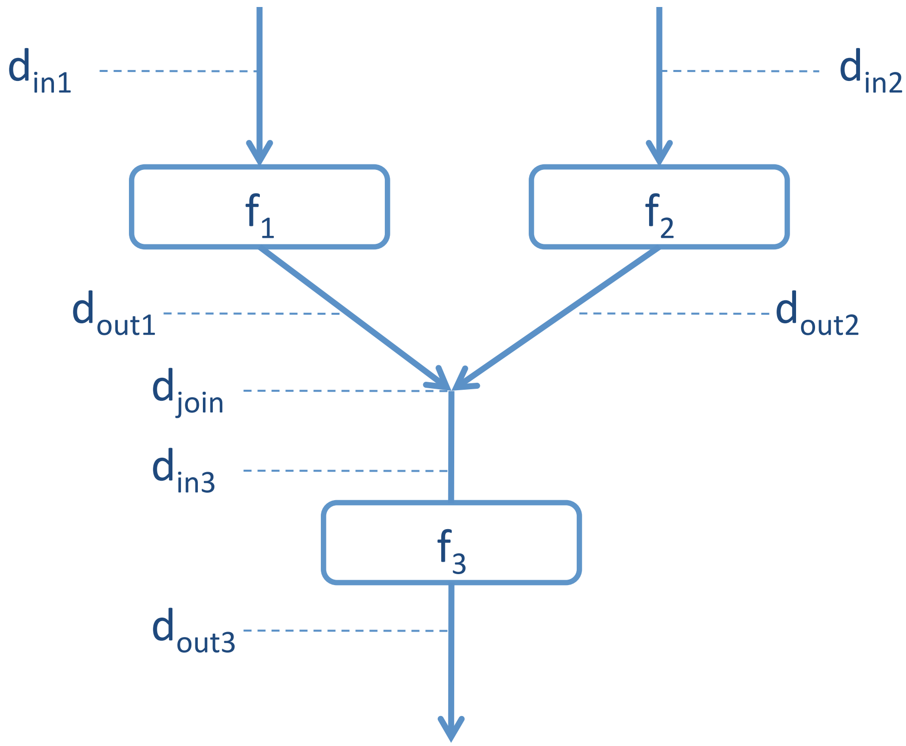




$$d_{out1} = f_1(d_{in1})$$

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

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



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

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

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

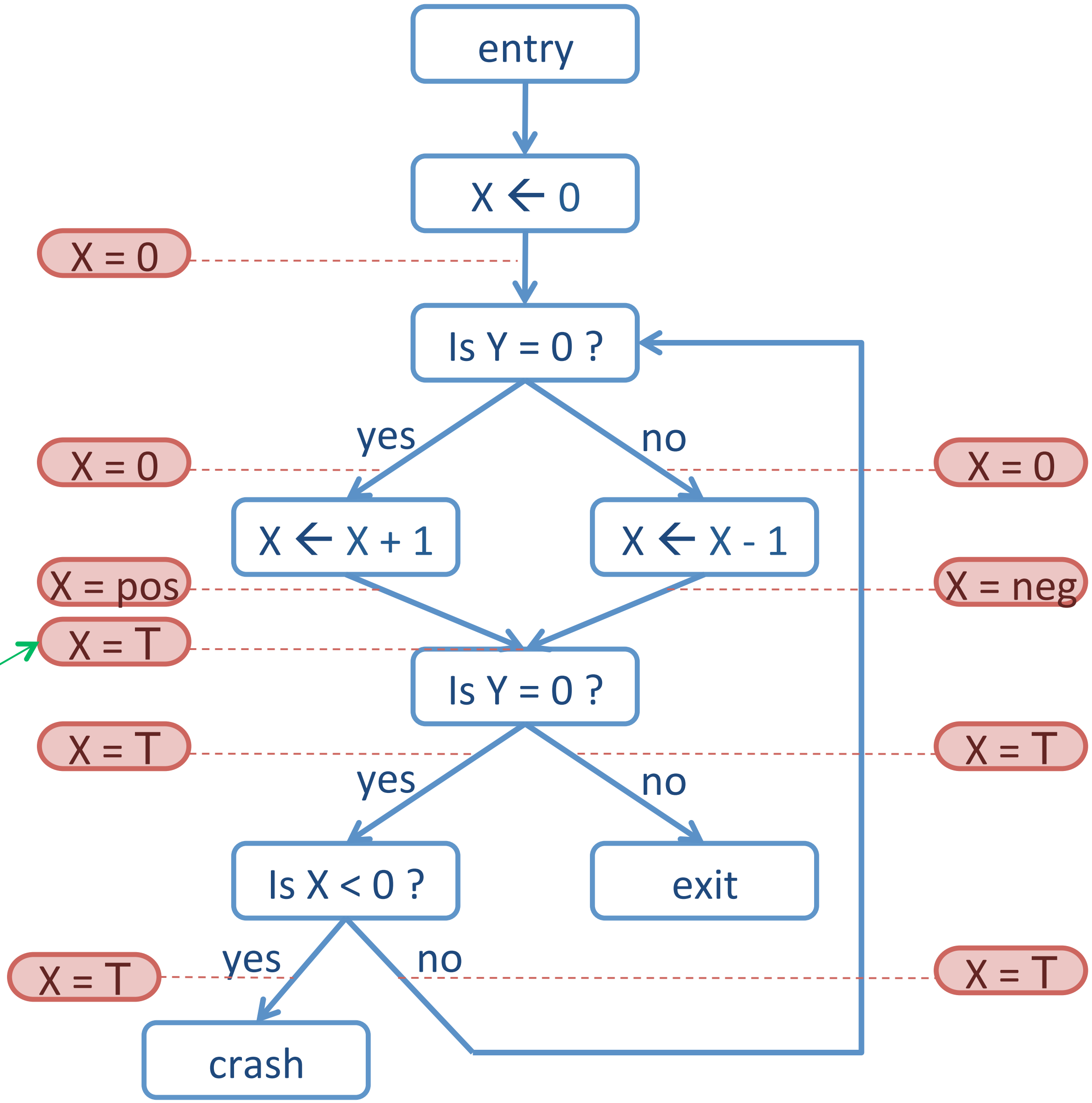
$$d_{join} = d_{in3}$$

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

least upper bound operator
Example: union of possible values

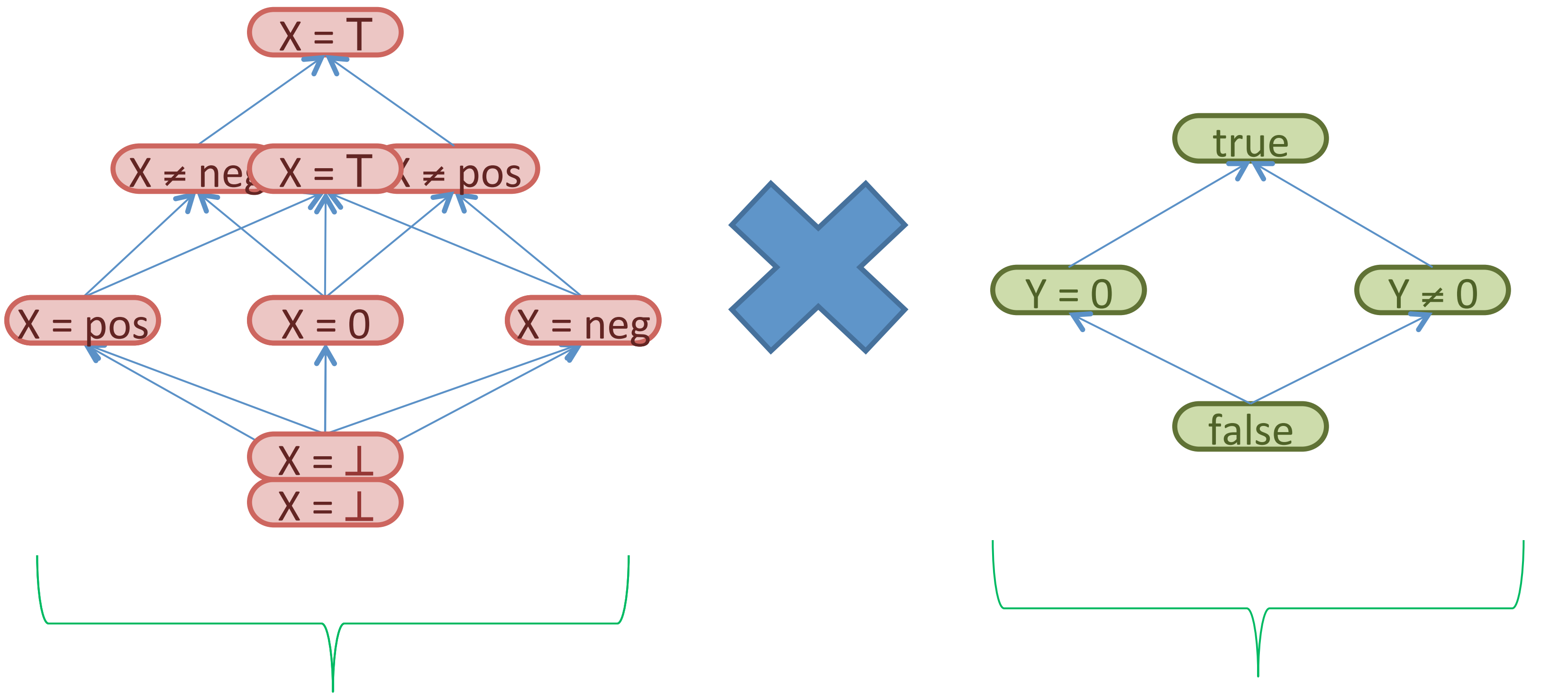
What is the space of dataflow elements, Δ?
What is the least upper bound operator, $\sqcup$?

Try analyzing with "signs" approximation…



lost precision

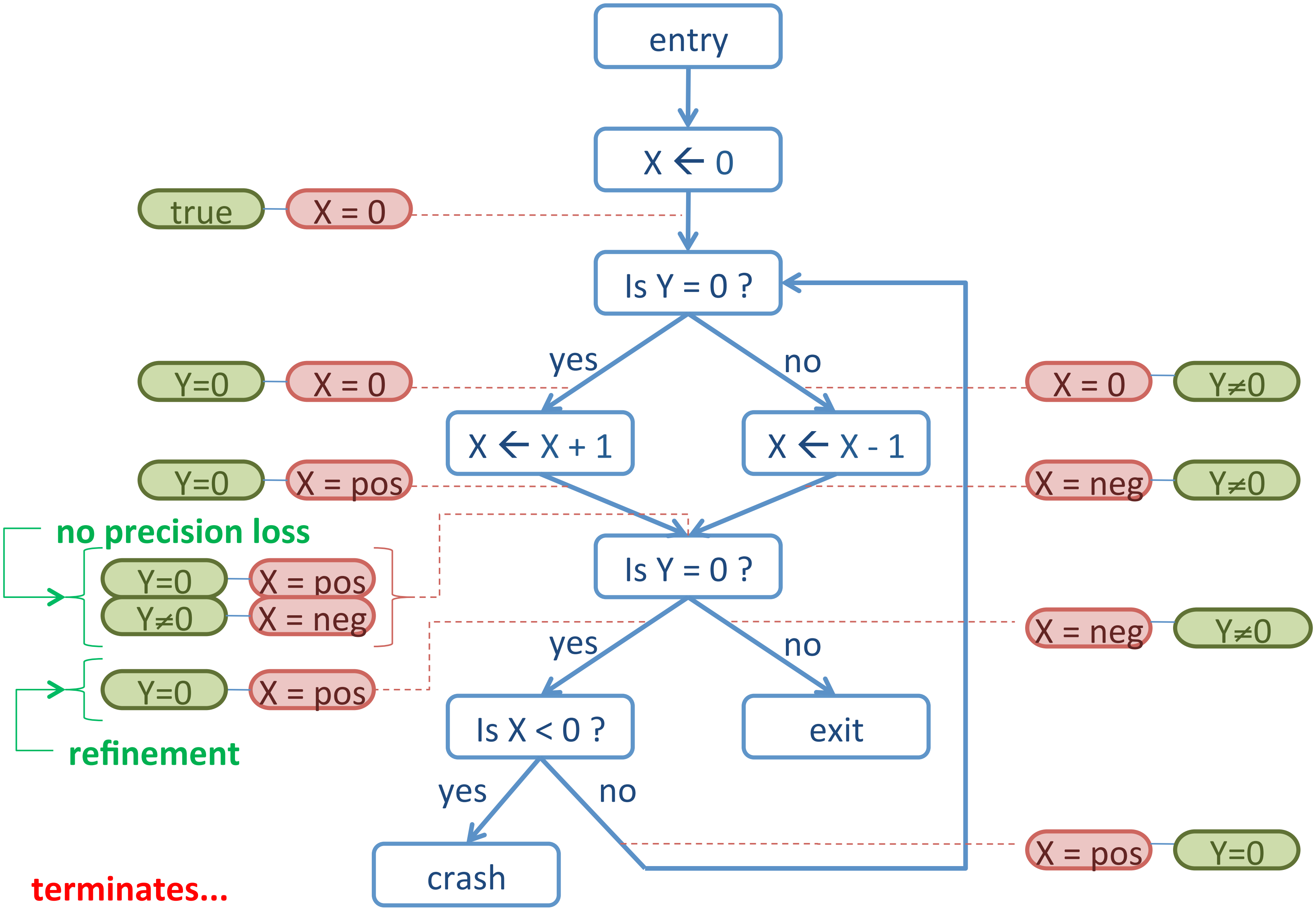
terminates…
… but reports false alarm
… therefore, need more precision



refined sign lattice
sign lattice

Boolean formula lattice

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



terminates...

... no false alarm

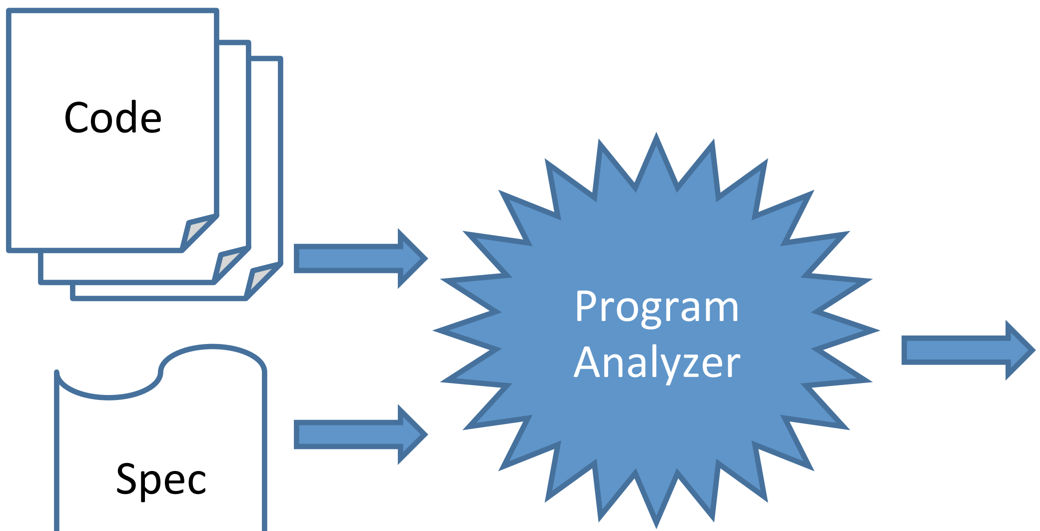
... soundly proved never crashes

Outline

- General discussion of tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for Android malware
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

Unsound Program Analyzer



analyze large code bases

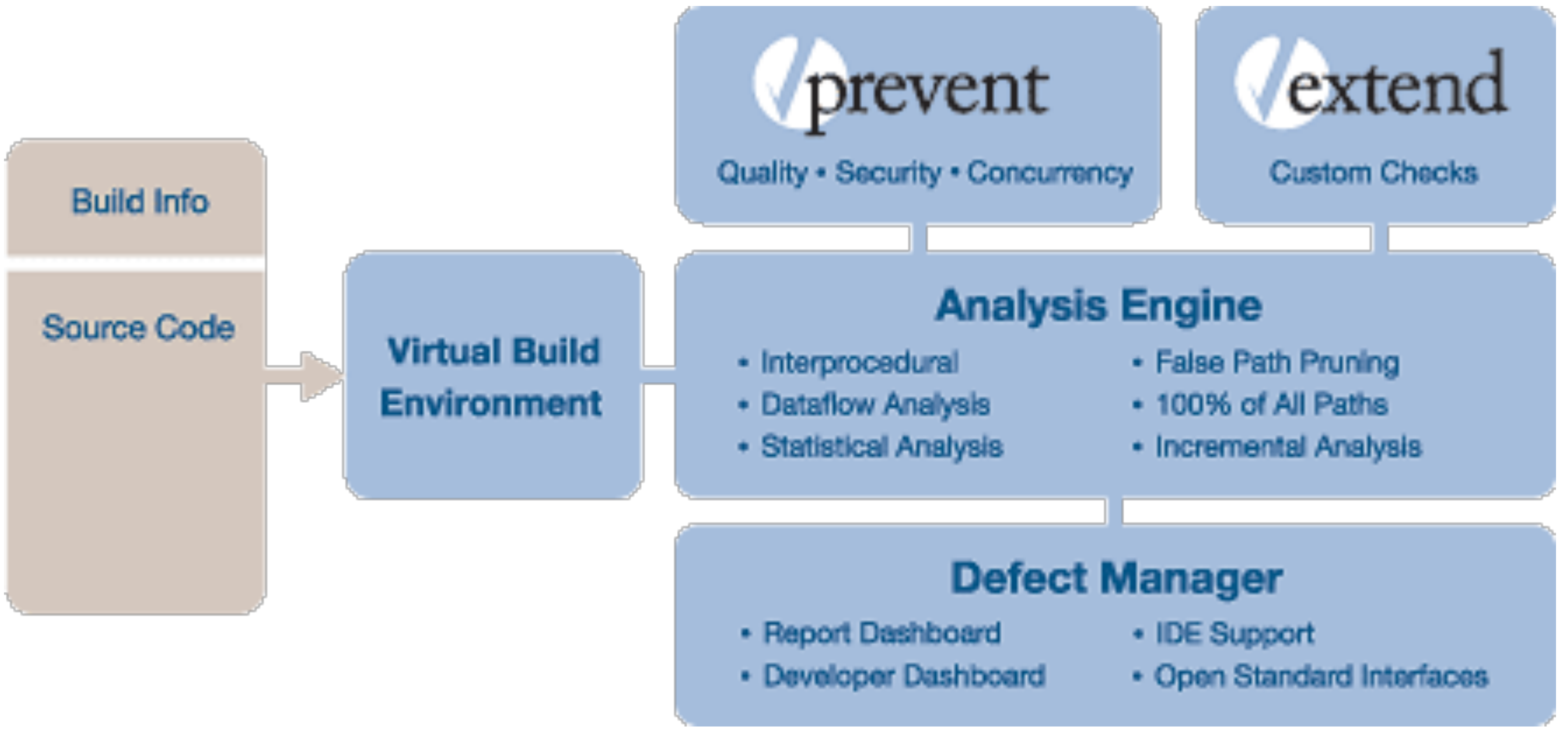
Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…

false alarm

false alarm

Not sound: may miss some bugs

may emit false alarms


Build Info
Source Code
Virtual Build Environment
prevent
Quality • Security • Concurrency
extend
Custom Checks
Analysis Engine
• Interprocedural
• Dataflow Analysis
• Statistical Analysis
• False Path Pruning
• 100% of All Paths
• Incremental Analysis
Defect Manager
• Report Dashboard
• Developer Dashboard
• IDE Support
• Open Standard Interfaces

Demo

- Coverity video: http://youtu.be/_Vt4niZfNeA
- Observations
  - Code analysis integrated into development workflow
  - Program context important: analysis involves sequence of function calls, surrounding statements
  - This is a sales video: no discussion of false alarms

Bugs to Detect

Some examples

- Crash Causing Defects
- Null pointer dereference
- Use after free
- Double free
- Array indexing errors
- Mismatched array new/delete
- Potential stack overrun
- Potential heap overrun
- Return pointers to local variables
- Logically inconsistent code
- Uninitialized variables
- Invalid use of negative values
- Passing large parameters by value
- Underallocations of dynamic data
- Memory leaks
- File handle leaks
- Network resource leaks
- Unused values
- Unhandled return codes
- Use of invalid iterators

Slide credit: Andy Chou

Example: Check for missing optional args

- **Prototype for open() syscall:**

```
int open(const char *path, int oflag, /* mode_t mode */...);
```

- **Typical mistake:**

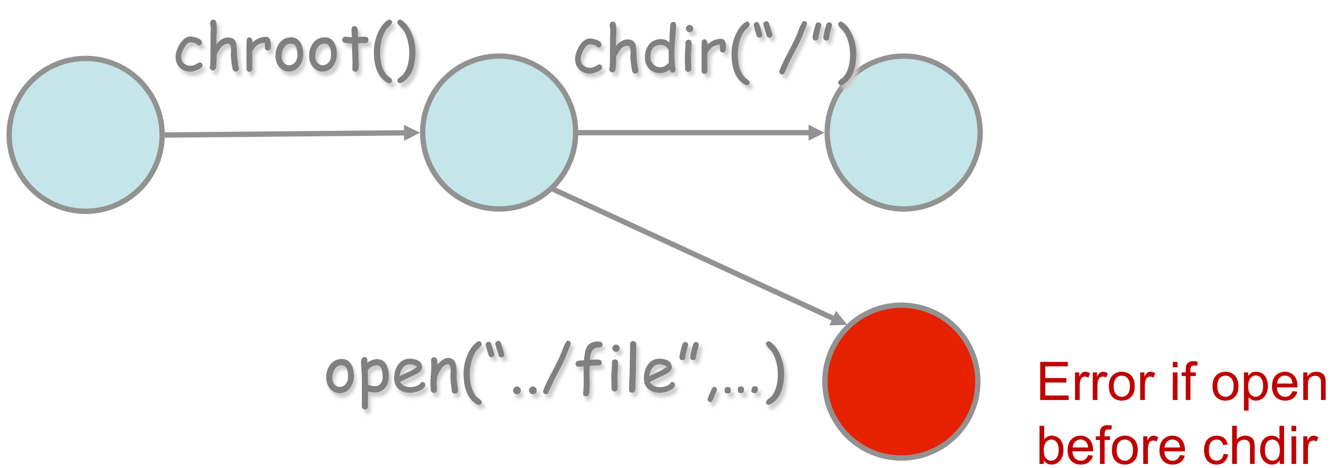
```
fd = open("file", O_CREAT);
```

- **Result: file has random permissions**

- **Check: Look for oflags == O_CREAT without mode argument**

Example: Chroot protocol checker

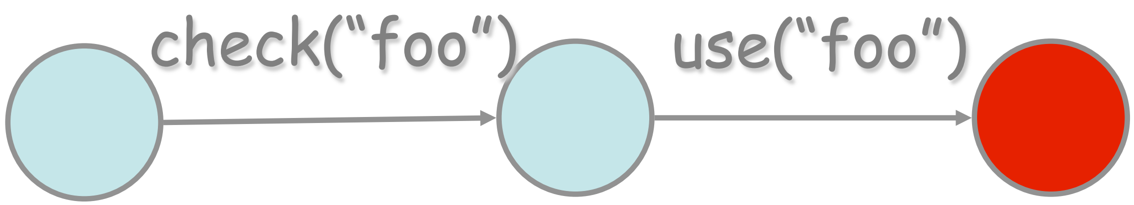
- **Goal: confine process to a "jail" on the filesystem**
  - chroot() changes filesystem root for a process
- **Problem**
  - chroot() itself does not change current working directory



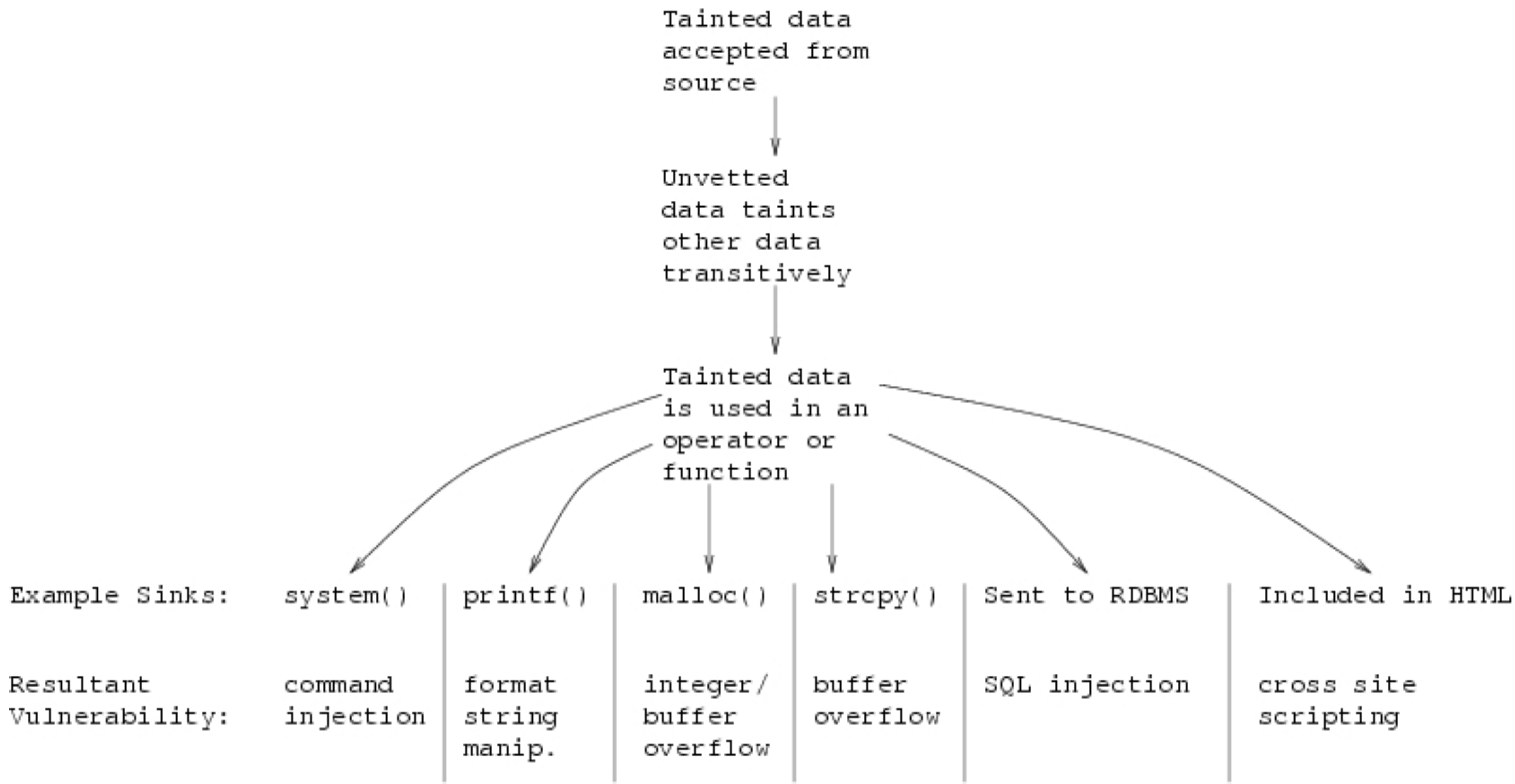
TOCTOU

- Race condition between time of check and use

- Not applicable to all programs



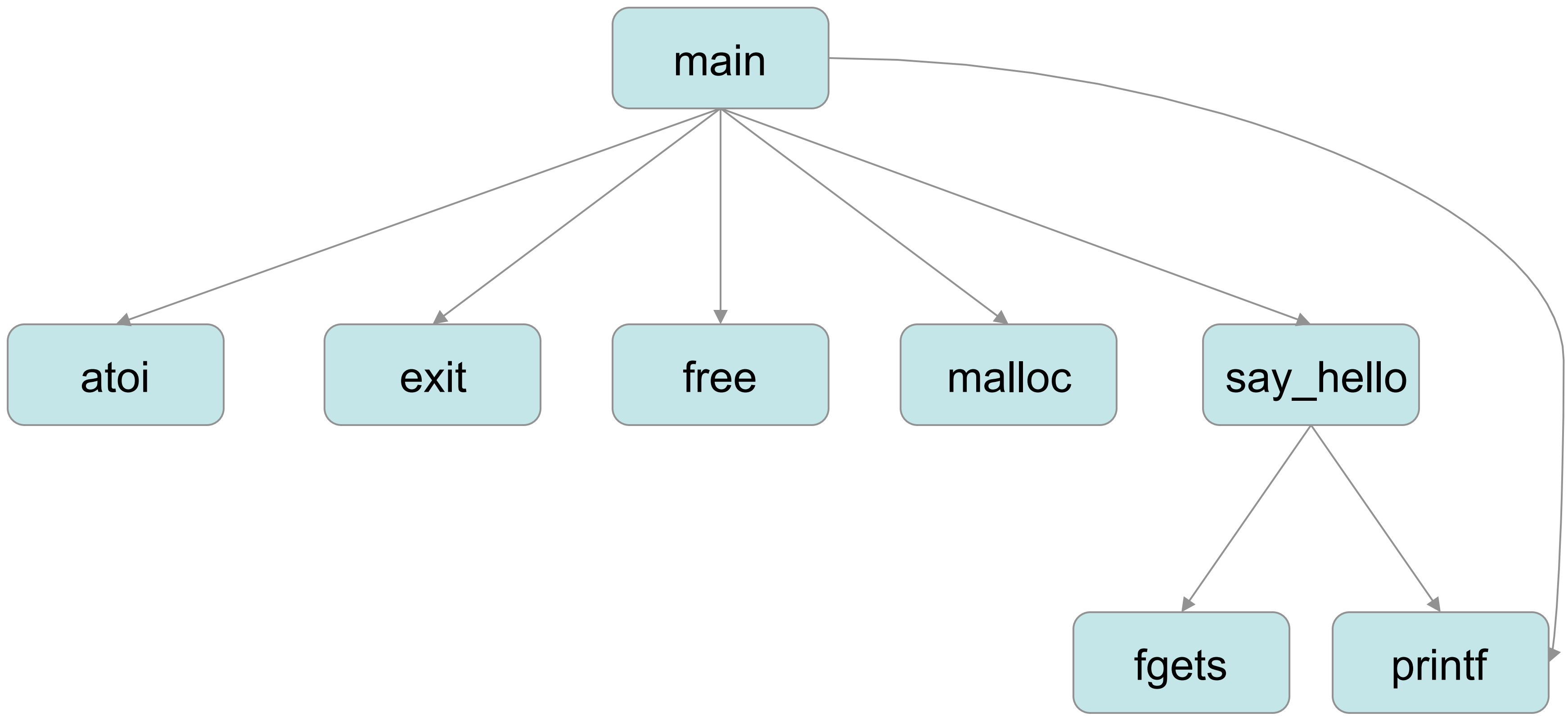
Tainting checkers



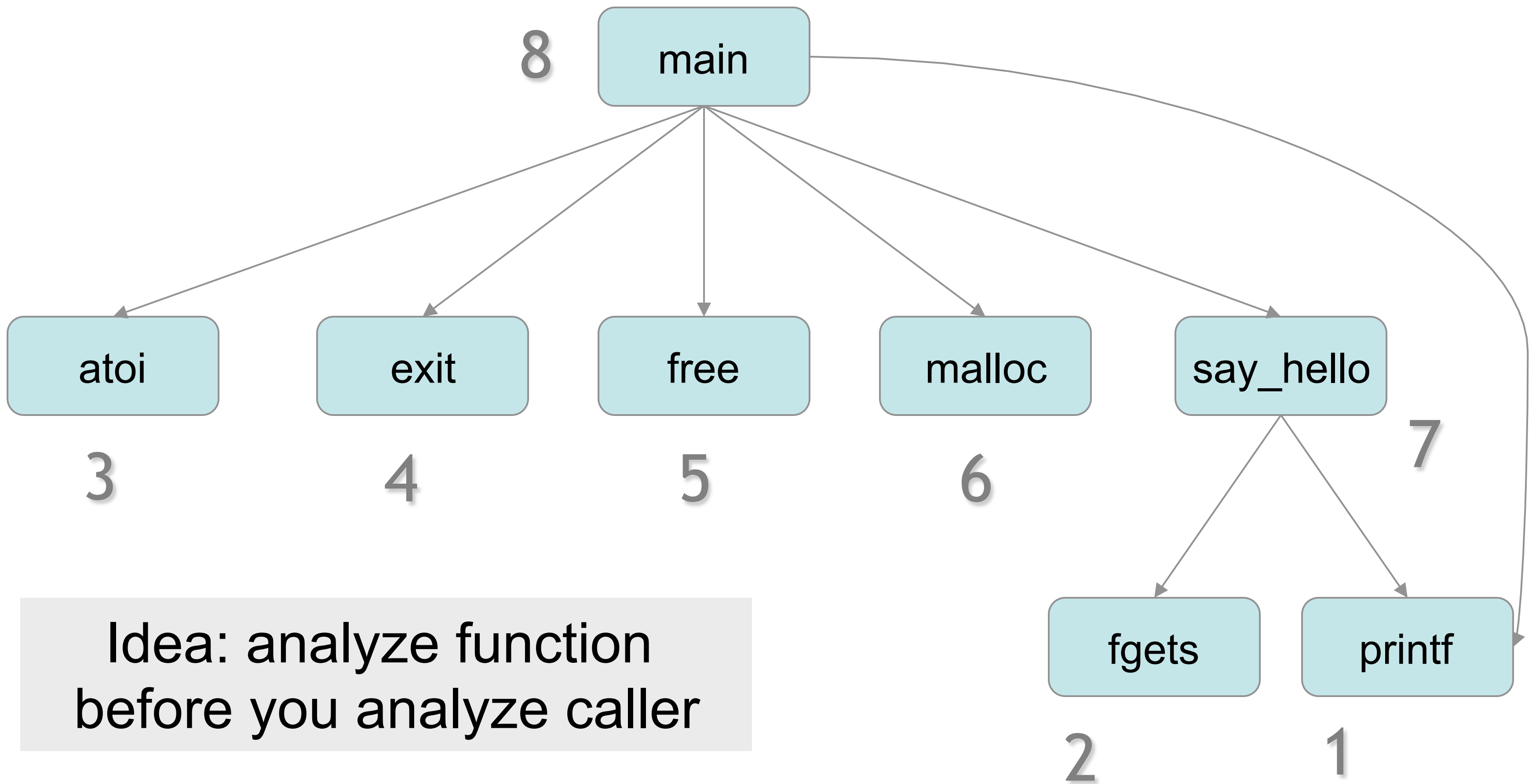
Example code with function def, calls

```
#include <stdlib.h>
#include <stdio.h>

void say_hello(char * name, int size) {
  printf("Enter your name: ");
  fgets(name, size, stdin);
  printf("Hello %s.\n", name);
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Error, must provide an input buffer size.\n");
    exit(-1);
  }
  int size = atoi(argv[1]);
  char * name = (char*)malloc(size);
  if (name) {
    say_hello(name, size);
    free(name);
  } else {
    printf("Failed to allocate %d bytes.\n", size);
  }
}
```

Callgraph

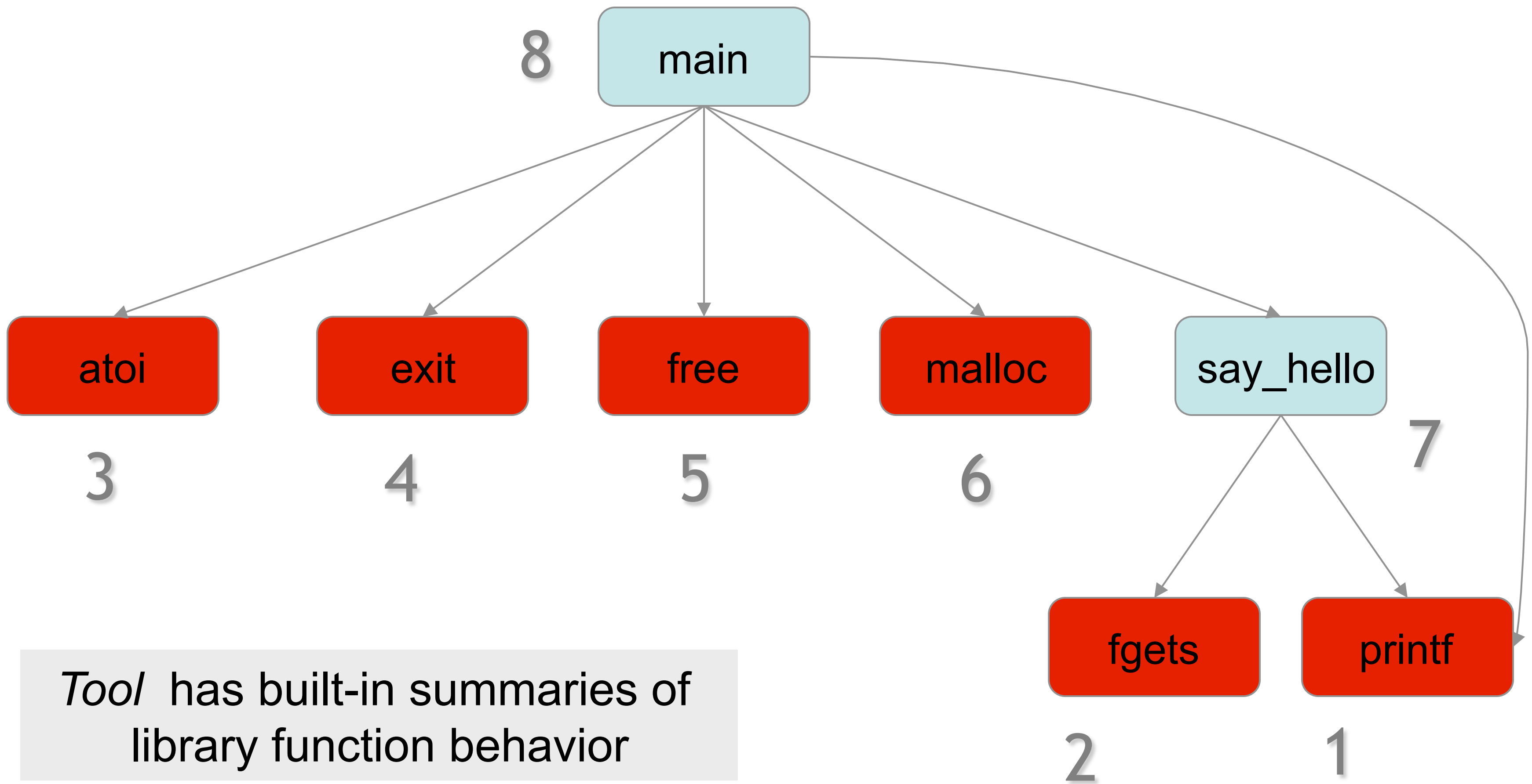


Reverse Topological Sort



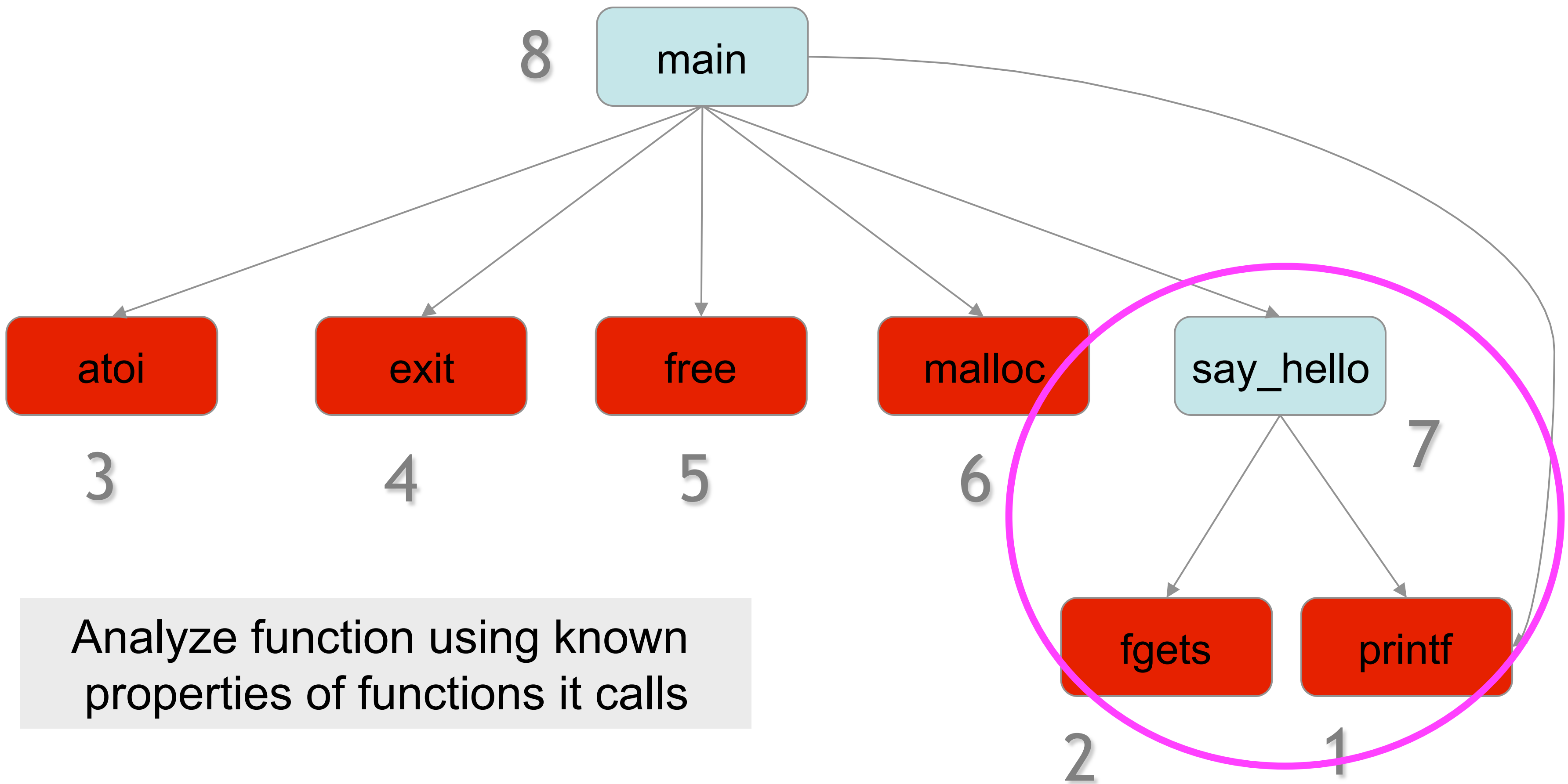
Idea: analyze function before you analyze caller

Apply Library Models



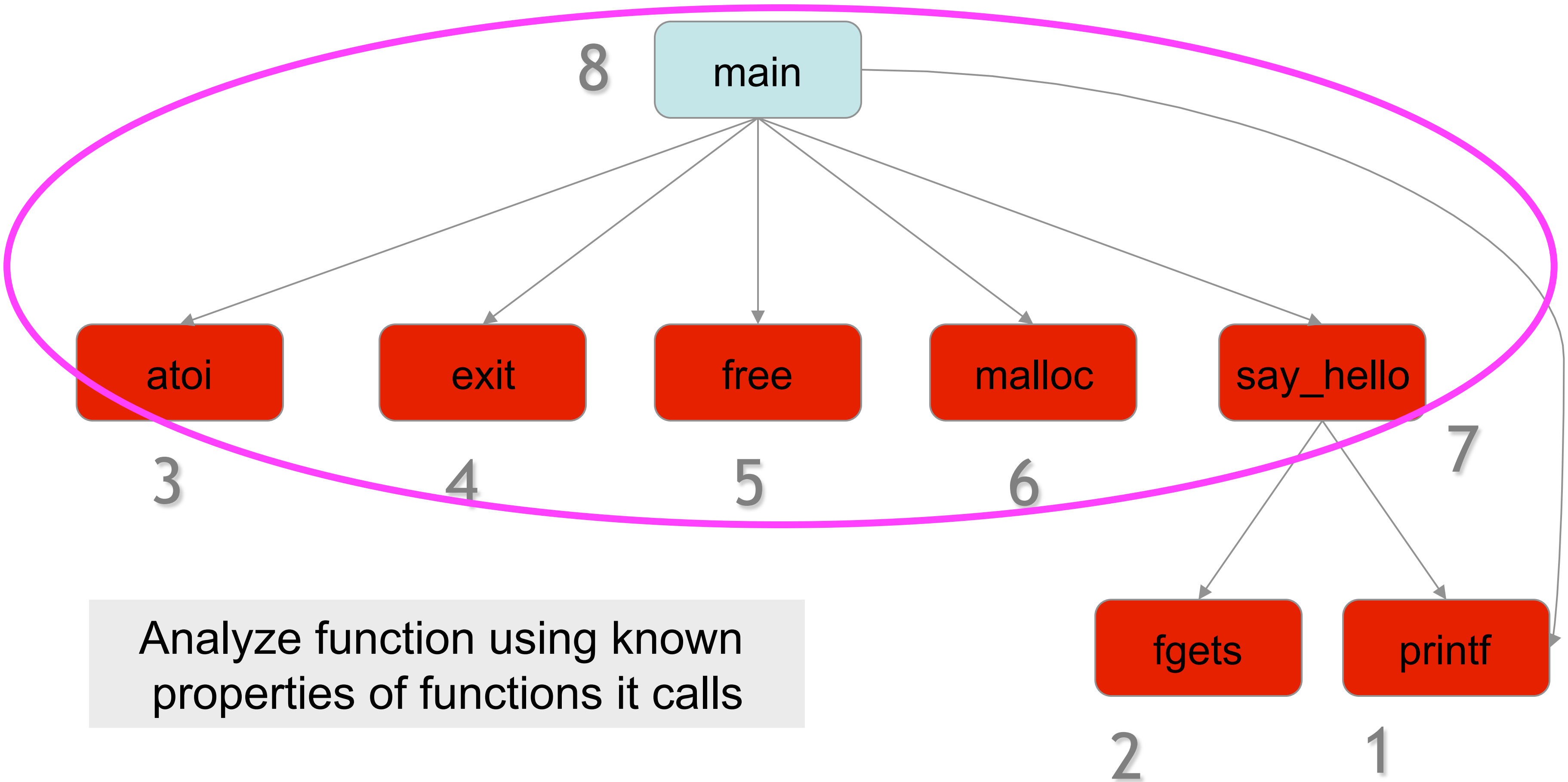
Tool has built-in summaries of library function behavior

Bottom Up Analysis



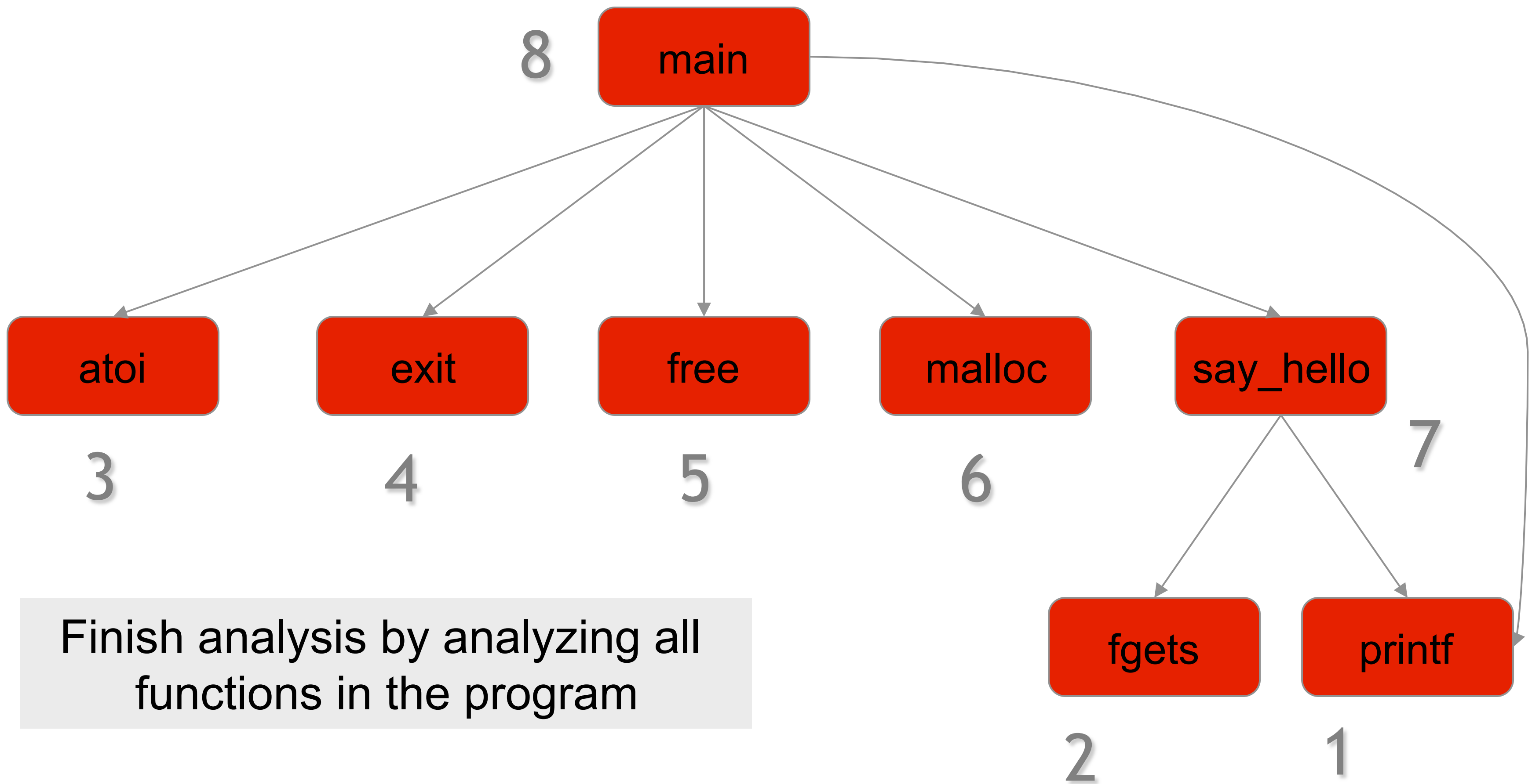
Analyze function using known properties of functions it calls

Bottom Up Analysis



Analyze function using known properties of functions it calls

Bottom Up Analysis

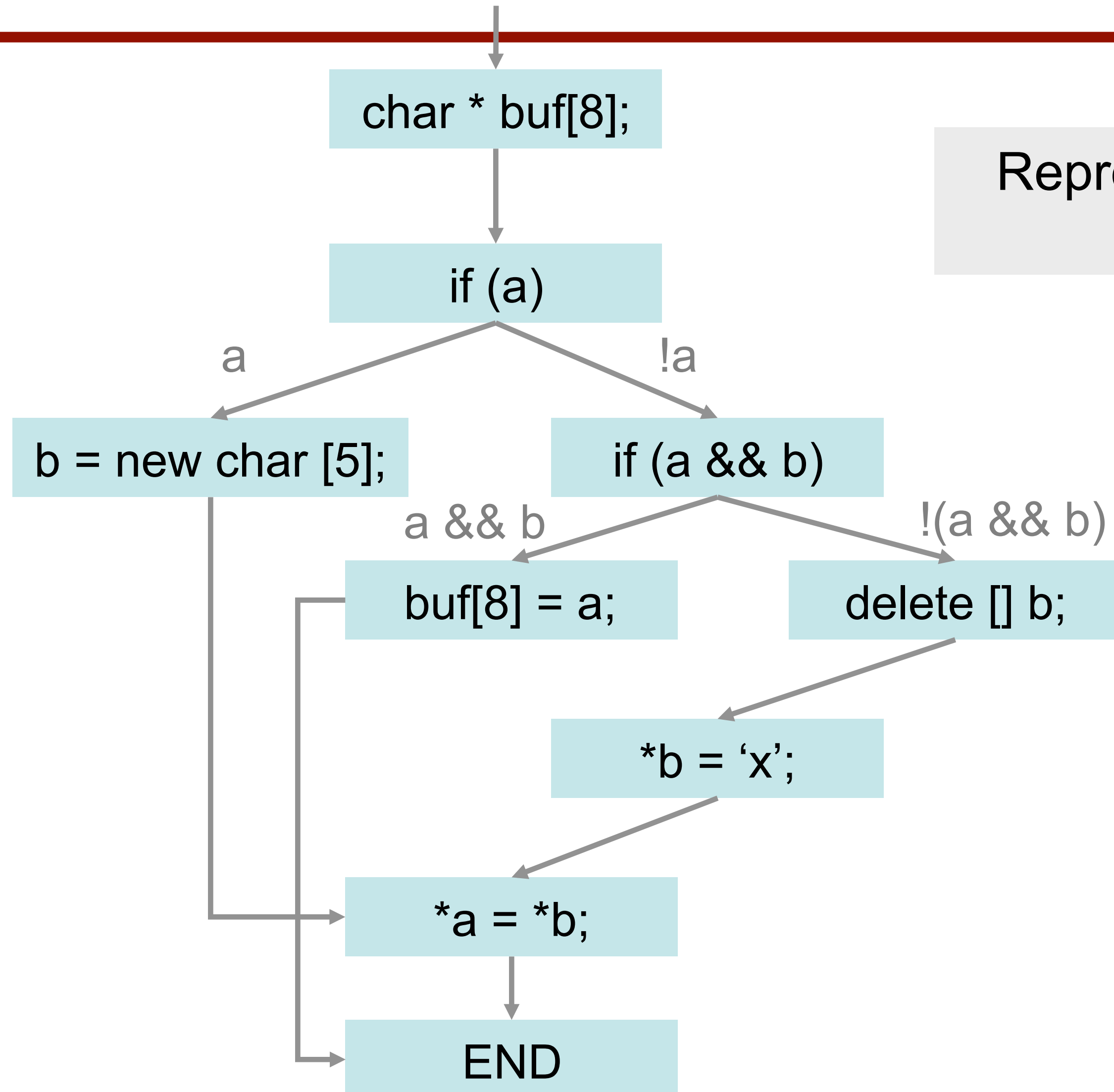


Finish analysis by analyzing all functions in the program

Finding Local Bugs

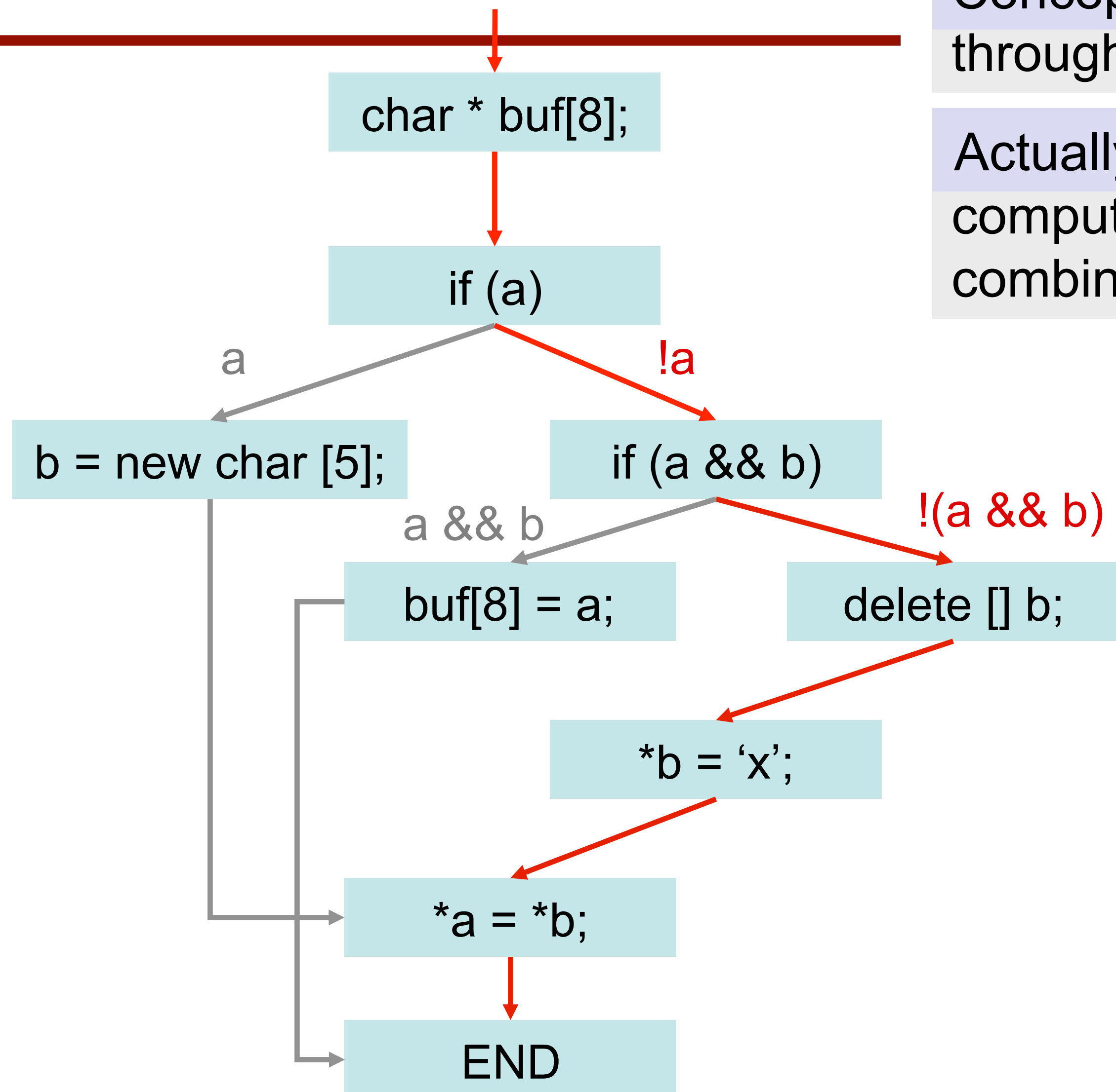
```
#define SIZE 8
void set_a_b(char * a, char * b) {
 char * buf[SIZE];
 if (a) {
     b = new char[5];
 } else {
     if (a && b) {
      buf[SIZE] = a;
      return;
     } else {
      delete [] b;
     }
     *b = 'x';
 }
 *a = *b;
}
```

Control Flow Graph



Represent logical structure of code in graph form

Path Traversal

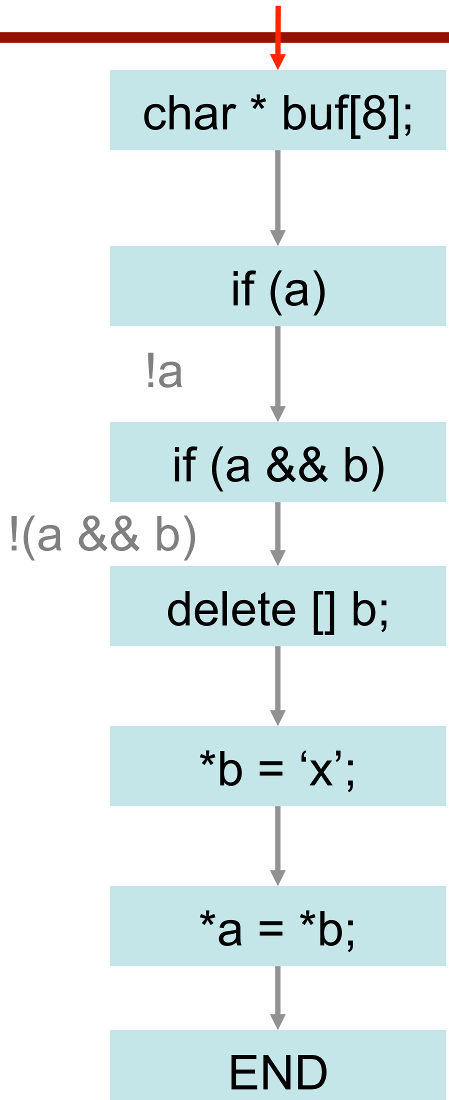


Conceptually Analyze each path through control graph separately

Actually Perform some checking computation once per node; combine paths at merge nodes

Apply Checking

Null pointers Use after free Array overrun



See how three checkers are run for this path

Checker

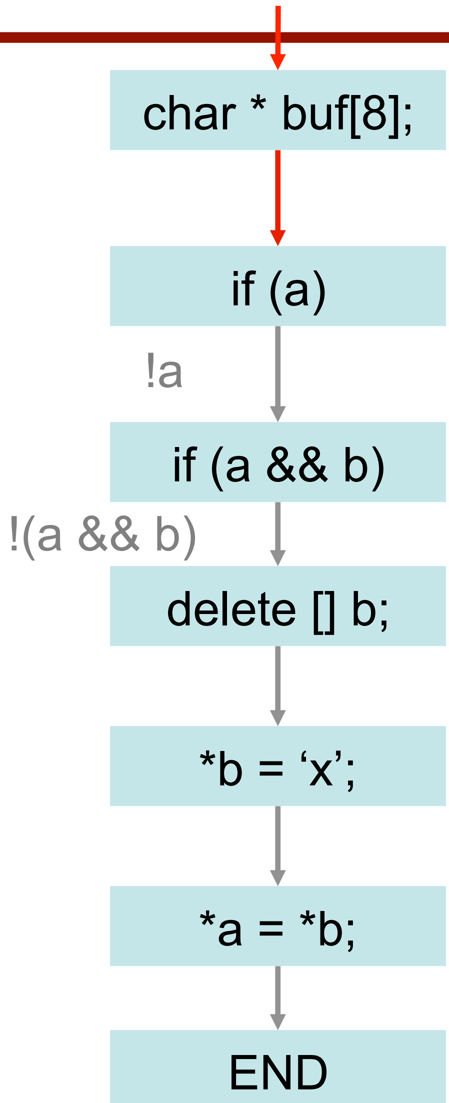
- Defined by a state diagram, with state transitions and error states

Run Checker

- Assign initial state to each program var
- State at program point depends on state at previous point, program actions
- Emit error if error state reached

Apply Checking

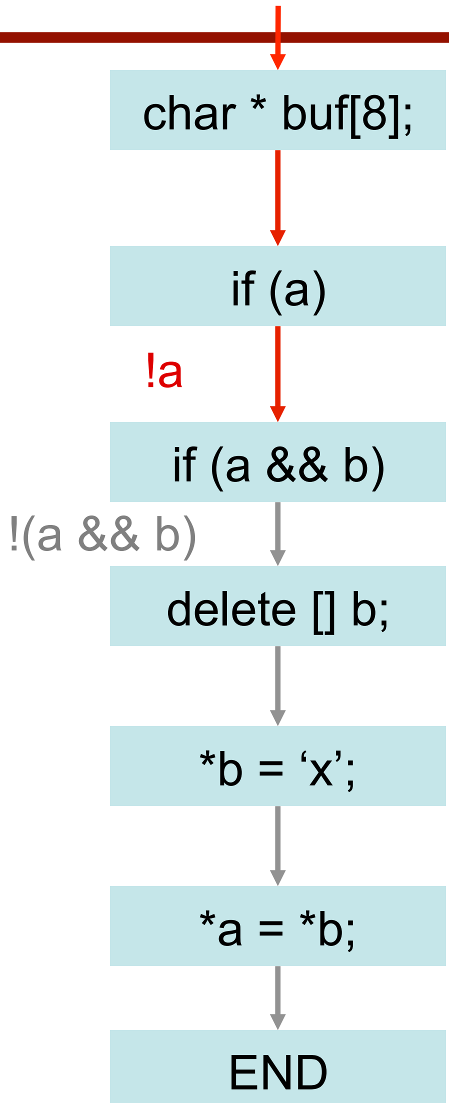
Null pointers Use after free Array overrun



“buf is 8 bytes”

Apply Checking

Null pointers Use after free Array overrun

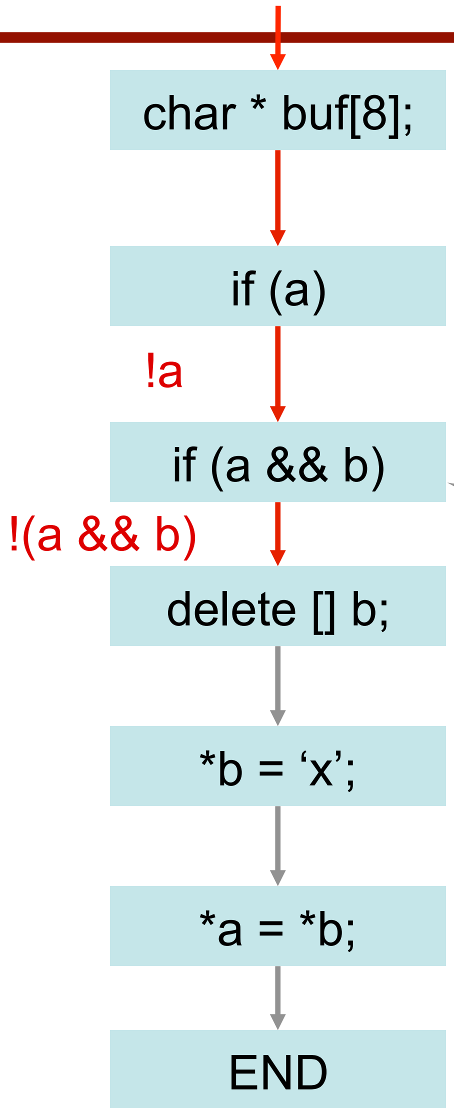


“buf is 8 bytes”

“a is null”

Apply Checking

Null pointers Use after free Array overrun

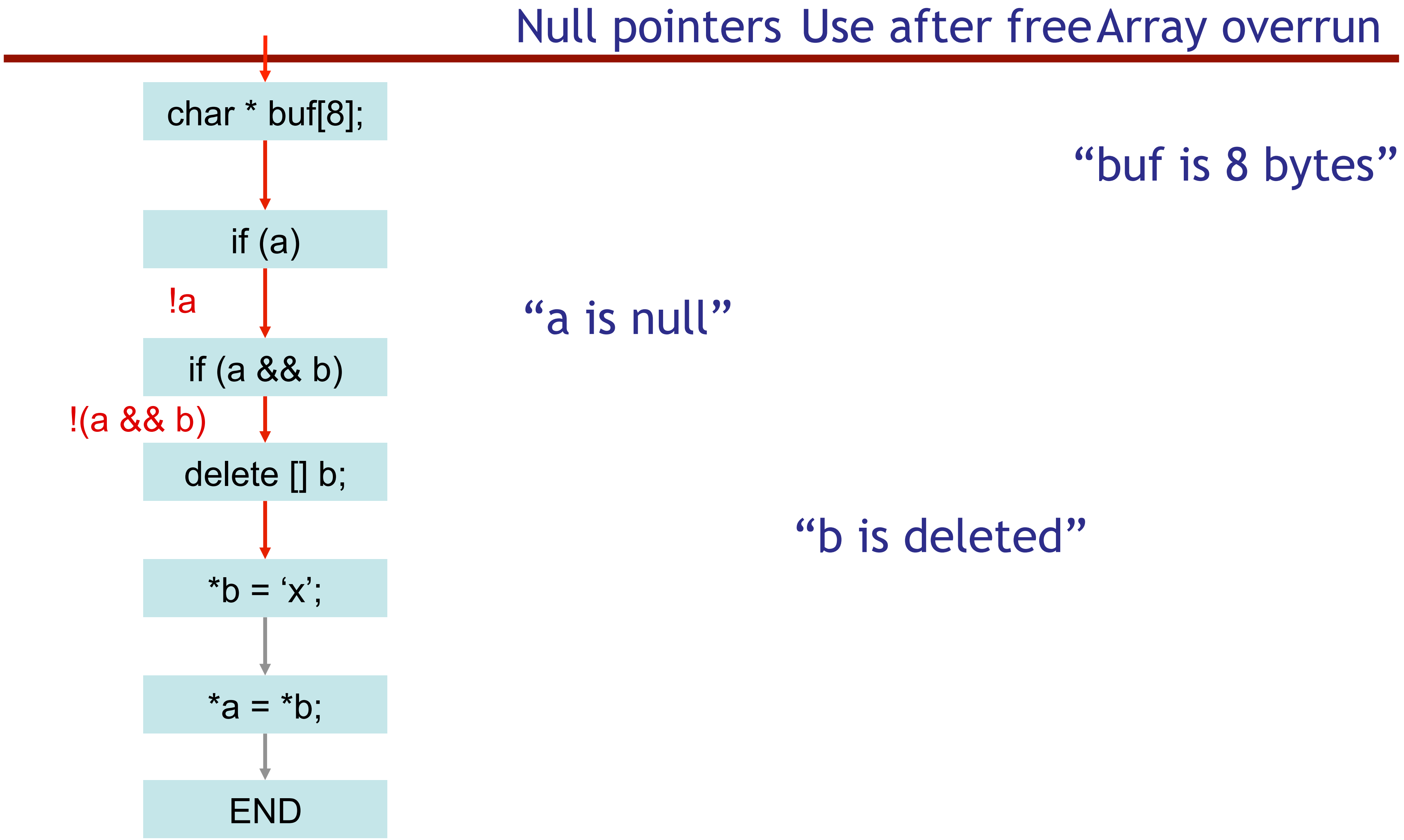


“buf is 8 bytes”

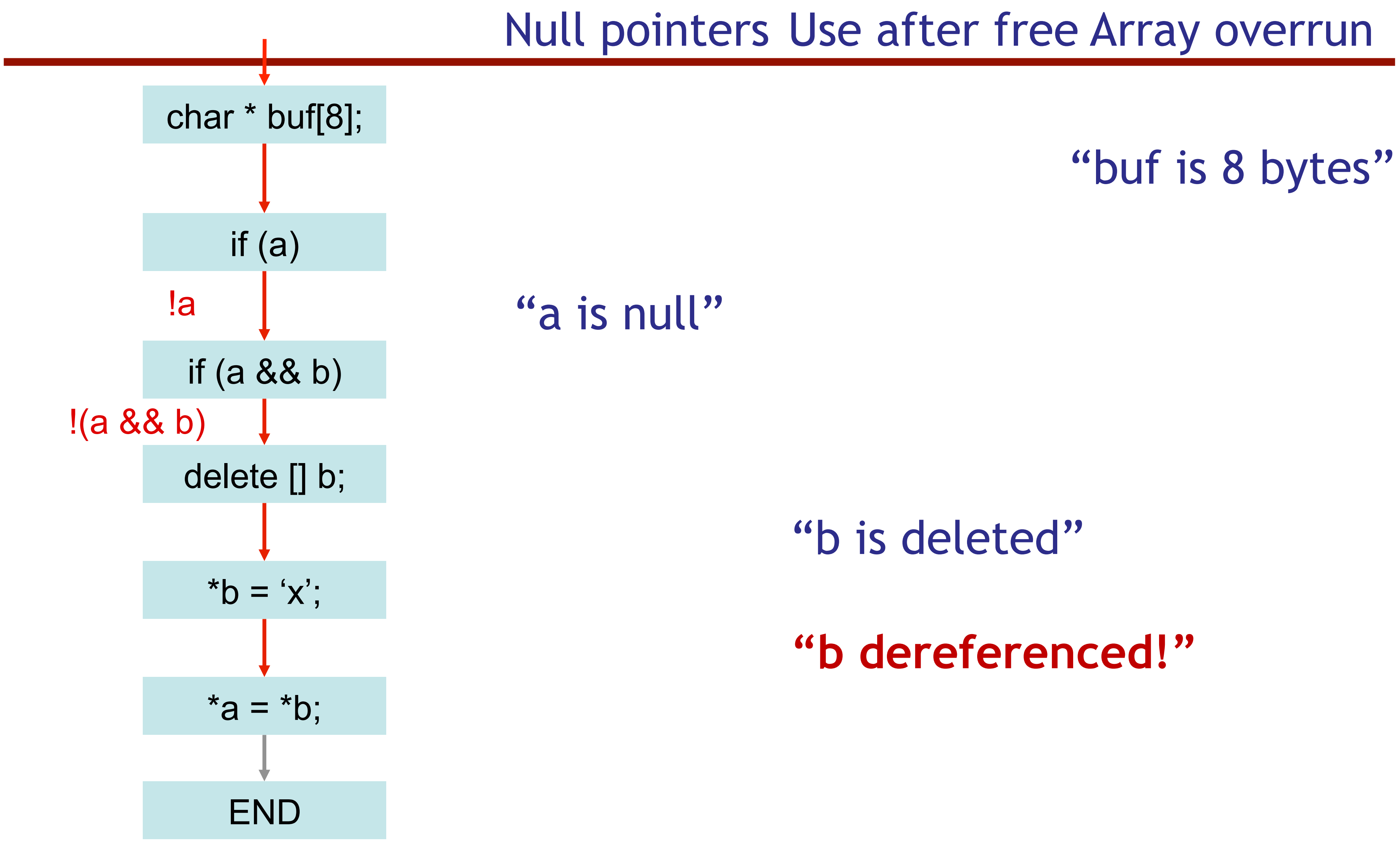
“a is null”

Already knew a was null

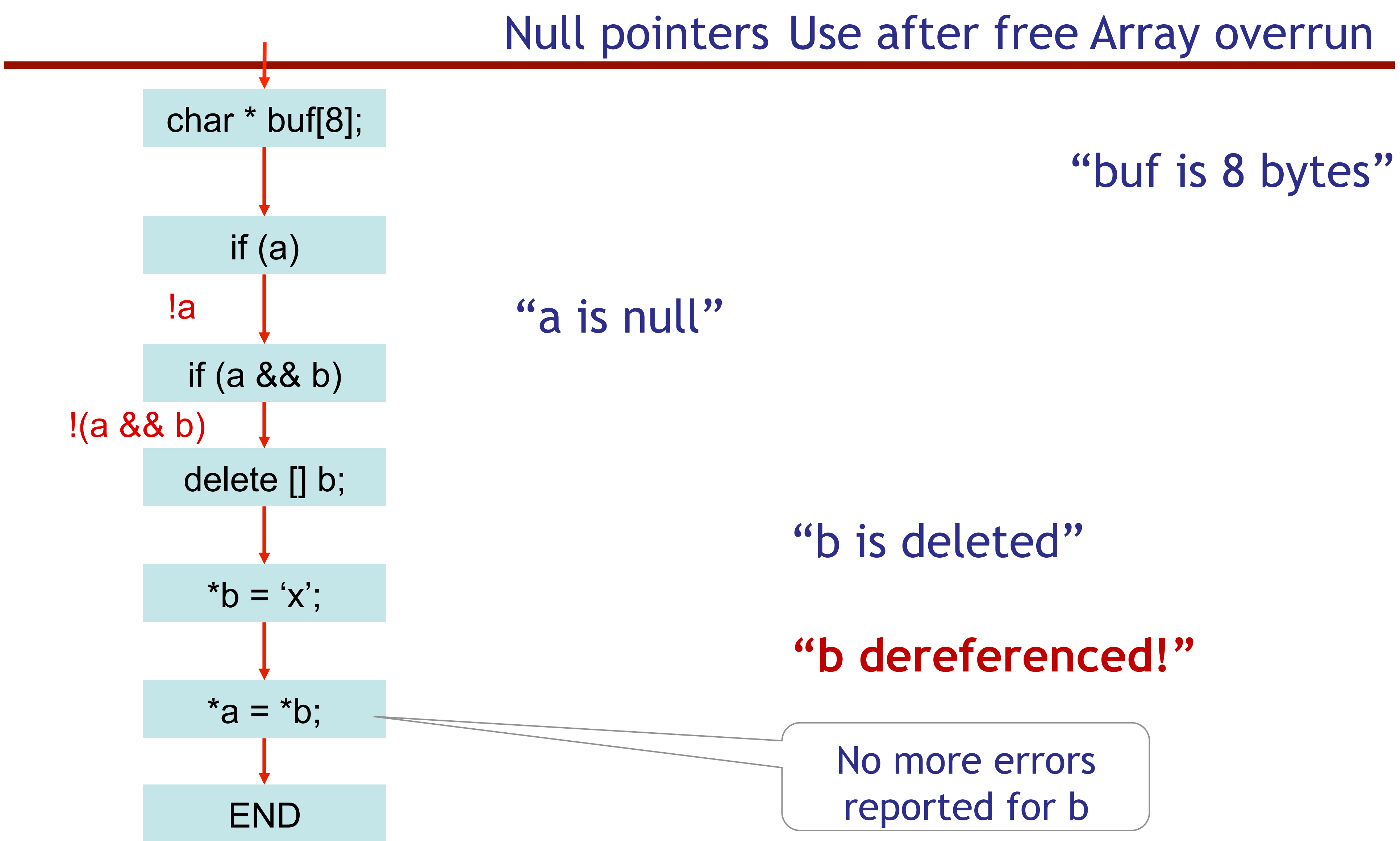
Apply Checking



Apply Checking



Apply Checking

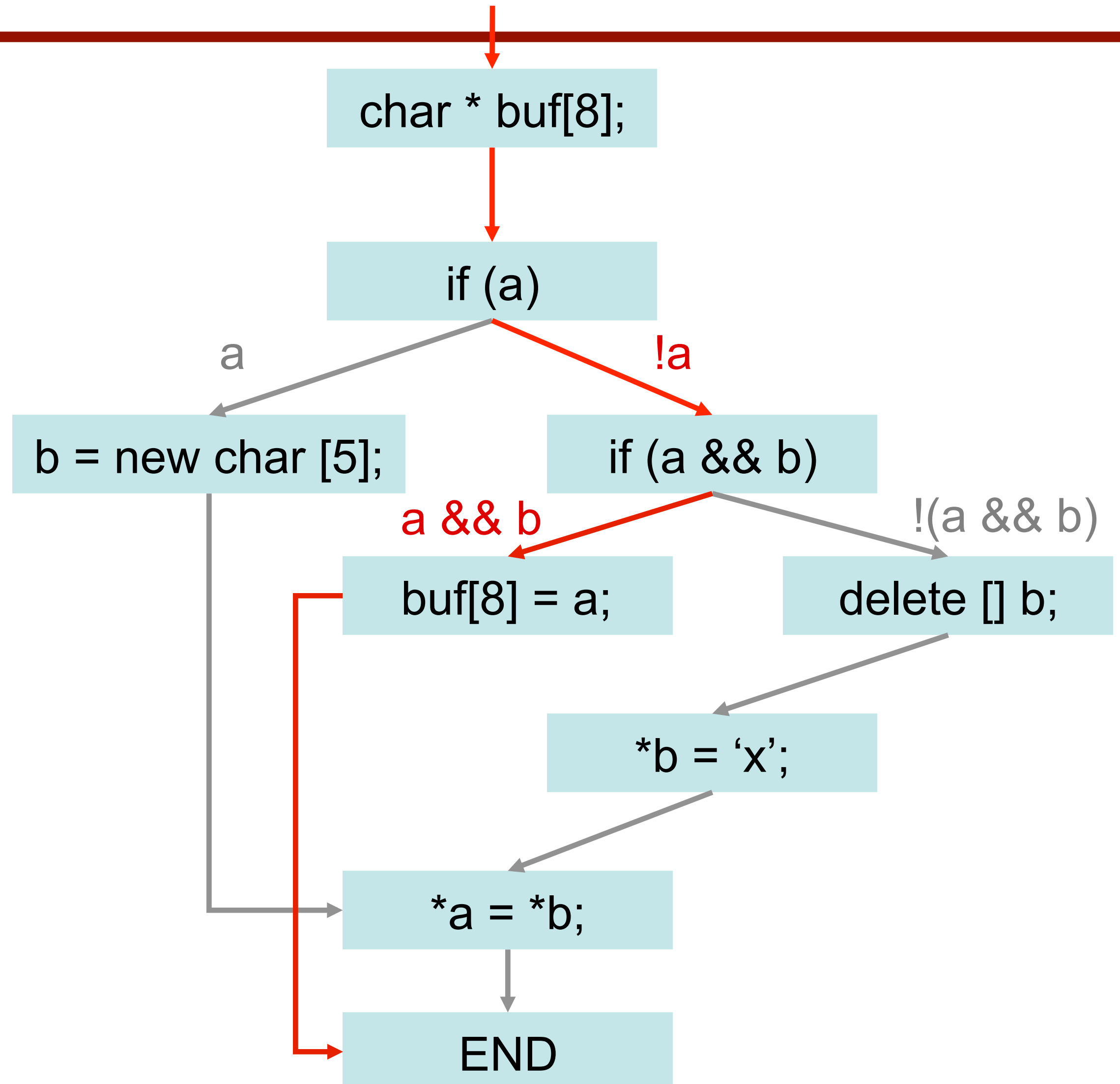


False Positives

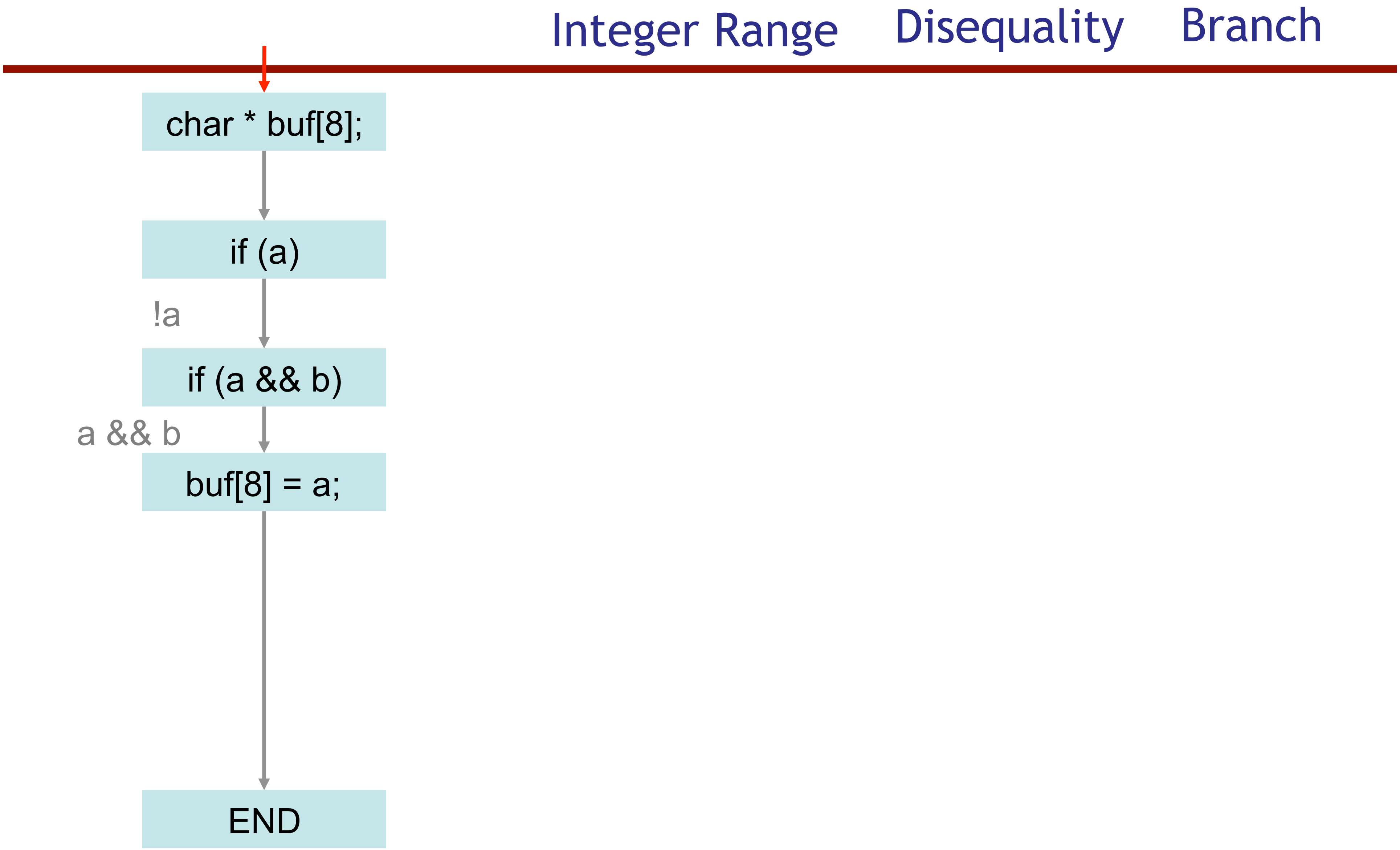
- **What is a bug? Something the user will fix.**

- **Many sources of false positives**
  - False paths
  - Idioms
  - Execution environment assumptions
  - Killpaths
  - Conditional compilation
  - “third party code”
  - Analysis imprecision
  - …

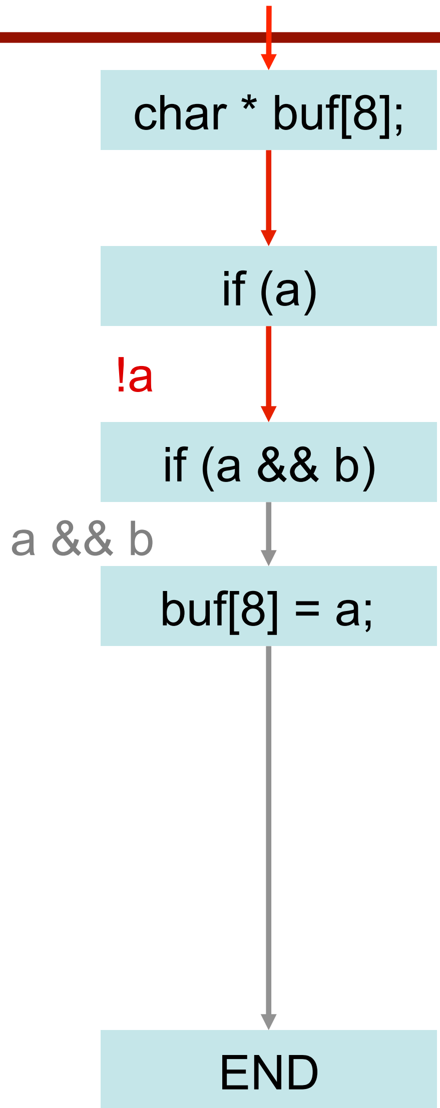
A False Path



False Path Pruning

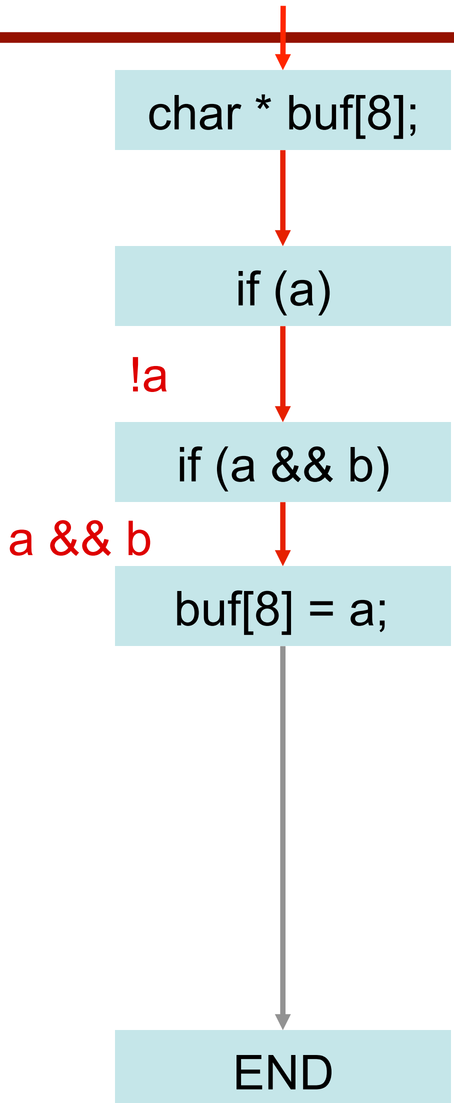


False Path Pruning



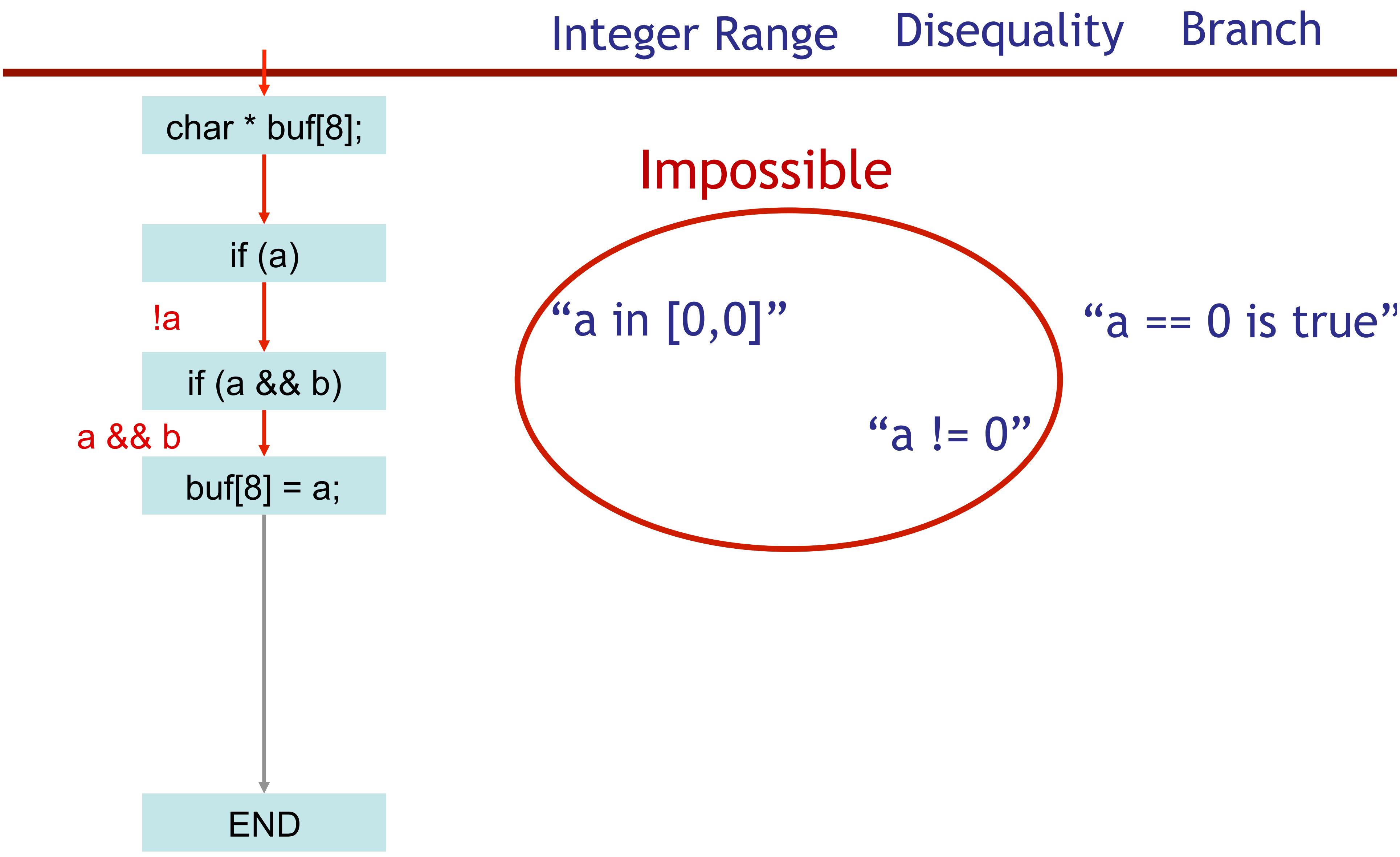
	Integer Range	Disequality	Branch
!a	"a in [0,0]"		"a == 0 is true"

False Path Pruning



Integer Range	Disequality	Branch
“a in [0,0]”		“a == 0 is true”
	“a != 0”	

False Path Pruning



Environment Assumptions

- **Should the return value of malloc() be checked?**

```
int *p = malloc(sizeof(int));
*p = 42;
```

OS Kernel:
Crash machine.

File server:
Pause filesystem.

Web application:
200ms downtime

Spreadsheet:
Lose unsaved changes.

Game:
Annoy user.

IP Phone:
Annoy user.

Library:
?

Medical device:
malloc?!

Statistical Analysis

- **Assume the code is usually right**

3/4 deref

```
int *p = malloc(sizeof(int));
*p = 42;

int *p = malloc(sizeof(int));
*p = 42;

int *p = malloc(sizeof(int));
*p = 42;

int *p = malloc(sizeof(int));
if(p)  *p = 42;
```

```
int *p = malloc(sizeof(int));
if(p) *p = 42;

int *p = malloc(sizeof(int));
if(p) *p = 42;

int *p = malloc(sizeof(int));
if(p) *p = 42;

int *p = malloc(sizeof(int));
*p = 42;
```

1/4 deref

Example security holes

- **Remote exploit, no checks**

```
/* 2.4.9/drivers/isdn/act2000/capi.c:actcapi_dispatch */
isdn_ctrl cmd;
...
while ((skb = skb_dequeue(&card->rcvq))) {
   msg = skb->data;
   ...
   memcpy(cmd.parm.setup.phone,
          msg->msg.connect_ind.addr.num,
          msg->msg.connect_ind.addr.len - 1);
```

Example security holes

- **Missed lower-bound check:**

```
/* 2.4.5/drivers/char/drm/i810_dma.c */

if(copy_from_user(&d, arg, sizeof(arg)))
   return -EFAULT;
if(d.idx > dma->buf_count)
   return -EINVAL;
buf = dma->buflist[d.idx];
Copy_from_user(buf_priv->virtual, d.address, d.used);
```

Summary

- Static vs dynamic analyzers
- General properties of static analyzers
  - Fundamental limitations
  - Basic method based on abstract states
- More details on one specific method
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for Android malware
  - STAMP method, sample studies

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …