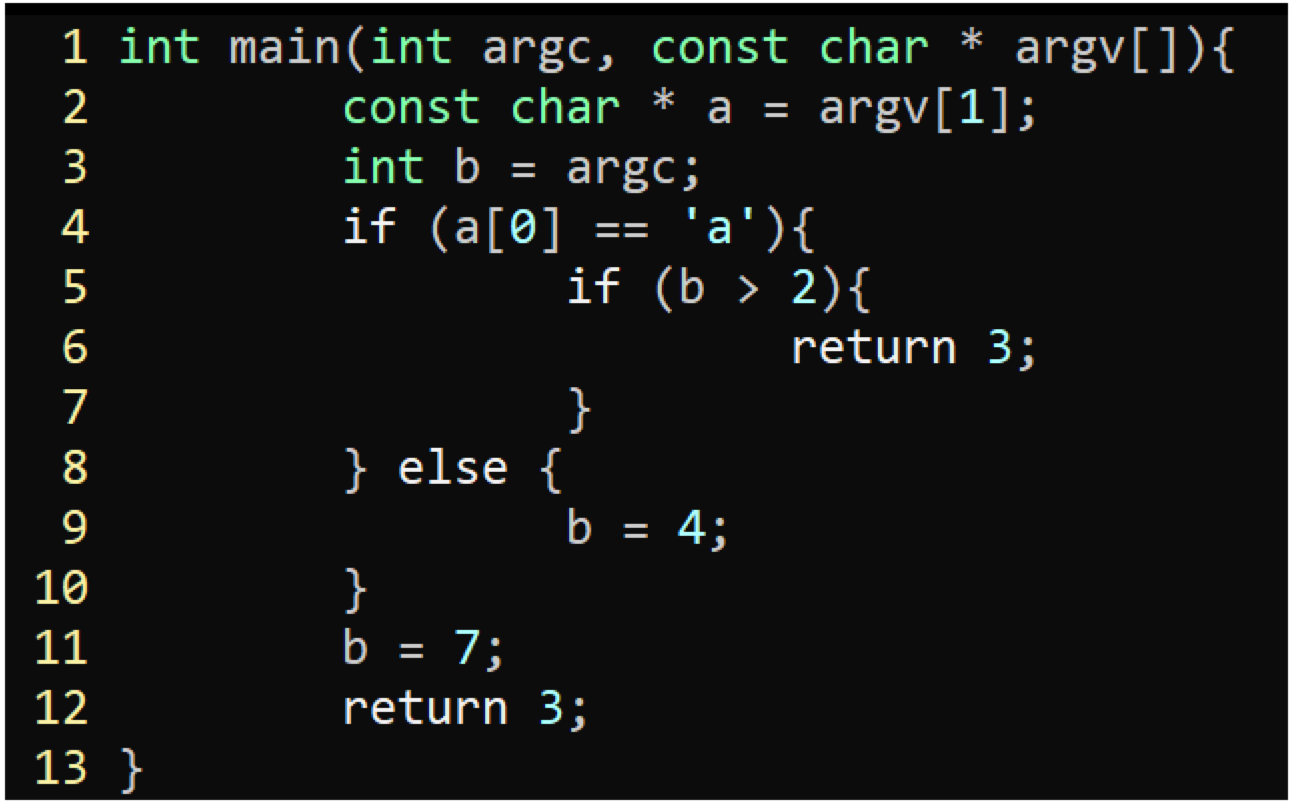
EXERCISE 17

PROGRAM SLICING REVIEW

Write your name and answer the following on a piece of paper

Draw the forward slice from line 2 in following program:



```
 1 int main(int argc, const char * argv[]){
 2         const char * a = argv[1];
 3         int b = argc;
 4         if (a[0] == 'a'){
 5                 if (b > 2){
 6                         return 3;
 7                 }
 8         } else {
 9                 b = 4;
10         }
11         b = 7;
12         return 3;
13 }
```

EXERCISE 17: SOLUTION

PROGRAM SLICING REVIEW

ADMINISTRIVIA AND ANNOUNCEMENTS

LAST TIME: THE PROGRAM SLICE

REVIEW: PROGRAM SLICING

EXTRACT A SUB-PROGRAM OF INTEREST BASED ON ONE (OR MORE) STATEMENTS

Forward slice

Capture all code *influenced by* a given statement

Backwards slide

Capture all code *influencing* a given statement

CONSTRUCTING THE SLICE

REVIEW: PROGRAM SLICING

Extract the Control-Flow Graph (CFG)
Construct Basic Blocks, make control transfers edges

Extract the Postdominator Tree from the CFG
(done via a dataflow analysis)

Capture the IFDOM relationship
Backwards edges in the postdominator tree

Build the Control-Dependence Graph (CDG)
Backwards edges in the postdominator tree

Build the Data-Dependence Graph (DDG)
Backwards edges in the reaching definitions

Build the Control-Dependence Graph (PDG)
Add all edges from the DDG to the CDG

Construct the transitive closure of PDG edges
Forward: against dependence, Backwards: with dependence

USES FOR PROGRAM SLICES

REVIEW: PROGRAM SLICING

Program Comprehension
What is this statement doing?

Debugging
(done via a dataflow analysis)

Scaling heavyweight analysis
Less program to test

VIBE CHECK

DATAFLOW ANALYSIS IS SUPER USEFUL!

We’ve achieved a milestone in our analysis

We did leave out a handful of program features...

- Functions
- Global Variables
- Classes / Dynamic Dispatch
- Pointers / References

IS THIS STUFF USEFUL?

VIBE CHECK

EMPIRICALLY, YES

Windows PREfast and Static Driver Verifier

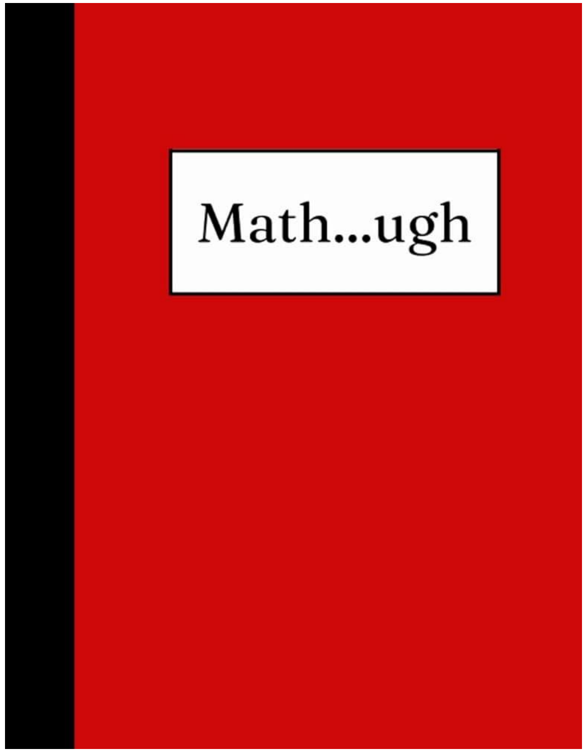
Even shipped with some versions of Visual Studio

PREfast for Drivers: only analyzes a single function

Official bug numbers are hard to come by, but anecdotally they have been crucial in reducing Windows DOS

Coverity

Static analysis tool originally from Stanford



IS THIS STUFF USEFUL?

VIBE CHECK

Empirically, Yes

Windows PREfast and Static Driver Verifier

Even shipped with some versions of Visual Studio

PREfast for Drivers: only analyzes a single function

Official bug numbers are hard to come by, but anecdotally they have been crucial in reducing Windows DOS

Coverity

Static analysis tool originally from Stanford

> Under a United States Department of Homeland Security contract in 2006, the tool was used to examine over 150 open source applications for bugs; 6000 bugs found by the scan were fixed across 53 projects.[4] National Highway Traffic Safety Administration used the tool in its 2010-2011 investigation into reports of sudden unintended acceleration in Toyota vehicles.[5][6] The tool was used by CERN on the software employed in the Large Hadron Collider[7][8] and in the NASA Jet Propulsion Laboratory during the flight software development of the Mars rover *Curiosity*.[9]
>
> \- Wikipedia

INTERPROCEDURAL ANALYSIS

EECS 677: Software Security Evaluation

Drew Davidson

SCALING UP OUR ANALYSIS

INTERPROCEDURAL ANALYSIS

INTRAPROCEDURAL ANALYSIS IS USEFUL!

PREfast Driver shows the importance in special cases

INTERPROCEDURAL ANALYSIS IS USEFUL!

Coverity shows the importance in more general cases

LECTURE OUTLINE

- Abject Pessimism
- ICFGs
- Context-Sensitivity
- Summary Functions

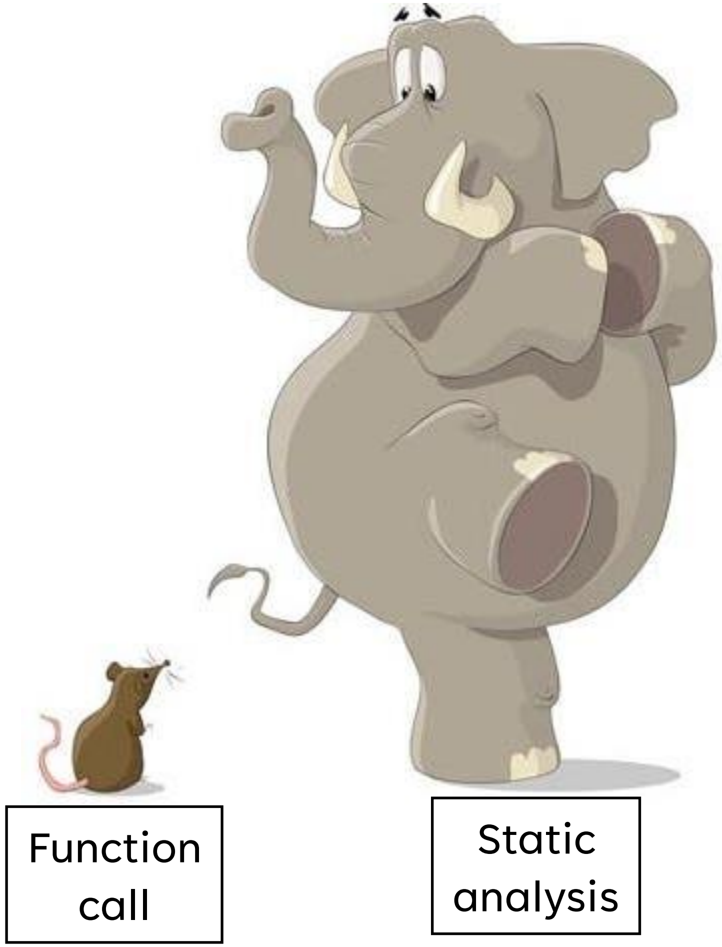
WORST-CASE ASSUMPTIONS

NAÏVE APPROACH

CREATE SIMPLE, "SAFE" OVER-APPROXIMATION

What constitutes "being safe" depends on your analysis

- **Example 1, confidentiality:** Assume a function call tags all reachable data as confidential
- **Example 2, integrity:** Assume a function call tags all reachable data as untrusted



Function call

Static analysis

WORST-CASE ASSUMPTIONS

NAÏVE APPROACH

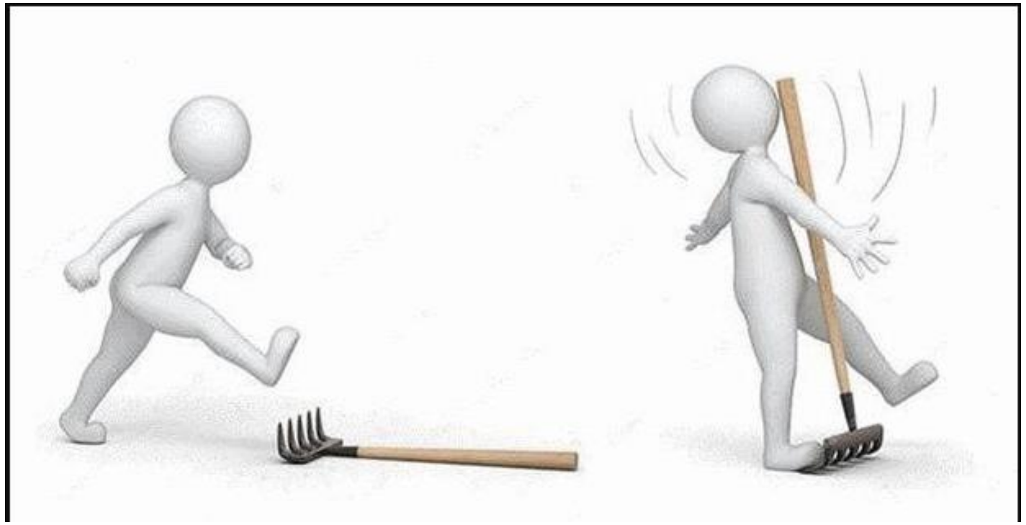
Our general philosophy: "Do no harm" guarantees

Recall our notions of soundness and completeness:
- Sound: no false positives ("tells no lie")
- Complete: no false negatives ("omits no truth")

"Being safe" Requires formulating analysis goal

bug hunting:
- Report buggy programs
- Safe means complete analysis

program verification:
- Report clean programs
- Safe means sound analysis



Anything that **CAN** go wrong **WILL** go wrong

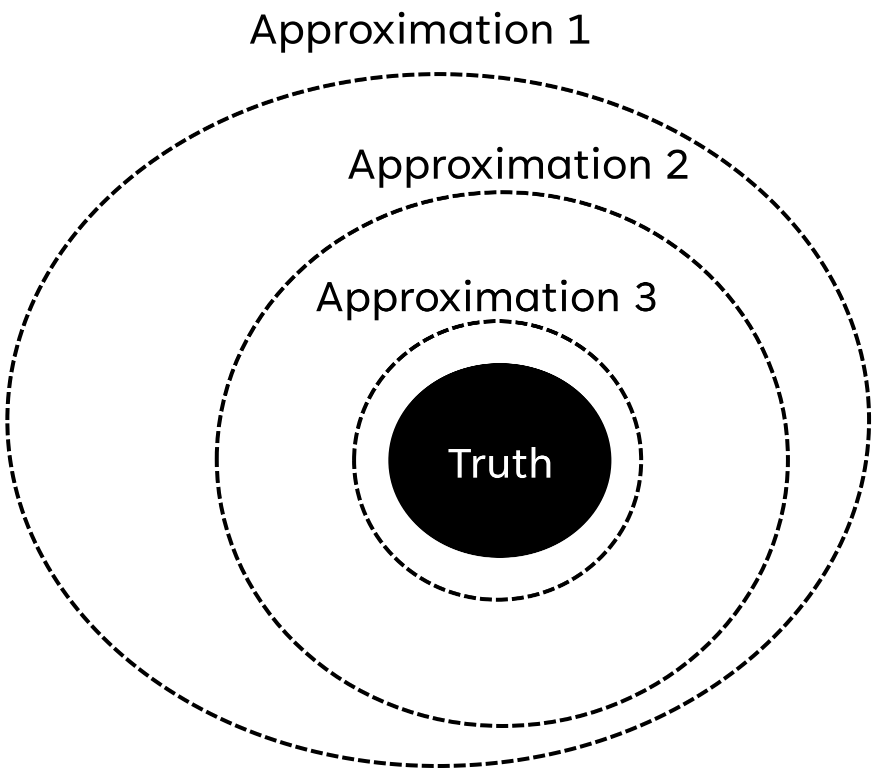
- Murphy's Law

TIGHTENING THE BOUNDS

NAÏVE APPROACH

ATTEMPT TO GET TIGHTER AND TIGHTER BOUNDS RETAINING COMPLETENESS

Address areas of imprecision that are only adding false positives.



LECTURE OUTLINE

- Abject Pessimism
- ICFGs
- Context-Sensitivity
- Summary Functions

THE OBVIOUS INTERPROCEDURAL SOLUTION

INTERPROCEDURAL ANALYSIS: ICFGS

JUST ADD EDGES FROM A CALL SITE TO THE CALLEE

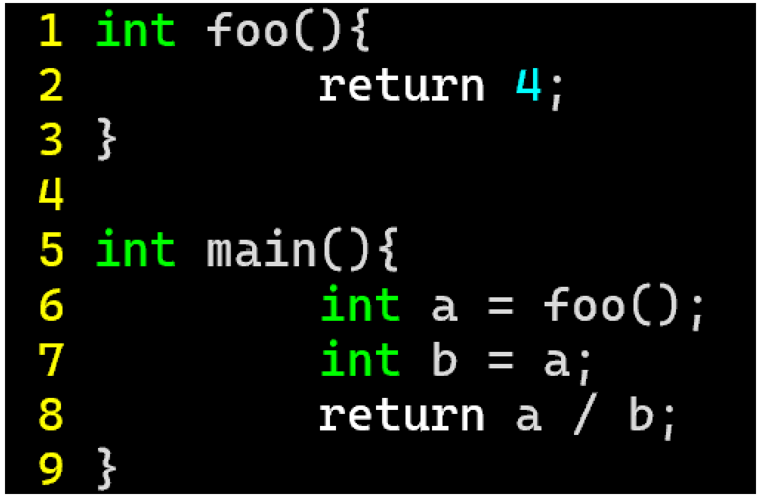
Builds the interprocedural control flow graph (ICFG) aka “supergraph”

THE OBVIOUS INTERPROCEDURAL SOLUTION

INTERPROCEDURAL ANALYSIS: ICFGS

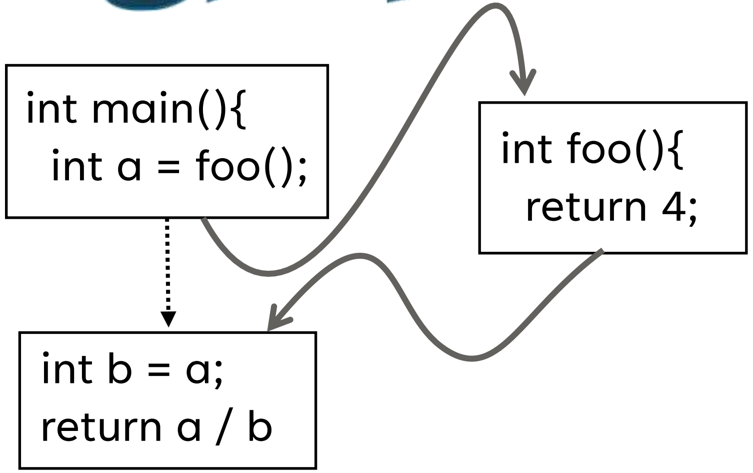
JUST ADD EDGES FROM A CALL SITE TO THE CALLEE

Builds the interprocedural control flow graph (ICFG) aka "supergraph"



```
int foo(){
        return 4;
}

int main(){
        int a = foo();
        int b = a;
        return a / b;
}
```



COST/BENEFIT OF SUPERGRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

Benefits of ICFGs

Better than abject pessimism!

Minimal modification to intraprocedural algorithms

Costs of ICFGs

May not be obvious what the callee is

Naïvely leads to some erroneous paths

COST/BENEFIT OF SUPERGRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

COSTS OF ICFGS

May not be obvious what the callee is

We can separate that concern into a call graph analysis

CALL GRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

COSTS OF ICFGS

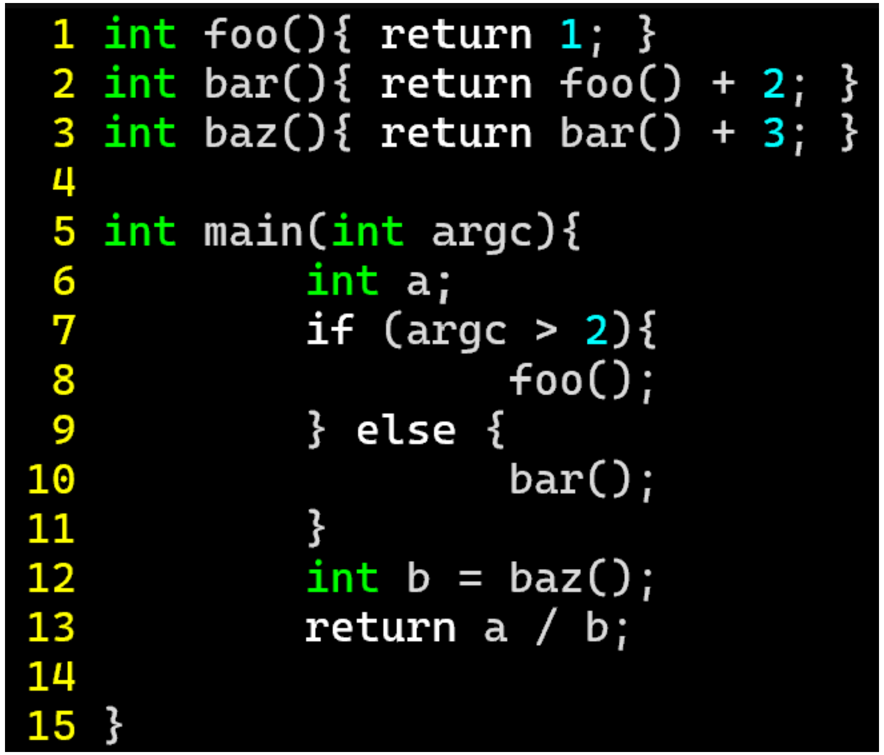
May not be obvious what the callee is

We can separate that concern into a call graph analysis

Simple call graph:

connect caller functions to callee functions

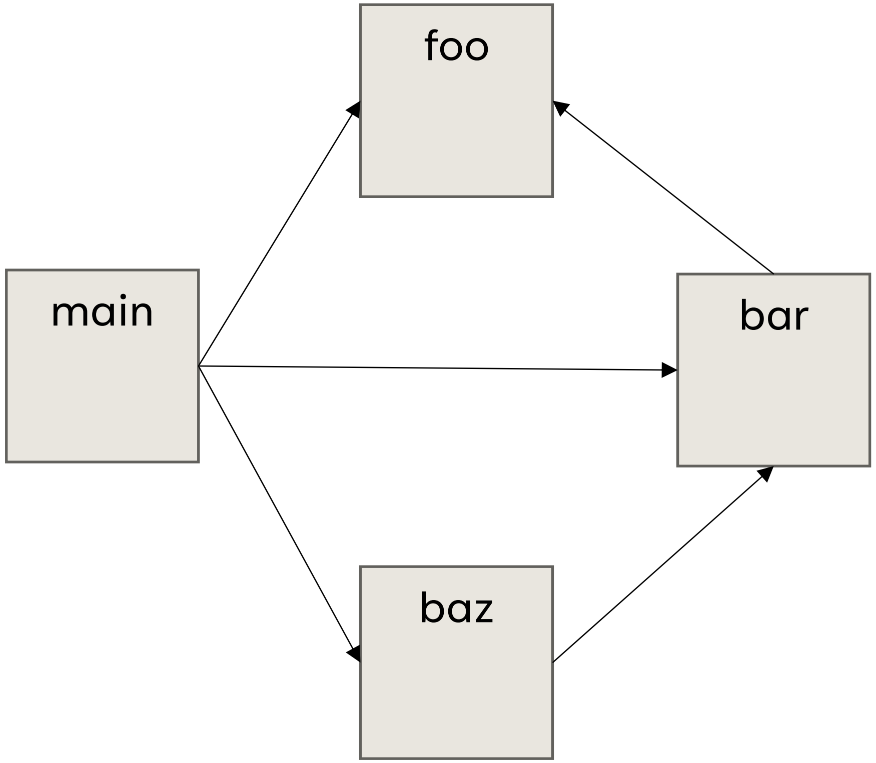
- Node: function
- Edge: function call



```
int foo(){ return 1; }
int bar(){ return foo() + 2; }
int baz(){ return bar() + 3; }

int main(int argc){
        int a;
        if (argc > 2){
                foo();
        } else {
                bar();
        }
        int b = baz();
        return a / b;

}
```



CALL GRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

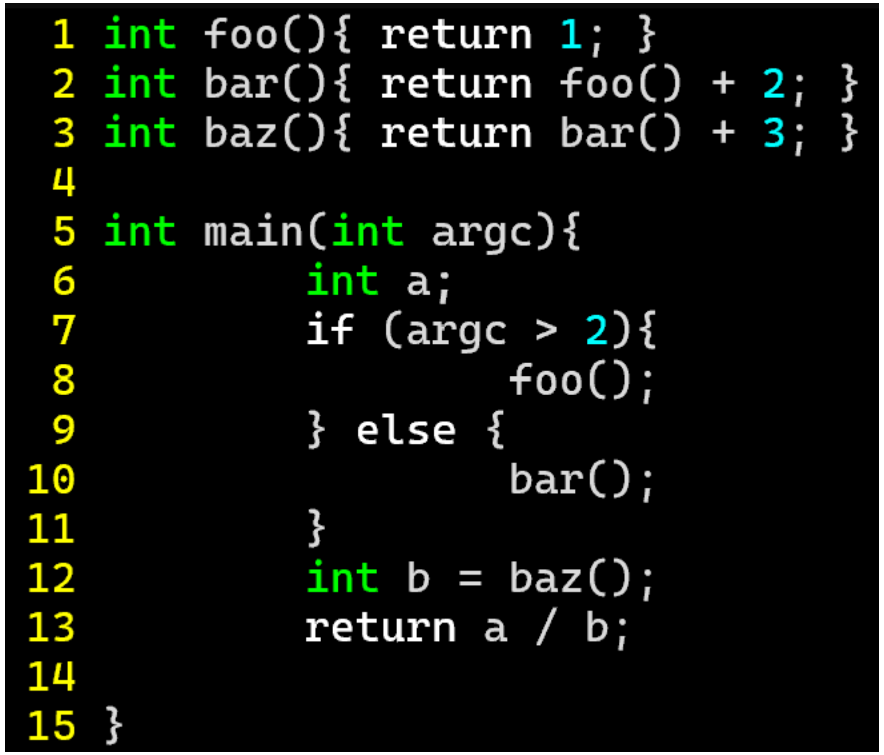
Costs of ICFGs

May not be obvious what the callee is

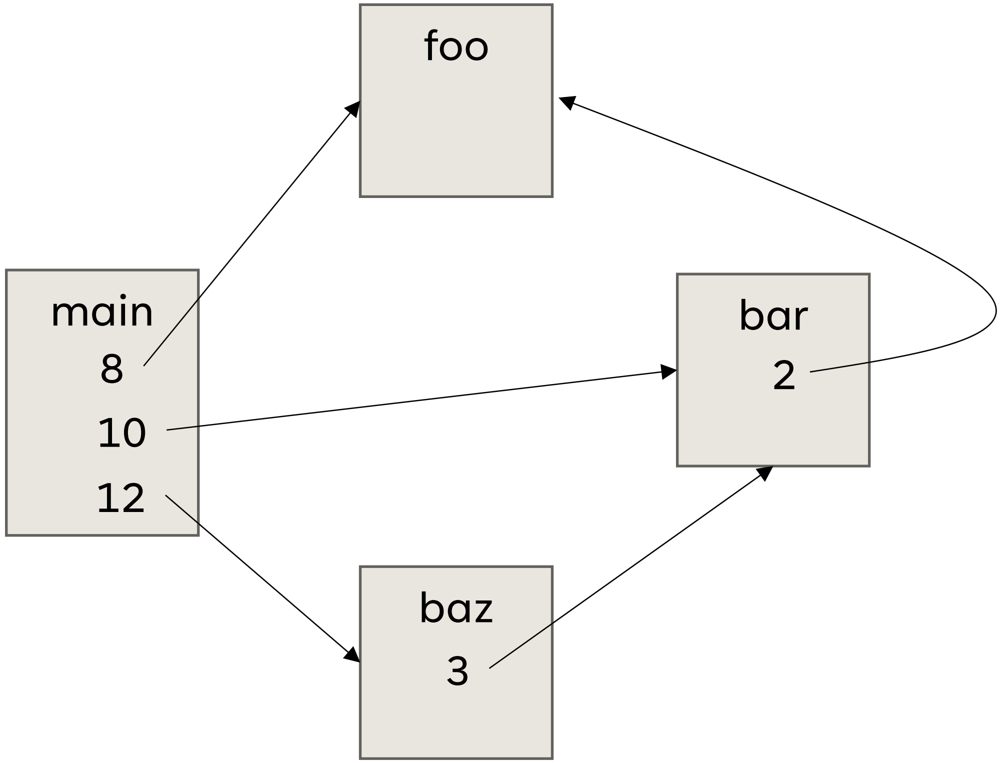
We can separate that concern into a call graph analysis

Better ~~Simple~~ call graph: Call sites connect ~~caller functions~~ to callee functions

- Node: function **with nested call sites**
- Edge: function call



```
 1 int foo(){ return 1; }
 2 int bar(){ return foo() + 2; }
 3 int baz(){ return bar() + 3; }
 4
 5 int main(int argc){
 6         int a;
 7         if (argc > 2){
 8                 foo();
 9         } else {
10                 bar();
11         }
12         int b = baz();
13         return a / b;
14
15 }
```



CALL GRAPHS

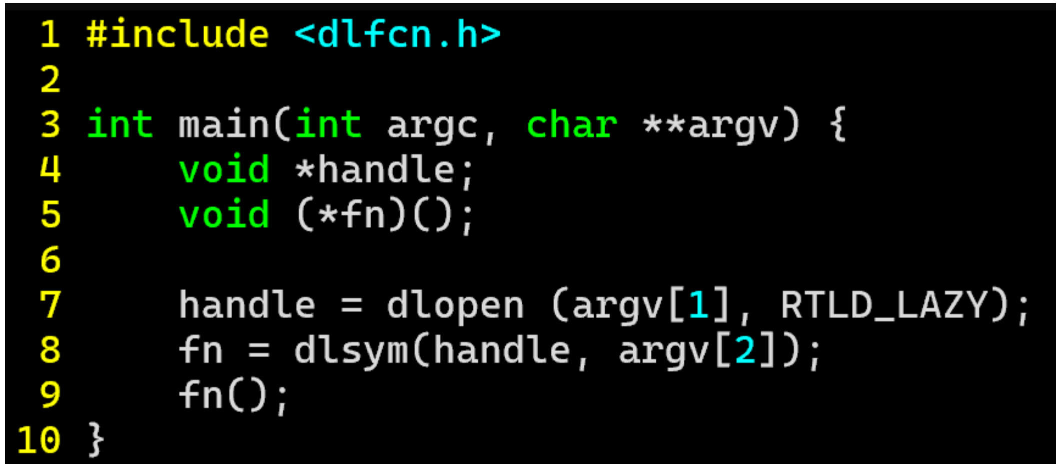
INTERPROCEDURAL ANALYSIS: ICFGS

Costs of ICFGs

May not be obvious what the callee is

We can separate that concern into a call graph analysis

In the case of imprecision, over-approximate behaviors



```
#include <dlfcn.h>

int main(int argc, char **argv) {
    void *handle;
    void (*fn)();

    handle = dlopen (argv[1], RTLD_LAZY);
    fn = dlsym(handle, argv[2]);
    fn();
}
```

THE OTHER THING ABOUT SUPERGRAPHS

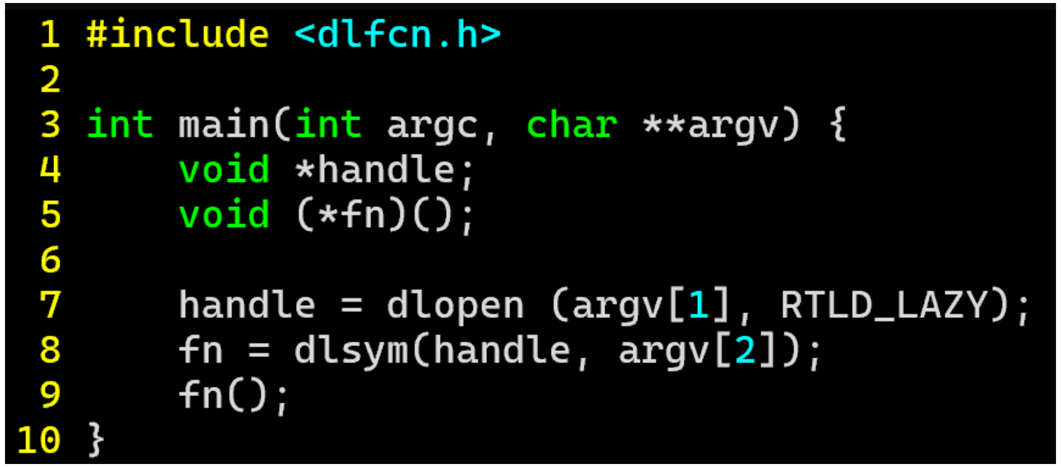
INTERPROCEDURAL ANALYSIS: ICFGS

Costs of ICFGs

May not be obvious what the callee is

We can separate that concern into a call graph analysis

In the case of imprecision, over-approximate behaviors



```c
#include <dlfcn.h>

int main(int argc, char **argv) {
    void *handle;
    void (*fn)();

    handle = dlopen (argv[1], RTLD_LAZY);
    fn = dlsym(handle, argv[2]);
    fn();
}
```

THE OTHER THING ABOUT SUPERGRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

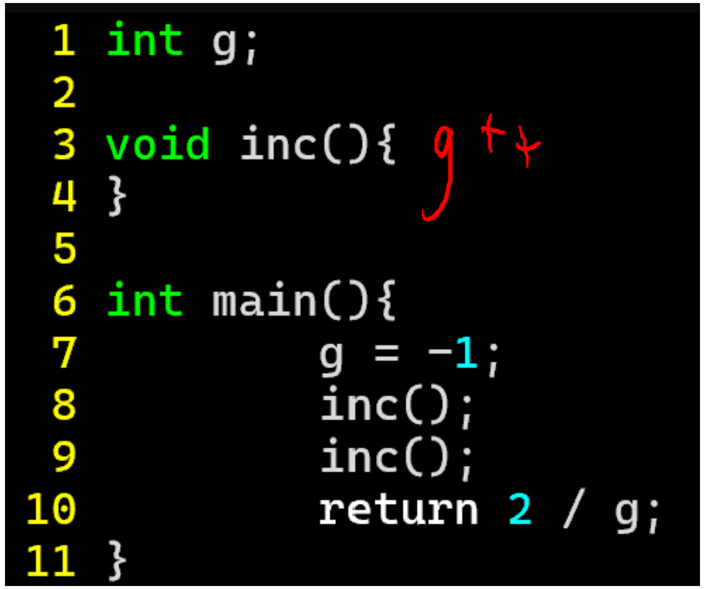
COSTS OF ICFGS

May not be obvious what the callee is

We can separate that concern into a call graph analysis

In the case of imprecision, over-approximate behaviors

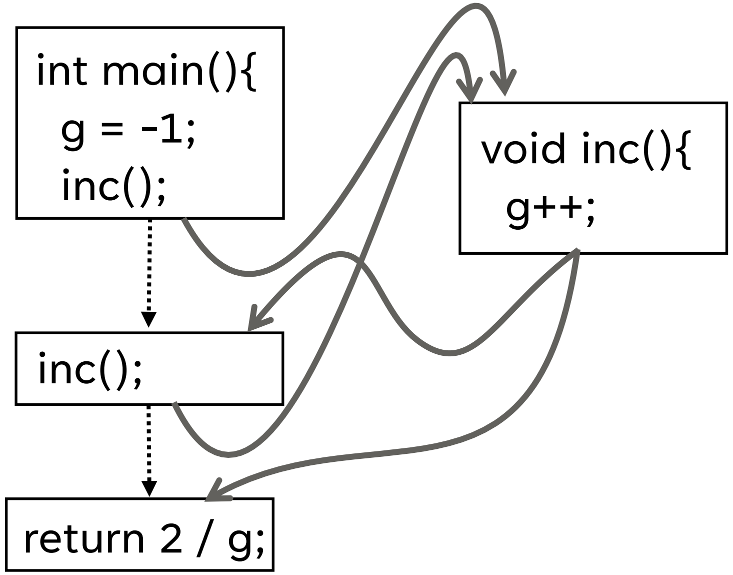
Naïvely leads to some impossible paths



```
int g;

void inc(){ g++
}

int main(){
        g = -1;
        inc();
        inc();
        return 2 / g;
}
```



int main(){
  g = -1;
  inc();

inc();

return 2 / g;

void inc(){
  g++;

THE OTHER THING ABOUT SUPERGRAPHS

INTERPROCEDURAL ANALYSIS: ICFGS

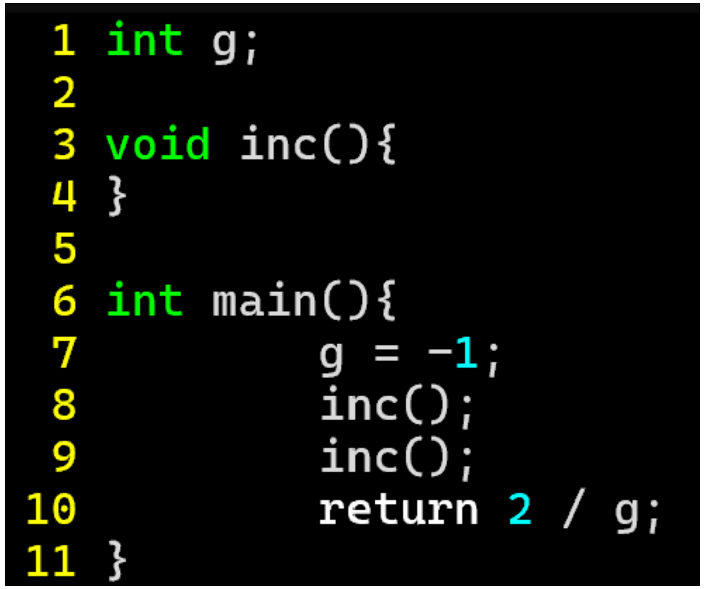
Costs of ICFGs

May not be obvious what the callee is

We can separate that concern into a call graph analysis

In the case of imprecision, over-approximate behaviors

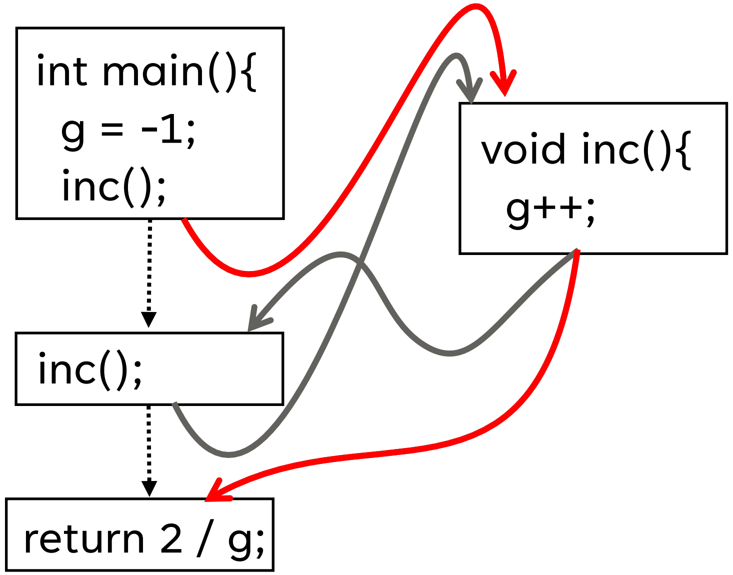
Naïvely leads to some impossible paths



```
int g;

void inc(){
}

int main(){
        g = -1;
        inc();
        inc();
        return 2 / g;
}
```



int main(){
  g = -1;
  inc();

void inc(){
  g++;

inc();

return 2 / g;

LECTURE OUTLINE

- Abject Pessimism
- ICFGs
- Context-Sensitivity
- Summary Functions

(CALLING) CONTEXT SENSITIVITY

INTERPROCEDURAL ANALYSIS: ICFGS

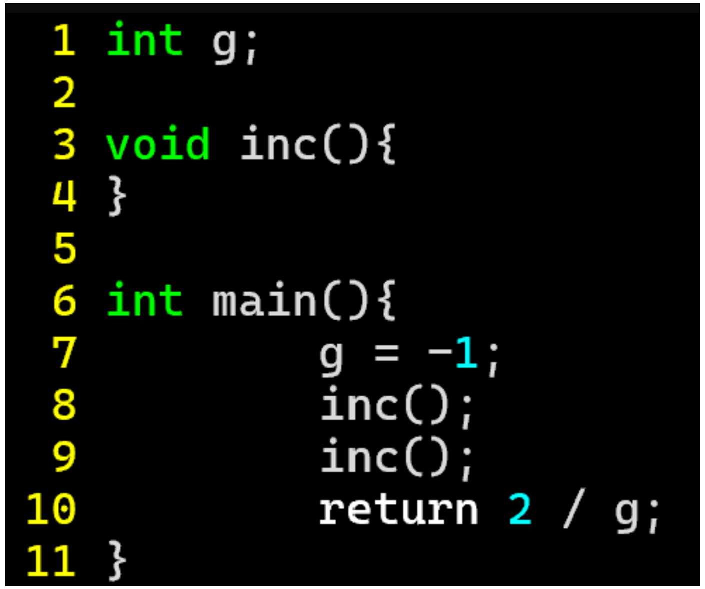
THE PROBLEM IN THE SIMPLE SUPERGRAPH ANALYSIS

A lock of calling context (return to the wrong call site)

This provides another way to “tune” a flow analysis

- Flow-sensitive vs Flow-insensitive
- Context-sensitive vs Context-insensitive

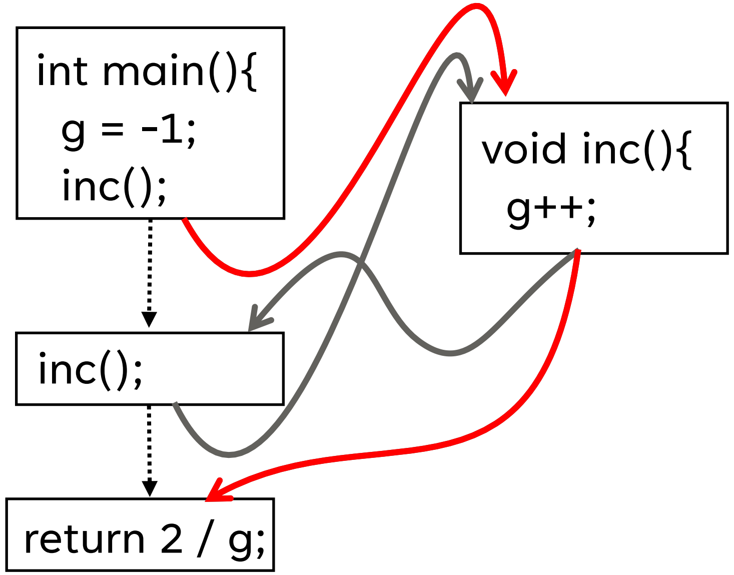
tracks some amount of context



```
int g;

void inc(){
}

int main(){
        g = -1;
        inc();
        inc();
        return 2 / g;
}
```



CALL STRINGS AND K-CFA

INTERPROCEDURAL ANALYSIS: ICFGS

How much context to Keep?

Obvious problem: can’t distinguish between callers (context-insensitive analysis)

Obvious solution: Call strings

Keep track of the caller 1-CFA

Obvious problem: what about the caller’s caller?

Obvious solution: keep track of the caller’s caller? 2-CFA

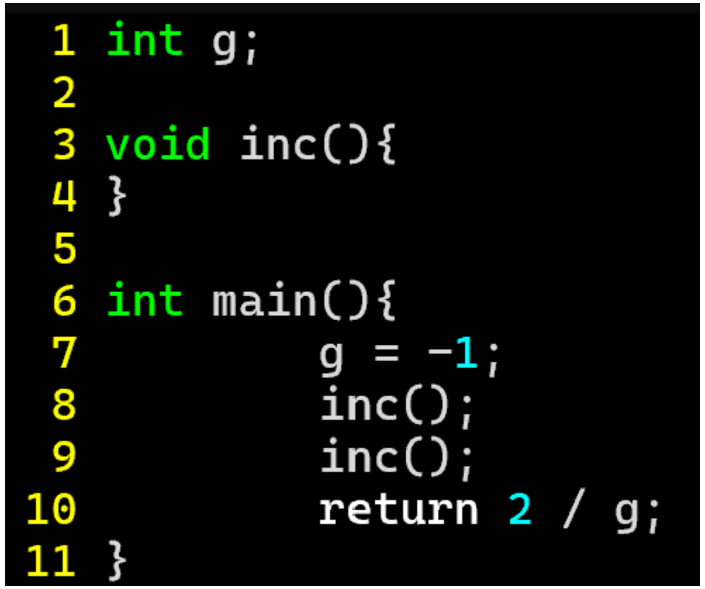
> “k-CFA popularized the idea of context-sensitive flow analysis. […] in the OO setting, where a 1- and 2-CFA analysis is considered heavy but certainly possible”
>
> - Might et al, Resolving and Exploiting the k-CFA Paradox

ANOTHER FORM OF CONTEXT SENSITIVITY

INTERPROCEDURAL ANALYSIS: CONTEXT-SENSITIVITY

A (PERHAPS) MORE CONCEPTUALLY STRAIGHTFORWARD APPROACH…

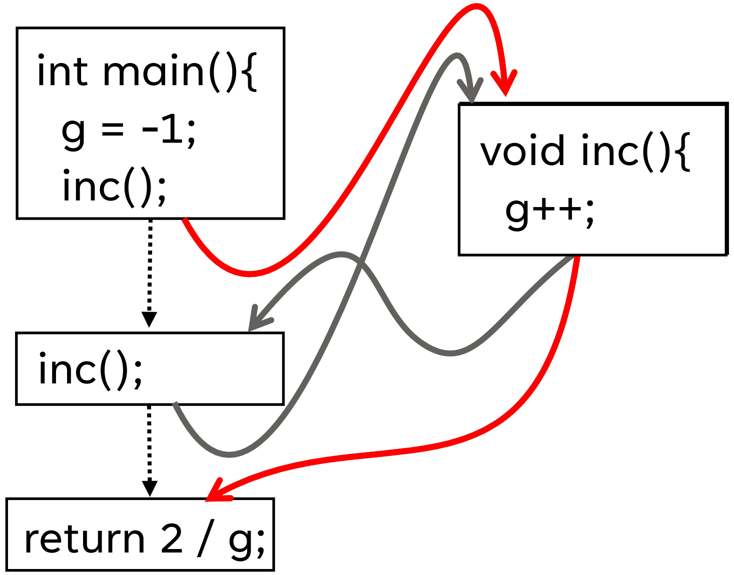
Rather than complicating the edges, what if we cloned the nodes



```
int g;

void inc(){
}

int main(){
        g = -1;
        inc();
        inc();
        return 2 / g;
}
```



ANOTHER FORM OF CONTEXT SENSITIVITY

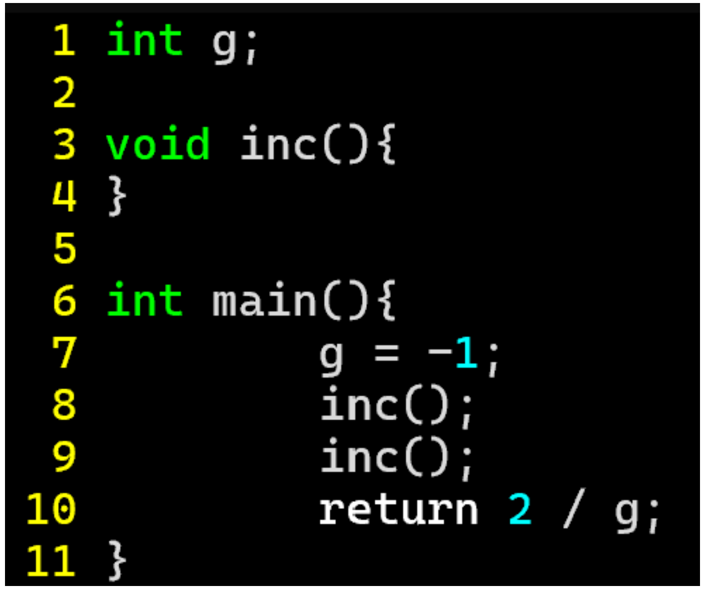
INTERPROCEDURAL ANALYSIS: CONTEXT-SENSITIVITY

A (PERHAPS) MORE CONCEPTUALLY STRAIGHTFORWARD APPROACH…

Rather than complicating the edges, what if we cloned the nodes

“Exploded supergraph”: 1 clone per static call site

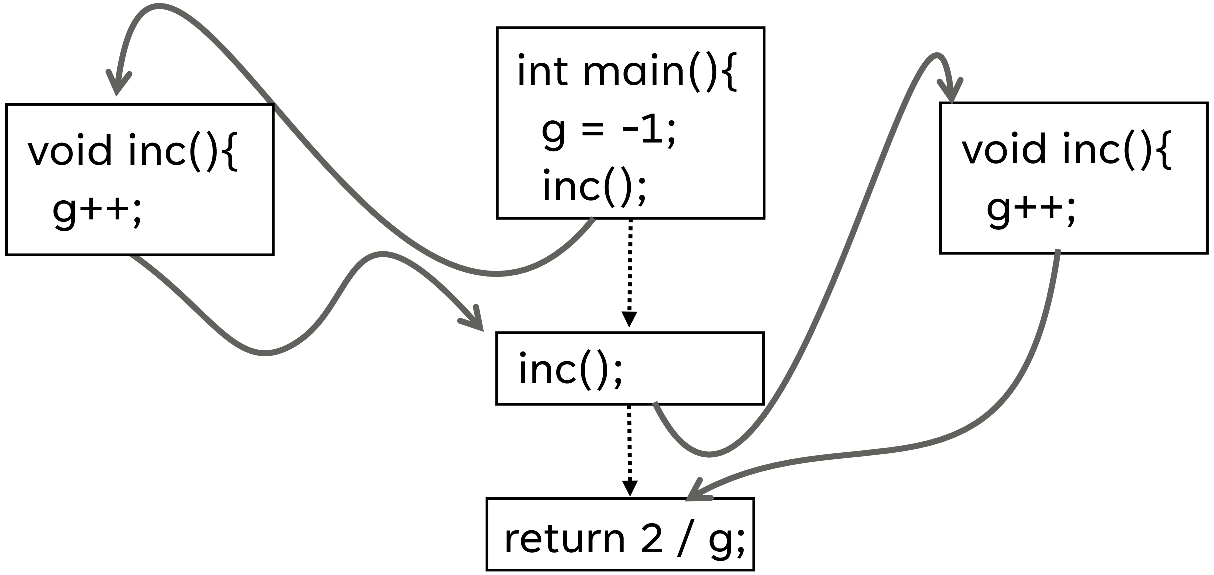
Still very much not foolproof



```
int g;

void inc(){
}

int main(){
        g = -1;
        inc();
        inc();
        return 2 / g;
}
```



LECTURE OUTLINE

- Abject Pessimism
- ICFGs
- Context-Sensitivity
- Summary Functions

SUMMARY FUNCTIONS

SUPERGRAPHS

BIG IDEA

Summarize callee analysis (rather than include it in the analysis)

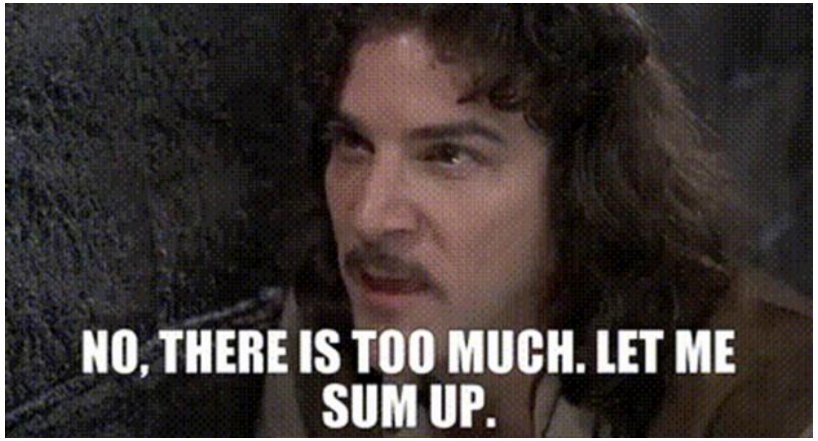
MANUAL MANIFESTATION

Ask the user to provide information

AUTOMATIC MANIFESTATION

Create a lightweight inference

- What variables are (transitively) modified as a result of a function call? GMOD
- What variables are (transitively) referenced as a result of a function call? GREF



WRAP-UP