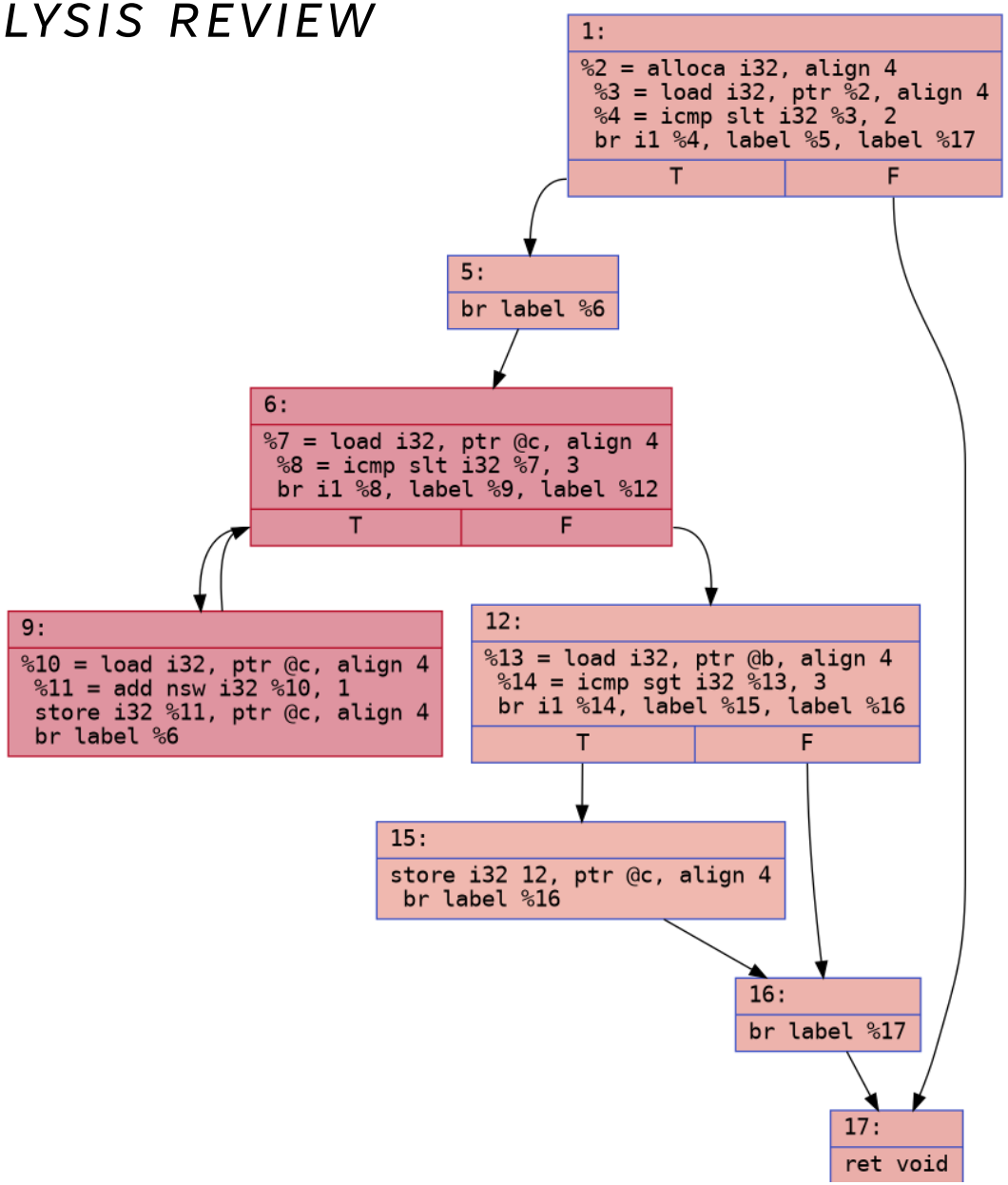
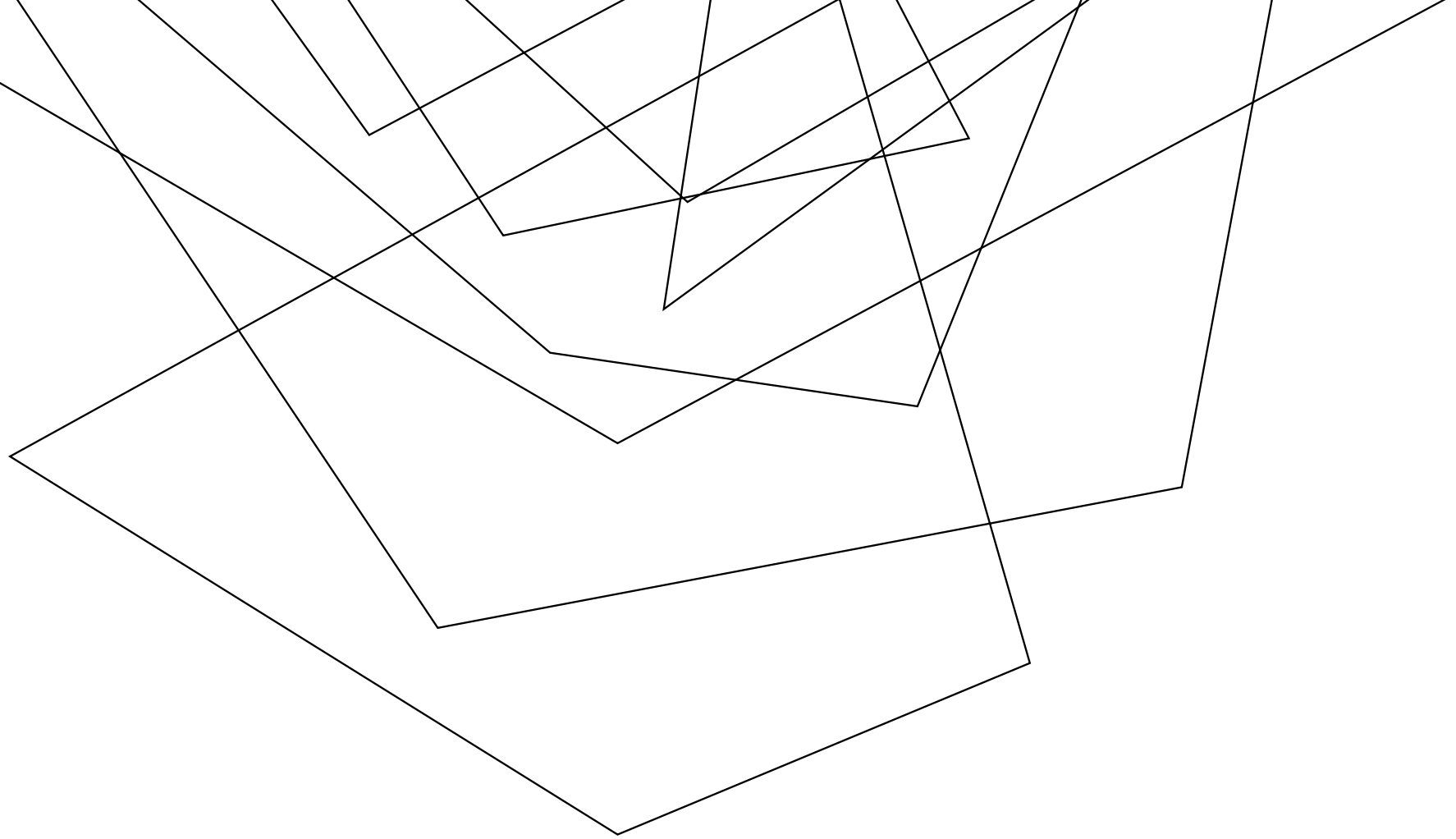
EXERCISE #9

STATIC ANALYSIS REVIEW

- Use path notation to indicate one path (or set of paths) through this CFG:



ADMINISTRIVIA AND ANNOUNCEMENTS



DATAFLOW ANALYSIS

EECS 677: Software Security Evaluation

Drew Davidson

CONTINUE TO EXPLORE STATIC ANALYSIS

CLASS PROGRESS

LOOK INTO CONCRETE FORMS OF STATIC ANALYSIS

- Particularly interested in dataflow analysis for now
- Building up the underlying abstractions / techniques to perform such analysis

LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

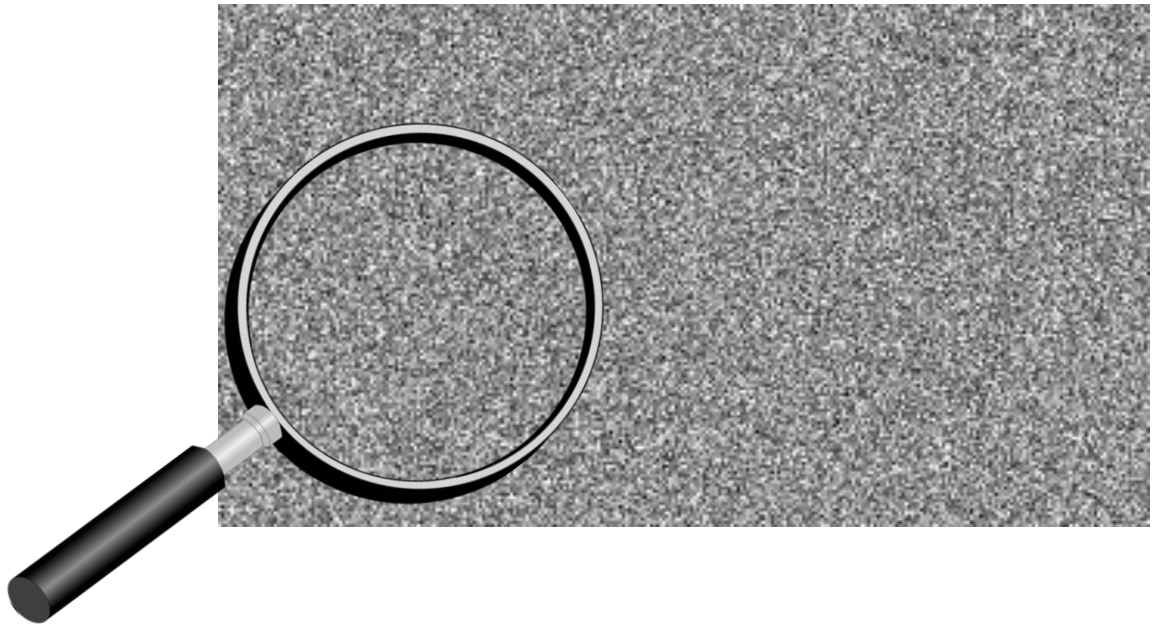
- Syntactic Analysis
- Dataflow Analysis
- Model Checking

auth.c

```
int main(int argc, char * argv[] ){
  return (strcmp(argv[1], "secretpw");
}
```

cmdline

```
$: sudo apt install binutils
$: gcc auth.c -o auth
$; strings auth | less
```



LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Syntactic Analysis
- Dataflow Analysis
- Model Checking

auth.c

```
int main(int argc, char * argv[] ){
  return (strcmp(argv[1], "secretpw");
}
```

cmdline

```
$: sudo apt install binutils
$: gcc auth.c -o auth
$; strings auth | less
```

output

```
/lib64/ld-linux-x86-64.so.2
__libc_start_main
__cxa_finalize
libc.so.6
GLIBC_2.2.5
GLIBC_2.34
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
secretpw
9*3$"
GCC: (Ubuntu 13.2.0-23ubuntu4) 13.2.0
Scrt1.o
…
```

LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Syntactic Analysis
- Dataflow Analysis
- Model Checking

Dataflow Idea: treat each statement as a program state transformer

- Transform a program state into a new (updated) program state
- Simple idea: assume a precondition, induce a postcondition

state M

%y has the value 1

$Stmt_1$: %x = add i32 %y, 0

state M'

%x has the value 1
%y has the value 1

LAST TIME: STATIC ANALYSIS

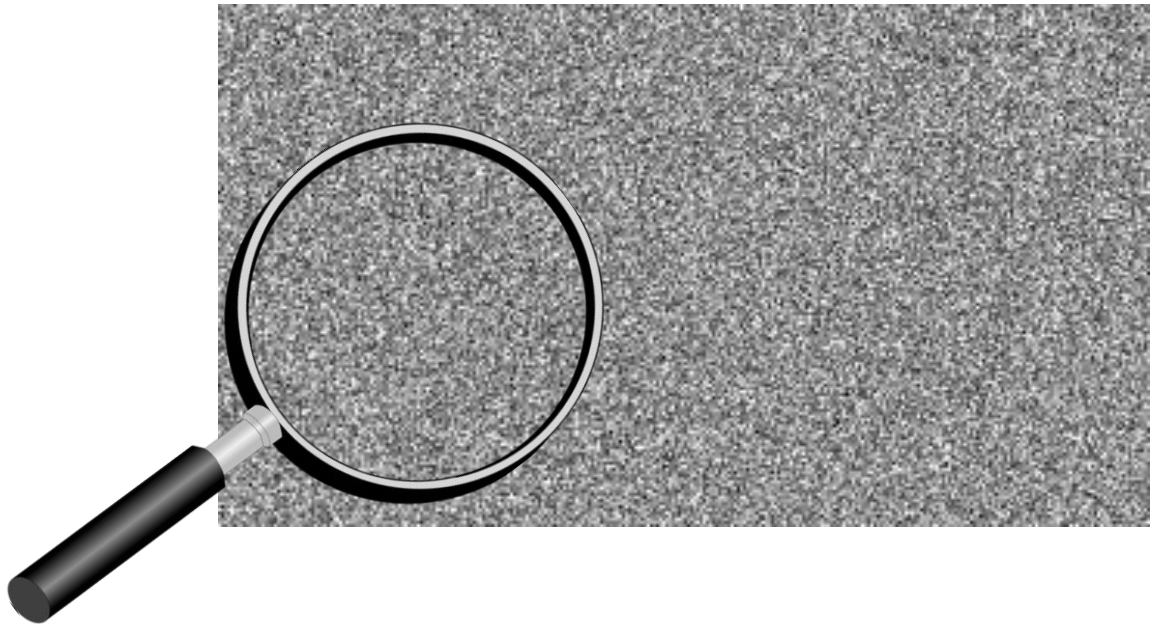
REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Syntactic Analysis
- Dataflow Analysis
- Model Checking

TRUE POWER OF STATIC ANALYSIS

- Unnecessary to supply a given program input
- Summarize the behavior of the program under ANY input
- Capturing all possible behaviors of a program



THE ART OF ABSTRACTION

CLASS PROGRESS

ENUMERATING ALL PROGRAM CONFIGURATIONS IS TOO EXPENSIVE

The trick is getting an approximation of the program's behavior that is both...

- Complete
- Close enough to avoid too many false positives

A complete approximation of program behavior
=
*an **over**-approximation of program behavior*

DATAFLOW ANALYSIS

CLASS PROGRESS

VIEW INSTRUCTIONS AS TRANSFORMERS OF PROGRAM STATE

Several dimensions to tune the state space, we started describing one:

Flow-insensitive analysis

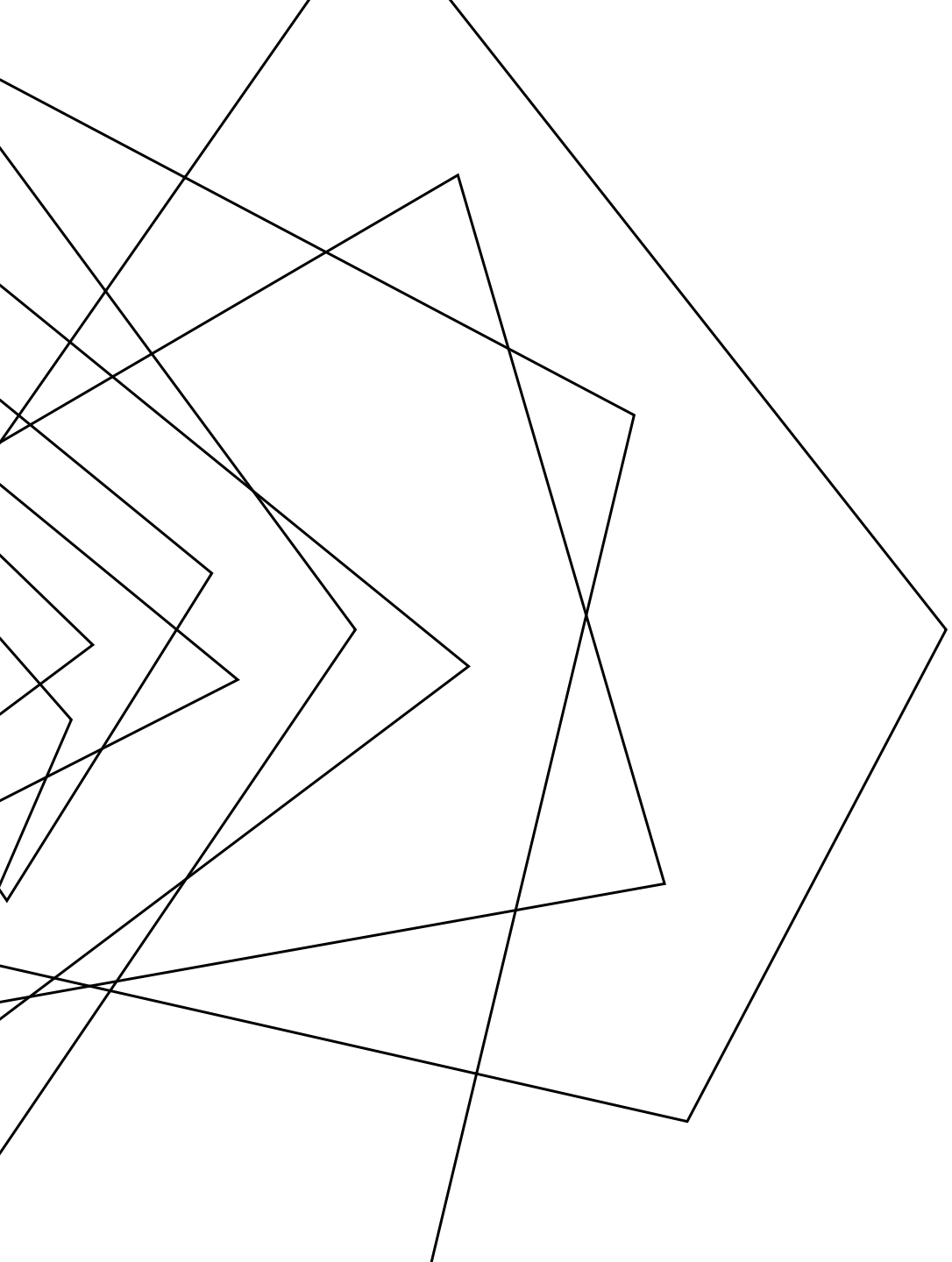
Too squishy

Flow-sensitive analysis

Just right

Path-sensitive analysis

Too hard



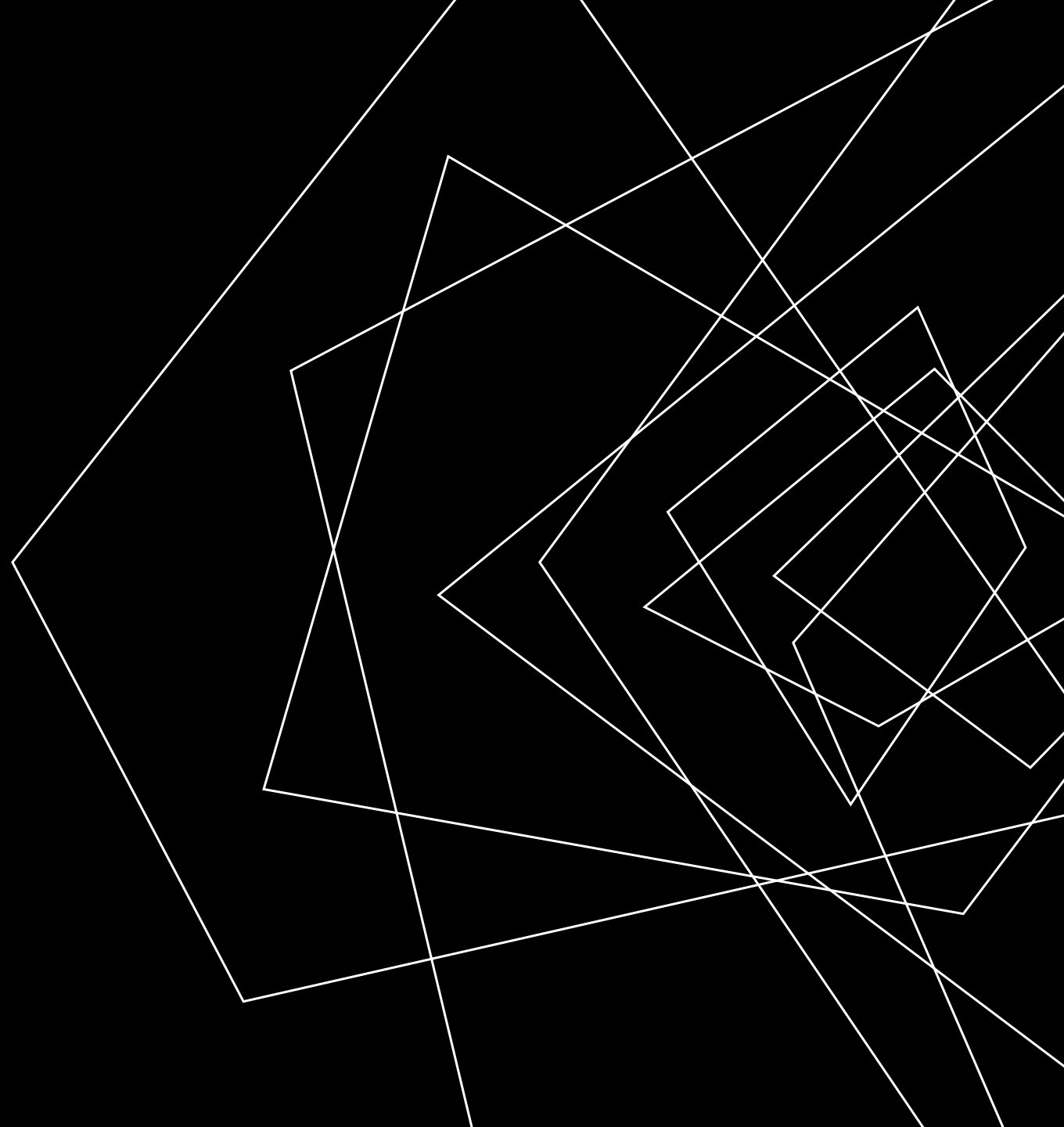
CLASS PROGRESS

WE KNOW SOME BAD BEHAVIORS THAT MIGHT BEAR DETECTION

WE KNOW SOME PROMISING TECHNIQUES FOR ANALYSIS

LECTURE OUTLINE

- Intuition: Flow-sensitive analysis
- Local Flow-sensitive analysis
- Global Flow-sensitive analysis



PRELIMINARIES: DOT

Flow-sensitive analysis relies heavily on the Control-Flow Graph concept

It's pretty helpful to have a CFG in hand

Good news! You know how to automatically induce the CFG structure

Gooder news! There's a format to visualize CFGs

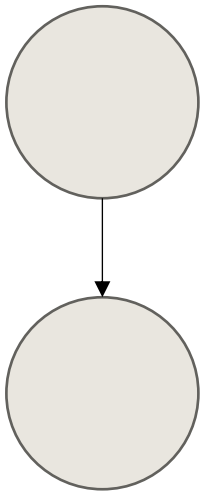
File graph.dot

```
digraph name {
  nodeA […];
  nodeB […];
  nodeA -> nodeB […];
}
```

cmdline

```
dot –Tpdf graph.dot –o graph.pdf
```

output



PRELIMINARIES: DOT

FLOW-SENSITIVE ANALYSIS RELIES HEAVILY ON THE CONTROL-FLOW GRAPH CONCEPT

It's pretty helpful to have a CFG in hand

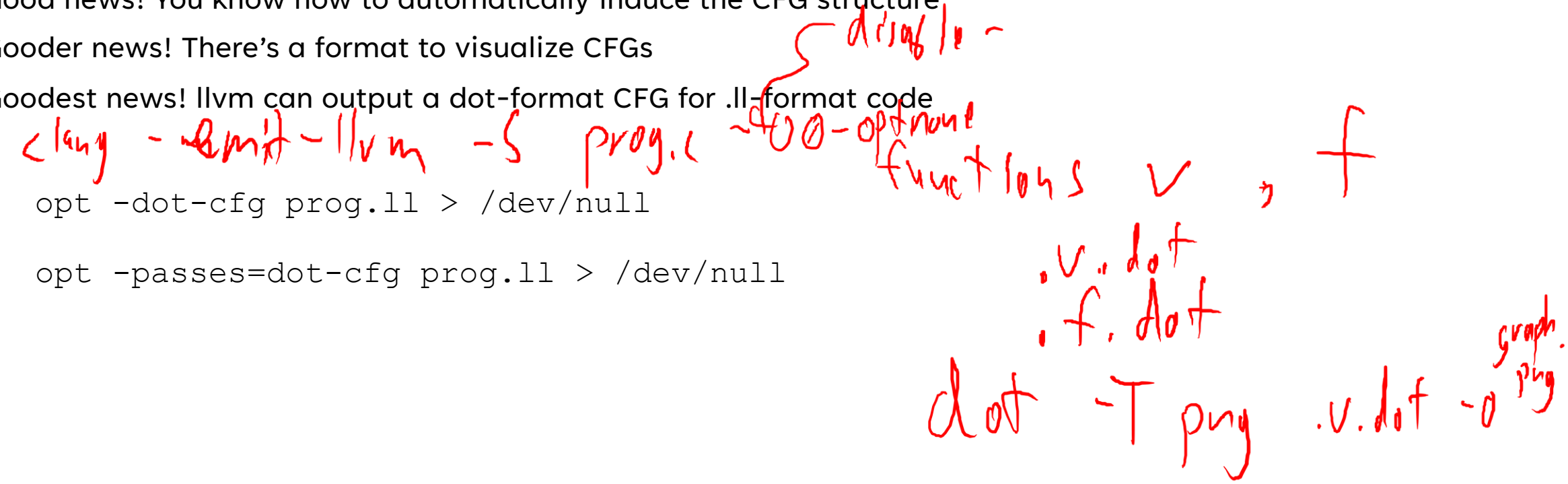
Good news! You know how to automatically induce the CFG structure

Gooder news! There's a format to visualize CFGs

Goodest news! llvm can output a dot-format CFG for .ll-format code

```
opt -dot-cfg prog.ll > /dev/null

opt -passes=dot-cfg prog.ll > /dev/null
```



FLOW-SENSITIVE ANALYSIS

DATAFLOW ANALYSIS

Consider the order of instructions along ANY Feasible control flow

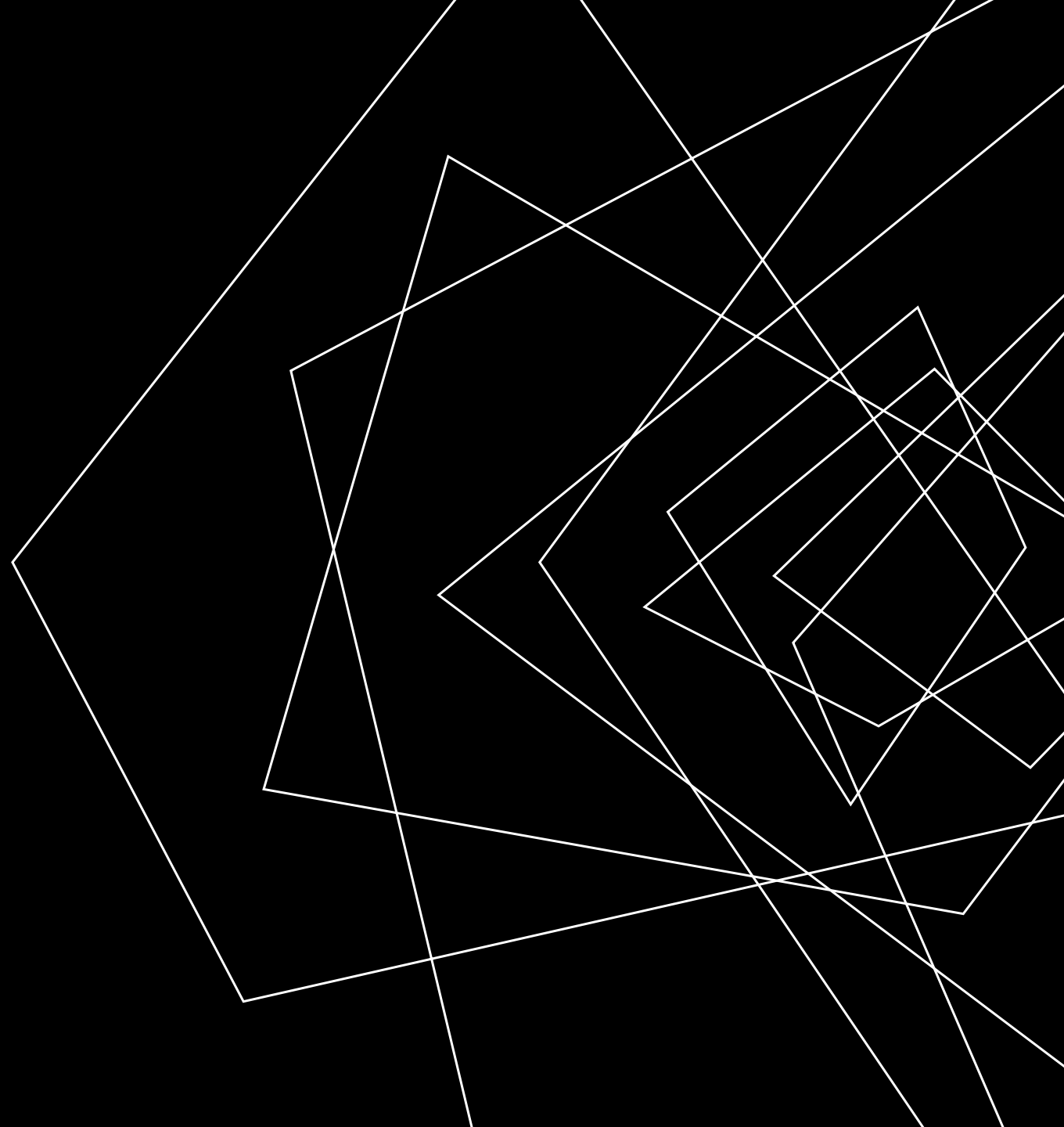
Glom together results of multiple paths

For now, let's start simple: analysis within a Basic Block

Known as local analysis

LECTURE OUTLINE

- Intuition: Flow-sensitive analysis
- Local Flow-sensitive analysis
- Global Flow-sensitive analysis



COMPOSING TRANSFER FUNCTIONS

DATAFLOW ANALYSIS

STATEMENTS COMPOSE NATURALLY WITH EACH OTHER (WITHIN A BASIC BLOCK)

state M

y has the value 1

$Stmt_1$: x = y ;

$Stmt_2$: z = x ;

state M'

x has the value 1
y has the value 1
z has the value 1

For now, we'll only think about analysis within a BBL

AN EARLY WIN

DATAFLOW ANALYSIS

EVEN WITH THIS VERY SIMPLE CONCEPT, MIGHT BE ABLE TO DETECT SOME ISSUES

state M

y has the value 1

Stmt$_1$: x = y ;

$\langle y: 1\rangle, \langle x: 1\rangle$

Stmt$_2$: z = 0 ;

$\langle y: 1\rangle, \langle x: 1\rangle, \langle z: 0\rangle$

Stmt$_3$: p = 1 / z ;

$CRASH$!

FORMALIZING TRANSFER FUNCTIONS

DATAFLOW ANALYSIS

If we want to Build an Automated (Local) Dataflow analysis, we need programmatic Precision

- Some sort of specification of what a statement does
- A statement is a memory state transformer

Memory state M

$Stmt_1$: k += 1 ;

Memory state M'

Need a semantics!

Representation mapping (large) set of memory states to each other

Depend somewhat on the analysis

Goals:

- Keep states manageable
- Handle the uncertainty inherent in static analysis

MEMORY AS VALUE SETS

DATAFLOW ANALYSIS

Let each memory Location correspond to a Set of Values it might Contain

- Define (informally) transfer functions as mapping elements of M to elements of M'

We're still kinda-dodging the larger semantic questions here, for now lets just say we're using a big ol' if statement to define an operator

Memory state M $\langle k: \{1\} \rangle$ $\langle k: \{3,4\} \rangle$

$Stmt_1$: k += 1 ;

Memory state M' $\langle k: \{2\} \rangle$ $\langle k: \{4,5\} \rangle$

COMPOSING VALUE SETS

DATAFLOW ANALYSIS

(example: assume a 1-bit data size)

$Stmt_0$: y = randomBit()

$$\langle y: \{0, 1\} \rangle$$

$Stmt_1$: x = y ;

$$\langle y: \{0, 1\}, x: \{0, 1\} \rangle$$

$Stmt_2$: z = x ;

$$\langle y: \{0, 1\}, x: \{0, 1\}, z: \{0, 1\} \rangle$$

$Stmt_3$: p = 1 / z ;

$$CRASH?!$$

MODELLING UNCERTAINTY

DATAFLOW ANALYSIS

WE CAN NOW HANDLE OPAQUE DATA SOMEWHAT CLEANLY

z : {0}

y : {0, 1}

$Stmt_1$: x = y ;

z : {0} x : {0, 1}, y : {0, 1}

$Stmt_2$: z = USER_INPUT ;

z : {0, 1} x : {0, 1}, y : {0, 1}

$Stmt_3$: p = 1 / z ;

CRASH ?!

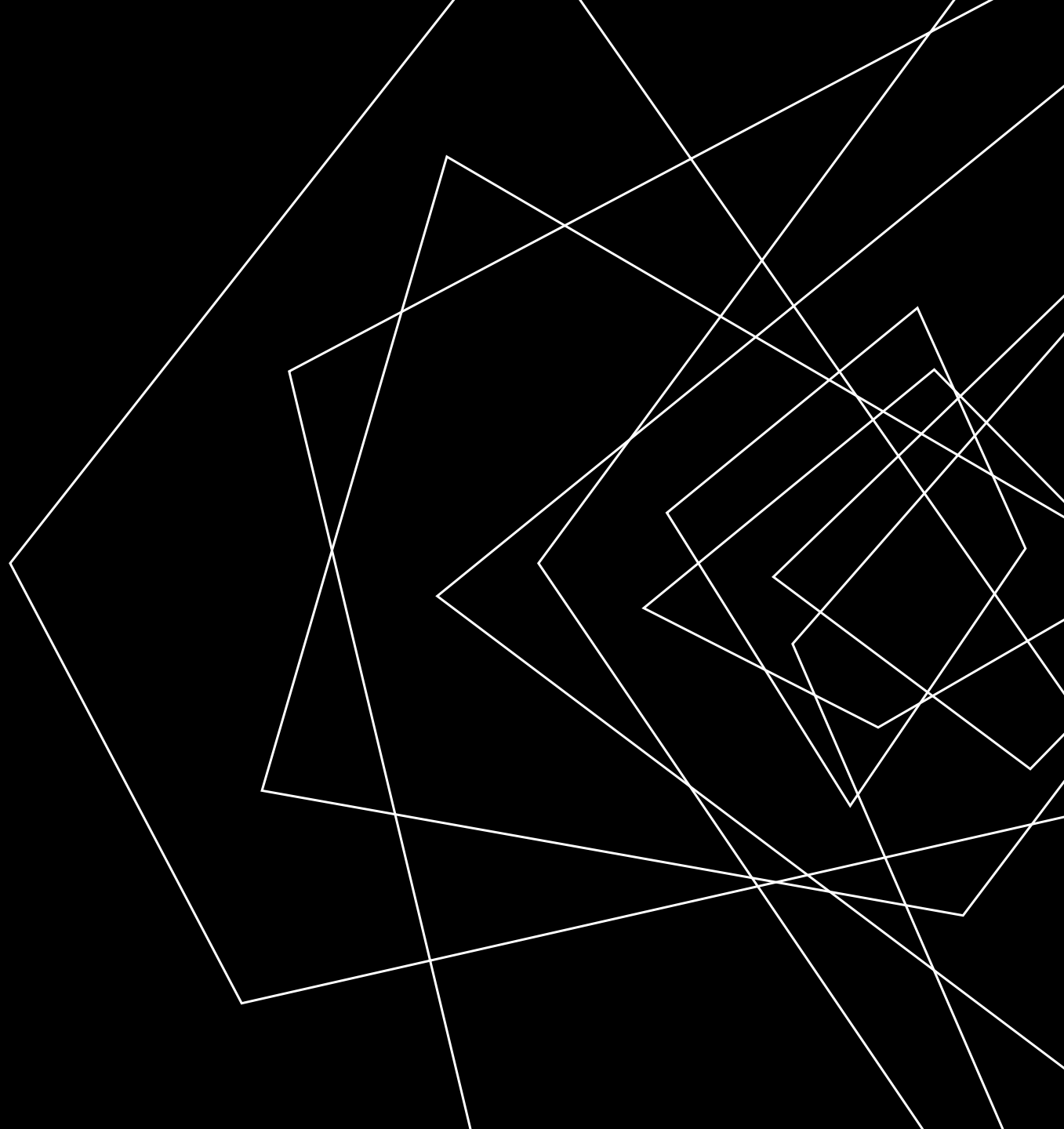
$Stmt_1$: x = y ;

$Stmt_2$: z = global ;

$Stmt_3$: p = 1 / z ;

LECTURE OUTLINE

- (Local) Dataflow analysis
- Global dataflow analysis



COMPOSING BLOCKS

GLOBAL DATAFLOW ANALYSIS

VALUE-SET MODEL OF MEMORY IMPLIES A METHOD TO EXTEND BEYOND LOCAL ANALYSIS

```
void f(bool a){
  bool b = a;
  bool c = a;
  if (a){
    b = true;
    c = true
  } else {
    b = true;
    c = false;
  }
  return b;
}
```

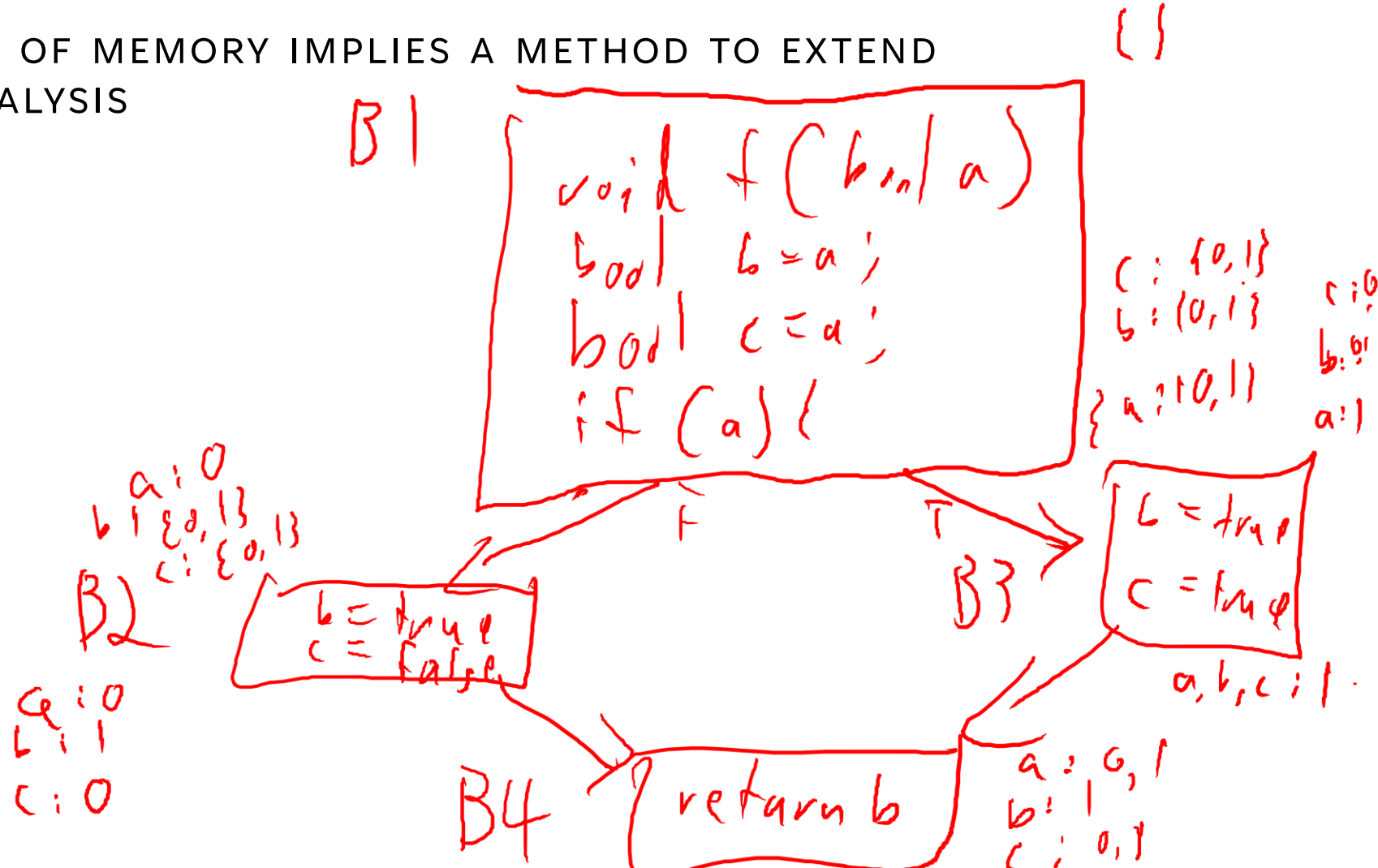
Go Global

COMPOSING BLOCKS

GLOBAL DATAFLOW ANALYSIS

VALUE-SET MODEL OF MEMORY IMPLIES A METHOD TO EXTEND BEYOND LOCAL ANALYSIS

```
void f(bool a){
  bool b = a;
  bool c = a;
  if (a){
    b = true;
    c = true
  } else {
    b = true;
    c = false;
  }
  return b;
}
```



MAY-BE VS MUST-BE ANALYSIS

GLOBAL DATAFLOW ANALYSIS

How we join values is based on the goal of our analysis

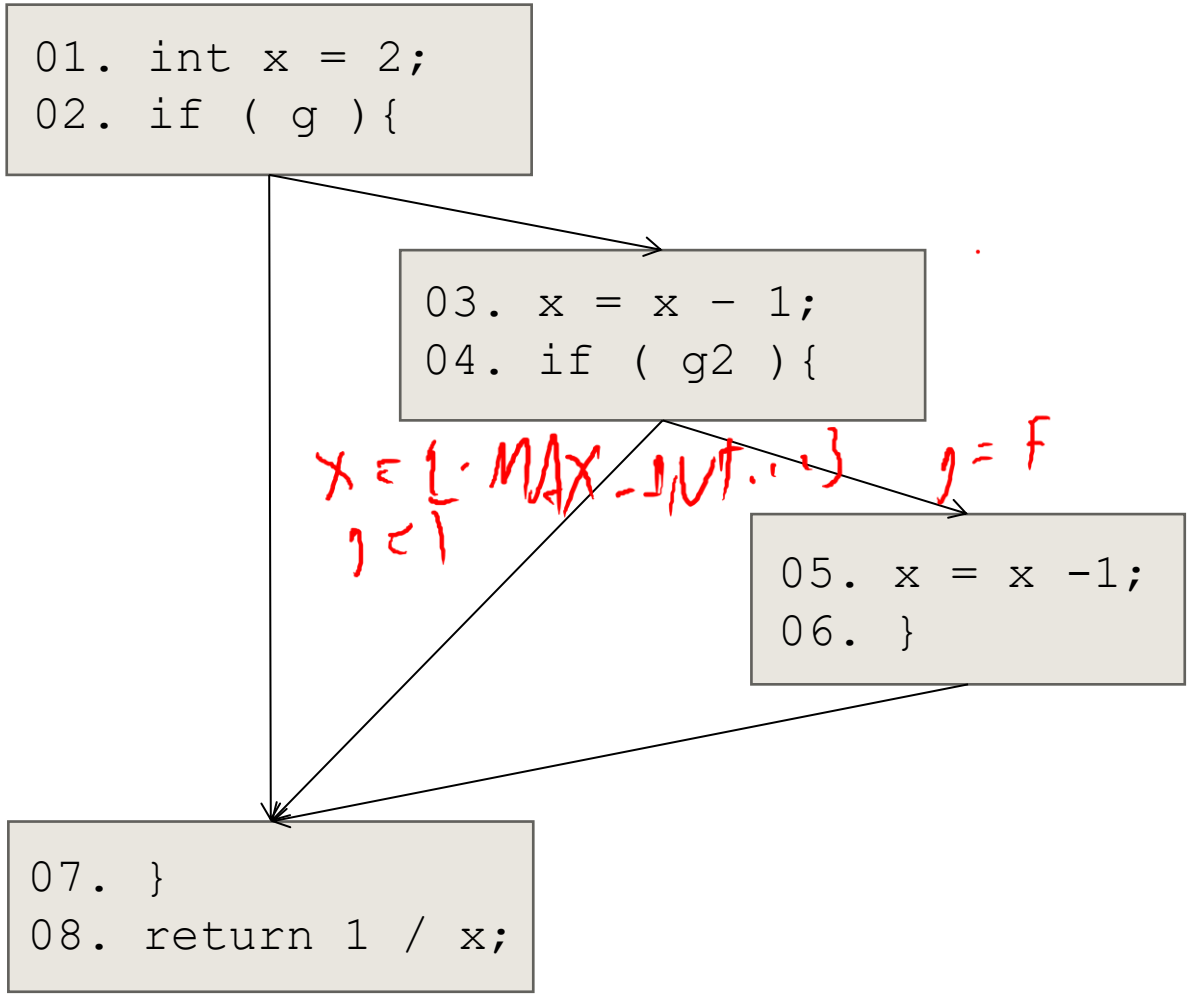
```
void f(bool a){
  bool b = a;
  bool c = a;
  if (a){
    b = true;
    c = true
  } else {
    b = true;
    c = false;
  }
  return b;
}
```

CHAOTIC ITERATION

GLOBAL DATAFLOW ANALYSIS

IN WHAT ORDER DO WE PROCESS BLOCKS?

```
01. int x = 2;
02. if ( g ){
03.     x = x - 1;
04.     if ( g2 ){
05.       x = x - 1;
06.     }
07. }
08. return 1 / x;
```



TROUBLE ON THE HORIZON

GLOBAL DATAFLOW ANALYSIS

LOOPS ARE TOUGH TO HANDLE!

GLOBAL DATAFLOW ANALYSIS

Issues with Loops

- Generate lots of paths
- Cyclic data dependency

Oh, brother! You may have some loops

LECTURE END!

- Local Dataflow analysis
- Global Dataflow analysis