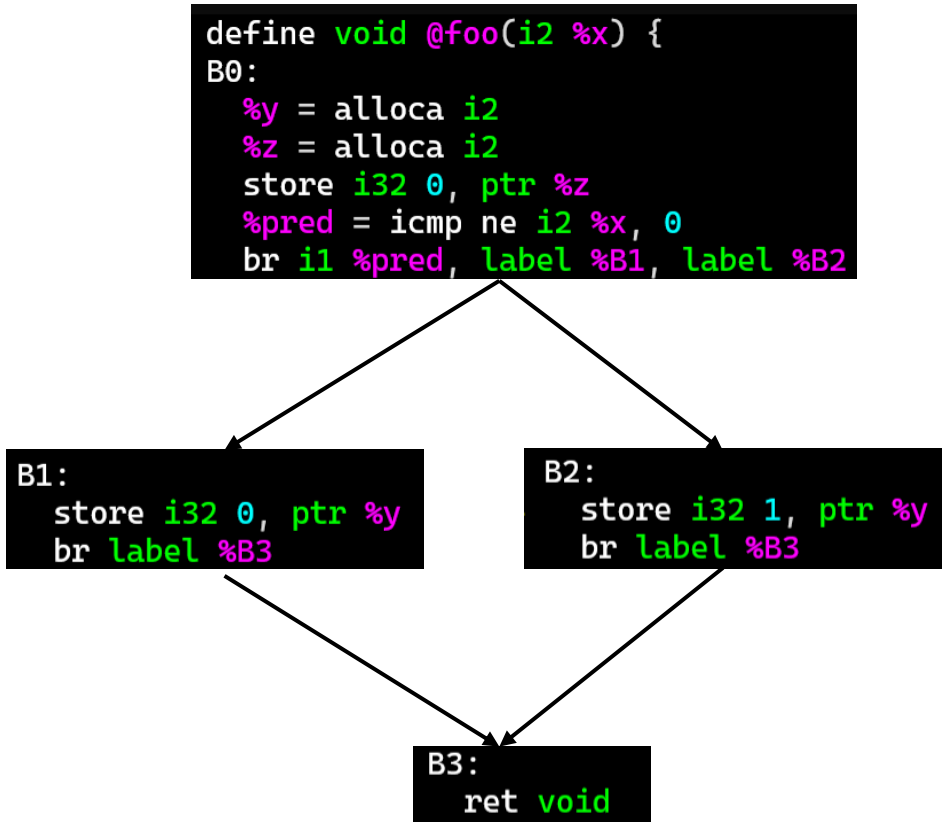
EXERCISE

DATAFLOW REVIEW

Fill out the value sets that a flow-sensitive dataflow analysis would assign



	x	*y	*z
B0	{0,1,2,3}	{0,1,2,3}	{0,1,2,3}
B0→B1	{1,2,3}	{0,1,2,3}	{0}
B0→B2	{0}	{0,1,2,3}	{0}
B1	{1,2,3}	{0,1,2,3}	{0}
B2	{0}	{0,1,2,3}	{0}
B1→B3	{1,2,3}	{0}	{0}
B2→B3	{0}	{1}	{0}
B3	{0,1,2,3}	{0,1}	{0}
exit	{0,1,2,3}	{0,1}	{0}

EXERCISE SOLUTION

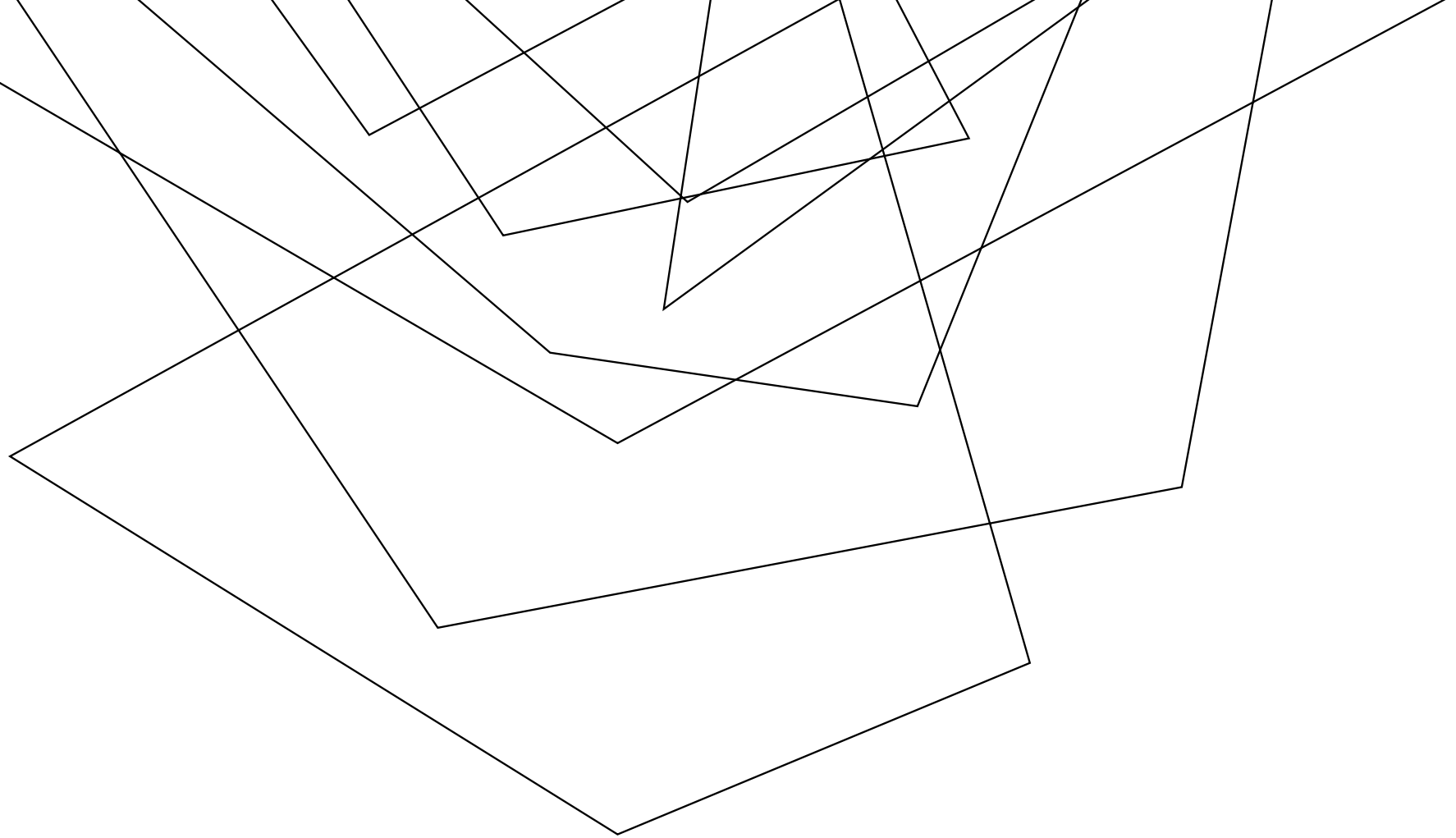
DATAFLOW REVIEW

	x	*y	*z
B0			
B0→B1			
B0→B2			
B1			
B2			
B1→B3			
B2→B3			
B3			
exit			

ADMINISTRIVIA AND ANNOUNCEMENTS

Test 1 next Friday

- Review Session TBA



DATAFLOW FIXPOINTS

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME: VALUE SET ANALYSIS

REVIEW: DATAFLOW

CONSERVATIVELY TRACK THE POSSIBLE SET OF VALUES TAKEN

```
 1 define void @foo(i2 %arg) {
 2   %xAddr = alloca i2
 3   store i2 %arg, ptr %yAddr
 4   %yAddr = alloca i2
 5   %t = call i1 (...) @rand()
 6   store i2 %t, ptr %yAddr
 7   ret void
 8 }
 9
10 declare i2 @rand(...)
```

*xAddr = {0,1,2,3} , *yAddr = {0,1,2,3}

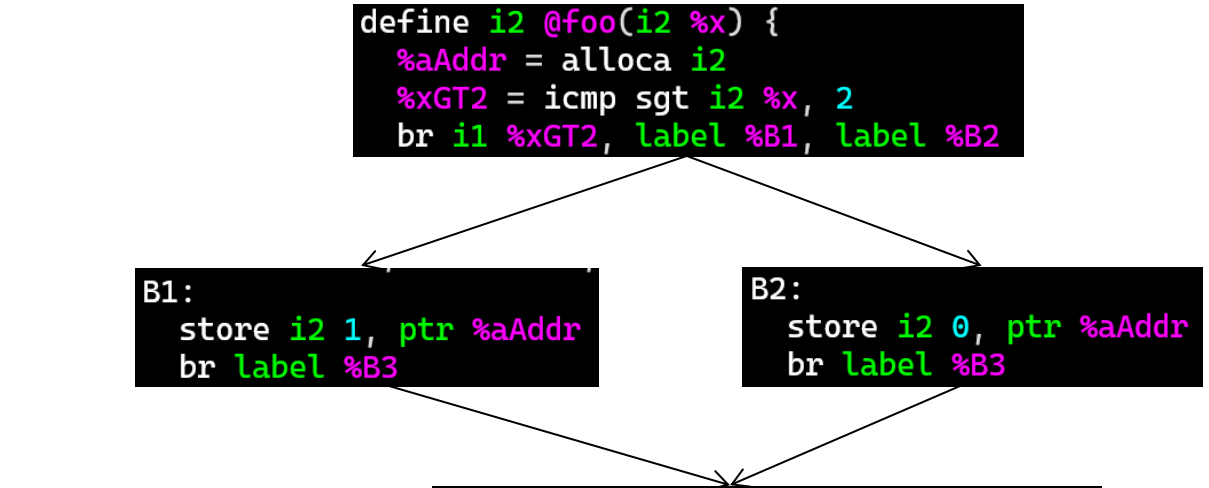
This approach is a complete over-approximation

LAST TIME: FLOW SENSITIVITY

REVIEW: DATAFLOW

ACCOUNT FOR PROGRAM FLOW, NOT PATHS: When control flow merges, merge the value sets

	x	xGT2	*aAddr
Entry	{0,1,2,3}		{0,1,2,3}
E→B1	{3}	{1}	{0,1,2,3}
B1	{3}	{1}	{0,1,2,3}
E→B2	{0,1,2}	{0}	{0,1,2,3}
B2	{0,1,2}	{0}	{0,1,2,3}
B1→B3	{3}	{1}	{1}
B2→B3	{0,1,2}	{0}	{0}
B3	{0,1,2,3}	{0,1}	{0,1}
B3→B4	{0,1,2,3}	{1}	{0,1}
B4	{0,1,2,3}	{1}	{0,1}
B3→B5	{0,1,2,3}	{0}	{0,1}
B4→B5	{0,1,2,3}	{1}	{1}
B5	{0,1,2,3}	{0,1}	{1}
exit	{0,1,2,3}	{0,1}	{0,1}




```
1  void foo(int x){
2          int a;
3          if (x > 2){
4                  a = 1;
5          else
6                  a = 0;
7
8          if (x > 2)
9                  x / a;
10 }
```

LOOPS ARE TOUGH TO HANDLE!

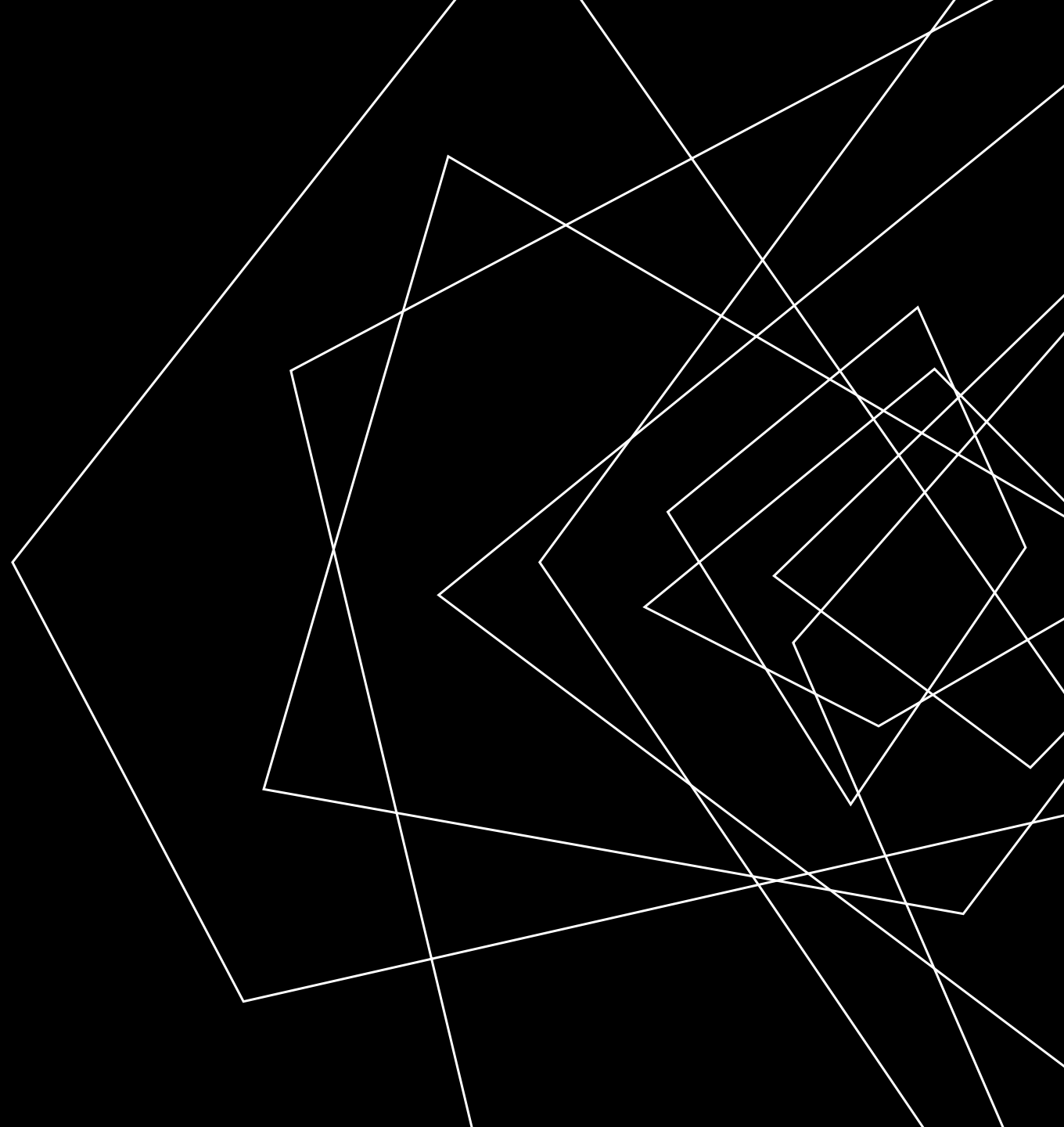
REVIEW: DATAFLOW ANALYSIS

ISSUES WITH LOOPS

- Generate lots of paths
- Cyclic data dependency

LECTURE OUTLINE

- Handling cyclic dependency
- Termination
- Handling large value sets



A WORD OF CAUTION

DATAFLOW ANALYSIS

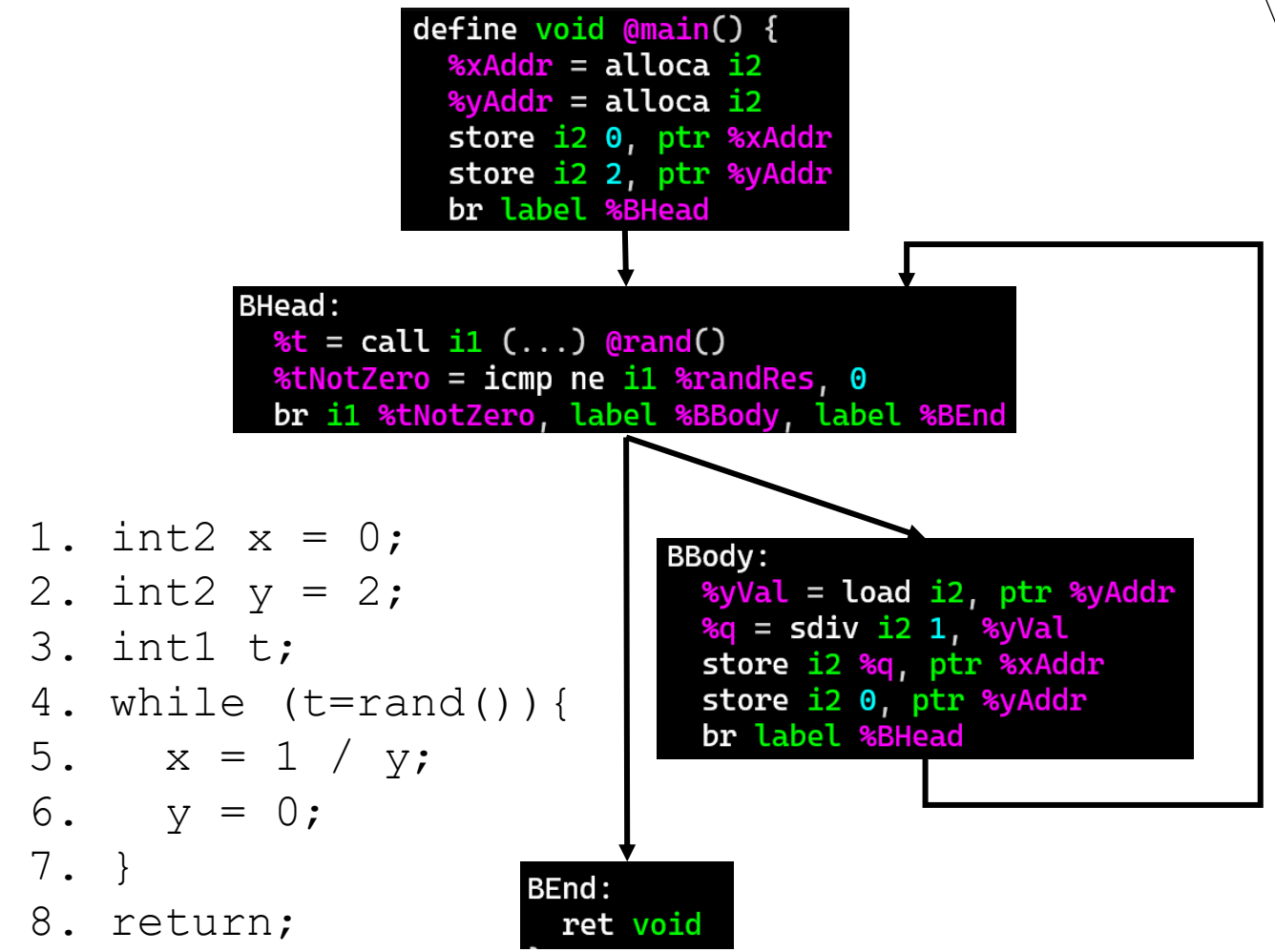
WE NEED TO BUILD UP A LOT OF INTERLOCKING MACHINERY FOR A "REAL" FLOW-SENSITIVE ANALYSIS

- I'll present a simplified algorithm here with some subtle problems, which we'll fix up next time

CYCLIC ANALYSIS

DATAFLOW ANALYSIS

	*x	*y	t
(entry)	{0,1,2,3}	{0,1,2,3}	
(entry)→BHead	{0}	{2}	
BHead	{0}	~~{2}~~ {0,2}	
BHead→BBody	{0}	~~{2}~~ {0,2}	{1}
BHead→BEnd	{0}	{2} ~~{0,2}~~	{0}
BBody	{0}	~~{2}~~ {0,2}	{1}
BBody→BHead	~~??? {}~~ {0}	~~??? {}~~ {0}	~~{}~~ {1}
BEnd	{0}	~~{2}~~ {0,2}	{0}
(exit)			




```
1. int2 x = 0;
2. int2 y = 2;
3. int1 t;
4. while (t=rand()){
5.    x = 1 / y;
6.    y = 0;
7. }
8. return;
```

CYCLIC ANALYSIS: INITIALIZATION

DATAFLOW ANALYSIS

START EVERY VALUE-SET WITH THE "EMPTY" VALUE

- Value will eventually be updated

IS EXPLORING EVERY EDGE ENOUGH?

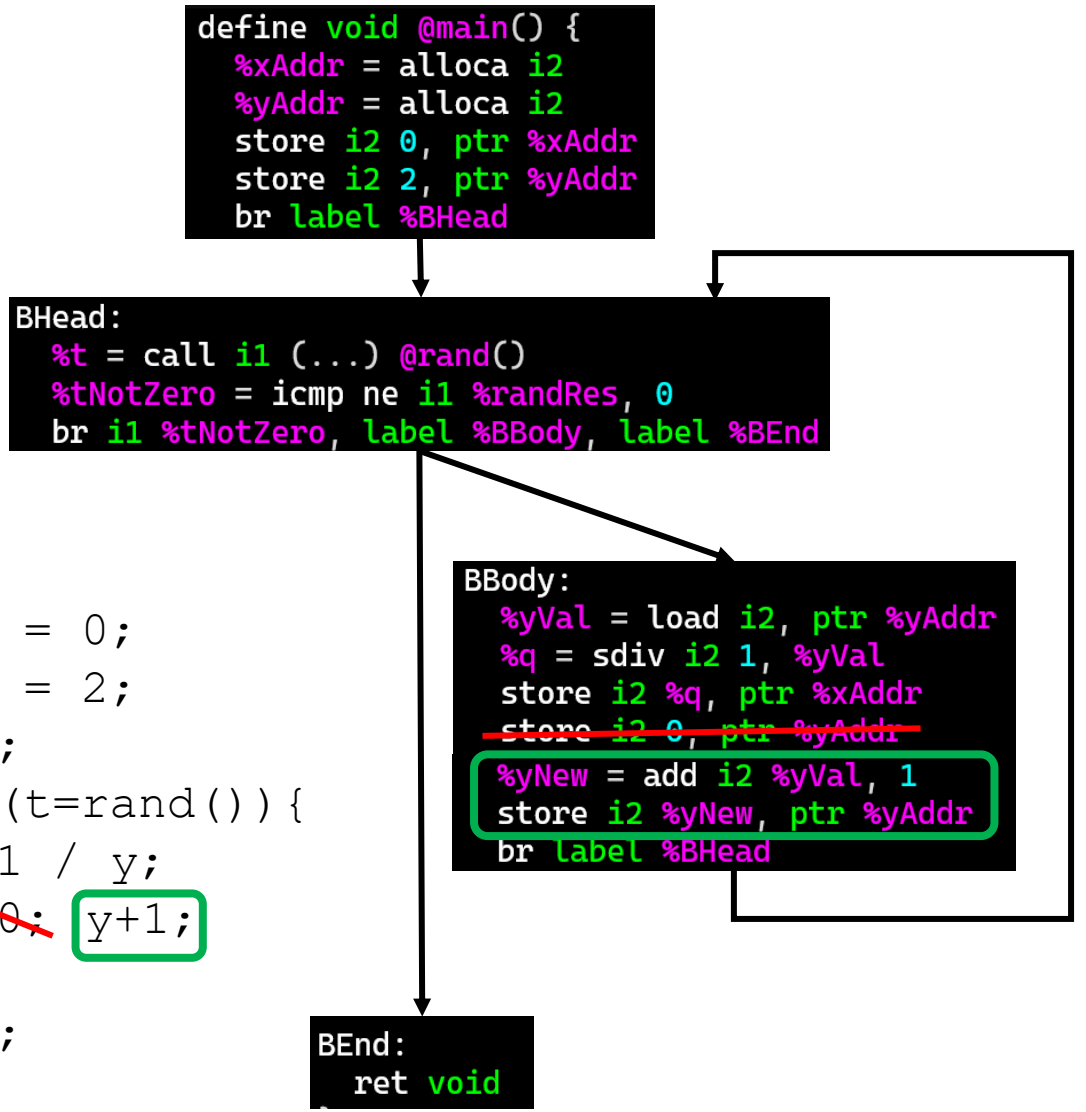
- No!

CYCLIC ANALYSIS

DATAFLOW ANALYSIS

	*x	*y	t
(entry)	{0,1,2,3}	{0,1,2,3}	
(entry)→BHead	{0}	{2}	
BHead	{0}	~~{2}~~ ~~{2,3}~~{2,3,0}	
BHead→BBody	{0}	~~{2}~~ ~~{2,3}~~{2,3,0}	{1}
BHead→BEnd	{0}	~~{2}~~ {2,3}	{0}
BBody	{0}	~~{2}~~ ~~{2,3}~~ {2,3,0}	{1}
BBody→BHead	~~??? { }~~ {0}	~~??? { }~~ ~~{3}~~{3,0}	~~{ }~~ {1}
BEnd	{0}	~~{2}~~ {2,3}	{0}
(exit)			

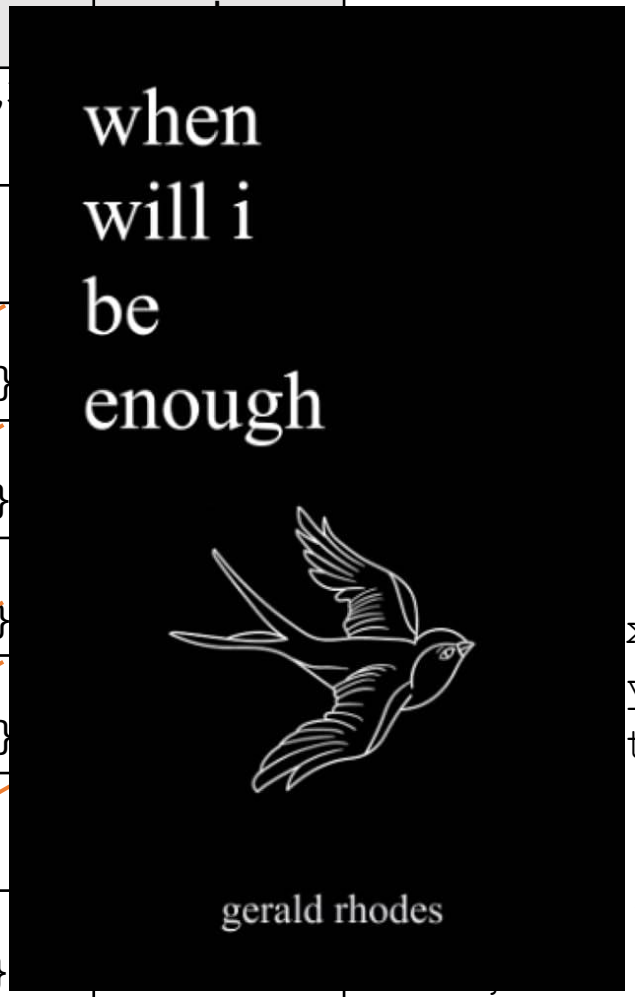
```
1. int2 x = 0;
2. int2 y = 2;
3. int1 t;
4. while (t=rand()){
5.    x = 1 / y;
6.    y = 0; y+1;
7. }
8. return;
```



CYCLIC ANALYSIS

DATAFLOW ANALYSIS

	*x	*y	
(entry)	{0,1,2,3}	{0,1,2,	
(entry)→BHead	{0}	{2}	
BHead	{0}	~~{2}~~ {2,3}	
BHead→BBody	{0}	~~{2}~~ {2,3}	
BHead→BEnd	{0}	{2} ~~{2,3}~~	
BBody	{0}	~~{2}~~ {2,3}	
BBody→BHead	~~??? { }~~ {0}	~~??? { }~~ {3}	
BEnd	{0}	~~{0}~~ {2,3}	
(exit)			




```
x = 0;
y = 2;
t;
 (t=rand()){
 1 / y;
0; y+1;
8. return;
```

```
define void @main() {
  %xAddr = alloca i2
  %yAddr = alloca i2
  store i2 0, ptr %xAddr
  store i2 2, ptr %yAddr
  br label %BHead
```

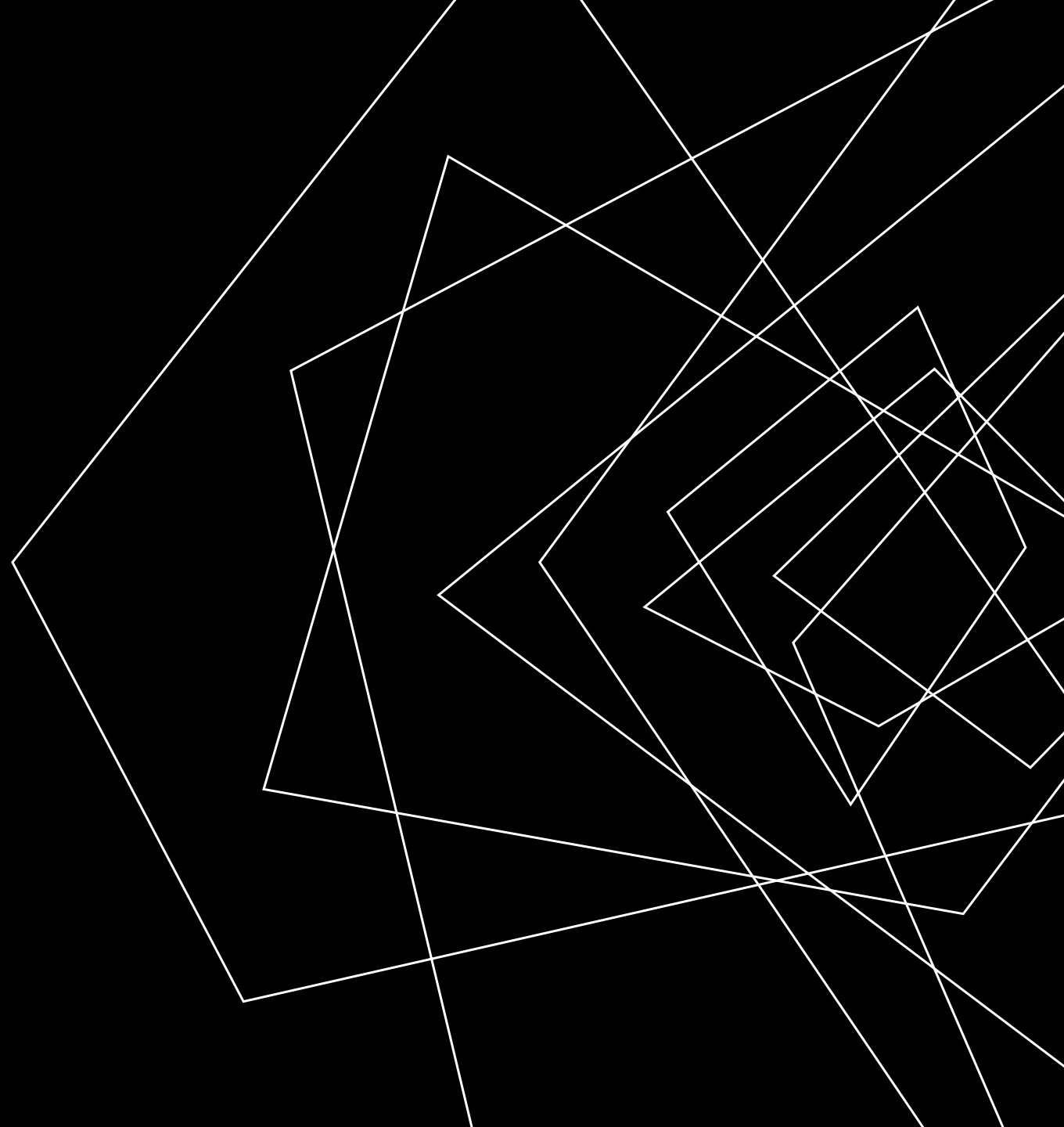
```
BHead:
  %t = call i1 (...) @rand()
  %tNotZero = icmp ne i1 %randRes, 0
  br i1 %tNotZero, label %BBody, label %BEnd
```

```
BBody:
  %yVal = load i2, ptr %yAddr
  %q = sdiv i2 1, %yVal
  store i2 %q, ptr %xAddr
  store i2 0, ptr %yAddr
  %yNew = add i2 %yVal, 1
  store i2 %yNew, ptr %yAddr
  br label %BHead
```

```
BEnd:
  ret void
```

LECTURE OUTLINE

- Handling cyclic dependency
- Termination
- Handling large value sets

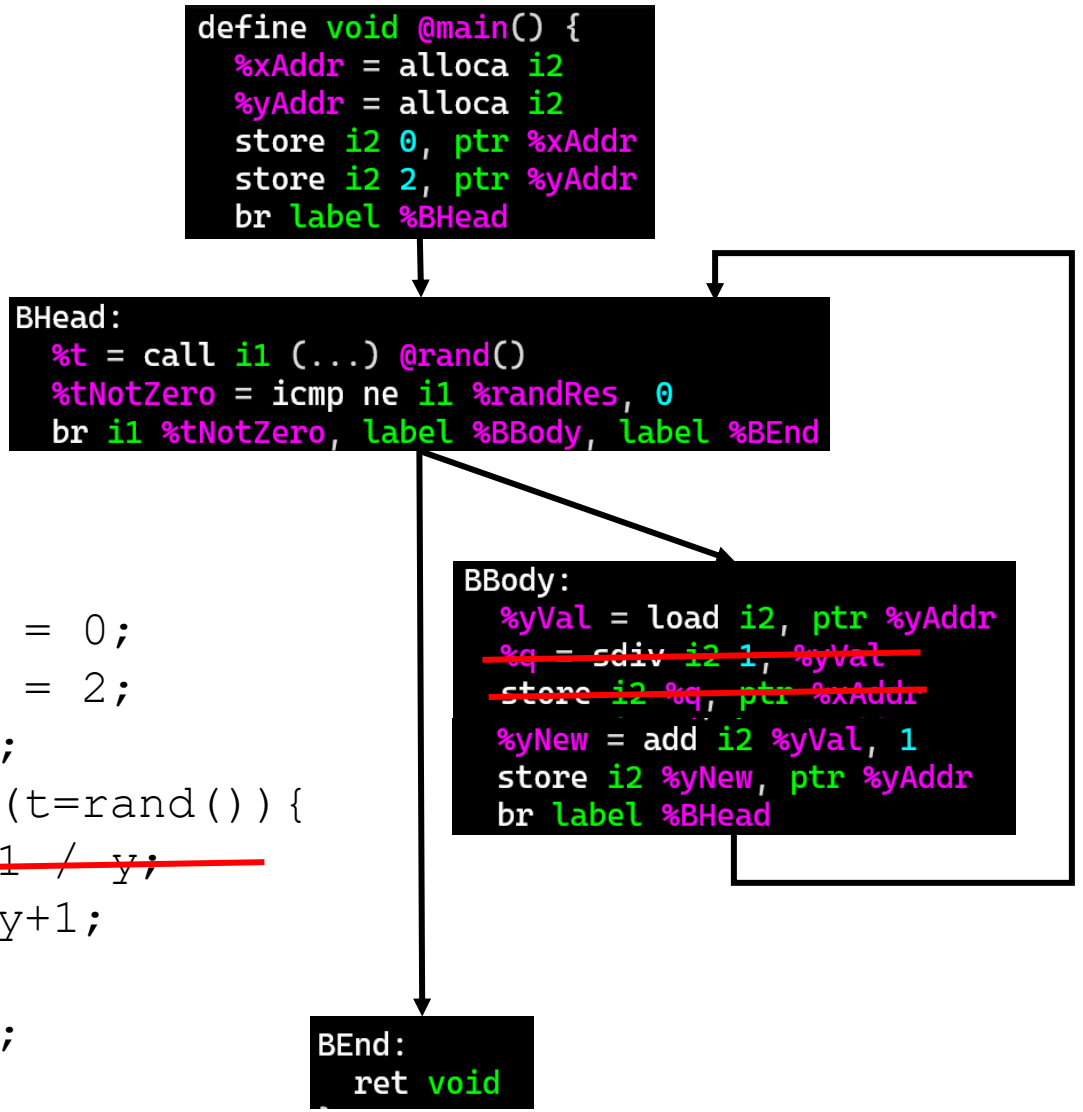


WHEN TO STOP?

DATAFLOW ANALYSIS

	*x	*y	t
(entry)	{0,1,2,3}	{0,1,2,3}	
(entry)→BHead	{0}	{2}	
BHead	{0}	{2}	
BHead→BBody	{0}	{2}	{1}
BHead→BEnd	{0}	{2}	{0}
BBody	{0}	{2}	{1}
BBody→BHead	~~??? { }~~ {0}	~~??? { }~~{3}	~~{ }~~ {1}
BEnd	{0}	{2}	{0}
(exit)			

```
1. int2 x = 0;
2. int2 y = 2;
3. int1 t;
4. while (t=rand()){
5.   x = 1 / y;
6.   y = y+1;
7. }
8. return;
```

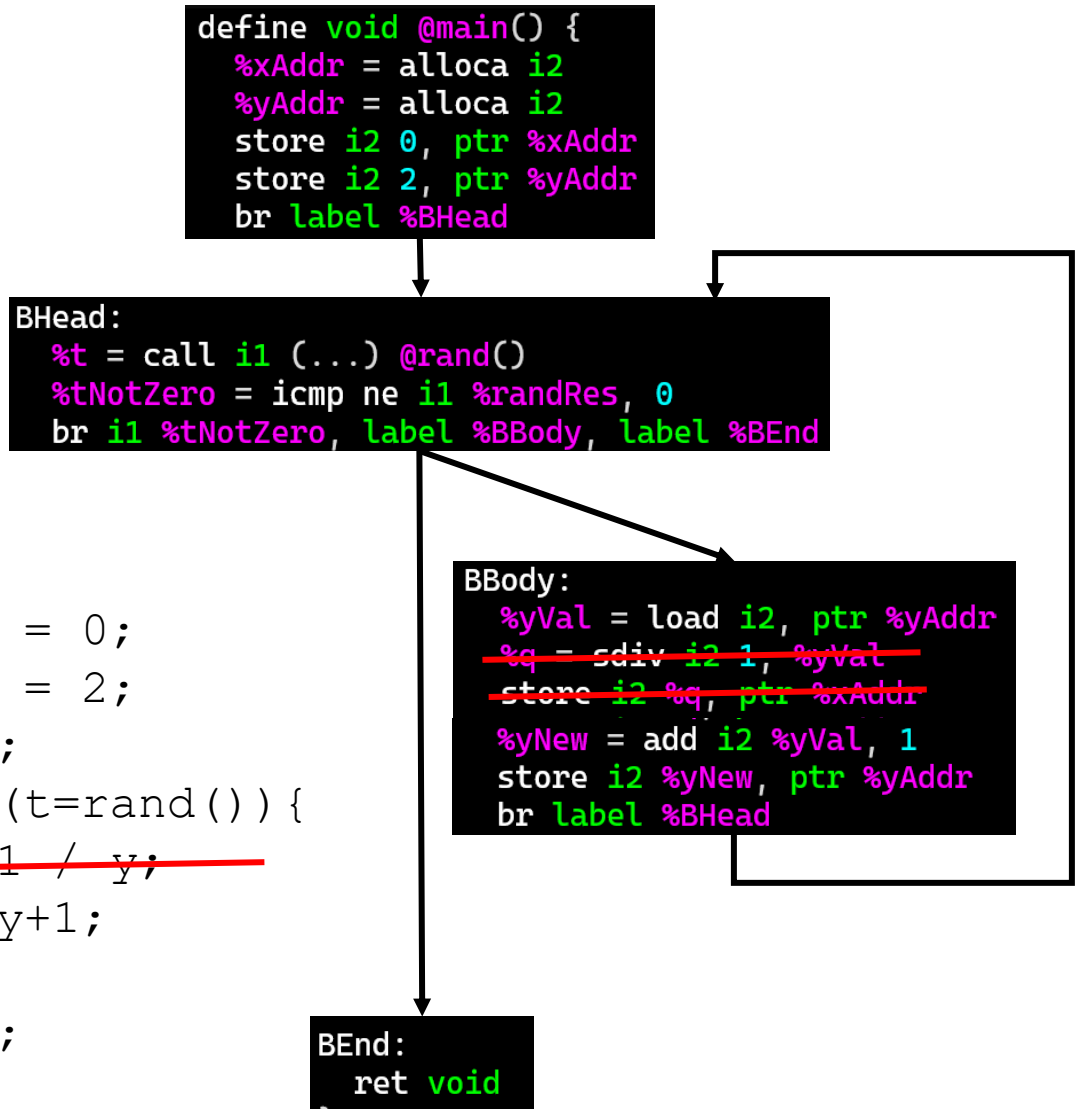


WHEN TO STOP?

DATAFLOW ANALYSIS

	*x	*y	t
(entry)	{0,1,2,3}	{0,1,2,3}	
(entry)→BHead	{0}	{2}	
BHead	{0}	~~{2}~~ ~~{2,3}~~ ~~{2,3,0}~~ {2,3,0,1}	{1}
BHead→BBody	{0}	~~{2}~~ ~~{2,3}~~ ~~{2,3,0}~~ {2,3,0,1}	{1}
BHead→BEnd	{0}	~~{2}~~ ~~{2,3}~~ ~~{2,3,0}~~ {2,3,0,1}	{0}
BBody	{0}	~~{2}~~ ~~{2,3}~~ ~~{2,3,0}~~ {2,3,0,1}	{1}
BBody→BHead	~~???~~ ~~{}~~ {0}	~~???~~ ~~{}~~ ~~{3}~~ ~~{3,0}~~ ~~{3,0,1}~~ {3,0,1,2}	~~{}~~ {1}
BEnd	{0}	~~{2}~~ ~~{2,3}~~ ~~{2,3,0}~~ {2,3,0,1}	{0}
(exit)			

```
1. int2 x = 0;
2. int2 y = 2;
3. int1 t;
4. while (t=rand()){
5.   x = 1 / y;
6.   y = y+1;
7. }
8. return;
```



ANALYSIS PROGRESS

STATIC ANALYSIS: CONTROL FLOW GRAPHS

ANALYSIS ENDS WHEN THE FACT SETS REACH *SATURATION*

- No additional elements will ever be added

When your fact sets couldn't possibly hold any more data

FIXED-POINTS

STATIC ANALYSIS: CONTROL FLOW GRAPHS

A Fixed-Point (AKA fixpoint, fixed point)

- A value that does not change under a given transformation

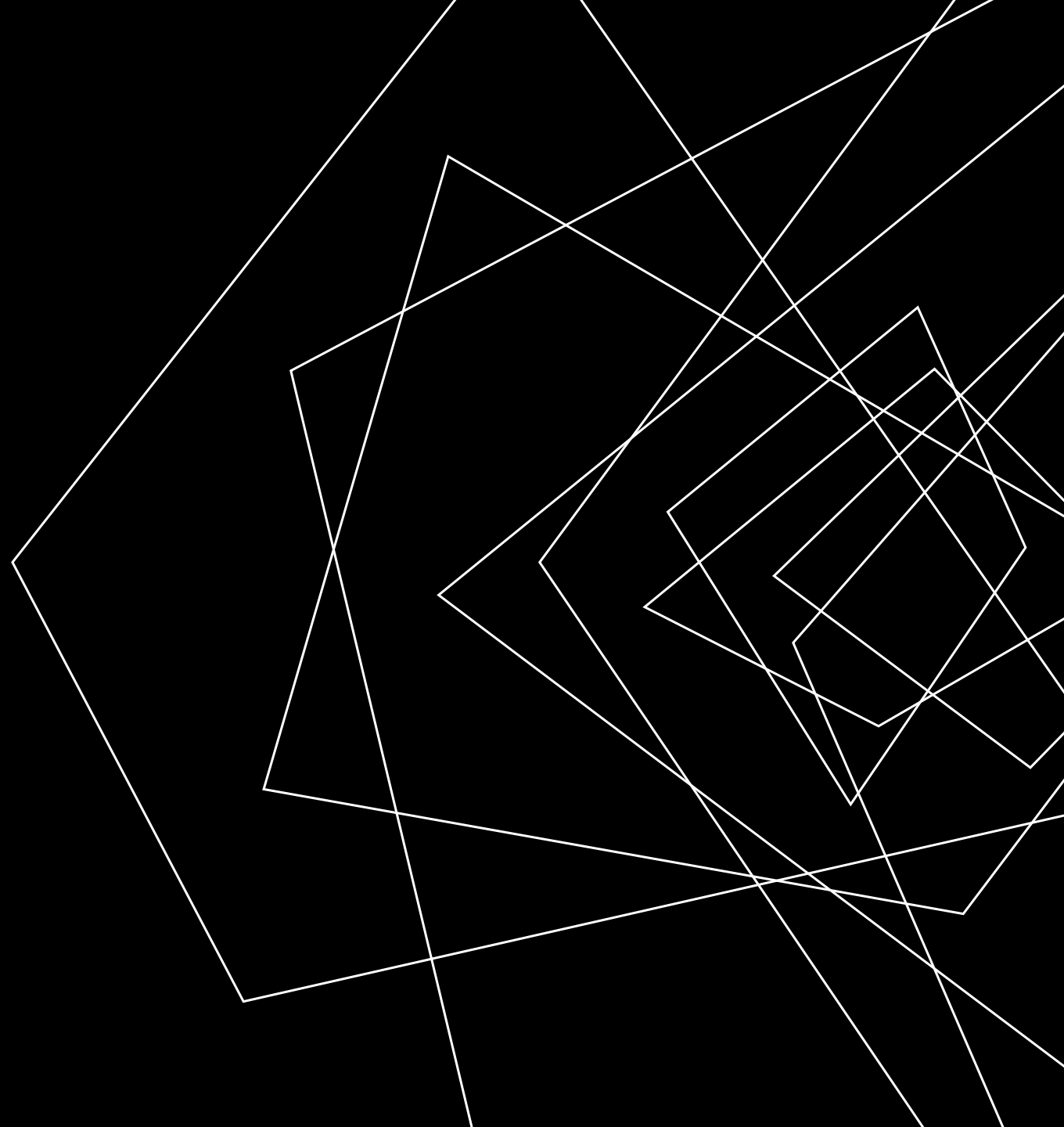
Our value-set analysis WILL have Facts that reach a Fixed-point

Why?

- Finite set of configurations over INT32s
- Data transforms only **add** data to fact sets

LECTURE OUTLINE

- Breaking cyclic dependency
- Termination
- Handling large value sets

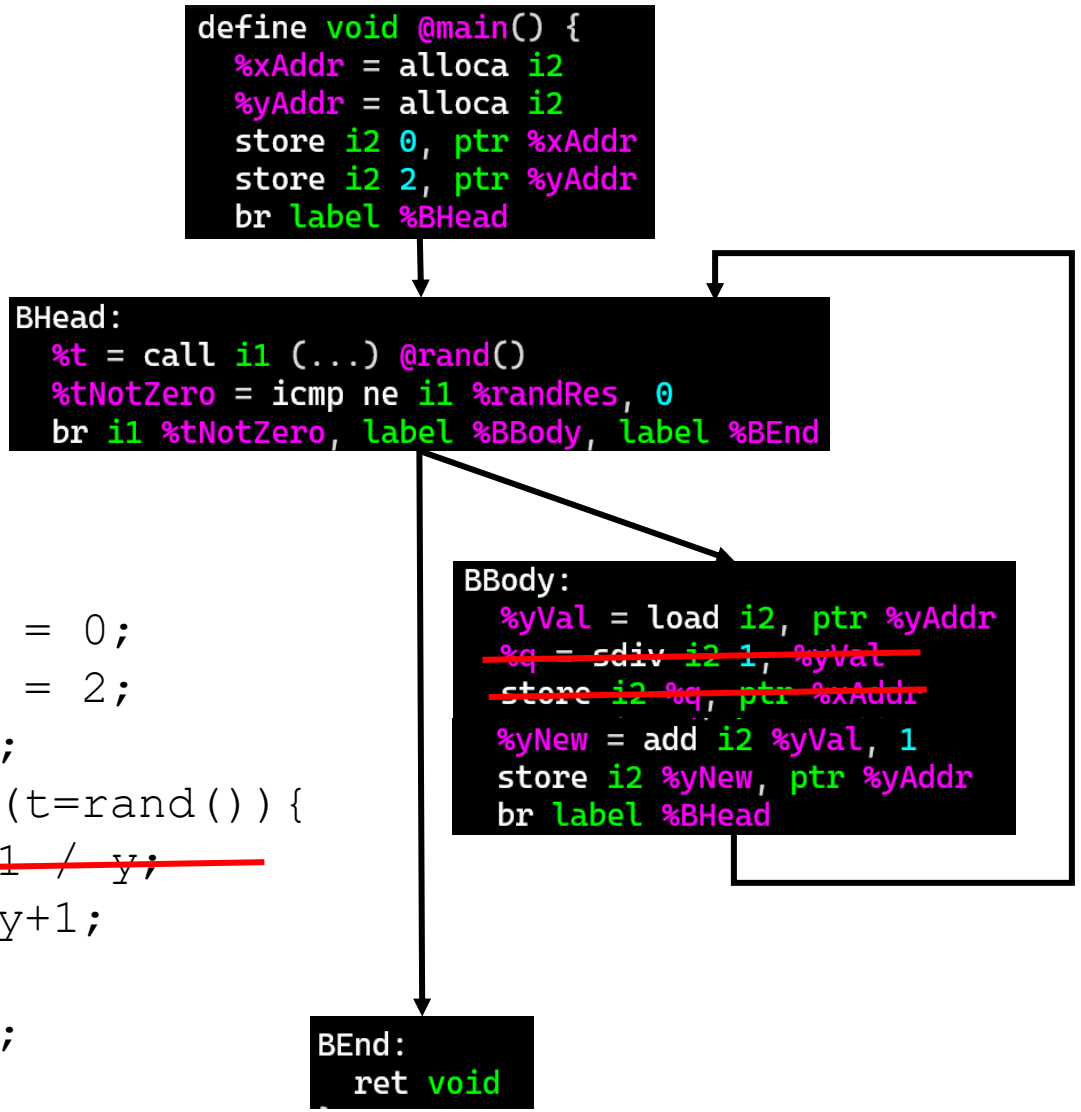


WHEN TO STOP?

DATAFLOW ANALYSIS

	*x	*y	t
(entry)	{0,1,2,3}	{0,1,2,3}	
(entry)→BHead	{0}	{2}	
BHead	{0}	~~{2}~~ ~~{2,3}~~{2,3,0}	
BHead→BBody	{0}	~~{2}~~ ~~{2,3}~~{2,3,0}	{1}
BHead→BEnd	{0}	{2} ~~{2,3}~~	{0}
BBody	{0}	~~{2}~~ ~~{2,3}~~ {2,3,0}	{1}
BBody→BHead	~~??? {}~~ {0}	~~??? {}~~ ~~{3}~~{3,0}	~~{}~~ {1}
BEnd	{0}	~~{0}~~ {2,3}	{0}
(exit)			

```
1. int2 x = 0;
2. int2 y = 2;
3. int1 t;
4. while (t=rand()){
5.   x = 1 / y;
6.   y = y+1;
7. }
8. return;
```



DOES OUR ANALYSIS TERMINATE?

ABSTRACT INTERPRETATION

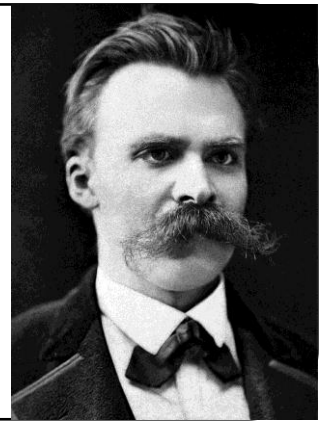
WE'VE SPENT SOME TIME ON SOME UNCOMFORTABLE TRUTHS:

- Programs are often full of subtle bugs / vulnerabilities
- It is effectively impossible to tell if an arbitrary program terminates

BAD NEWS: OUR ANALYSIS ENGINE IN A PROGRAM

- What assurance do we have that our algorithm terminates as it analyzes an unknown / untrusted program?

> "Whoever fights monsters should see to it that in the process he does not become a monster. And when you look long into an abyss, the abyss also looks into you."
>
> - Friedrich Nietzsche



GUARANTEEING HALTING
FIXPOINTS

TERMINATE WHEN WE ALL VALUE SETS SATURATE

GUARANTEE SATURATION

- Ensure the abstract values never shrink
- Ensure the set of abstract values is finite

SUMMARY
ABSTRACT INTERPRETATION

TERMINATE WHEN WE ALL VALUE SETS SATURATE

GUARANTEE SATURATION

- Ensure the abstract values never shrink
- Ensure the set of abstract values is finite

NEXT TIME

ABSTRACT INTERPRETATION

TERMINATE WHEN WE ALL VALUE SETS SATURATE

GUARANTEE SATURATION

- Ensure the abstract values never shrink
- Ensure the set of abstract values is finite

DEALING WITH REALITY

- Keeping abstract value domains manageable
- Getting to saturation faster

NEXT TIME

WRAP UP

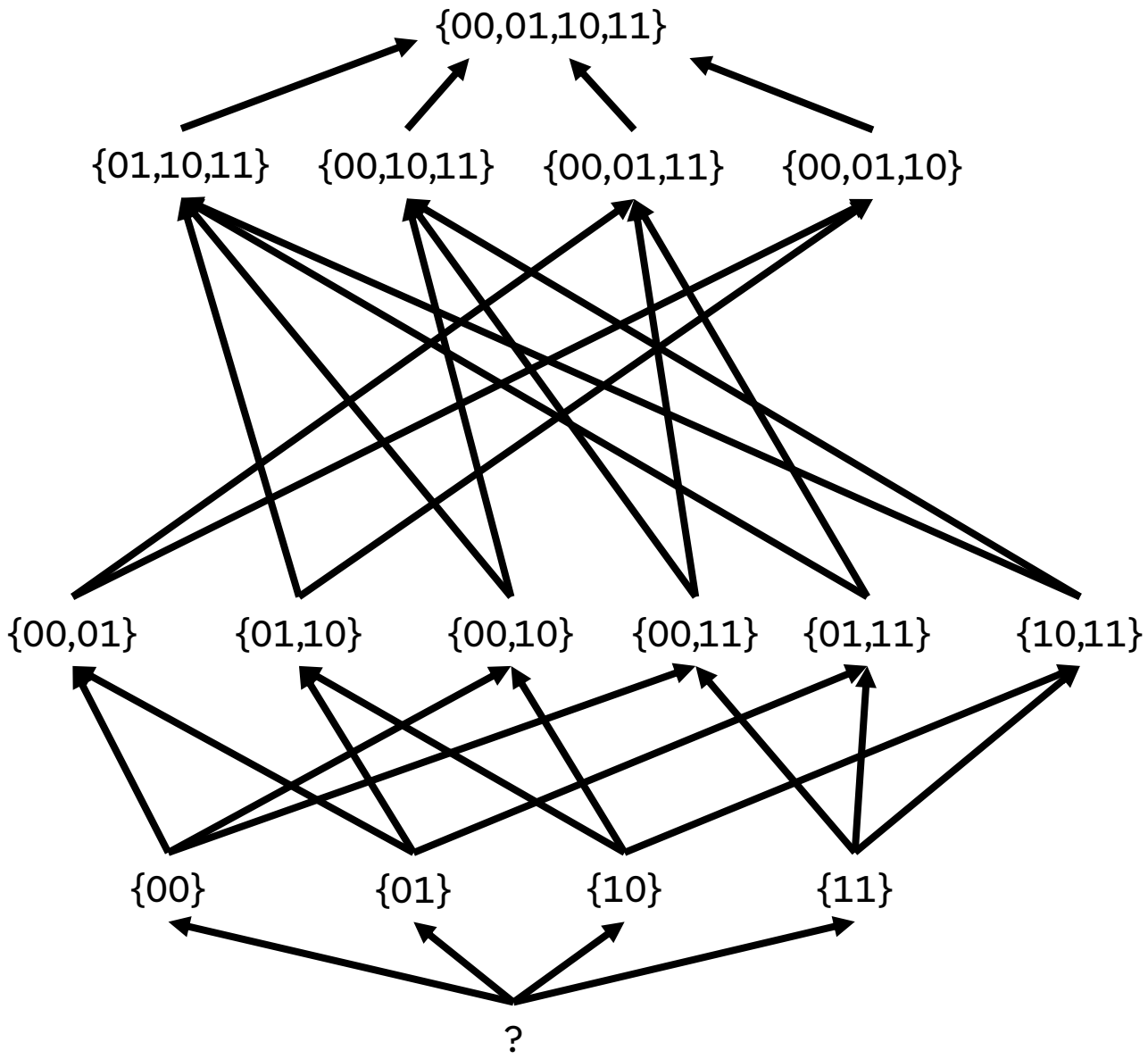
DEALING WITH REALITY

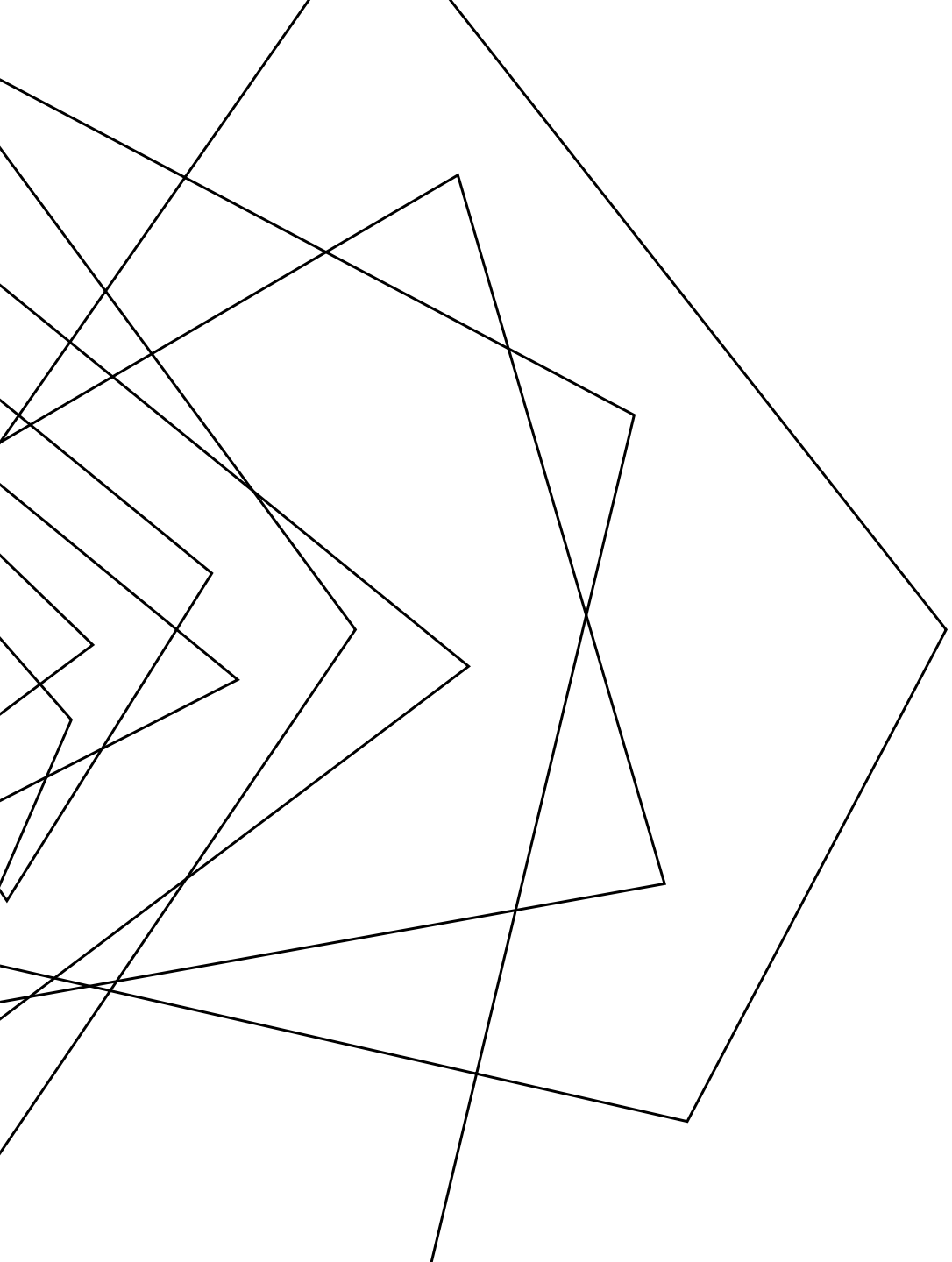
- Getting to saturation faster
- Ensure the set of abstract values is finite

FORMALIZING TERMINATION

DATAFLOW FRAMEWORKS

Value-Set “Rank”
(2-bit computer)





LECTURE END!

Described some of the issues and Fixes for dataflow in the presence of loops