Program Analysis & Software Engineering

Hira Agrawal
hira@research.telcordia.com

Motivation

- Common Scenarios
  - No idea why the program fails for a certain input
  - No idea which modules implement function X
  - No time to achieve desired block coverage
  - No time to run even half of the regression tests

Outline

- Program Analysis
- Static v/s Dynamic
- Program Analysis in Software Engineering
  - Efficient Coverage Testing
  - Smart Debugging
  - Incremental Regression Testing
- A Short Demo
- Future Directions

Program Analysis

- Compiler Optimizations
  - Common Subexpression Elimination
  - Constant Folding
  - Code Motion
  - etc.

Static Program Analysis

- Control Flow Analysis
  - Basic Blocks
  - Dominator Trees
  - Natural Loops
  - etc.
- Data Flow Analysis
  - Def-Use Chains
  - Live Variables
  - Available Expressions
  - etc.

Dynamic Program Analysis

- Which paths were traversed?
- Which def-use associations were exercised?
- Which functions got invoked from where?
- Which statements were exercised?
- Which statement should I try to cover next?
- etc.

Static v/s Dynamic

Static	**Dynamic**
What *could* happen?	What *did* happen?
Foresight	Hindsight
Conservative	Precise

Program Analysis & Software Engineering

- Traditional Applications
  - Software Metrics
  - Coverage Testing
  - Execution Profiling
- Newer Applications
  - Efficient Coverage Testing
  - Incremental Regression Testing
  - Smart Debugging
  - Program Understanding
  - White Box Reliability Engineering
  - Probabilistic Optimizations

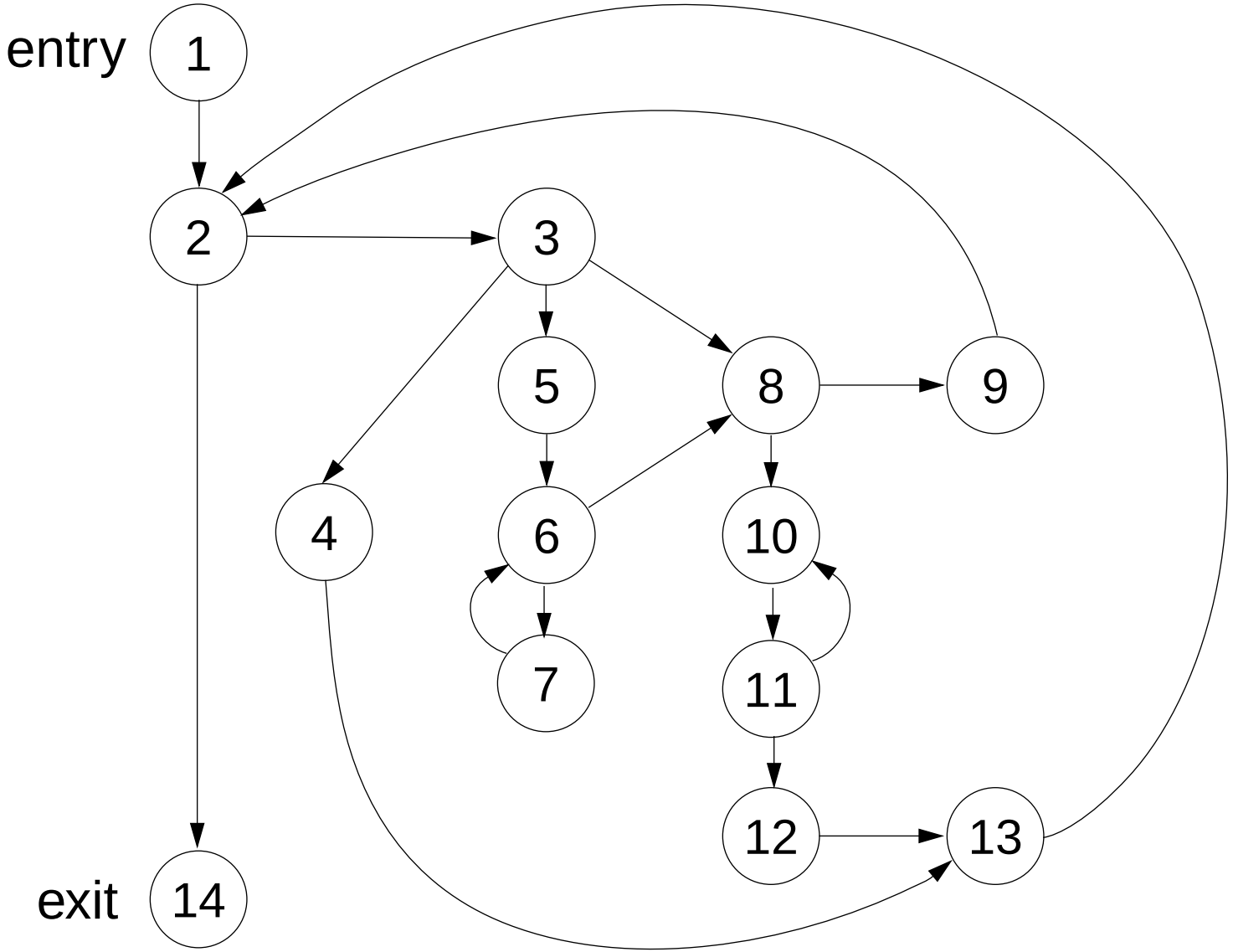
An Example

```
exp1;
while (exp2) {
    switch (exp3) {
        case 1: exp4;
                break;
        case 2: exp5;
                while (exp6) exp7;
        default:
            if (exp8) {
                exp9;
                continue;
            }
            do exp10; while (exp11);
            exp12;
    }
    exp13;
}
exp14;
```

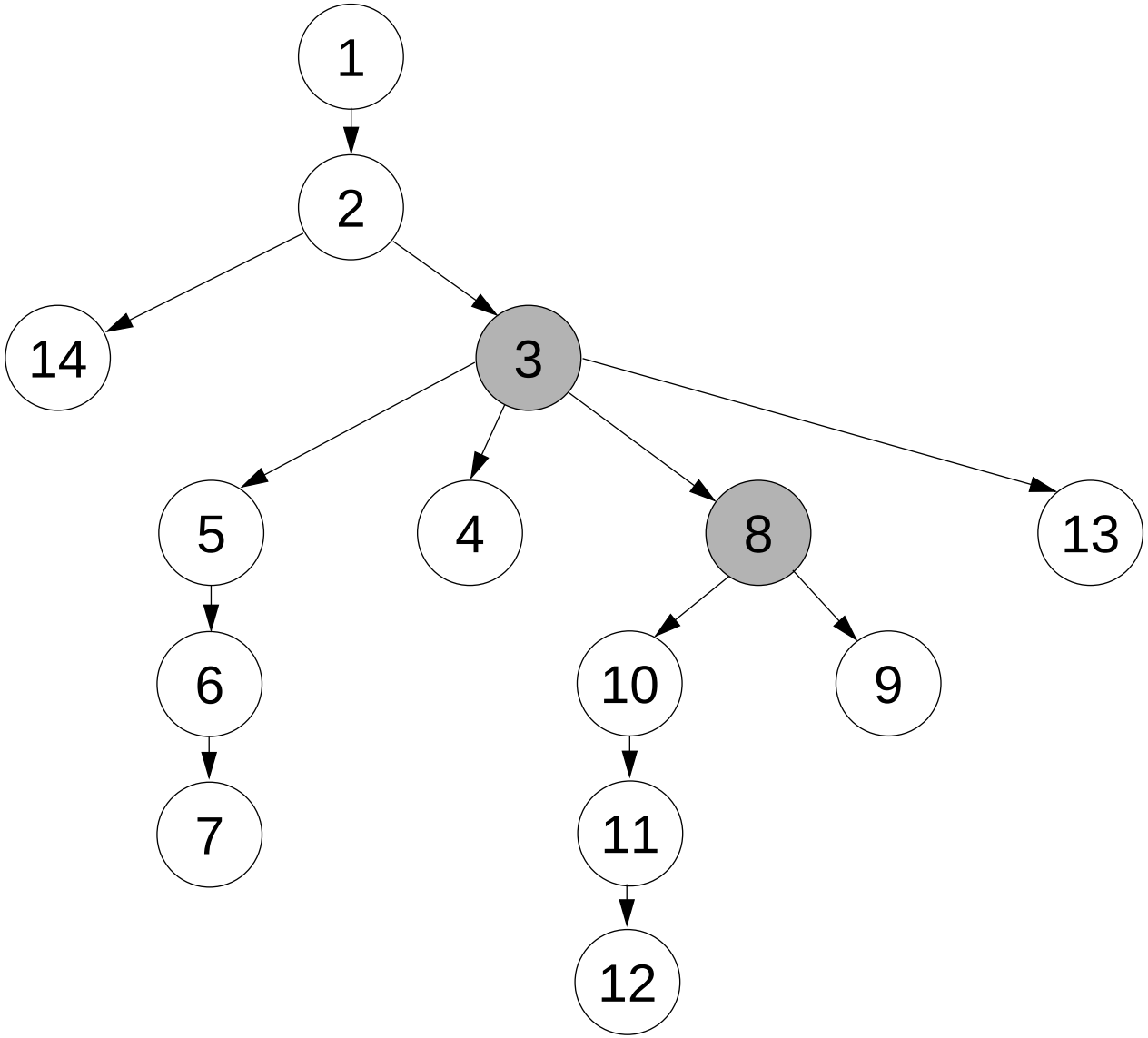
Efficient Coverage Testing

```
exp1;
while (exp2) {
    switch (exp3) {
        case 1: exp4;
                break;
        case 2: exp5;
                while (exp6) exp7;
        default:
            if (exp8) {
                exp9;
                continue;
            }
            do exp10; while (exp11);
            exp12;
    }
    exp13;
}
exp14;
```

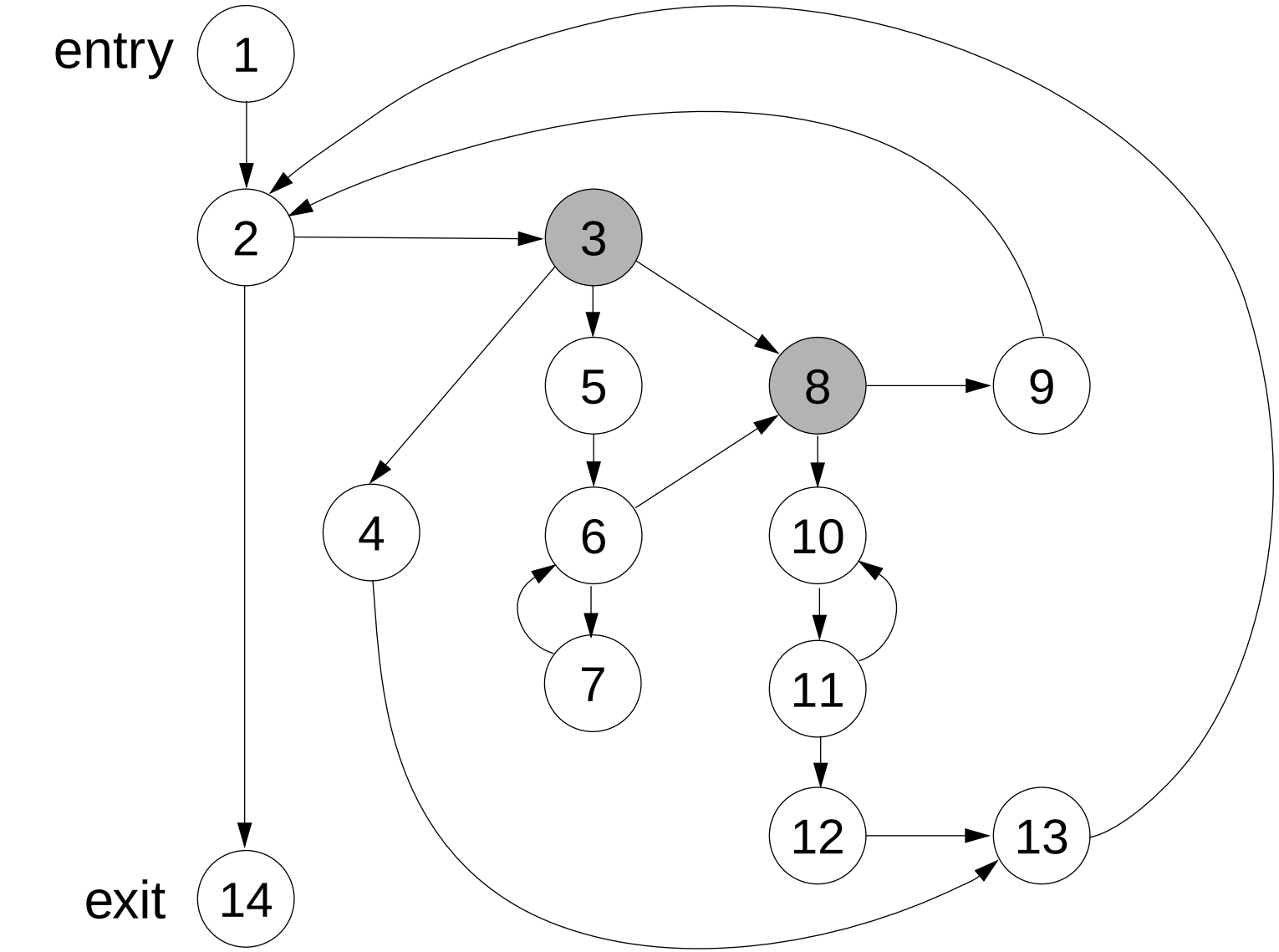
Control Flow Graph



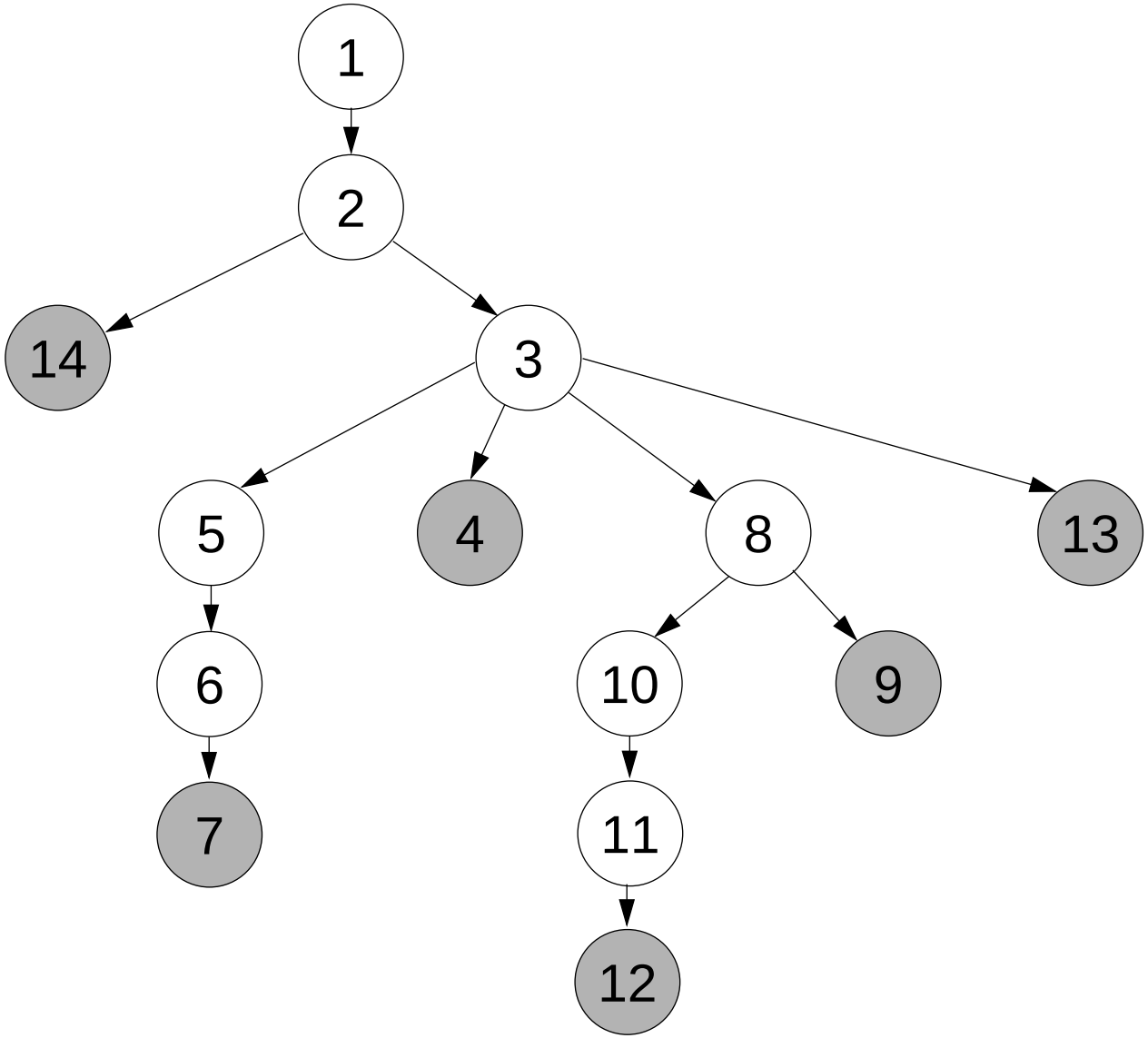
Predominator Tree



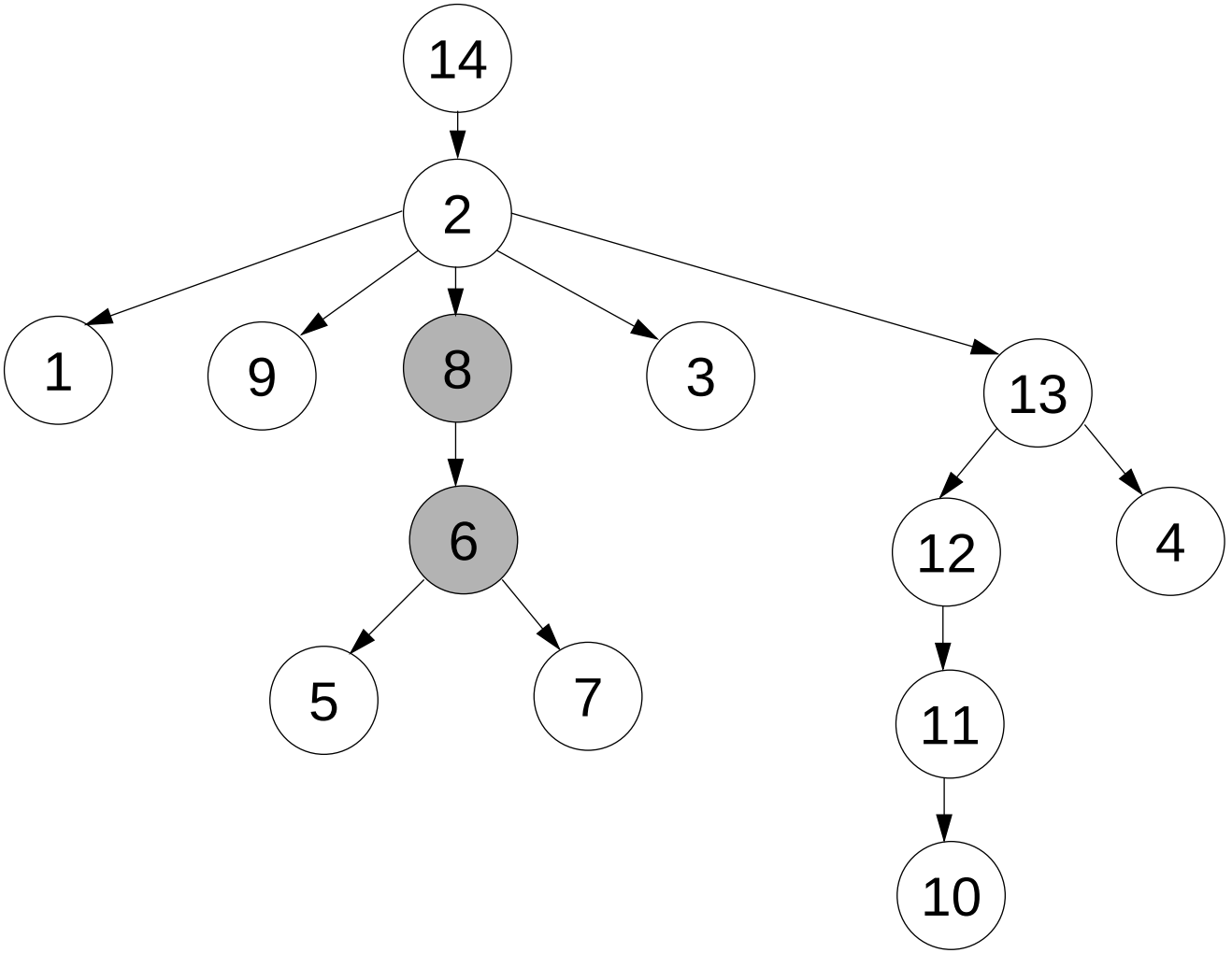
Control Flow Graph



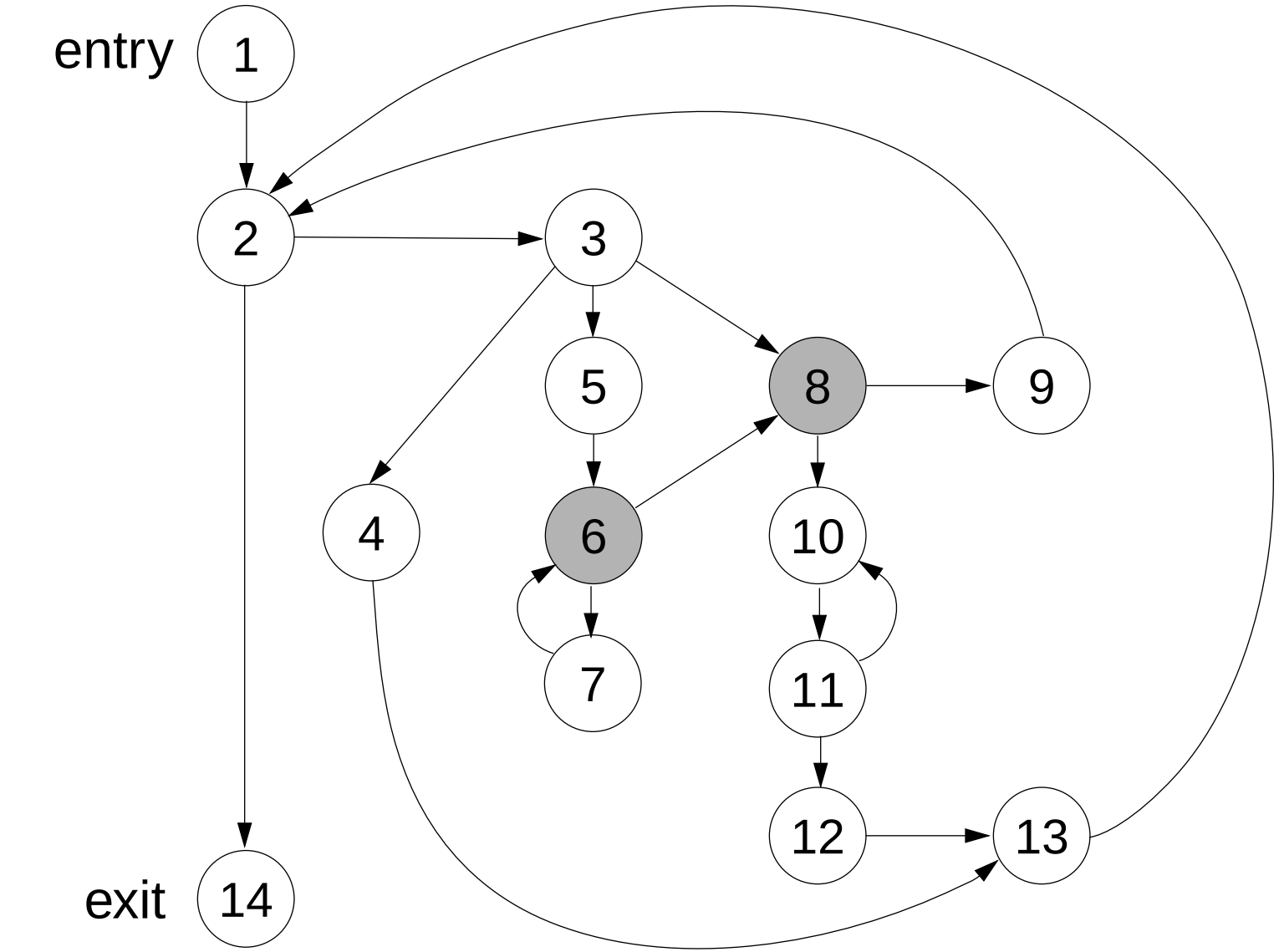
Predominator Tree (cont'd)



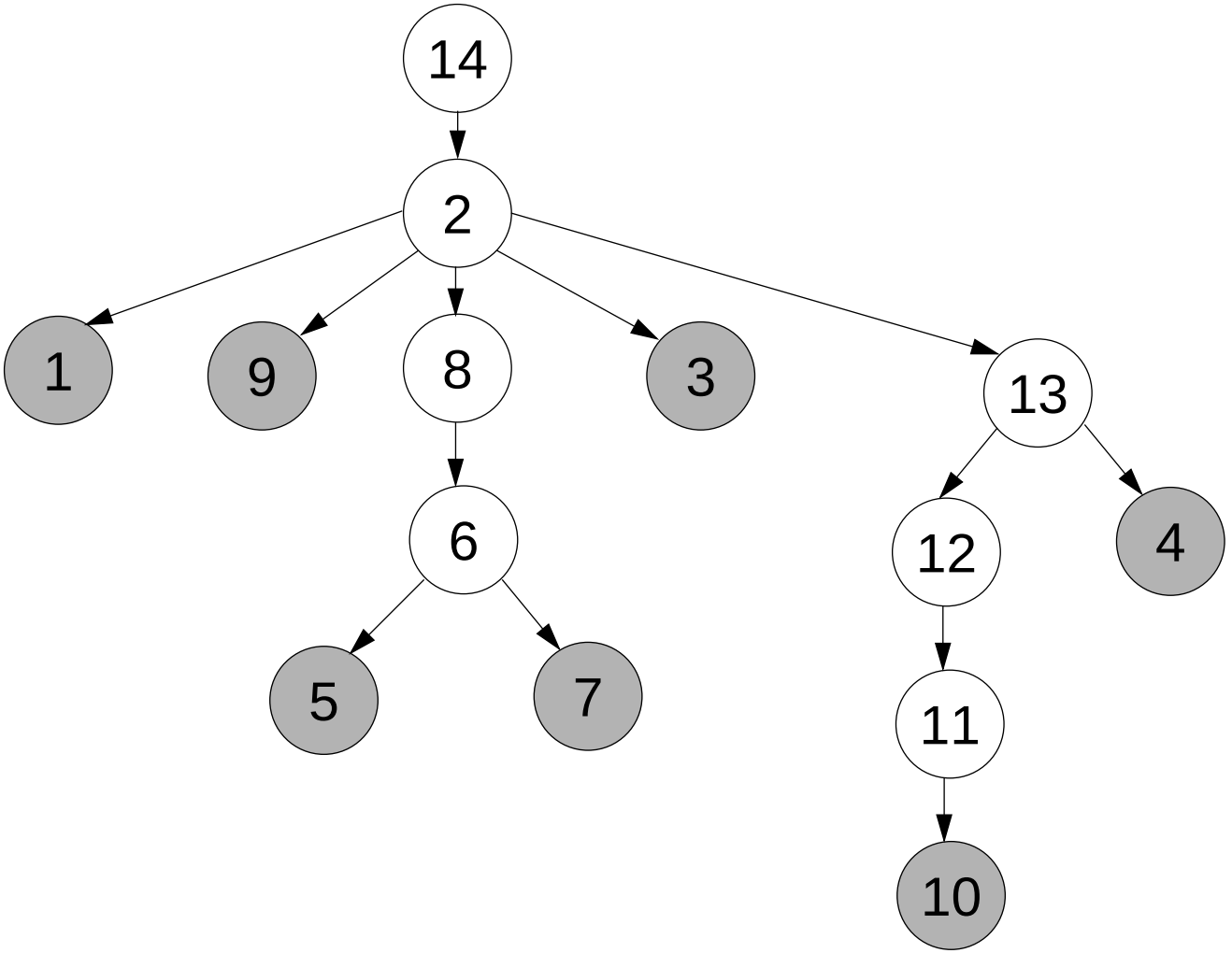
Postdominator Tree



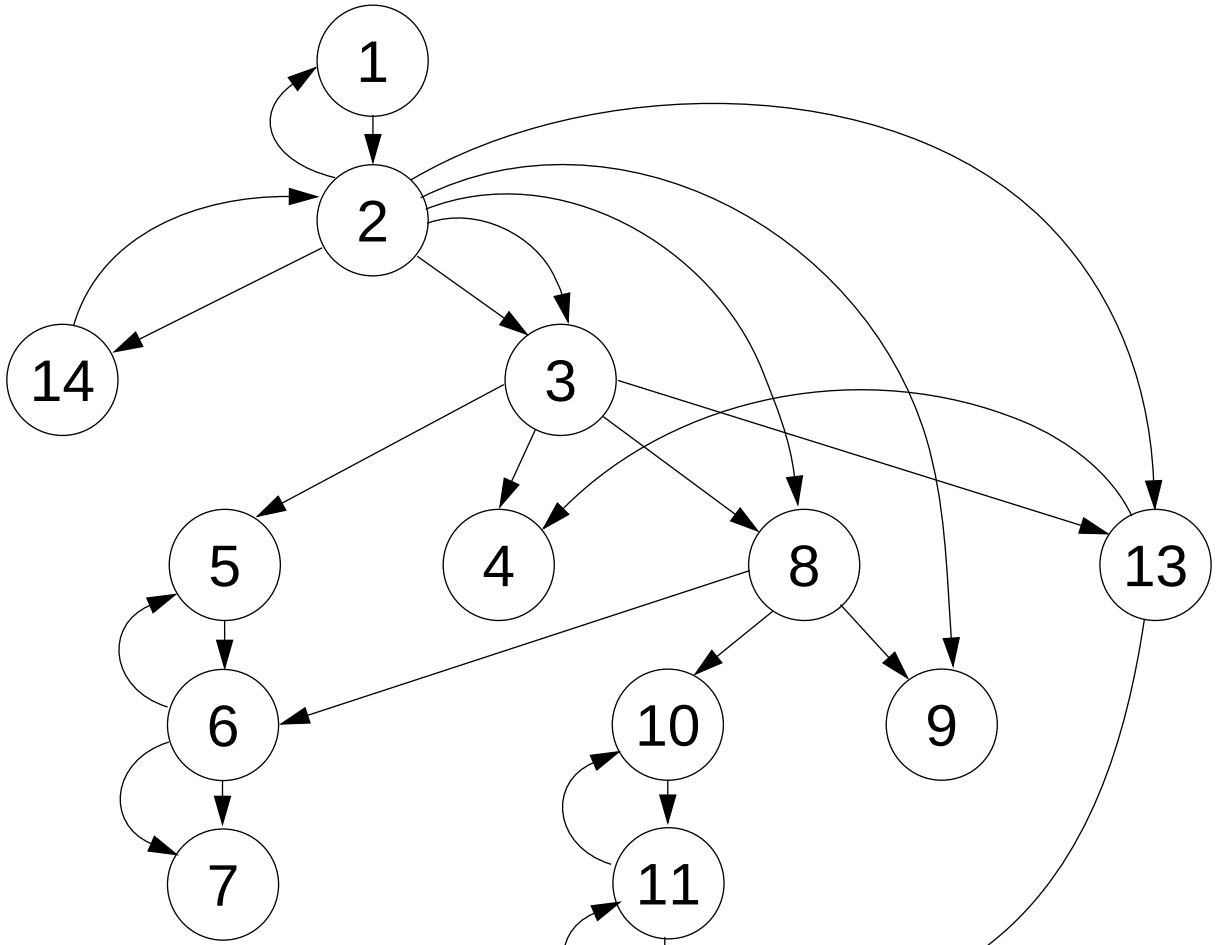
Control Flow Graph



Postdominator Tree (cont'd)

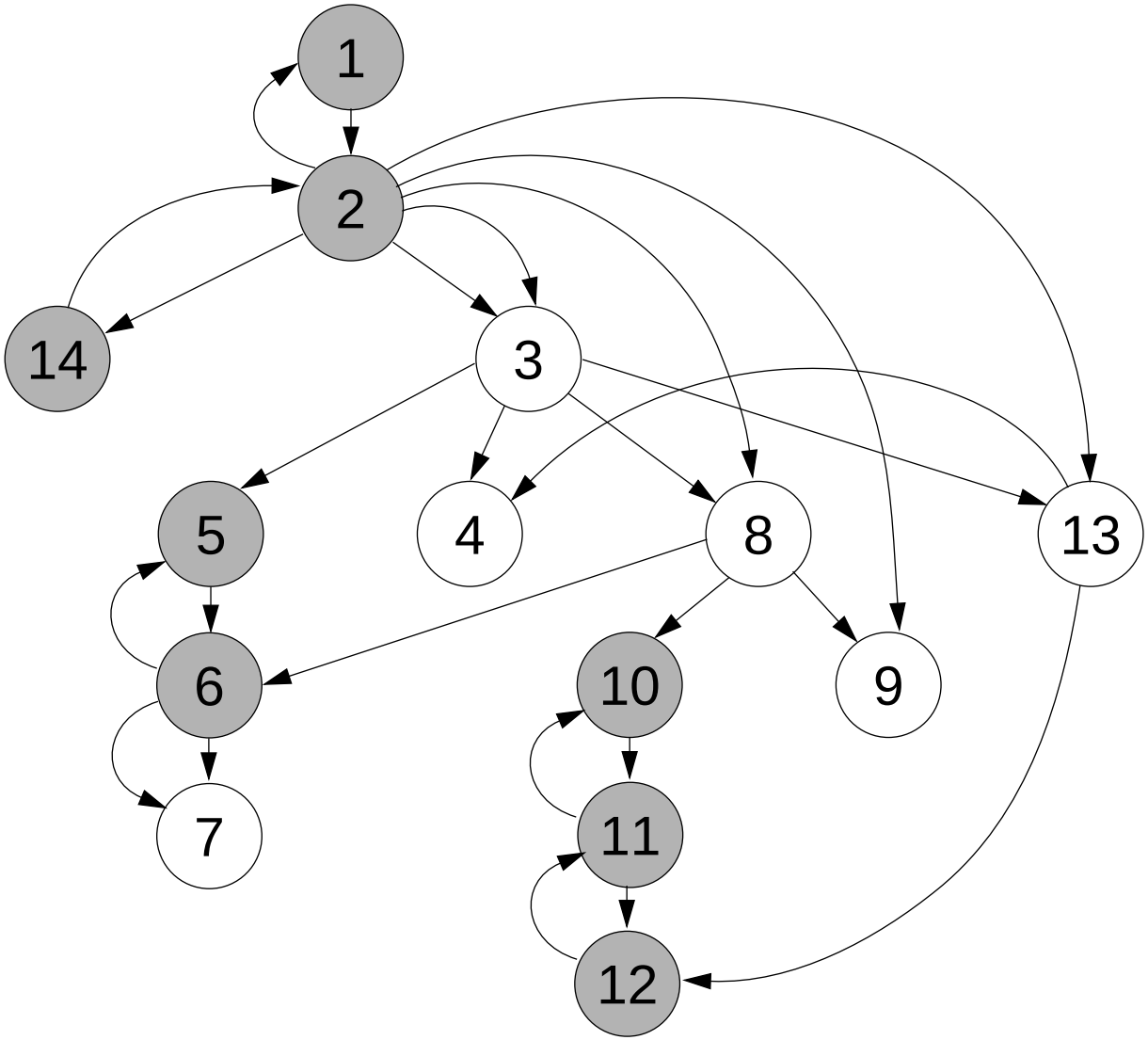


Dominator Graph

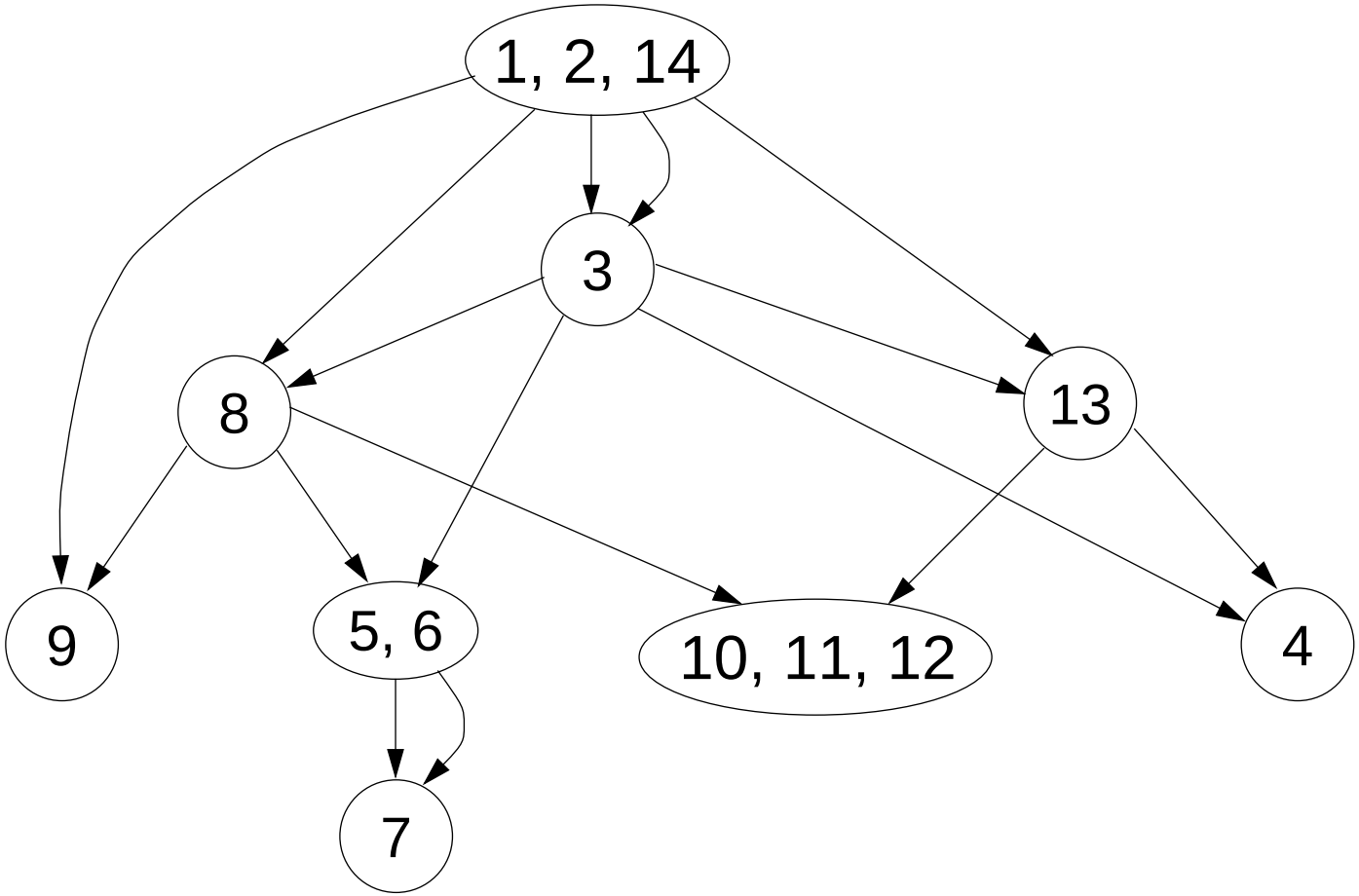


union of pre- & post-dominator trees

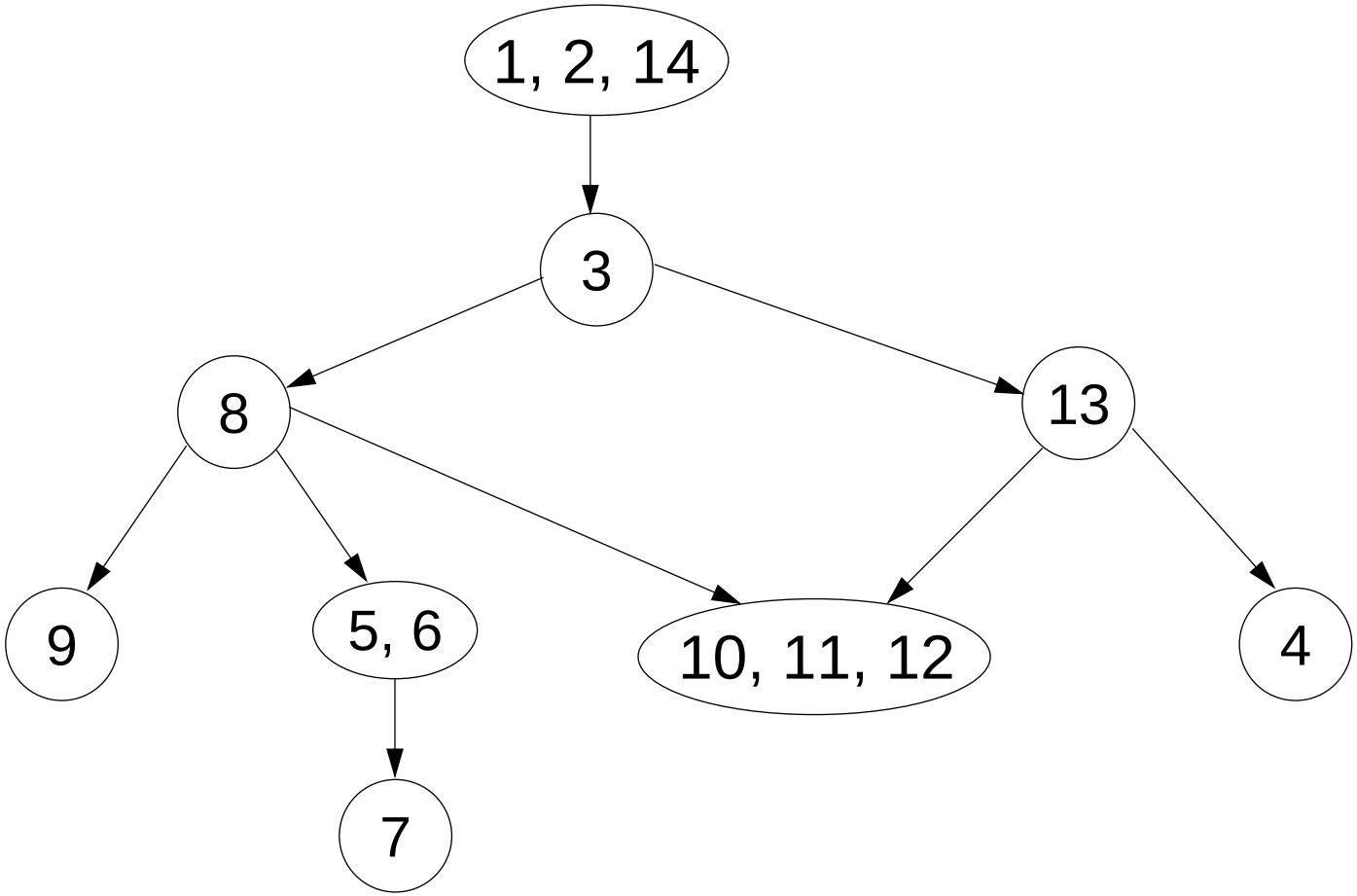
Dominator Graph (cont'd)



Condensed Dominator Graph



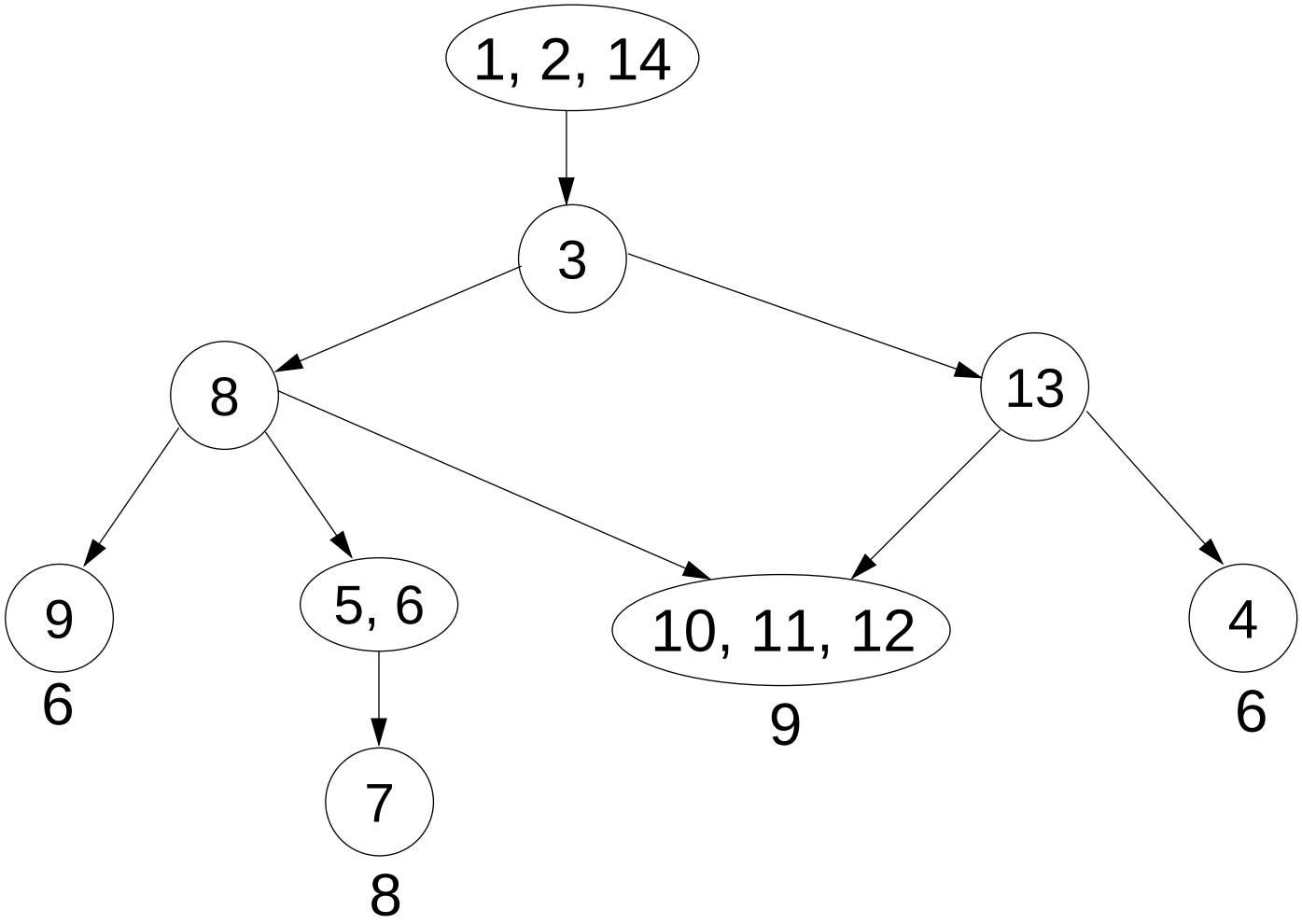
Super Block Dominator Graph



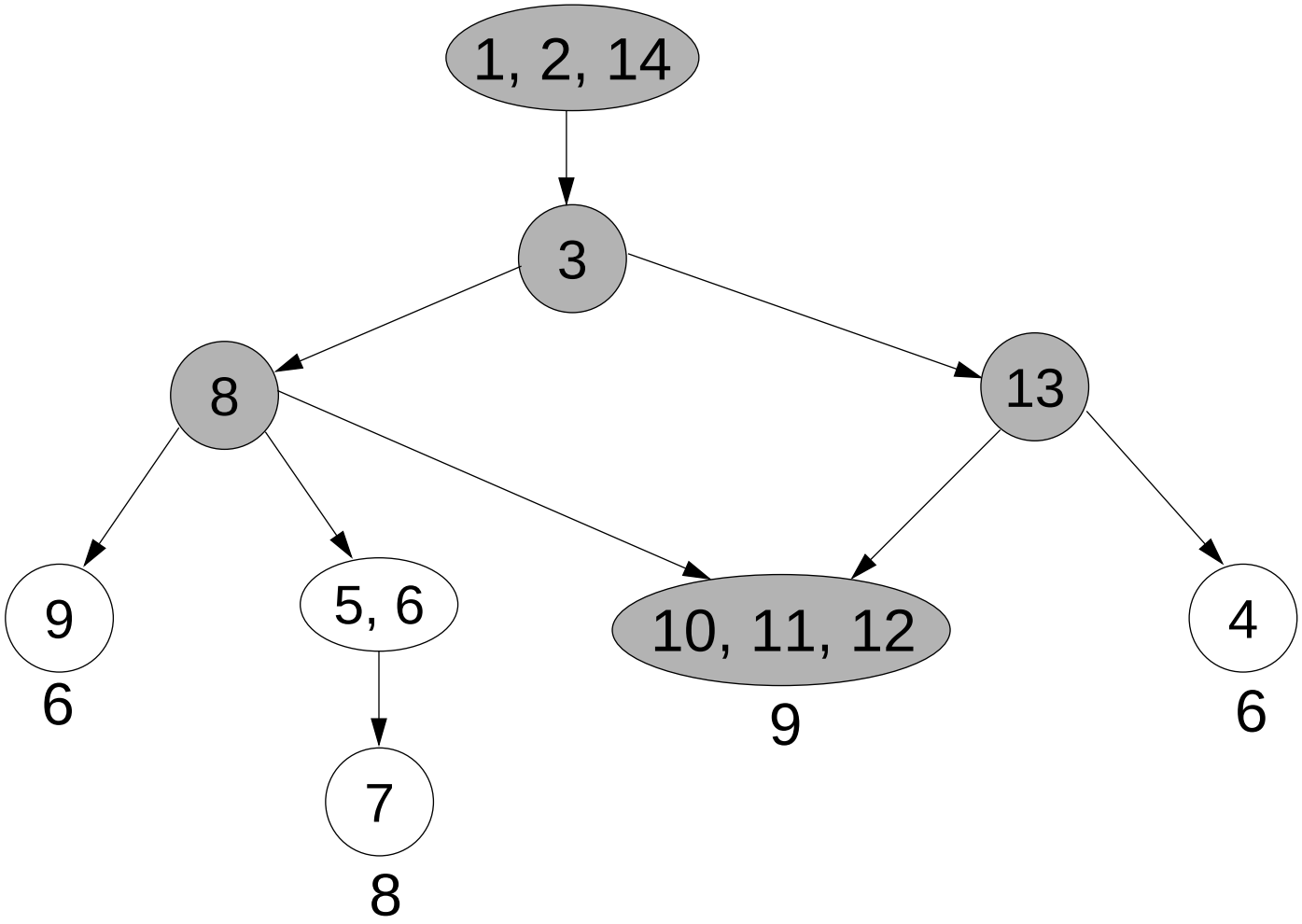
Super Blocks v/s Basic Blocks

- A super block contains one or more basic blocks.
- A super block need not be contiguous.
- If any statement in a super block is visited, then all statements in it must be visited, *provided the execution terminates on that input*.
- We only need one instrument per super block.

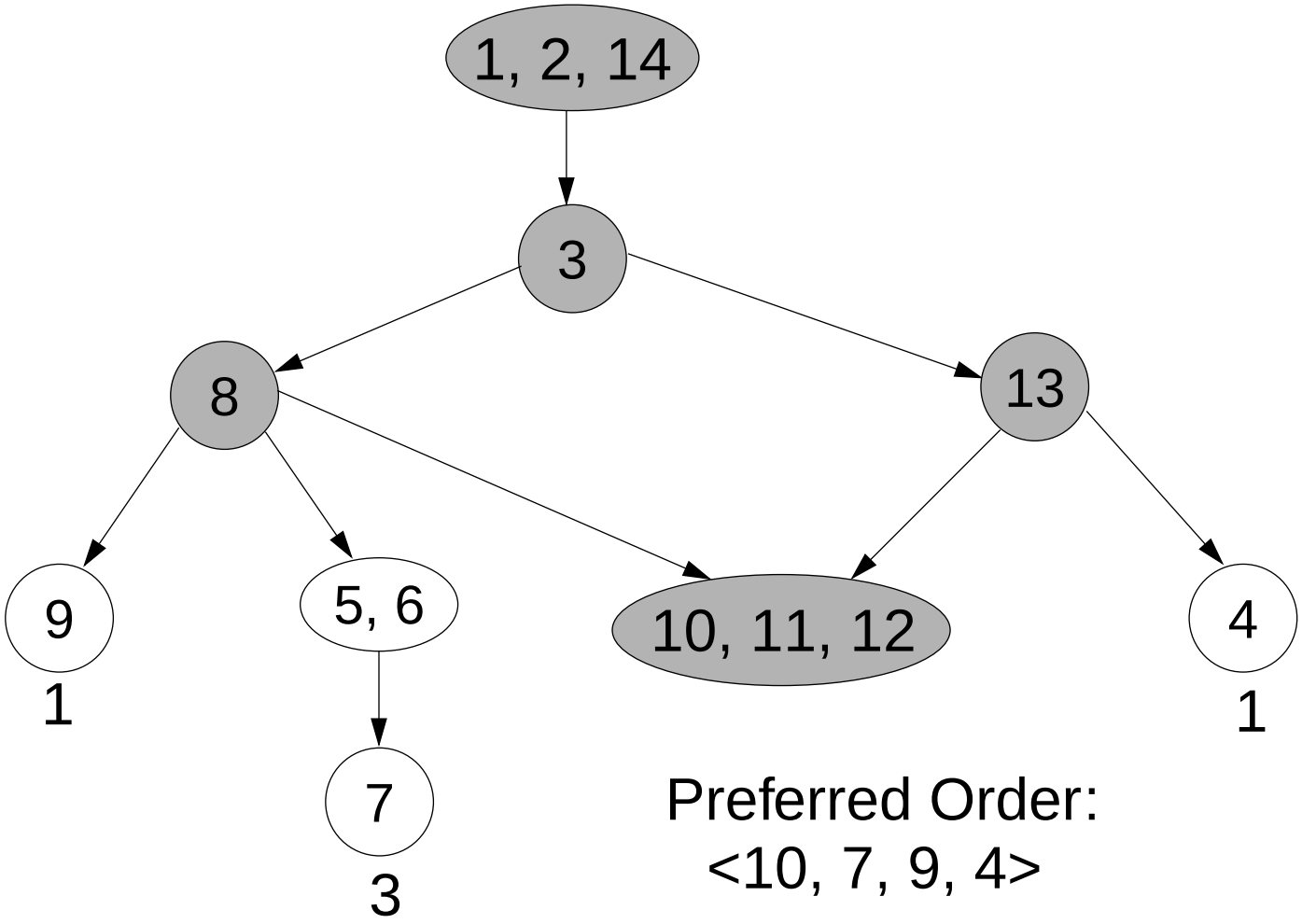
Weighted Super Block Dominator Graph



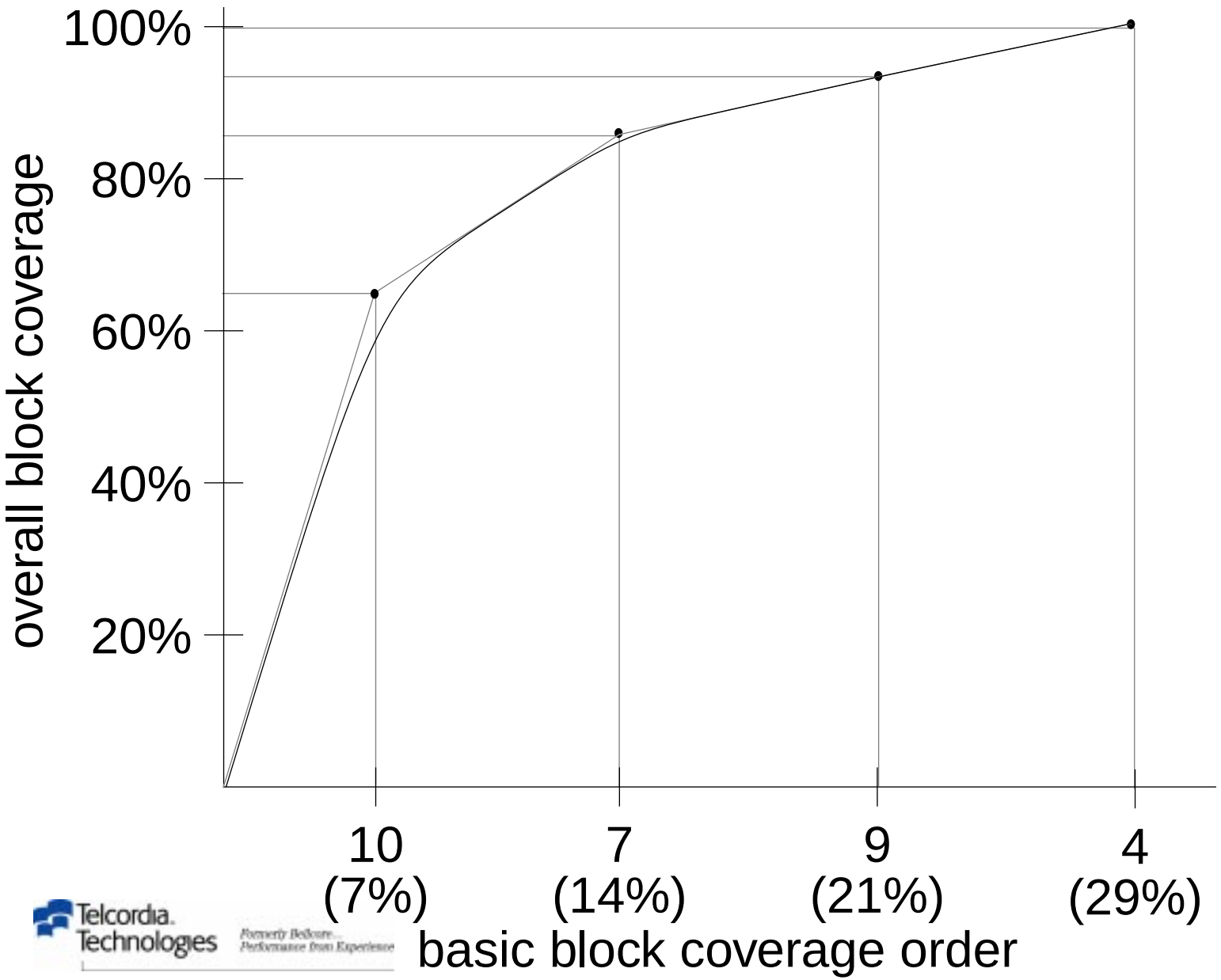
Cover the Heaviest Leaf First



Recompute Weights



Coverage Rate

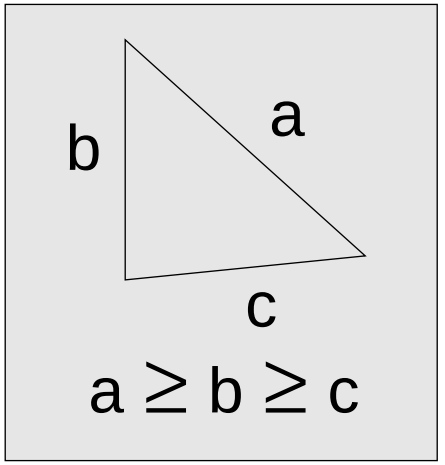


Experimental Results

program	basic blocks	blocks that need to be covered		probes required	
sort	455	138	30%	152	33%
spiff	1266	361	29%	404	32%
mgr	3848	1043	27%	1233	32%
ion	4886	1280	26%	1507	31%
atac	8737	2574	29%	2971	34%
odin	9870	2344	24%	2944	30%
xlib	15580	5111	33%	6016	39%
tvo	17680	6267	35%	8150	46%

Smart Debugging

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right        :  area := b*c / 2;
    equilateral  :  area := a*a * sqrt(3)/4;
    otherwise    :  s := (a+b+c)/2;
                    area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```



Test Suite

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90

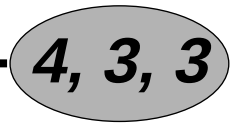
Failure Detected!

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

Where's the Bug?

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right          :  area := b*c / 2;
    equilateral    :  area := a*a * sqrt(3)/4;
    otherwise      :  s := (a+b+c)/2;
                      area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```



Execution Slice w.r.t. T_6 = (4 3 3)

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right       :  area := b*c / 2;
    equilateral :  area := a*a * sqrt(3)/4;
    otherwise   :  s := (a+b+c)/2;
                   area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Failure Detected!

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

Execution Slice w.r.t. Failed Test (T_6)

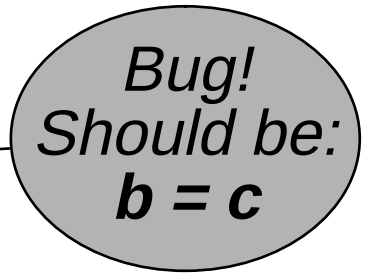
```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right          :  area := b*c / 2;
    equilateral    :  area := a*a * sqrt(3)/4;
    otherwise      :  s := (a+b+c)/2;
                      area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Execution Slice w.r.t. Passed Test, (T_2)

```
read (a, b, c);
class := scalene;
if a = b || b = c
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right       :  area := b*c / 2;
    equilateral :  area := a*a * sqrt(3)/4;
    otherwise   :  s := (a+b+c)/2;
                   area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Resulting Dice [Slice (4 3 3) – Slice (4 4 3)]

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right       :  area := b*c / 2;
    equilateral :  area := a*a * sqrt(3)/4;
    otherwise   :  s := (a+b+c)/2;
                   area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```



Slicing Experiment

- Unix Sort Program
- 508 Basic Blocks
- 25 Independently Seeded Faults
- 56 Tests
- 96% Block Coverage, 89% Branch Coverage
- 6 Faults Not Detected By Any Test

Slicing Experiment Results

fault id	# of dices	# good	avg. size
F117	159	84%	26%
F206	55	93%	11%
F336	783	100%	24%
F414	55	100%	22%
F439	208	94%	18%
F442	495	100%	14%
F507	684	50%	6%
F608	55	100%	31%
F631	423	100%	19%
F634	300	100%	23%

fault id	# of dices	# good	avg. size
F691	735	100%	16%
F772	55	98%	19%
F782	384	79%	11%
F783	384	100%	11%
F806	55	96%	29%
F811	159	85%	23%
F851	768	100%	15%
F882	423	100%	14%
F910	255	86%	15%
Avg.	**338**	**93%**	**18%**

Incremental Regression Testing

```
read (a, b, c);
class := scalene;
if a = b || b = c
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right          :  area := b*c / 2;
    equilateral    :  area := a*a * sqrt(3)/4;
    otherwise      :  s := (a+b+c)/2;
                      area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Which Tests Should We Rerun?

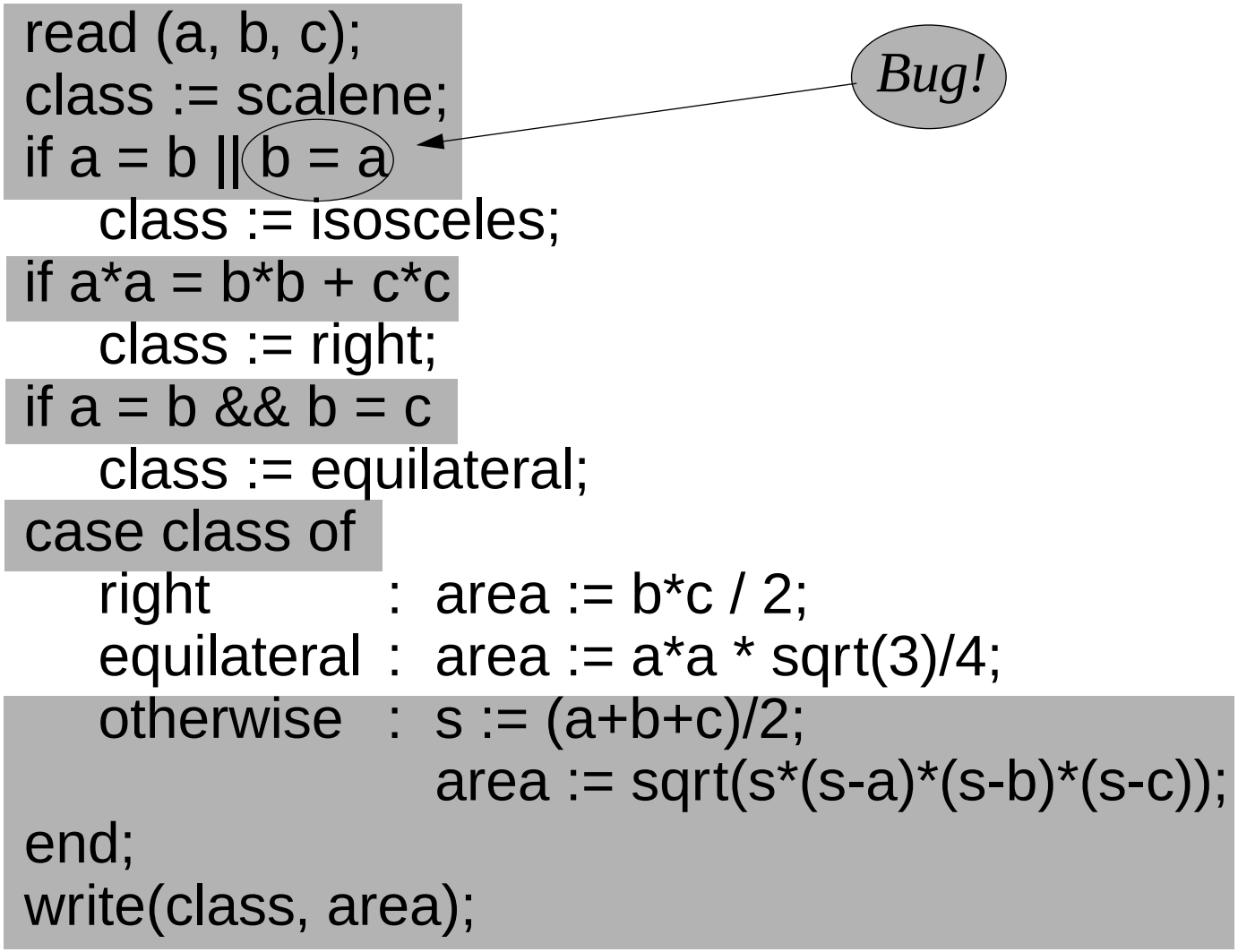
Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

Execution Slice w.r.t. T_2 = (4 4 3)

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right       :  area := b*c / 2;
    equilateral :  area := a*a * sqrt(3)/4;
    otherwise   :  s := (a+b+c)/2;
                   area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Execution Slice w.r.t. T_4 = (6 5 4)



Which Test Cases Should We Rerun?

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

Execution Slice w.r.t. T_3 = (5 4 3)

```
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right          :  area := b*c / 2;
    equilateral    :  area := a*a * sqrt(3)/4;
    otherwise      :  s := (a+b+c)/2;
                      area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Bug!

scalene *Failure!*

Which Test Cases Should We Rerun?

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

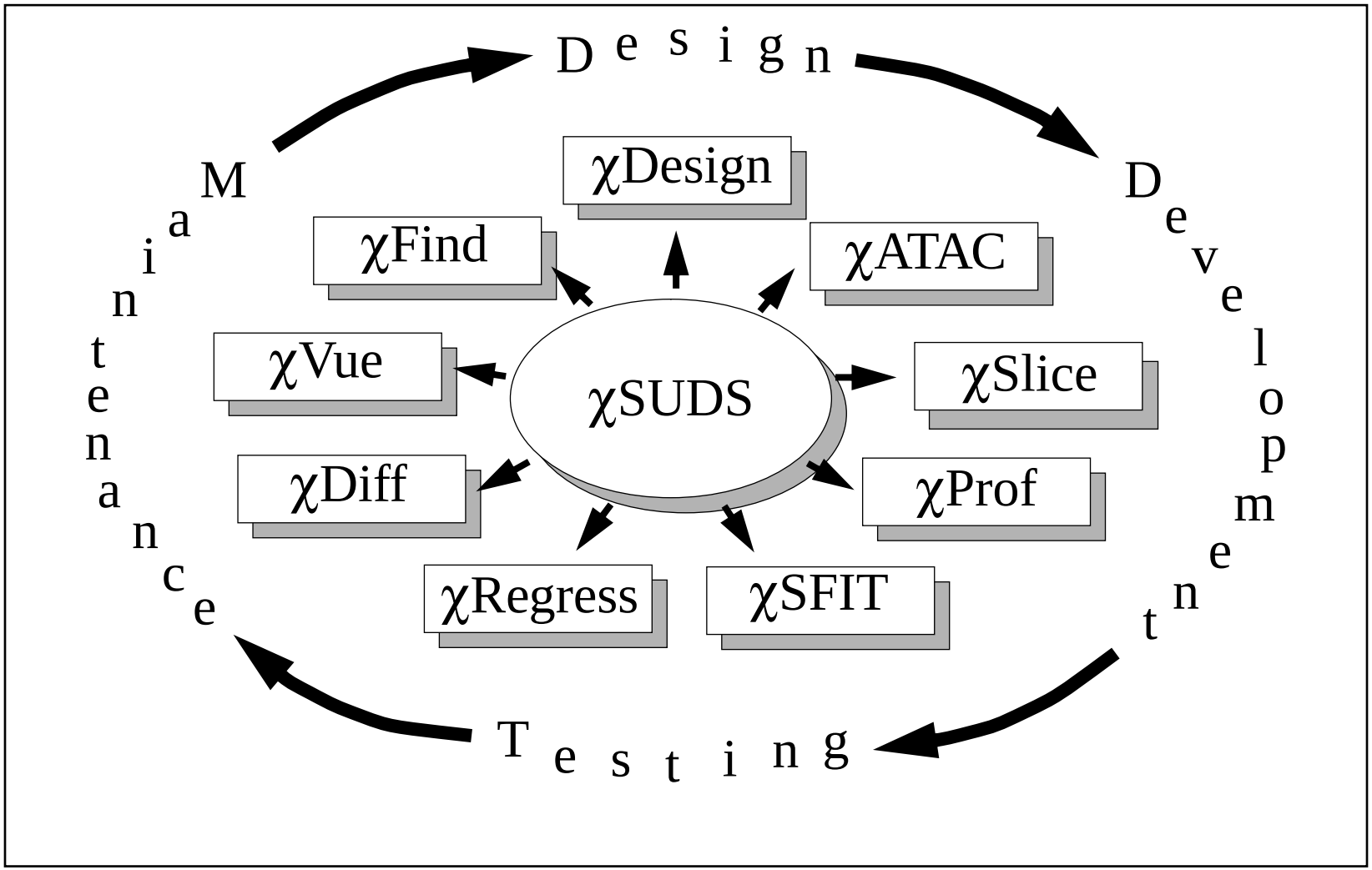
Regression Testing Experiment

- A Space Program from European Space Agency
- About 10,000 lines of C code
- 10 Actual Faults Obtained from Its Error Logs
- 1,000 Regression Tests With Operational Profile

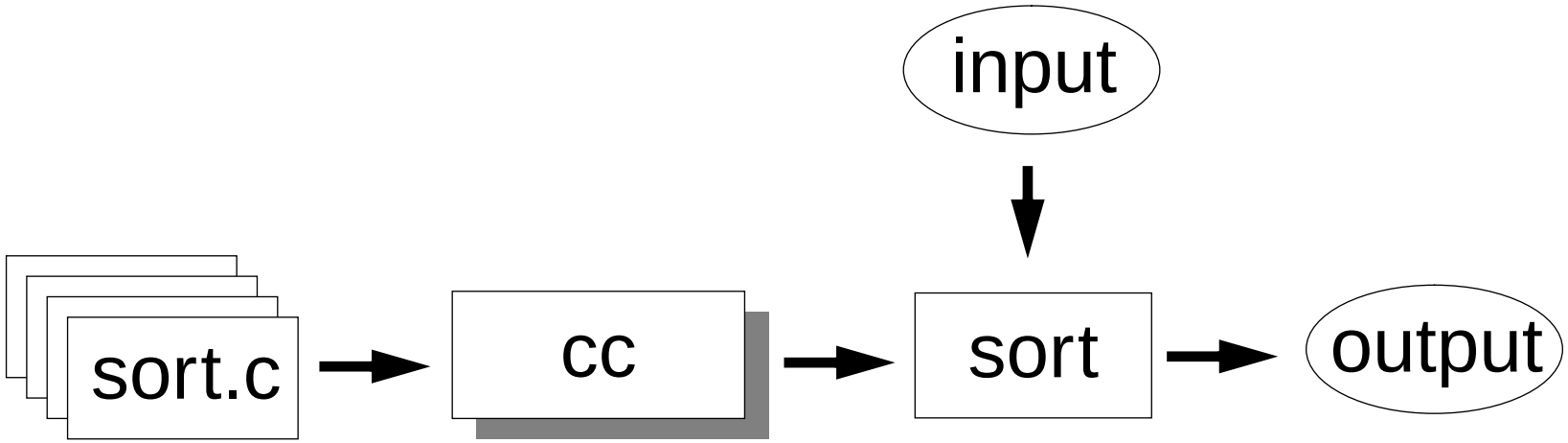
Regression Testing Experiment Results

fault id	# tests rerun	% tests rerun	minimum # tests
F1	38	3.8%	26
F2	22	2.2%	16
F3	84	8.4%	36
F4	4	0.4%	4
F5	4	0.4%	4
F6	38	3.8%	32
F7	38	3.8%	32
F8	34	3.4%	6
F9	71	7.1%	46
F10	470	47%	320

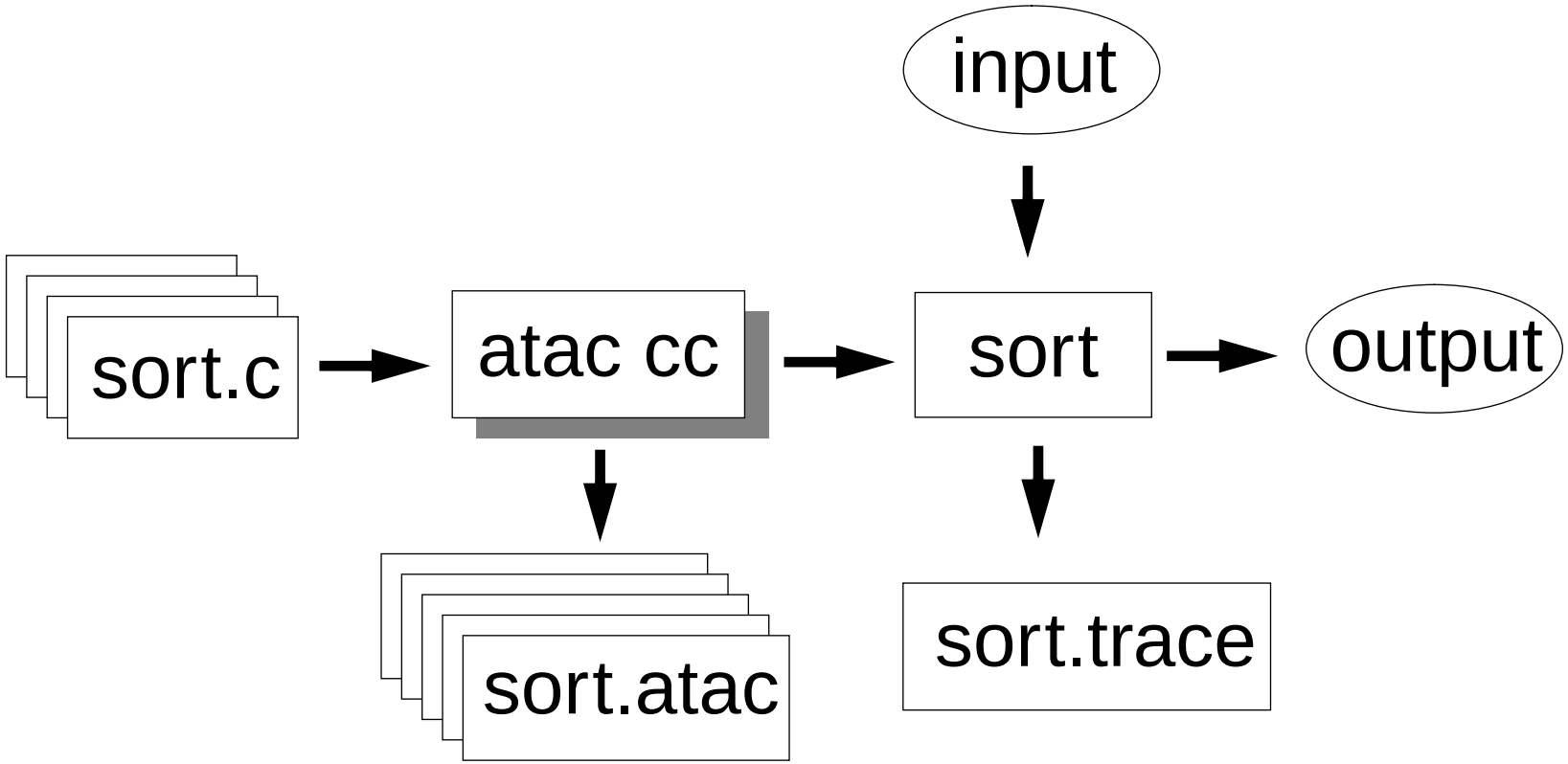
χSuds ToolSuite



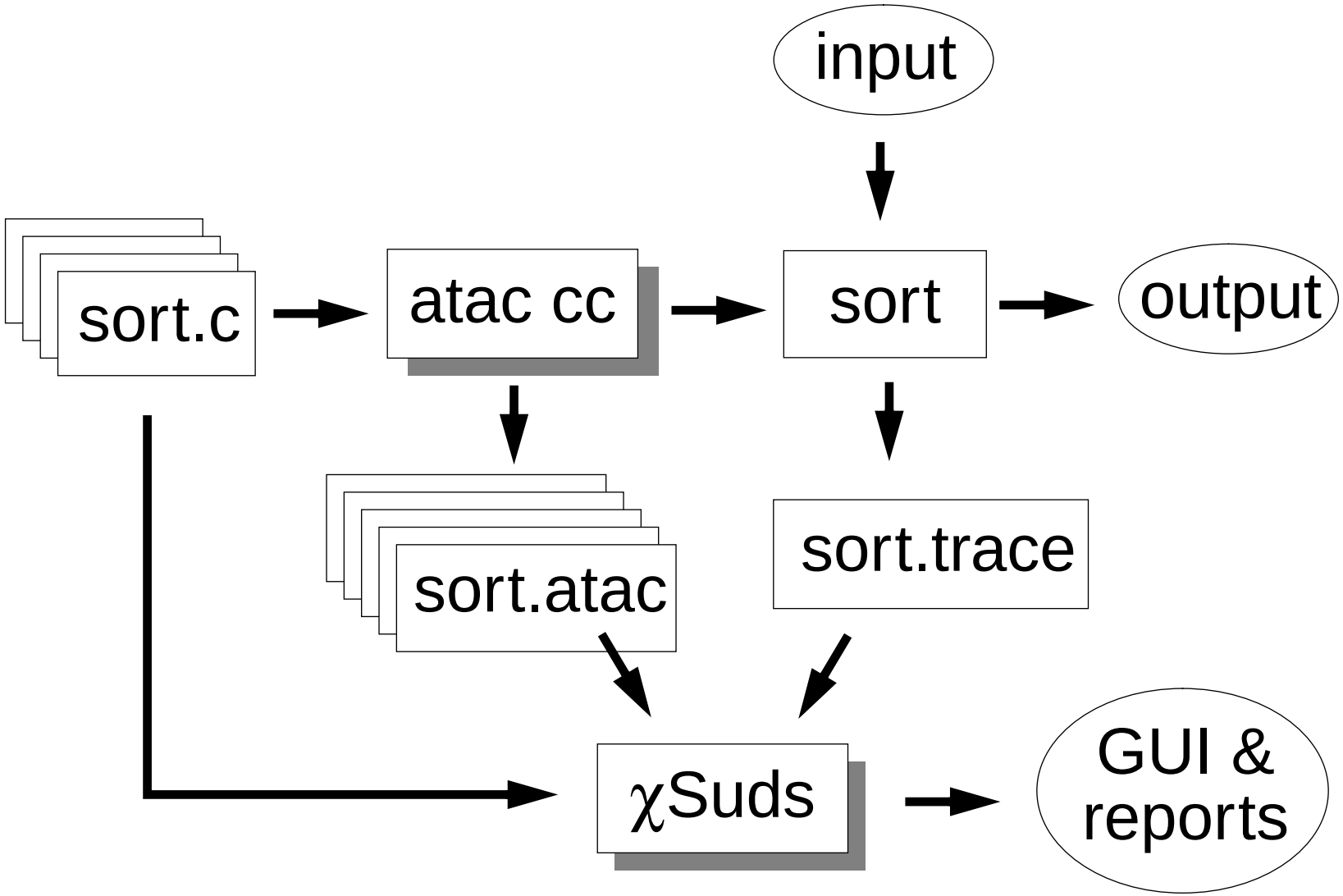
Program Instrumentation



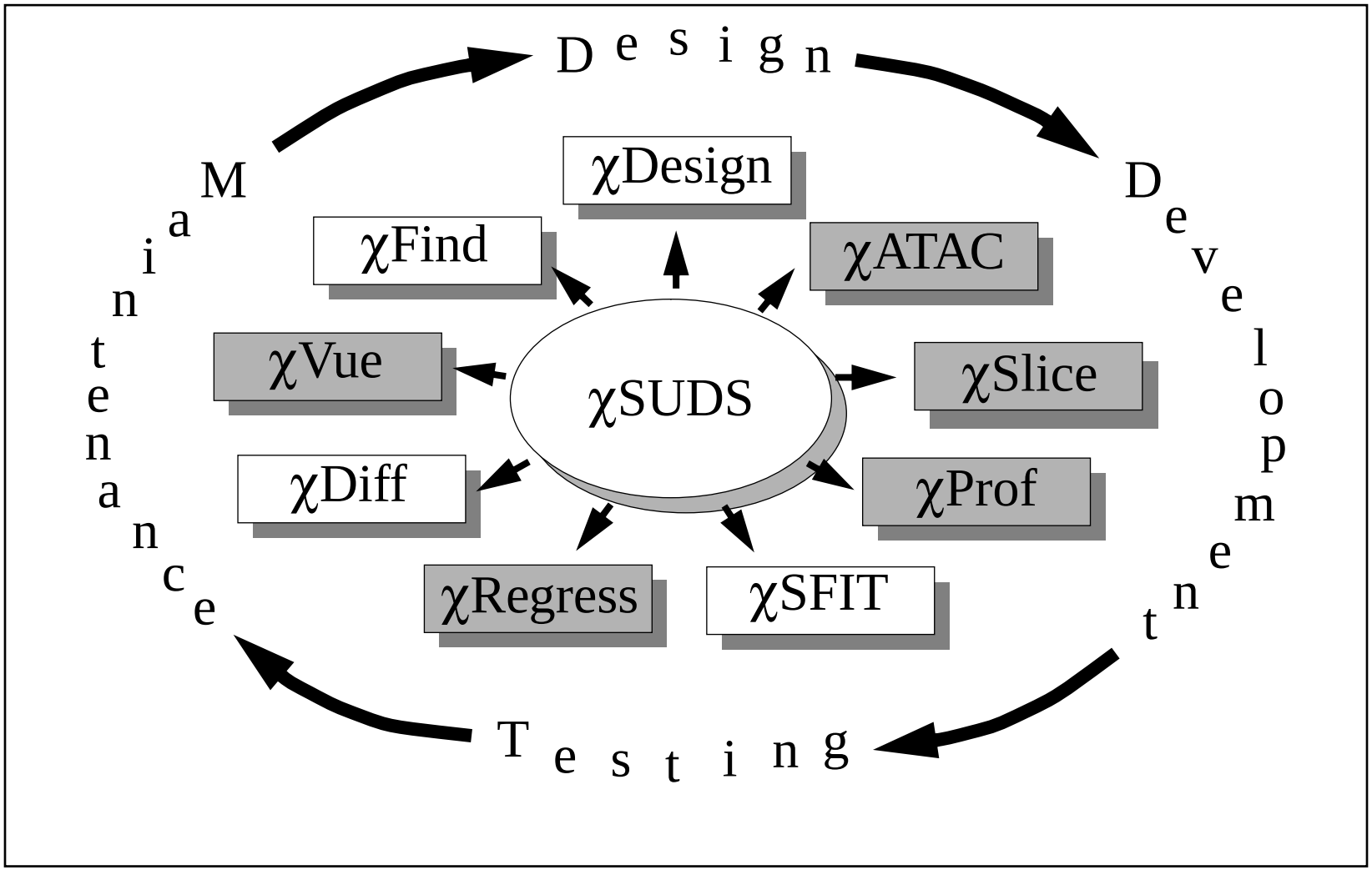
Program Instrumentation (cont'd)



Program Instrumentation (cont'd)



A Short Demo


Design
Development
Testing
Maintenance
χDesign
χFind
χATAC
χVue
χSUDS
χSlice
χDiff
χProf
χRegress
χSFIT

For More Information:

- www.research.telcordia.com/demos/#xsuds
- xsuds.argreenhouse.com
- www.research.telcordia.com/papers/hira

Future Directions

- Smart Program Diffs and Incremental Analysis
- More Empirical Studies
- More Advanced Heuristics
- High Level User Interfaces
- Automatic Test Case Generation