

Chopped Symbolic Execution

David Trabish, Andrea Mattavelli, Noam Rinetzky and Cristian Cadar

**KLEE Workshop, London, UK
19-20 April, 2018**

Symbolic Execution: Main Challenges

Path Explosion

Constraint Solving

Symbolic Execution: Main Challenges

Path Explosion

Constraint Solving

Motivating Example: CVE-2015-3622

- An *out-of-bound-read* vulnerability in **GNU libtasn1**
- GNU Libtasn1 is a library for decoding/encoding ASN.1 data
 - Used in *GnuTLS*

Motivating Example: CVE-2015-3622

Two intertwined flows:

- **Parsing**
- **AST Building**

```
len2 = decode_length(...);  
while (counter < str_len) {  
    len2 = decode_length(...);  
    if (len2 >= 0) {  
        append_value(...);  
    } else {  
        result = extract_octet(...);  
    }  
    ...  
}
```



Motivating Example: CVE-2015-3622

Key Observation:

- **Blue** flow performs **symbolically expensive** operations
- Can be avoided on most of the paths



Chopped Symbolic Execution: Main Idea

- Exploration skips **unwanted** functions
 - User specified
 - Expensive to analyze
- Resolving side effects on demand

Chopped SE by Example

```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

```
void f() {  
    if (k > 0)  
        x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0  
}
```

```
int j; // symbolic  
int k; // symbolic  
int x = 0;  
int y = 0;
```

Chopped SE by Example

```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

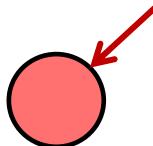
```
void f() {  
    if (k > 0)  
        x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0  
}
```

Chopped SE by Example

```
void main() {  
    f();  
    if (j > 0)  
        if (y) // Chopped  
            bug();  
}
```

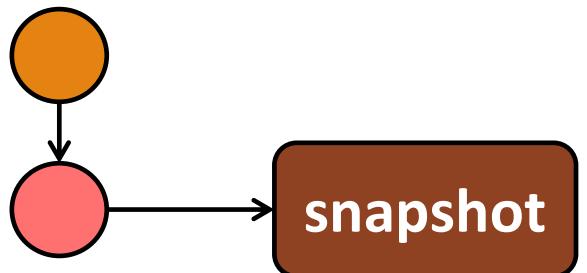
```
void f() {  
    if (k > 0)  
        x = 1;  
    else  
        if (j > 0)  
            y = 1; // Chopped  
        else  
            y = 0  
}
```

Chopped SE by Example



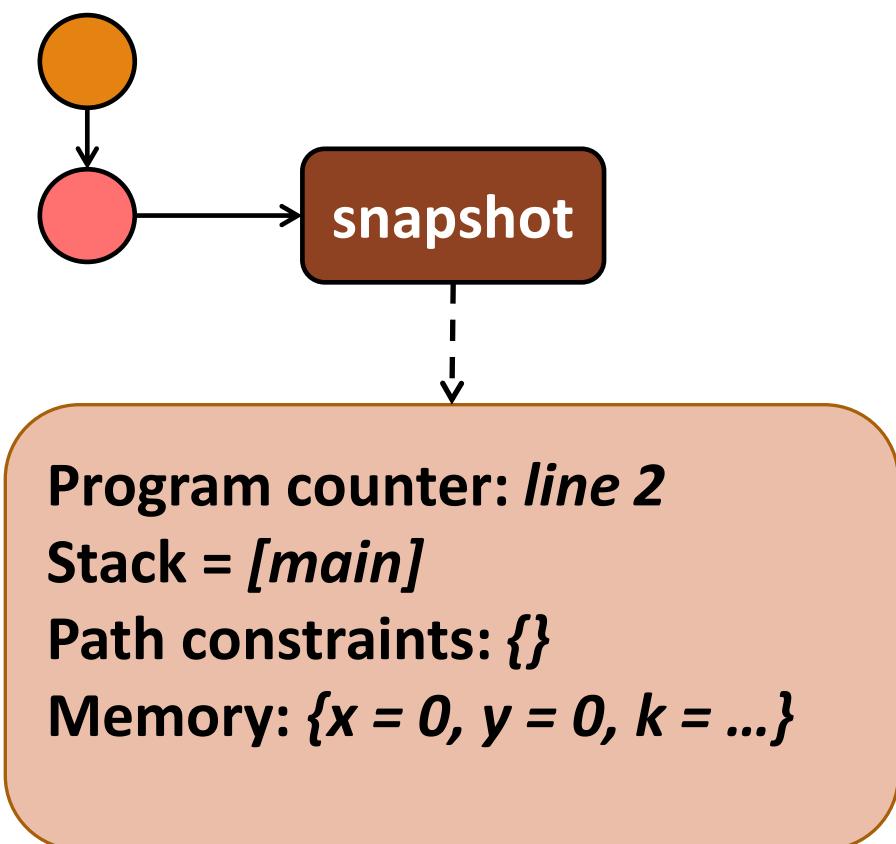
```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

Chopped SE by Example



```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

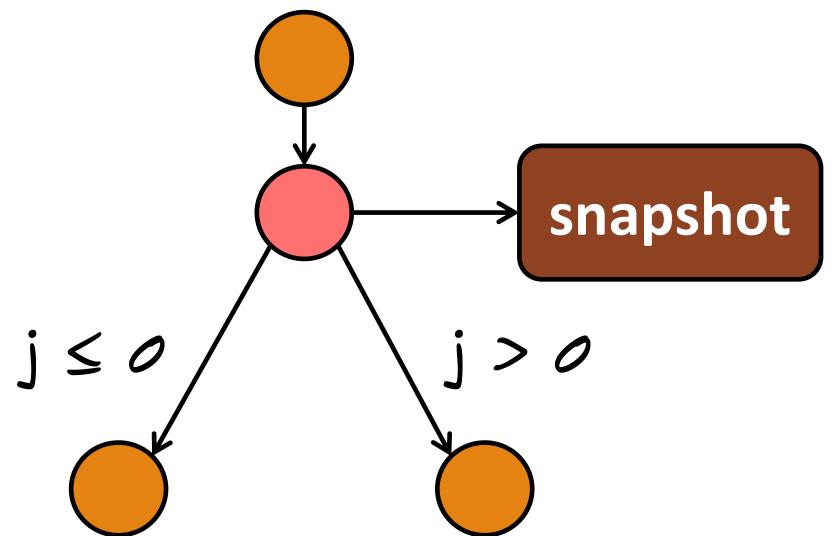
Chopped SE by Example



→

```
void main() {
    f();
    if (j > 0)
        if (y)
            bug();
}
```

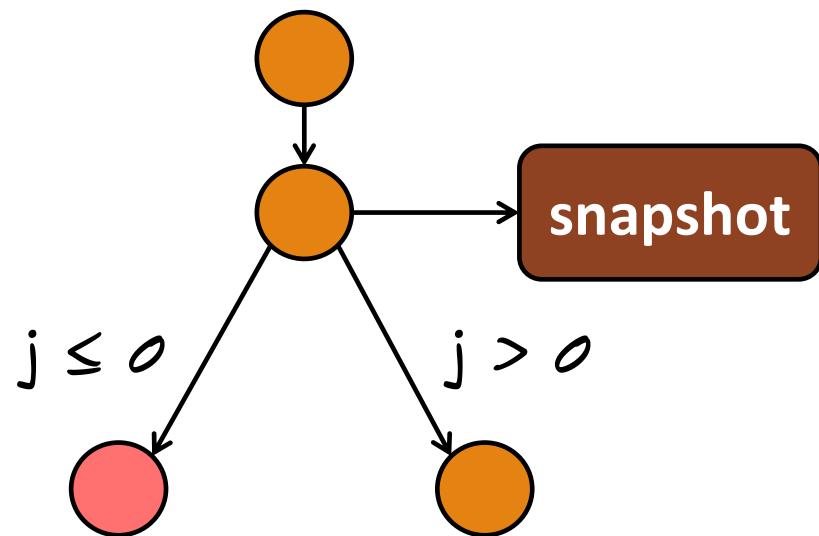
Chopped SE by Example



→

```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

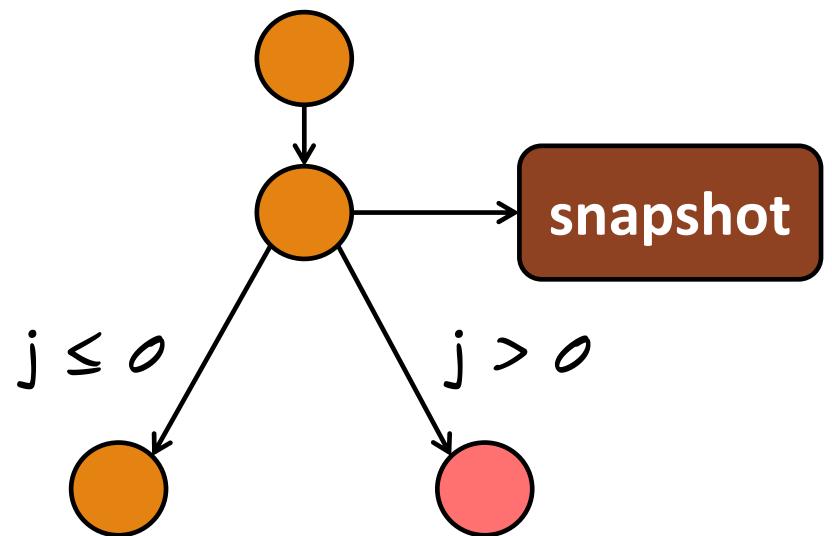
Chopped SE by Example



```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

A red arrow points from the "snapshot" node in the diagram to the "if (y)" line in the code snippet, indicating a specific execution path or state being highlighted.

Chopped SE by Example

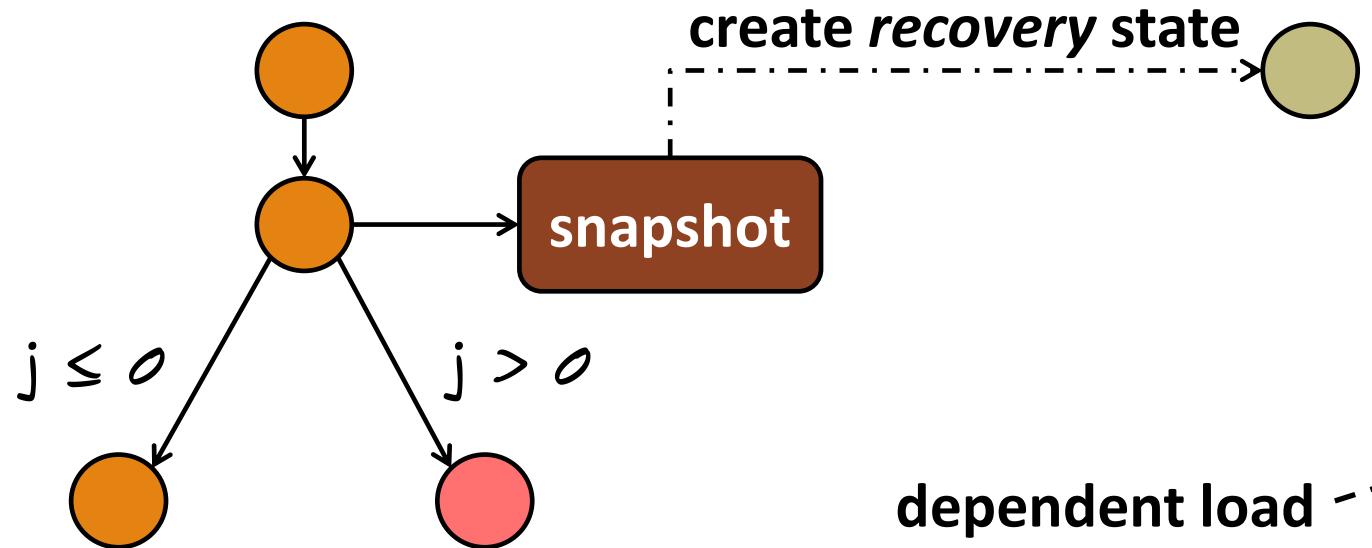


dependent load

```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

A code snippet in C-like syntax. It defines a `main` function containing a call to `f()`. It then checks if `j > 0`. If true, it checks if `y`. If true, it calls `bug()`. A red arrow points from the "snapshot" node in the graph to the first `if (j > 0)` statement. A dashed arrow points from the label "dependent load" to the inner `if (y)` statement.

Chopped SE by Example

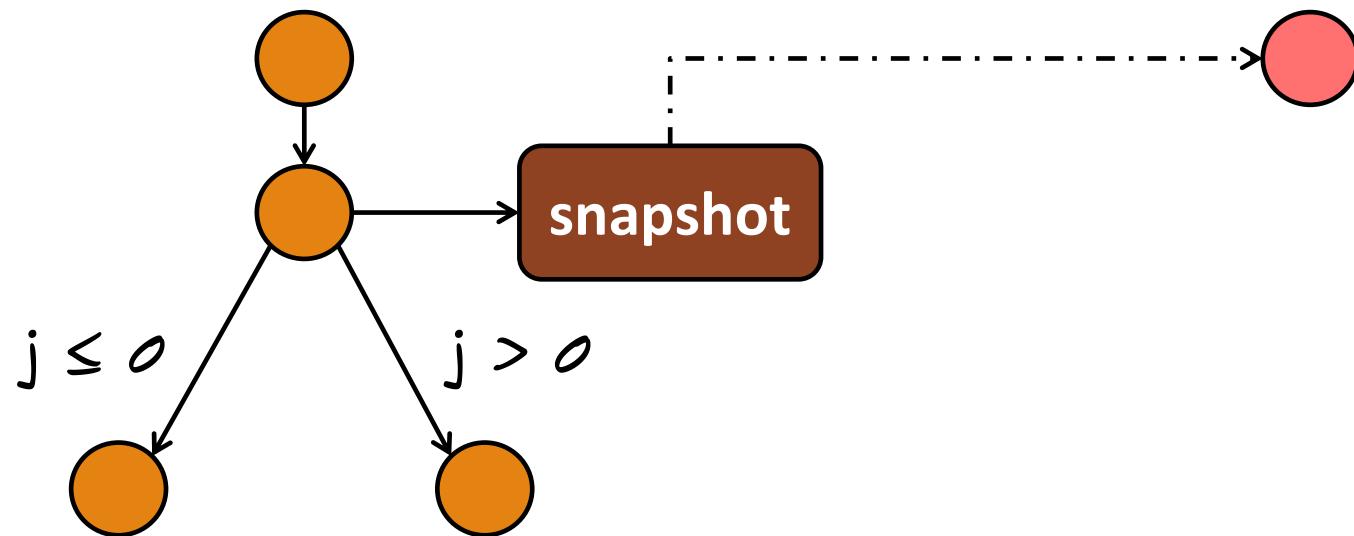


create recovery state

dependent load

```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

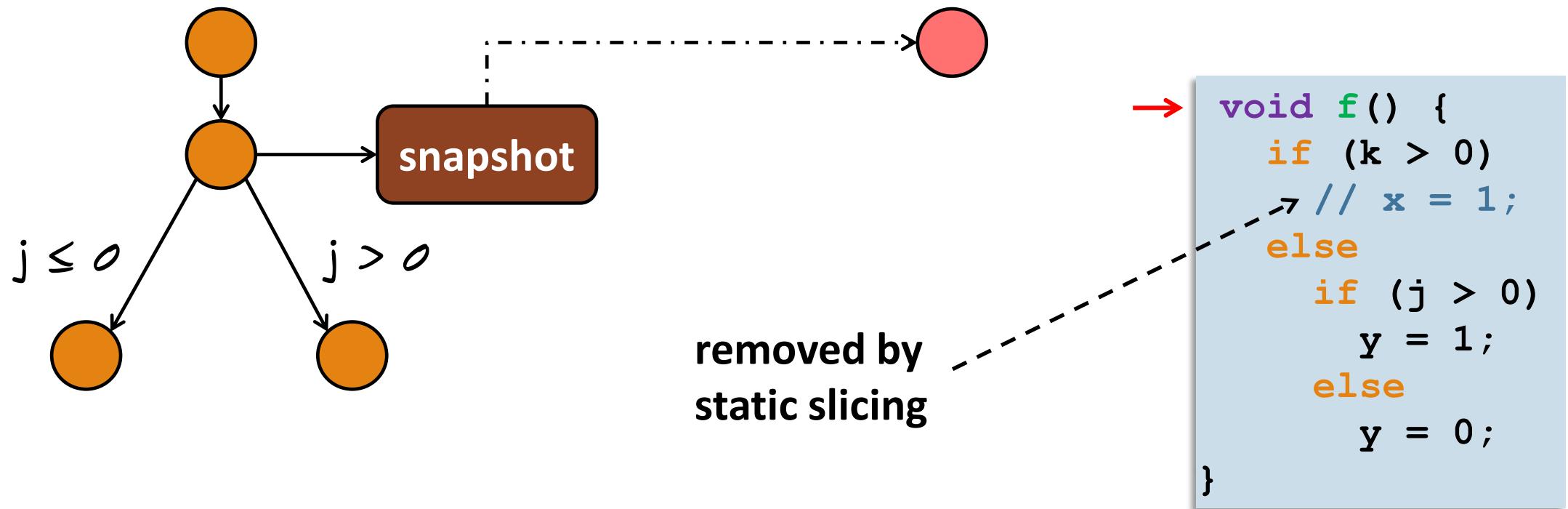
Chopped SE by Example



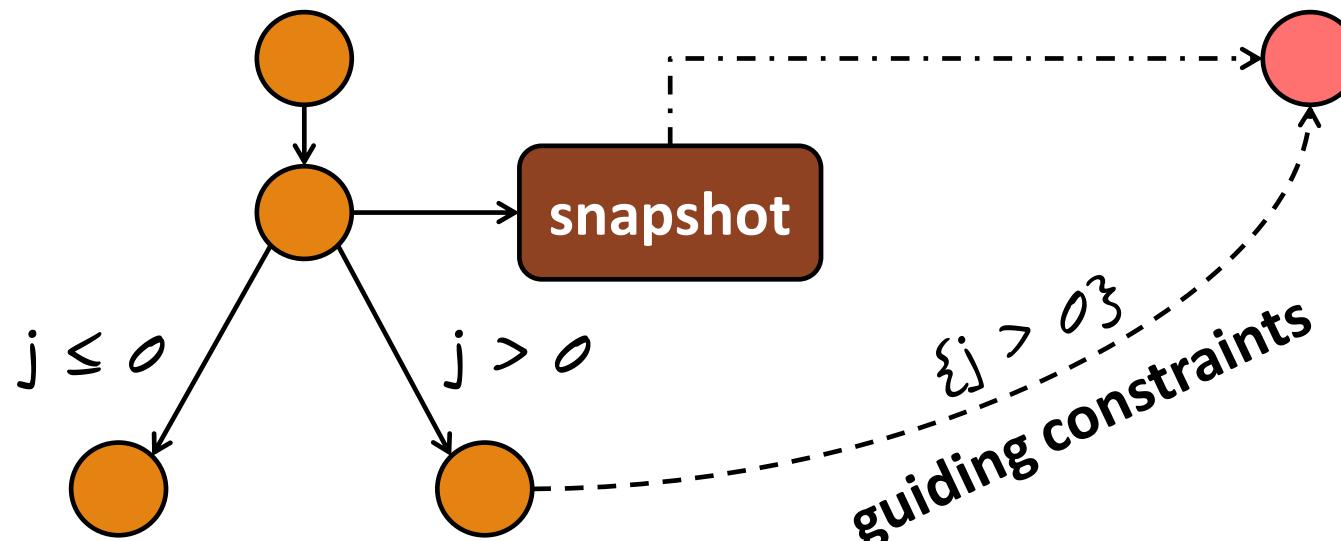
→

```
void f() {  
    if (k > 0)  
        x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

Chopped SE by Example

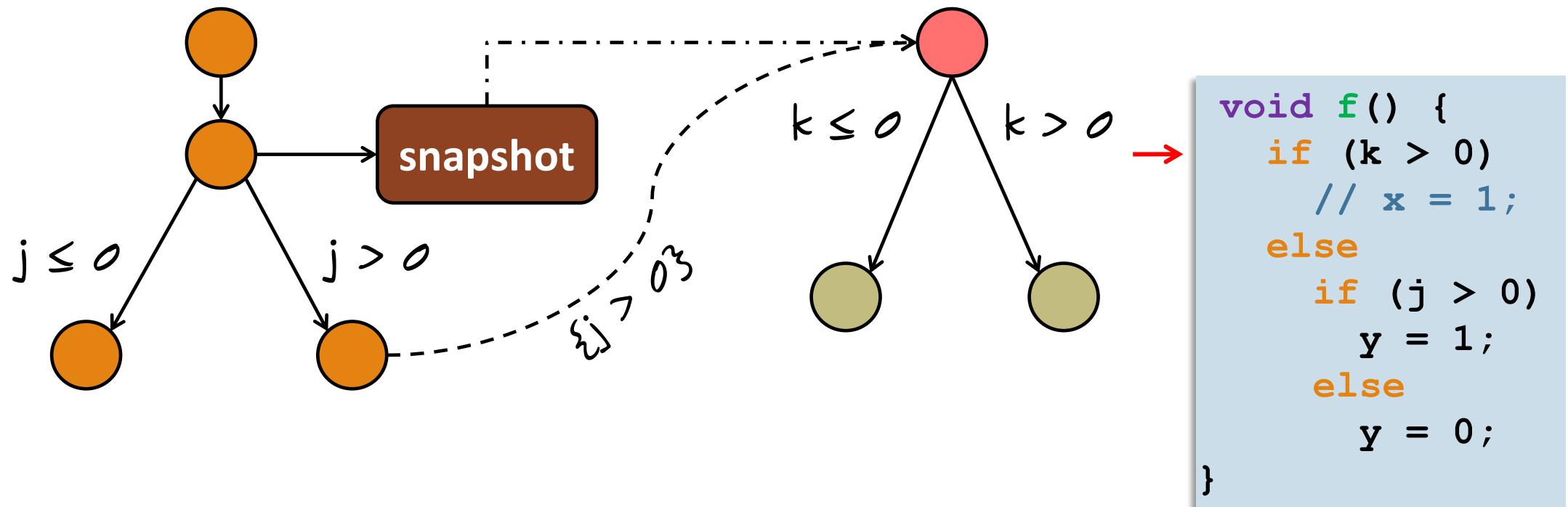


Chopped SE by Example

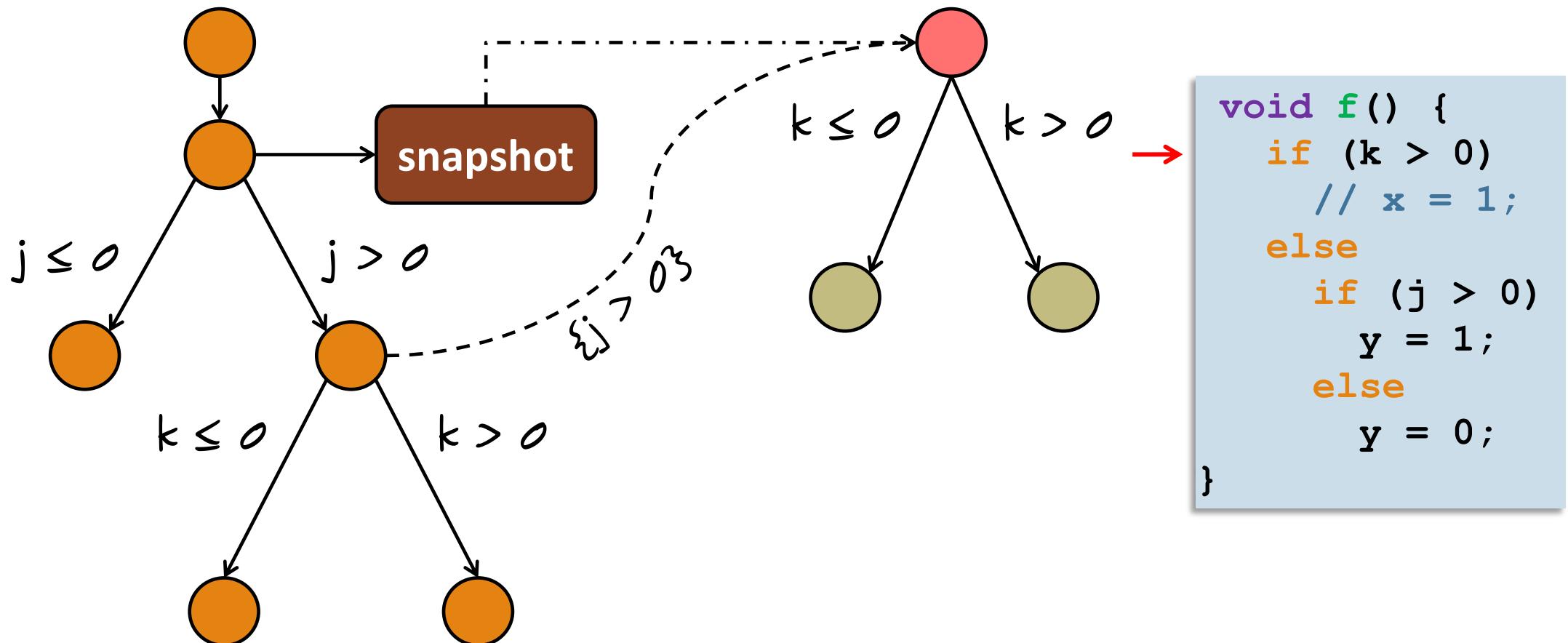


```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

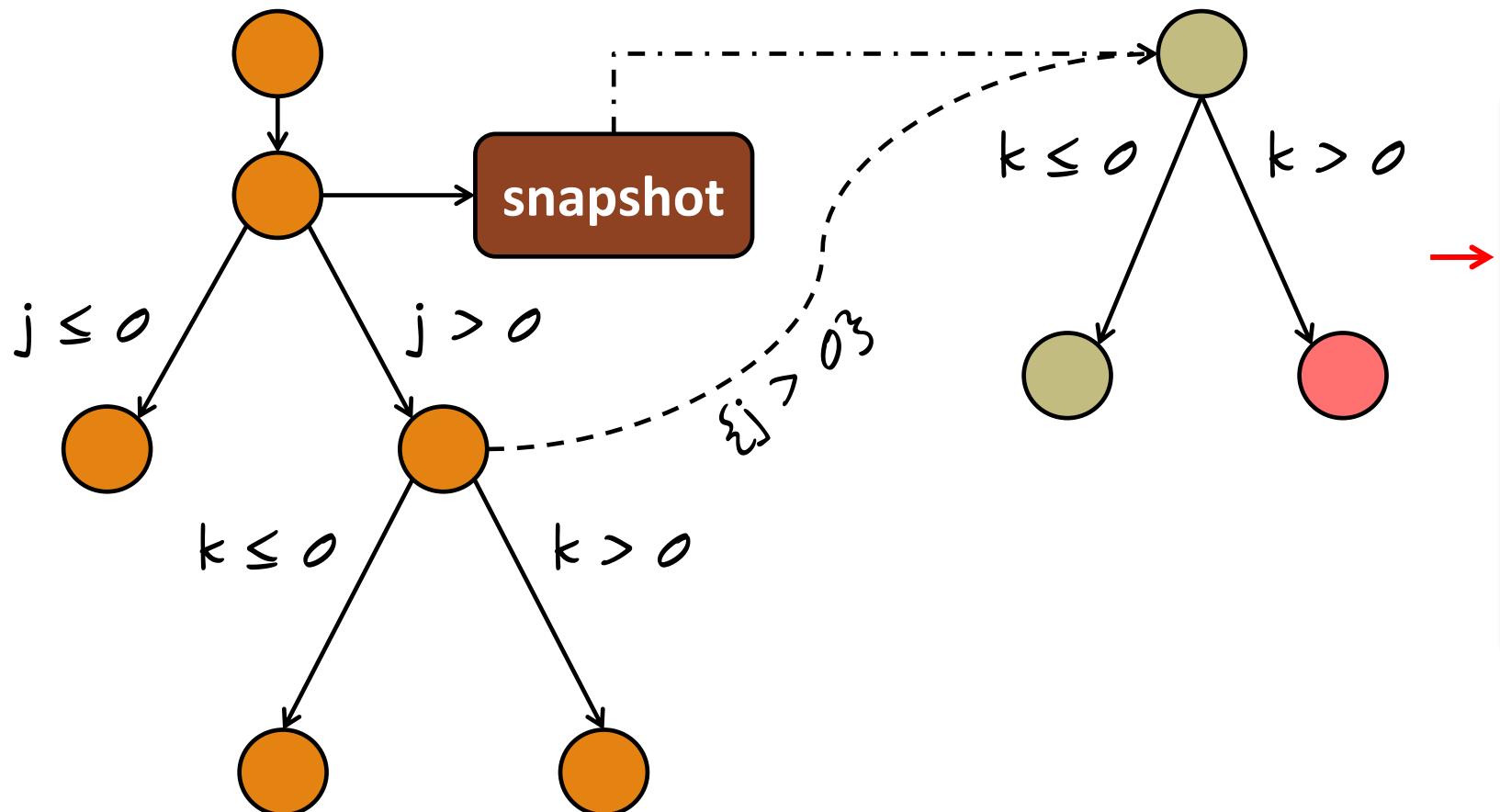
Chopped SE by Example



Chopped SE by Example

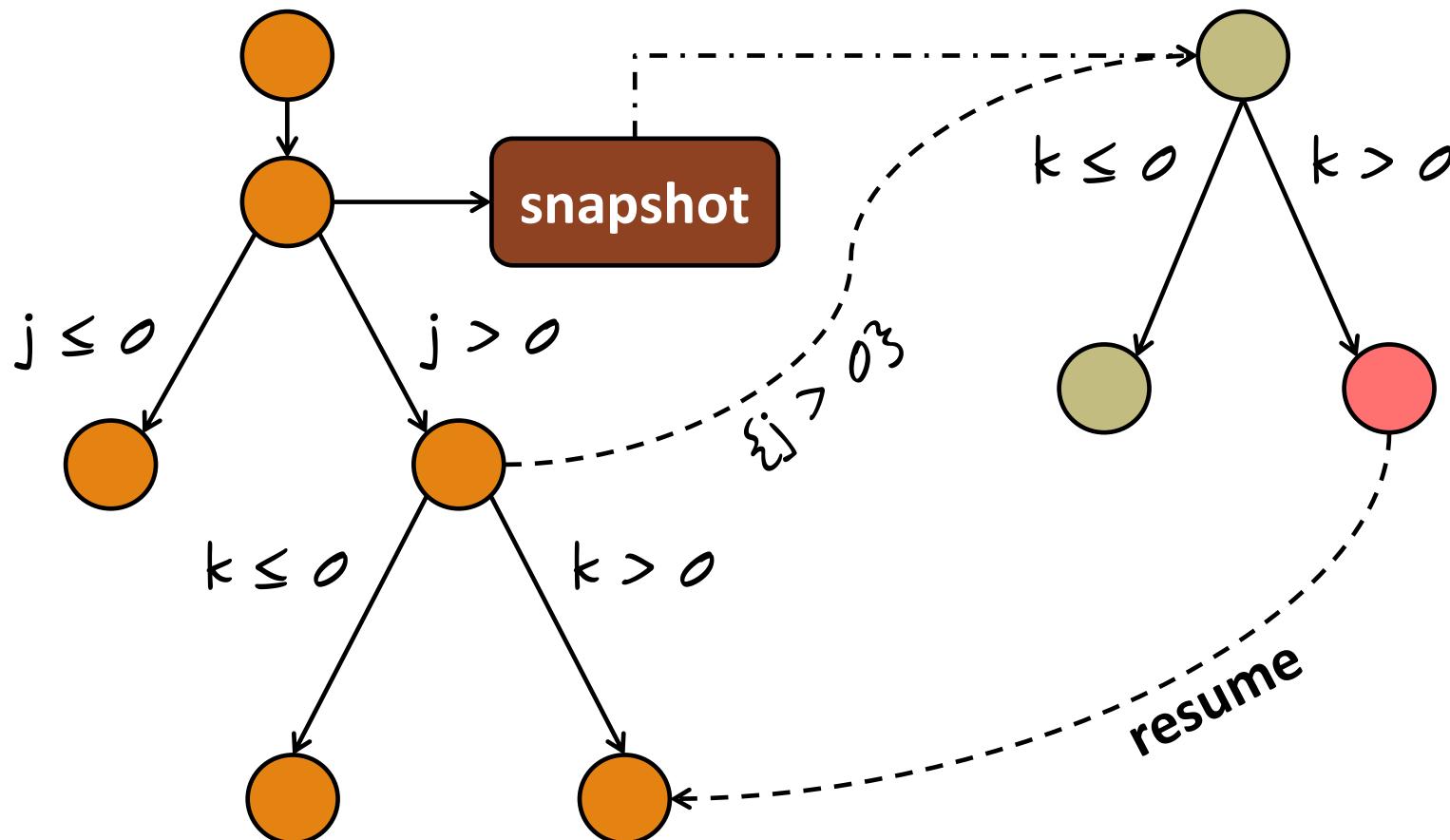


Chopped SE by Example



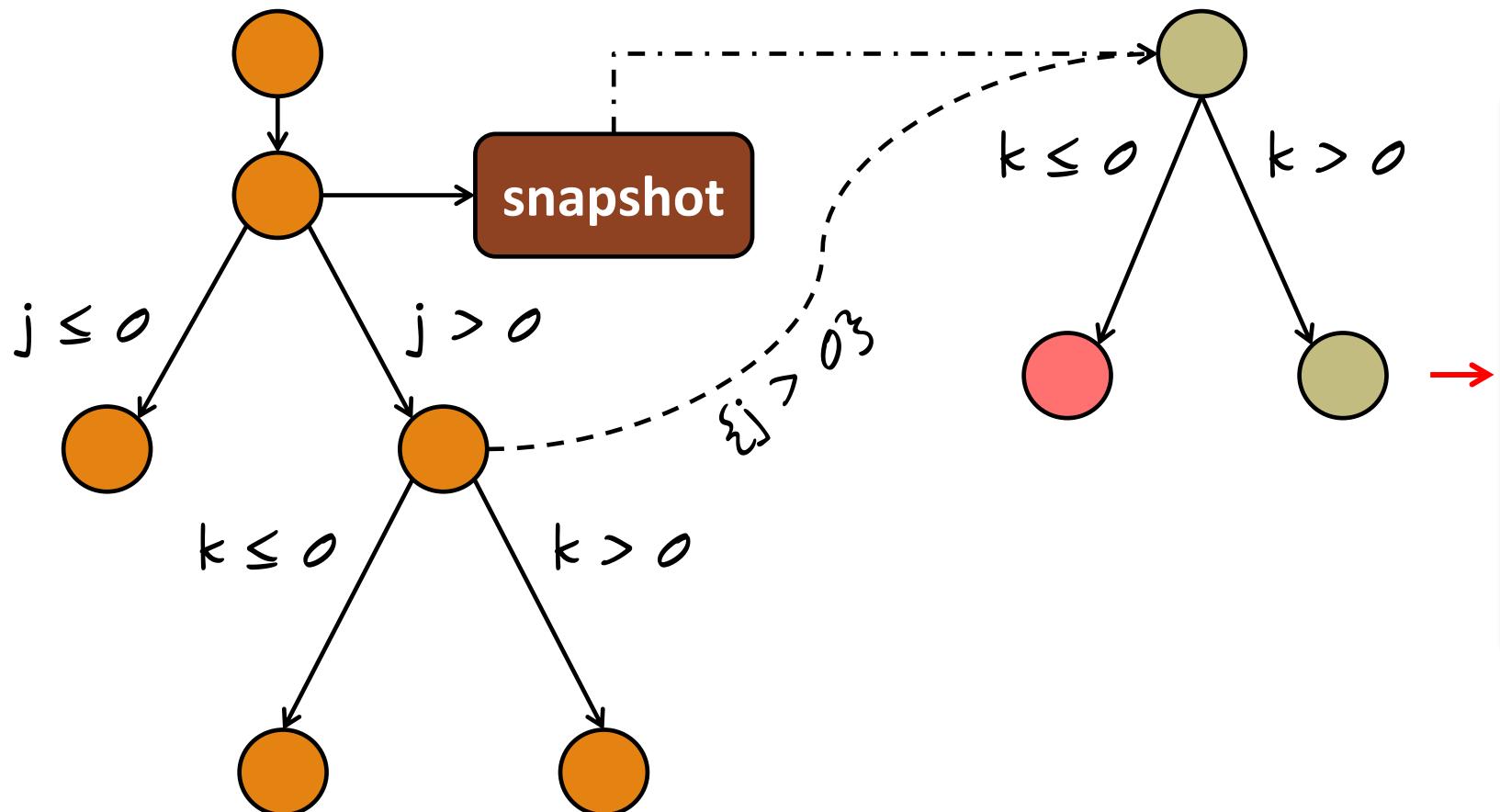
```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

Chopped SE by Example



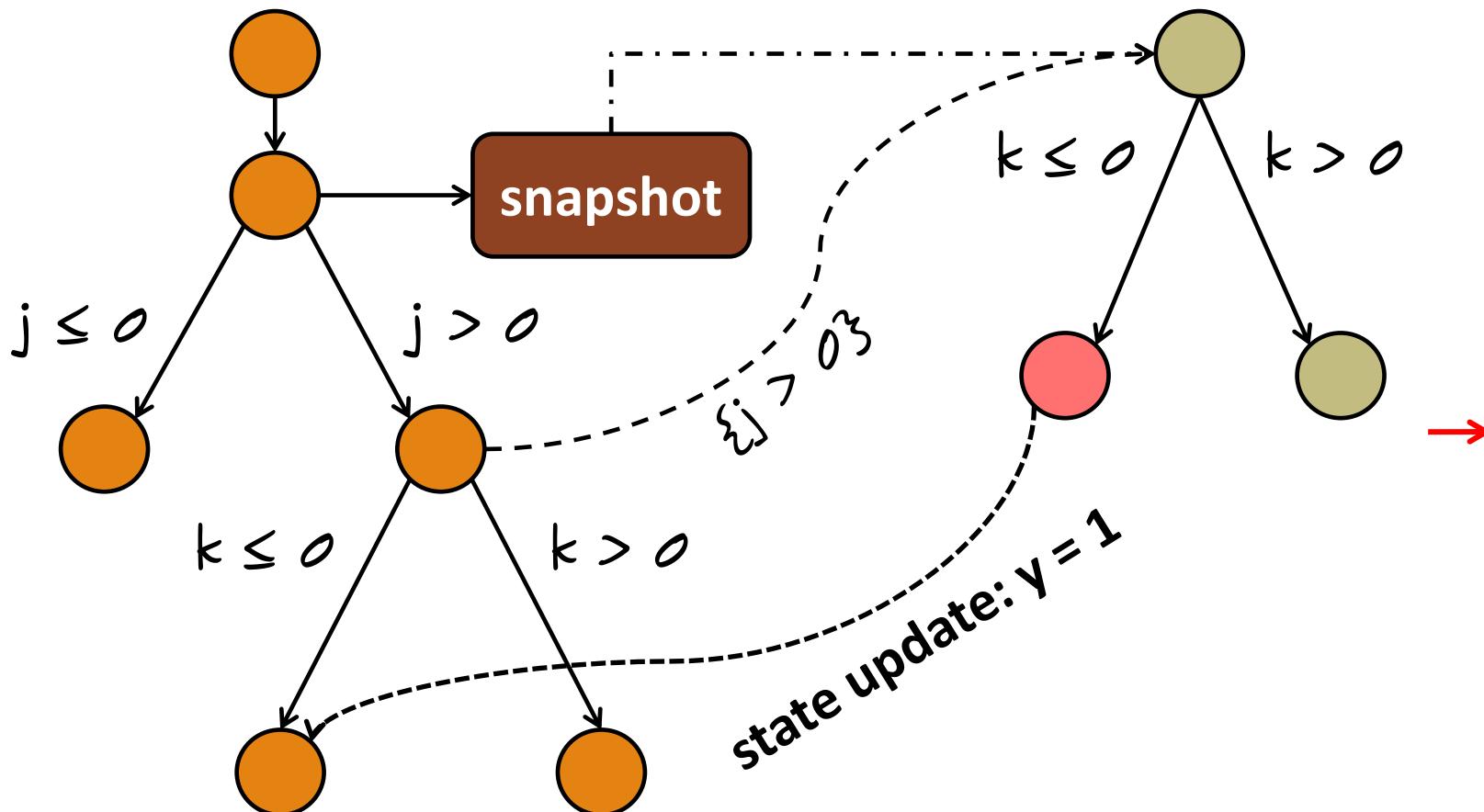
```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

Chopped SE by Example



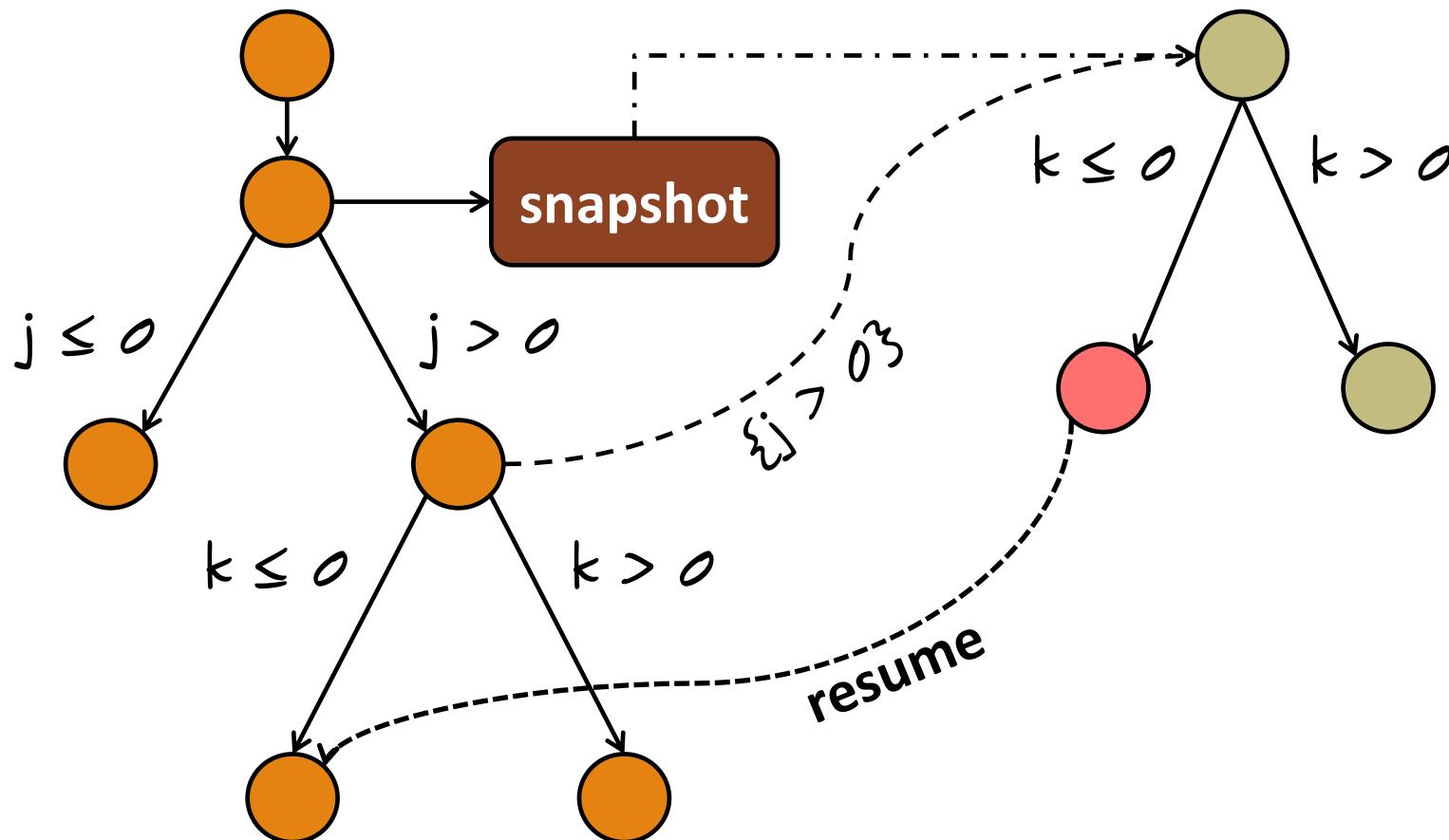
```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

Chopped SE by Example



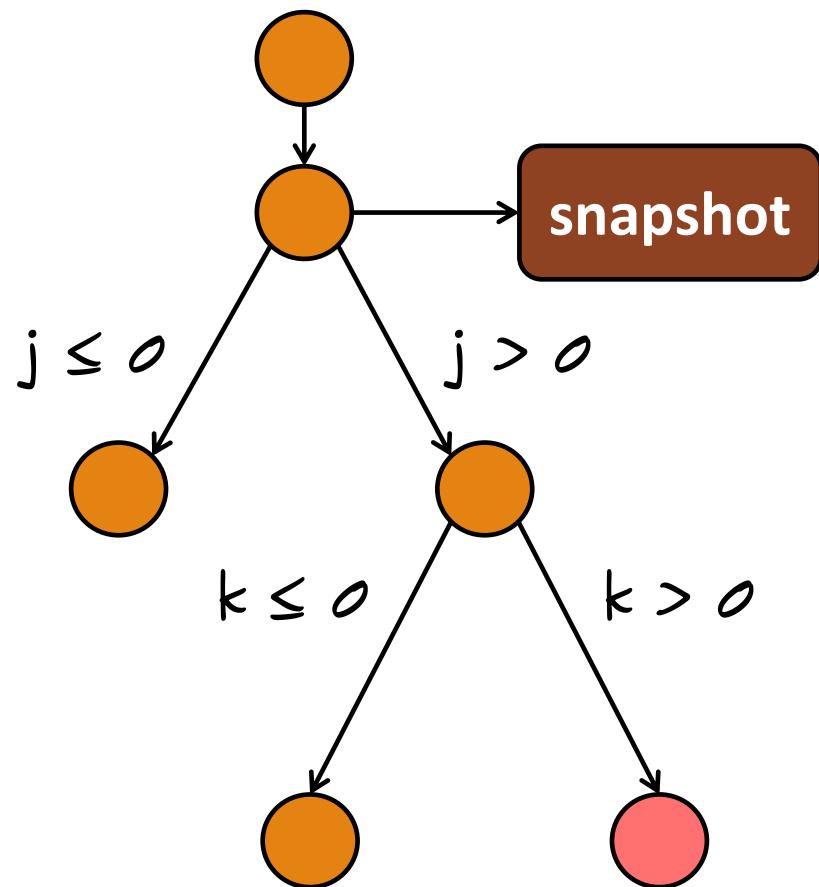
```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
    else  
        v = 0;
```

Chopped SE by Example



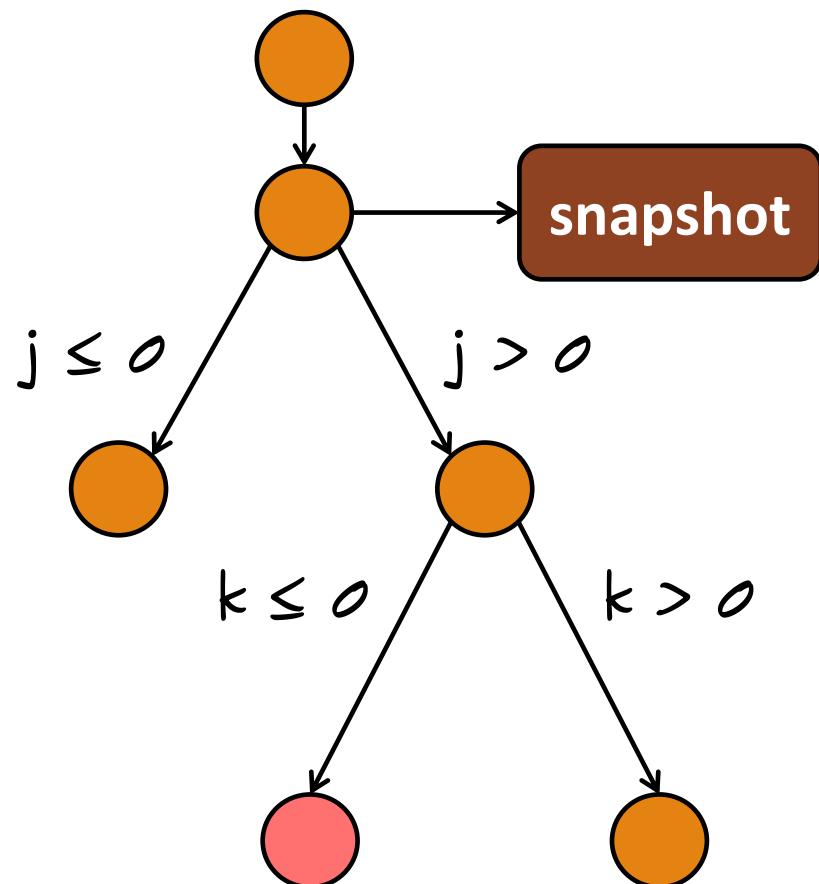
```
void f() {  
    if (k > 0)  
        // x = 1;  
    else  
        if (j > 0)  
            y = 1;  
        else  
            y = 0;  
}
```

Chopped SE by Example



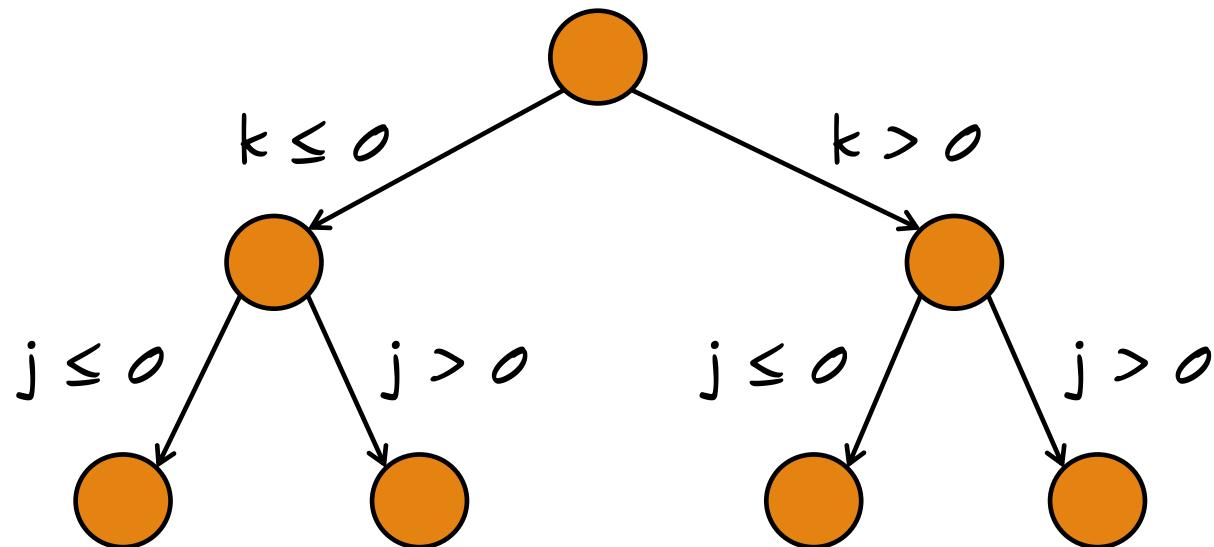
```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

Chopped SE by Example

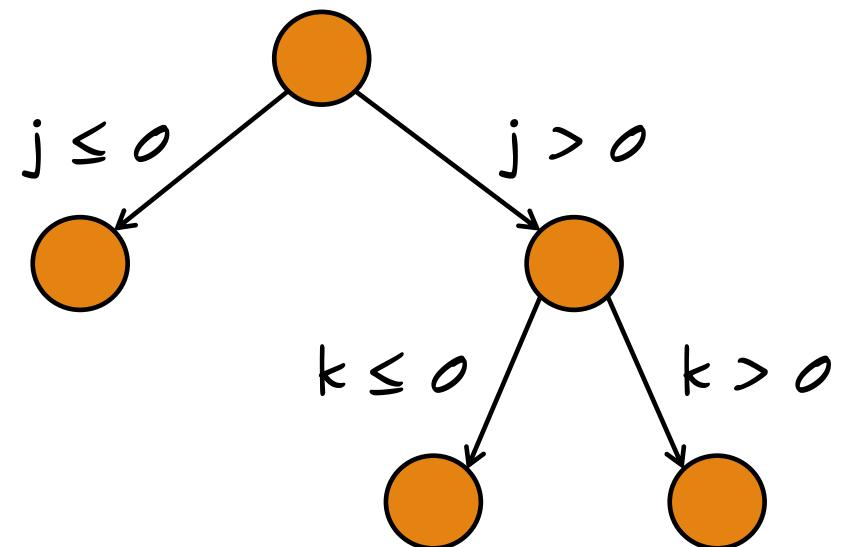


```
void main() {  
    f();  
    if (j > 0)  
        if (y)  
            bug();  
}
```

Process Trees



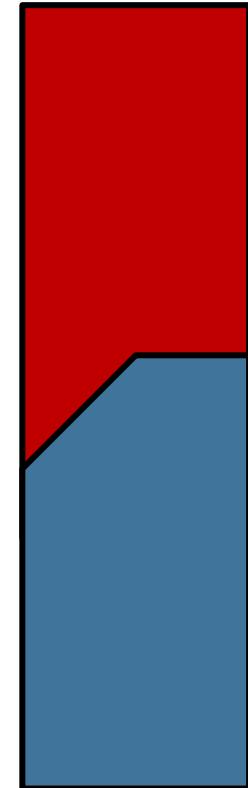
Standard SE



Chopped SE

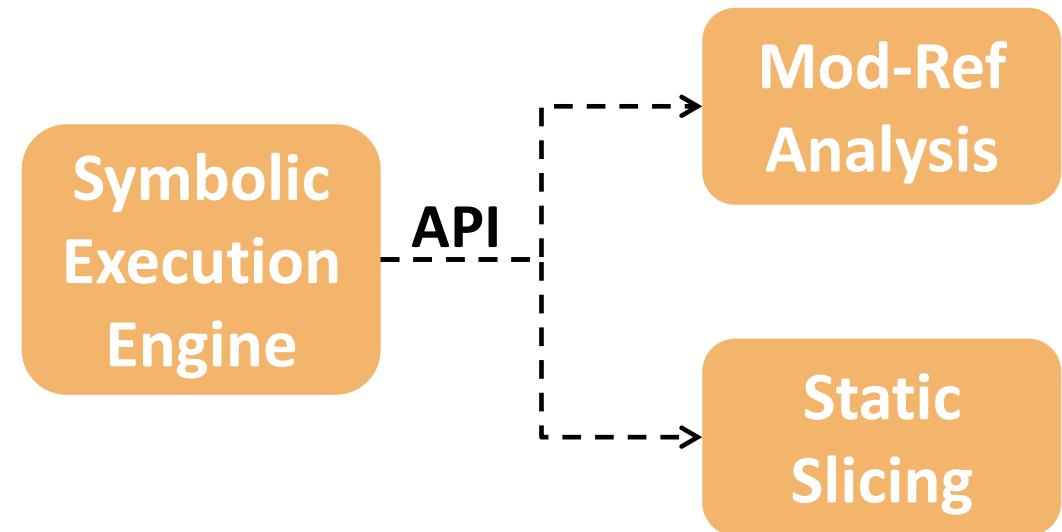
Search Heuristic

- The state space is divided into two parts:
 - **Normal** states
 - **Recovery** states
- **Normal states** are selected with **higher** probability



Implementation: CHOPPER

- Mod-Ref analysis
 - Based on the pointer analysis provided by the *SVF* project
- Static slicing
 - Based on the *DG* project
- Symbolic Execution Engine
 - Built on top of *KLEE*



Experiments

Scenarios:

- Bug reproduction
- Test case augmentation

Benchmarks:

- GNU libtasn1
- GNU BC
- LibYAML
- GNU oSIP

Bug Reproduction

Benchmark: GNU libtasn1

Methodology:

- Creating a test driver for the *libtasn1* library
- Manually identifying the functions to skip
- Time limit of 24 hours
- Memory limit of 4 GB
- Execution is terminated once the error is discovered

Bug Reproduction

CVE	Heuristic	Random	
		KLEE	CHOPPER
2012-1569		<i>OOM</i>	00:02
2014-3467 (1)		00:01	00:01
2014-3467 (2)		01:02	00:06
2014-3467 (3)		<i>Timeout</i>	00:10
2015-2806		01:07	00:02
2015-3622		<i>Timeout</i>	00:01

Bug Reproduction

CVE	Heuristic	Random		DFS		Coverage	
		KLEE	CHOPPER	KLEE	CHOPPER	KLEE	CHOPPER
2012-1569		OOM	00:02	OOM	00:03	OOM	00:01
2014-3467 (1)		00:01	00:01	00:17	00:01	00:01	00:01
2014-3467 (2)		01:02	00:06	Timeout	00:01	01:34	00:03
2014-3467 (3)		Timeout	00:10	Timeout	00:13	Timeout	00:10
2015-2806		01:07	00:02	02:46	00:12	OOM	00:01
2015-3622		Timeout	00:01	Timeout	00:18	20:25	00:01

Test Suite Augmentation

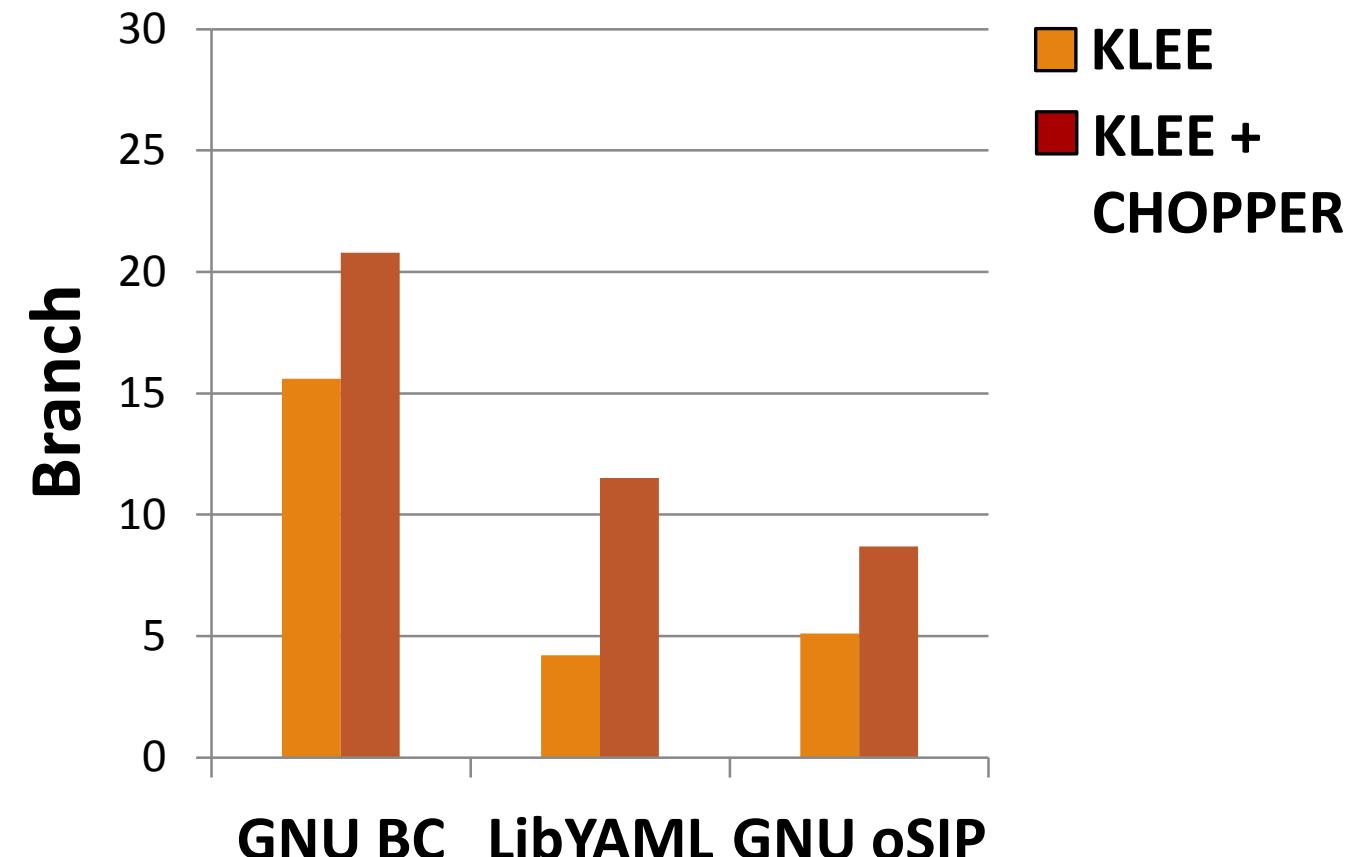
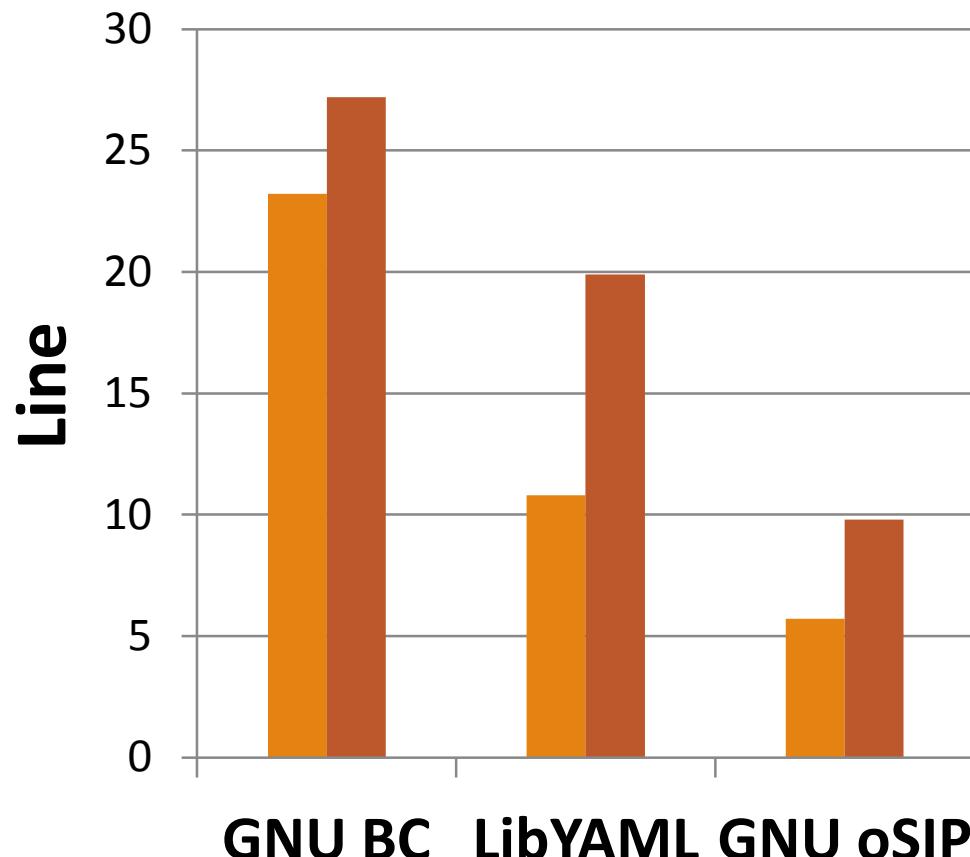
Benchmarks:

- GNU BC
- LibYAML
- GNU oSIP

Methodology:

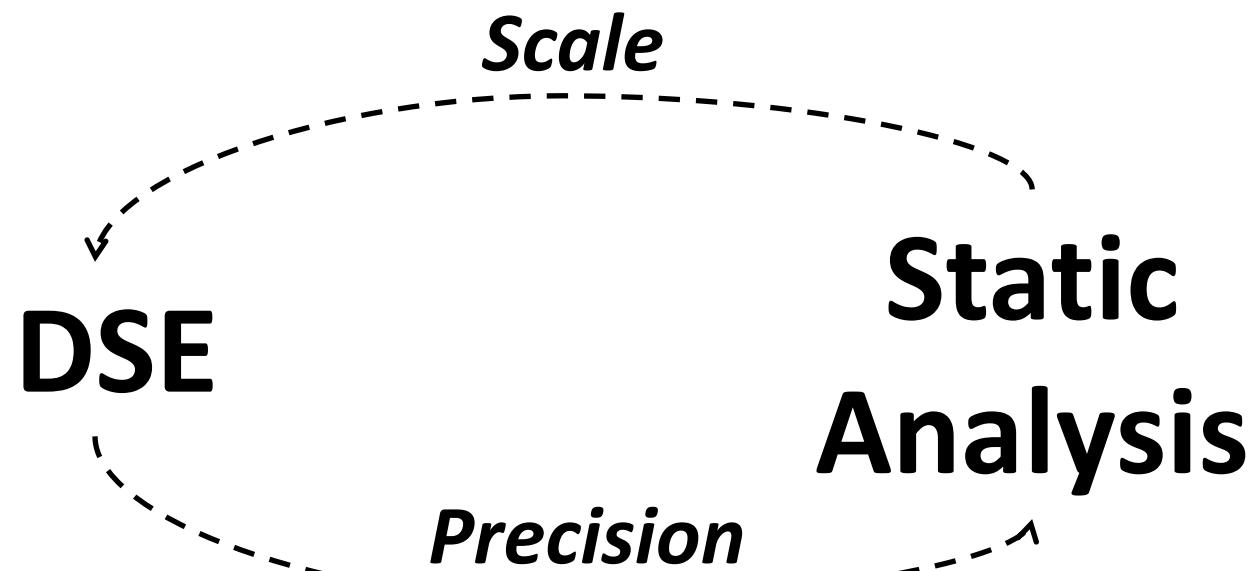
- Run KLEE with coverage based search heuristic for one hour
- Compute the structural coverage using *GCov*
- Identifying the functions to skip
- Run CHOPPER with coverage based search heuristic for one hour

Test Suite Augmentation



Conclusion

- Excluding parts of the CFG is useful in different scenarios
- DSE benefits from static analysis (and vice versa)



Thanks!