Ben Greenman presents:

ESP: Path-Sensitive Program Verification in Polynomial Time

Manuvir Das
Sorin Lerner
Mark Seigle

PLDI 2002

"Full-scale verification of large code bases is infeasible"

Benchmark

- **`gcc 2.5.3`**, from the SPEC'95 benchmark suite
- 140K LOC
- 2,149 functions
- 66 files
- 1,086 global & static variables
- 450-function strongly-connected-component

Safe I/O

- 646 calls to **`printf`** print to valid, open files

Challenges

- 15 file handles
  - 32,768 "initial" states
- "many" branch points

~~Challenges~~ Observations

- 15 file handles
- ~~32,768 "initial" states~~
- "many" branch points

Independent

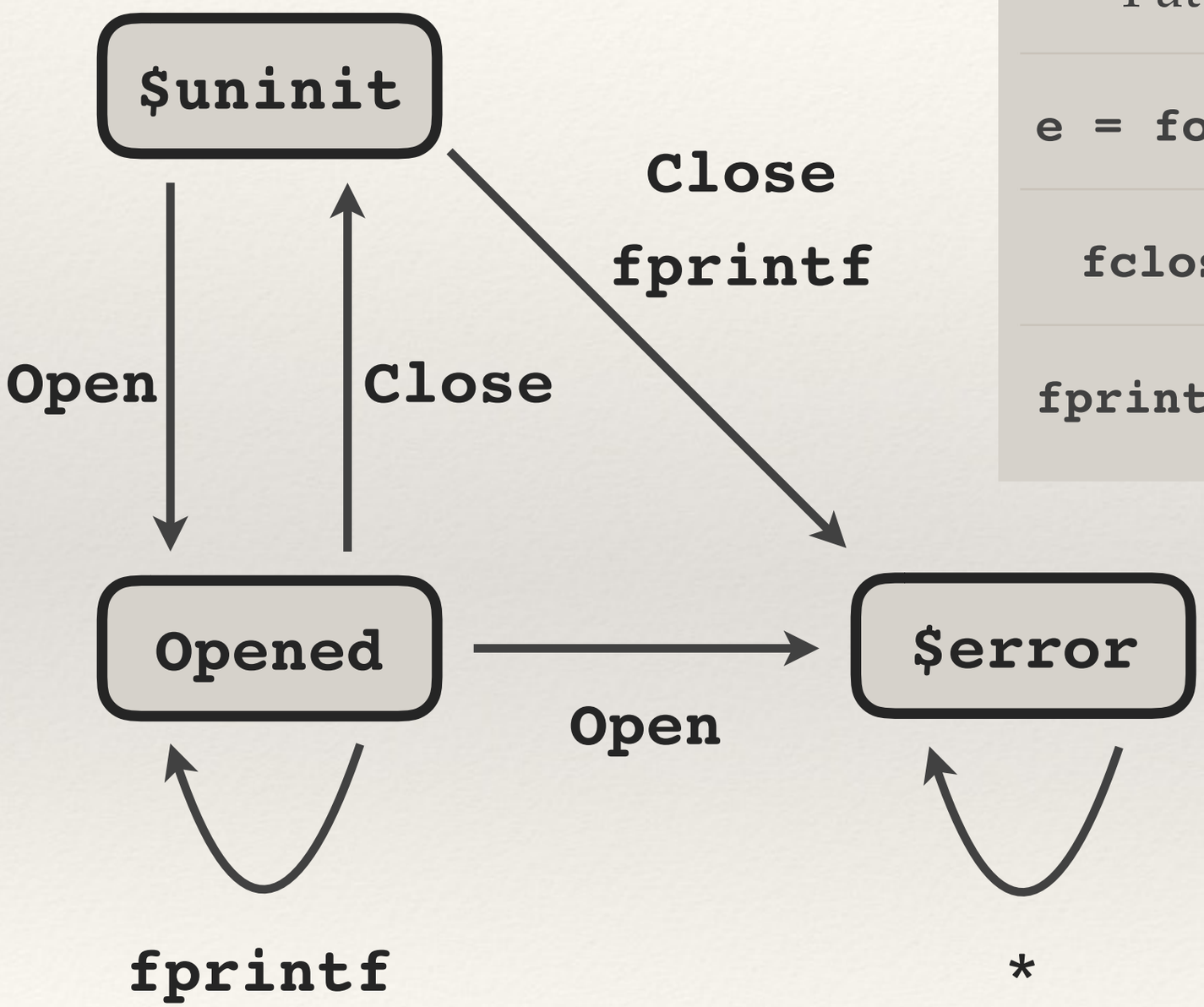
30 initial states

Mostly "boring"

Goal: Safe I/O

```
void main(){
    if (b1)
      f = fopen(fname, "w");
    if (b2)
      x = 0;
    else
      x = 1;
    if (b1)
      fclose(f);
}
```

Property FSM



Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

ESP

- CFG construction
- Value flow computation
- Abstract CFG construction
- Interface expression computation
- Property simulation

CFG construction

```
foo(&s1);
foo(&s2);
bar(&s3);

foo(struct s *p) {
   *p.a = 3;
   bar(p);
}
bar(struct s *q) {
   *q.b = 4
}
```

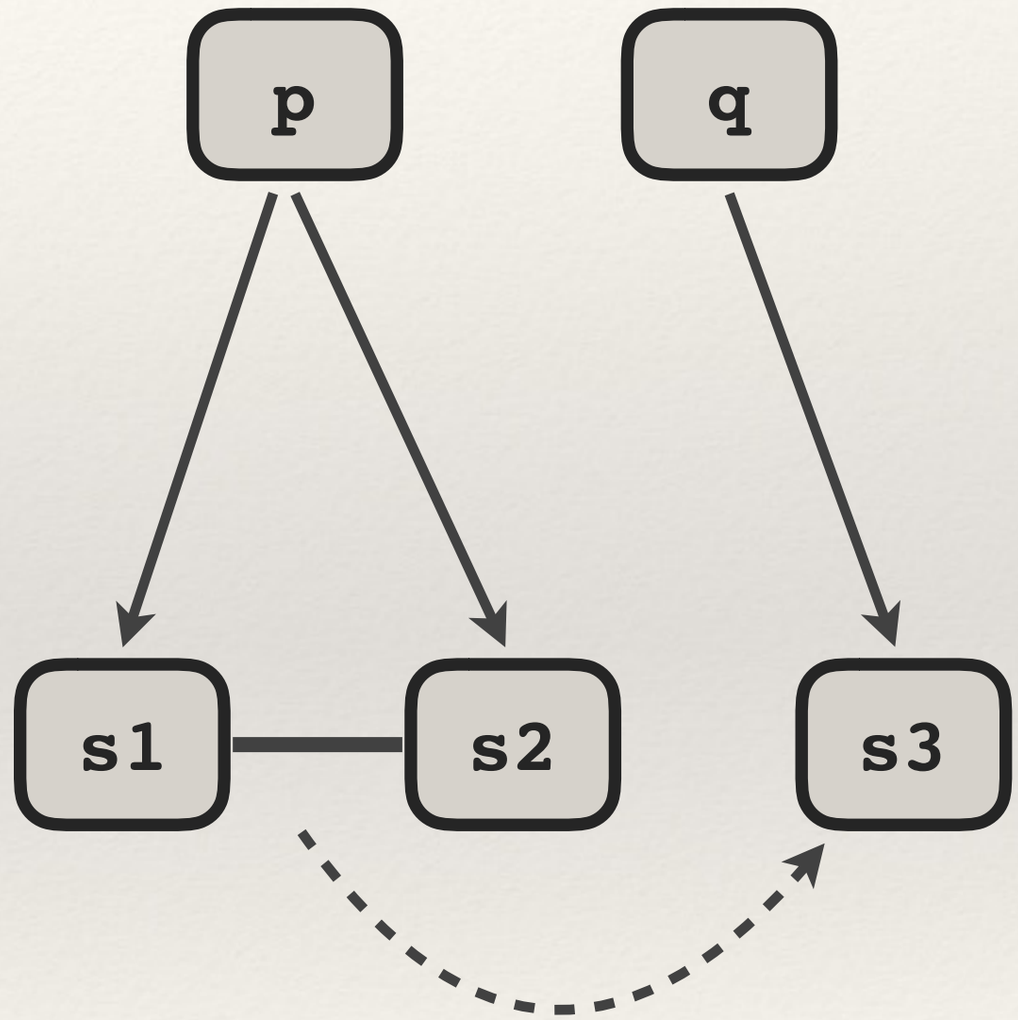
Das, PLDI'00

CFG construction

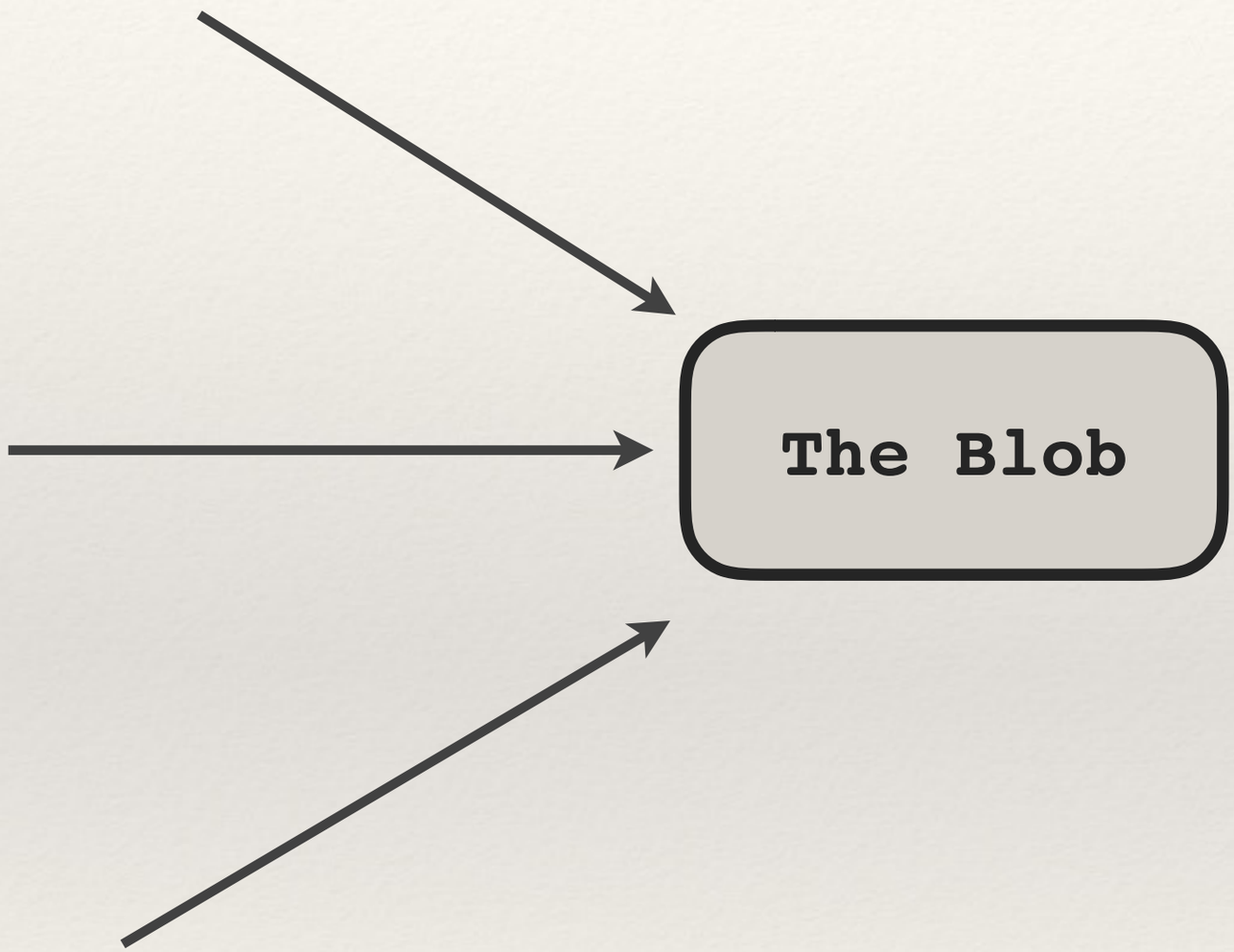
```
foo(&s1);
foo(&s2);
bar(&s3);

foo(struct s *p) {
   *p.a = 3;
   bar(p);
}
bar(struct s *q) {
   *q.b = 4
}
```

CFG construction

```
foo(&s1);
foo(&s2);
bar(&s3);

foo(struct s *p) {
   *p.a = 3;
   bar(p);
}
bar(struct s *q) {
   *q.b = 4
}
```



CFG construction



Value-Flow computation

```
id(r) {
   return r;
}

p = id(&x); // A
q = id(&y); // B
*p = 3
```

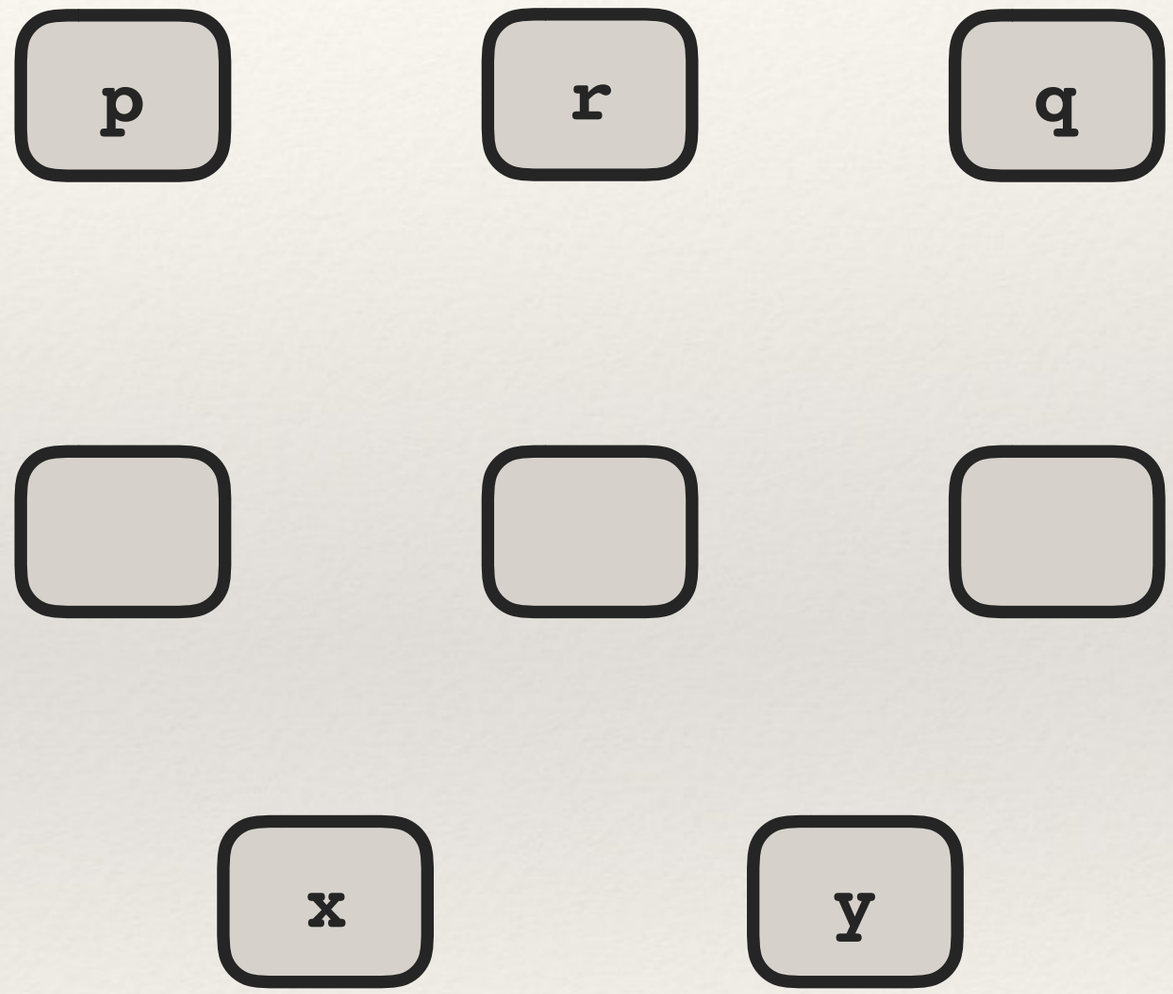
Das, Liblit, Fahndrich, Rehof; PLDI'01

Value-Flow computation

```
id(r) {
   return r;
}

p = id(&x); // A
q = id(&y); // B
*p = 3
```

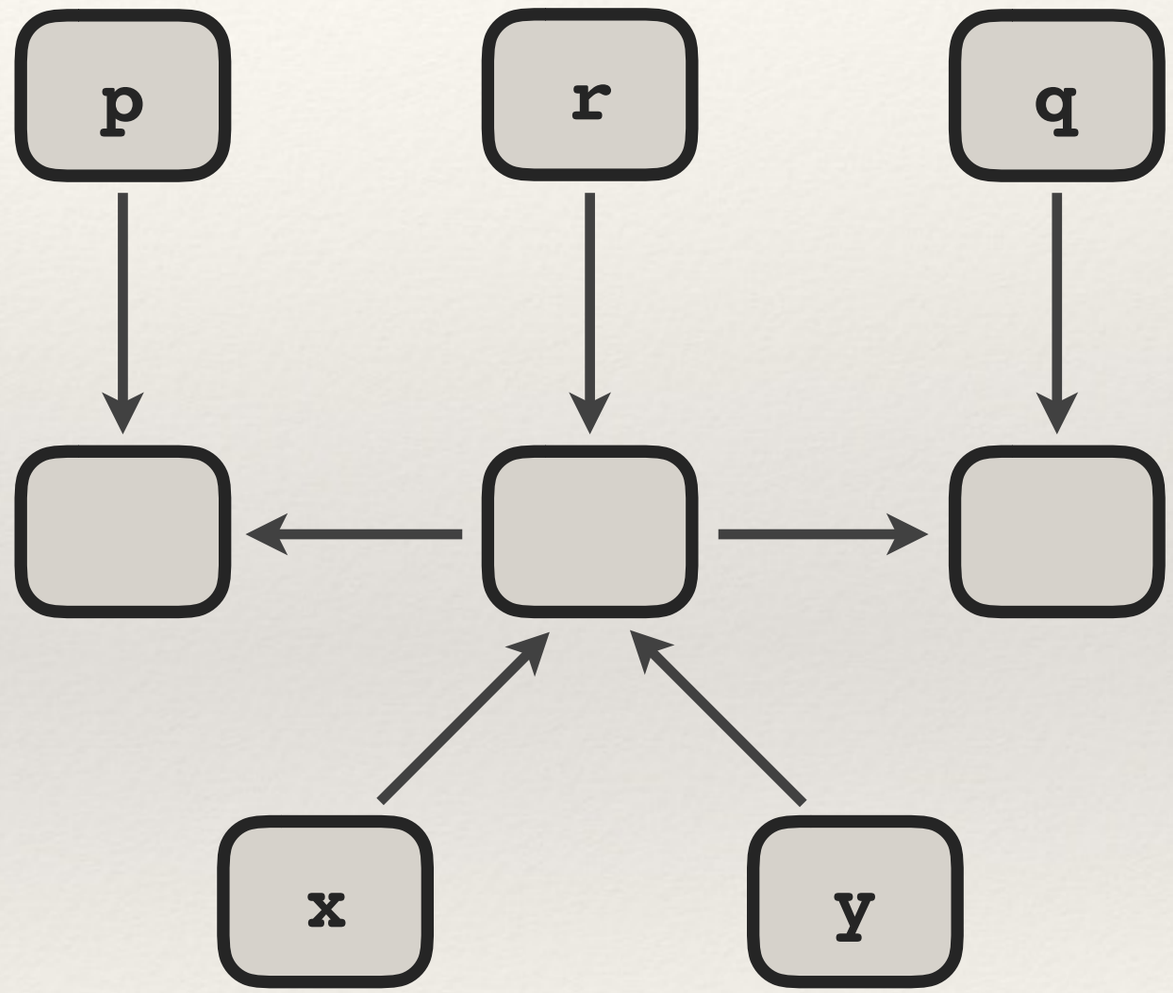
Value-Flow computation

```
id(r) {
   return r;
}

p = id(&x); // A
q = id(&y); // B
*p = 3
```



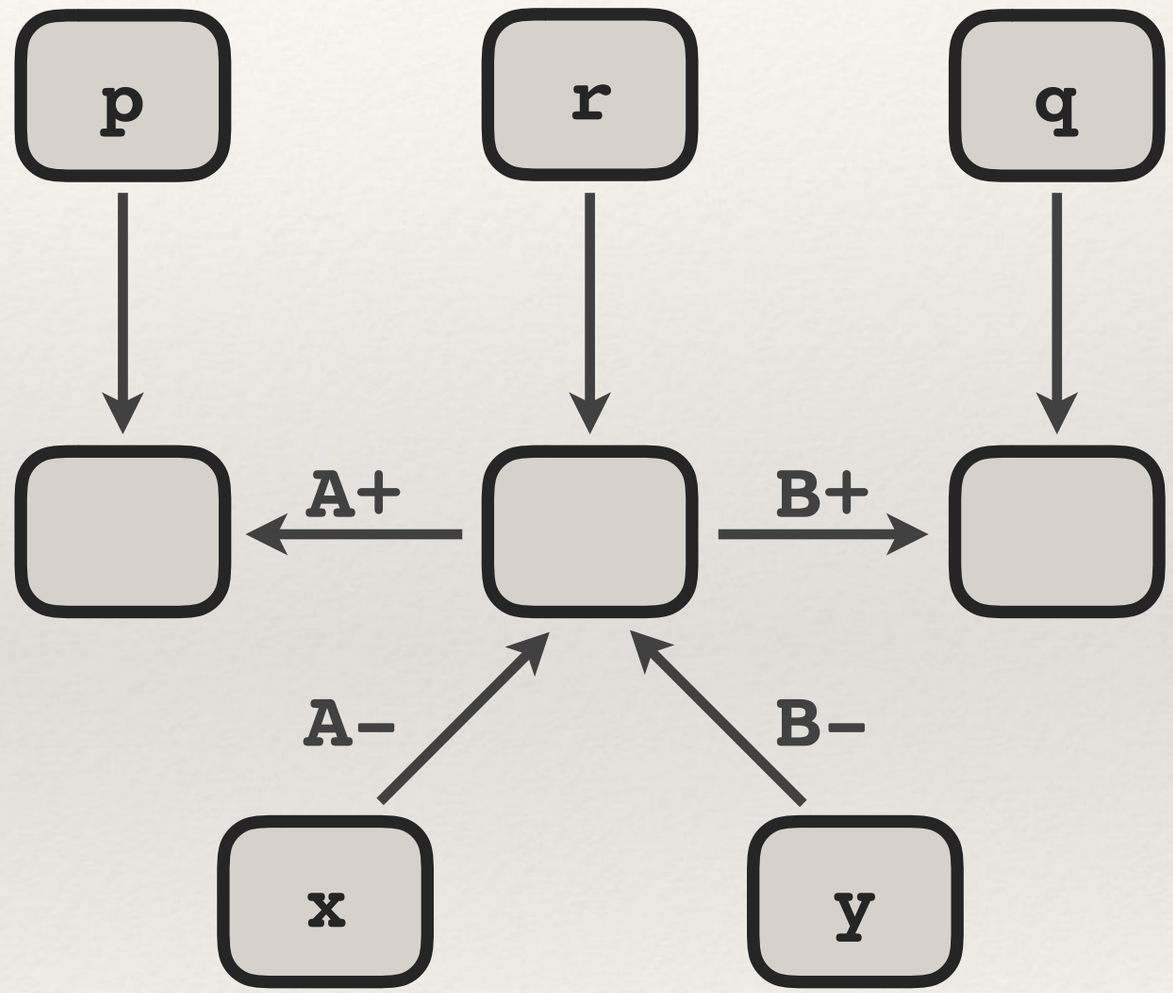
Value-Flow computation

```
id(r) {
   return r;
}

p = id(&x); // A
q = id(&y); // B
*p = 3
```



Value-Flow computation

```
id(r) {
   return r;
}

p = id(&x); // A
q = id(&y); // B
*p = 3
```



Abstract CFG construction

```
FILE *f1, *f2;
int p1, p2;

B: doStuff() {
      if (p1)
        x: rtl(f1);
      if (p2)
        y: rtl(f2);
      doStuff();
   }
C: rtl(FILE *f) {
      fprintf(f,?);
}
```

Abstract CFG construction

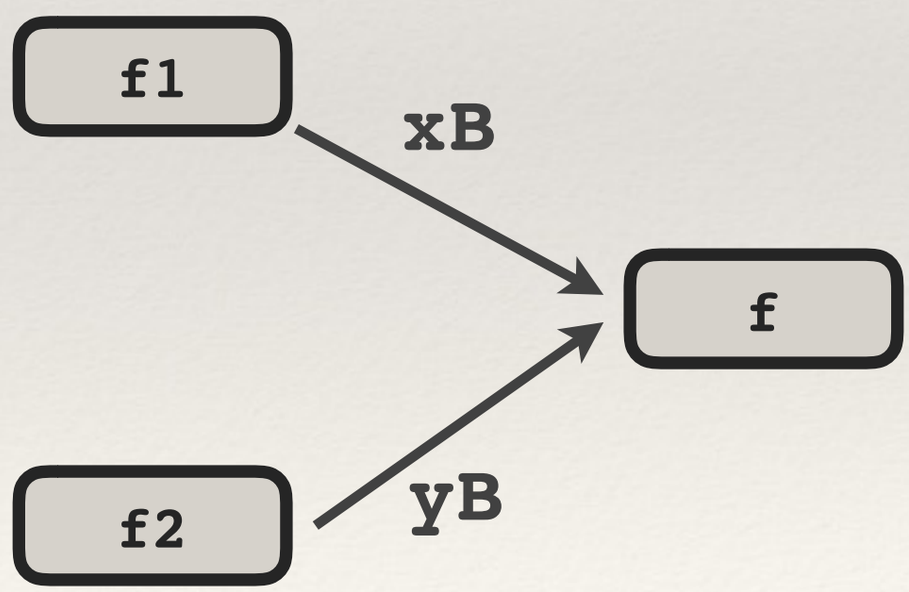
```
FILE *f1, *f2;
int p1, p2;

B: doStuff() {
     if (p1)
       x: rtl(f1);
     if (p2)
       y: rtl(f2);
     doStuff();
   }
C: rtl(FILE *f) {
     fprintf(f,?);
}
```

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

Abstract CFG construction

```
FILE *f1, *f2;
int p1, p2;

B: doStuff() {
     if (p1)
       x: rtl(f1);
     if (p2)
       y: rtl(f2);
     doStuff();
   }
C: rtl(FILE *f) {
     fprintf(f,?);
}
```

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`



Interface Expression computation

- Interface = **`inNodes`** + **`outNodes`**
  - **`inNodes`** = `globals U params U *(globals U params)`
  - **`outNodes`** = `globals U ret U *(globals U ret) U params`

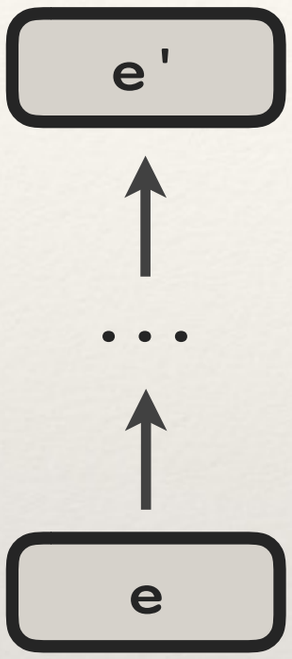
- **`Mod_Set(f)`** = all variables **`f`** may modify
- **`Alias_Set(x)`** = all exprs. that may get the same value

Property Simulation (intra)

`e'` in a Pattern

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

Property Simulation (intra)

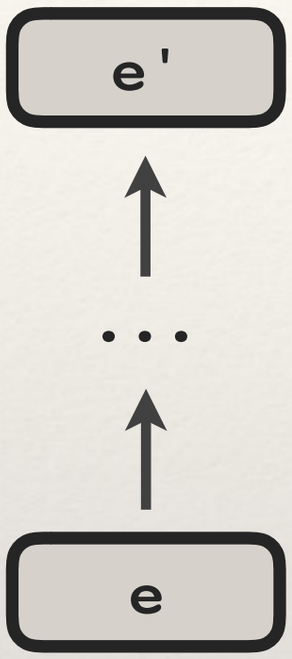


in a Pattern

in an **`Alias_Set`**

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

Property Simulation (intra)



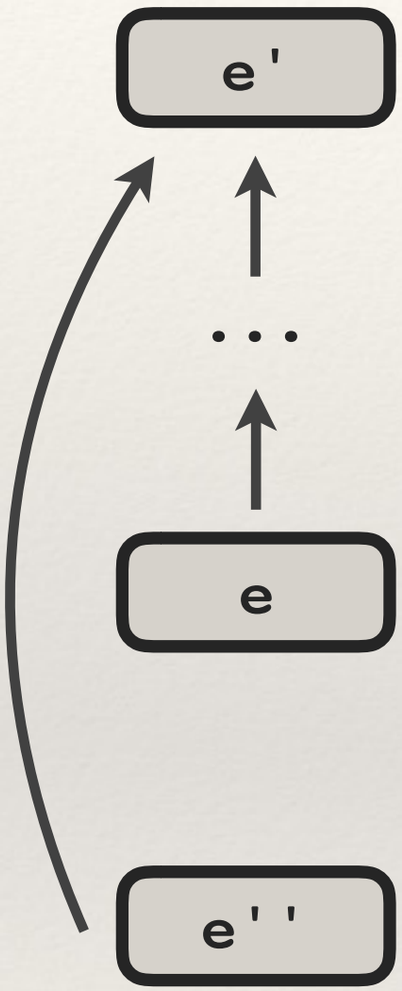
in a Pattern

in an **`Alias_Set`**

then update the *state* of the **`Alias_Set`**

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

Property Simulation (intra)



in a Pattern

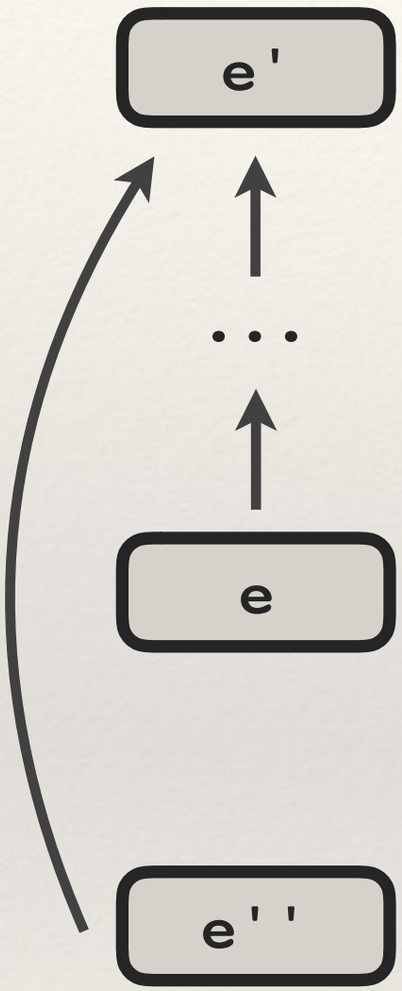
in an **`Alias_Set`**

then update the *state* of the **`Alias_Set`**

also reaches **`e'`**

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

Property Simulation (intra)



in a Pattern

Pattern	Transition	New?
`e = fopen(_)`	`Open`	`Y`
`fclose(e)`	`Close`	`N`
`fprintf(e,_)`	`Print`	`N`

in an **`Alias_Set`**

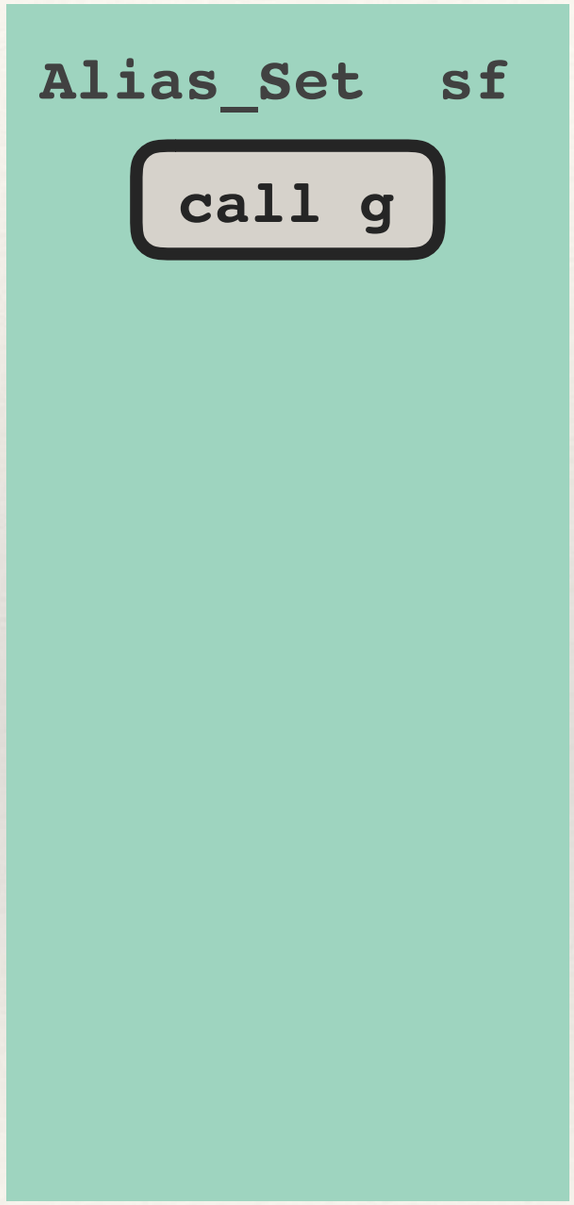
then update the *state* of the **`Alias_Set`**

also reaches **`e'`**

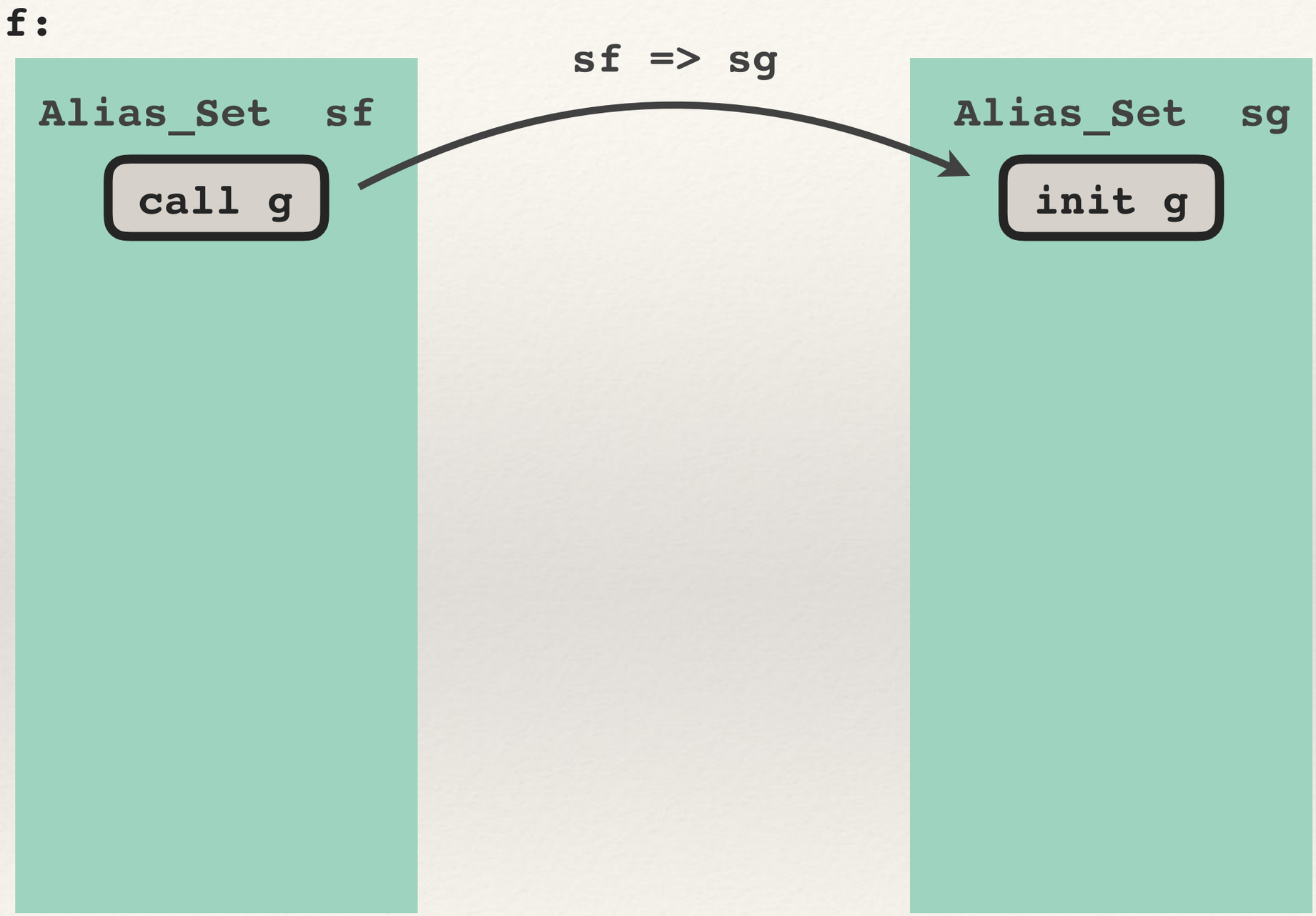
then add **`id`** transition for the **`Alias_Set`**

Property Simulation (inter)

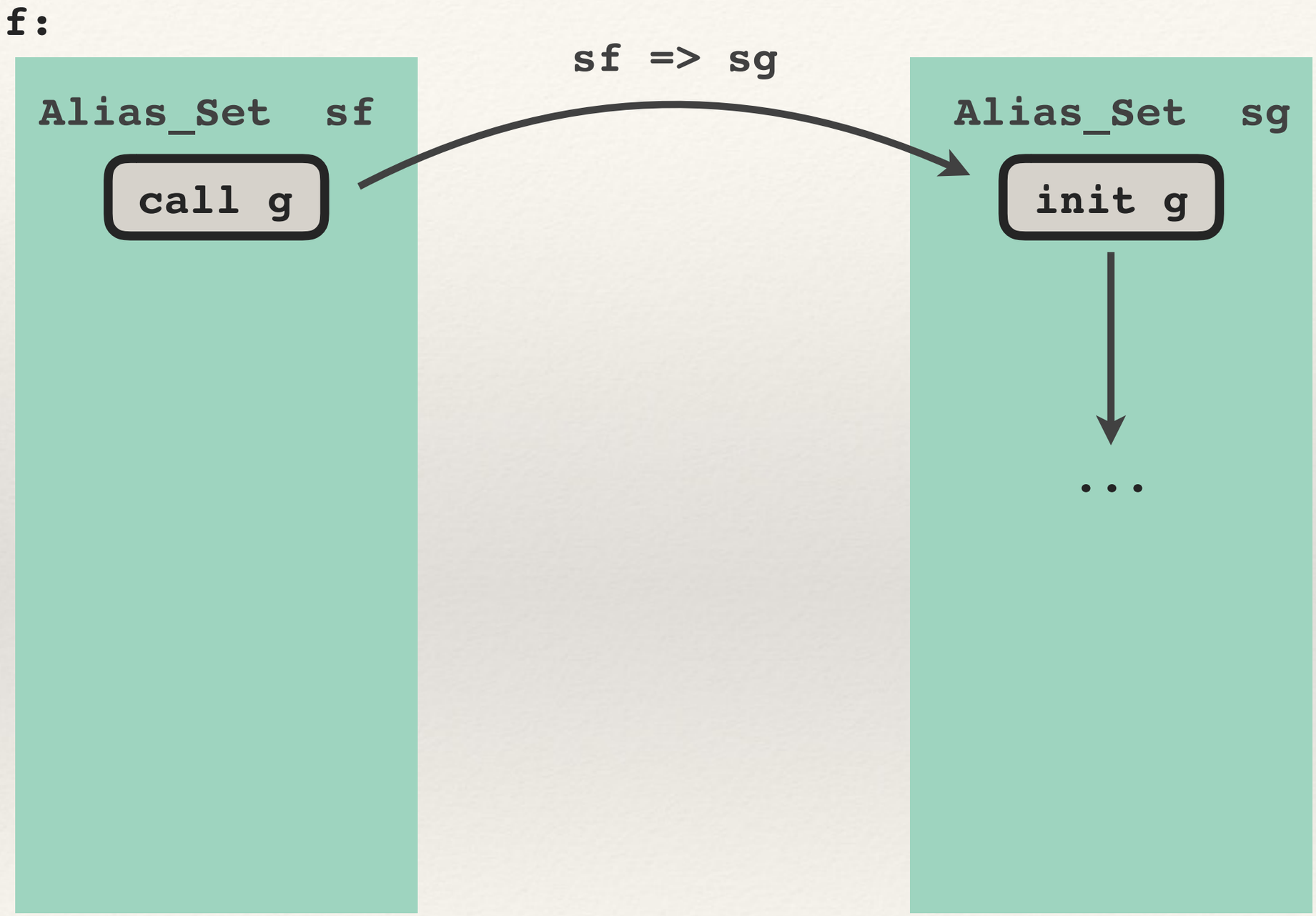
f:



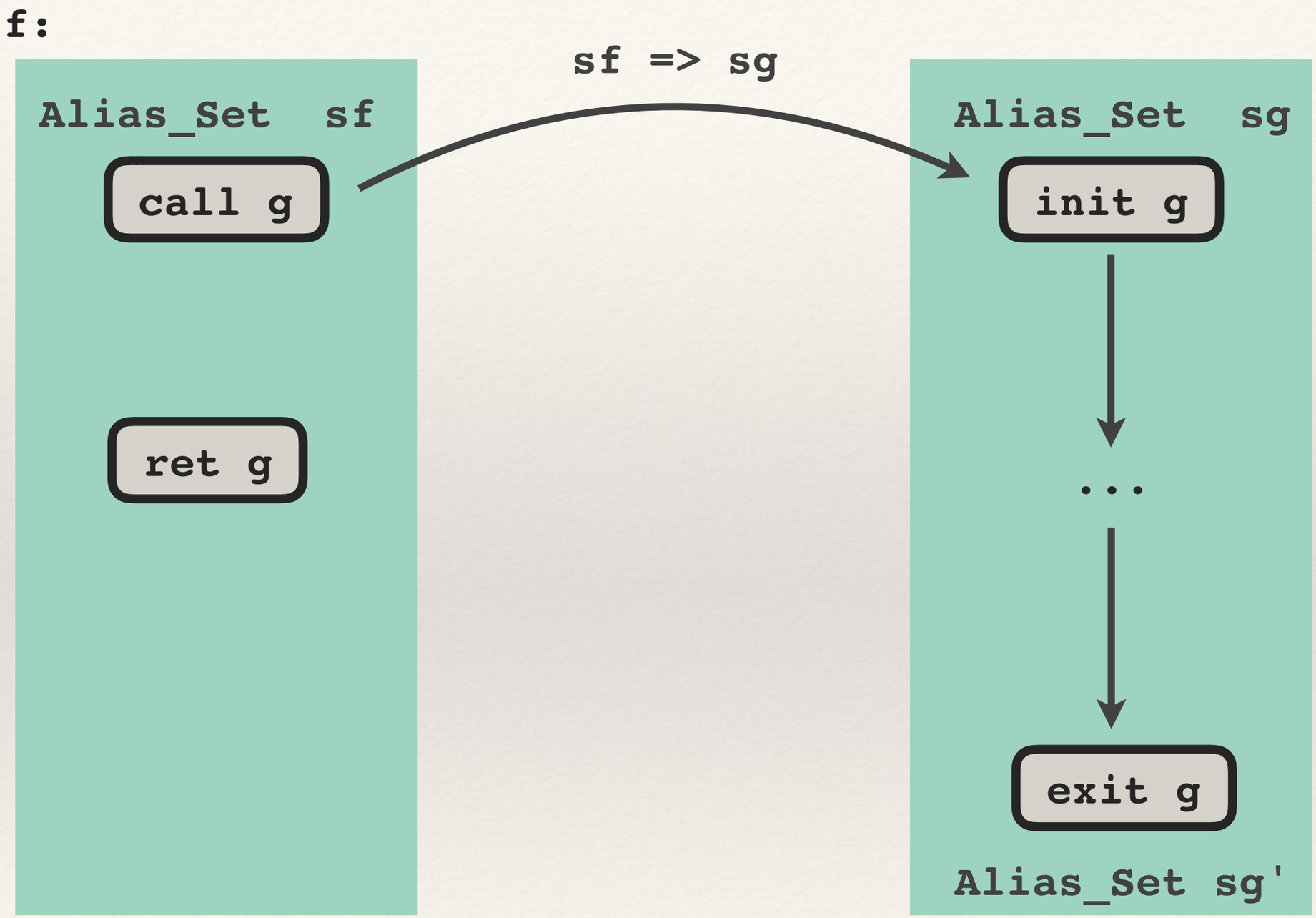
Property Simulation (inter)



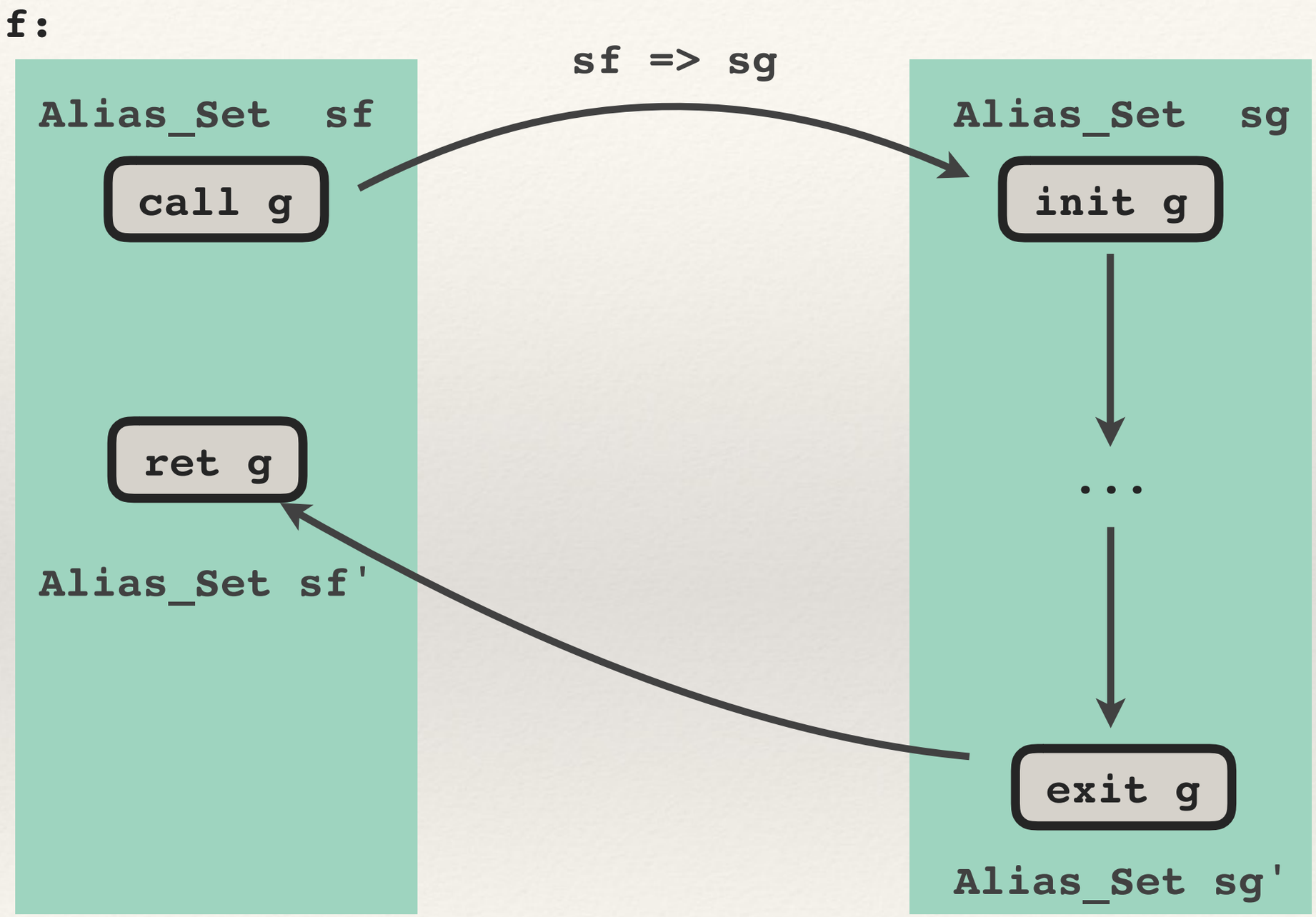
Property Simulation (inter)



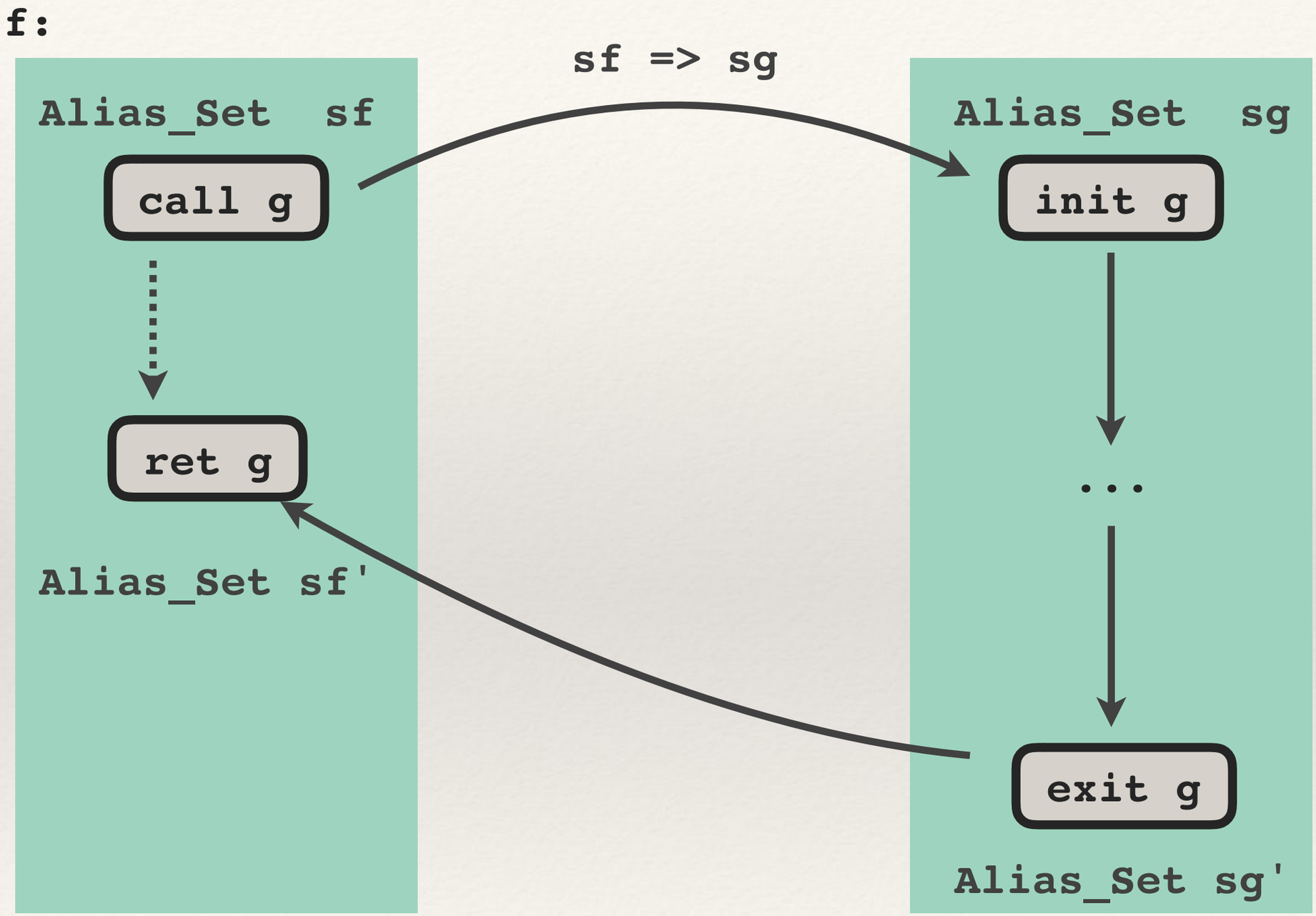
Property Simulation (inter)



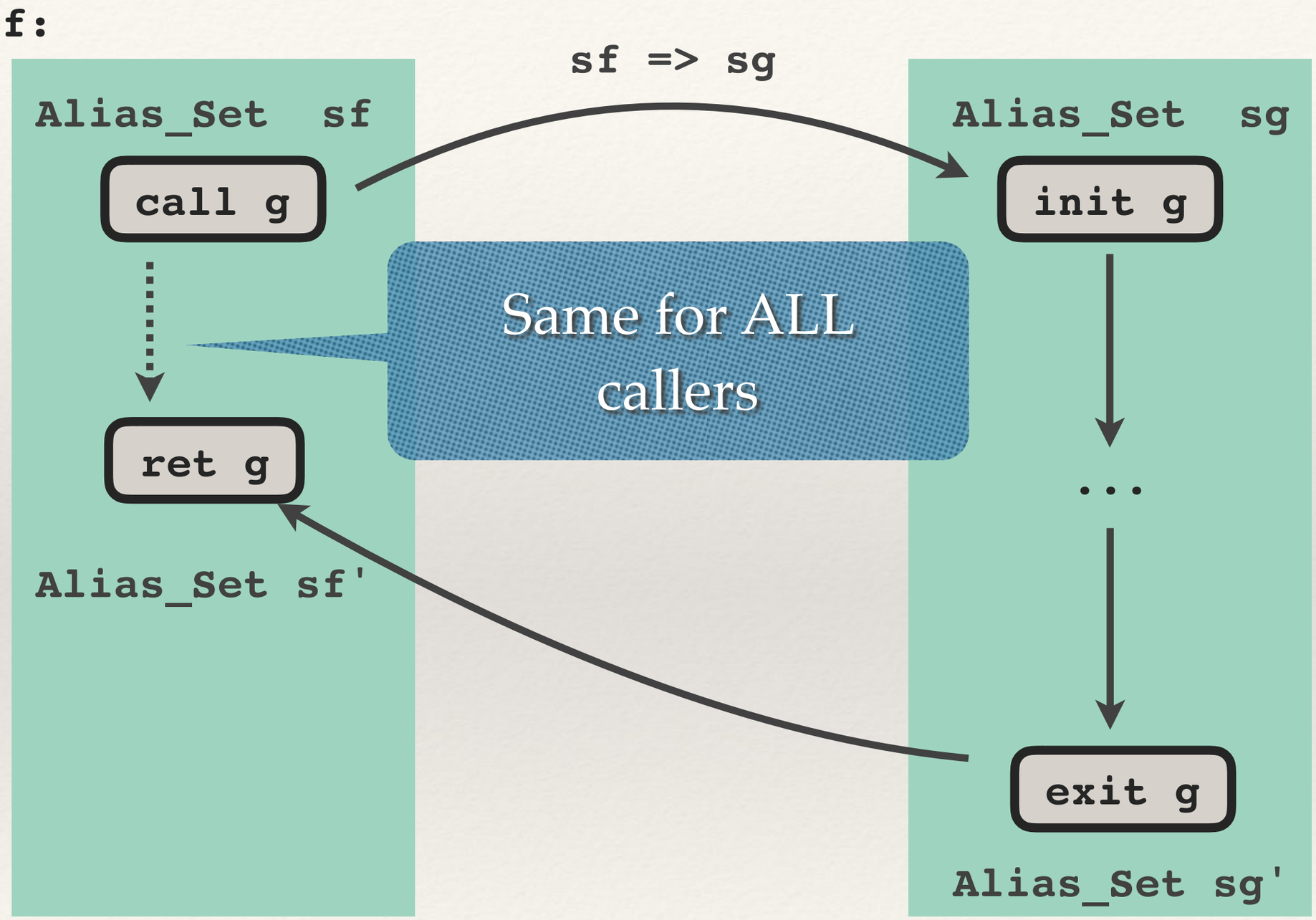
Property Simulation (inter)



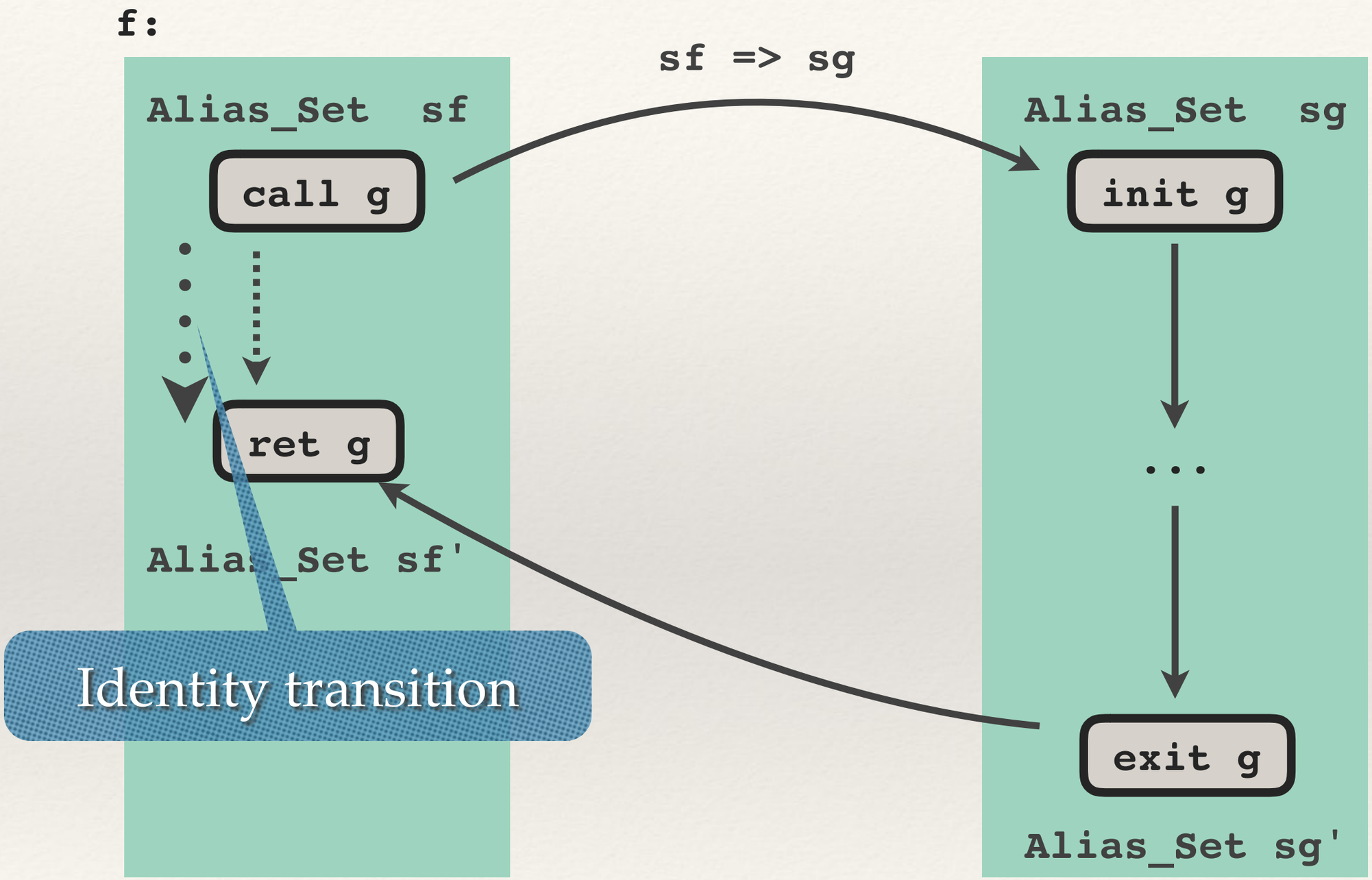
Property Simulation (inter)



Property Simulation (inter)



Property Simulation (inter)



Example

```
FILE *f1, *f2;
int p1, p2;

A: compile() {
     if (p1)
       f1 = fopen();
     if (p2)
       f2 = fopen();
     doStuff();
   }
```

```
B: doStuff() {
     if (p1)
       x: rtl(f1);
     if (p2)
       y: rtl(f2);
     doStuff();
   }

C: rtl(FILE *f) {
     fprintf(f,?);
}
```

Example

```
FILE *f1, *f2;
int p1, p2;

A: compile() {
      if (p1)
         f1 = fopen();
      if (p2)
         f2 = fopen();
      doStuff();
   }
```

```
B: doStuff() {
      if (p1)
         x: rtl(f1);
      if (p2)
         y: rtl(f2);
      doStuff();
   }

C: rtl(FILE *f) {
      fprintf(f,?);
}
```

Example

```
FILE *f1, *f2;
int p1, p2;

A: compile() {
     if (p1)
       f1 = fopen();
     if (p2)
       f2 = fopen();
     doStuff();
   }
```

```
B: doStuff() {
     if (p1)
       x: rtl(f1);
     if (p2)
       y: rtl(f2);
     doStuff();
   }

C: rtl(FILE *f) {
     fprintf(f,?);
}
```

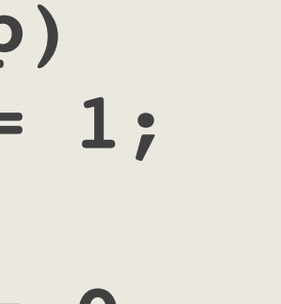
Example

```
FILE *f1, *f2;
int p1, p2;

A: compile() {
      if (p1)
         f1 = fopen();
      if (p2)
         f2 = fopen();
      doStuff();
   }
```

```
B: doStuff() {
      if (p1)
         x: rtl(f1);
      if (p2)
         y: rtl(f2);
      doStuff();
   }

C: rtl(FILE *f) {
      fprintf(f,?);
}
```

2 More Examples

```
if (dump)
   flag = 1;
else
   flag = 0;

if (dump)
   f = fopen();

if (flag)
   fclose(f);
```



```
if (dump) {
   f = fopen();
   flag = 1;
} else
   flag = 0;

if (flag)
   fclose(f);
```

ESP

- CFG construction
- Value flow computation
- Abstract CFG construction
- Interface expression computation
- Property simulation

Summary

- independent values
- temporal properties
- few relevant branches
- implicit correlations

Discussion I

- "This is a useful, non-trivial property *that cannot be expressed using types*."

Agree / Disagree?

Discussion II

- related work: **Vault** keeps typestate

  Do you think Vault (or Rust) will be "the future"?

Discuss III

- Benchmark: **`gcc 2.5.3`** from SPEC'95
  - SPEC is not free
  - SPEC'95 was retired in 2000

Ben Greenman presents:

ESP: Path-Sensitive Program Verification in Polynomial Time

Manuvir Das
Sorin Lerner
Mark Seigle

PLDI 2002

Discuss III

- Benchmark: **`gcc 2.5.3`** from SPEC'95
  - SPEC is not free (~$300)
  - SPEC'95 was retired in 2000
- Only evaluates correct code!
- Evaluation doesn't measure CFG-building time

Discuss VI

- Das, PLDI 2000 analyzed a 1.4 million LOC program
  - ... it was MS Word 97
- Why is Word so big?

Notes

- implementation "extends" IFDS
- polynomial runtime (fast "in practice")
- useful starting point for SLAM