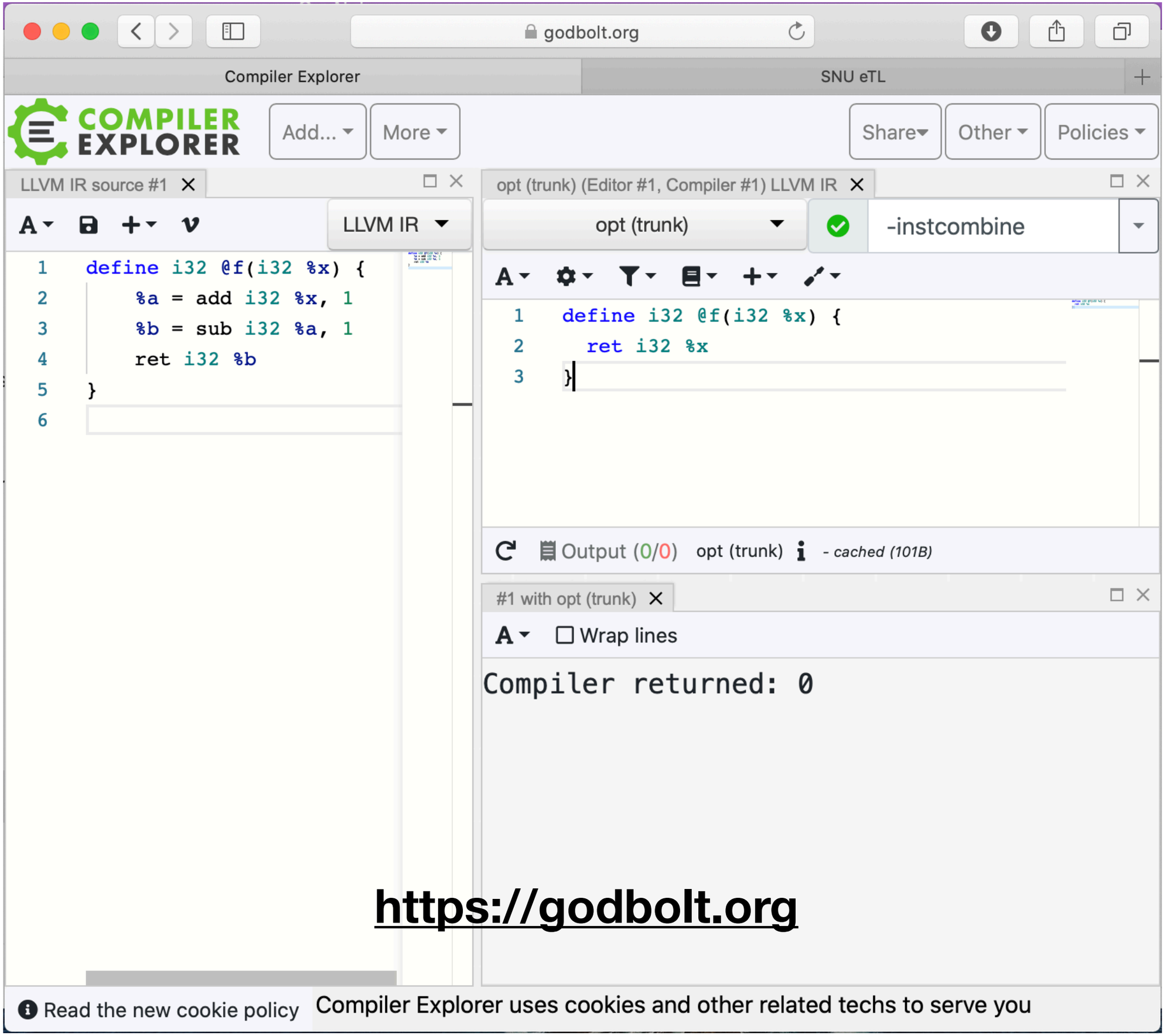
Using LLVM IR

2019. 3. 26
SWPP Practice Session
Juneyoung Lee

Getting LLVM

- If you already have compiled LLVM, you can use it
- Otherwise, using the prebuilt binary is enough for assn. 2
  - https://releases.llvm.org
  - 9.0, Pre-Built Binaries
    (10.0: may not have binaries for your OS yet)

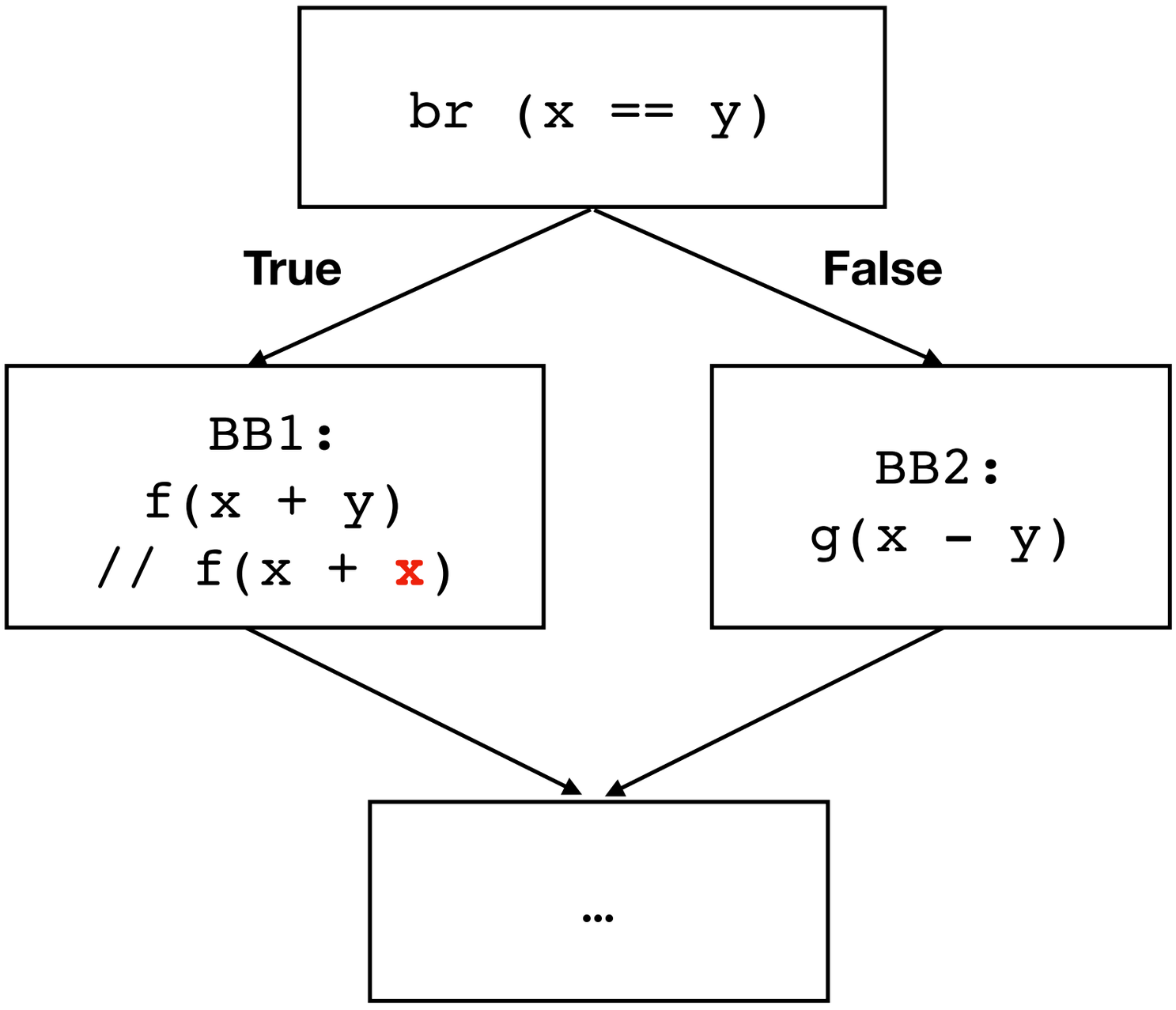

godbolt.org
Compiler Explorer
SNU eTL
COMPILER EXPLORER
Add...
More
Share
Other
Policies
LLVM IR source #1
LLVM IR
1 define i32 @f(i32 %x) {
2 %a = add i32 %x, 1
3 %b = sub i32 %a, 1
4 ret i32 %b
5 }
6
opt (trunk) (Editor #1, Compiler #1) LLVM IR
opt (trunk)
-instcombine
1 define i32 @f(i32 %x) {
2 ret i32 %x
3 }
Output (0/0) opt (trunk) - cached (101B)
#1 with opt (trunk)
Wrap lines
Compiler returned: 0
https://godbolt.org
Read the new cookie policy
Compiler Explorer uses cookies and other related techs to serve you

Assignment 2

- Count the # of evens from an i32 array
- Skeleton will be posted today

```
define i32 @count_even(i32* %arr, i32 %N) {
    ; Count the number of evens from
    ; %arr[0 … %N - 1]
    ; precond: 0 <= N <= 20
}
```

Critical Edge Splitting (revisited)



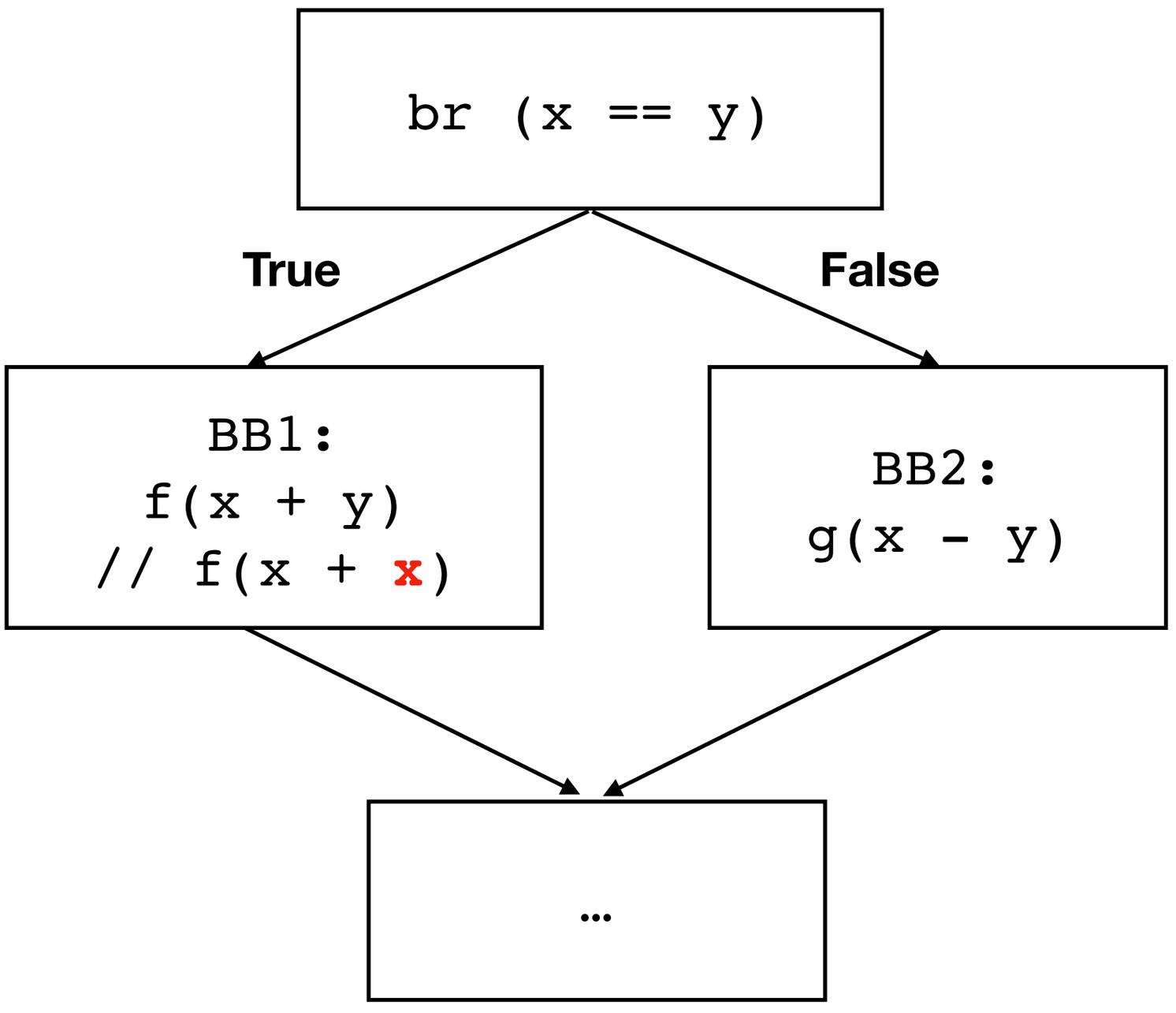
Critical Edge Splitting (revisited)

Okay, let's write an algorithm for optimization!

Whenever we visit **br(x == y)**,

(1) We iterate over **uses** of y

(2) If the use is **dominated** by BB1, let's replace it with x!



Critical Edge Splitting (revisited)

Okay, let's write an algorithm for optimization!

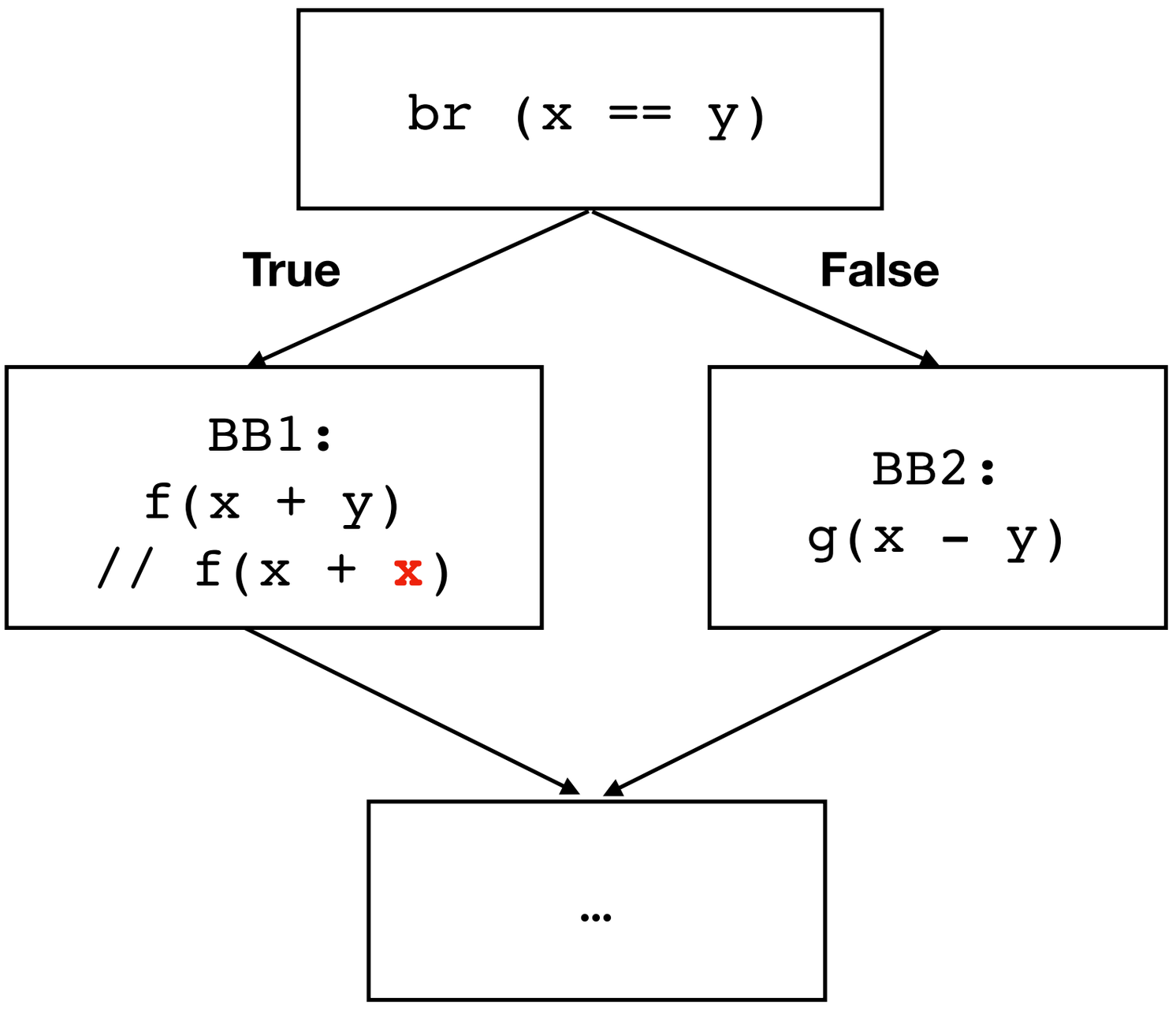
Whenever we visit **br(x == y)**,

(1) We iterate over **uses** of y

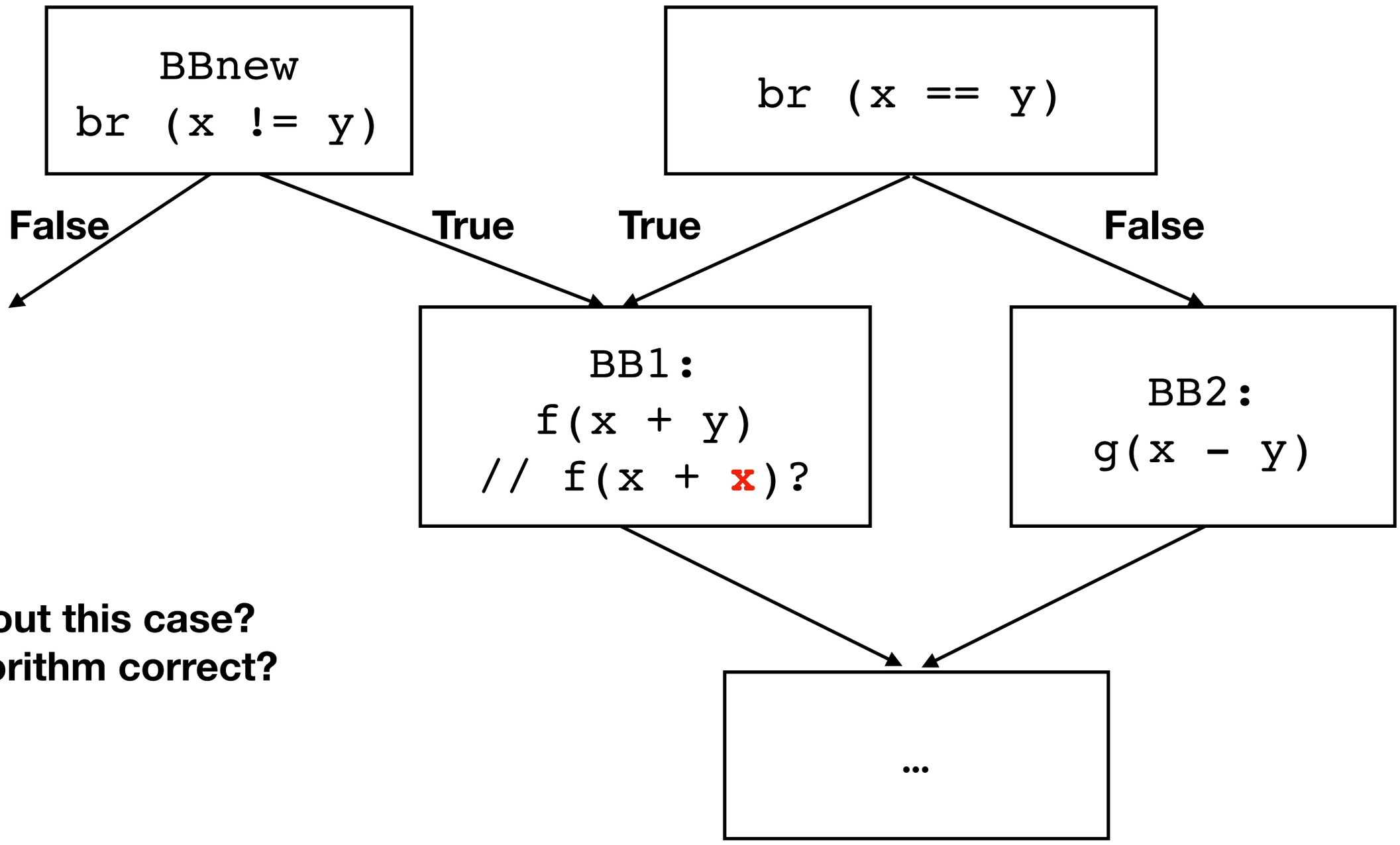
(2) If the use is **dominated** by BB1, let's replace it with x!

Is this algorithm correct?

🤔

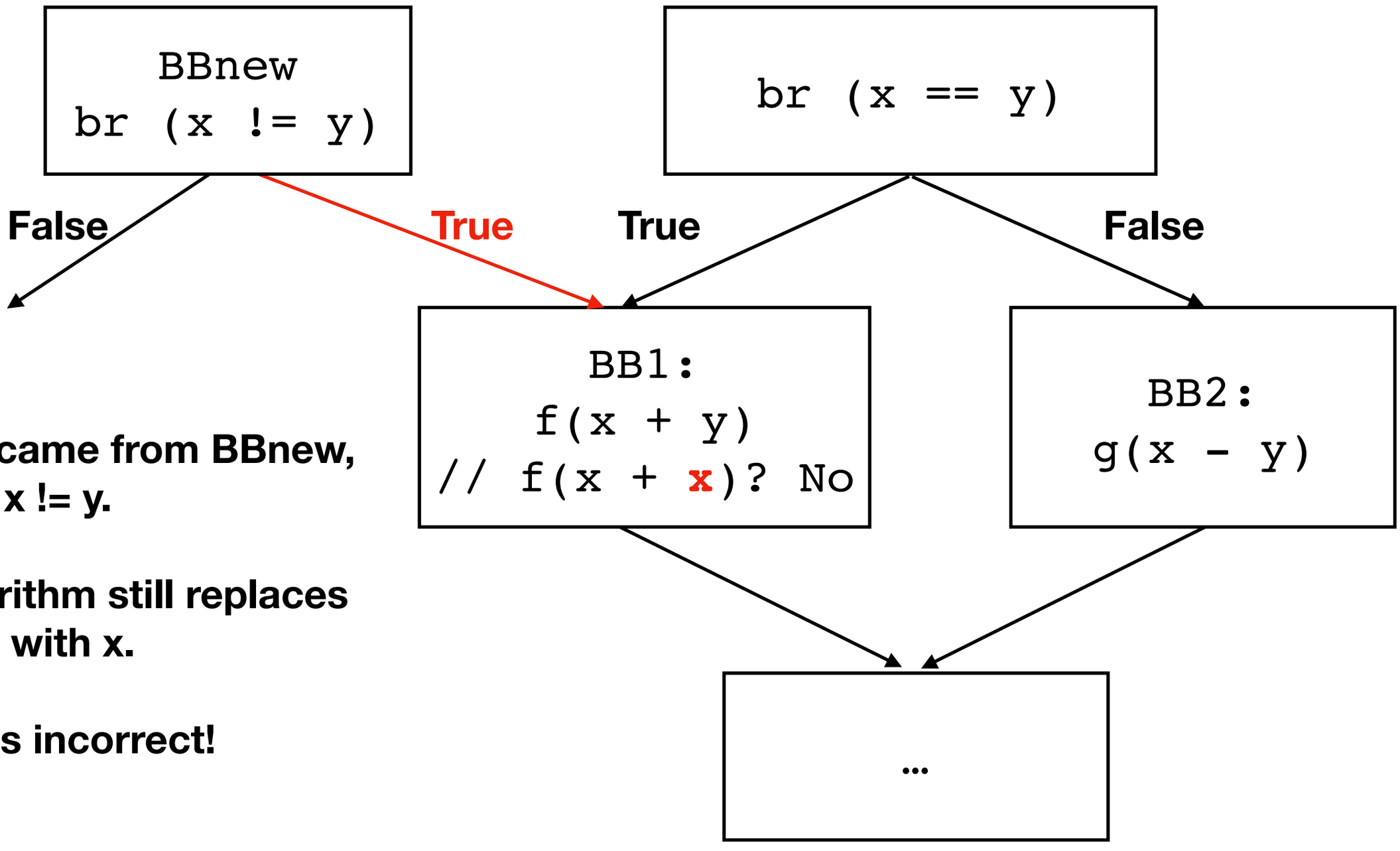


Critical Edge Splitting (revisited)



What about this case?
Is the algorithm correct?

Critical Edge Splitting (revisited)

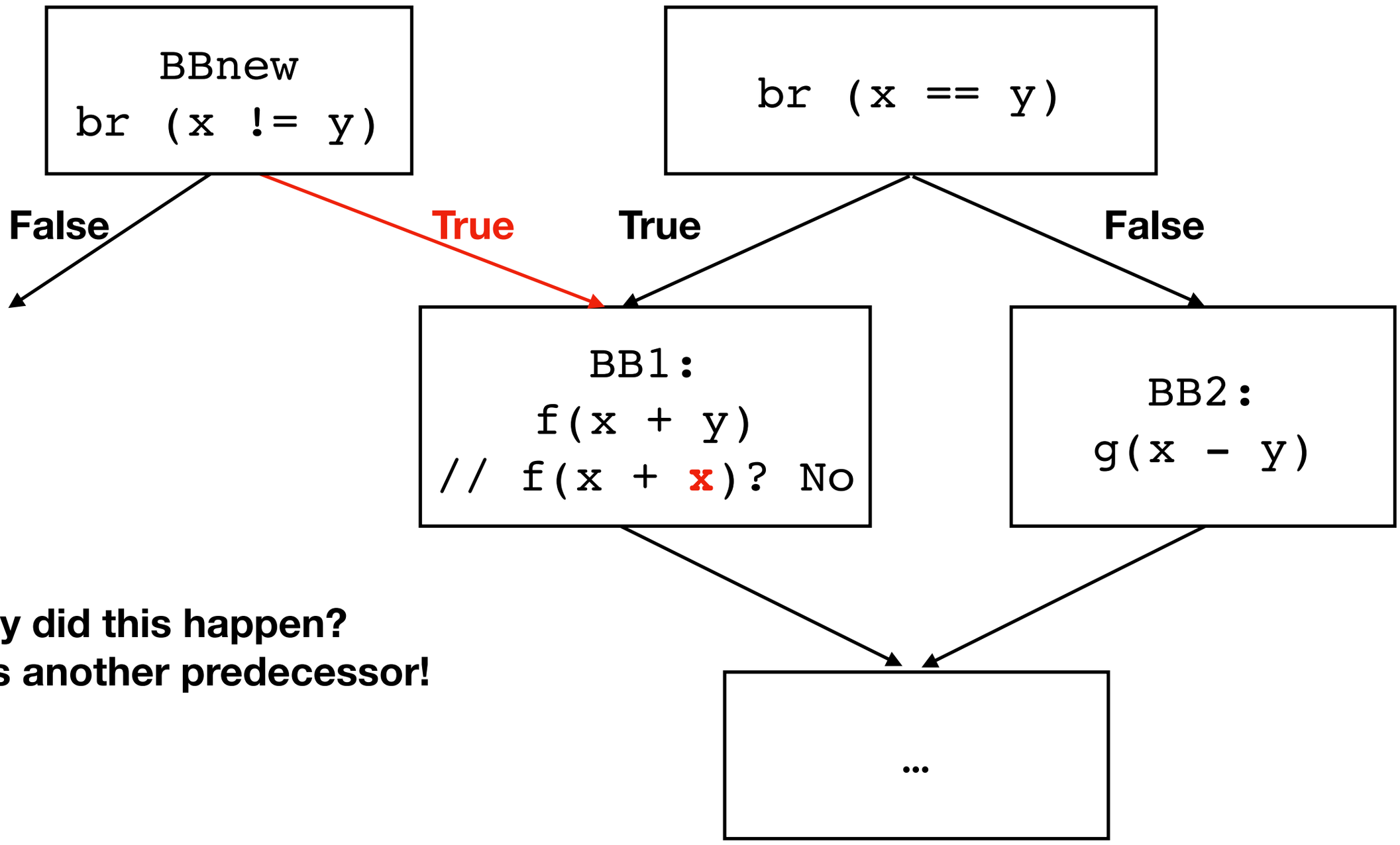


If execution came from BBnew, x != y.

But the algorithm still replaces y with x.

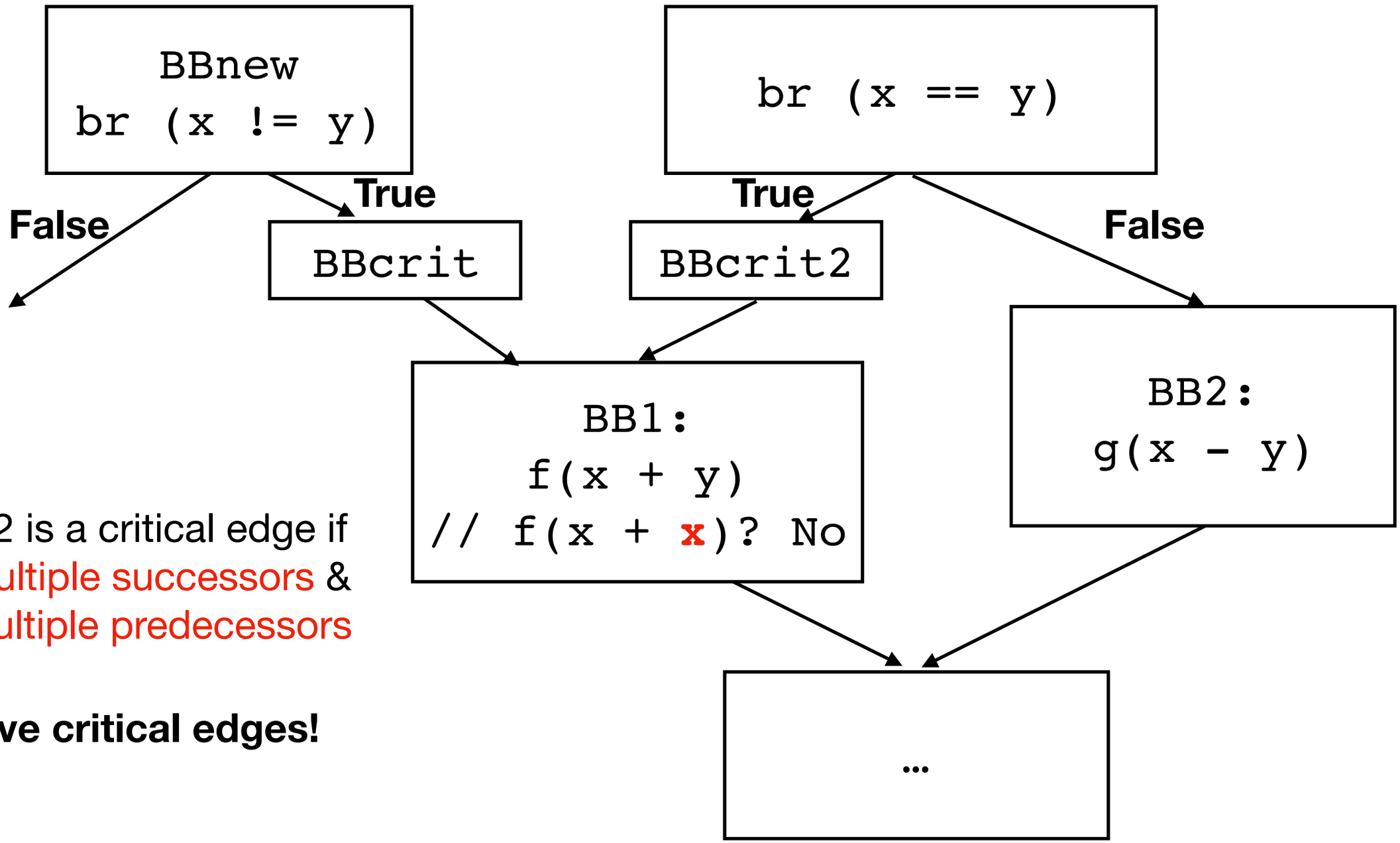
This is incorrect!

Critical Edge Splitting (revisited)



Q: Why did this happen?
A: BB1 has another predecessor!

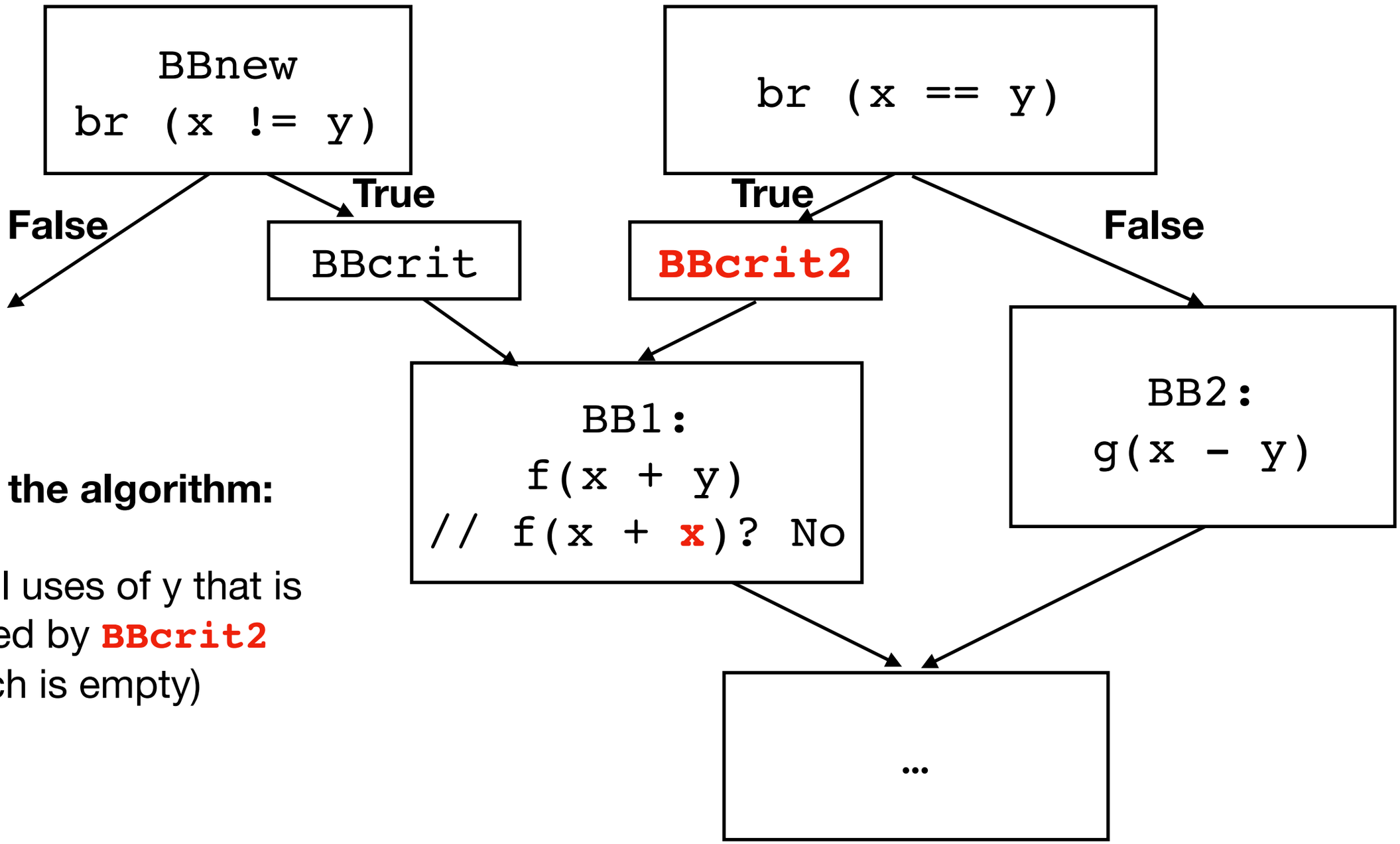
Critical Edge Splitting (revisited)



BB1 -> BB2 is a critical edge if
BB1 has multiple successors &
BB2 has multiple predecessors

Let's remove critical edges!

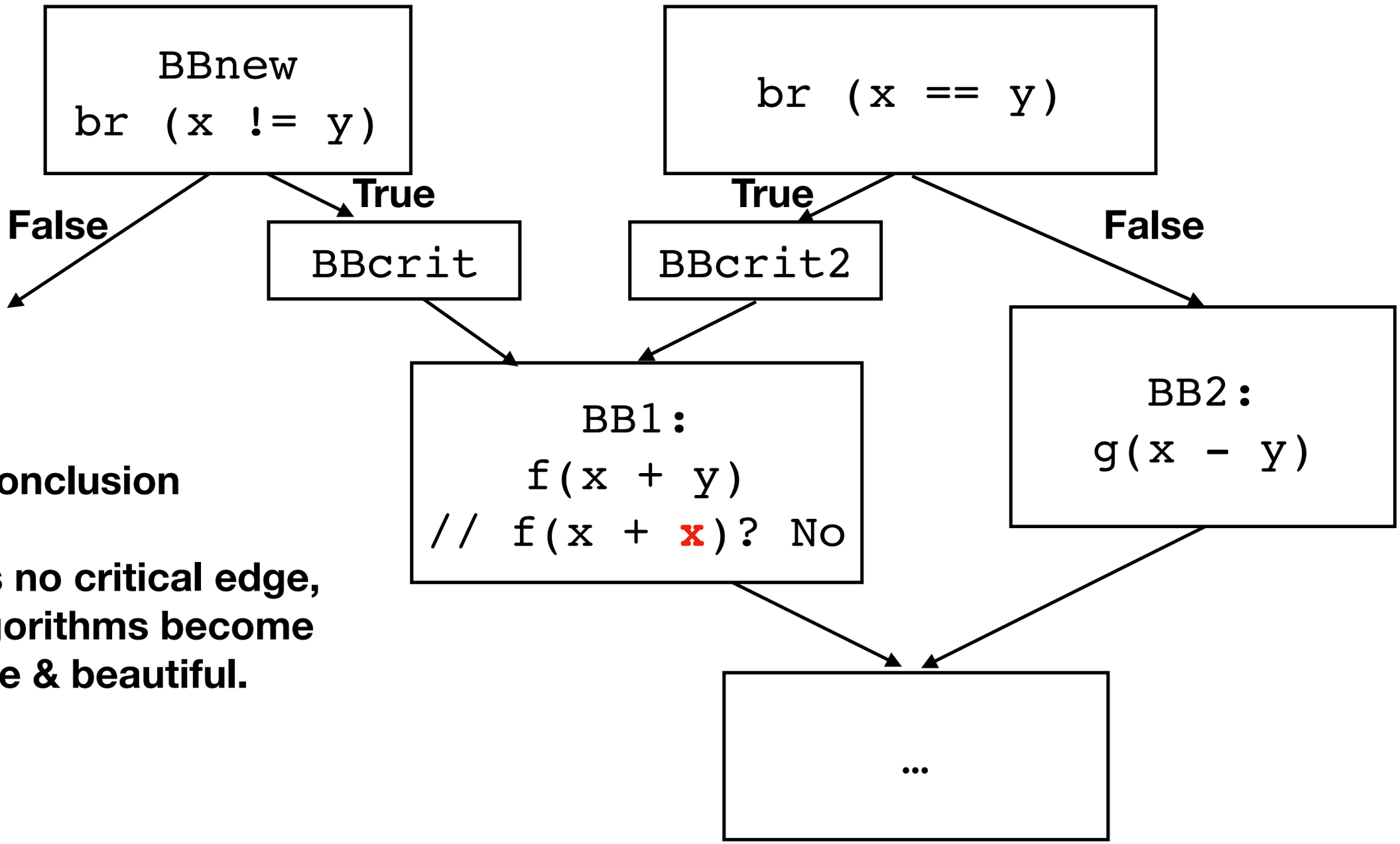
Critical Edge Splitting (revisited)



Reapply the algorithm:

Replace all uses of y that is dominated by **BBcrit2** (which is empty)

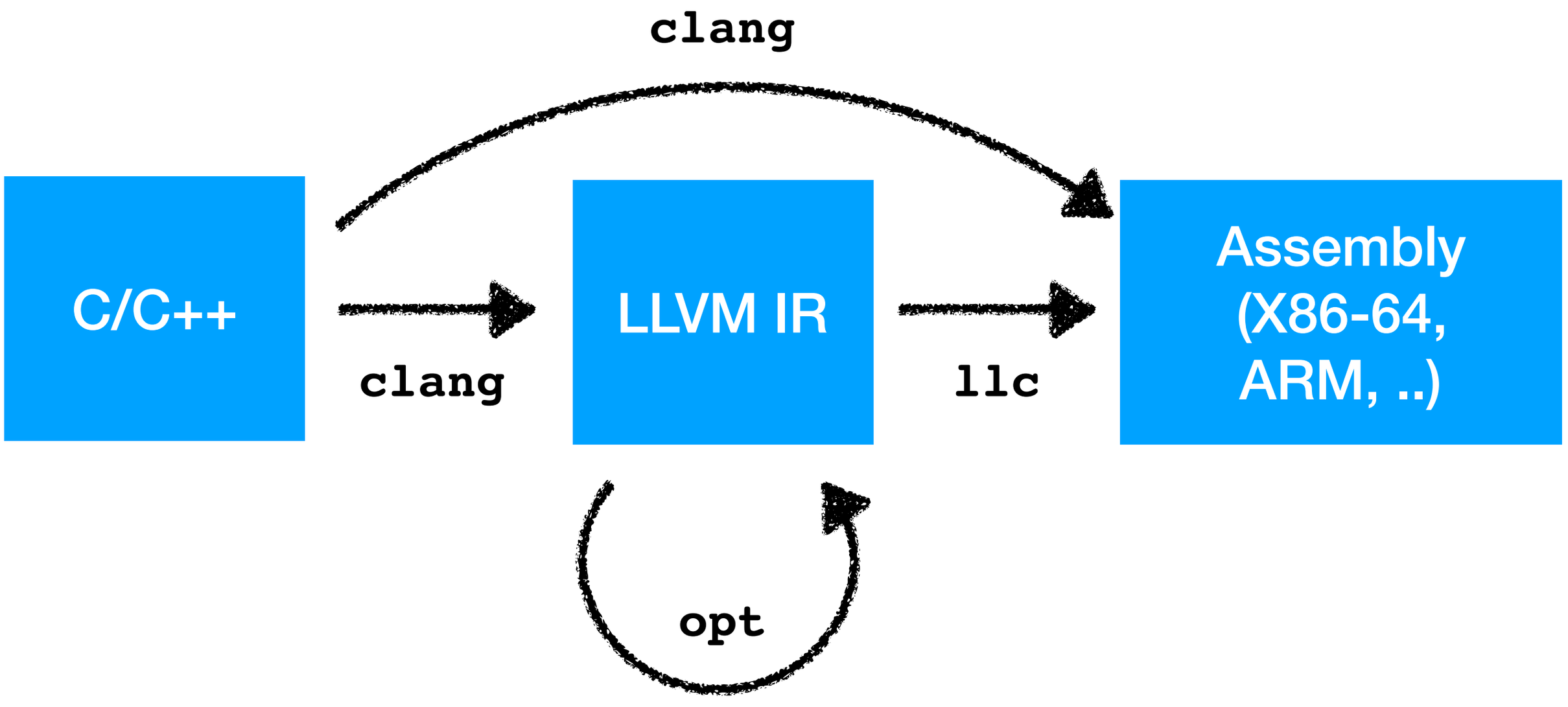
Critical Edge Splitting (revisited)



Conclusion

If there is no critical edge, many algorithms become simple & beautiful.

Converting LLVM IR from/to *



Example - fibonacci

```
1   unsigned fib(unsigned n) {
2       unsigned answ;
3       if (n <= 1)
4           answ = n;
5       else
6           answ = fib(n - 1) + fib(n - 2);
7       return answ;
8   }
```

https://godbolt.org/z/QkuFvZ

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

3 Basic blocks: entry, if.else, if.end

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

icmp: integer comparison
ult: unsigned comparison, less than

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

LLVM Language Reference Manual: https://llvm.org/docs/LangRef.html

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

Note that the branch condition is inversed
There is no special reason in this case.. :/

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

Multiple definitions of variables in different blocks are merged with a phi node.

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

Play with fibonacci

- C program & IR program (fib.c / fib.ll): see the class GitHub repo
- C -> IR:

```
bin/clang -S -emit-llvm -O1 -g0 \
          -fno-discard-value-names fib.c -o -
```

- IR -> Assembly:

```
bin/llc -o fib.s fib.ll
```

fib.ll vs. fib.bc

- .ll file: textual form (human understandable form)
- .bc file: binary form (compact, faster for a machine to read)
- `bin/llvm-as fib.ll -o fib.bc`
- `bin/llvm-dis fib.bc -o fib.ll`

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

Global variables have prefix @

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

Dereference `%numbers`

A variable with numeric name (should increase by 1)

```
ouble 0.000000e+00

3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

`numbers[1]` **is** `*(numbers + 1)`
Let's calculate `(numbers + 1)` **first**

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

Dereference `(numbers+1)`

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2
7   }
```

Calculate its average

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2
7   }
```

Store the result to a global variable

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

Command

- C program & IR program (average.c / average.ll): see the repo

```
bin/clang -S -emit-llvm -O1 -g0 \
          -fno-discard-value-names \
          -fno-strict-aliasing average.c -o -
```