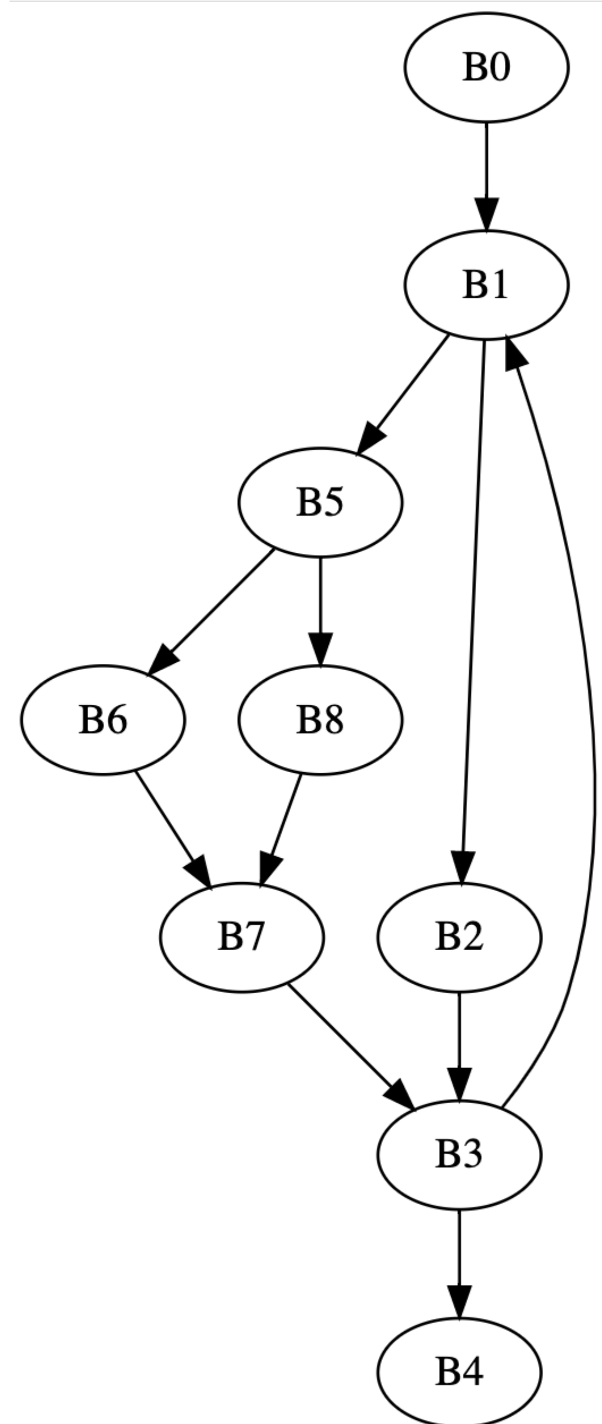
CSE211: Compiler Design

Oct. 15, 2021

- **Topic**: Regional optimizations, intro to global optimizations
- **Questions**:
  - *Can we apply local value numbering to an entire program?*
  - *What are examples of having unlimited registers vs. having limited registers?*



Announcements

- Homework 1:
  - Due on Monday (at 11:59 pm)
  - Help will be sparse in evenings and weekends!
  - zip up files and submit on Canvas
    - one or two zip files, doesn’t matter as long as I can easily get to the code!

- Homework 2:
  - Released Monday by midnight
  - 2 weeks to complete
    - Local Value Numbering
    - Live variable analysis (Monday)

Announcements

Next week:

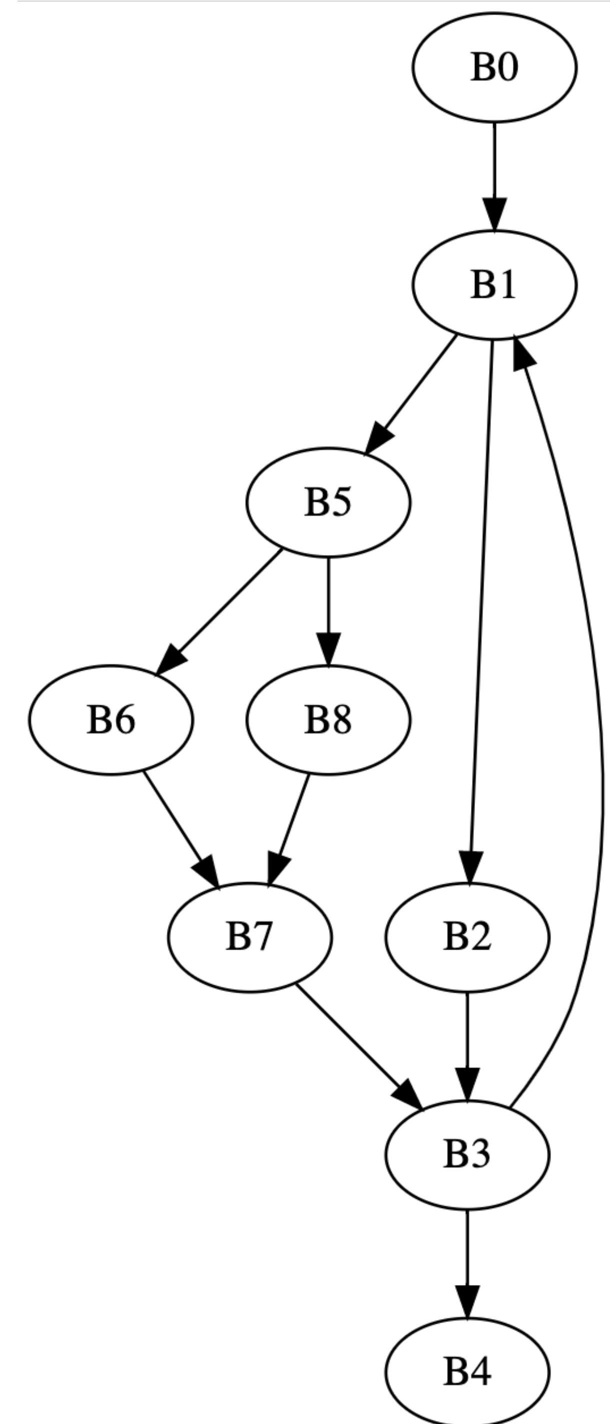
- Wednesday and Friday’s class will be remote:
  - I will be in Chicago
  - I will give a live lecture (zoom link on canvas), I would appreciate it if you attended
  - I will record the lecture and make it available online if you would prefer to attend asynchronously

CSE211: Compiler Design

Oct. 15, 2021

- **Topic**: Regional optimizations, intro to global optimizations

- **Questions**:
  - *Can we apply local value numbering to an entire program?*
  - *What are examples of having unlimited registers vs. having limited registers?*



Review local value numbering

- First step?

```
a2 = b0 + c1;
b4 = a2 - d3;
c = b + c;
d = a - d;
```

global_counter: 5

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      "b0 + c1" : "a2",
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
        "b0 + c1" : "a2",
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      “b0 + c1” : “a2”,
      “a2 - d3” : ”b4”,
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      “b0 + c1” : “a2”,
      “a2 - d3” : ”b4”,
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      "b0 + c1" : "a2",
      "a2 - d3" : "b4",
}
```

mismatch due to numberings!

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      "b0 + c1" : "a2",
      "a2 - d3" : "b4",
      "b4 + c1" : "c5",
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = a2 - d3;
```

```
H = {
      "b0 + c1" : "a2",
      "a2 - d3" : "b4",
      "b4 + c1" : "c5",
}
```

Review local value numbering

```
a2 = b0 + c1;
b4 = a2 - d3;
c5 = b4 + c1;
d6 = b4;
```

```
H = {
      "b0 + c1" : "a2",
      "a2 - d3" : "b4",
      "b4 + c1" : "c5",
}
```

match!

Optimizing over wider regions

- Local value numbering operated over just one basic block.

- We want optimizations that operate over:
  - several basic blocks (regional)
  - across an entire procedure (global)

- For this, we need Control Flow Graphs

Control flow graphs

A graph where:

- nodes are basic blocks
- edges mean that it is possible for one block to branch to another

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

else:
r3 = ...;

end_if:
r4 = ...;
```

Control flow graphs

A graph where:

- nodes are basic blocks
- edges mean that it is possible for one block to branch to another

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;
```

```
if:
r2 = ...;
br end_if;
```

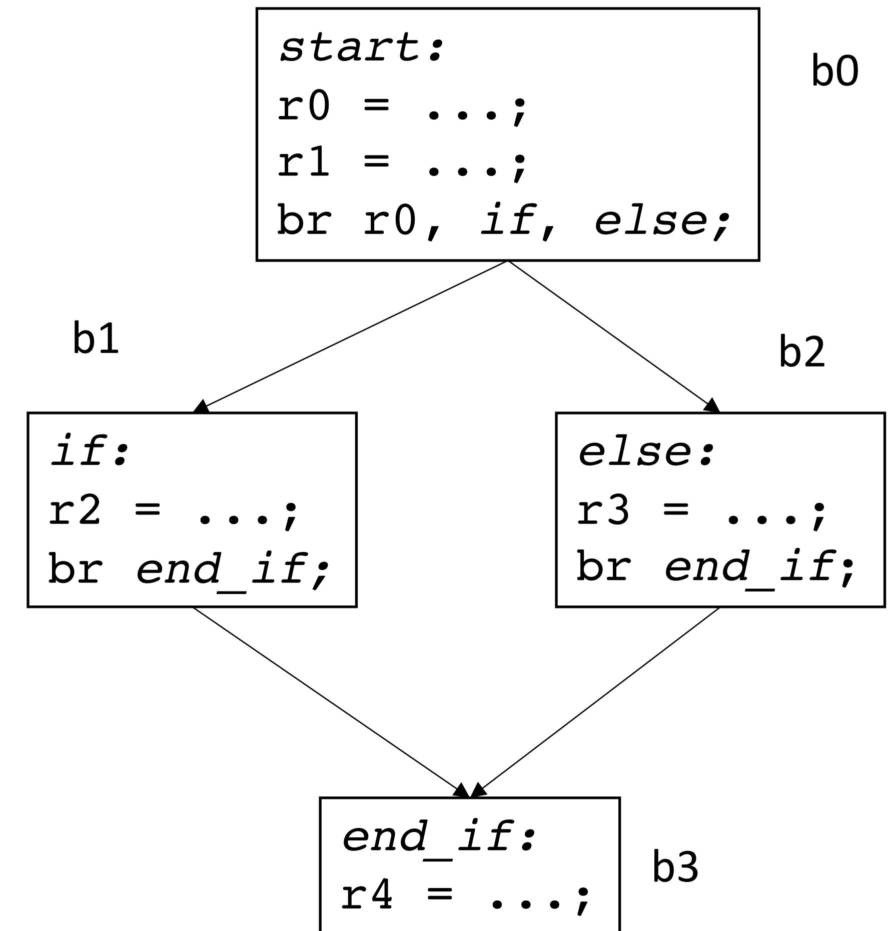
```
else:
r3 = ...;
br end_if;
```

```
end_if:
r4 = ...;
```

Control flow graphs

A graph where:

- nodes are basic blocks
- edges mean that it is possible for one block to branch to another



Interesting CFGs

Interesting CFGs

- Exceptions
- Break in a loop
- Switch statement (consider break, no break)
- first class branches (or functions)

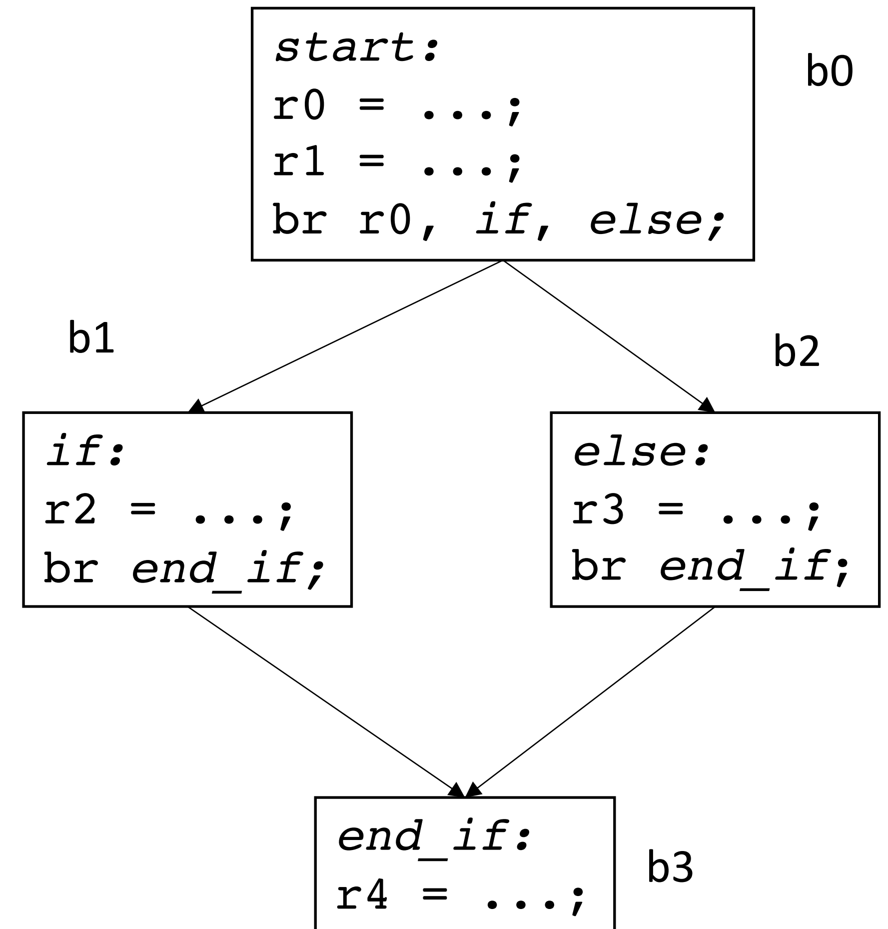
Regional optimizations

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

else:
r3 = ...;

end_if:
r4 = ...;
```

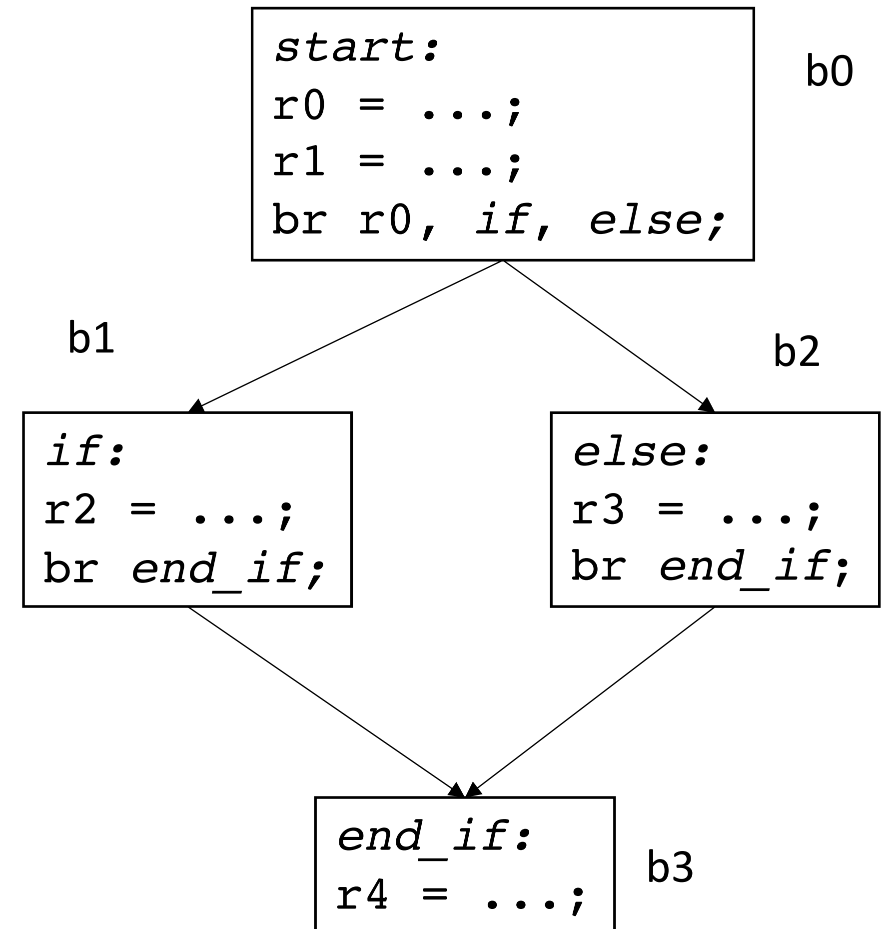
Regional optimizations

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements



Super local value numbering

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements

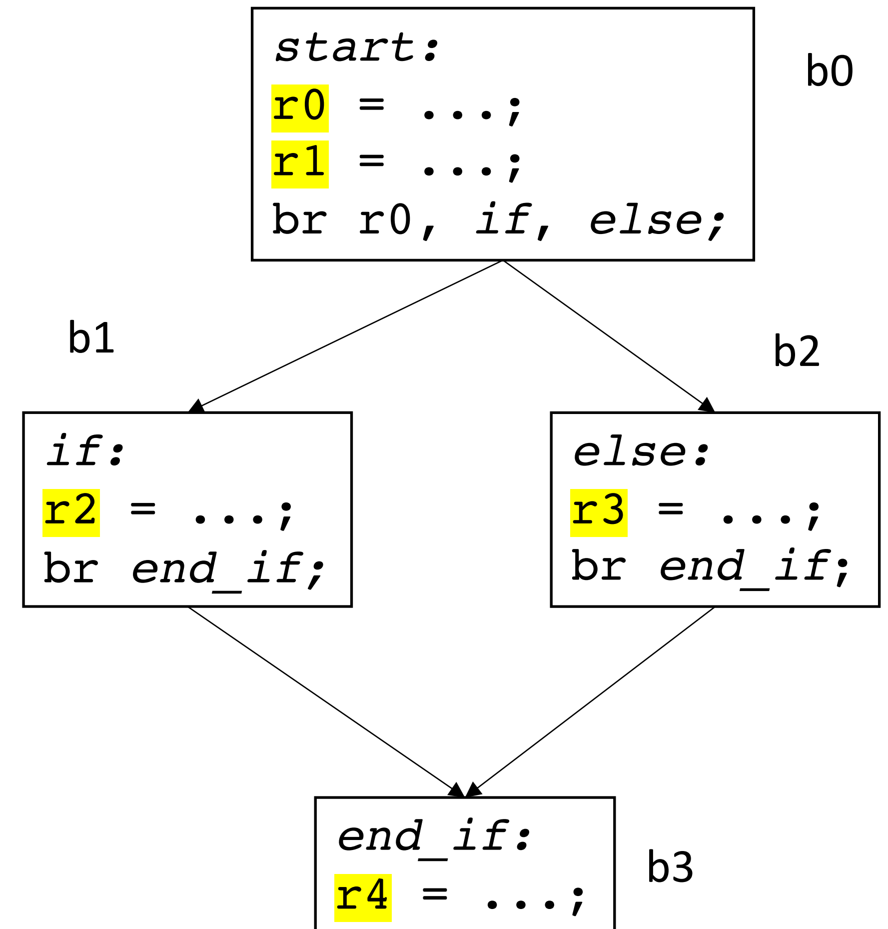


What are the implications of doing local value numbering in each of the basic blocks?

Super local value numbering

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements

Global counter would need to be kept across blocks when numbering



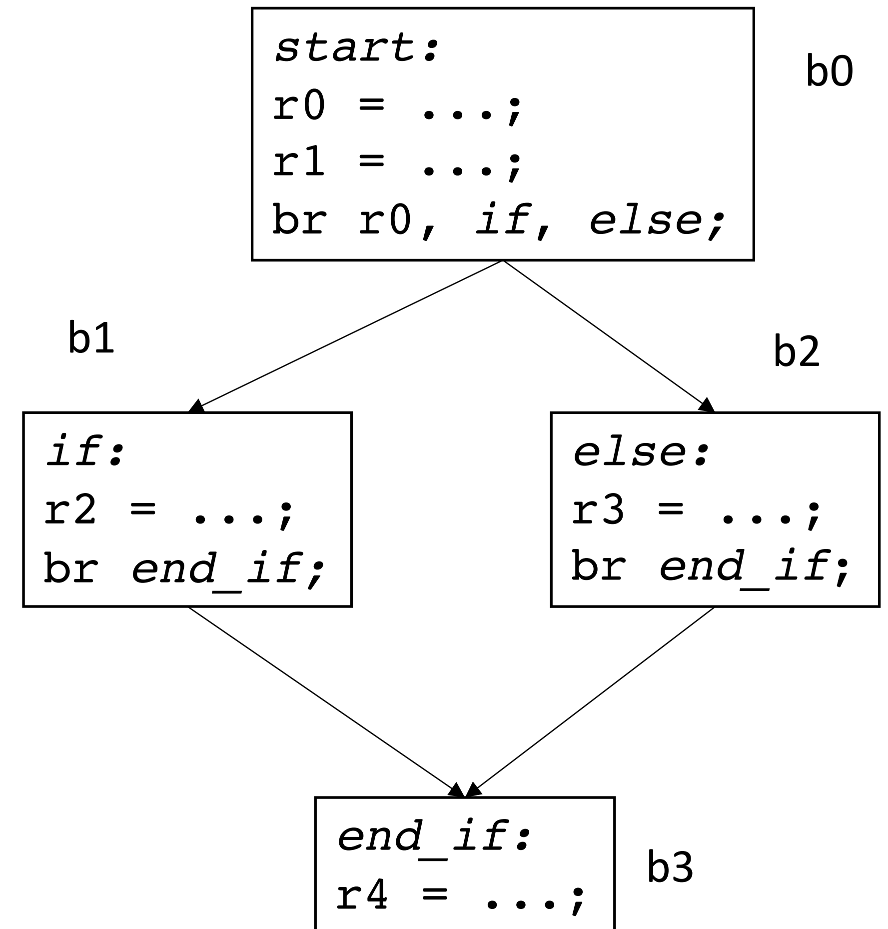
What are the implications of doing local value numbering in each of the basic blocks?

Super local value numbering

breadth first traversal, creating hash tables for each block

```
b0_H = {
        "..." : "r0",
        "..." : "r1",
}
```

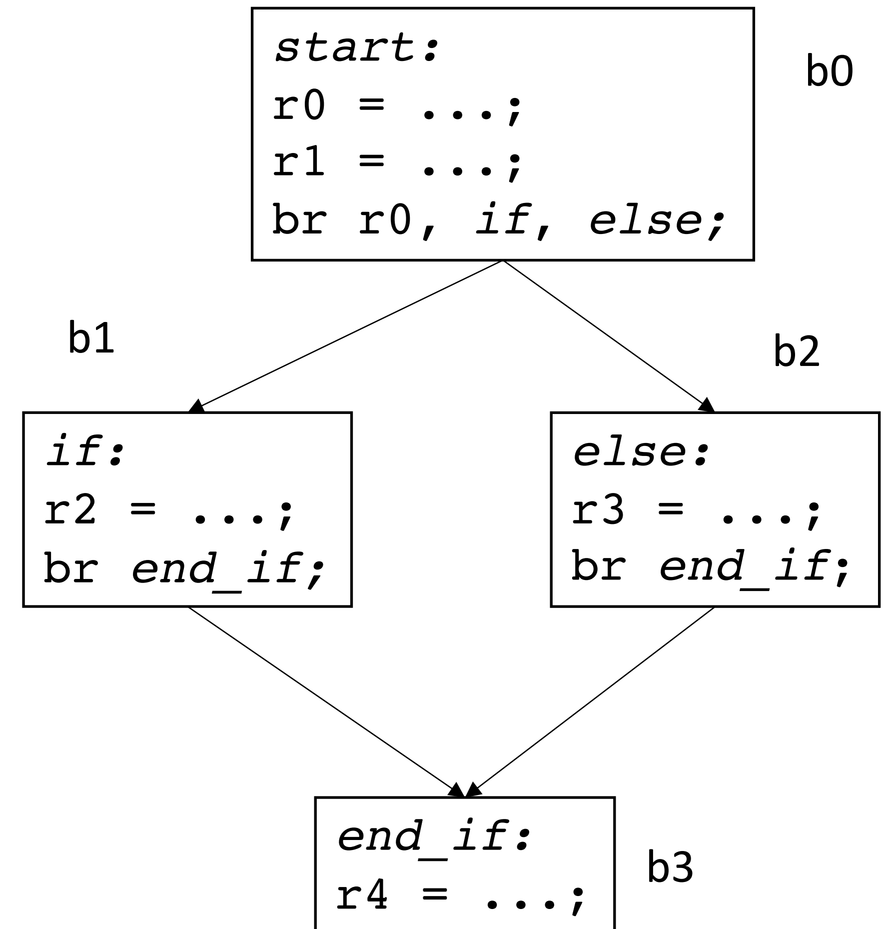
- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements



What are the implications of doing local value numbering in each of the basic blocks?

Super local value numbering

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements



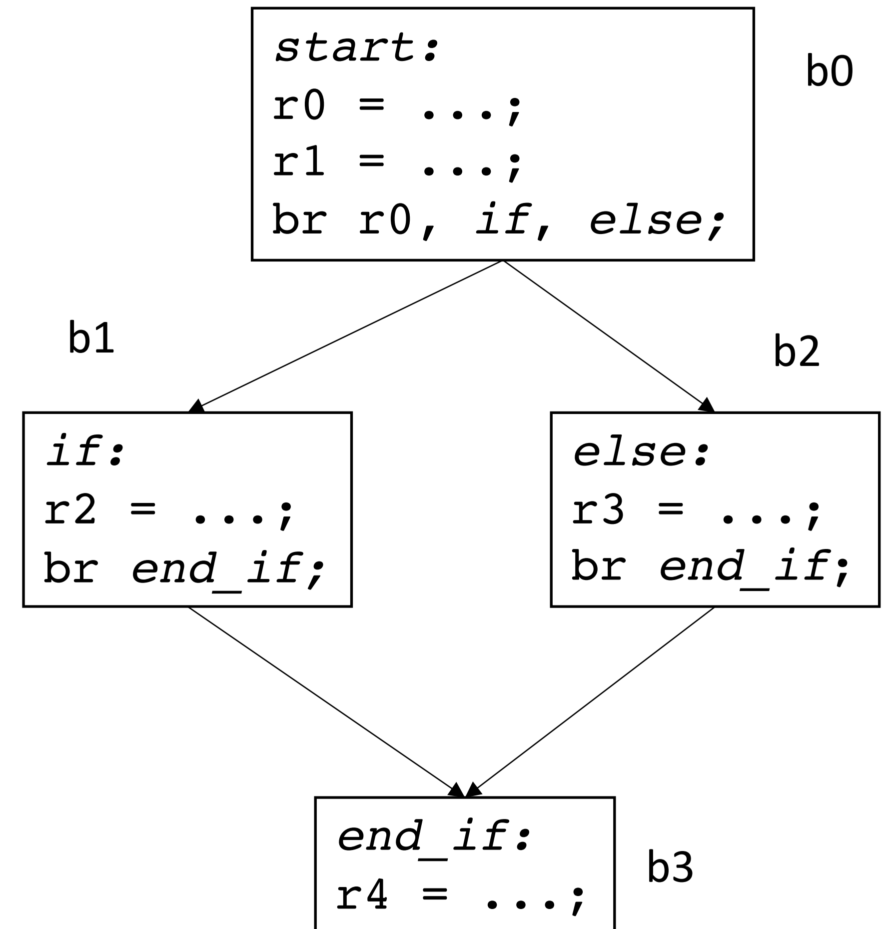
Do local value numbering, but start off with a non-empty hash table!

Which blocks can use which hash tables?

Super local value numbering

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements

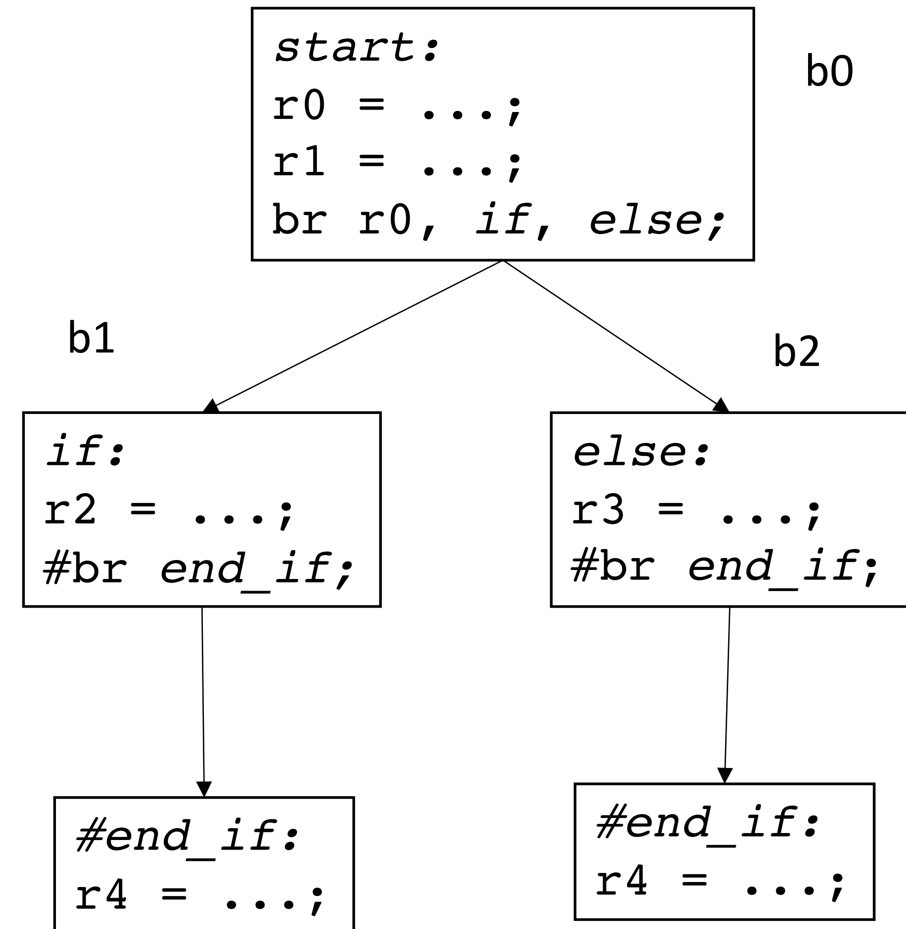
Is it possible to re-write so that b3 can use expressions from b1 or b2?



Super local value numbering

- Usually constrained to a "common" subset of the CFG:
- For example: if/else statements

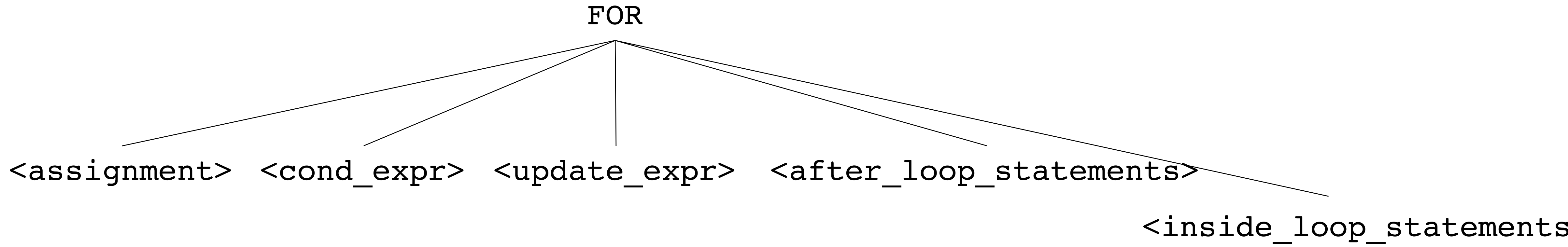
```
b0_H = {
        "..." : "r0",
        "..." : "r1",
}
```



Is it possible to re-write so that b3 can use expressions from b1 and b2? Duplicate blocks and merge!

Pros? Cons?

Loop unrolling:



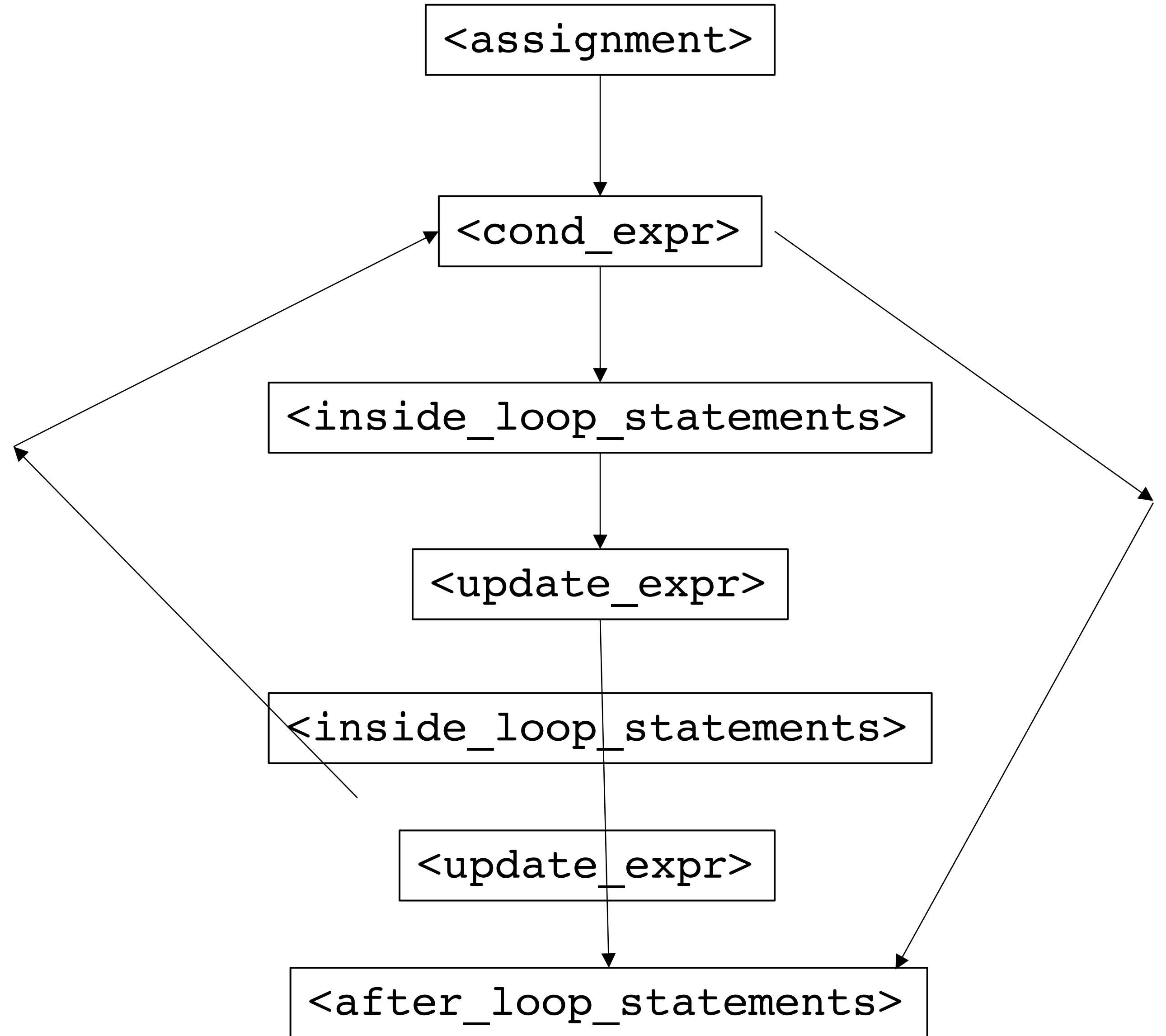
If all of these are basic blocks then the CFG looks like:

Loop unrolling:

```
<assignment>

<cond_expr>

<inside_loop_statements>

<update_expr>

<after_loop_statements>
```

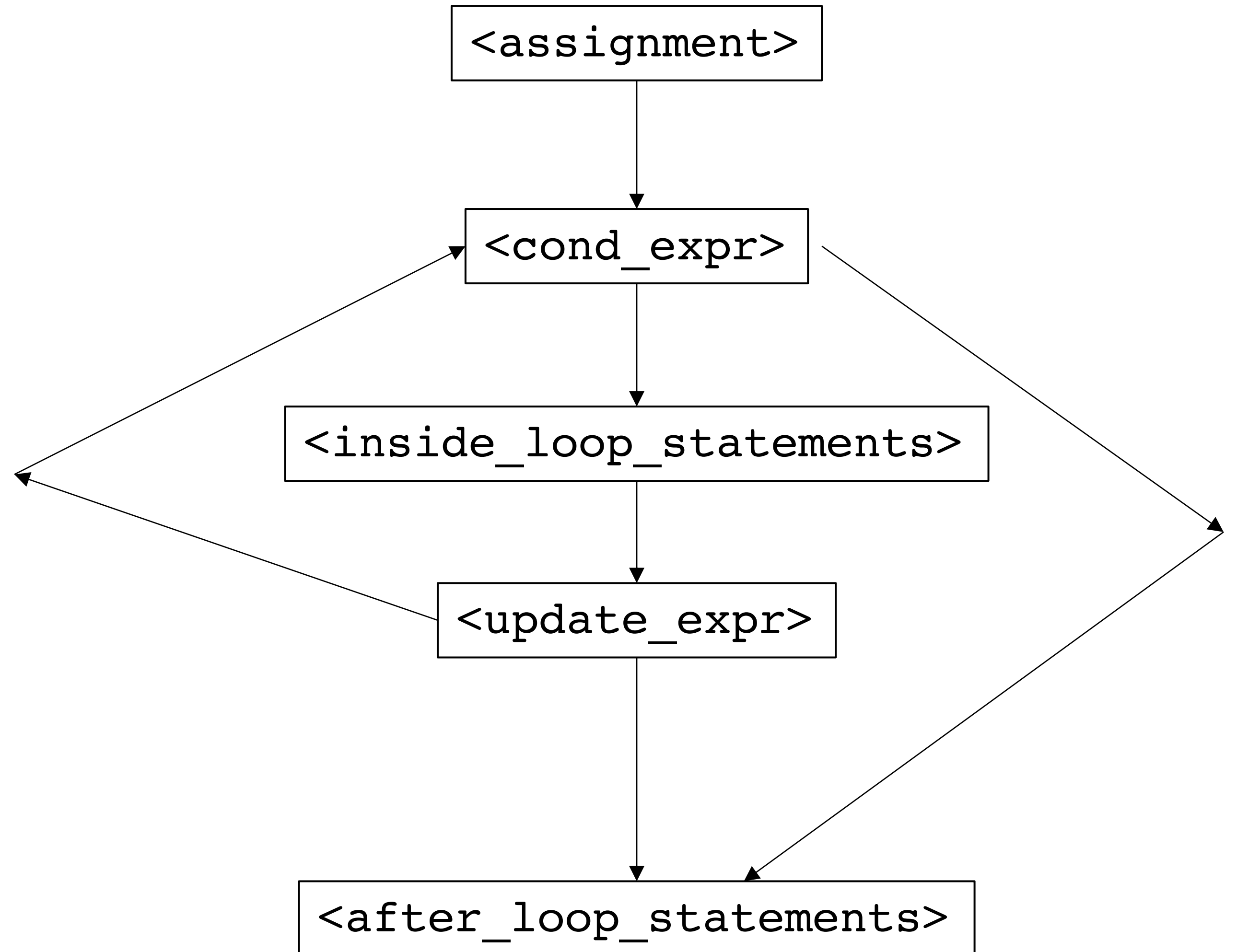
If all of these are basic blocks then the CFG looks like:

Loop unrolling:



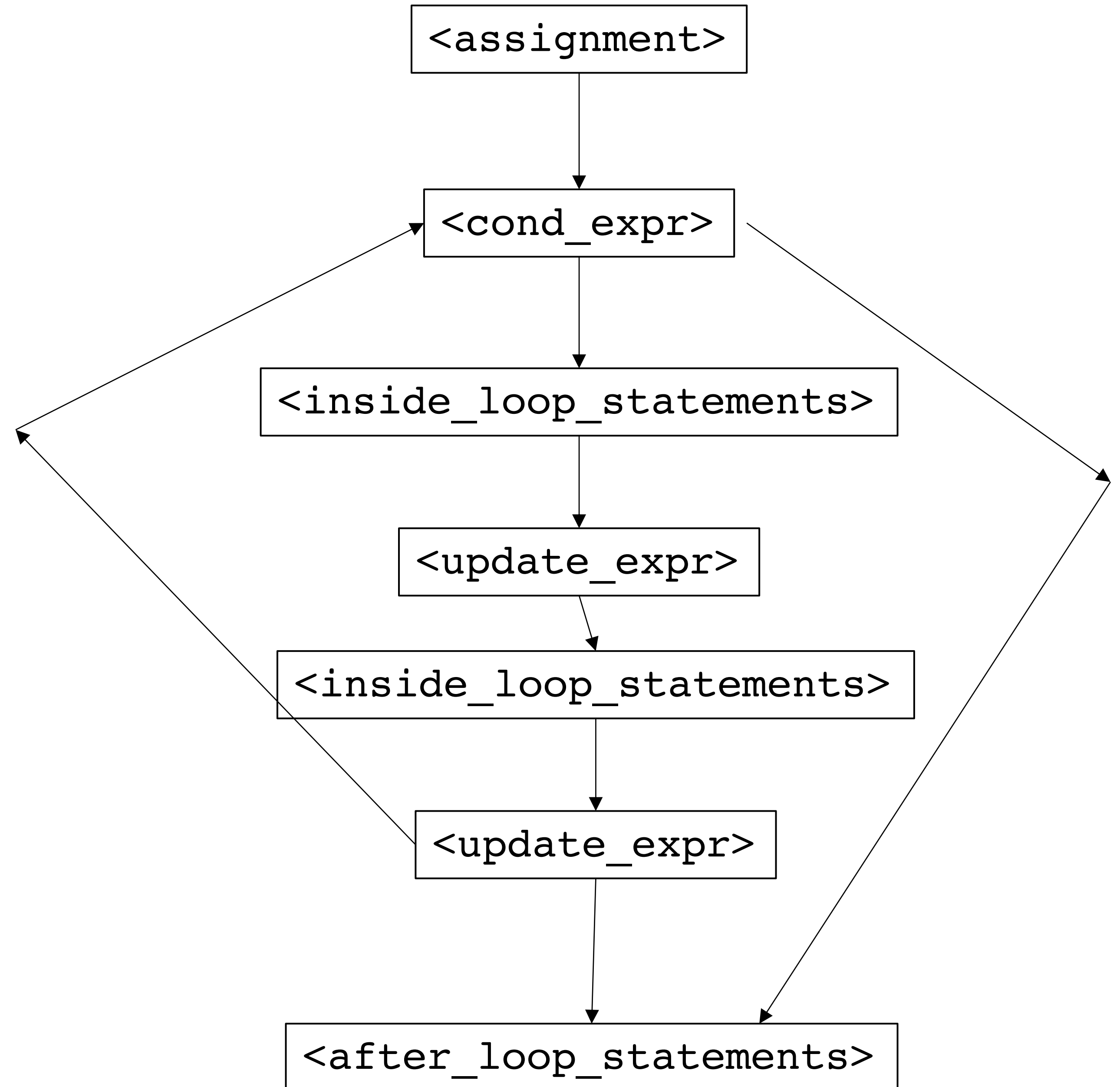
Loop unrolling:

Assume we
know that the loop will
iterate an even number
of times:



Loop unrolling:

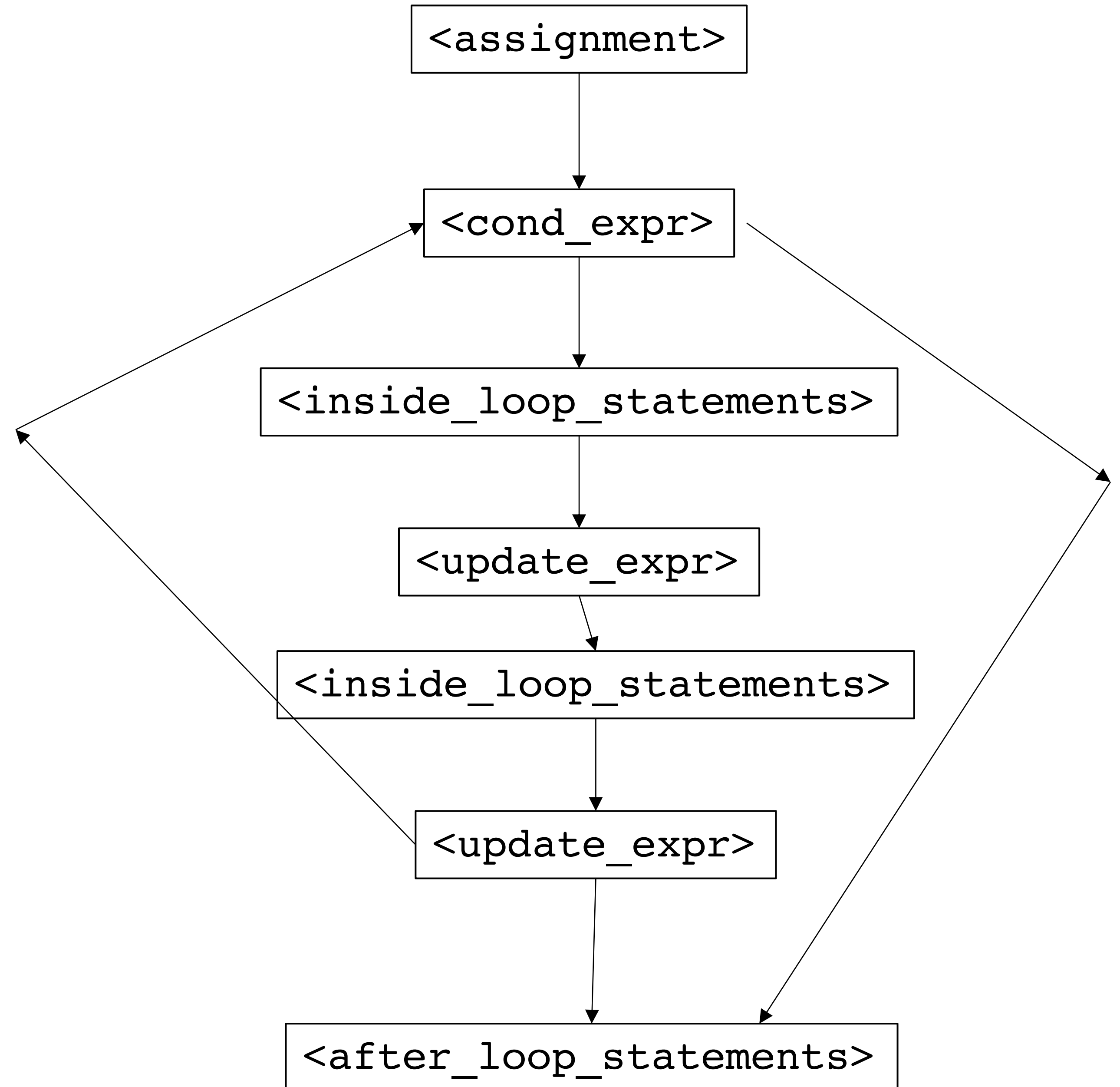
Assume we
know that the loop will
iterate an even number
of times:



Loop unrolling:

Assume we
know that the loop will
iterate an even number
of times:

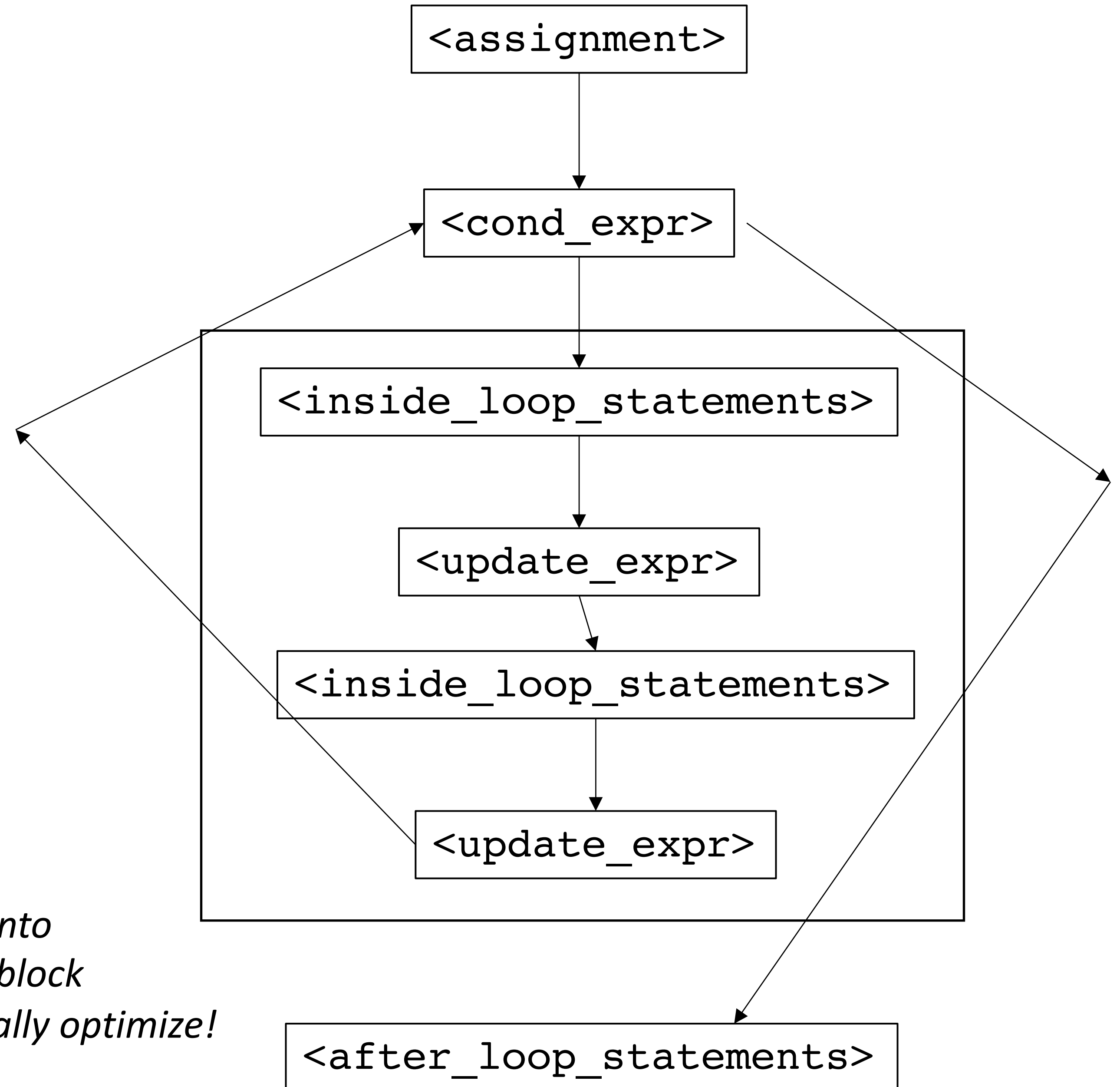
What have we saved here?



Loop unrolling:

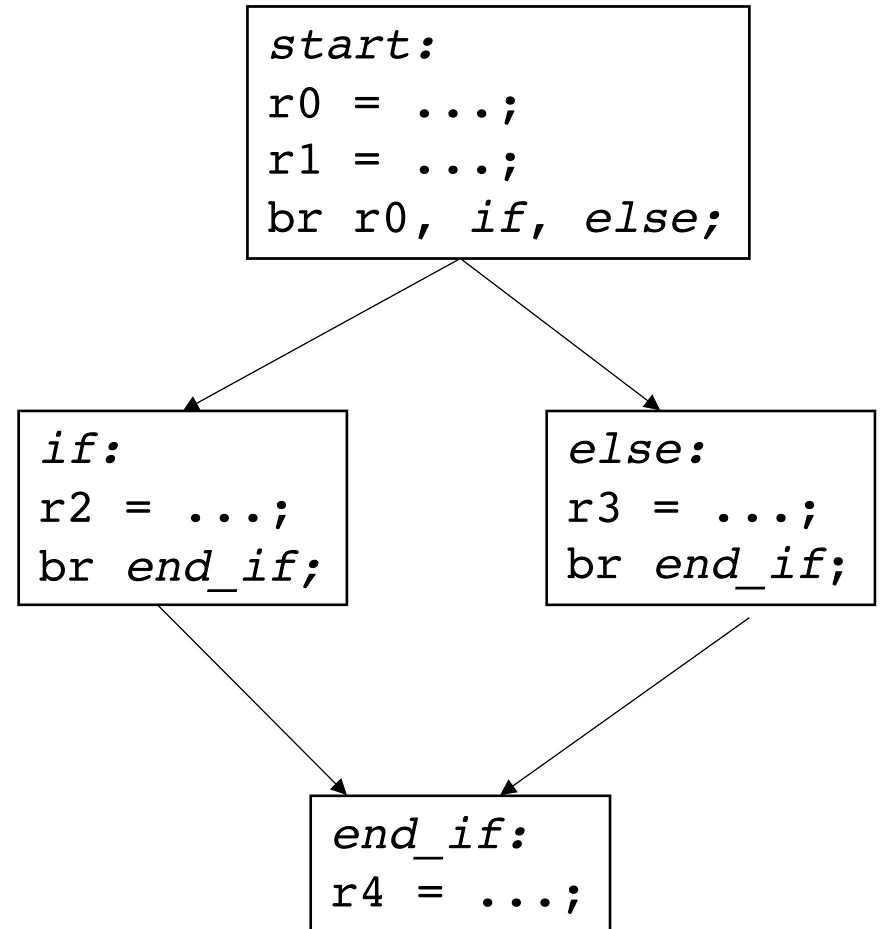
Assume we
know that the loop will
iterate an even number
of times:

What have we saved here?



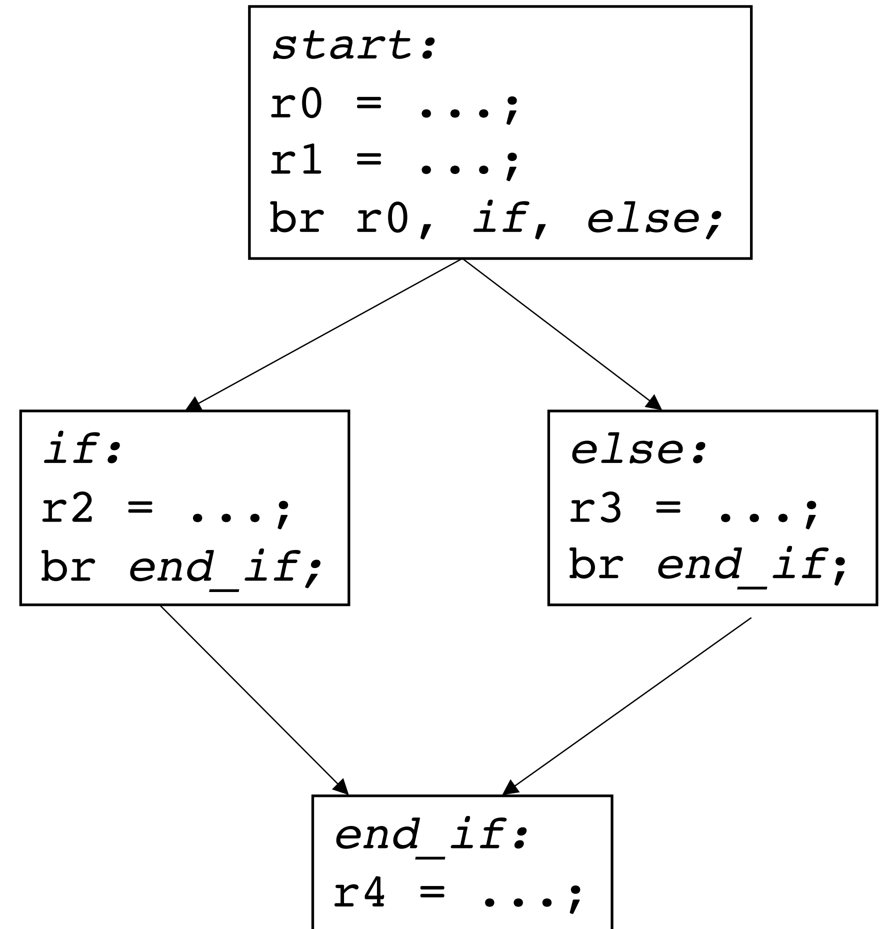
Code placement:

- Back to if/else



Code placement:

- Back to if/else
- Eventually we will straight line the code:



Code placement:

- Back to if/else
- Eventually we will straight line the code:

one option, what else?

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;
```

```
if:
r2 = ...;
br end_if;
```

```
else:
r3 = ...;
br end_if;
```

```
end_if:
r4 = ...;
```

Code placement:

- Back to if/else
- Eventually we will straight line the code:

one option, what else?

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

else:
r3 = ...;
br end_if;

end_if:
r4 = ...;
```

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

else:
r3 = ...;
br end_if;

if:
r2 = ...;
br end_if;

end_if:
r4 = ...;
```

Performance impact between the two?

Code placement:

- Back to if/else
- Eventually we will straight line the code:

one option, what else?

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

else:
r3 = ...;
br end_if;

end_if:
r4 = ...;
```

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

else:
r3 = ...;
br end_if;

if:
r2 = ...;
br end_if;

end_if:
r4 = ...;
```

If we know that one branch is taken more often than the other...
say the branch is true most often

Code placement:

- Back to if/else
- Eventually we will straight line the code:

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;
```

```
if:
r2 = ...;
br end_if;
```

```
else:
r3 = ...;
br end_if;
```

```
end_if:
r4 = ...;
```

If we know that one branch is taken more often than the other...
say the branch is true most often

How many branches here

Code placement:

- Back to if/else
- Eventually we will straight line the code:

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;
```

```
if:
r2 = ...;
br end_if;
```

```
else:
r3 = ...;
br end_if;
```

```
end_if:
r4 = ...;
br next_lbl
```

If we know that one branch is taken more often than the other...
say the branch is true most often

How many branches here

Code placement:

- Back to if/else
- Eventually we will straight line the code:

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

else:
r3 = ...;
br end_if;

end_if:
r4 = ...;
br next_lbl
```

```
start:
r0 = ...;
r1 = ...;
br r0, if, else;

if:
r2 = ...;
br end_if;

end_if:
r4 = ...;
br next_lbl

else:
r3 = ...;
br end_if;
```

If we know that one branch is taken more often than the other...
say the branch is true most often

How many branches here reduced branching by 1

Global optimizations

- Difference between regional:
  - handle arbitrary CFGs, cannot rely on structure!
  - Algorithms become more general
  - Potential for more optimizations!

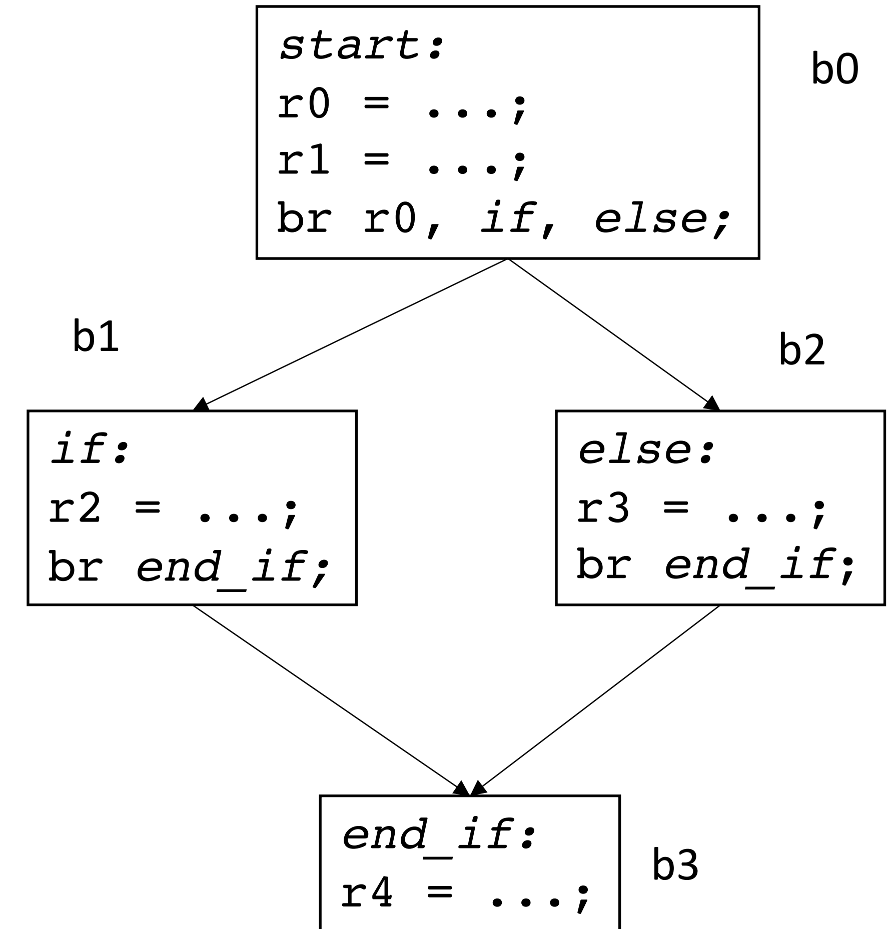
- Highly suggest reading for this part of the class
  - Chapter 9 of EAC

First concept:

- Dominance in a CFG
- Builds up a framework for reasoning
- Building block for many algorithms
  - global local value numbering when unlimited registers
  - Conversion to SSA

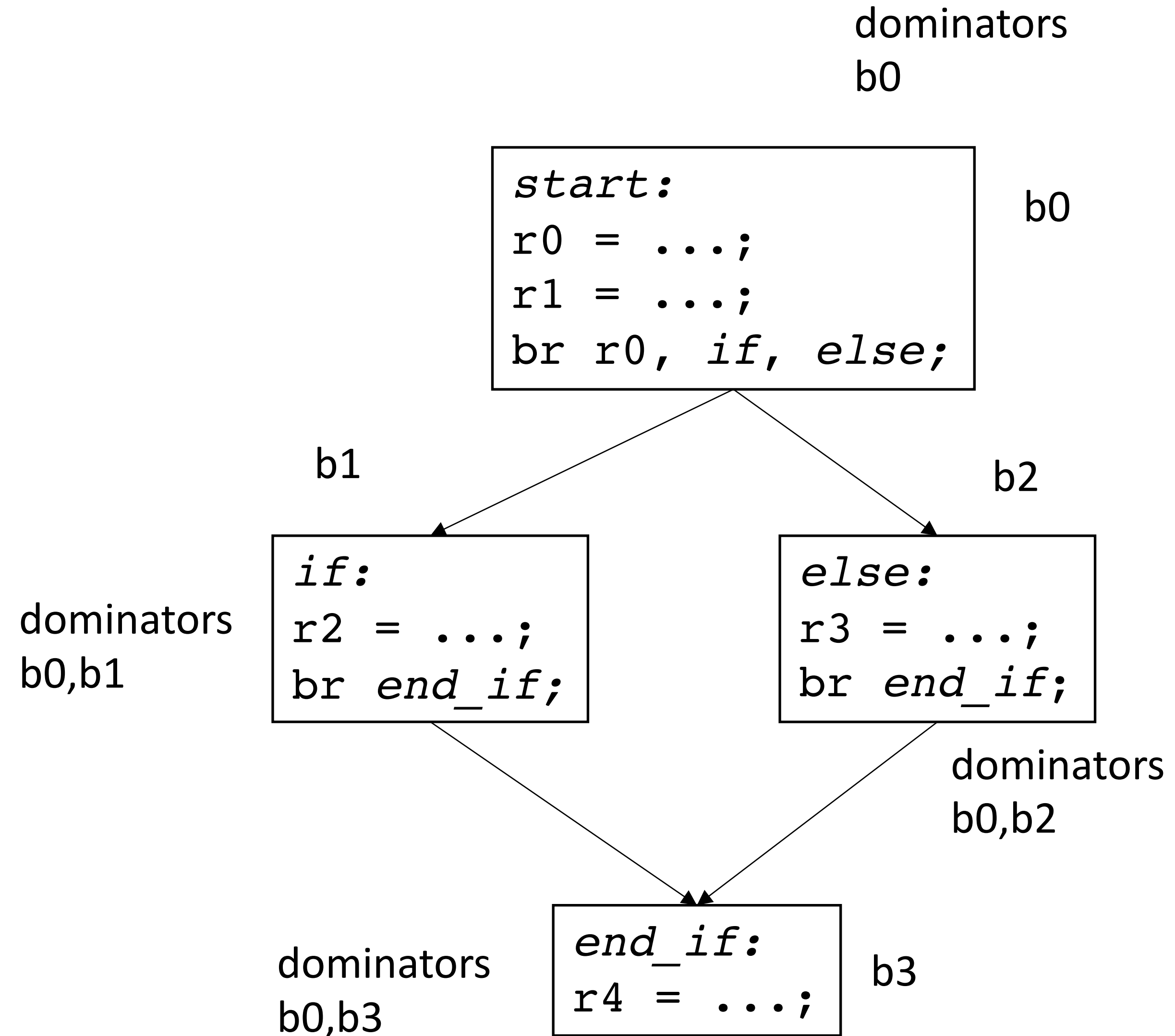
Dominance

- a block b_x dominates block b_y iff every path from the start to block b_x goes through b_y
- definition:
  - domination (includes itself)
  - strict domination (does not include itself)

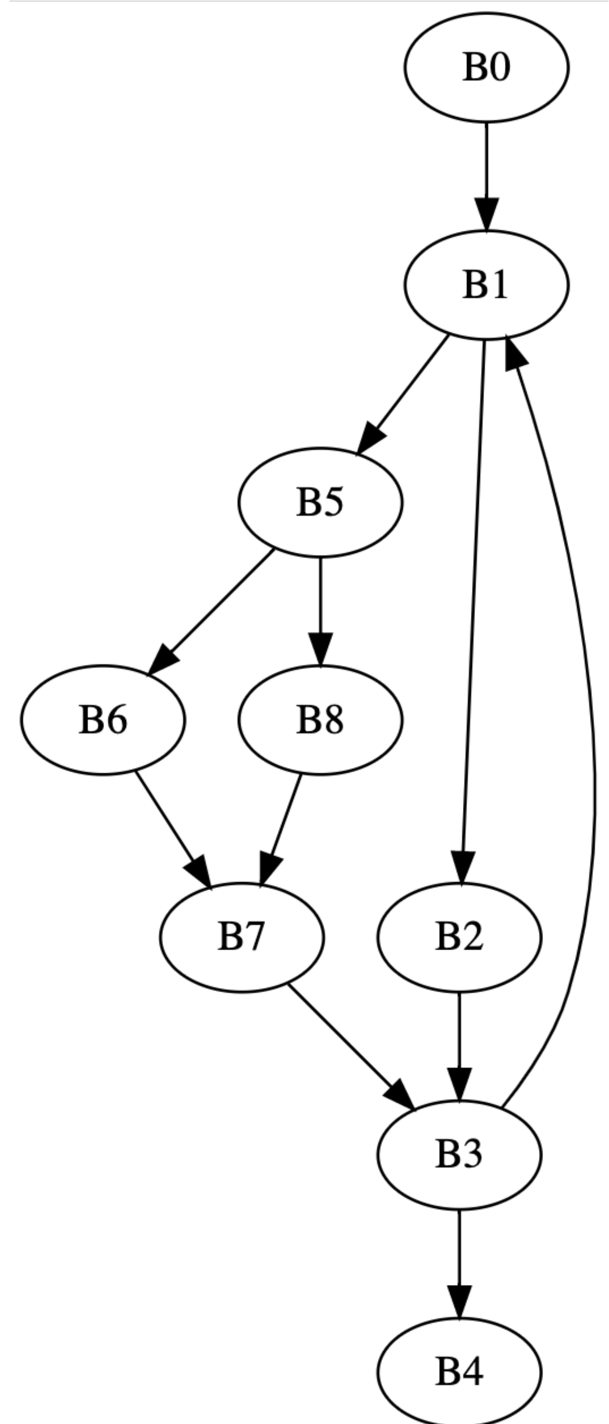


Dominance

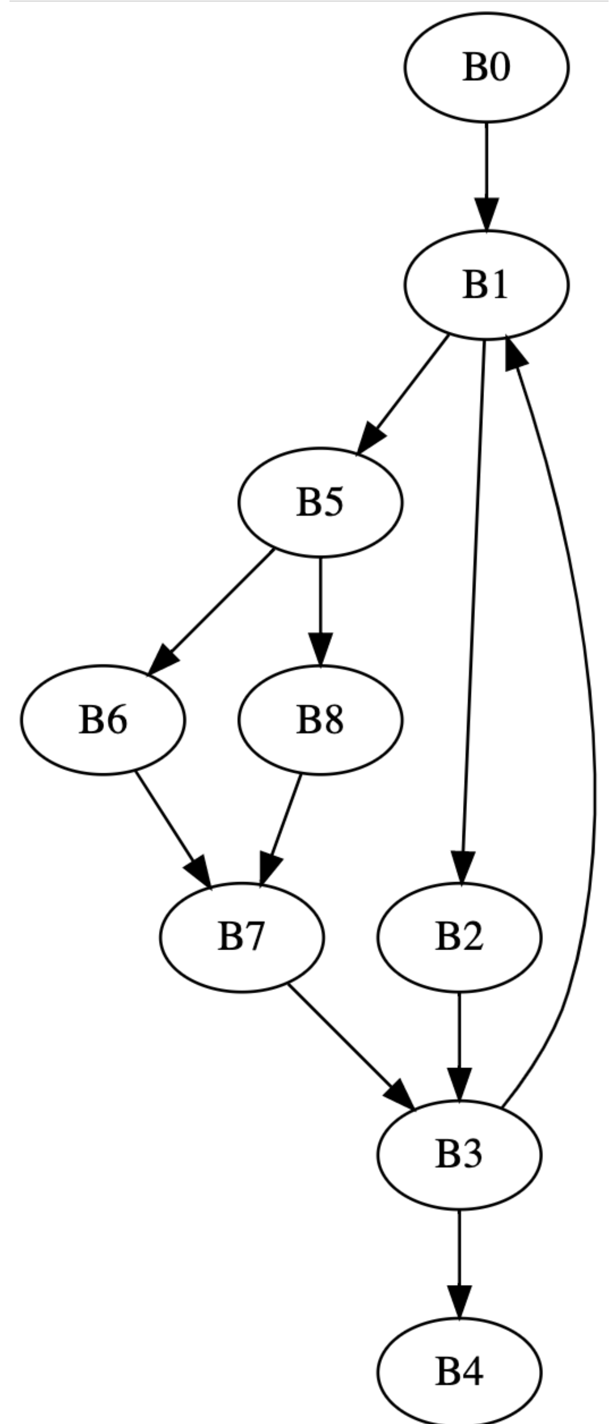
- a block b_x dominates block b_y iff every path from the start to block b_x goes through b_y
- definition:
  - domination (includes itself)
  - strict domination (does not include itself)
- Can we apply this to local value numbering?



Node	Dominators
B0	B0
B1	B0, B1
B2	B0, B1, B2
B3	B0, B1, B3
B4	B0,B1,B3,B4
B5	B0, B1,B5
B6	B0, B1, B5, B6
B7	B0, B1, B5, B7
B8	B0, B1, B5, B8

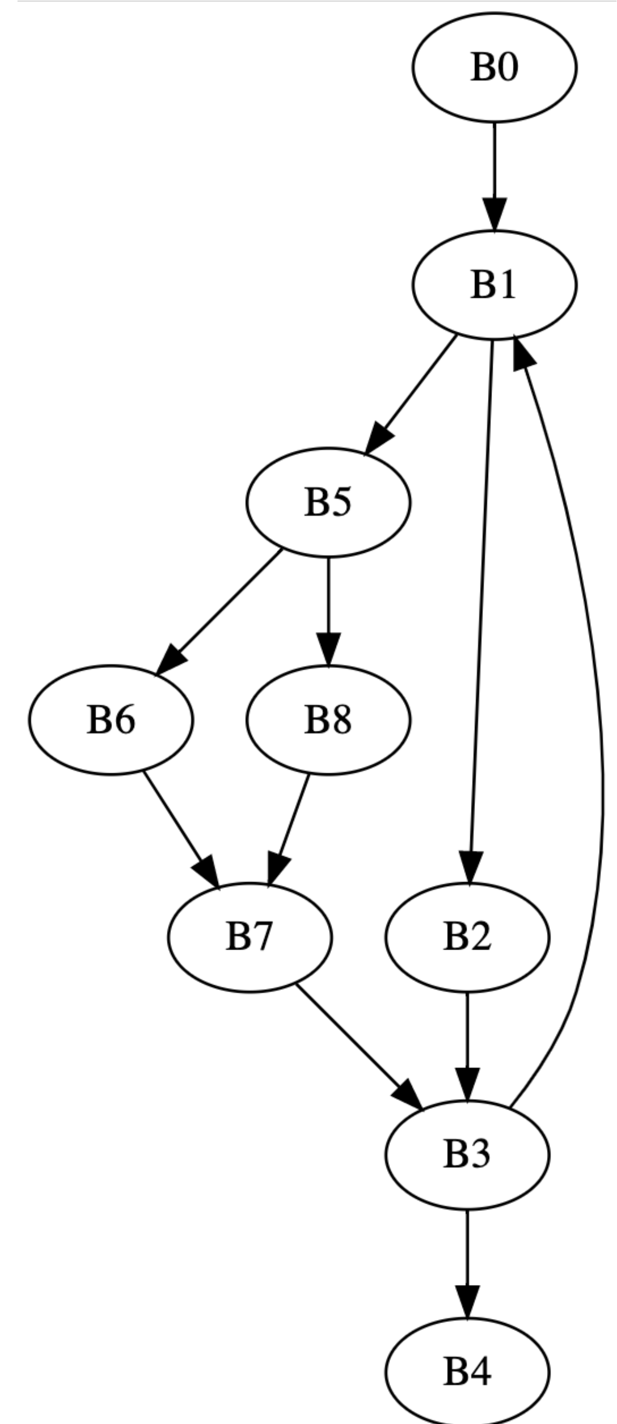


Node	Dominators
B0	B0
B1	B0, B1
B2	B0, B1, B2
B3	B0, B1, B3
B4	B0, B1, B3, B4
B5	B0, B1, B5
B6	B0, B1, B5, B6
B7	B0, B1, B5, B7
B8	B0, B1, B5, B8



Node	Dominators
B0	B0
B1	B0, B1
B2	B0, B1, B2
B3	B0, B1, B3
B4	B0, B1, B3, B4
B5	B0, B1, B5
B6	B0, B1, B5, B6
B7	B0, B1, B5, B7
B8	B0, B1, B5, B8

Concept introduced in 1959, algorithm not not given until 10 years later



Have a nice weekend!

- We will discuss other flow algorithms on Monday

- Remember:
  - Wednesday and Friday class next week is virtual
  - Homework due on Monday!