Loop Unrolling

Source:

```
for i := 1 to 100 by 1
  A[i] := A[i] + B[i];
endfor
```

Transformed Code:

```
for i := 1 to 100 by 4
  A[i  ] := A[i  ] + B[i  ];
  A[i+1] := A[i+1] + B[i+1];
  A[i+2] := A[i+2] + B[i+2];
  A[i+3] := A[i+3] + B[i+3];
endfor
```


Loop Unrolling

Source:

```
for i := 1 to 100 by 1
  A[i] := A[i] + B[i];
endfor
```

Transformed Code:

```
for i := 1 to 100 by 4
  A[i  ] := A[i  ] + B[i  ];
  A[i+1] := A[i+1] + B[i+1];
  A[i+2] := A[i+2] + B[i+2];
  A[i+3] := A[i+3] + B[i+3];
endfor
```


Benefits:

- The overhead of testing and branching is reduced.
- This optimization may "enable" other optimizations.

Loop Unrolling

Source:

```
for i := 1 to 100 by 1
  A[i] := A[i] + B[i];
endfor
```

Transformed Code:

```
for i := 1 to 100 by 4
  A[i  ] := A[i  ] + B[i  ];
  A[i+1] := A[i+1] + B[i+1];
  A[i+2] := A[i+2] + B[i+2];
  A[i+3] := A[i+3] + B[i+3];
endfor
```

Larger Basic Blocks are Good!
More opportunities for optimizations such as scheduling

Benefits:

- The overhead of testing and branching is reduced.
- This optimization may "enable" other optimizations.

Loop Unrolling

Source:

```
for i := 1 to MAX by 1
  A[i] := A[i] + B[i];
endfor
```

Number of iterations is not known at compile-time.

Transformed Code:

```
i := 1;
 while (i+3 <= MAX) do
  A[i  ] := A[i  ] + B[i  ];
  A[i+1] := A[i+1] + B[i+1];
  A[i+2] := A[i+2] + B[i+2];
  A[i+3] := A[i+3] + B[i+3];
  i := i + 4;
endwhile
while (i <= MAX) do
  A[i] := A[i] + B[i];
  i := i + 1;
endwhile
```

Do 0 to 3 more iterations, as necessary, to finish

Loop-Invariant Computations

An assignment

`x := y ⊕ z`

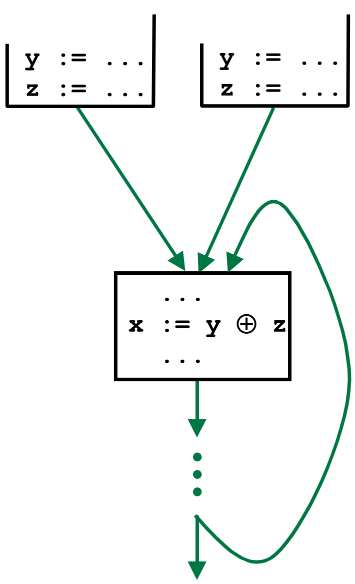
is "Loop-Invariant" if..

- It is in a loop, and
- All definitions of y and z that reach the statement are outside the loop.

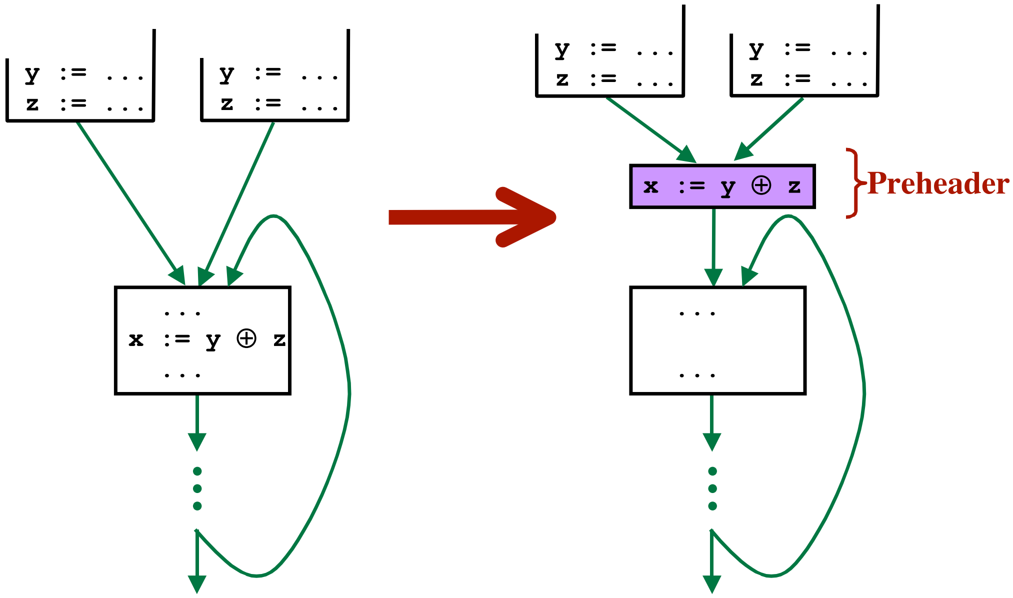
We may be able to move the computation into the "preheader".

Step 1: Detect the Loop-Invariant Computations.
Step 2: See if it is okay to move the statement into the pre-header.

Example



Example



Detecting Loop-Invariant Computations

Input:

Loop L (= a set of basic blocks)
U-D Chain information

Output:

The set of loop-invariant statements.

Idea:

- Mark some of the statements as "loop-invariant".
- This may allow us to mark even more statements as loop-invariant.
- Remember the order in which theses statements are marked.

Detecting Loop-Invariant Computations

```
repeat until no new statements are marked...
  Look at each statement in the loop.
  If all its operands are unchanging then
    mark the statement as "loop-invariant".
  An operand is "unchanging" if...
      • It is a constant
      • It has all reaching definitions
            outside of the loop
      • It has exactly one reaching definition
            and that definition has already
            been marked "loop-invariant".
end
```

Remember the order in which statements are marked "loop-invariant."

Moving Loop-Invariant Computations

Consider moving statement

S: `x := y ⊕ z`

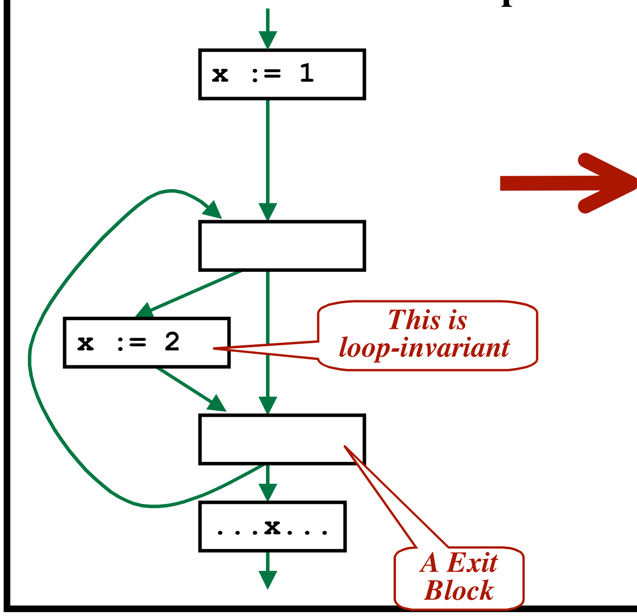
into the loop's preheader.

The statement must satisfy three conditions.

If it satisfies all conditions, then it can be moved.

Condition 1

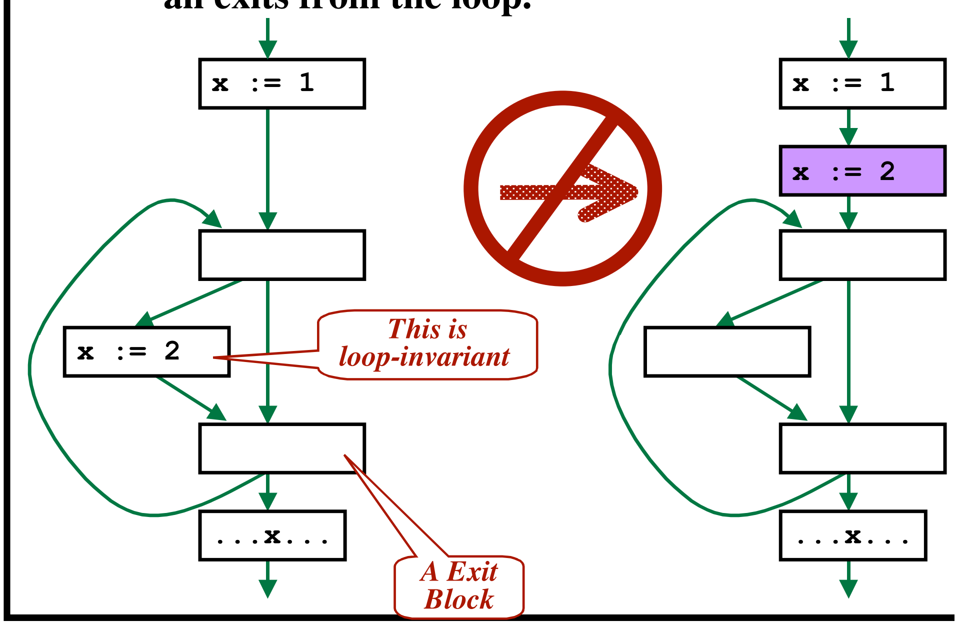
The block containing S must dominate all exits from the loop.



CS-322 Optimization, Part 4

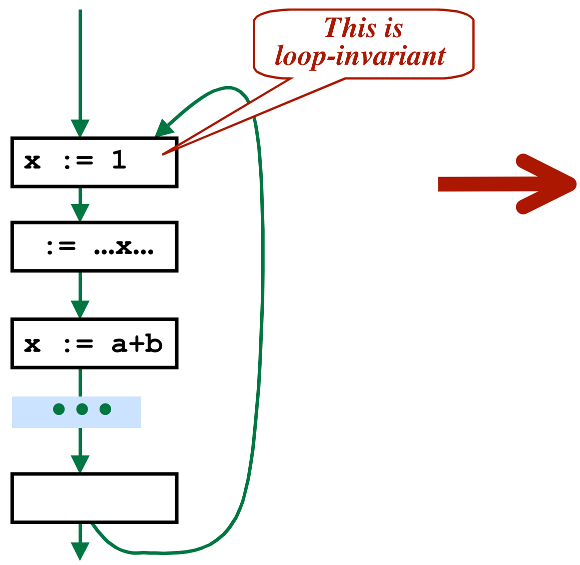
Condition 1

The block containing S must dominate all exits from the loop.



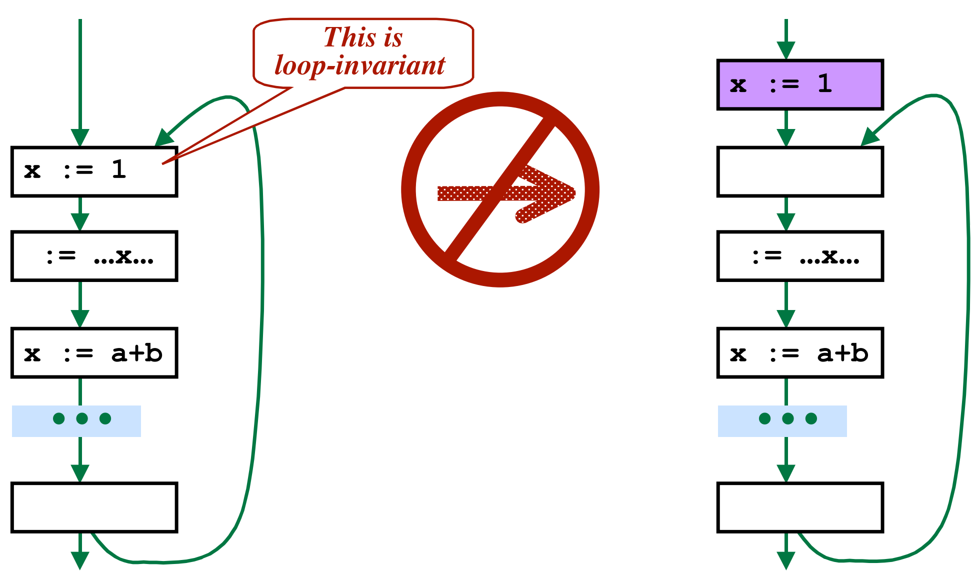
Condition 2

There must be no other assignments to "x" in the loop.



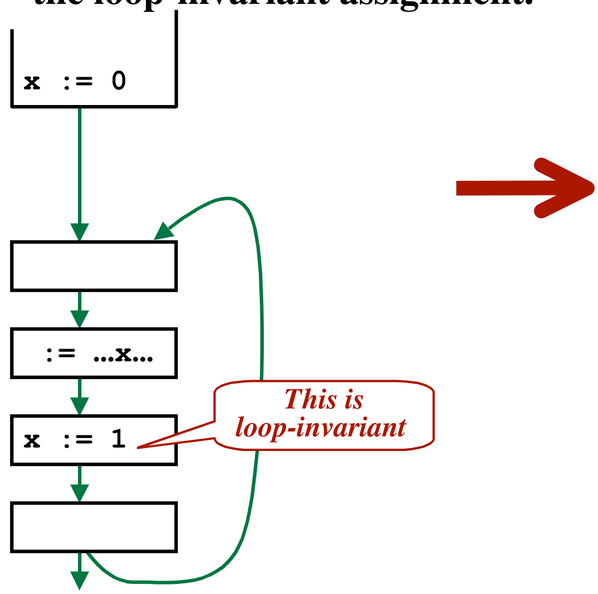
Condition 2

There must be no other assignments to "x" in the loop.



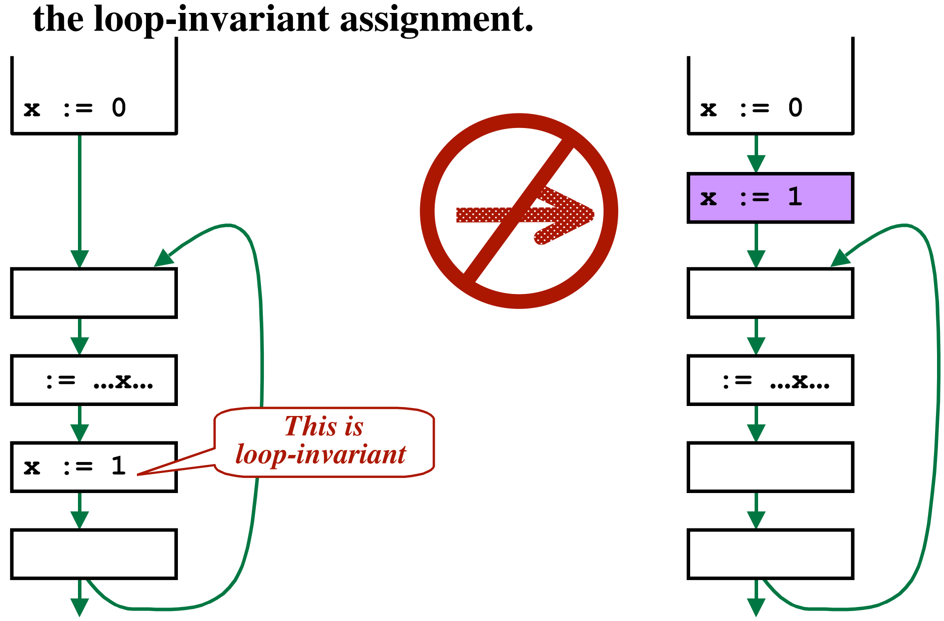
Condition 3

All uses of "x" in the loop must be reached by ONLY the loop-invariant assignment.

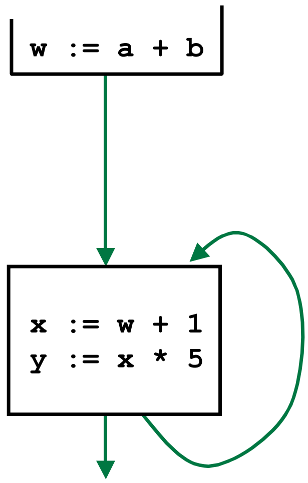


Condition 3

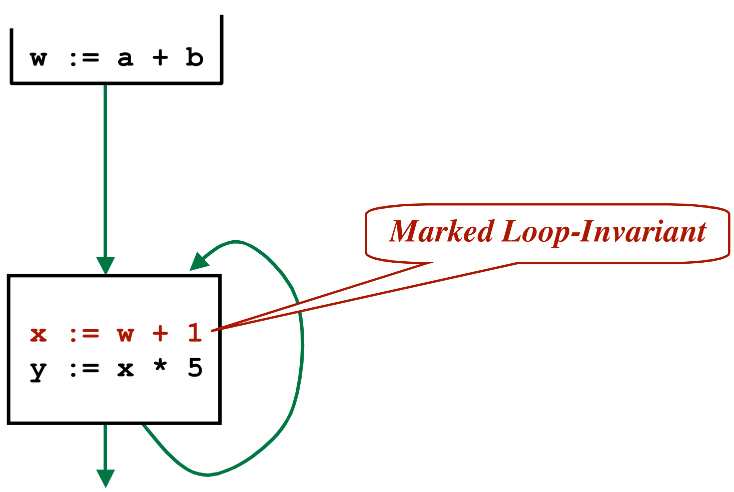
All uses of "x" in the loop must be reached by ONLY the loop-invariant assignment.



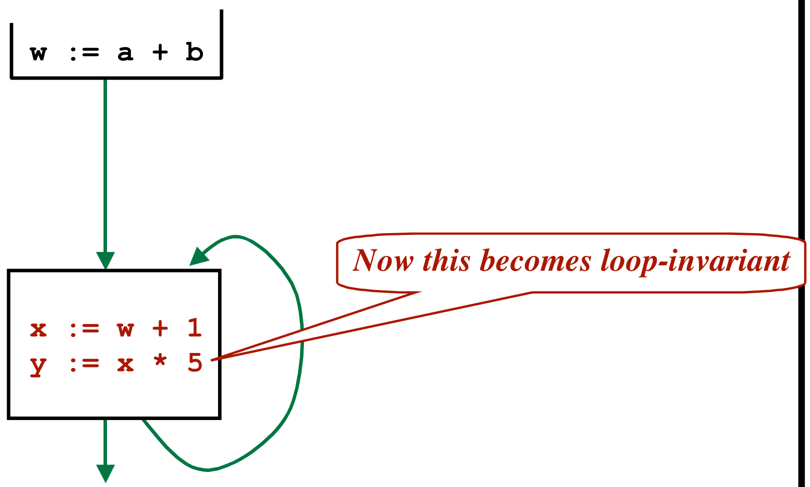
If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.



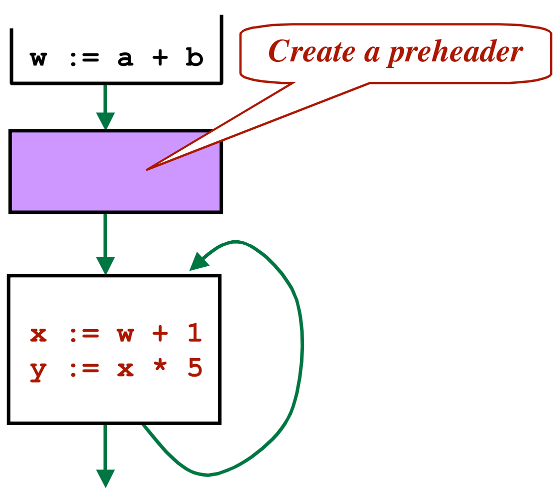
If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.



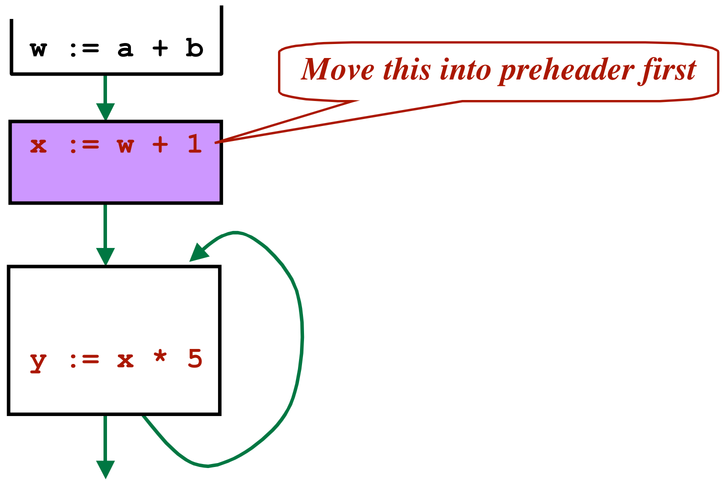
If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.



If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.



**If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.**



**If all three conditions are satisfied,
move the statements into the preheader
in the order they were marked Loop-Invariant.**