Lecture #32

Code Optimization

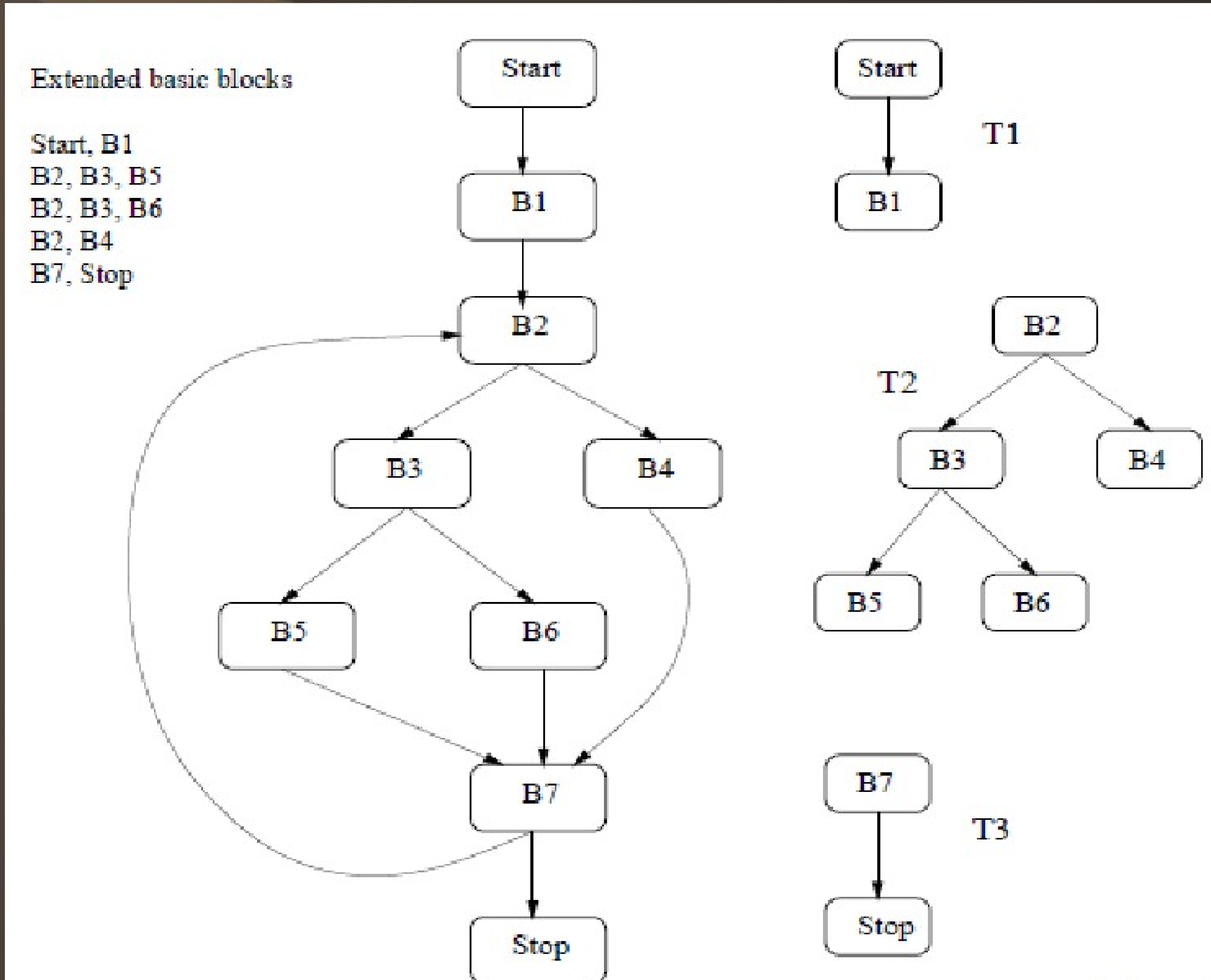
Extended Basic Blocks

- A sequence of basic blocks $B_1, B_2, \ldots B_k$, such that B_i is the unique predecessor of $B_{i+1} (1 \leq i < k)$, and B_1 is either the start block or has no unique predecessor

- Extended basic blocks with shared blocks can be represented as a tree
- Shared blocks in extended basic blocks require scoped versions of tables

- The new entries must be purged and changed entries must be replaced by old entries
- Preorder traversal of extended basic block trees is used

Extended Basic Blocks



Extended Basic Blocks

```
function visit-ebb-tree(e) // e is a node in the tree
begin
   // From now on, the new names will be entered with a new scope into the tables.
   // When searching the tables, we always search beginning with the current scope
   // and move to enclosing scopes. This is similar to the processing involved with
   // symbol tables for lexically scoped languages
   value-number(e.B);
   // Process the block e.B using the basic block version of the algorithm
   if (e.left ≠ null) then visit-ebb-tree(e.left);
   if (e.right ≠ null) then visit-ebb-tree(e.right);
   remove entries for the new scope from all the tables
   and undo the changes in the tables of enclosing scopes;
end

begin // main calling loop
   for each tree t do visit-ebb-tree(t);
   // t is a tree representing an extended basic block
end
```

Peephole Optimization

- A machine dependent optimization technique
- Peephole optimizations are replacement rules of the form:

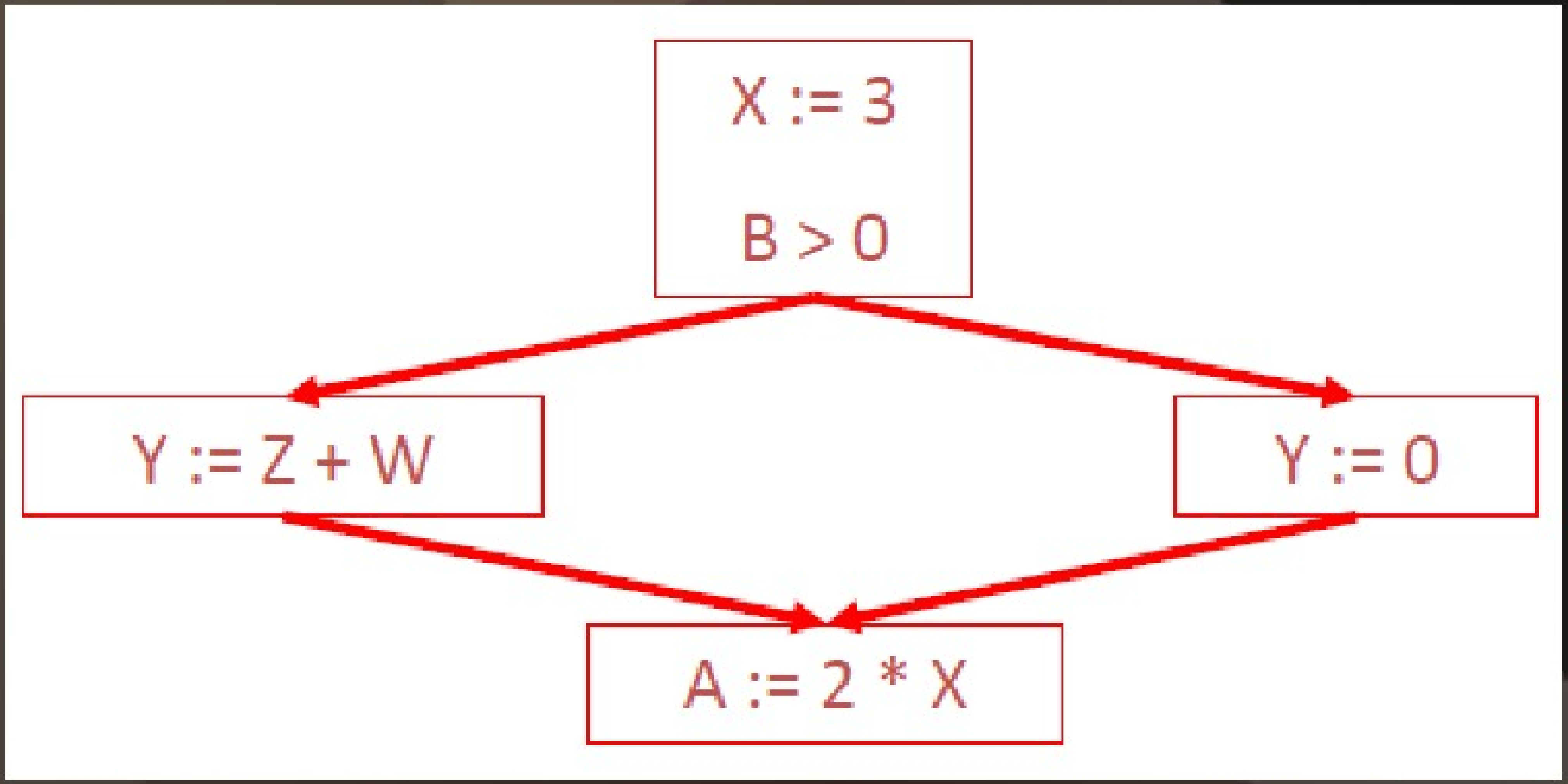
$$i_1, \ldots, i_n \rightarrow j_1, \ldots, j_m$$

where the RHS is the improved version of the LHS

- Example:
  - move \$a \$b, move \$b \$a → move \$a \$b
  - Works if move \$b \$a is not the target of a jump
- Another example
  - addiu \$a \$a i, addiu \$a \$a j → addiu \$a \$a i+j

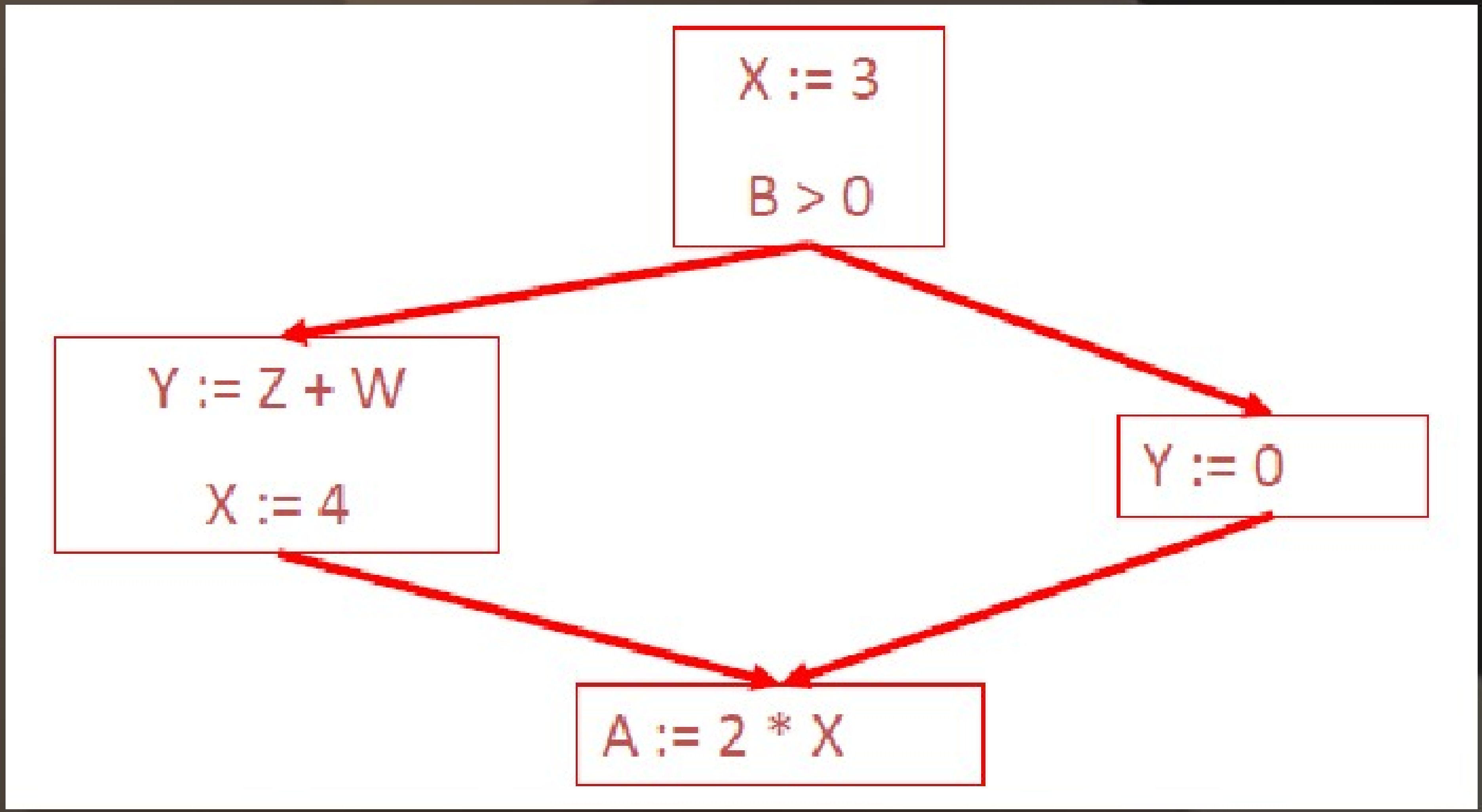
Dataflow Analysis

- Simple optimizations over a basic block may be extended over the entire CFG
- Example: Global Constant Propagation



Dataflow Analysis

- To replace a use of **x** by a constant **k** we must know:
  - On every path to the use of **x**, the last assignment to **x** is **x = k**



Global Optimization

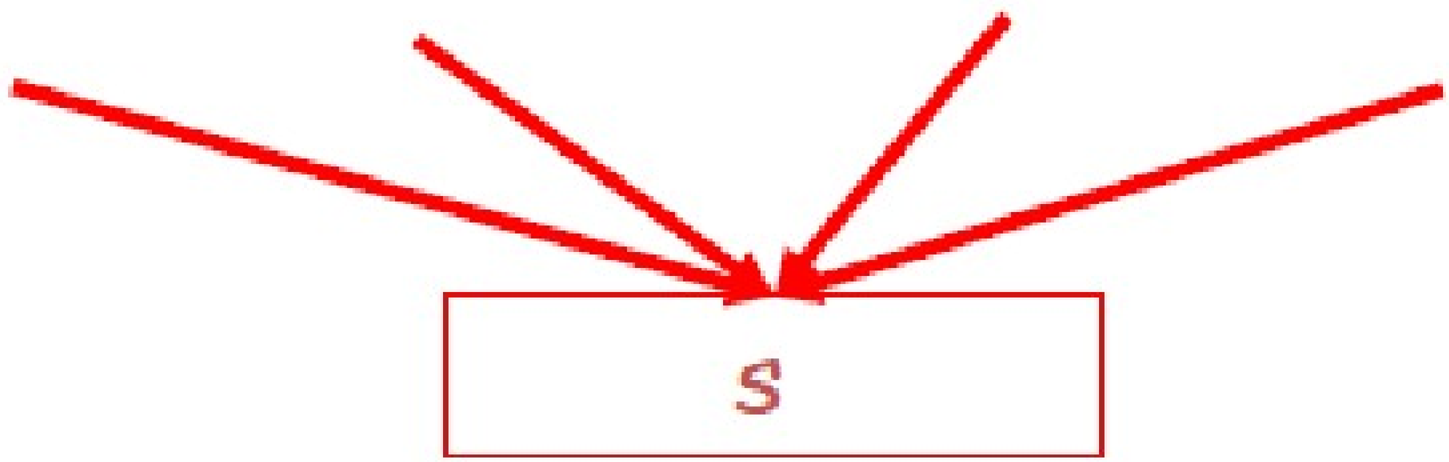
- The correctness condition is not trivial to check
  - *Paths* include paths around loops and through branches of conditionals
- Generally global optimization depends on knowing a property **X** at a particular point in program execution
  - Proving **X** at any point requires knowledge of the entire program
- It is OK to be conservative. If the optimization requires **X** to be true, then want to know either
  - **X** is definitely true
  - Don't know if **X** is true
    - It is always safe to say "don't know"

Global Constant Propagation

- We must know whether:
  - On every path to the use of **x**, the last assignment to **x** is **x = k**
- We associate one of the following values with **x** at every program point:
  - $\perp$ (Bottom) : This statement never executes
  - C : **x** equals to constant **C**
  - $\top$ (Top) : **x** is not a constant
- For each statement **s**, the value of **x** immediately before and after **x** is calculated
  - C(s, x, in): Value of **x** before **s**
  - C(s, x, out): Value of **x** after **s**

Global Constant Propagation

- Rule #1

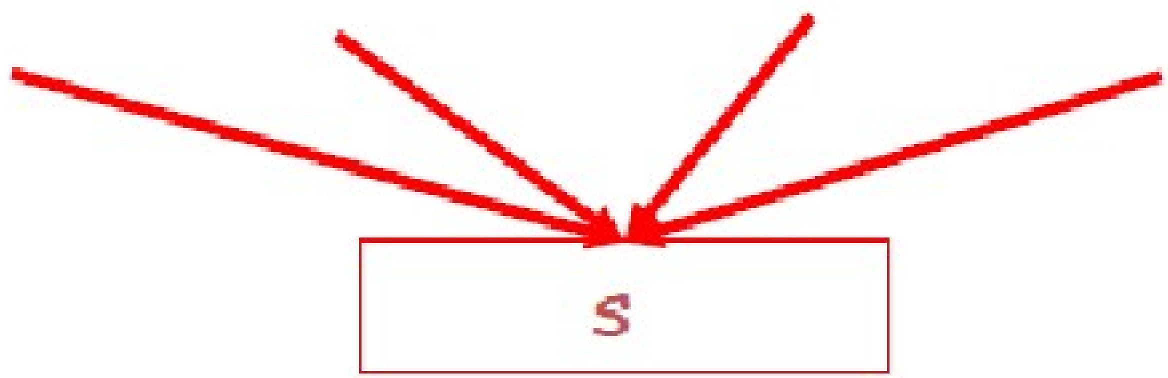


if $C(p_i, x, out) = \top$

for any i, then $C(s, x, in) = \top$

Global Constant Propagation

- Rule #2

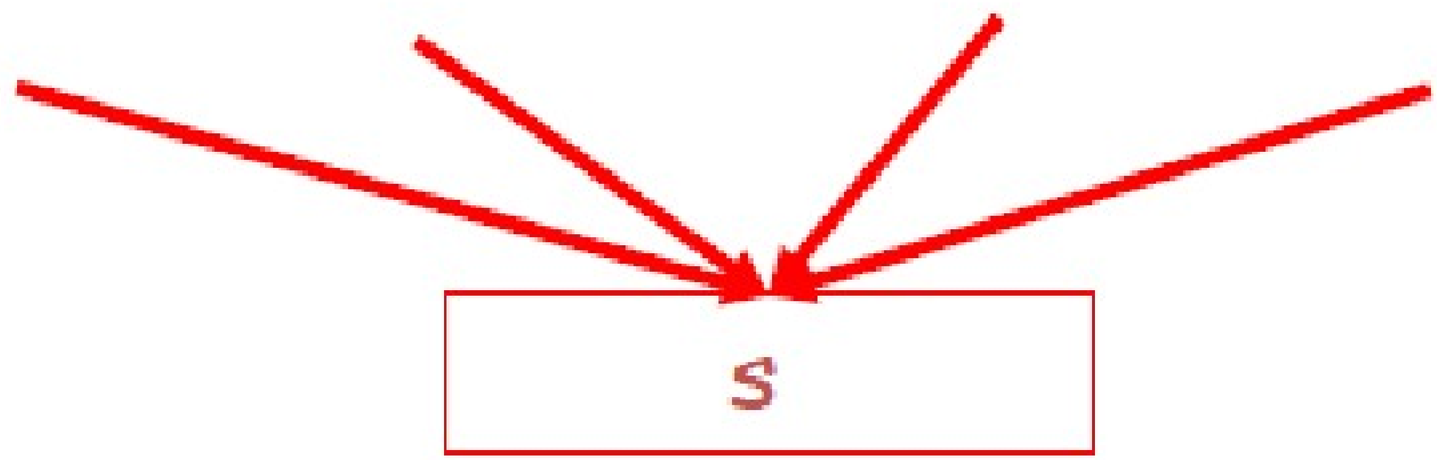


if $C(p_i, x, out) = c$ & $C(p_j, x, out) = d$ & $d <> c$

then $C(s, x, in) = \top$

Global Constant Propagation

- Rule #3

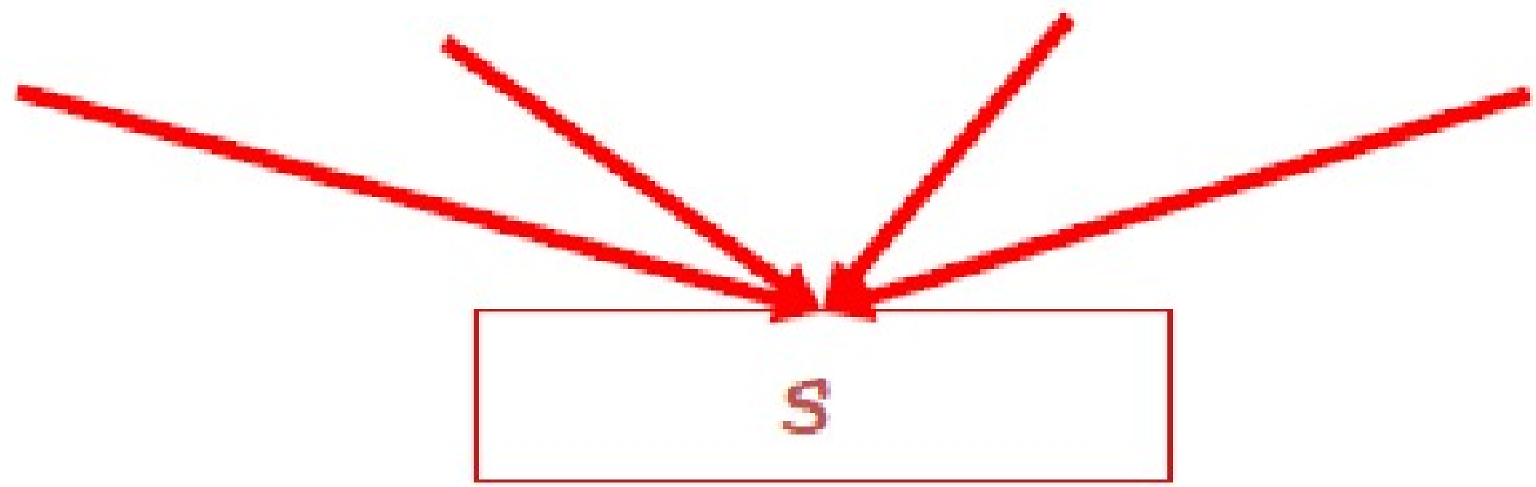


if $C(p_i, x, out) = c$ or $\perp$ for all i,

then $C(s, x, in) = c$

Global Constant Propagation

- Rule #4

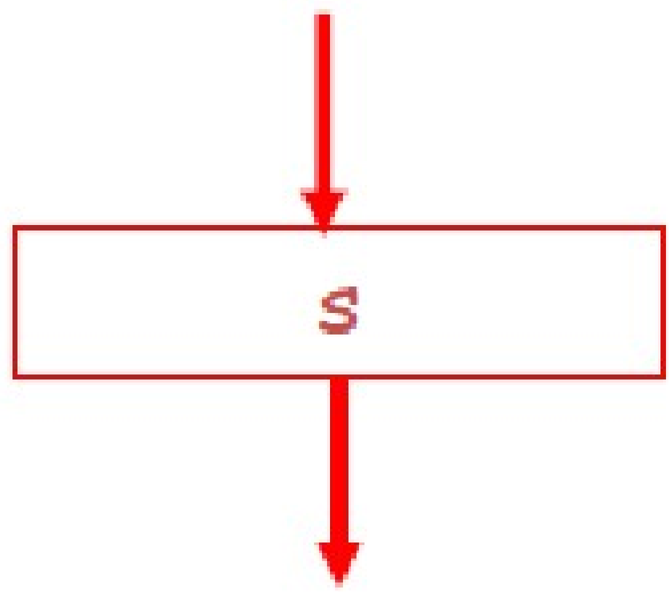


if $C(p_i, x, out) = \perp$ for all i,

then $C(s, x, in) = \perp$

Global Constant Propagation

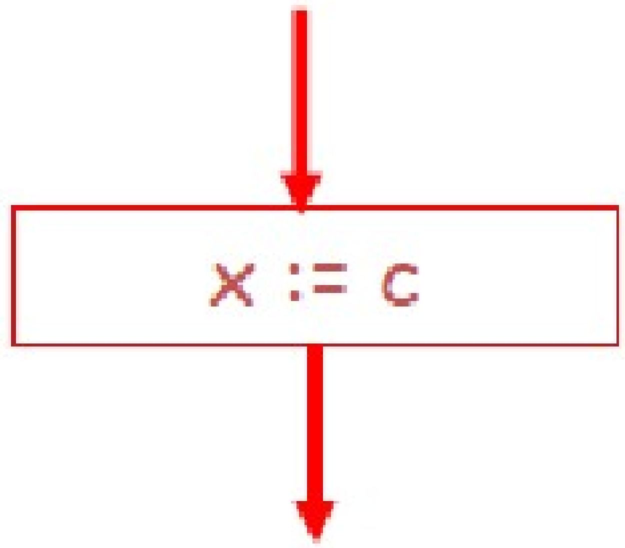
- Rule #5



$$C(s, x, out) = \perp \text{ if } C(s, x, in) = \perp$$

Global Constant Propagation

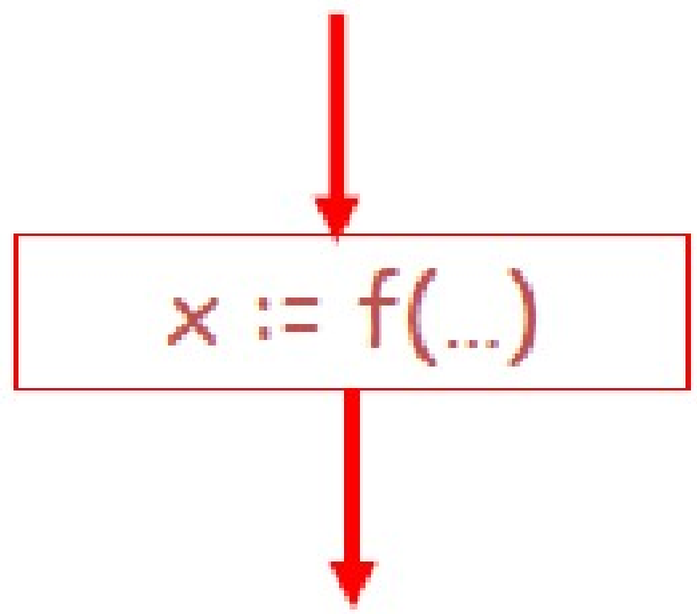
- Rule #6



$C(x := c, x, out) = c$ if c is a constant

Global Constant Propagation

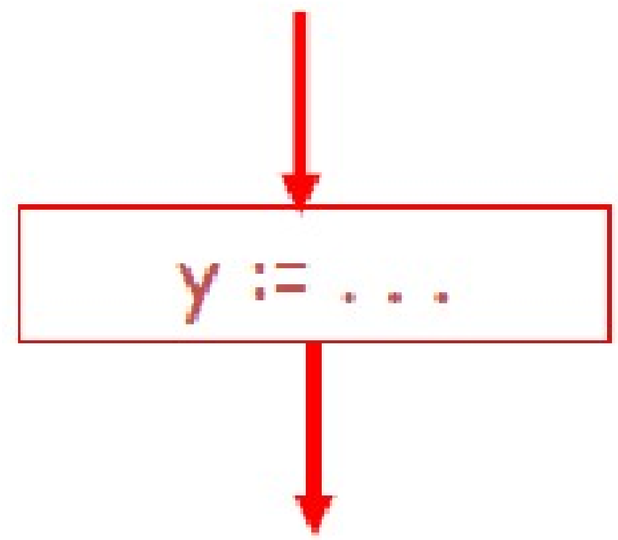
- Rule #7



$$C(x := f(\ldots), x, out) = \top$$

Global Constant Propagation

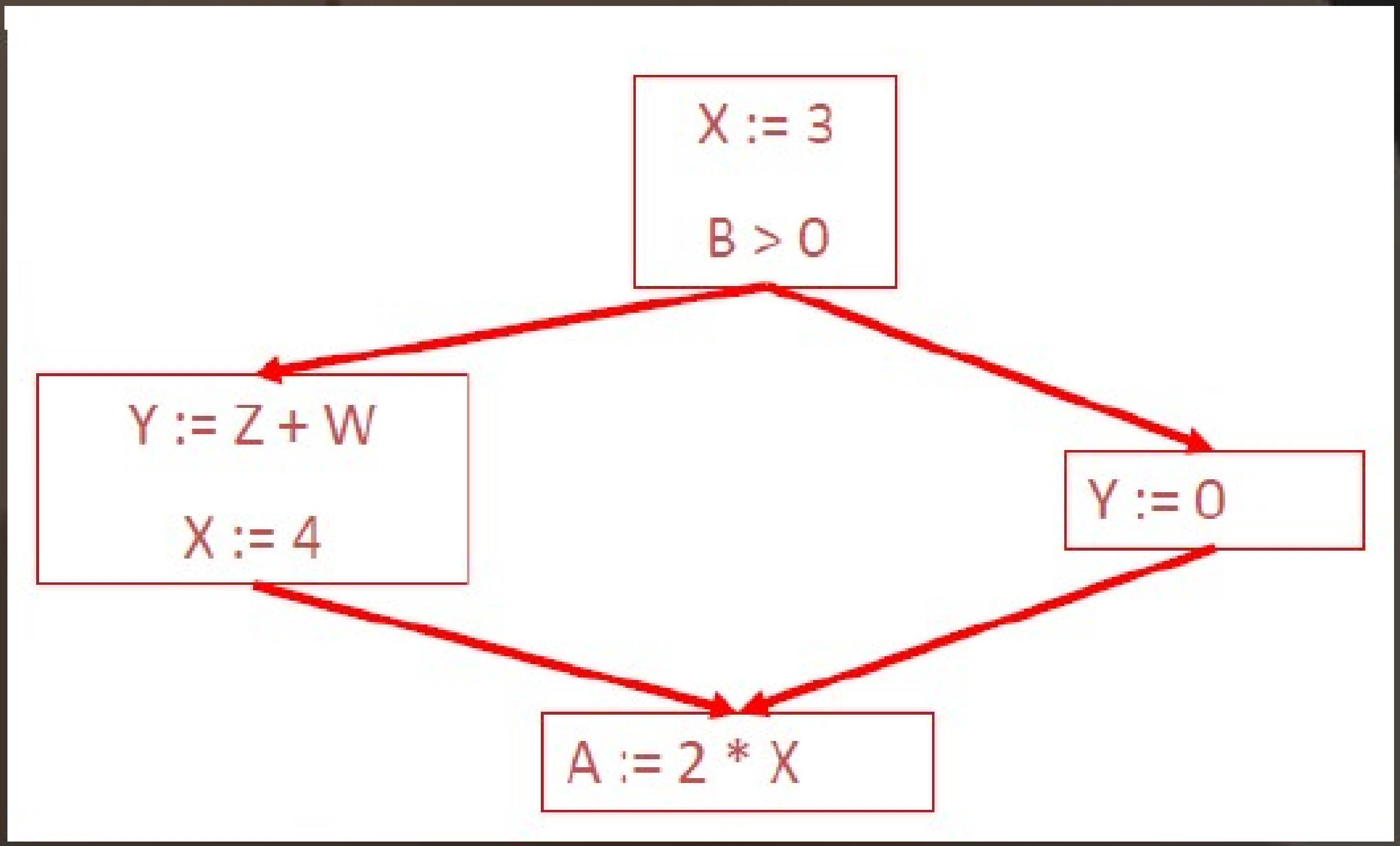
- Rule #8



C(y := …, x, out) = C(y := …, x, in) if x <> y

Global Constant Propagation

- For every entry s to the program, set C(s, x, in) = $\top$
- Set C(s, x, in) = C(s, x, out) = $\perp$ everywhere else
- Repeat until all points satisfy 1-8:
  - Pick s not satisfying 1-8 and update using the appropriate rule



Analysis of Loops

- How can global constant propagation for the following CFG be performed?