

# Code Generation

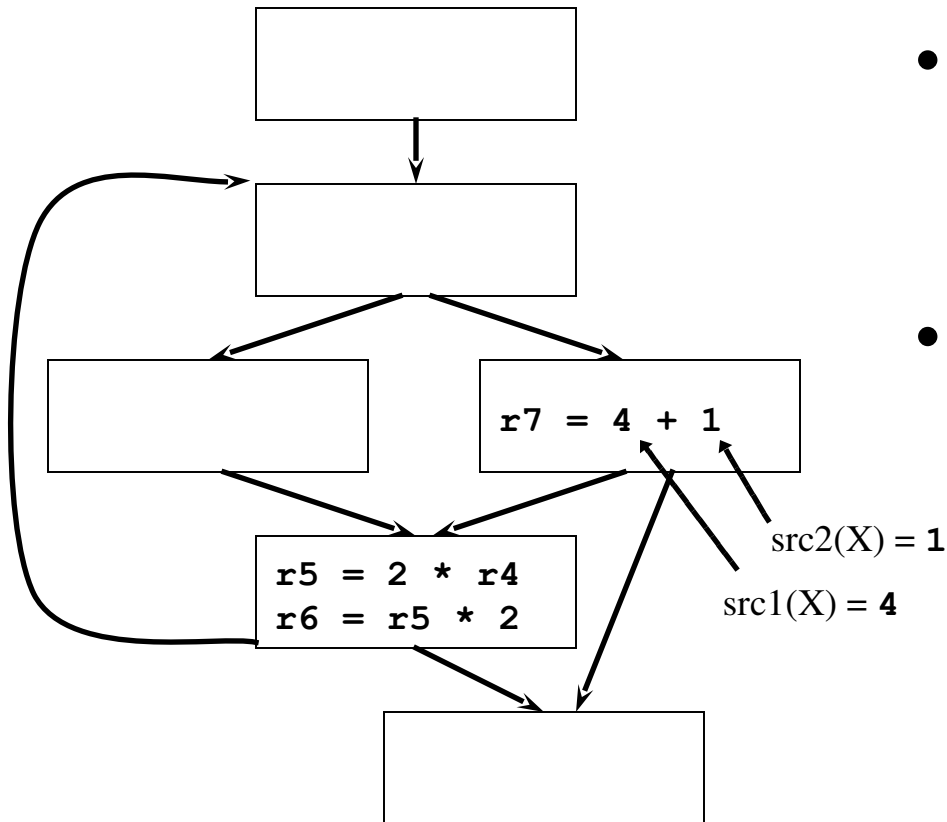
## Part III

### Chapter 9+10

# Classic Examples of Local and Global Code Optimizations

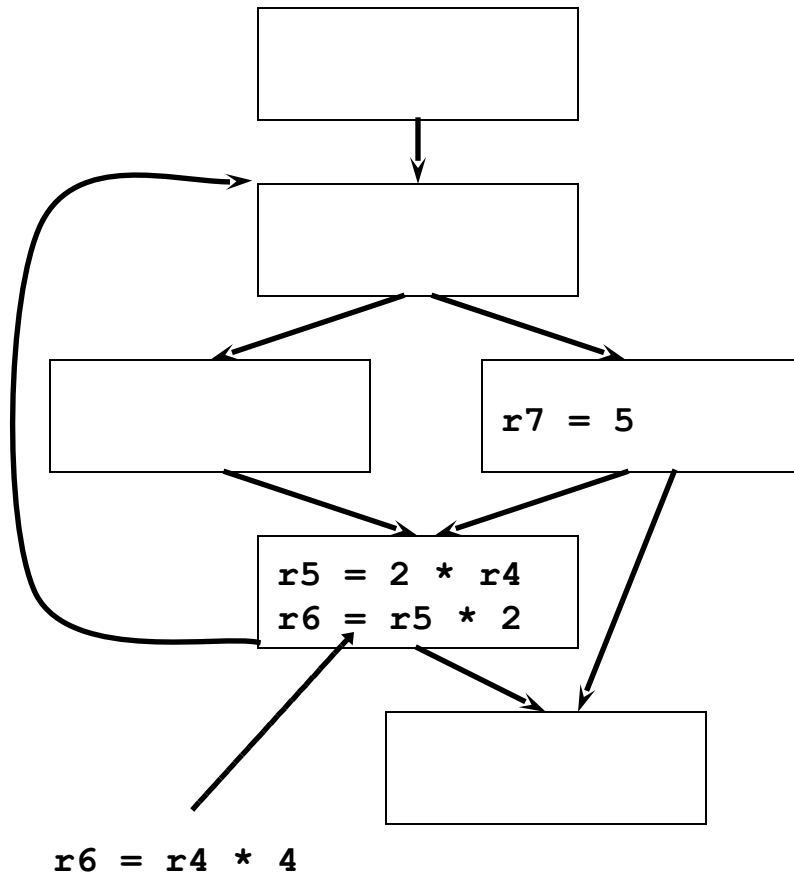
- Local
  - Constant folding
  - Constant combining
  - Strength reduction
  - Constant propagation
  - Common subexpression elimination
  - Backward copy propagation
- Global
  - Dead code elimination
  - Constant propagation
  - Forward copy propagation
  - Common subexpression elimination
  - Code motion
  - Loop strength reduction
  - Induction variable elimination

# Local: Constant Folding



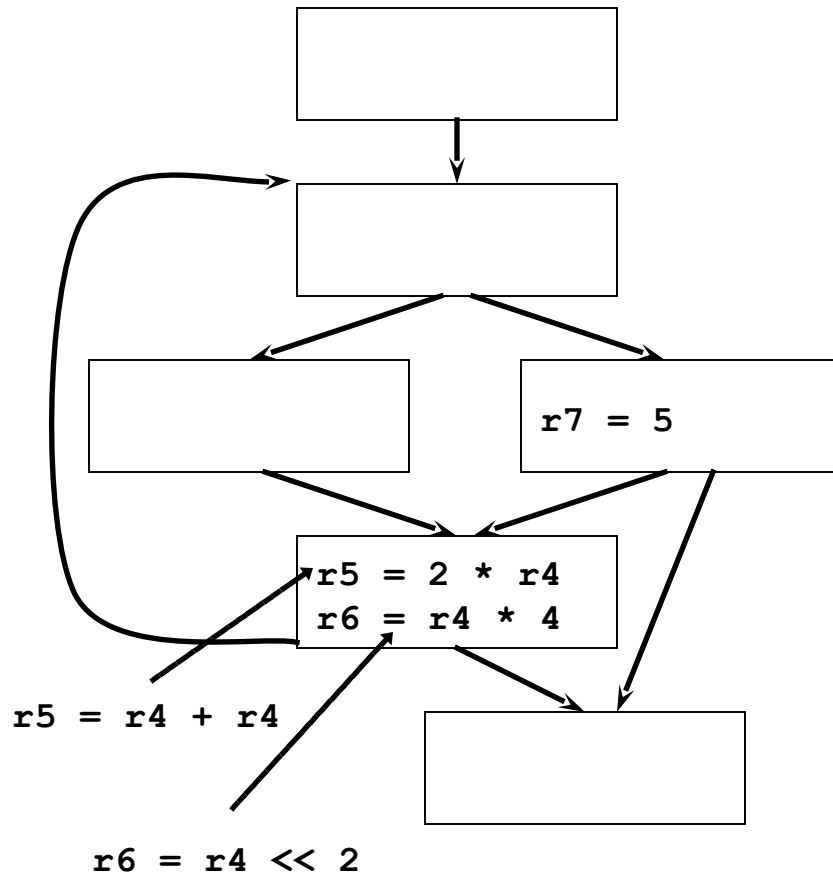
- Goal: eliminate unnecessary operations
- Rules:
  1.  $X$  is an arithmetic operation
  2. If  $src1(X)$  and  $src2(X)$  are constant, then change  $X$  by applying the operation

# Local: Constant Combining



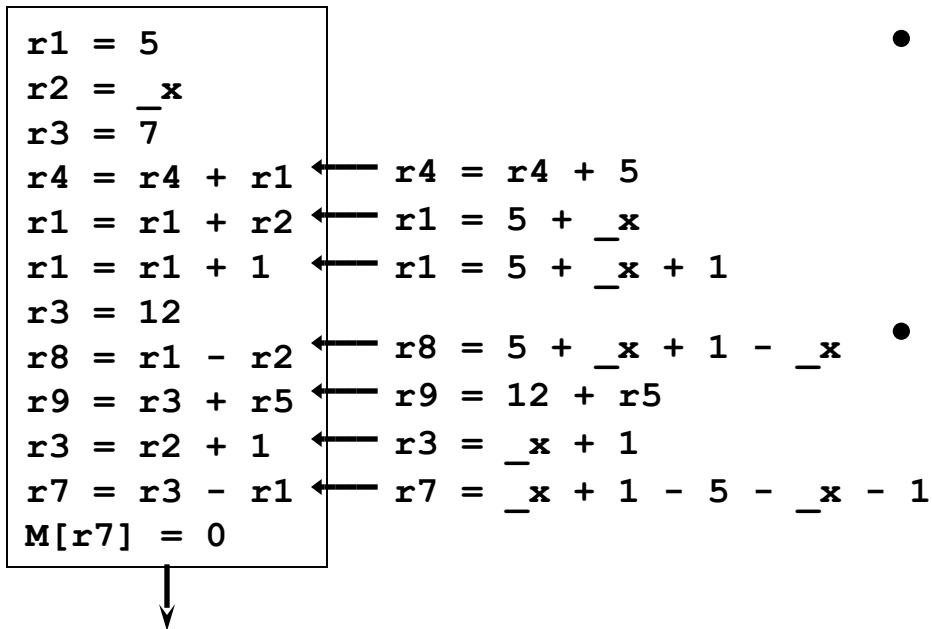
- Goal: eliminate unnecessary operations
  - First operation often becomes dead after constant combining
- Rules:
  1. Operations X and Y in same basic block
  2. X and Y have at least one literal src
  3. Y uses dest(X)
  4. None of the srcs of X have defs between X and Y (excluding Y)

# Local: Strength Reduction



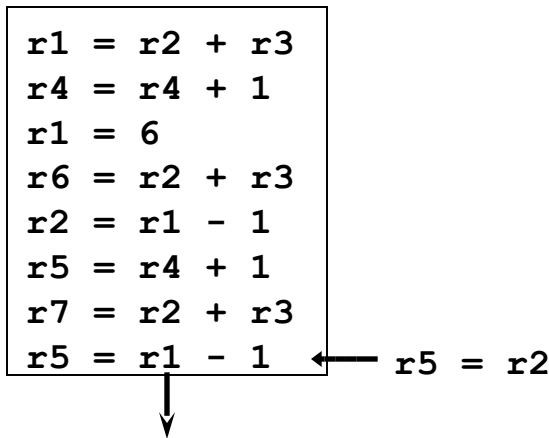
- Goal: replace expensive operations with cheaper ones
- Rules (common):
  1. X is an multiplication operation where  $\text{src1}(X)$  or  $\text{src2}(X)$  is a const  $2^k$  integer literal
  2. Change X by using shift operation
  3. For  $k=1$  can use add

# Local: Constant Propagation



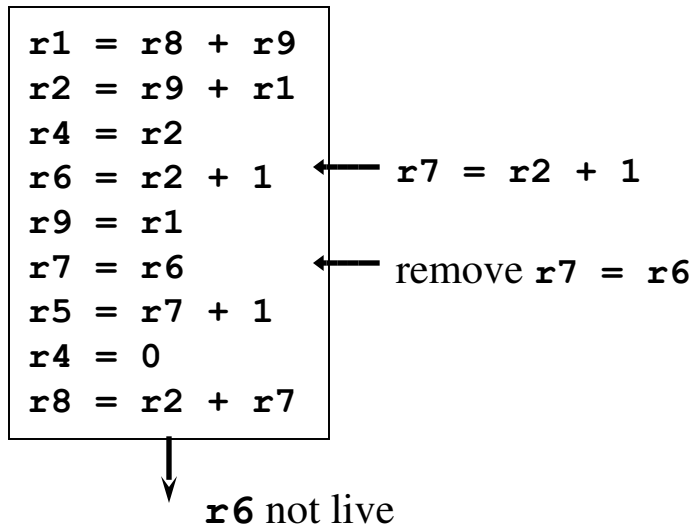
- Goal: replace register uses with literals (constants) in a single basic block
- Rules:
  1. Operation X is a move to register with src1(X) literal
  2. Operation Y uses dest(X)
  3. There is no def of dest(X) between X and Y (excluding defs at X and Y)
  4. Replace dest(X) in Y with src1(X)

# Local: Common Subexpression Elimination (CSE)



- Goal: eliminate re-computations of an expression
  - More efficient code
  - Resulting moves can get copy propagated (see later)
- Rules:
  1. Operations X and Y have the same opcode and Y follows X
  2.  $\text{src}(X) = \text{src}(Y)$  for all srcs
  3. For all srcs, no def of a src between X and Y (excluding Y)
  4. No def of  $\text{dest}(X)$  between X and Y (excluding X and Y)
  5. Replace Y with move  $\text{dest}(Y) = \text{dest}(X)$

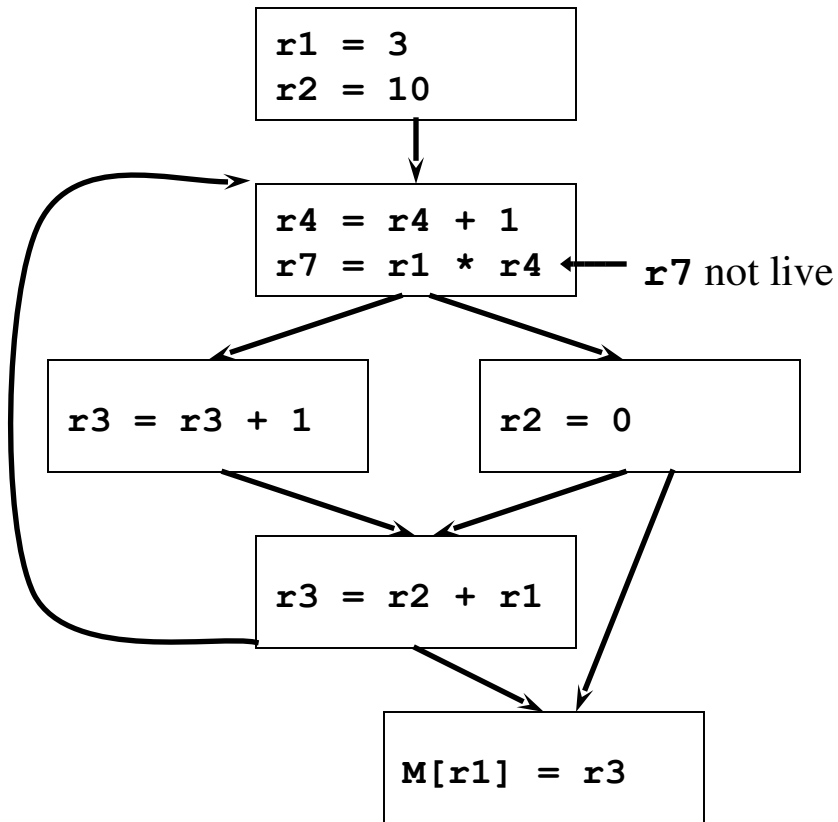
# Local: Backward Copy Propagation



- Goal: propagate LHS of moves backward
  - Eliminates useless moves
- Rules (dataflow required)
  1. X and Y in same block
  2. Y is a move to register
  3.  $\text{dest}(X)$  is a register that is not live out of the block
  4. Y uses  $\text{dest}(X)$
  5.  $\text{dest}(Y)$  not used or defined between X and Y (excluding X and Y)
  6. No uses of  $\text{dest}(X)$  after the first redef of  $\text{dest}(Y)$
  7. Replace  $\text{src}(Y)$  on path from X to Y with  $\text{dest}(X)$  and remove Y

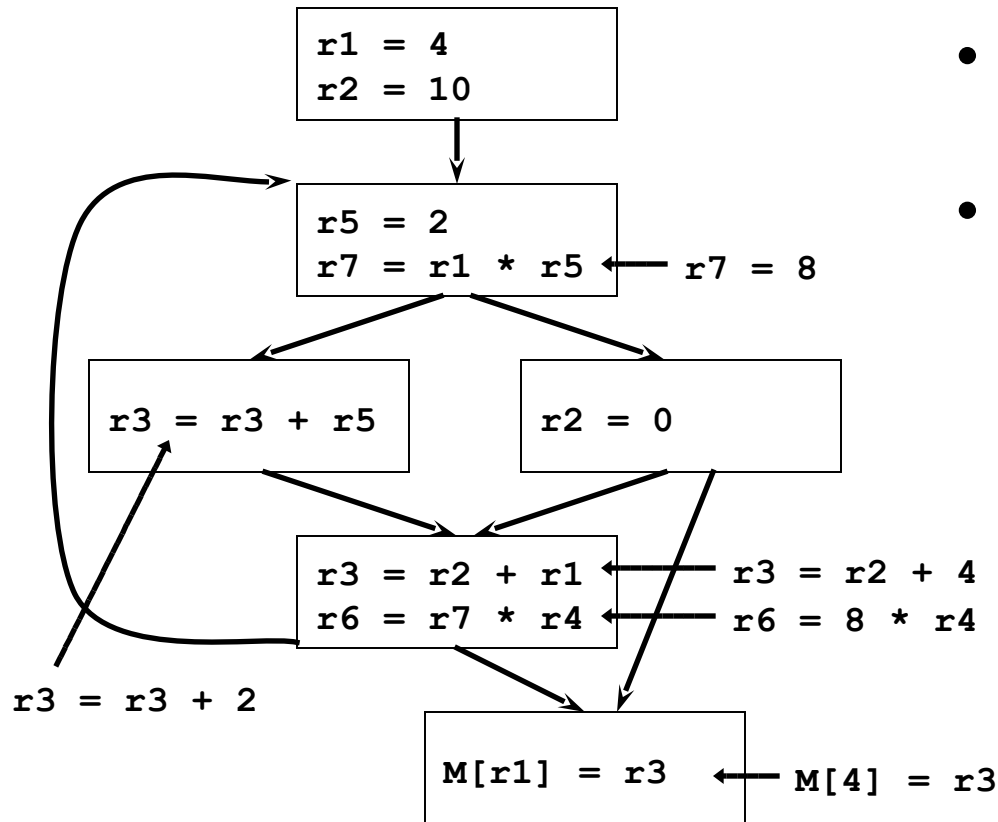


# Global: Dead Code Elimination



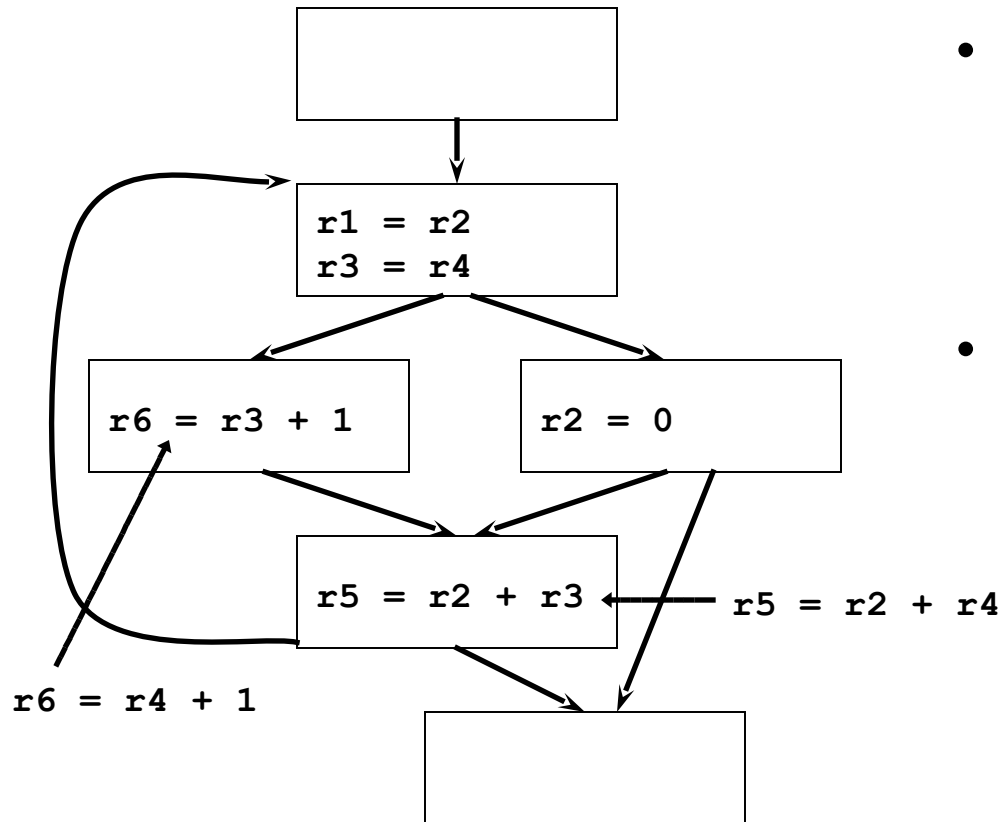
- Goal: eliminate any operation whose result is never used
- Rules (dataflow required)
  1. `X` is an operation with no use in def-use (DU) chain, i.e. `dest(X)` is not live
  2. Delete `X` if removable (not a mem store or branch)
- Rules too simple!
  - Misses deletion of `r4`, even after deleting `r7`, since `r4` is live in loop
  - Better is to trace UD chains backwards from “critical” operations

# Global: Constant Propagation



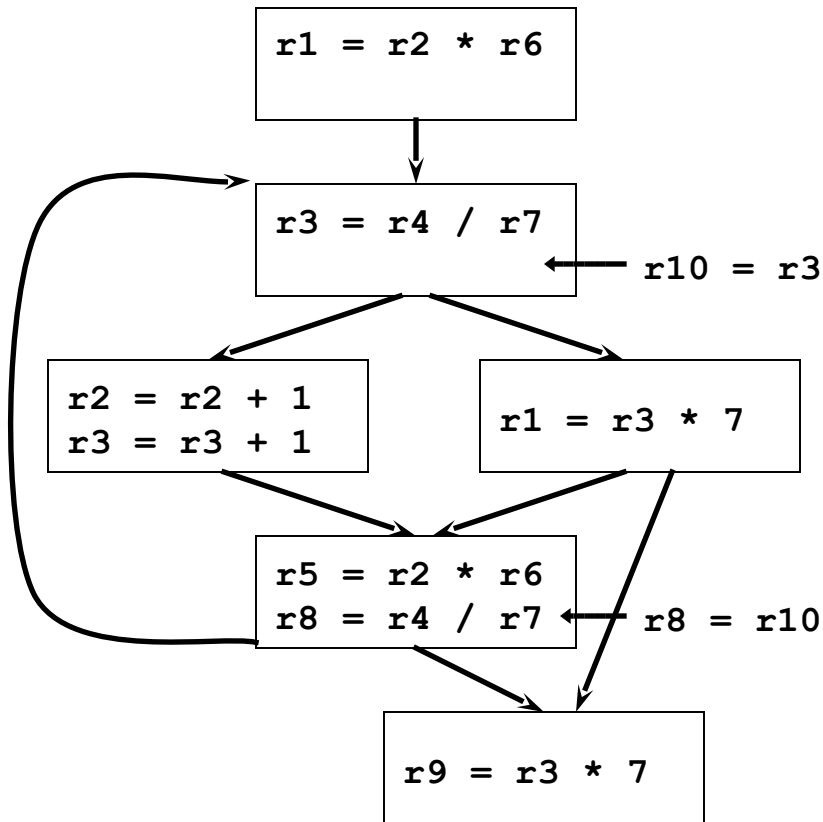
- Goal: globally replace register uses with literals
- Rules (dataflow required)
  1. X is a move to a register with  $\text{src1}(X)$  literal
  2. Y uses  $\text{dest}(X)$
  3.  $\text{dest}(X)$  has only one def at X for use-def (UD) chains to Y
  4. Replace  $\text{dest}(X)$  in Y with  $\text{src1}(X)$

# Global: Forward Copy Propagation



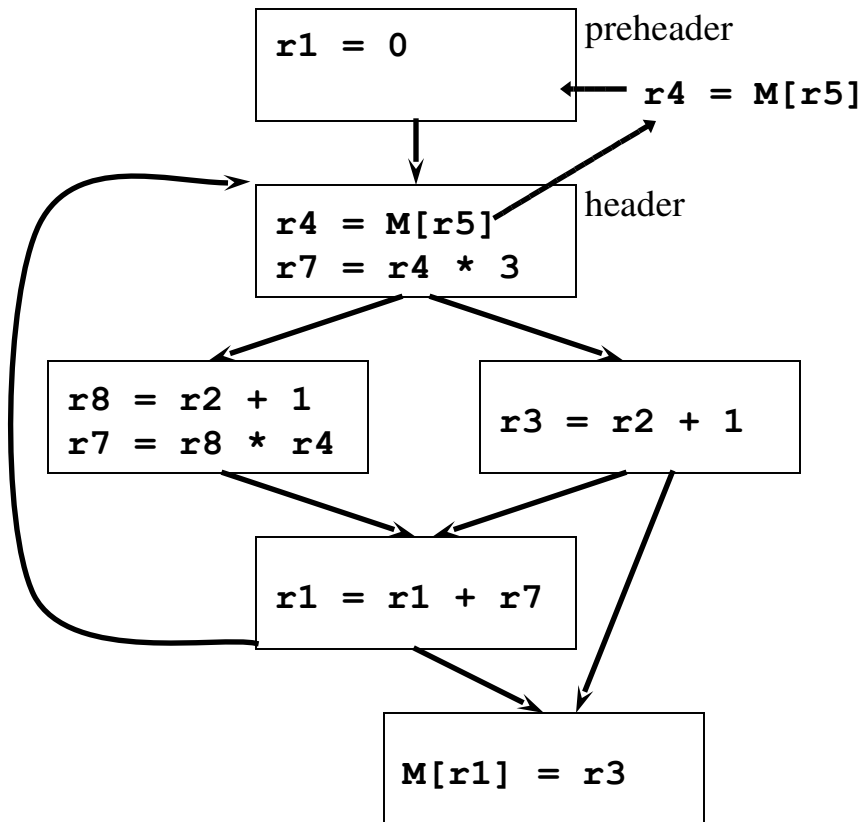
- Goal: globally propagate RHS of moves forward
  - Reduces dependence chain
  - May be possible to eliminate moves
- Rules (dataflow required)
  1. X is a move with  $\text{src1}(X)$  register
  2. Y uses  $\text{dest}(X)$
  3.  $\text{dest}(X)$  has only one def at X for UD chains to Y
  4.  $\text{src1}(X)$  has no def on any path from X to Y
  5. Replace  $\text{dest}(X)$  in Y with  $\text{src1}(X)$

# Global: Common Subexpression Elimination (CSE)



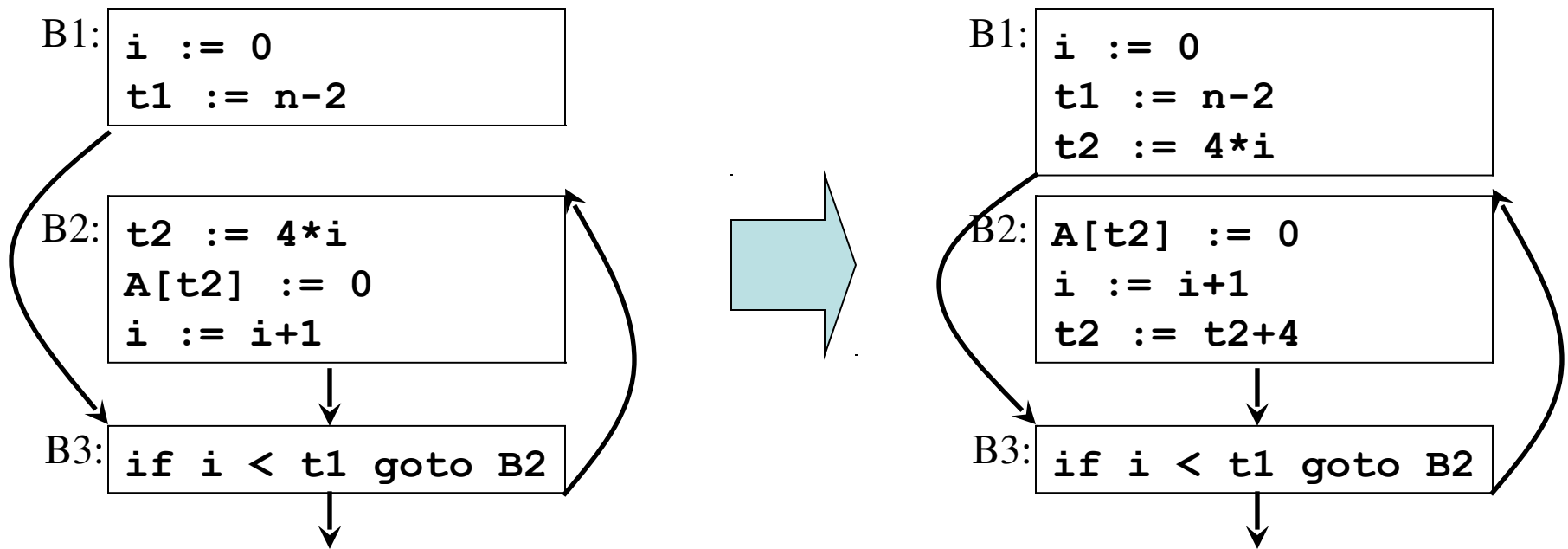
- Goal: eliminate recomputations of an expression
- Rules:
  1. X and Y have the same opcode and X dominates Y
  2.  $\text{src}(X) = \text{src}(Y)$  for all srcs
  3. For all srcs, no def of a src on any path between X and Y (excluding Y)
  4. Insert  $rx = \text{dest}(X)$  immediately after X for new register rx
  5. Replace Y with  $\text{move dest}(Y) = rx$

# Global: Code Motion



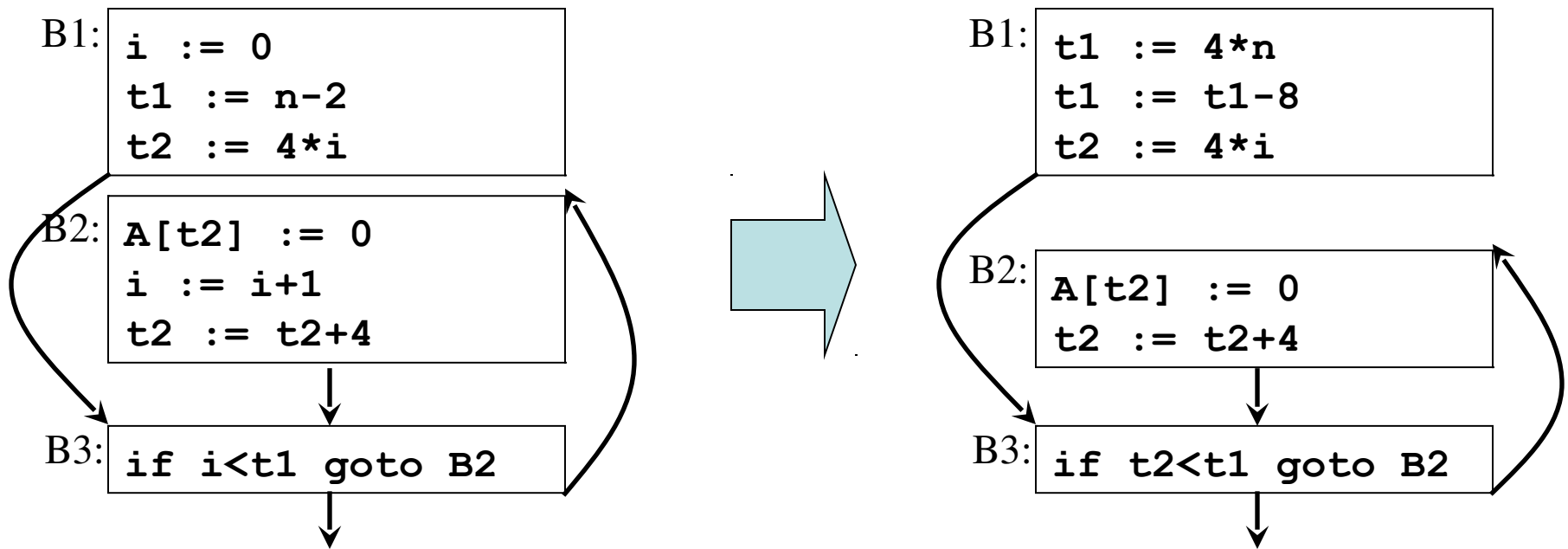
- Goal: move loop-invariant computations to preheader
- Rules:
  1. Operation X in block that dominates all exit blocks
  2. X is the only operation to modify `dest(X)` in loop body
  3. All srcs of X have no defs in any of the basic blocks in the loop body
  4. Move X to end of preheader
  5. Note 1: if one src of X is a memory load, need to check for stores in loop body
  6. Note 2: X must be movable and not cause exceptions

# Global: Loop Strength Reduction



Replace expensive computations with *induction variables*

# Global: Induction Variable Elimination



Replace induction variable in expressions with another