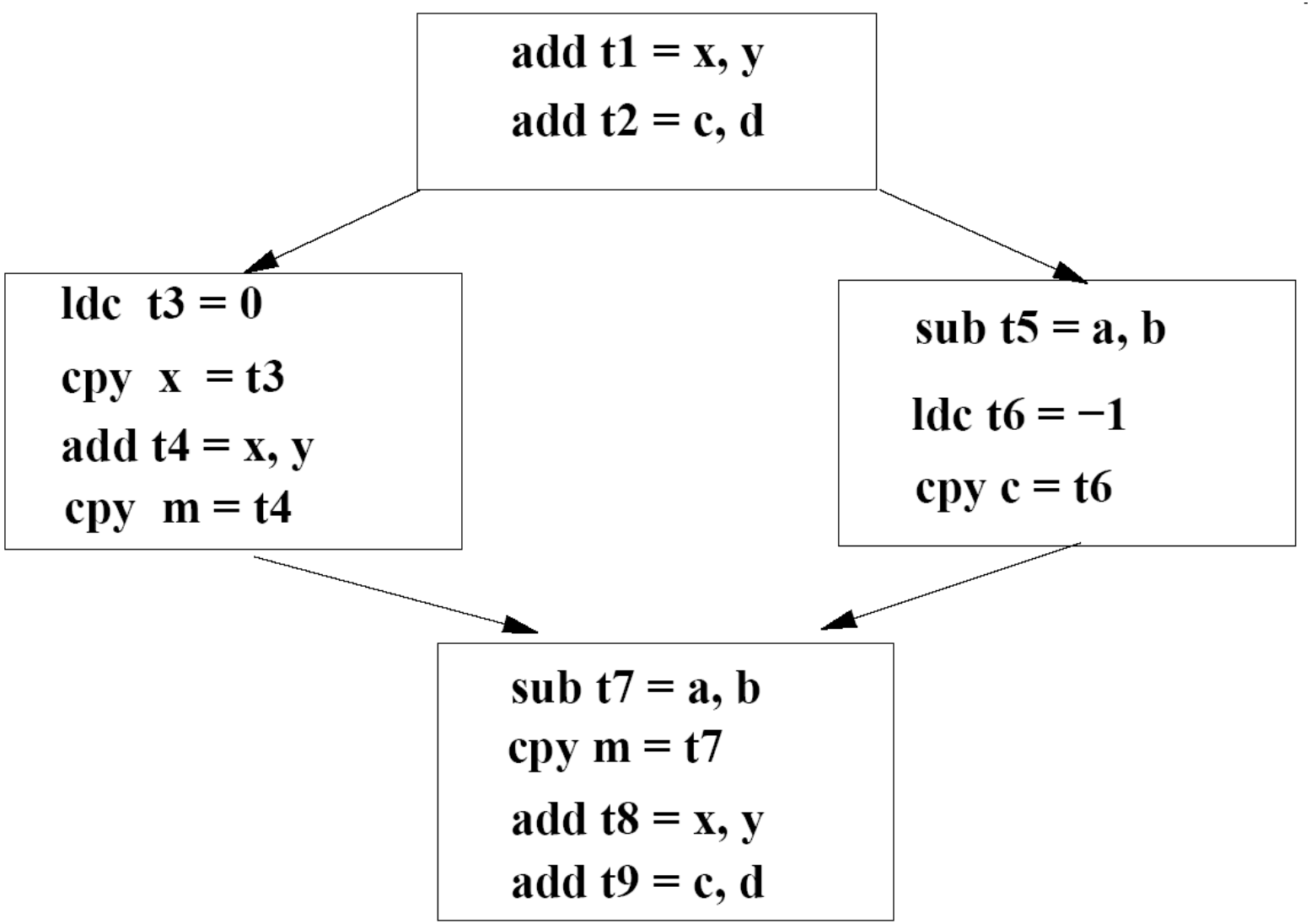
Global Common Subexpression Elimination (Global CSE)

- Available Expression Analysis
- Eliminating CSEs
- PRE (Partial Redundancy Elimination)

Global Common Subexpression



Available Expression

- Availability of an expression *E* at point *P*
  - Definition: Along every path to *P* in the flow graph
    - *E* must be evaluated at least once
    - No variables in *E* redefined after the last evaluation
  - One observation
    - *E* may have different values on different paths; variables holding value may have been overwritten

Formulating the Data Flow Problem

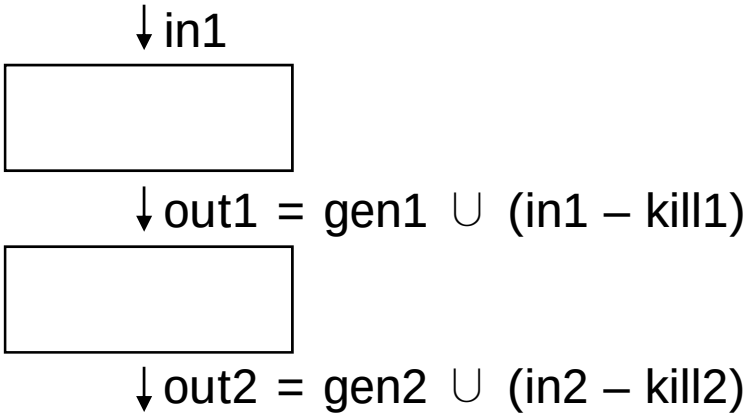
- Domain: A bit vector, a bit for each **lexically unique** expression in the program
- Forward or Backward problem?
- Meet operator?
  - check commutative, idempotent, associative
  - Partial ordering
  - Lattice elements?
  - Top? Bottom?
  - Initialization for the iterative algorithm

Transfer Functions

- Can use same equation as reaching definitions
  - out[b] = gen[b] $\cup$ (in[b] – kill[b])
- Start with a function for a single instruction
  - What expression does the instruction generate?
  - When does it kill an expression?
- Calculate transfer functions for basic blocks
  - Compose individual instruction transfer functions

Composing Transfer Functions

- Derive the transfer function for an entire block



- Since out1 = in2, we can simplify:
  - out2 = gen2 ∪ ((gen1 ∪ (in1 – kill1)) – kill2)

    out2 = gen2 ∪ (gen1 – kill2) ∪ (in1 – (kill1 ∪ kill2))
- Result
  - gen = gen2 ∪ (gen1 – kill2)
  - kill = kill1 ∪ kill2

Eliminating CSEs

- Available expressions (across basic blocks)
  - Provide the set of available expressions at each block header
- Value numbering (within a basic block)
  - Initialize value table with available expressions
  - Eliminate redundant expressions

Redundancy Elimination

- Global CSE removes redundant computations
- Some computations are partially redundant

```
if  (a > 0)          ➔          t = b + c
    d = b + c                   if ( a > 0 )
e = b + c                          d = t
                                e = t
```

- `B + c` is evaluated twice if a > 0
- Inserting `t = b+ c` earlier makes both computations completely redundant, allowing to remove both
- This technique is referred to as Partial Redundancy Elimination (PRE)

Partial Redundancy Elimination (PRE)

- Global CSE removes redundant computations
- PRE inserts new computations which render others redundant and allow their removal by global CSE
- Correctness issues: the following is incorrect

```
                                        t = b + c
if  (a > 0)          ➔          if ( a > 0 )
    b = 1                           b = 1
    d = b + c                       d = t
e = b + c                       e = t
```

- Need a dataflow analysis to determine where to place the new computation

Anticipation Analysis

- An expression E is anticipated at a point P if the same expression, which computes the same value, occur after P in every execution path starting at P
  - This would be a necessary condition for inserting a calculation of E at P
- Anticipation analysis determines whether a particular expression a + b is anticipated at basic block boundaries, which can also be generalized to determine which expressions are anticipated at basic block boundaries
  - Do it in your homework