Machine Independent Optimizations

- **Two classical data-flow problems**
  - **Reaching definitions**
  - **Live variables**
    - **UD, DU Chains**

Definitions

Flow analysis:

Fact finding about a program before its execution

Control-flow analysis:

Discerning possible execution paths.

Data-flow analysis:

Determining information about modification, preservation, and use of data entities in a program.

Two classic data-flow problems

Reaching definitions (REACH), Live uses of variables (LIVE)

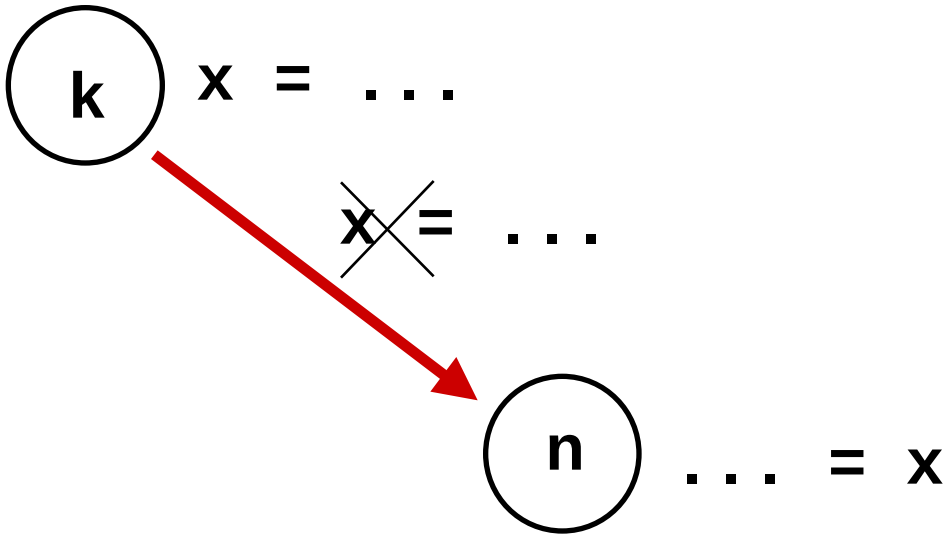
Def-use and Use-def chains, built from REACH and LIVE, used for many optimizations

Reaching Definitions (REACH)

Definition:

A statement that can modify the value of a variable.

A definition of a variable x at node k reaches node n if there is a definition-clear path from k to n.

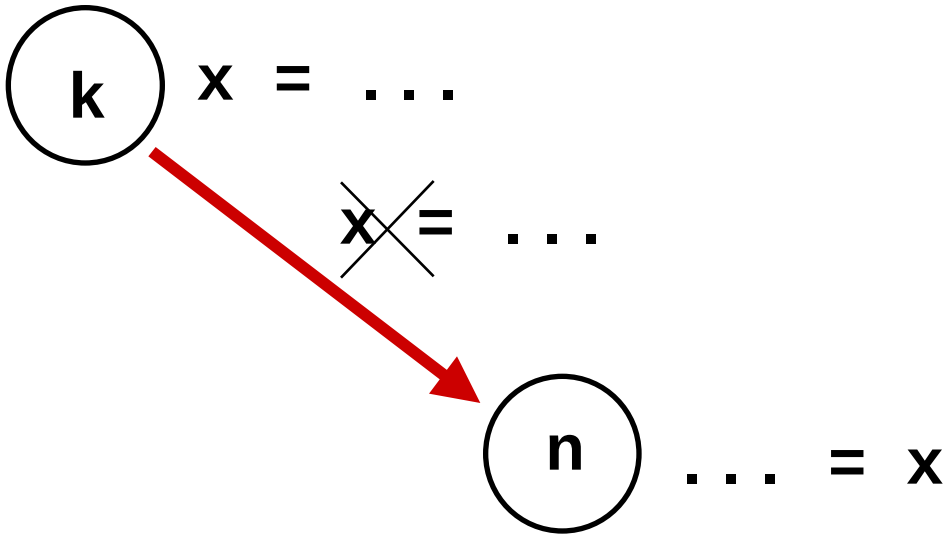


Live Uses of Variables (LIVE)

Use:

An appearance of a variable as an operand in 3 address code.

A use of a variable x at node n is live on exit from node k if there is a definition-clear path for x from k to n.



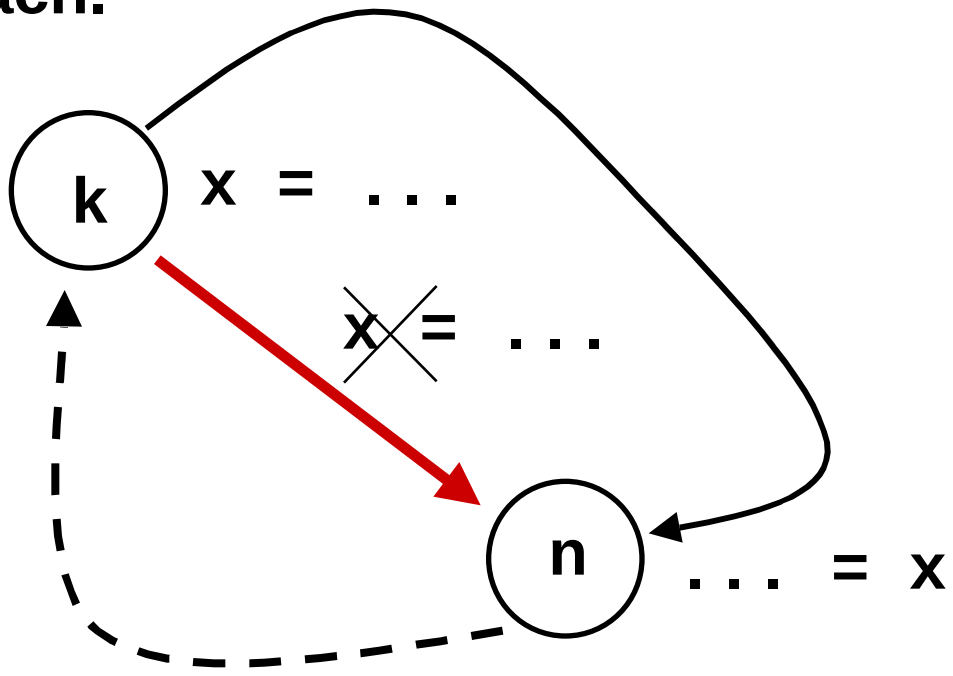
REACH and LIVE

UD- Chain: – – – – –▶

Links each use of variable x to definition(s) which reach that use.

DU - Chain: ⟶

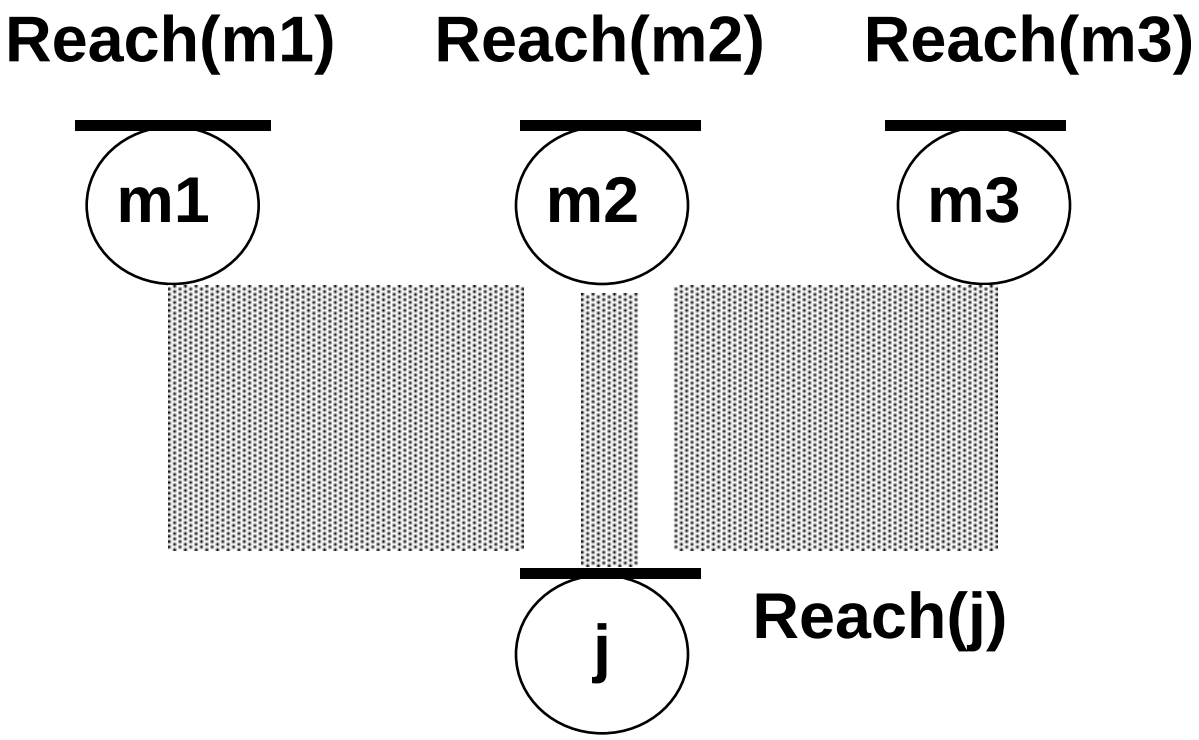
Links each definition of variable x to those uses which that definition can reach.



Global Optimizations Needing DU - UD Chains

- **Live ranges for global register allocation(DU)**
- **Dead code elimination (DU)**
- **Code motion (UD)**
- **Strength reduction (UD)**
- **Test elision (UD)**
- **Constant propagation (UD)**
- **Copy propagation (DU)**

Reaching Definitions



forward
data-flow
problem

Data-Flow Equations

REACH

Reach(j) =

$$\bigcup_{m \in pred(j)} \{ Reach(m) \cap pres(m) \cup dgen(m) \}$$

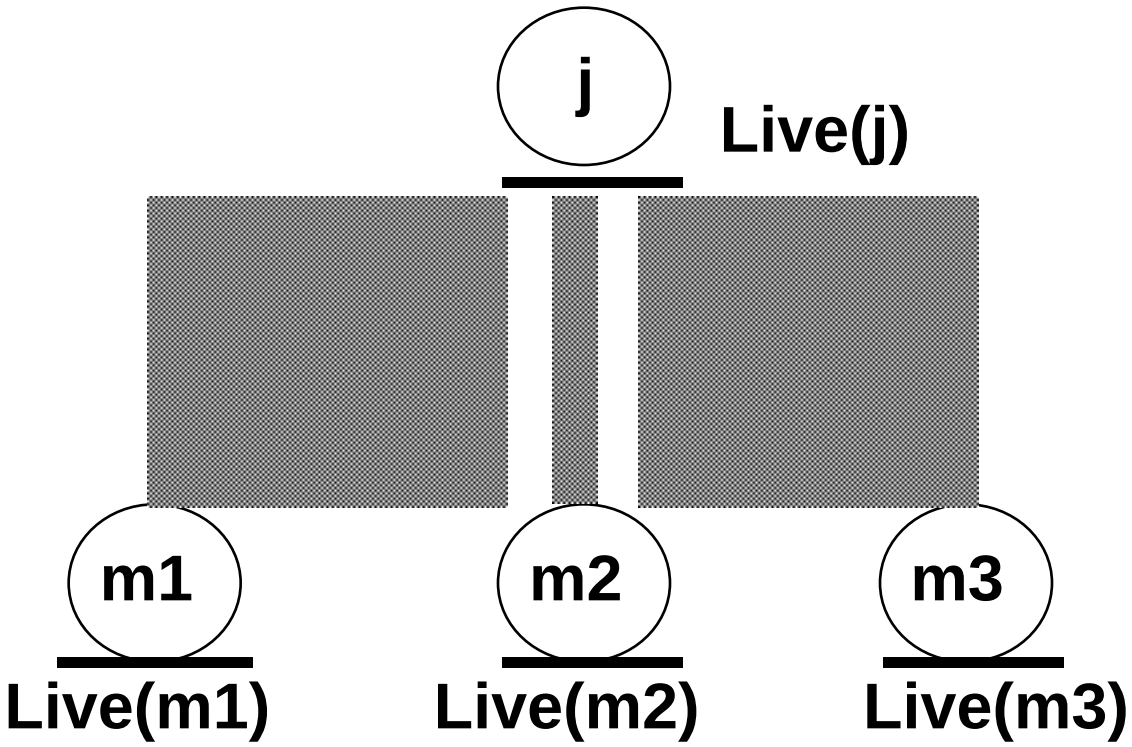
where:

pres(m) is the set of defs preserved through node m

dgen(m) is the set of defs generated at node m

pred(j) is the set of immediate predecessors of node j

Live Uses of Variables



backward
data-flow
problem

Data-Flow Equations

LIVE

Live(j) =

$$\bigcup_{m \in succ(j)} \{ Live(m) \cap upres(m) \cup ugen(m) \}$$

where:

upres(m) is the set of uses preserved through node m

ugen(m) is the set of uses generated at node m

succ(j) is the set of immediate successors of node j

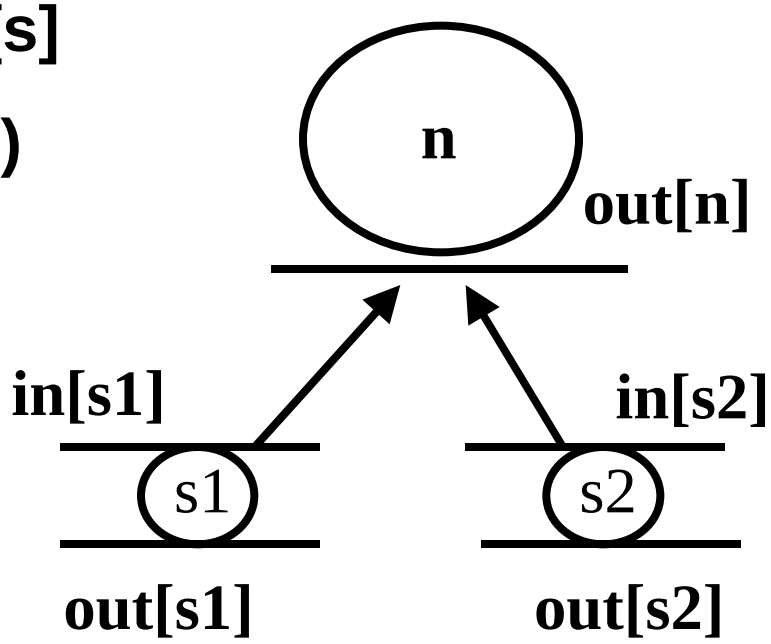
Data-flow Equations

Compare with textbook's equations, in[n] holds on entry to the node; out[n] holds on exit from the node.

in[n] := use[n] ⋃ (out[n] - def[n])

out[n] := ⋃ in[s]

s ∈ succ(n)

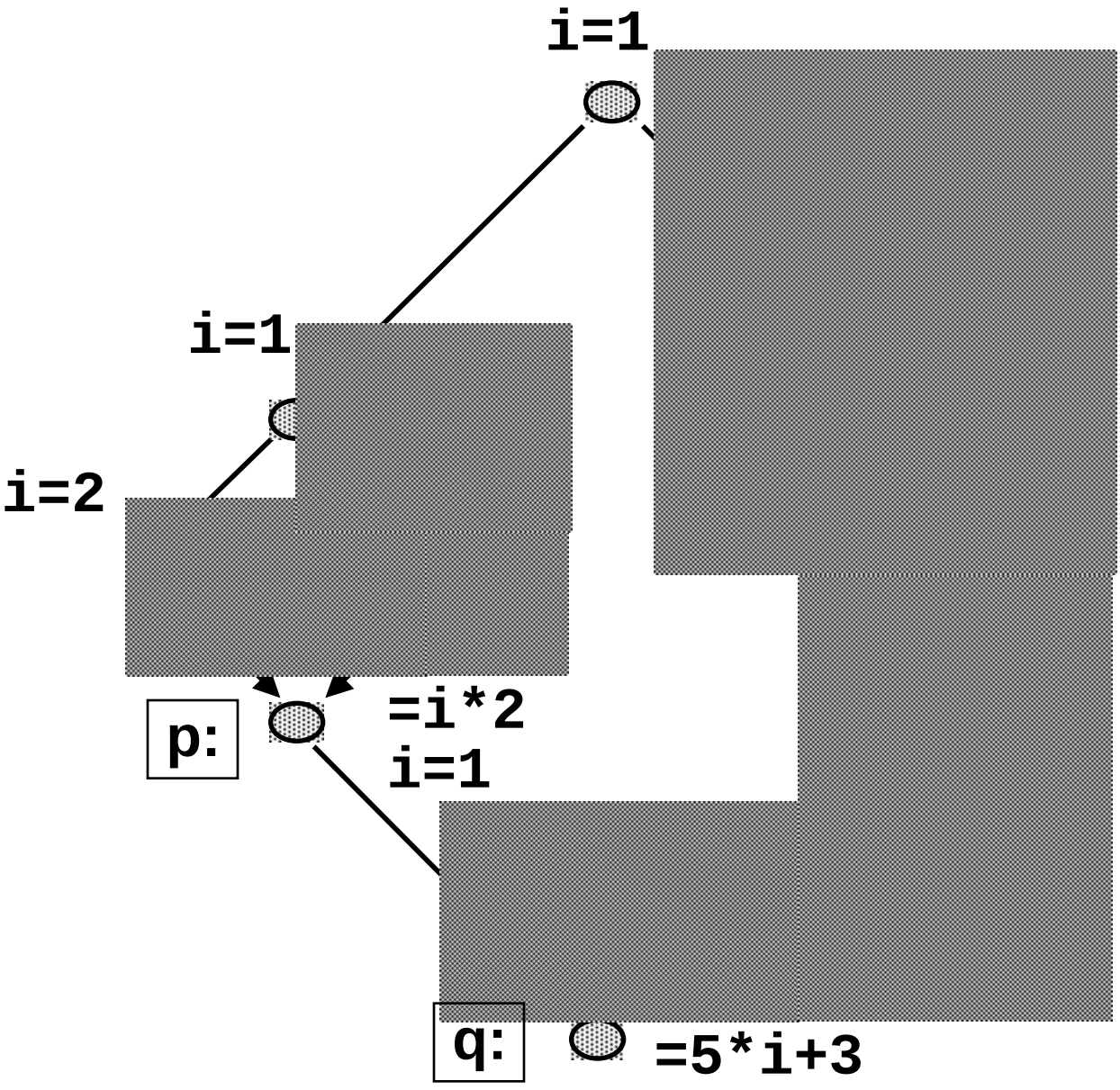


out[n] = (use[s1] ⋃ (out[s1] - def[s1])) ⋃ (use[s2] ⋃ (out[s2] - def[s2]))

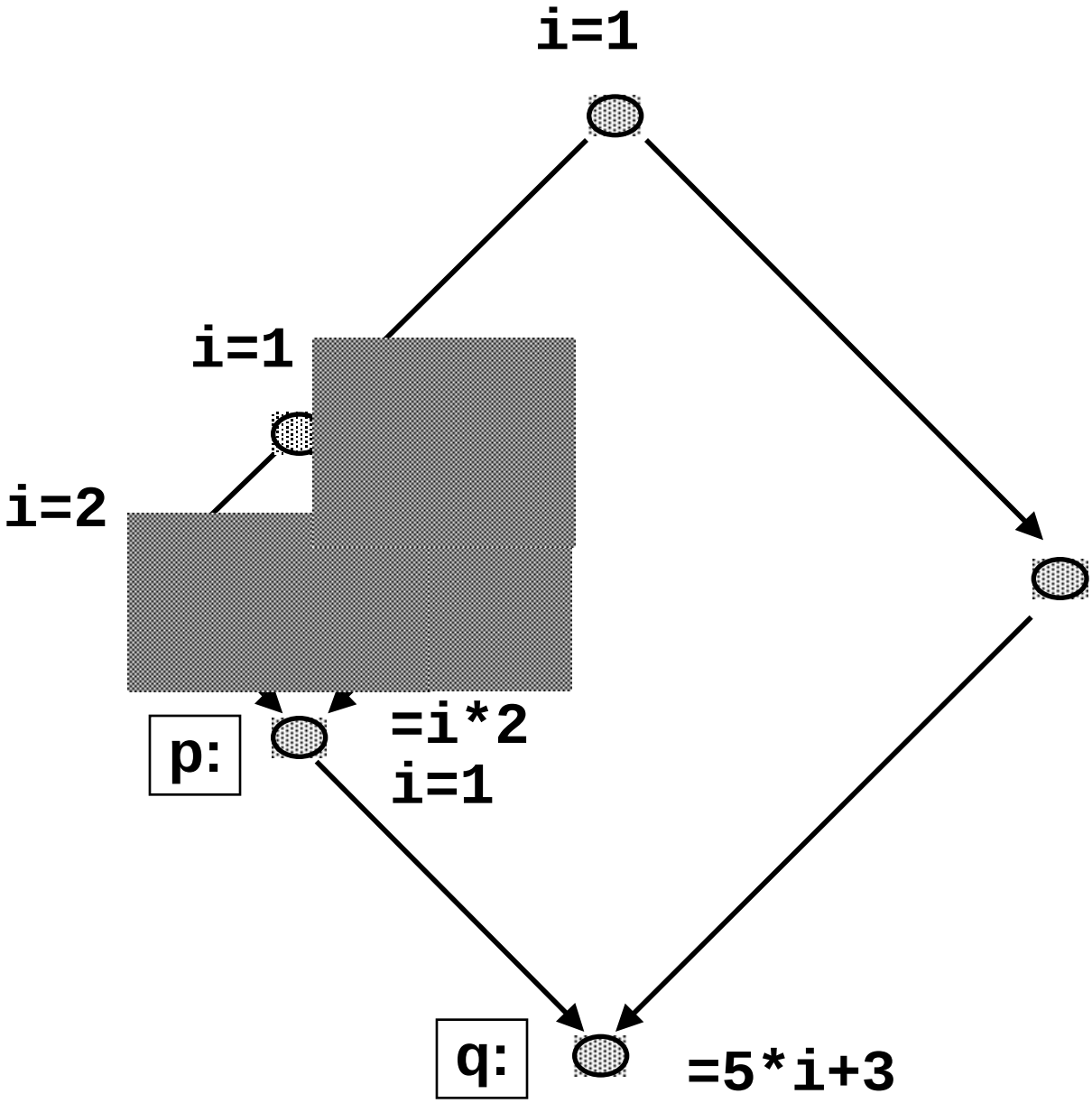
where use==ugen,

out - def == out ∩ upres

Constant Propagation

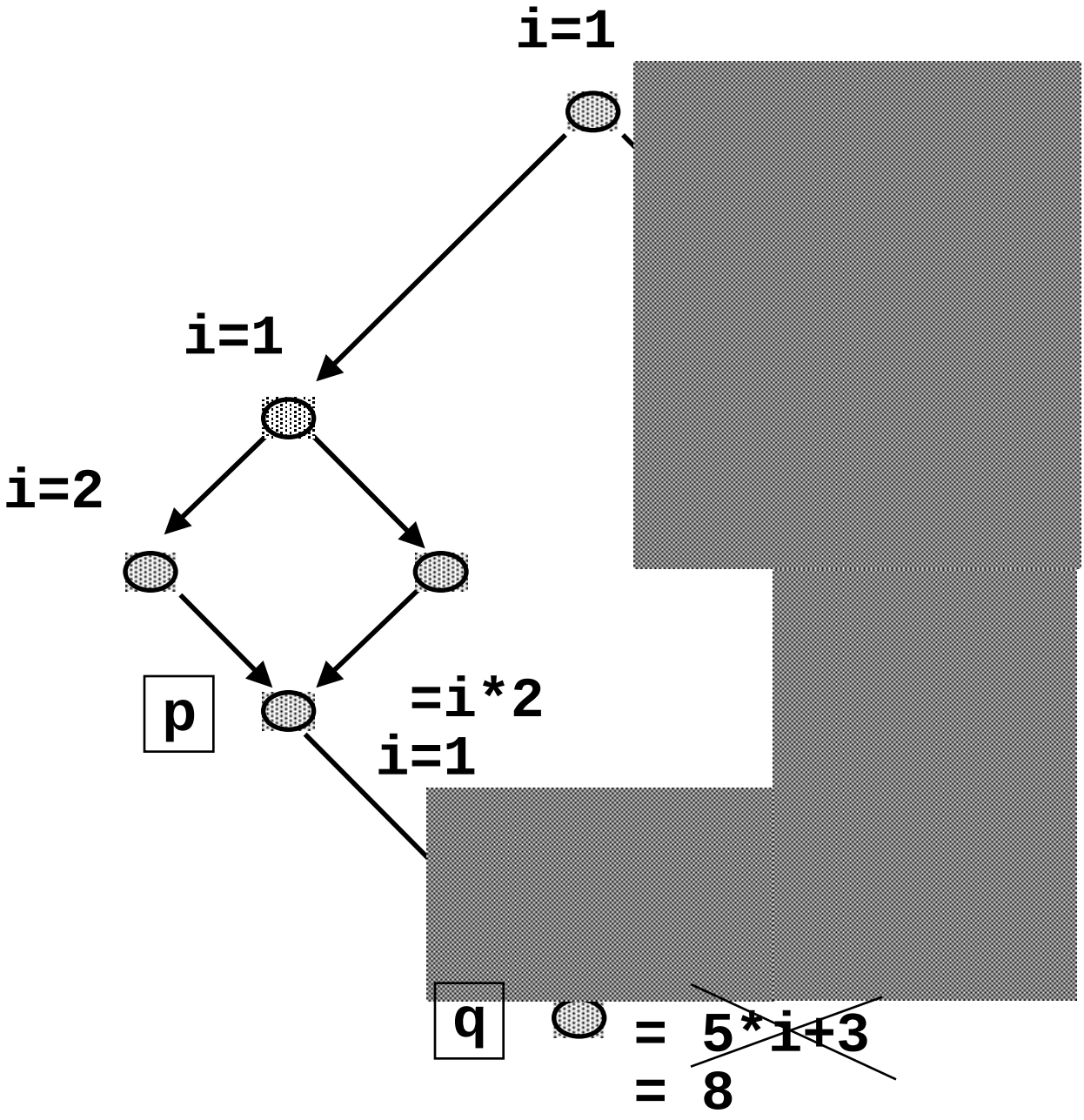


Constant Propagation



At program point p, UD chain shows all definitions reaching this use are constant - but not the same constant.
No propagation.

Constant Propagation



At program point q, UD chain shows all defs reaching this use are constant - and the same constant.