

SWEN430 - Compiler Engineering

Lecture 22 - Register Allocation II

Lindsay Groves and David J. Pearce

*School of Engineering and Computer Science
Victoria University of Wellington*

Live Variables Analysis

- Find live variables by *data flow analysis*.
- For each node, v , define:
 - $LIVE_{IN}(v)$ to be the set of variable which are live on entry to v .
 - $LIVE_{OUT}(v)$ to be the set of variable which are live on exit from v .
- A variable is live on entry to v if is is live on exit from v but not defined at v , or iof it is used at v .
- A variable is live on exit from v if it is live on entry to some successor of v .

Live Variables Analysis

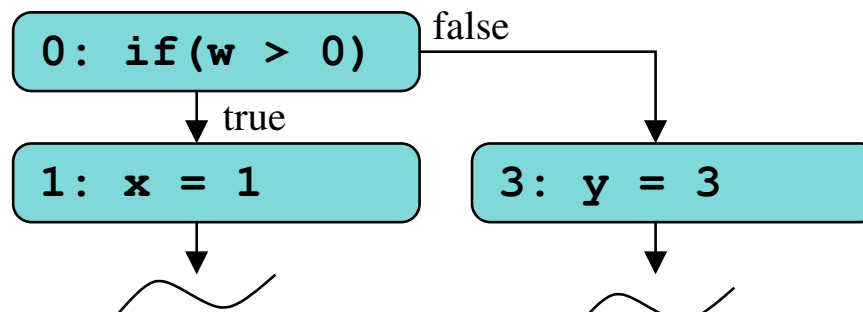
Dataflow Equations

Let $D = (V, E)$ represent the CFG of a method. Then, the dataflow equations for a node $v \in V$ are:

$$LIVE_{IN}(v) = \left((LIVE_{OUT}(v) - DEF_{AT}(v)) \cup USES(v) \right)$$

$$LIVE_{OUT}(v) = \bigcup_{v \rightarrow w \in E} LIVE_{IN}(w)$$

- This is a **backwards** flow-analysis. Compute $LIVE_{OUT}(v)$ by unioning $LIVE_{IN}(w)$ for each successor w of v :



Here, $LIVE_{OUT}(0) = LIVE_{IN}(1) \cup LIVE_{IN}(3)$

Live Variables: Solving Dataflow Equations

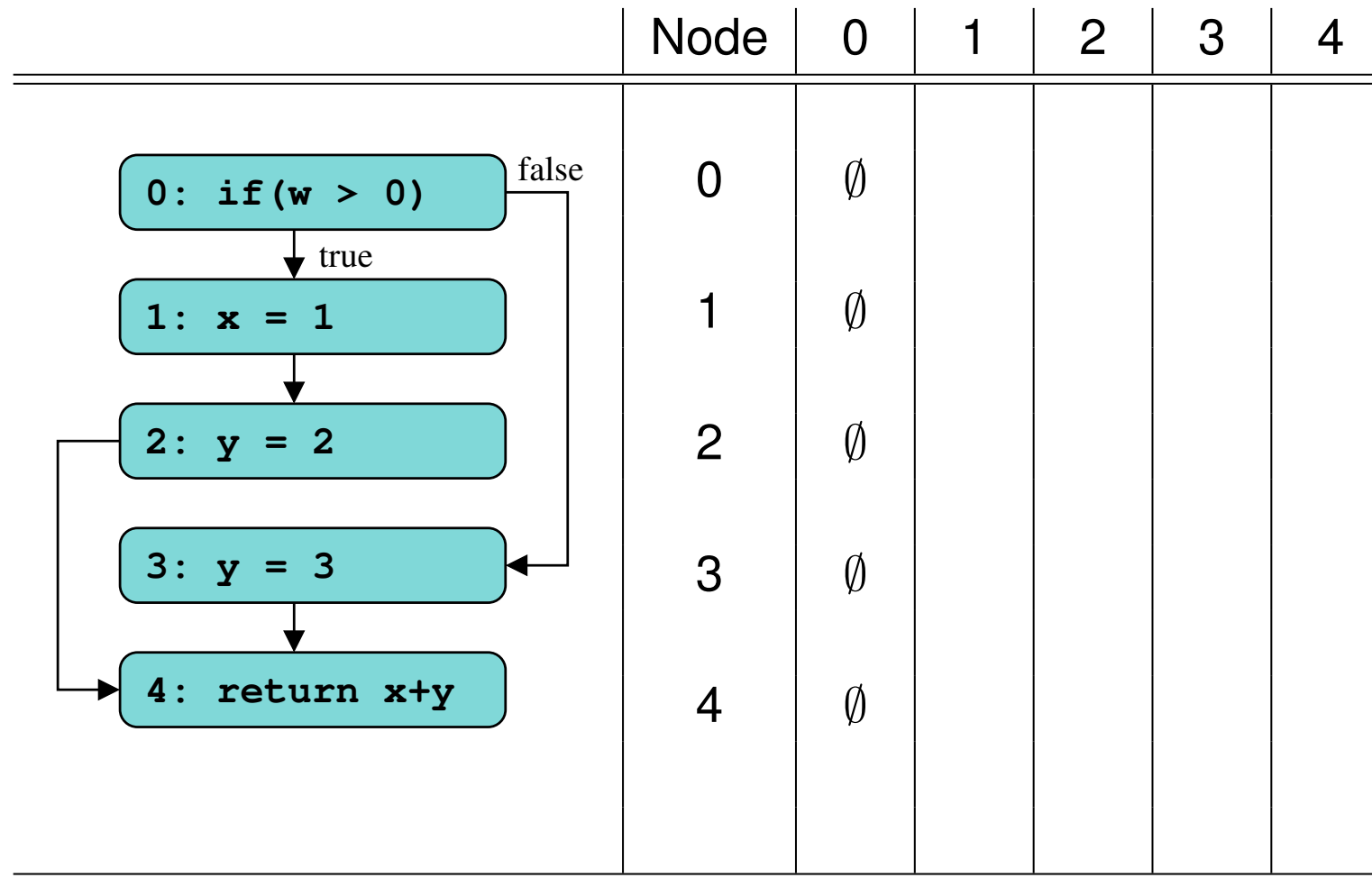
- Basic Algorithm:

- ① Set $LIVE_{OUT}(v) = LIVE_{IN}(v) = \emptyset$ for all nodes v
- ② Select node v
- ③ Calculate $LIVE_{OUT}(v)$ based on current solution
- ④ Calculate $LIVE_{IN}(v)$ using $LIVE_{OUT}(v)$
- ⑤ Repeat from step 2 — *until solution sets reach fixed point*
 - » Guaranteed to *terminate* and yield *smallest* (i.e. most precise) solution for dataflow equations (provided select is fair)
 - » Solution sets reach fixed point when they no longer change

- Iteration Strategies:

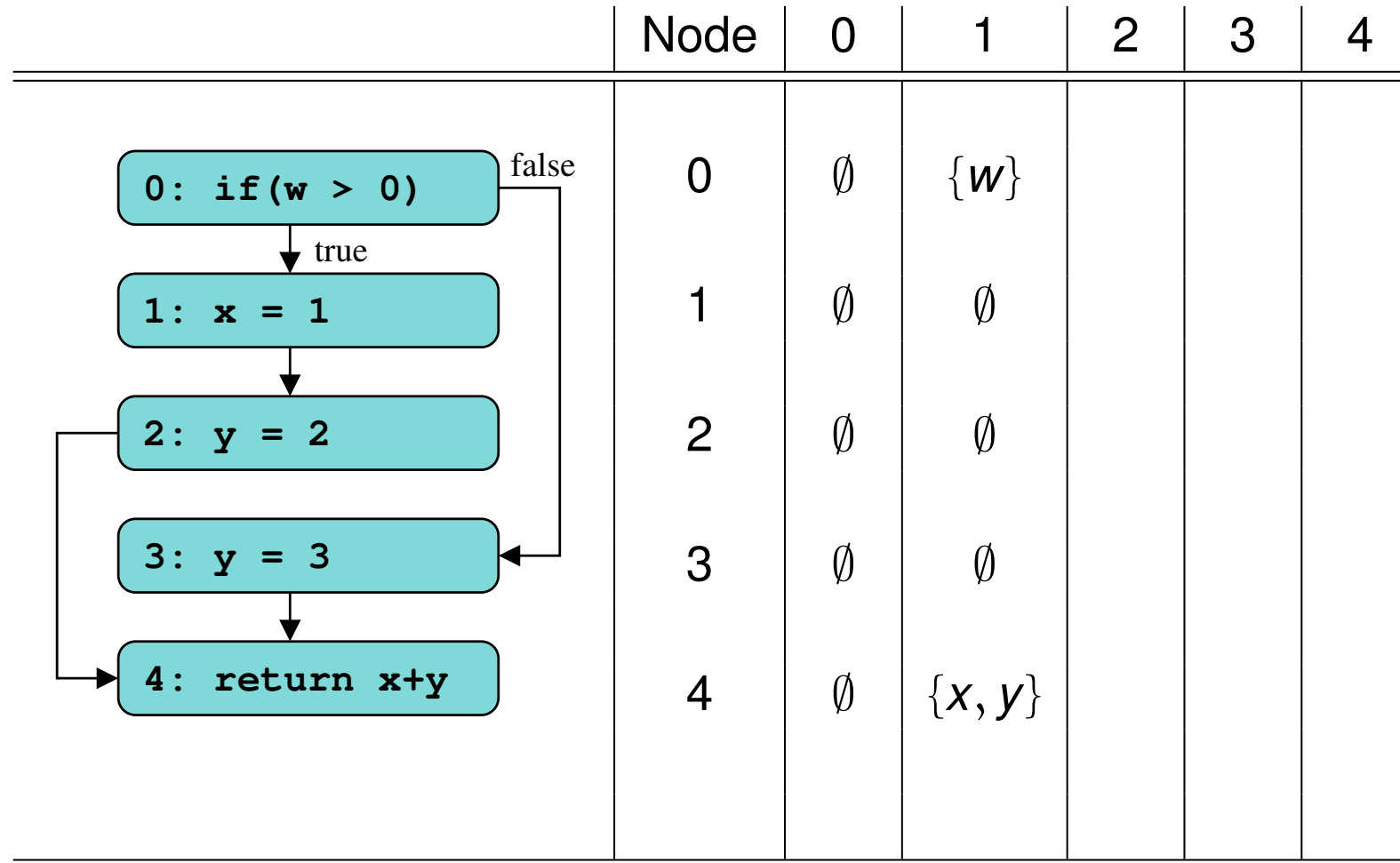
- » Choice of select function affects **performance**
- » **Chaotic** — select nodes at *random*
- » **Forward Iterative** — select nodes in *increasing order*; when all done start over
- » **Backward Iterative** — select nodes in *decreasing order*; when all done start over

Example: Forward Iterative Strategy



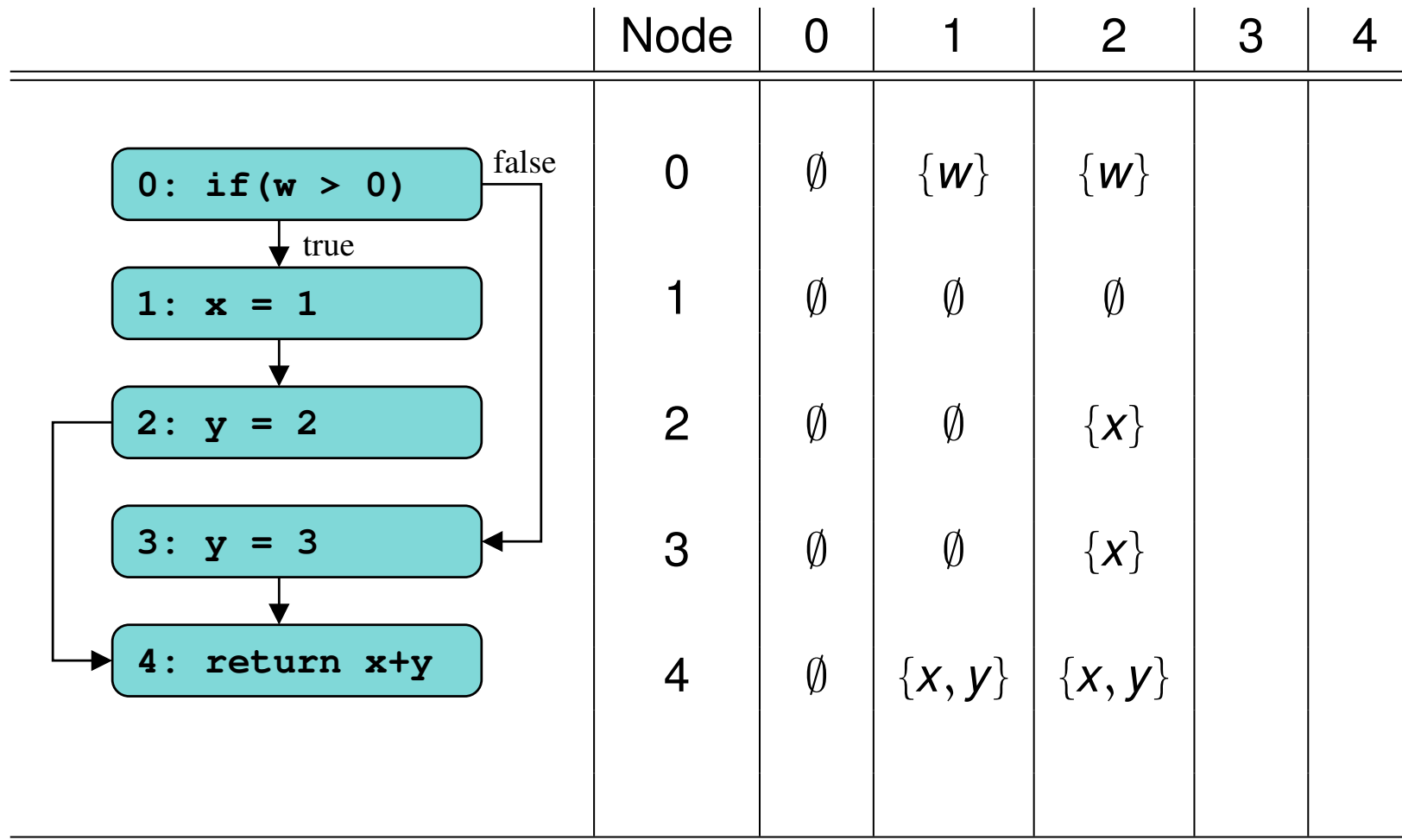
(Initial state)

Example: Forward Iterative Strategy



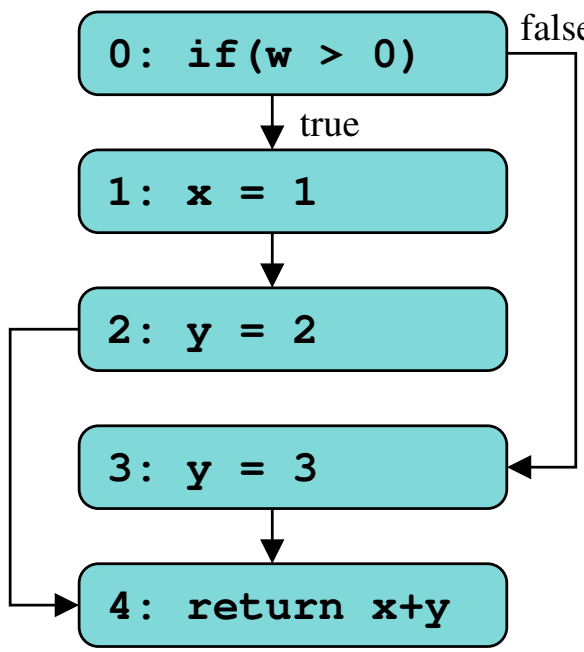
(State after one iteration)

Example: Forward Iterative Strategy



(State after two iterations)

Example: Forward Iterative Strategy

| | Node | 0 | 1 | 2 | 3 | 4 |
|--|------|-------------|-------------|-------------|-------------|---|
|  | 0 | \emptyset | $\{w\}$ | $\{w\}$ | $\{w, x\}$ | |
| | 1 | \emptyset | \emptyset | \emptyset | \emptyset | |
| | 2 | \emptyset | \emptyset | $\{x\}$ | $\{x\}$ | |
| | 3 | \emptyset | \emptyset | $\{x\}$ | $\{x\}$ | |
| | 4 | \emptyset | $\{x, y\}$ | $\{x, y\}$ | $\{x, y\}$ | |

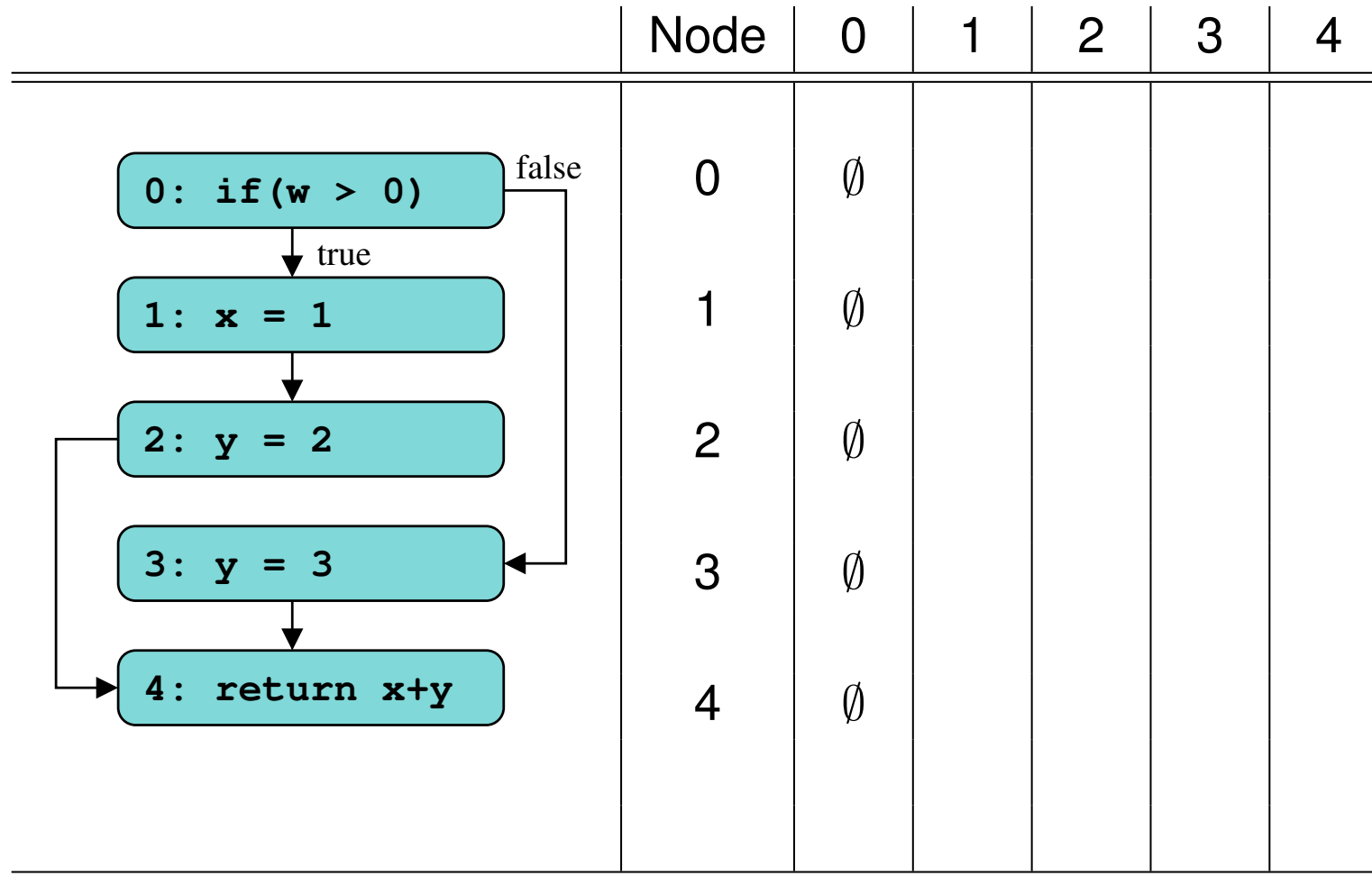
(State after three iterations)

Example: Forward Iterative Strategy

| | Node | 0 | 1 | 2 | 3 | 4 |
|--|------|-------------|-------------|-------------|-------------|-------------|
| <pre> graph TD N0[0: if(w > 0)] -- true --> N1[1: x = 1] N0 -- false --> N3[3: y = 3] N1 --> N2[2: y = 2] N2 --> N3 N3 --> N4[4: return x+y] N4 --> N2 </pre> | 0 | \emptyset | $\{w\}$ | $\{w\}$ | $\{w, x\}$ | $\{w, x\}$ |
| | 1 | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset |
| | 2 | \emptyset | \emptyset | $\{x\}$ | $\{x\}$ | $\{x\}$ |
| | 3 | \emptyset | \emptyset | $\{x\}$ | $\{x\}$ | $\{x\}$ |
| | 4 | \emptyset | $\{x, y\}$ | $\{x, y\}$ | $\{x, y\}$ | $\{x, y\}$ |

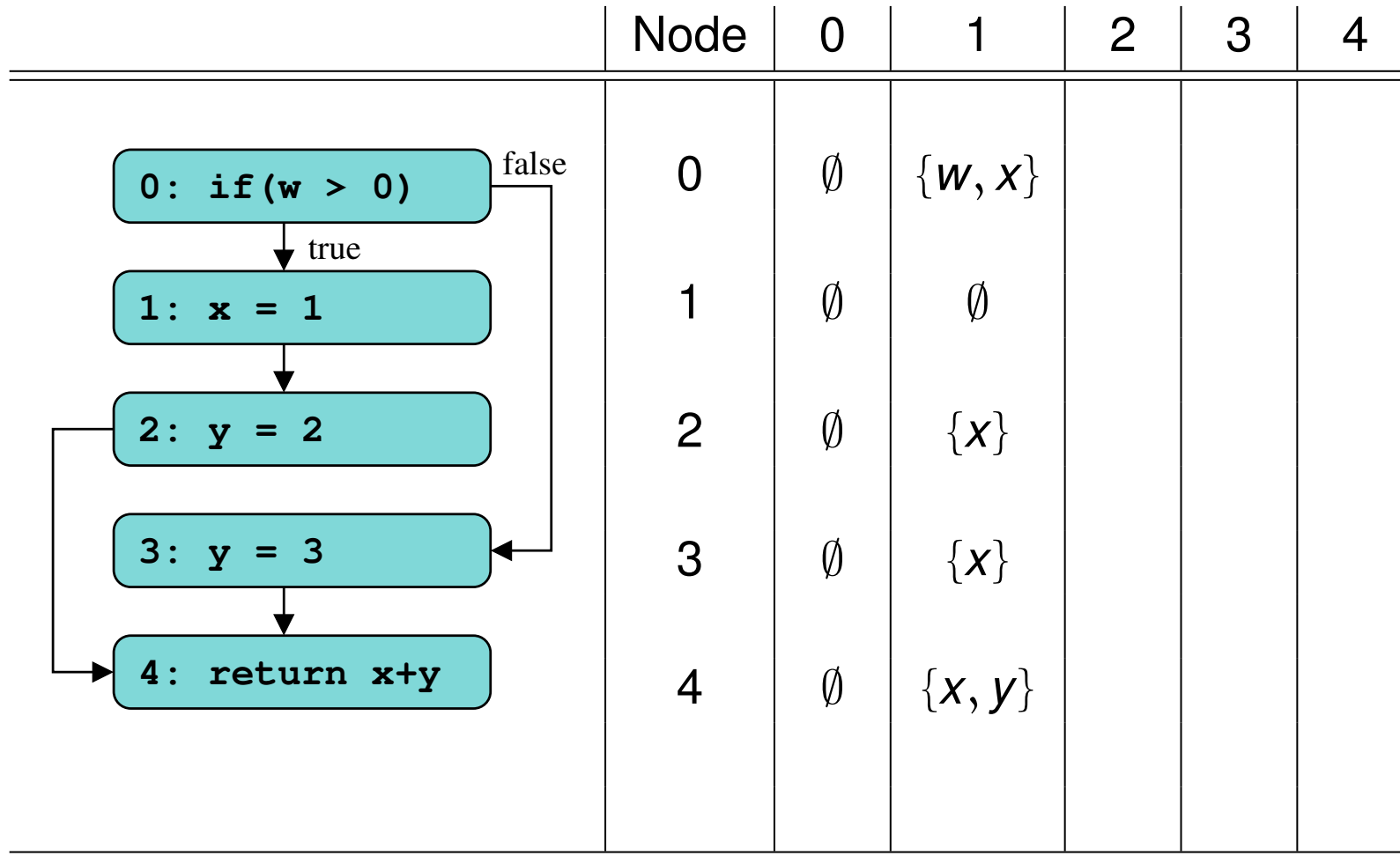
(State after four iterations — fixed-point reached)

Example: **Backward** Iterative Strategy



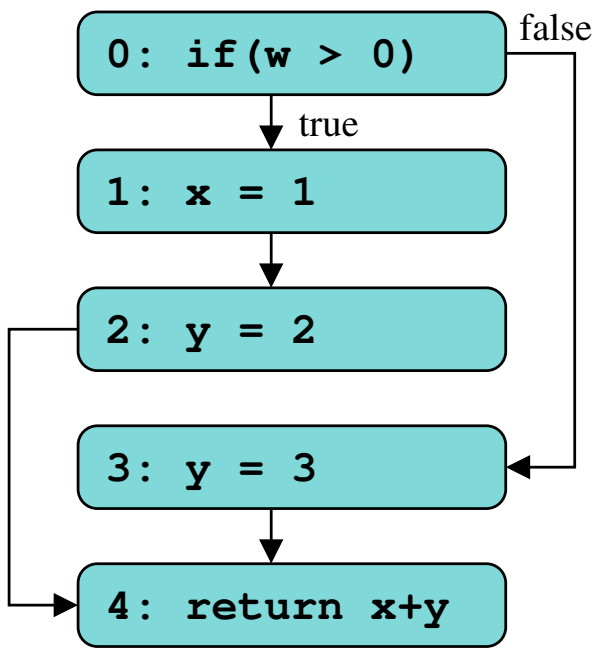
(Initial State)

Example: **Backward** Iterative Strategy



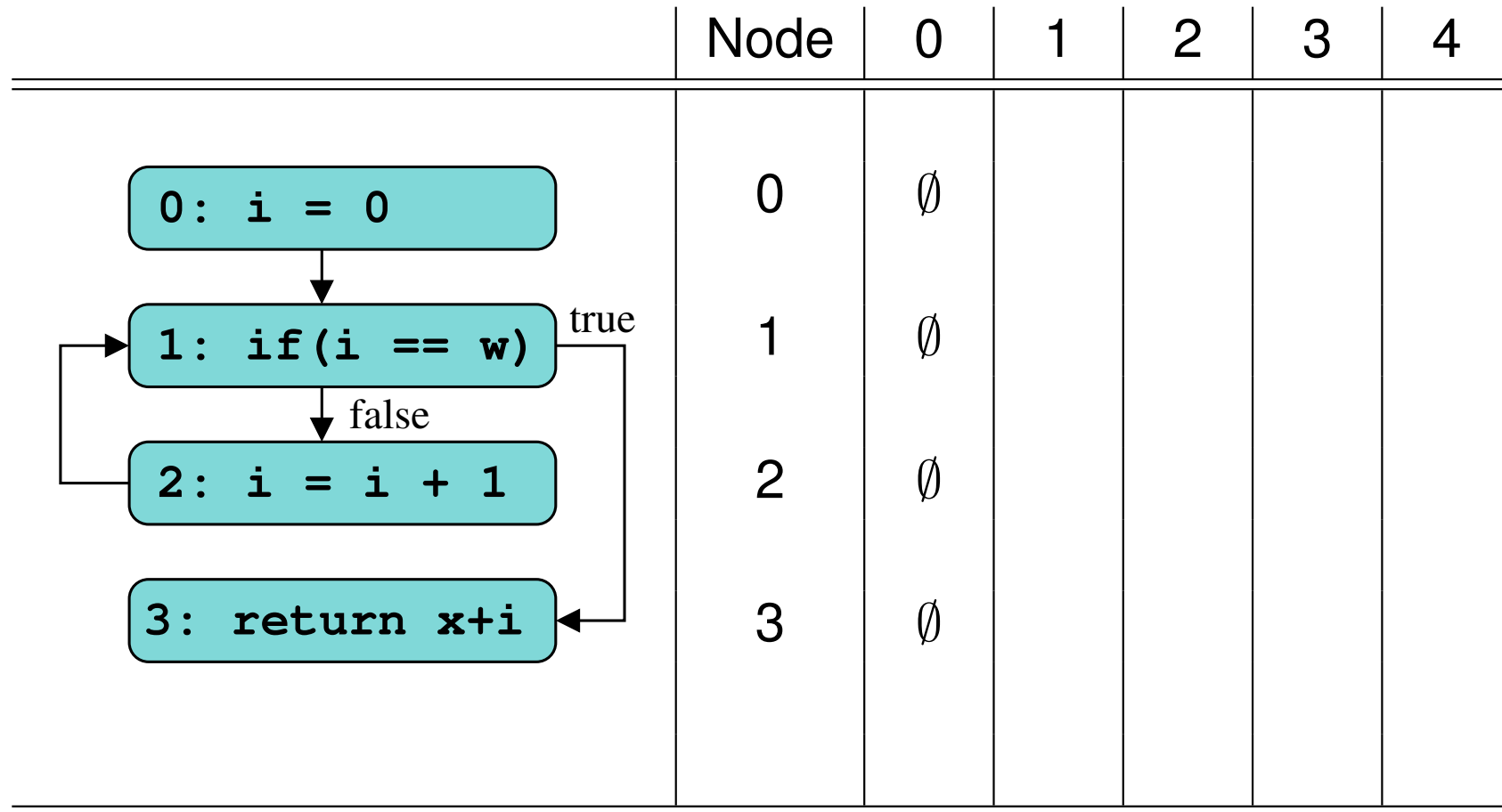
(State after one iteration)

Example: Backward Iterative Strategy

| | Node | 0 | 1 | 2 | 3 | 4 |
|--|------|-------------|-------------|-------------|---|---|
|  | 0 | \emptyset | $\{w, x\}$ | $\{w, x\}$ | | |
| | 1 | \emptyset | \emptyset | \emptyset | | |
| | 2 | \emptyset | $\{x\}$ | $\{x\}$ | | |
| | 3 | \emptyset | $\{x\}$ | $\{x\}$ | | |
| | 4 | \emptyset | $\{x, y\}$ | $\{x, y\}$ | | |

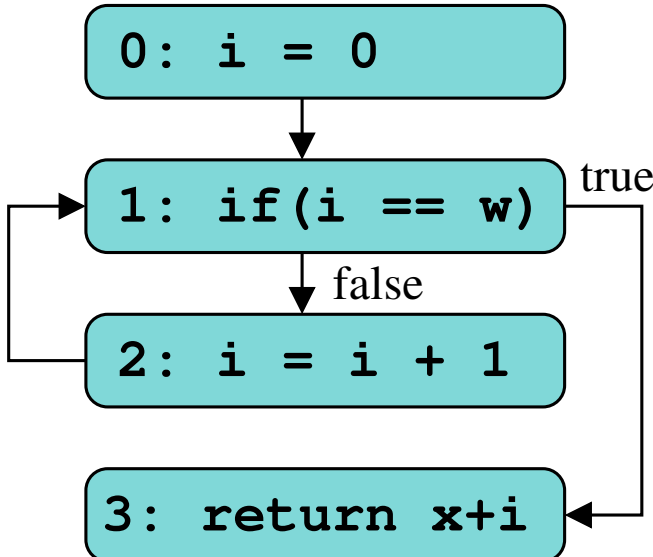
(State after two iterations — fixed-point reached)

Another Example: **Backward** Iterative Strategy



(Initial State)

Another Example: **Backward** Iterative Strategy

| | Node | 0 | 1 | 2 | 3 | 4 |
|--|------|-------------|---------------|---|---|---|
|  | 0 | \emptyset | $\{x, w\}$ | | | |
| | 1 | \emptyset | $\{x, w, i\}$ | | | |
| | 2 | \emptyset | $\{i\}$ | | | |
| | 3 | \emptyset | $\{x, i\}$ | | | |

(State after one iteration)

Another Example: **Backward** Iterative Strategy

| | Node | 0 | 1 | 2 | 3 | 4 |
|---|------|-------------|---------------|---------------|---|---|
| <pre> graph TD 0[0: i = 0] --> 1[1: if (i == w)] 1 -- true --> 1 1 -- false --> 2[2: i = i + 1] 2 --> 3[3: return x+i] </pre> | 0 | \emptyset | $\{x, w\}$ | $\{x, w\}$ | | |
| | 1 | \emptyset | $\{x, w, i\}$ | $\{x, w, i\}$ | | |
| | 2 | \emptyset | $\{i\}$ | $\{x, w, i\}$ | | |
| | 3 | \emptyset | $\{x, i\}$ | $\{x, i\}$ | | |

(State after two iterations)

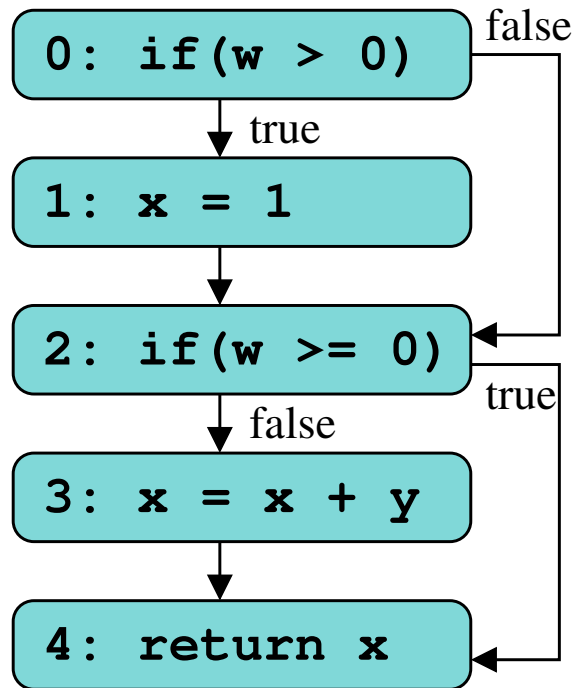
Another Example: **Backward** Iterative Strategy

| | Node | 0 | 1 | 2 | 3 |
|---|------|-------------|---------------|---------------|---------------|
| <pre> graph TD N0[0: i = 0] --> N1[1: if (i == w)] N1 -- true --> N1 N1 -- false --> N2[2: i = i + 1] N2 --> N3[3: return x+i] </pre> | 0 | \emptyset | $\{x, w\}$ | $\{x, w\}$ | $\{x, w\}$ |
| | 1 | \emptyset | $\{x, w, i\}$ | $\{x, w, i\}$ | $\{x, w, i\}$ |
| | 2 | \emptyset | $\{i\}$ | $\{x, w, i\}$ | $\{x, w, i\}$ |
| | 3 | \emptyset | $\{x, i\}$ | $\{x, i\}$ | $\{x, i\}$ |

(State after three iterations — fixed-point reached)

Limitations of Live Variable Analysis

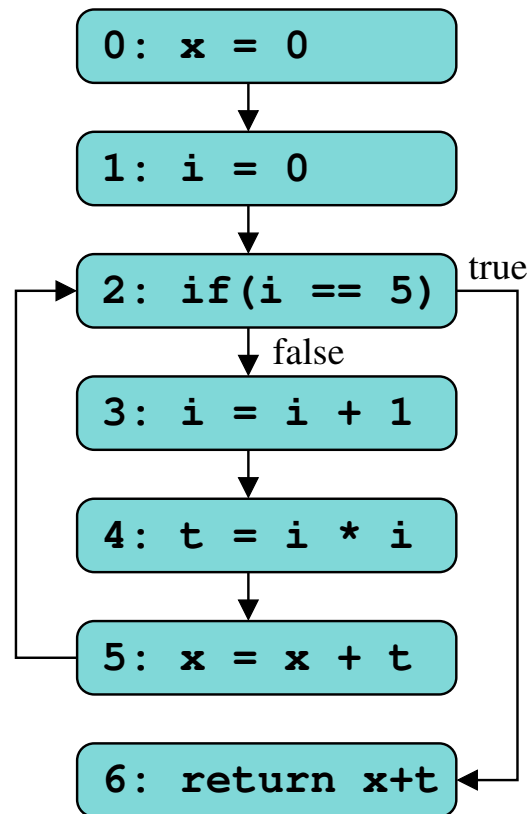
- Consider the solution for this method:



Does our dataflow algorithm produce most accurate solution possible?

Limitations of Live Variable Analysis

- Consider the solution for this method:



Does our dataflow algorithm produce most accurate solution possible?

Generating Interference Graph

Interference Graph

An *interference graph* is an undirected graph containing exactly one node per live range. There is an edge between two nodes iff their live ranges interfere.

- Simple algorithm:
 - ① Compute Live Variables Analysis
 - ② Create graph G with one node per variable
 - ③ Consider $\{a, b\} \subseteq LIVE_{OUT}(v)$ for any node v , then $(a, b) \in G$
 - ④ Consider $\{a, b\} \subseteq LIVE_{IN}(v)$ for any node v , then $(a, b) \in G$
- Do we need last step?
- More precise approach is to split out *live ranges*
- How do we determine *live range* information from solution to live variables analysis?