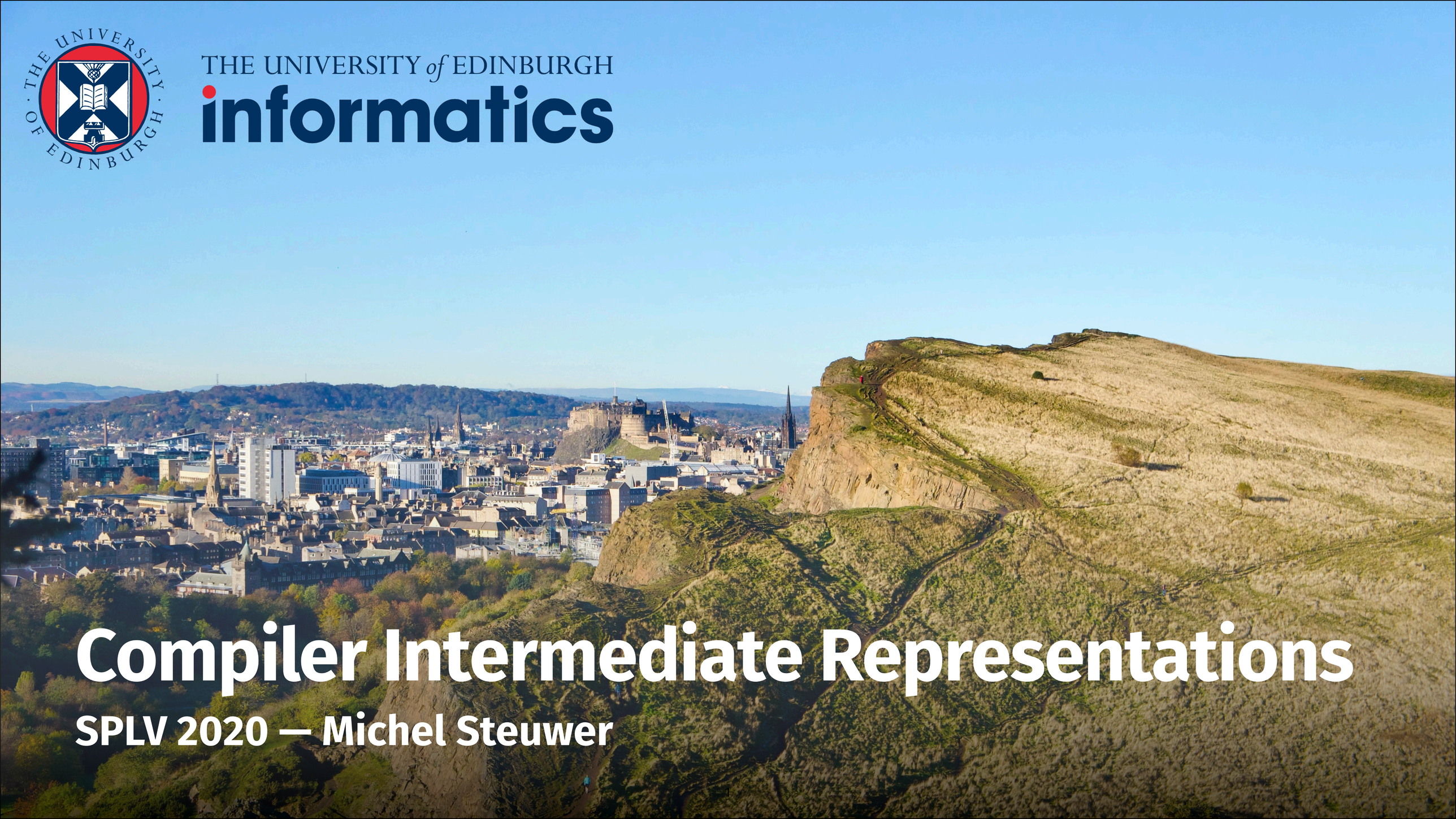


Compiler Intermediate Representations

SPLV 2020 — Michel Steuwer

Outline of Lectures over the week

- **Tuesday:** Functional Intermediate Representations
  - Lambda Calculus and the Lambda Cube
  - Implementation Strategies for System F (ADTs across different PLs)
  - Compiler transformations as rewrite rules
- **Wednesday:** Imperative Intermediate Representations
  - Data-flow analysis
  - Control-Flow Graphs
  - Foundations of Single Static Assignment (SSA)
  - LLVM IR
- **Thursday:** Domain-Specific Intermediate Representations
  - MLIR — a compiler infrastructure for building domain-specific intermediate representations
  - Dataflow graphs — TensorFlow
  - Pattern-based (and functional) — RISE

Data-flow Analysis

Data-flow analysis gathers information for *each program point* by analysing the **static** code approximating its **dynamic** behaviour

- Examples:
  - Reaching Definitions
  - Initialised Variables
  - Constant Propagation
  - Sign Analysis
  - Liveness of variables

```
1  int foo(int input) {
2    int x,y,z;
3    x = input;
4    while (x > 1) {
5      y = x / 2;
6      if (y > 3) x = x - y;
7      z = x - 4;
8      if (z > 0) x = x / 2;
9      z = z-1;
10   }
11   return x;
12 }
```

Is `z` ever initialised?

What values are possible for `y` here?

Is this computation ever used?

Liveness Analysis - What & why?

- **Intuition:** A variable is *live* at a program point if its current value may be read during the remaining execution of the program, otherwise the variable is *dead*.
- Useful for *register allocation* and *dead code elimination*

```
 1 int foo(int input) {
 2   int x,y,z;
 3   x = input;
 4   while (x > 1) {
 5     y = x / 2;
 6     if (y > 3) x = x - y;
 7     z = x - 4;
 8     if (z > 0) x = x / 2;
 9     z = z-1;
10   }
11   return x;
12 }
```

Legal transformation due to liveness information

```
 1 int foo_opt(int input) {
 2   int x, yz;
 3   x = input;
 4   while (x > 1) {
 5     yz = x / 2;
 6     if (yz > 3) x = x - yz;
 7     yz = x-4;
 8     if (yz > 0) x = x / 2;
 9
10   }
11   return x;
12 }
```

`y` and `z` can be stored in the same register

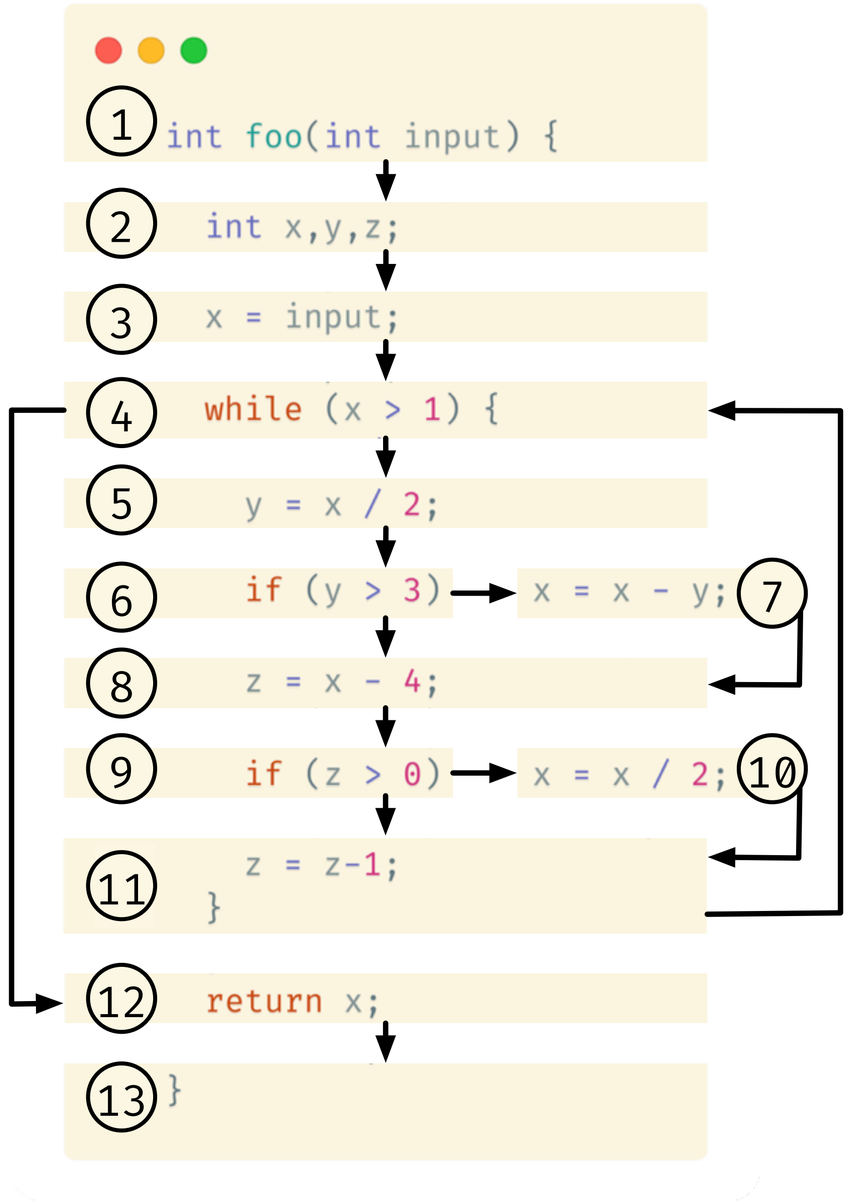
Computation was never used

Control Flow Graphs

- **Definition:** A Control Flow Graph (CFG) is a graph who's *nodes* represent **program statements** and who's *directed edges* represent **control flow.**

- Example:

```
 1 int foo(int input) {
 2   int x,y,z;
 3   x = input;
 4   while (x > 1) {
 5     y = x / 2;
 6     if (y > 3) x = x - y;
 7     z = x - 4;
 8     if (z > 0) x = x / 2;
 9     z = z-1;
10   }
11   return x;
12 }
```

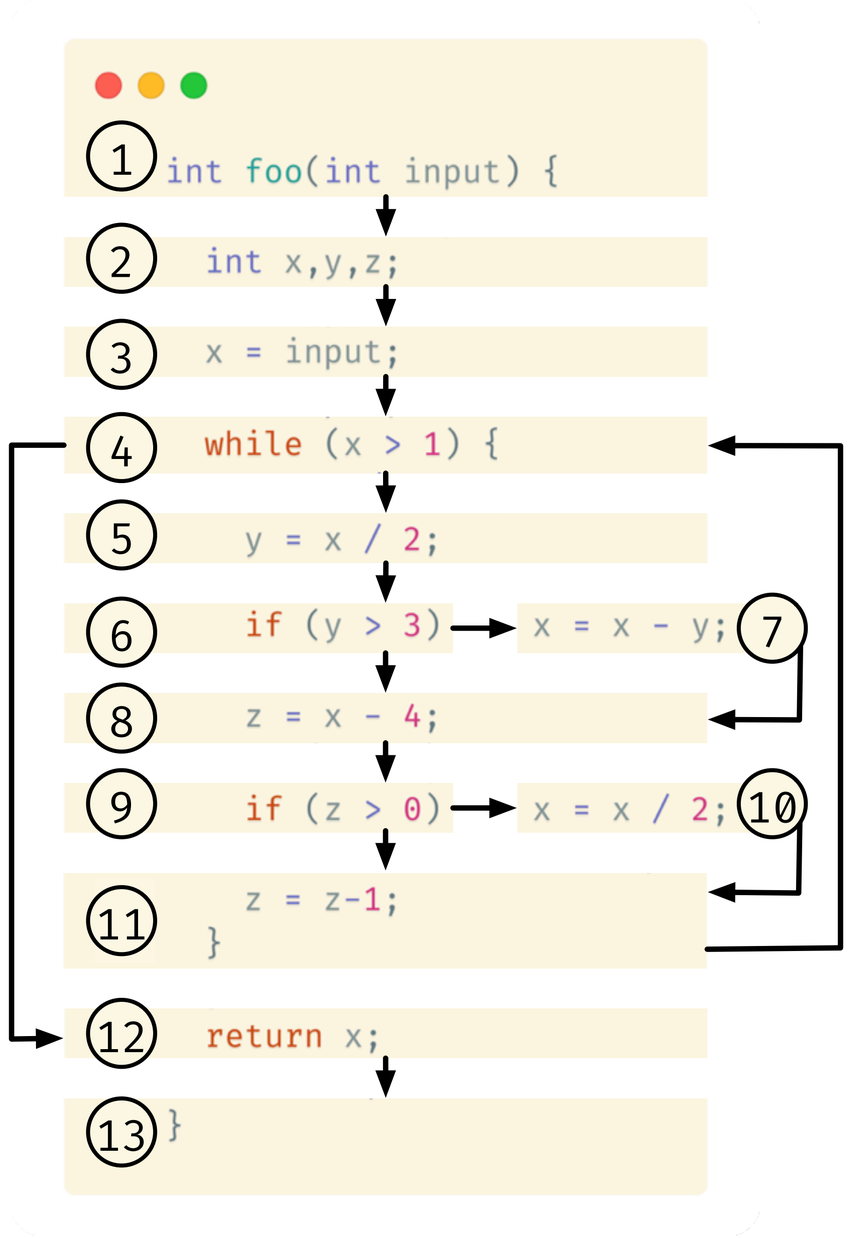


Definition of def and use

- For *defining* (or *writing*) a variable we write:
  - $def_{nodes}(v)$ = set of nodes that define variable v
  - $def_{var}(n)$ = set of variables defined at node **n**
- For *using* (or *reading*) a variable we write:
  - $use_{nodes}(v)$ = set of nodes that use variable v
  - $use_{var}(n)$ = set of variables used at node **n**
- Examples:

  $def_{nodes}(z) = \{ 8, 11 \}$

  $def_{var}(4) = \{ \}$ $use_{var}(4) = \{ x \}$

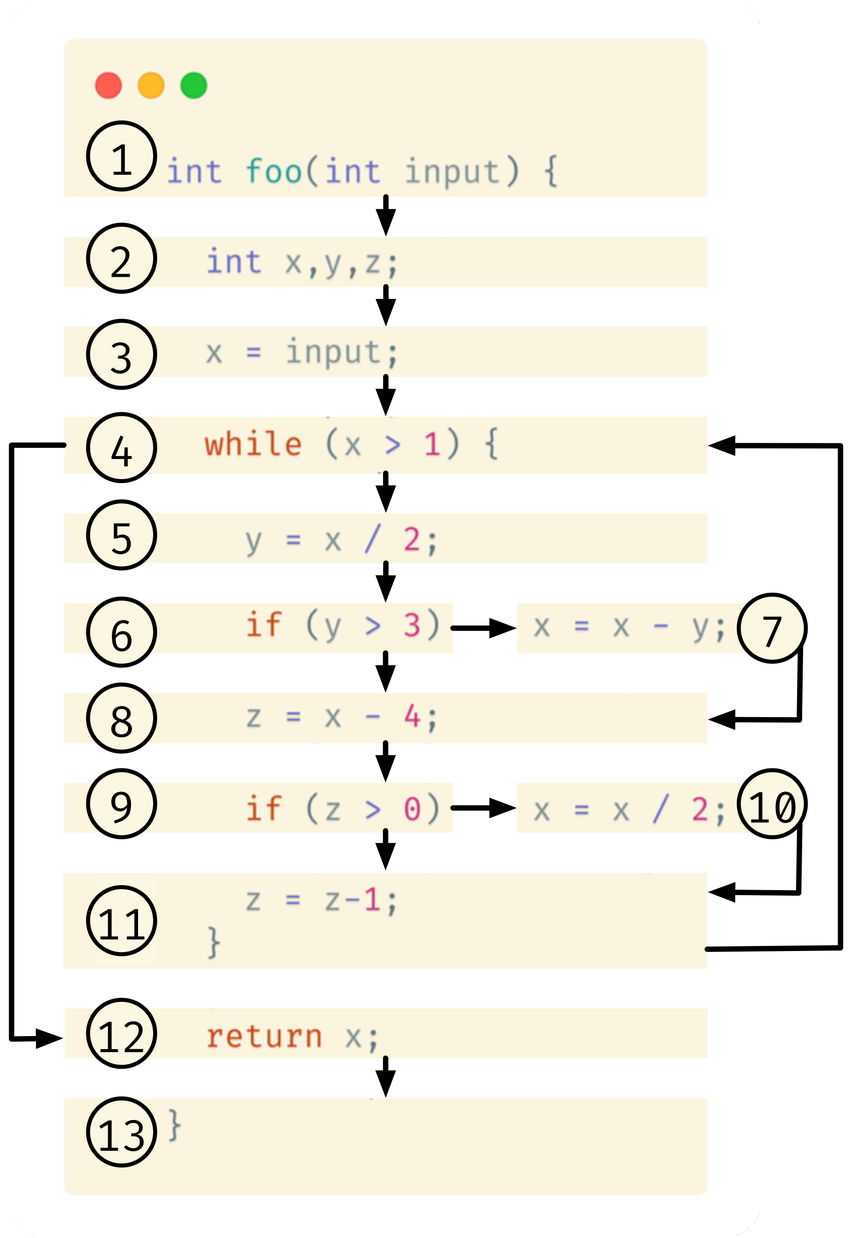


Definition of def-use chains

- The def-use chains for a variable contain all definitions and all possible uses

For variable x:

```
[x, x = input, while(x>1)]
[x, x = input, x = x - y ]
[x, x = input, x = x / 2 ]
[x, x = input, return x  ]
[x, x = x - y, while(x>1)]
[x, x = x - y, x = x - y ]
[x, x = x - y, x = x / 2 ]
[x, x = x - y, return x  ]
[x, x = x / 2, while(x>1)]
[x, x = x / 2, x = x - y ]
[x, x = x / 2, x = x / 2 ]
[x, x = x / 2, return x  ]
```



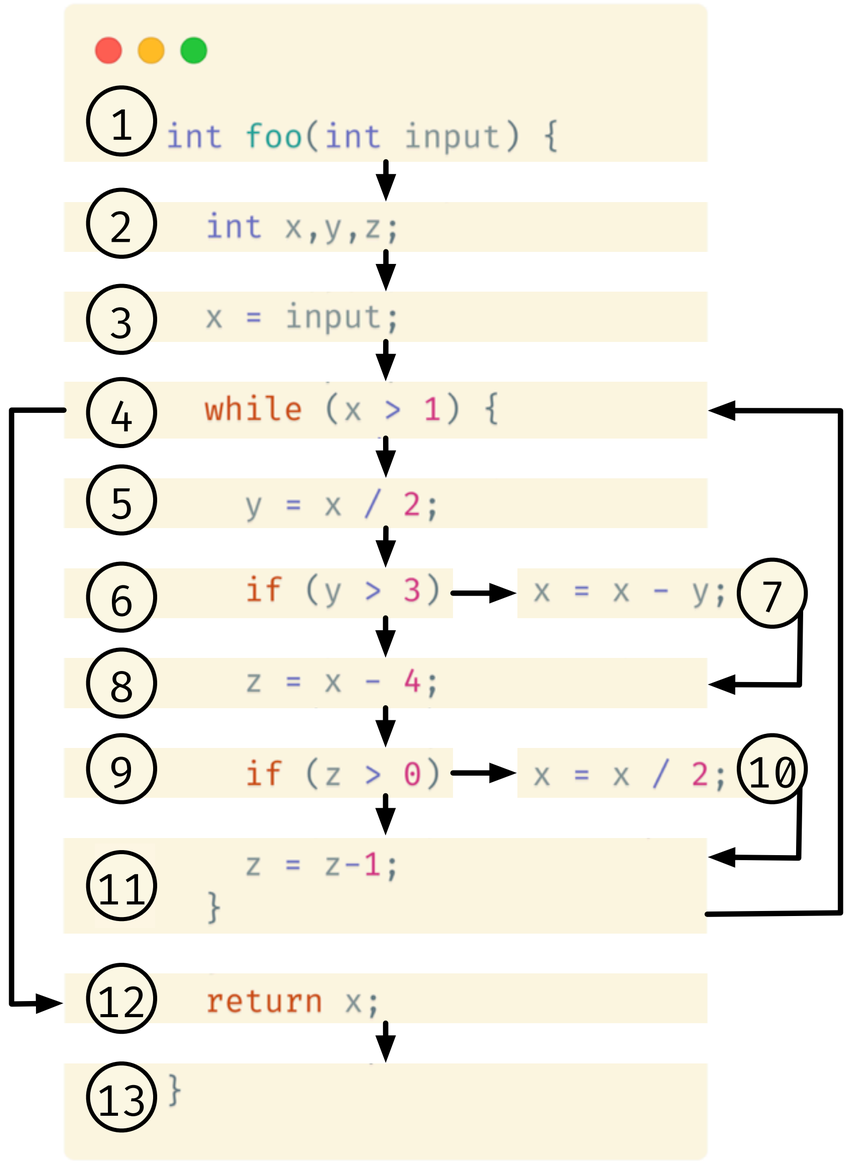
Definition of Liveness

- **Definition:**
  A variable v is live *before* a CFG node **n** if

  (1) $v \in$ **use$_{var}$(n)**, or:

  (2) $\exists$ a directed path from **n** to a node in **use$_{nodes}$(v)**, and that path does not go through a node in **def$_{nodes}$(v).**

- Examples:
  Is x life before **n** = 5? Yes, $x \in \{x\}$ = **use$_{var}$(5)**
  Is z life before **n** = 5? No, we first hit a def at 8



Computing Liveness per node

Three cases to consider for a node:

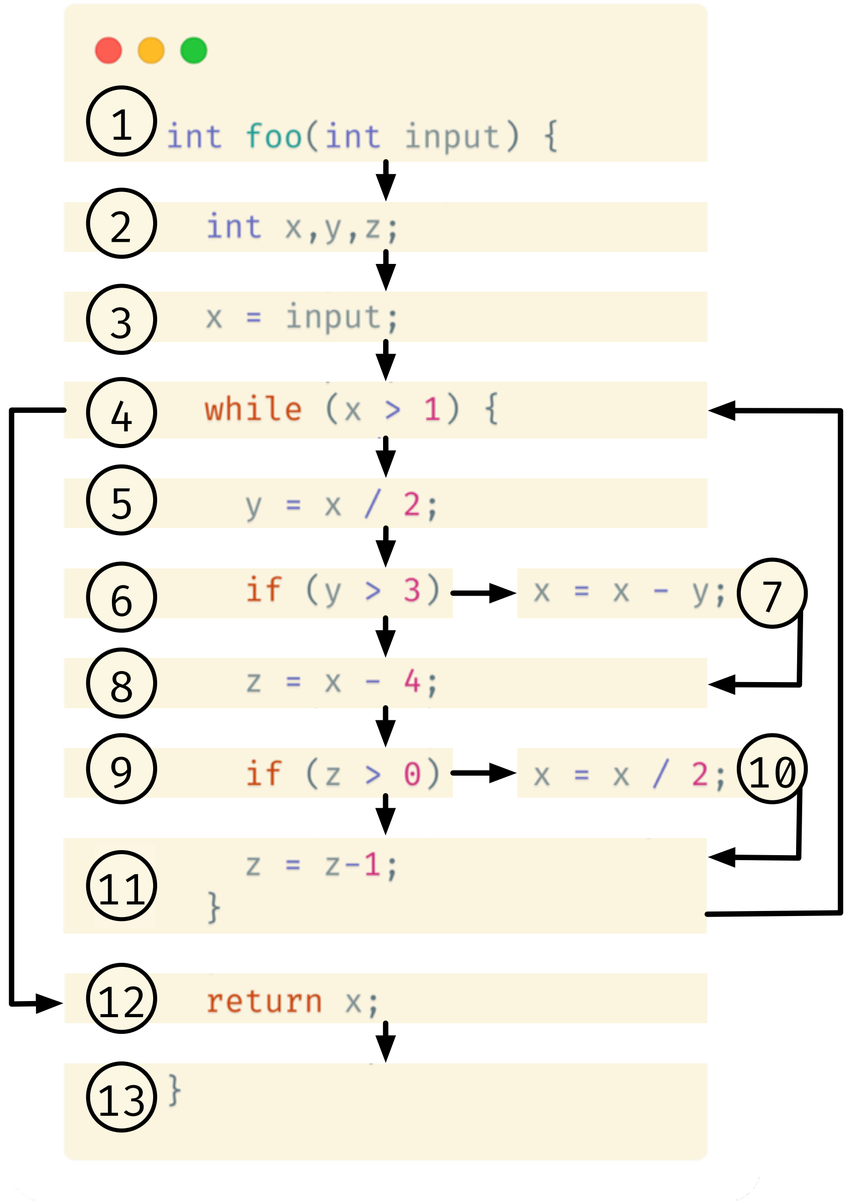
(1) $\mathbf{use_{var}(n)}$

(2) if a variable is live at a successor of **n** then we consider it a *candidate* for **n**:

$$\texttt{candidates[n]} = \bigcup_{s \in succ(n)} \texttt{live[s]}$$

(3) if a variable is a candidate at **n** and not in $\mathbf{def_{var}(n)}$ then it is live at **n**:

$$\texttt{candidates[n]} - \mathbf{def_{var}(n)}$$



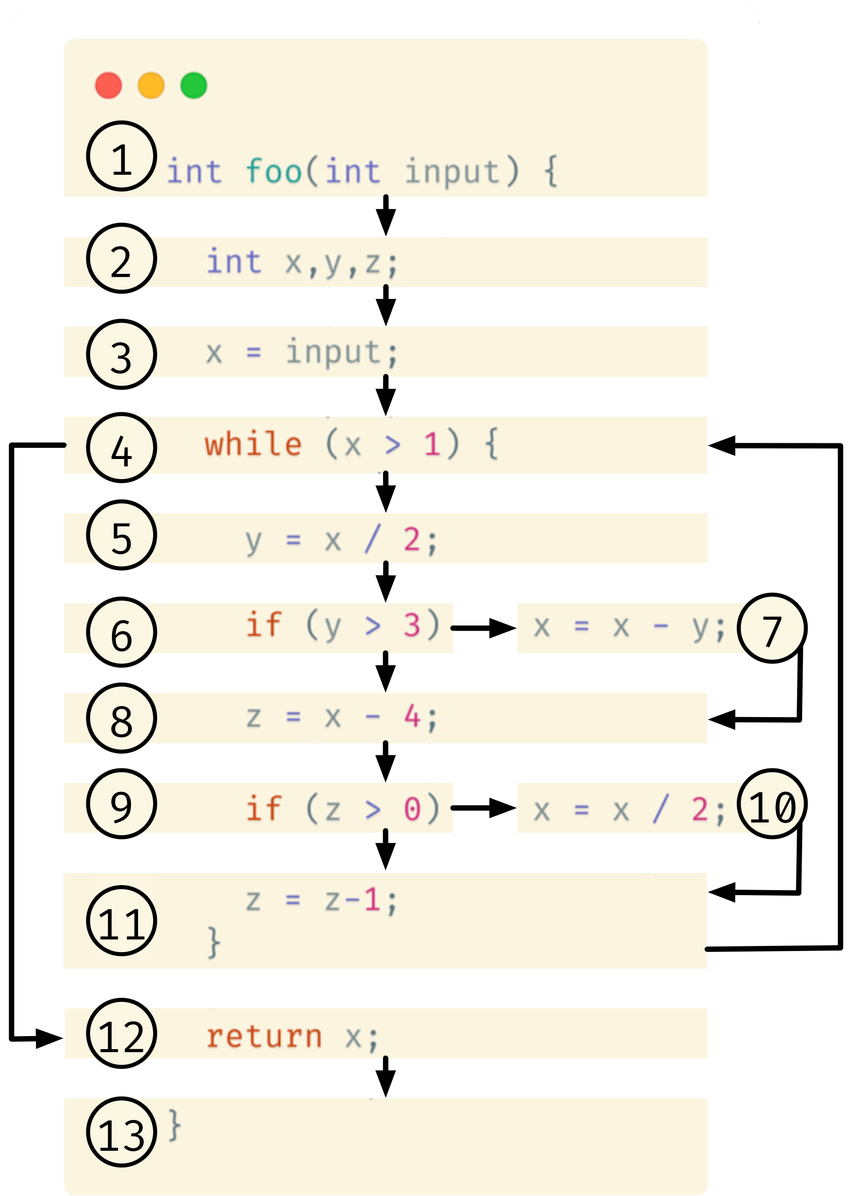
Iterative fixpoint algorithm

```
for each node n in CFG:
  live[n] = {}; candidates[n] = {};

repeat for each node n in CFG in reverse top. sorted order:
  live'[n]       = live[n];
  candidates'[n] = candidates[n];

  candidates[n]  = ⋃_{s∈succ(n)} live[s];
  live[n]        = use_var(n) ⋃ (candidates[n] - def_var(n));
until live'[n]      == live[n]      and
      candidate'[n] == candidate[n] for all n
```

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}		
12	{x}	{}		
11	{z}	{z}		
10	{x}	{x}		
9	{z}	{}		
8	{x}	{z}		
7	{x,y}	{x}		
6	{y}	{}		
5	{x}	{y}		
4	{x}	{}		
3	{}	{x}		
2	{}	{}		
1	{}	{}		



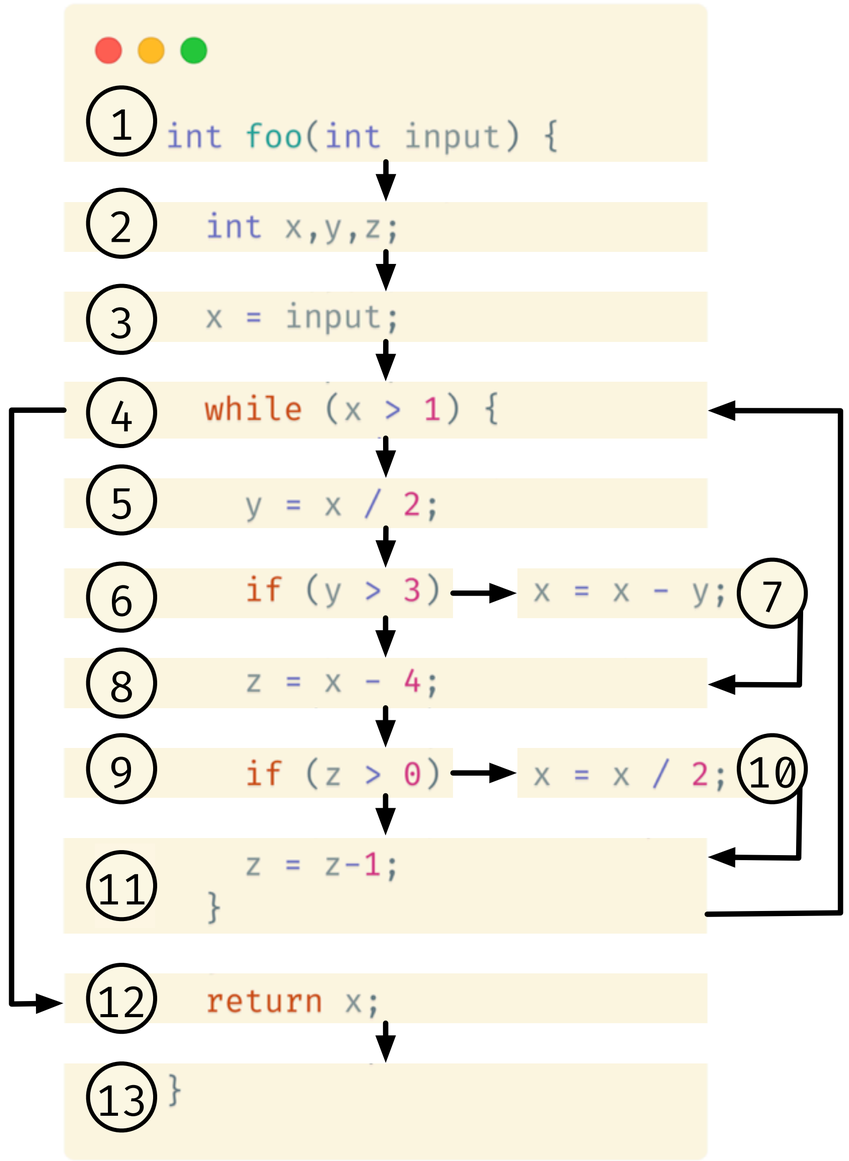
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{ }
11	{z}	{z}	{ }	{ }
10	{x}	{x}	{ }	{ }
9	{z}	{ }	{ }	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



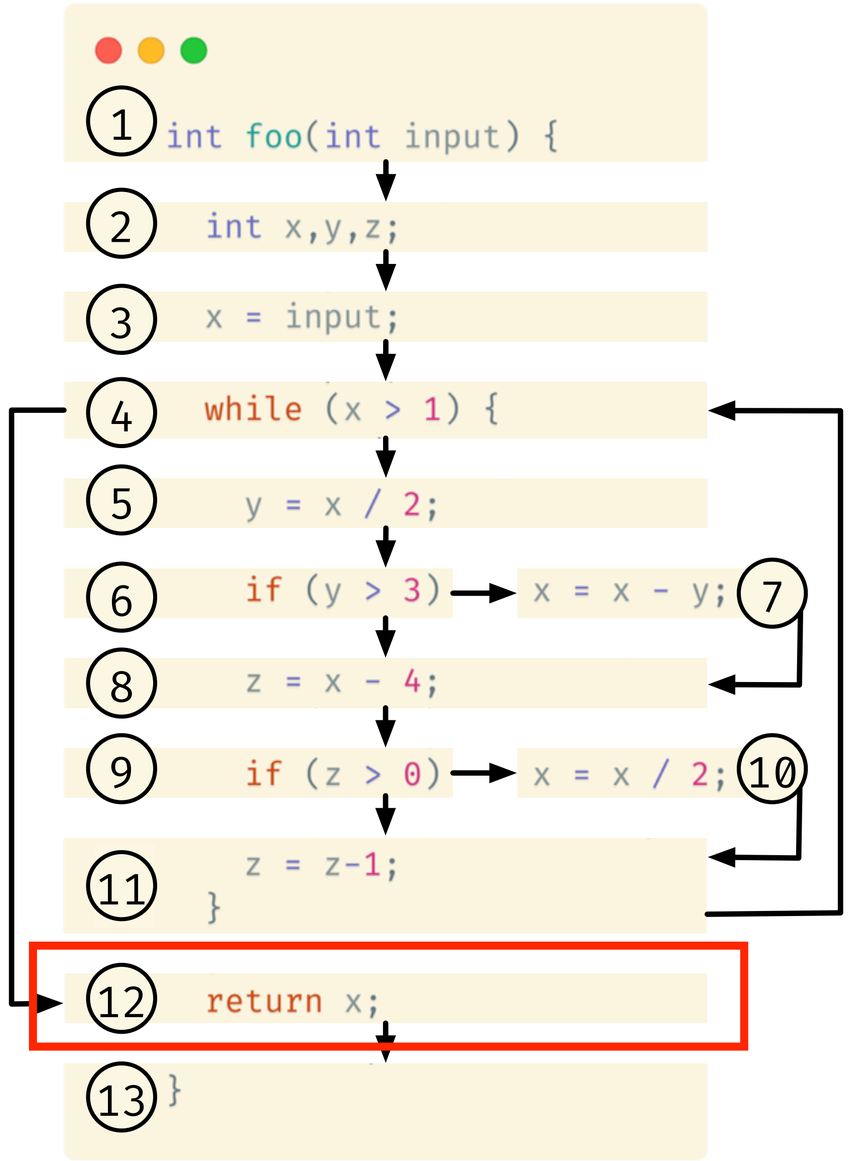
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	**{ }**	{ }
11	{z}	{z}	{ }	{ }
10	{x}	{x}	{ }	{ }
9	{z}	{ }	{ }	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



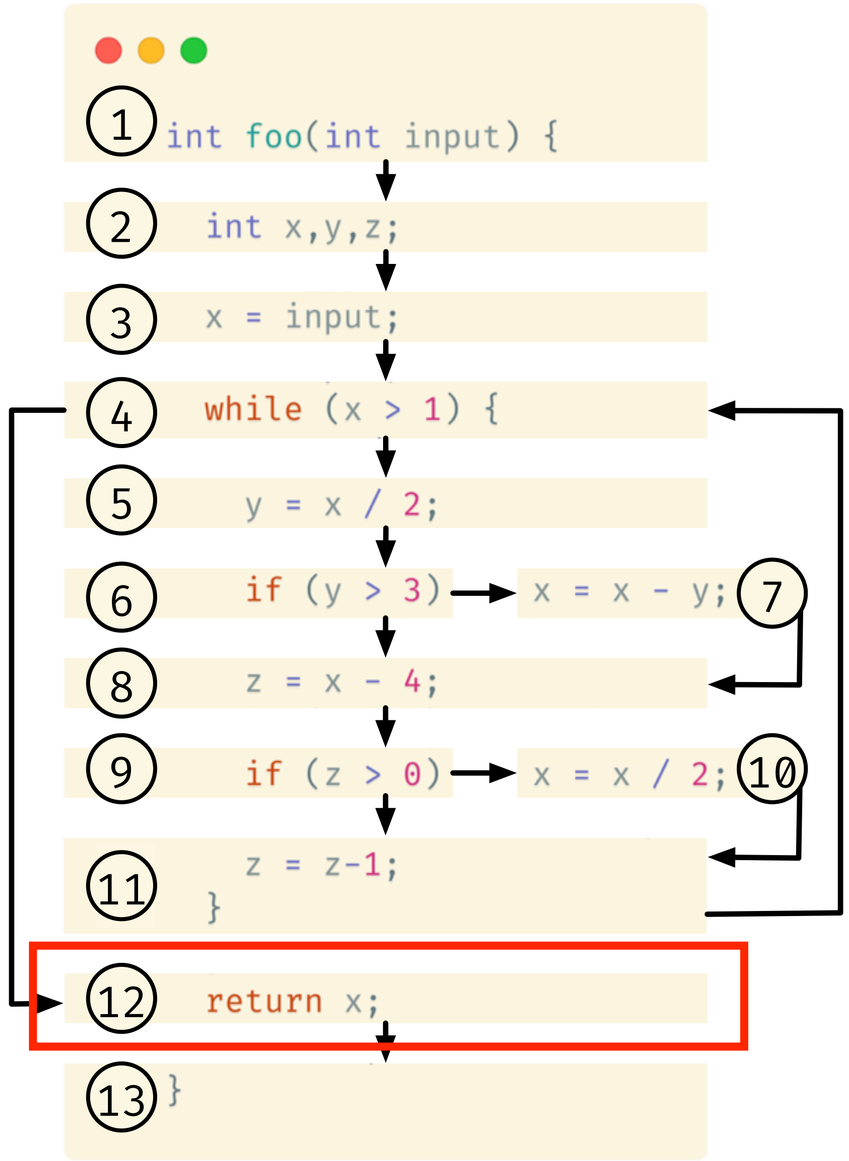
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{}
10	{x}	{x}	{}	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x,y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}



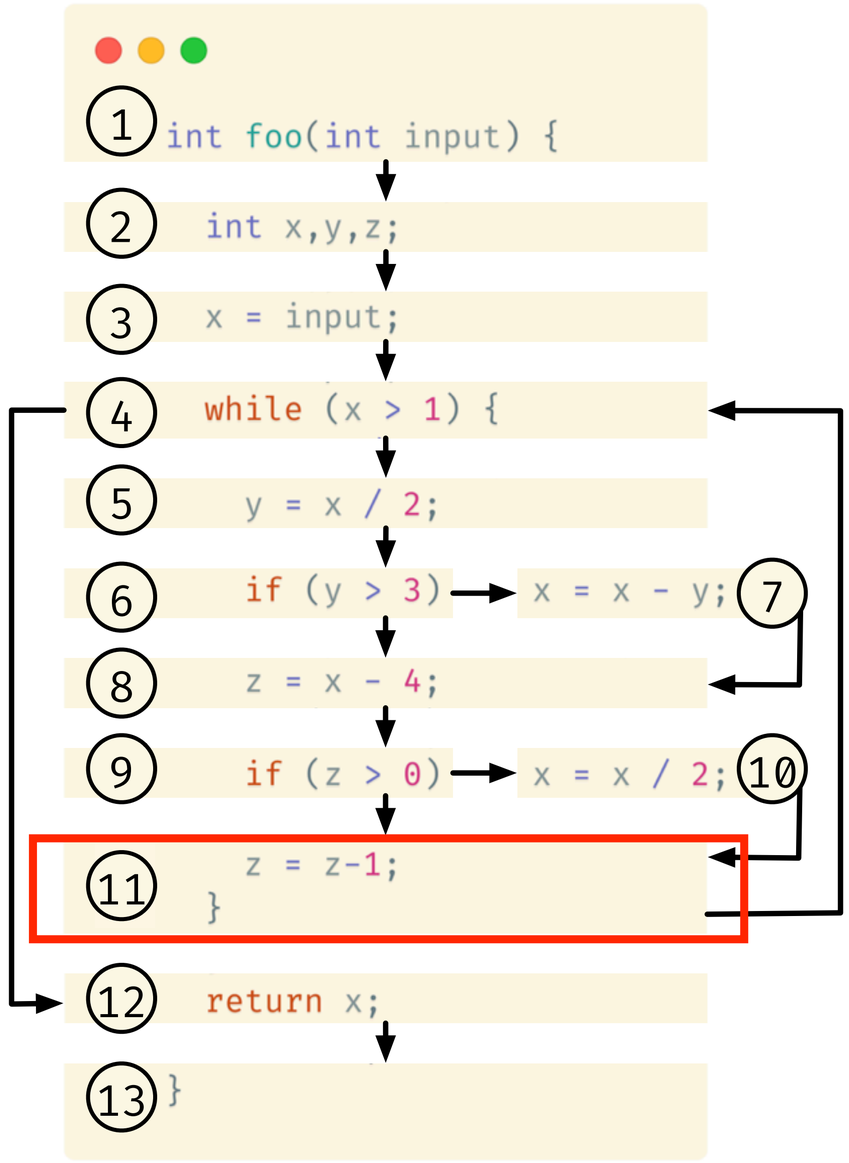
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	**{ }**	{ }
10	{x}	{x}	{ }	{ }
9	{z}	{ }	{ }	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



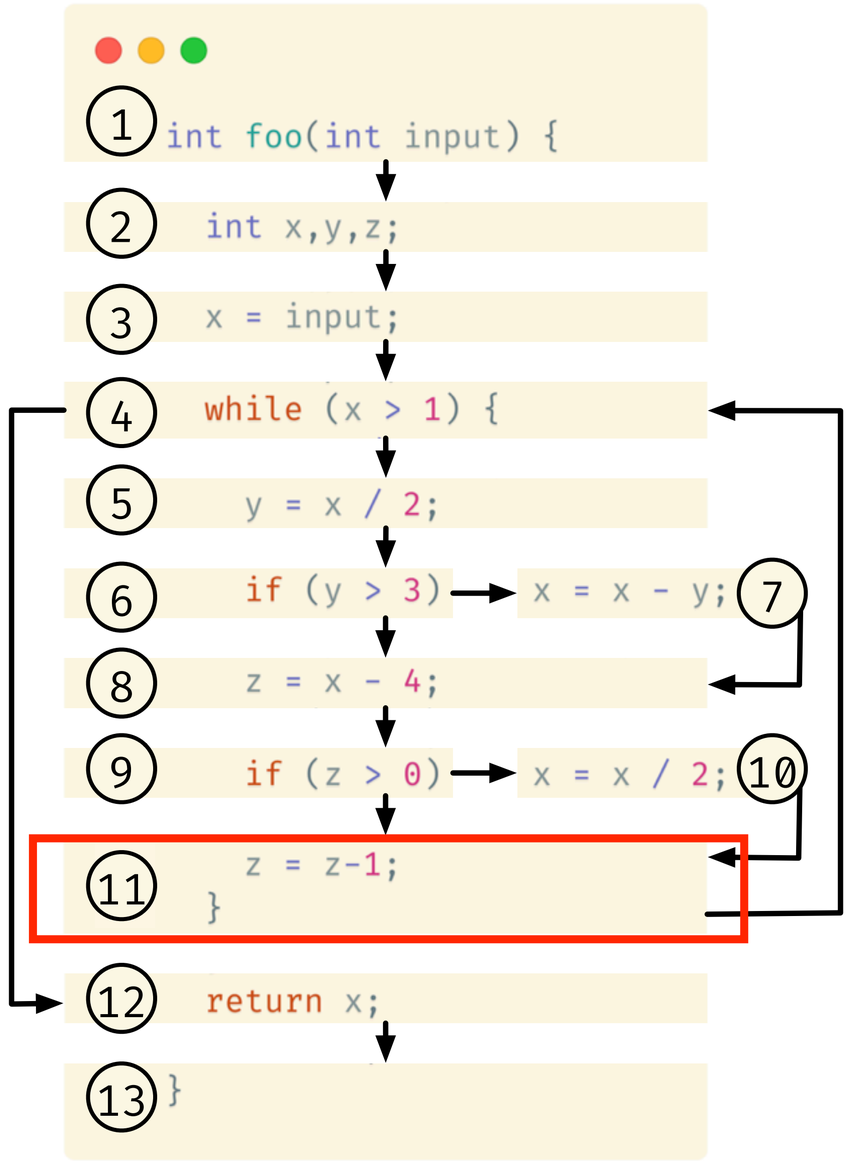
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	**{ }**	**{z}**
10	{x}	{x}	{ }	{ }
9	{z}	{ }	{ }	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



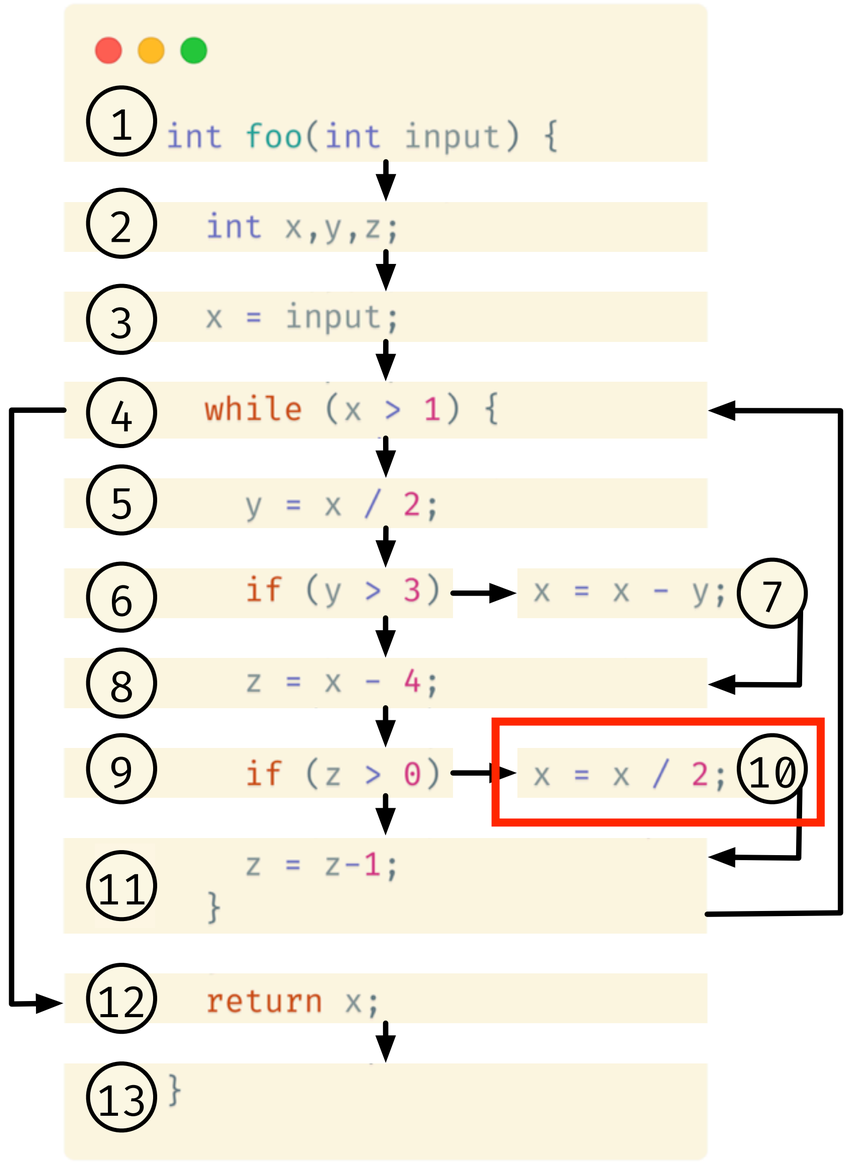
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	**{z}**	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x,y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}



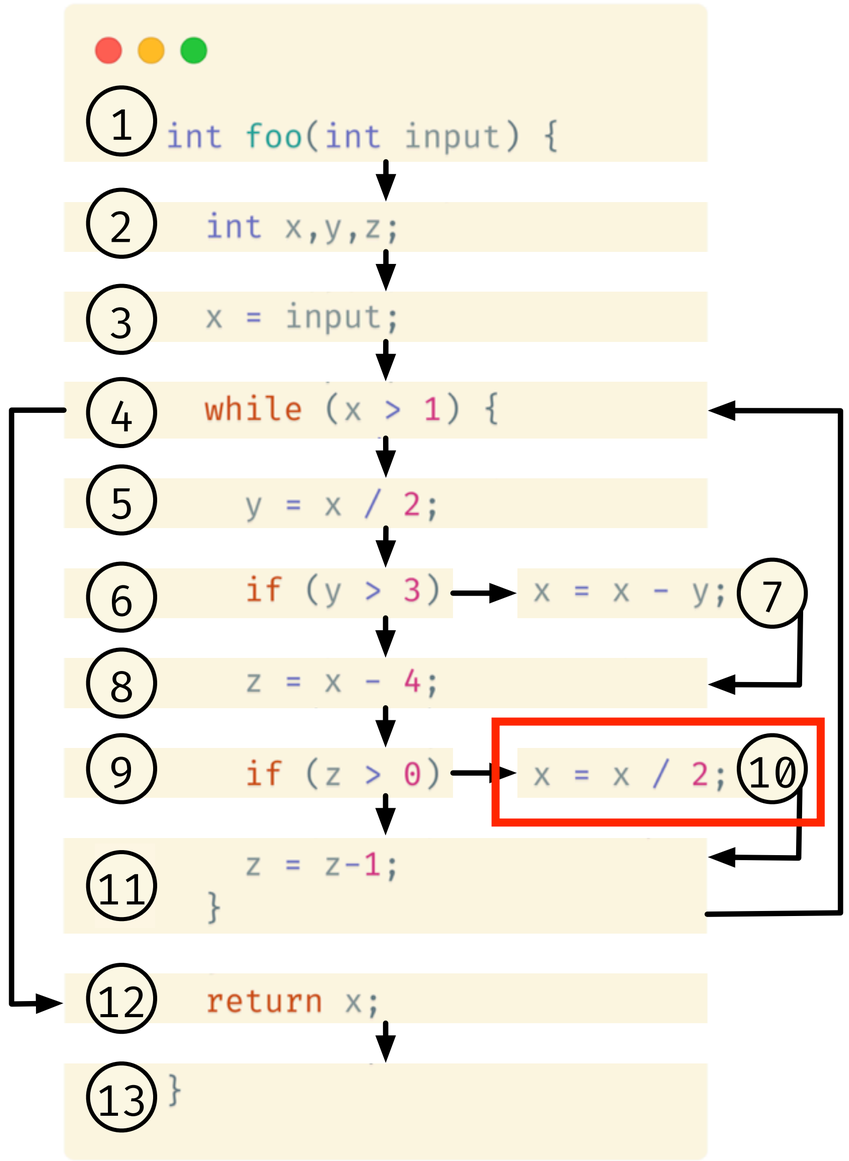
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	**{z}**	**{x,z}**
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x,y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}



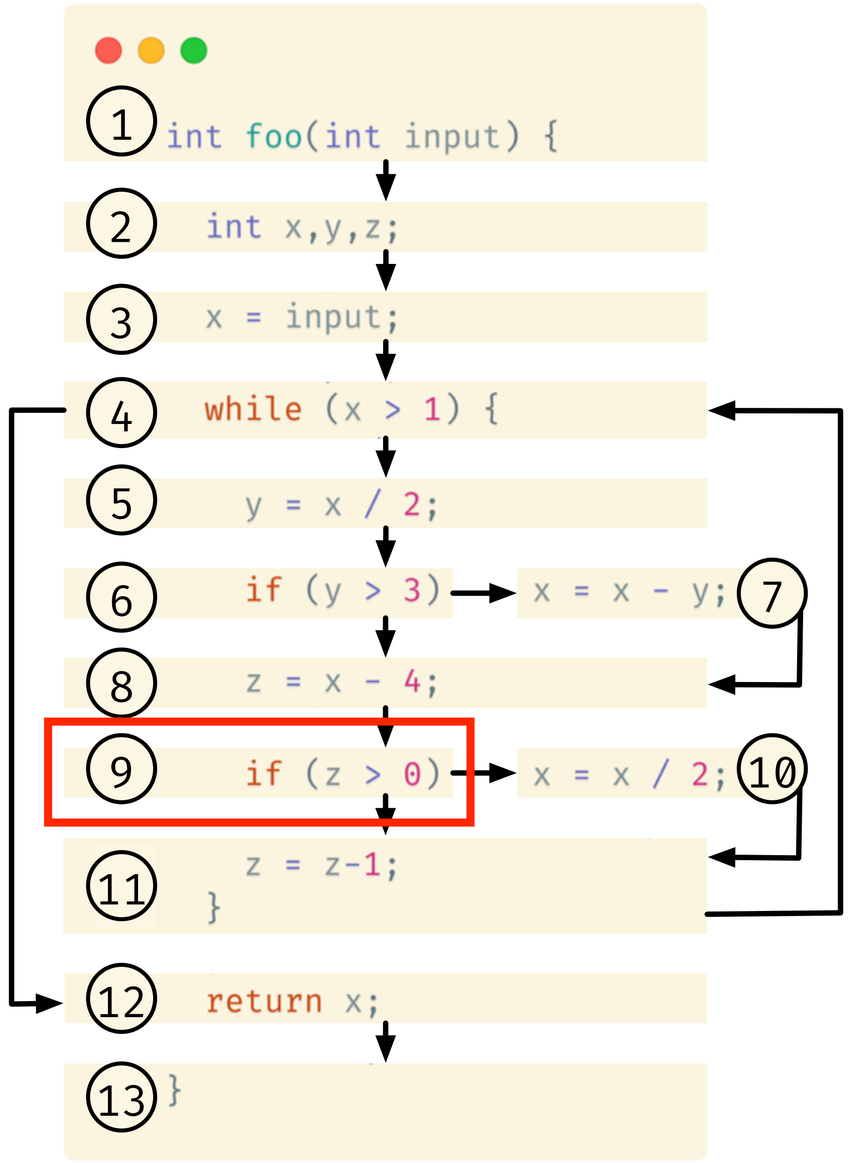
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	**{x,z}**	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



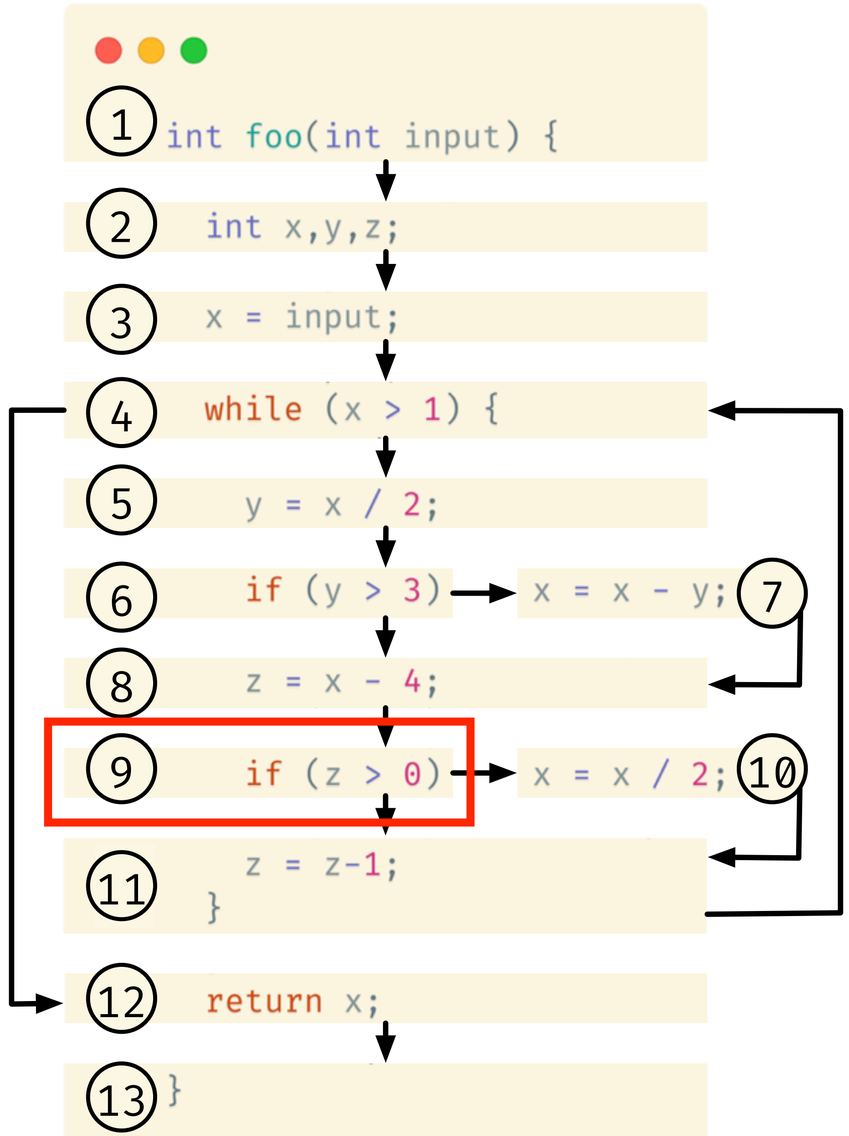
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	**{x,z}**	**{x,z}**
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



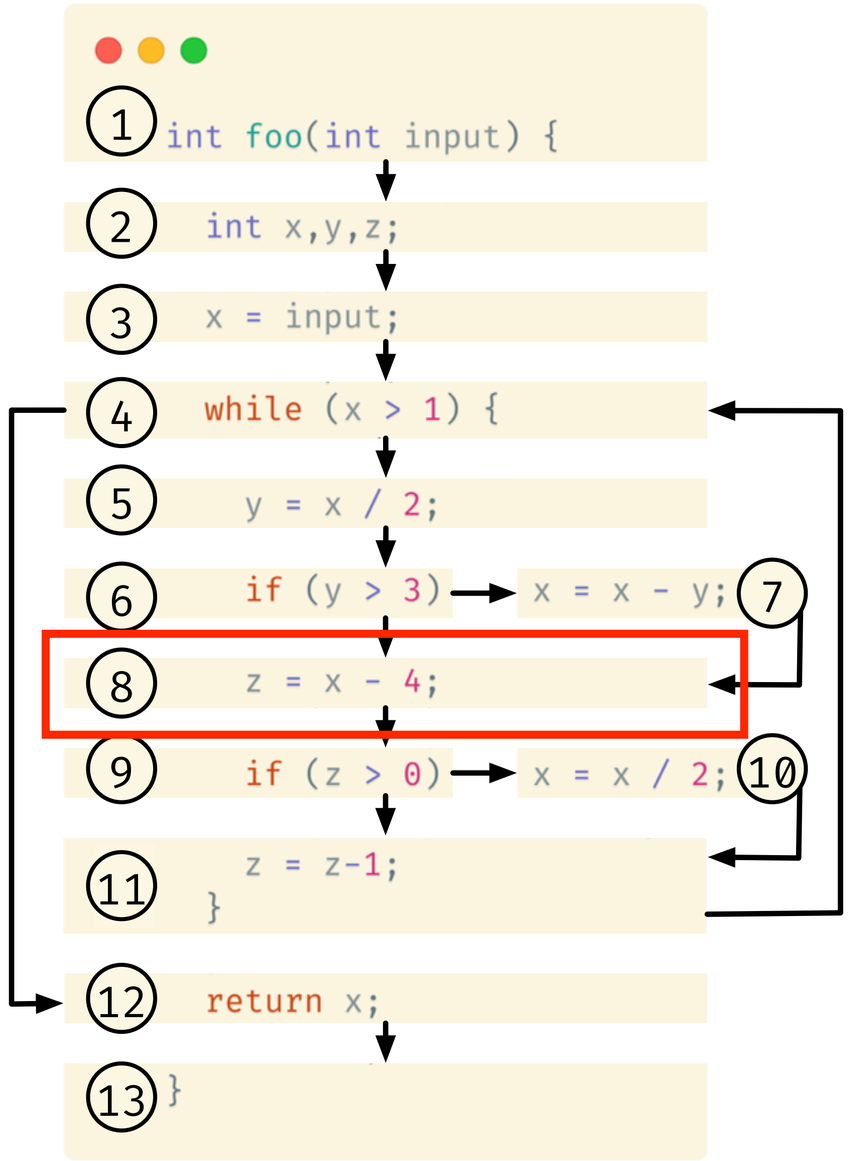
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	**{x,z}**	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



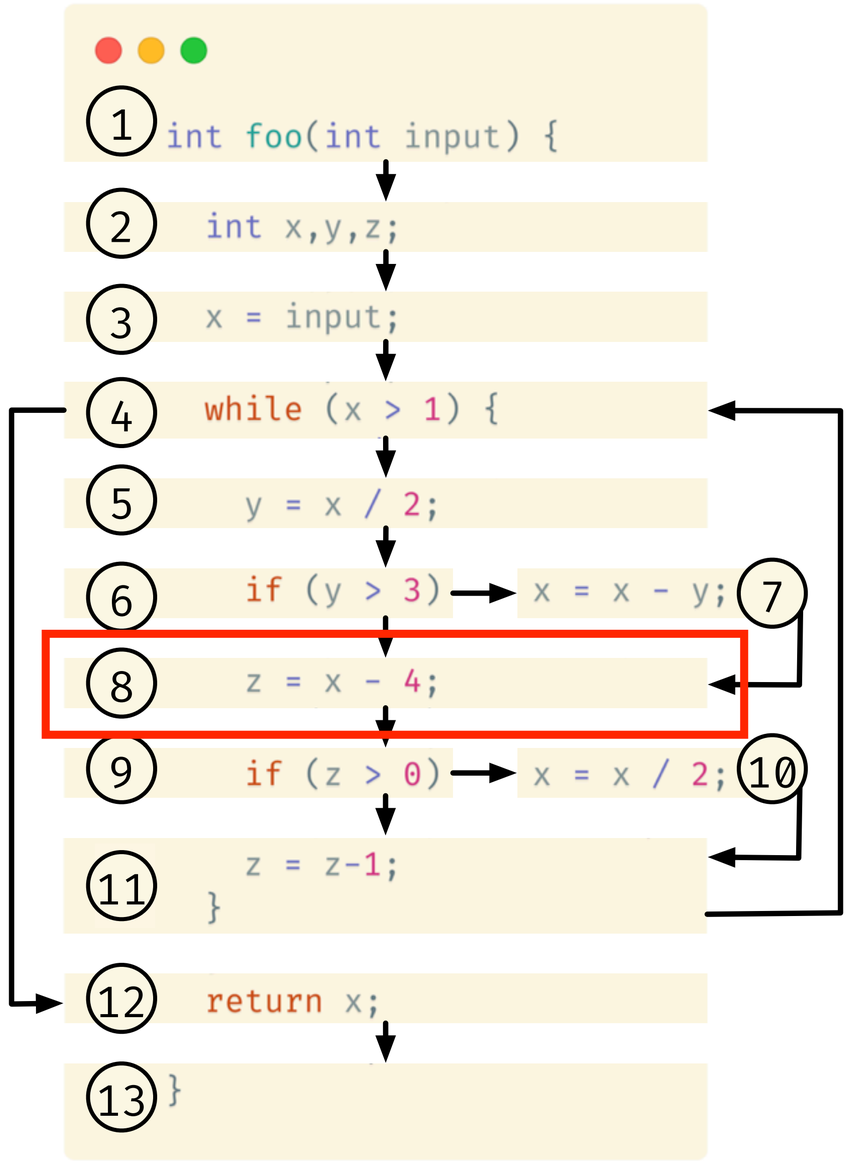
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	**{x,z}**	**{x}**
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



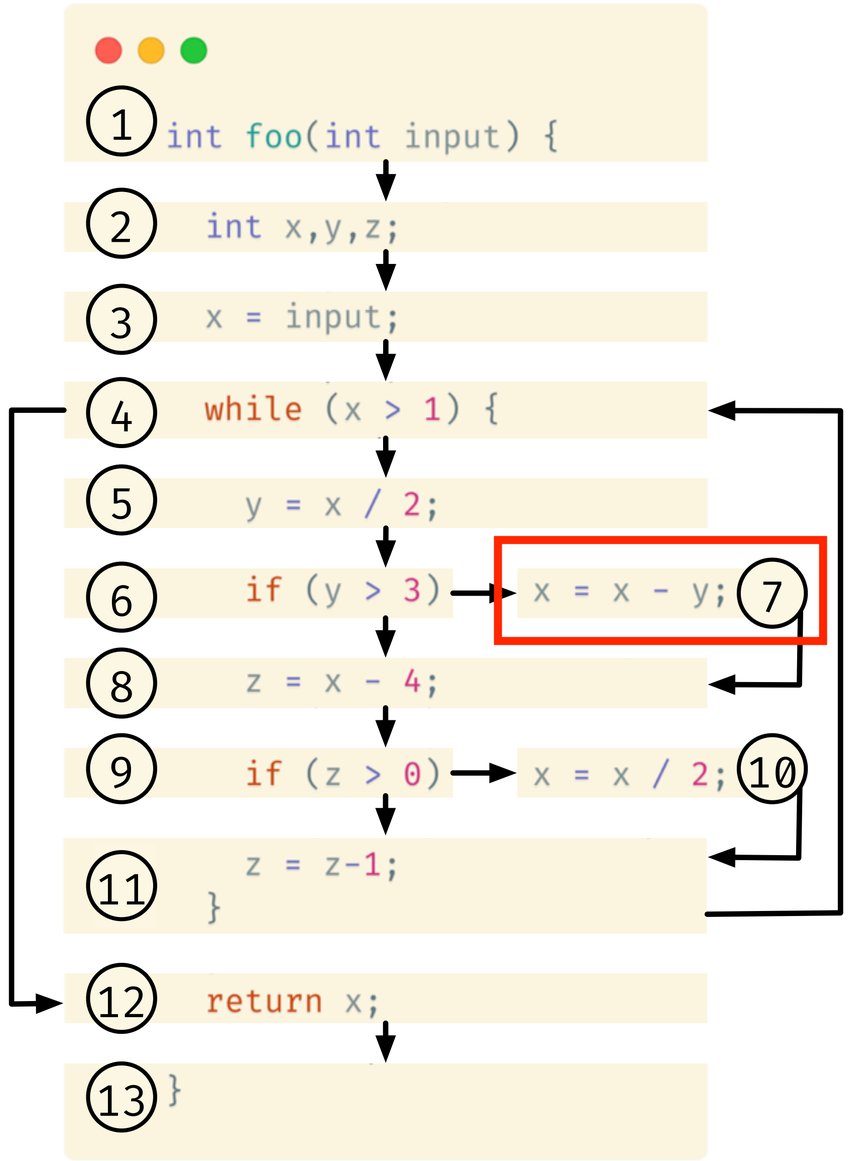
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	**{x}**	**{x,y}**
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



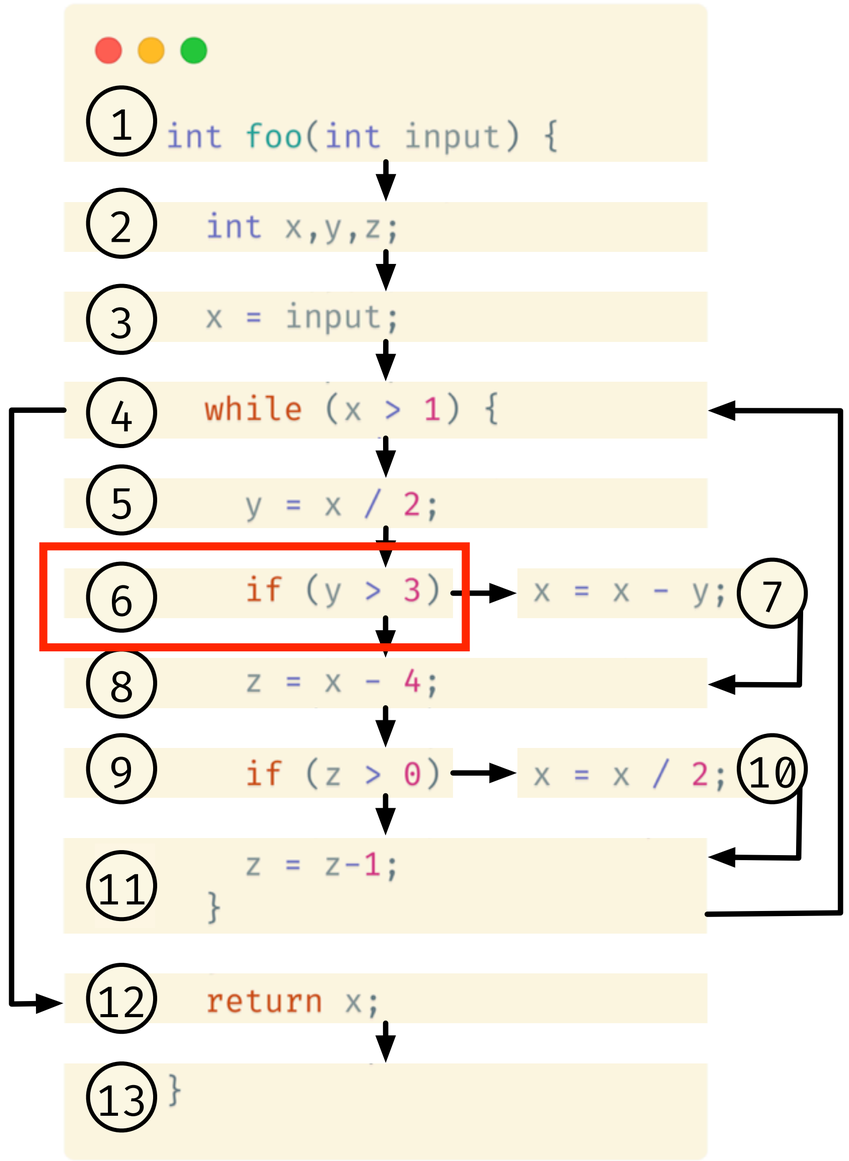
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{}	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}
6	{y}	{}	**{x,y}**	**{x,y}**
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}



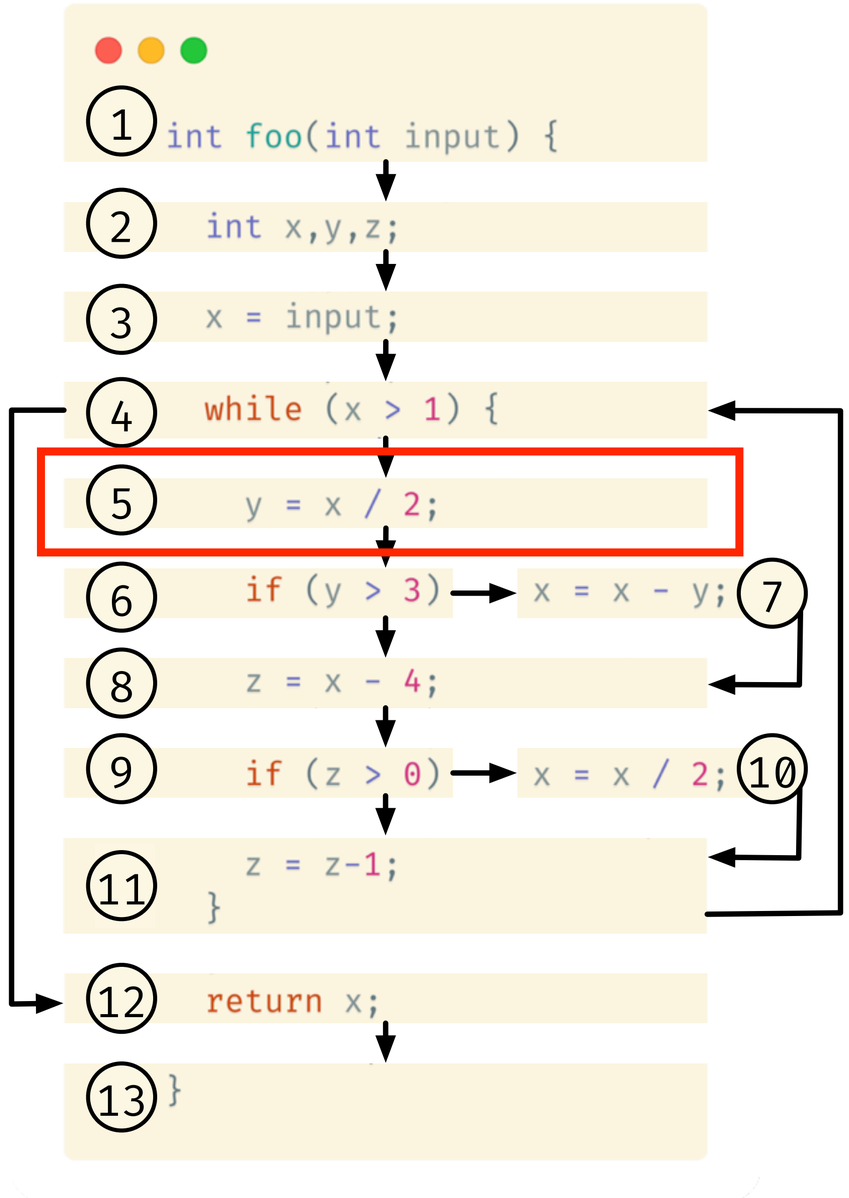
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{}	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}
6	{y}	{}	{x,y}	{x,y}
5	{x}	{y}	**{x,y}**	**{x}**
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}



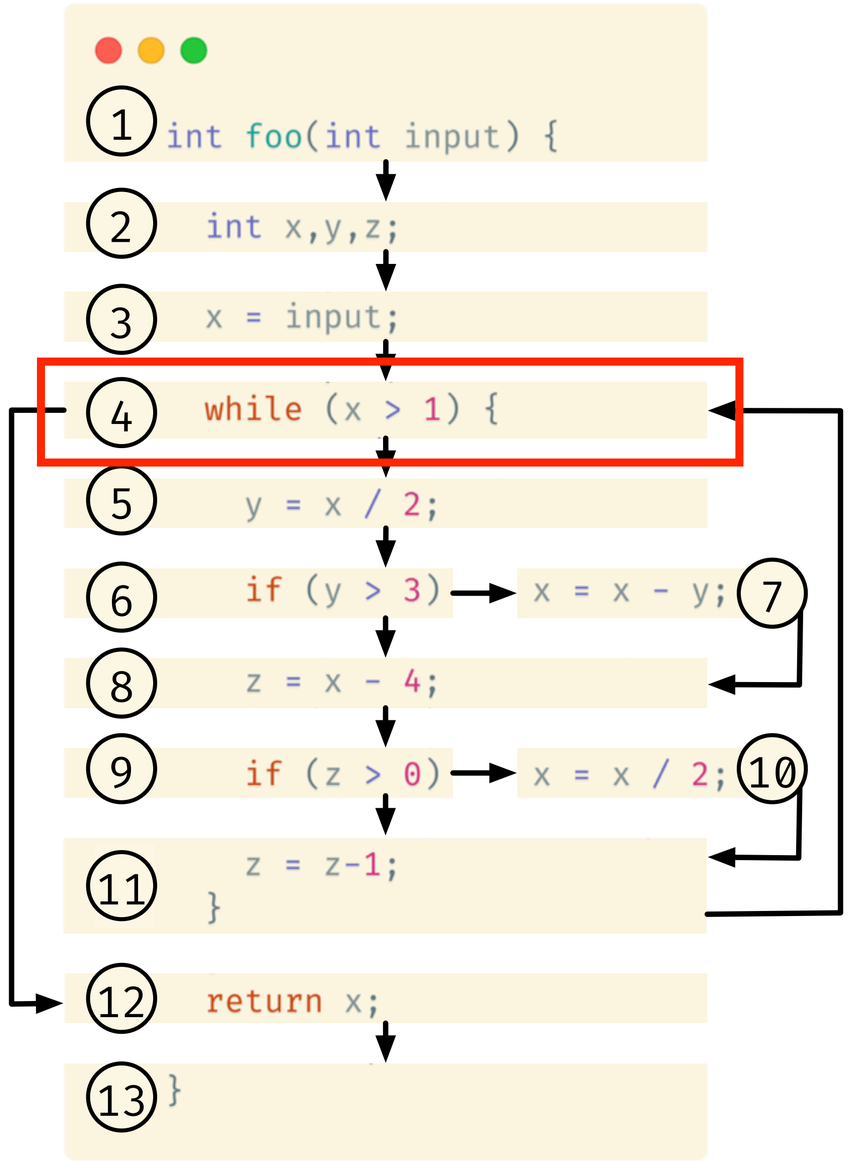
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}
6	{y}	{ }	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}
4	{x}	{ }	**{x}**	**{x}**
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



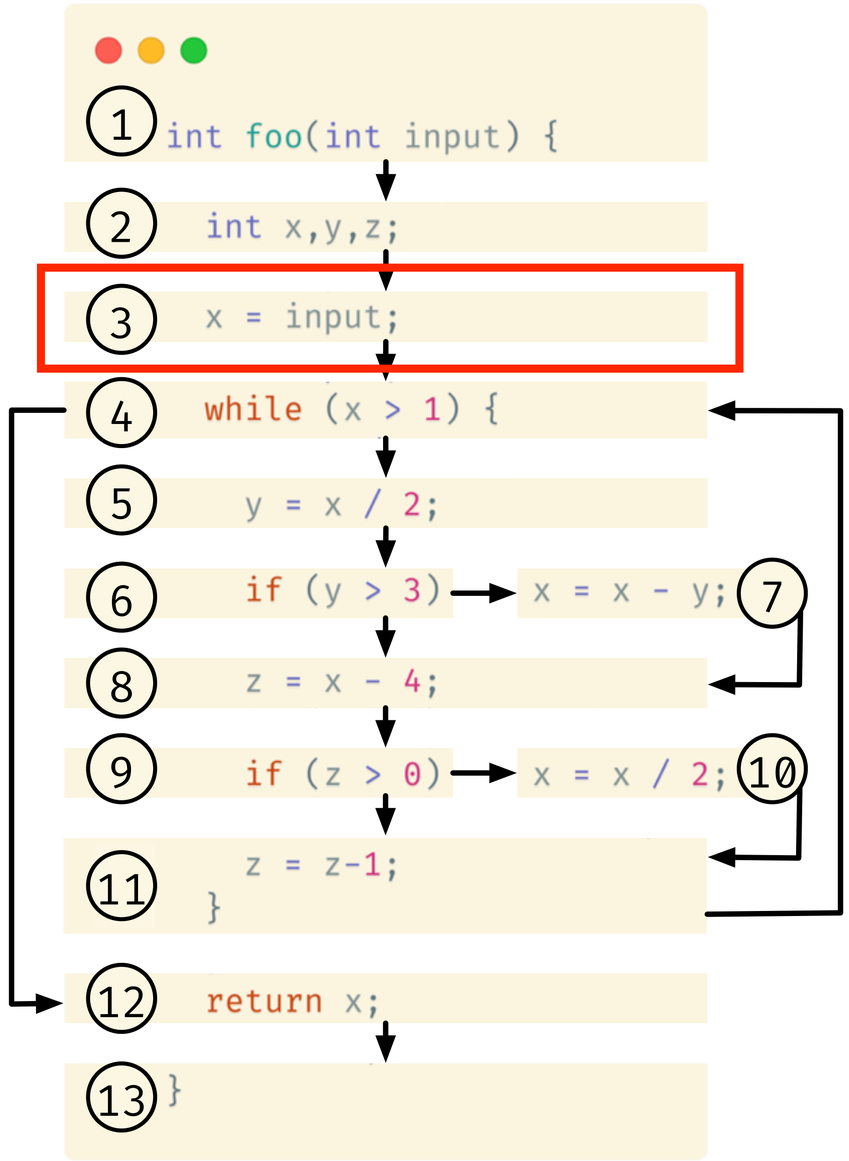
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}
6	{y}	{ }	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}
4	{x}	{ }	{x}	{x}
3	{ }	{x}	**{x}**	**{ }**
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



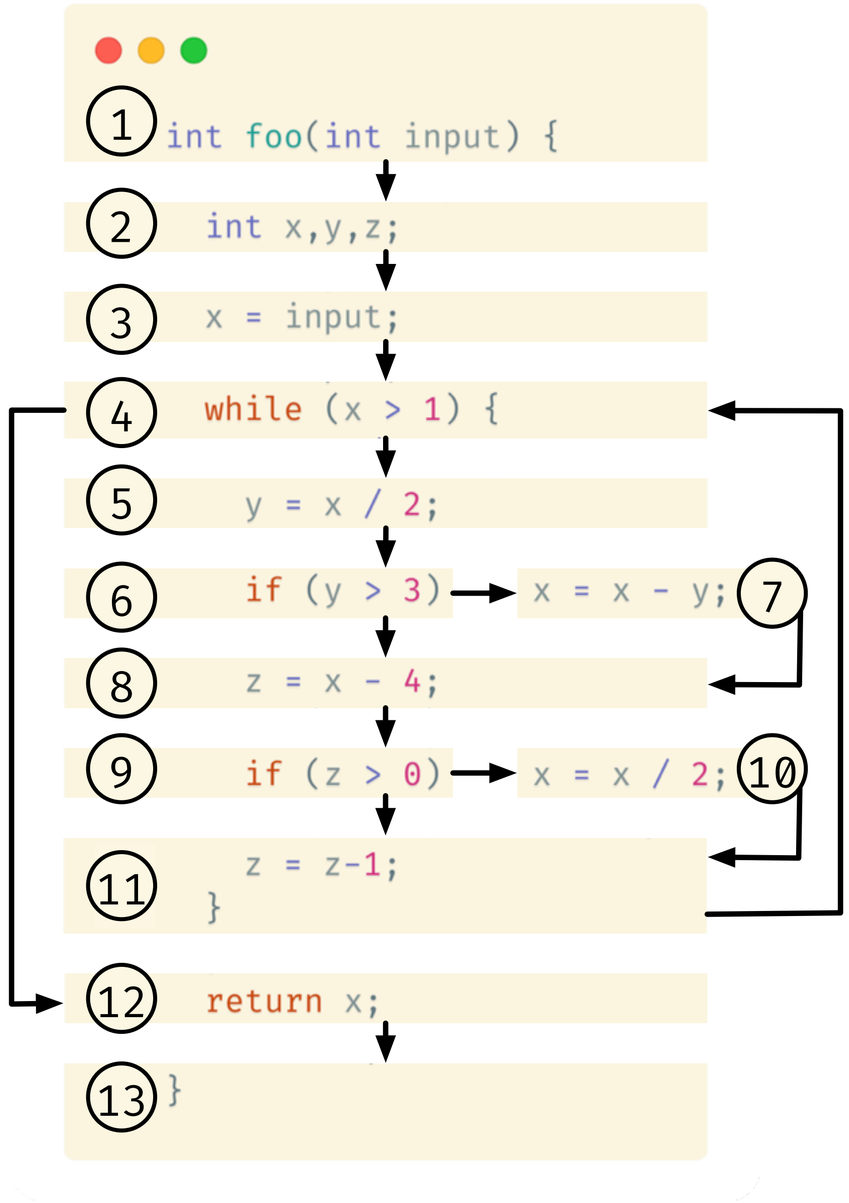
$$candidates[n] = \bigcup_{s \in succ(n)} live[s];$$

$$live[n] = use_{var}(n) \cup (candidates[n] - def_{var}(n));$$

Liveness Analysis Example

1st

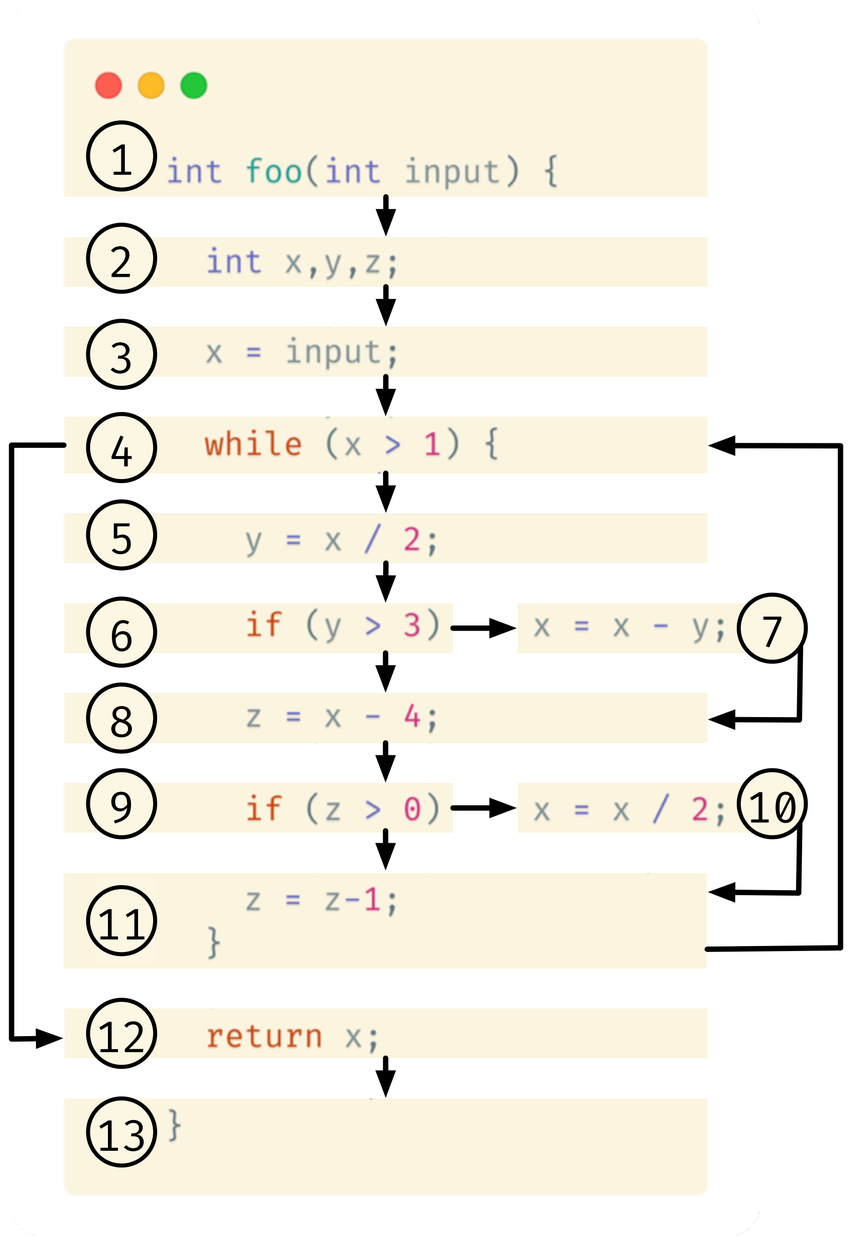
Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{x,z}
9	{z}	{ }	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}
6	{y}	{ }	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}
4	{x}	{ }	{x}	{x}
3	{ }	{x}	{x}	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }



Completed fist iteration

Liveness Analysis Example

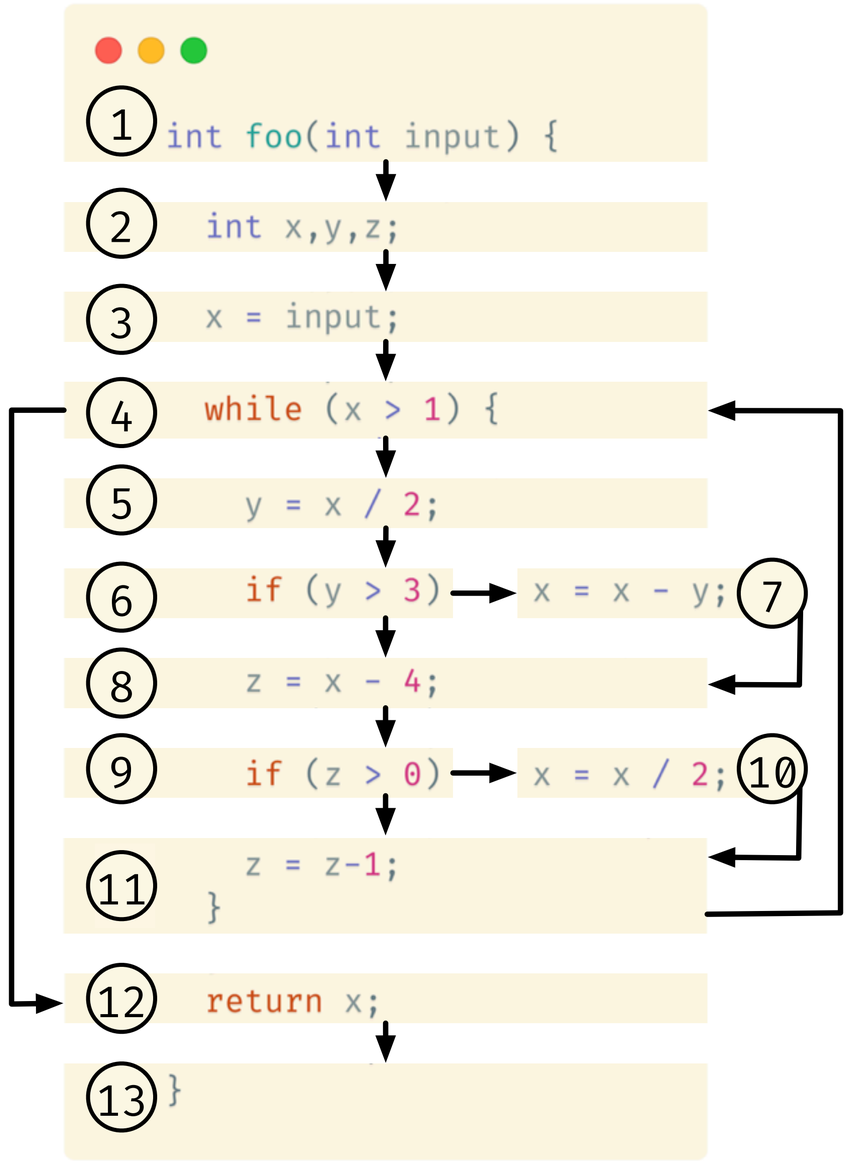
			1st		2nd	
Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]	candidate[n]	live[n]
13	{}	{}	{}	{}	{}	{}
12	{x}	{}	{}	{x}	{}	{x}
11	{z}	{z}	{}	{z}	{x}	{x,z}
10	{x}	{x}	{z}	{x,z}	{x,z}	{x,z}
9	{z}	{}	{x,z}	{x,z}	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}	{x}	{x,y}
6	{y}	{}	{x,y}	{x,y}	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}	{x,y}	{x}
4	{x}	{}	{x}	{x}	{x}	{x}
3	{}	{x}	{x}	{}	{x}	{}
2	{}	{}	{}	{}	{}	{}
1	{}	{}	{}	{}	{}	{}



Completed second iteration

Liveness Analysis Example

			1st		2nd		3rd	
Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]	candidate[n]	live[n]	candidate[n]	live[n]
13	{}	{}	{}	{}	{}	{}	{}	{}
12	{x}	{}	{}	{x}	{}	{x}	{}	{x}
11	{z}	{z}	{}	{z}	{x}	{x,z}	{x}	{x,z}
10	{x}	{x}	{z}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}
9	{z}	{}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}	{x,z}	{x}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}	{x}	{x,y}	{x}	{x,y}
6	{y}	{}	{x,y}	{x,y}	{x,y}	{x,y}	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}	{x,y}	{x}	{x,y}	{x}
4	{x}	{}	{x}	{x}	{x}	{x}	{x}	{x}
3	{}	{x}	{x}	{}	{x}	{}	{x}	{}
2	{}	{}	{}	{}	{}	{}	{}	{}
1	{}	{}	{}	{}	{}	{}	{}	{}



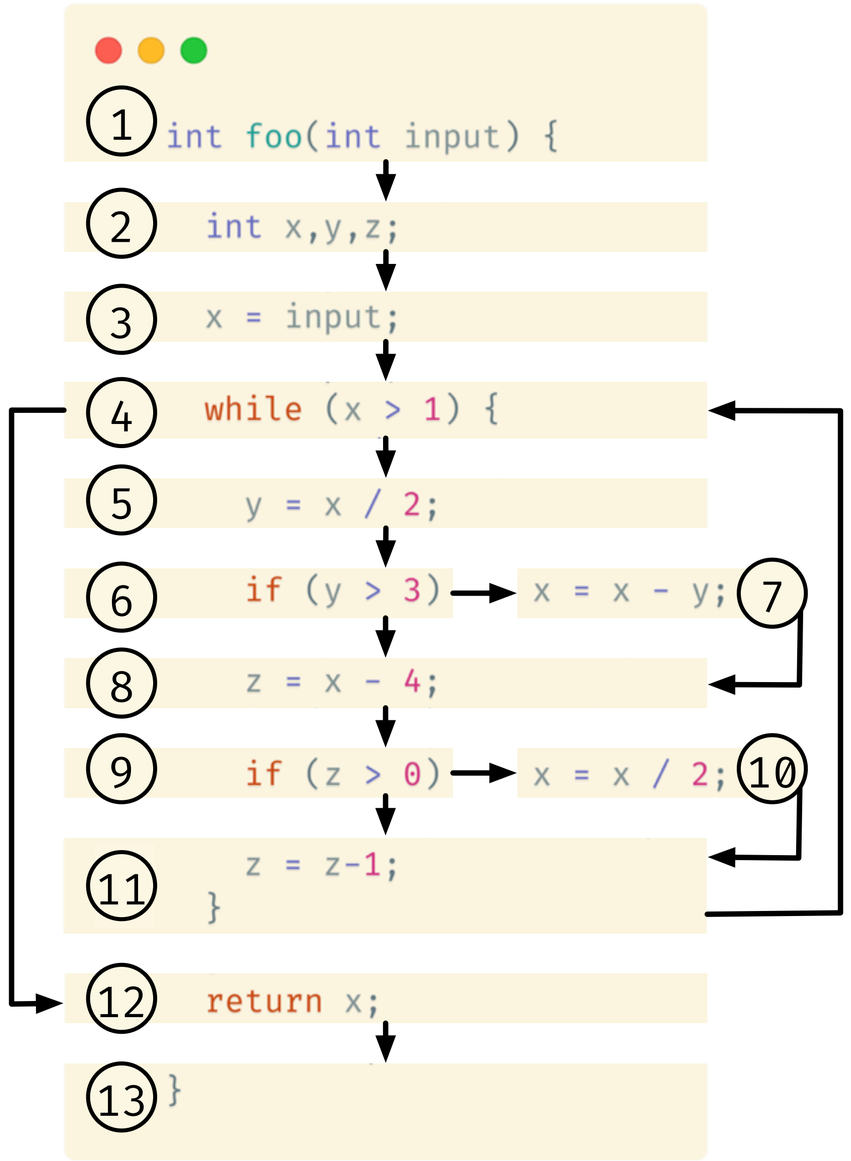
No changes: fixpoint reached

Liveness Analysis Example

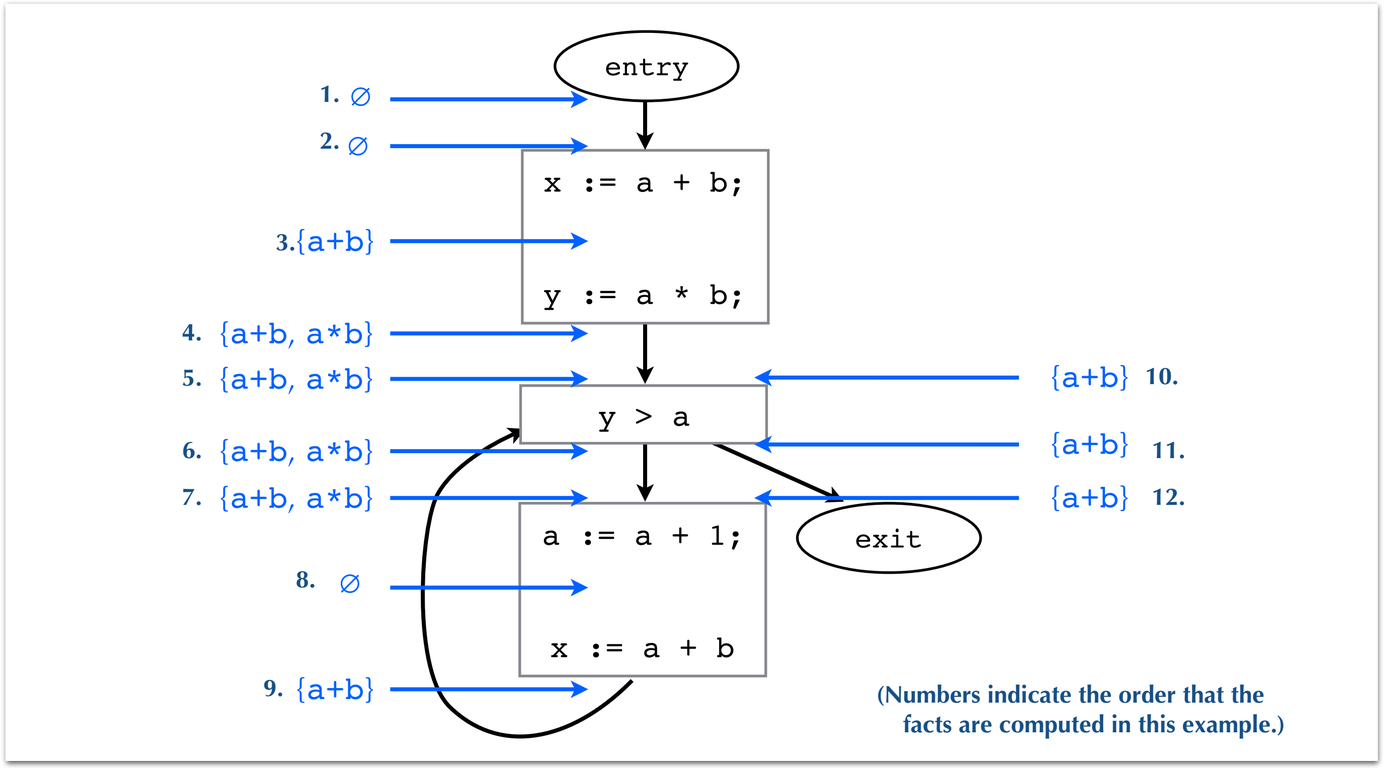
			1st		2nd		3rd	
Node n	$use_{var}(n)$	$def_{var}(n)$	candidate[n]	live[n]	candidate[n]	live[n]	candidate[n]	live[n]
13	{}	{}	{}	{}	{}	{}	{}	{}
12	{x}	{}	{}	{x}	{}	{x}	{}	{x}
11	{z}	{z}	{}	{z}	{x}	{x,z}	{x}	{x,z}
10	{x}	{x}	{z}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}
9	{z}	{}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}	{x,z}
8	{x}	{z}	{x,z}	{x}	{x,z}	{x}	{x,z}	{x}
7	{x,y}	{x}	{x}	{x,y}	{x}	{x,y}	{x}	{x,y}
6	{y}	{}	{x,y}	{x,y}	{x,y}	{x,y}	{x,y}	{x,y}
5	{x}	{y}	{x,y}	{x}	{x,y}	{x}	{x,y}	{x}
4	{x}	{}	{x}	{x}	{x}	{x}	{x}	{x}
3	{}	{x}	{x}	{}	{x}	{}	{x}	{}
2	{}	{}	{}	{}	{}	{}	{}	{}
1	{}	{}	{}	{}	{}	{}	{}	{}

y and z are never live together

No changes: fixpoint reached



Available Expressions



Stmt	Gen	Kill
x := a+b	{a+b}	
y := a*b	{a*b}	
y > a		
a := a+1	{a+1}	{a+1, a+b, a*b}

add to set of available expressions

remove from set of available expressions

Stmt	Gen	Kill
x := v	{v}	{e | x in e}

Prof. Stephen Chong Compiler Course at Harvard https://www.seas.harvard.edu/courses/cs153/2018fa

What is SSA and how does it help Data-flow analysis?

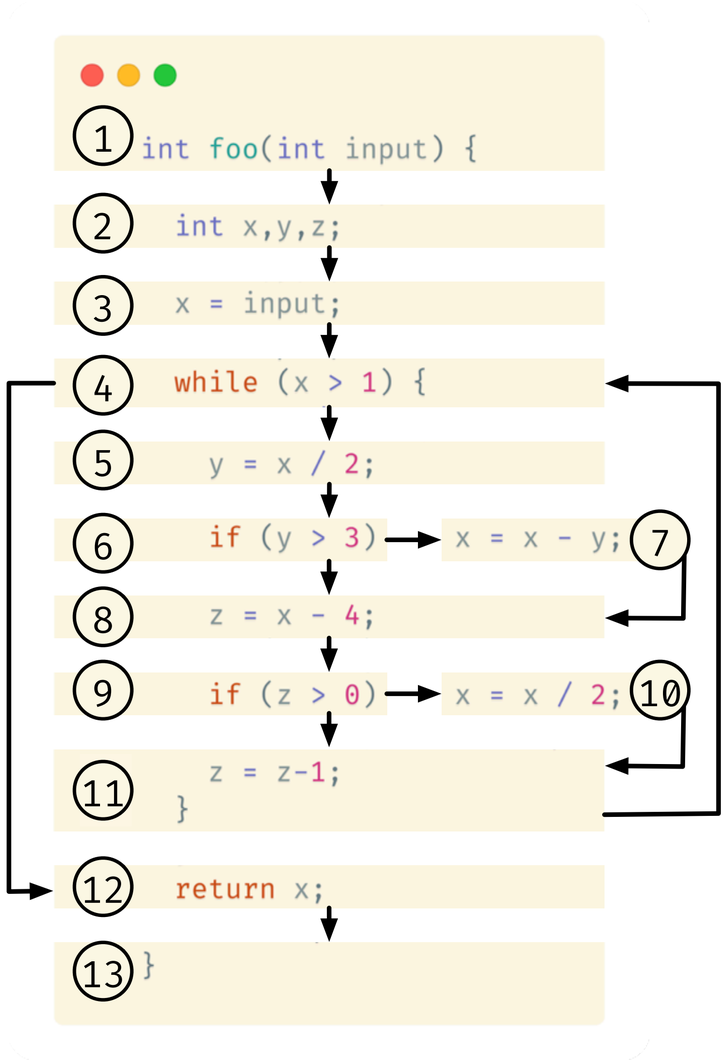
- Single Static Assignment (SSA) enforces two important properties on the IR:
  - variables are assigned exactly once
  - every variable is defined before it is used (the definition dominates all uses)

- This simplifies data-flow analysis!

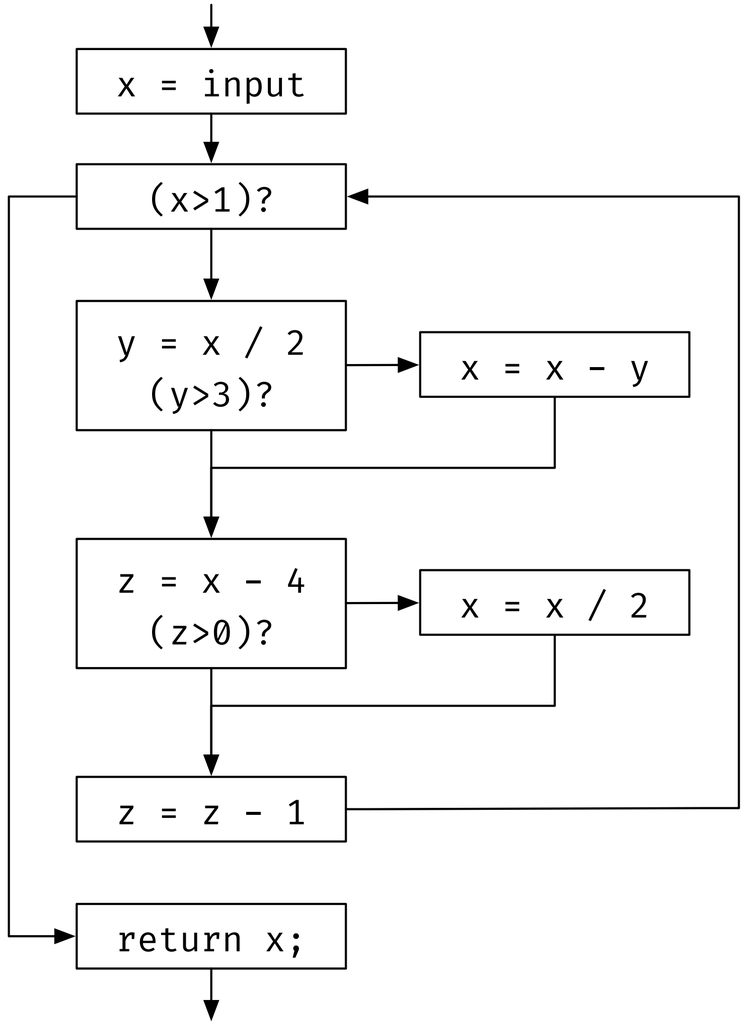
Kill set not longer required!

Stmt	Gen	Kill
x := v	{v}	

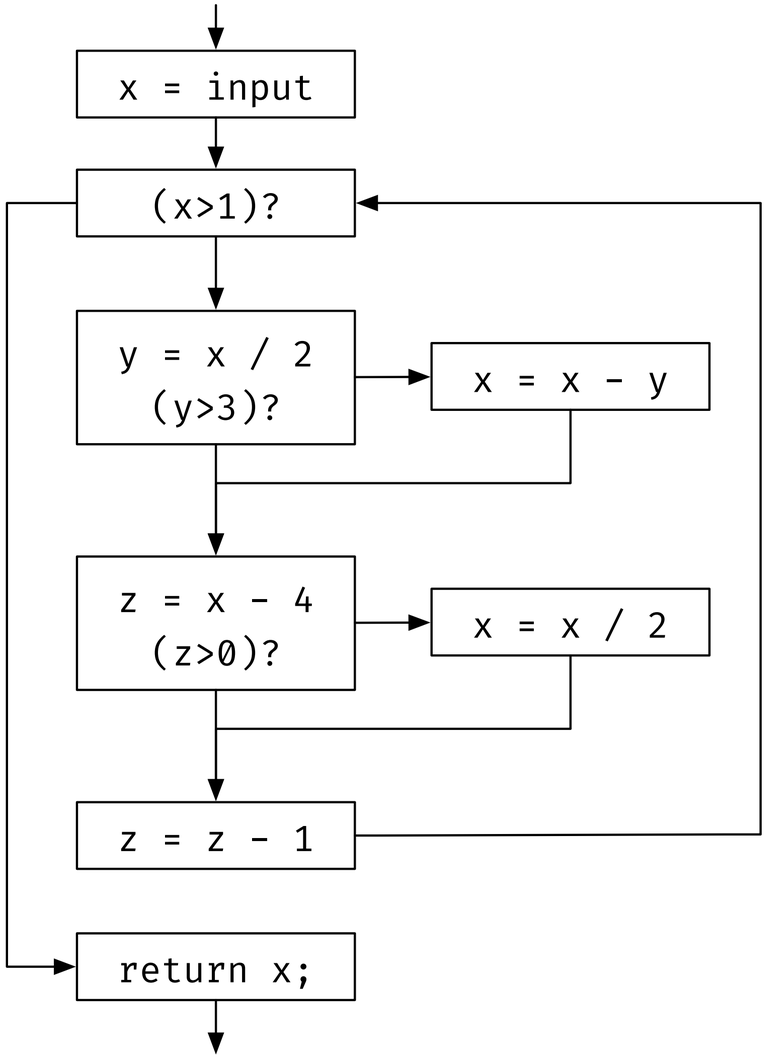
Constructing SSA



Lower control flow to branches

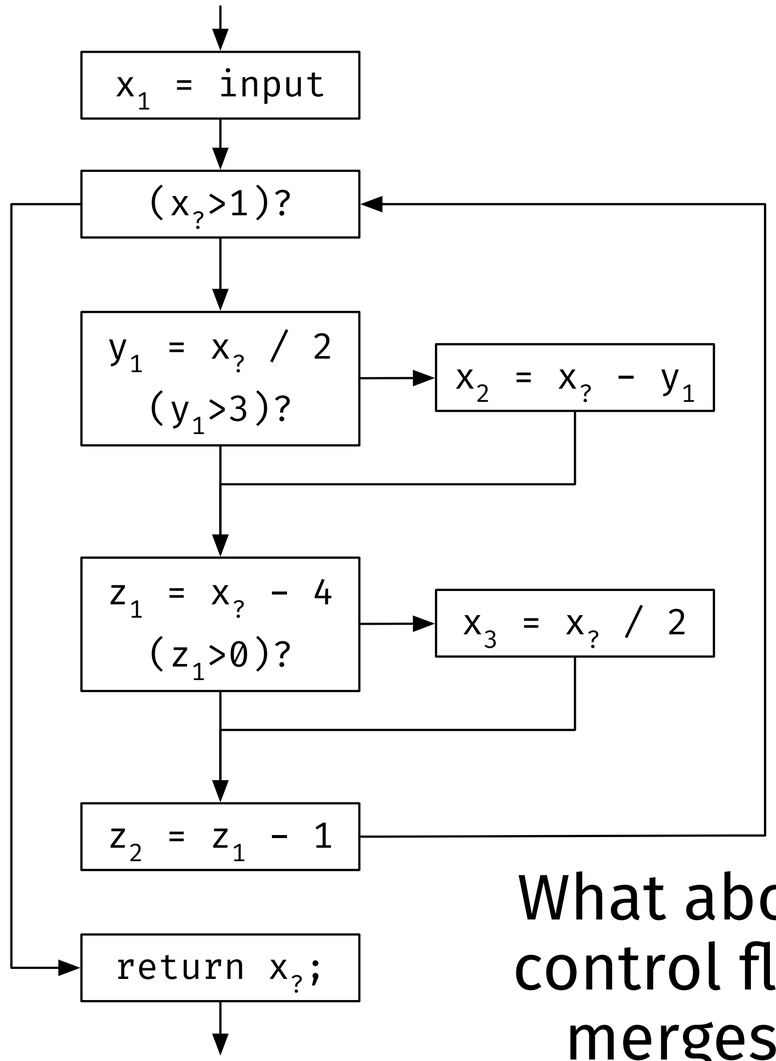


Constructing SSA



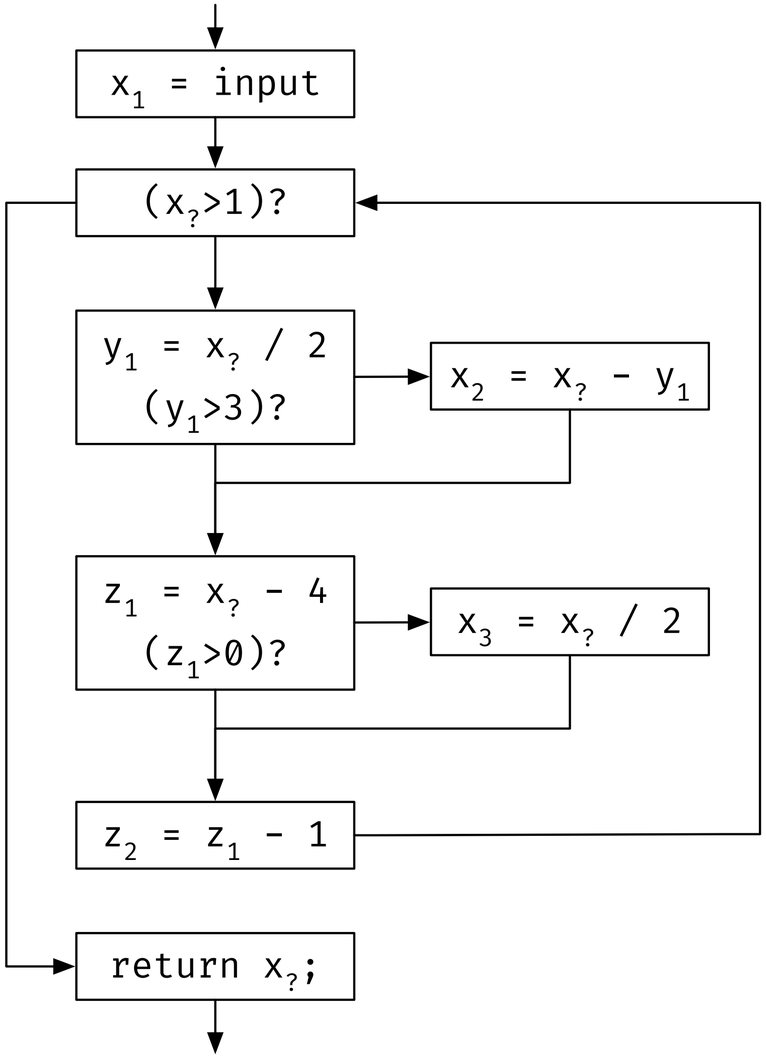
Each definition gets a fresh name

Propagate fresh name to uses

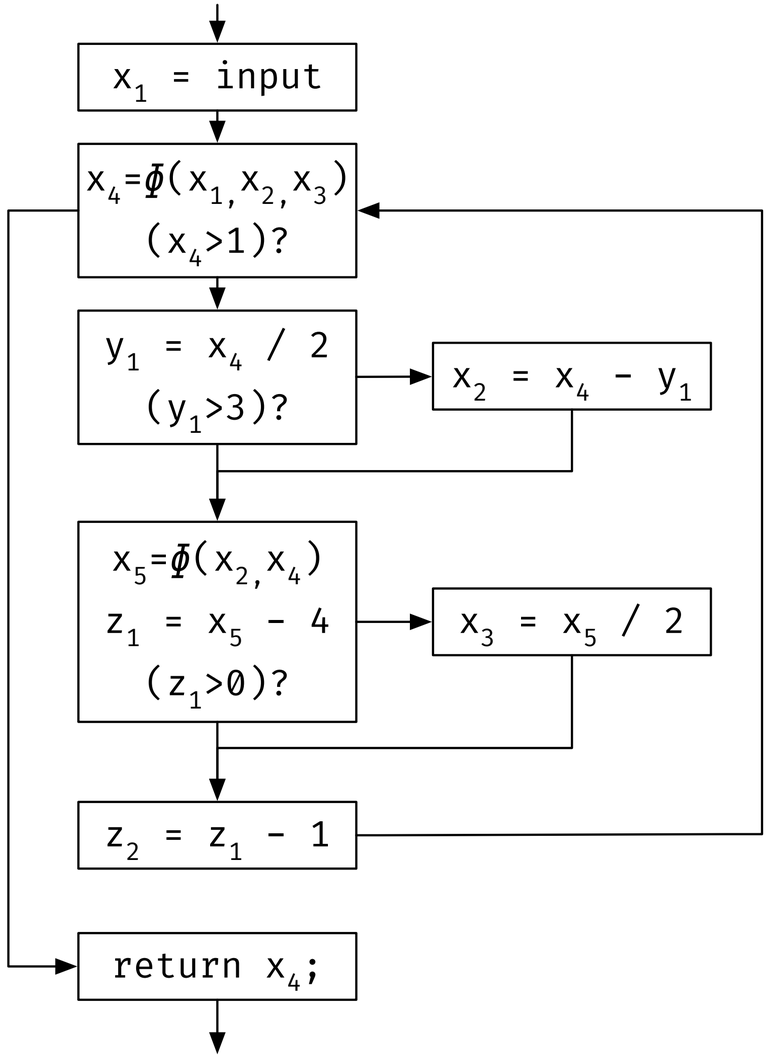


What about control flow merges?

Constructing SSA



Insert additional ϕ-nodes to merge control flow



Constant Folding is super easy in SSA

- For any statement `x=c` replace use of `x` with `c`
- Replace `x=Φ(c, …, c)` with `x=c`

```
W = list of all SSA statements

while (!W.isEmpty):
  S = W.head
  W = W.tail
  if (S == `x=Φ(c, …, c)`):
    S.replace(`x=c`)
  if (S == `x=c`):
    S.delete
    foreach T in use(x):
      replace c for x in T
      W.add(T)
```

LLVM IR - Textual representation

```c
int foo(int input) {
  int x,y,z;
  x = input;
  while (x > 1) {
    y = x / 2;
    if (y > 3) x = x - y;
    z = x - 4;
    if (z > 0) x = x / 2;
    z = z-1;
  }
  return x;
}
```

```
; ModuleID = 'foo.c'
source_filename = "foo.c"
target datalayout = "e-m:o-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64
target triple = "x86_64-apple-macosx10.15.0"

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @foo(i32 %0) #0 {
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32 %0, i32* %2, align 4
  %6 = load i32, i32* %2, align 4
  store i32 %6, i32* %3, align 4
  br label %7

7:                                                ; preds = %27, %1
  %8 = load i32, i32* %3, align 4
  %9 = icmp sgt i32 %8, 1
  br i1 %9, label %10, label %30

10:                                               ; preds = %7
  %11 = load i32, i32* %3, align 4
  %12 = sdiv i32 %11, 2
  store i32 %12, i32* %4, align 4
  %13 = load i32, i32* %4, align 4
  %14 = icmp sgt i32 %13, 3
  br i1 %14, label %15, label %19

15:                                               ; preds = %10
  %16 = load i32, i32* %3, align 4
  %17 = load i32, i32* %4, align 4
  %18 = sub nsw i32 %16, %17
```

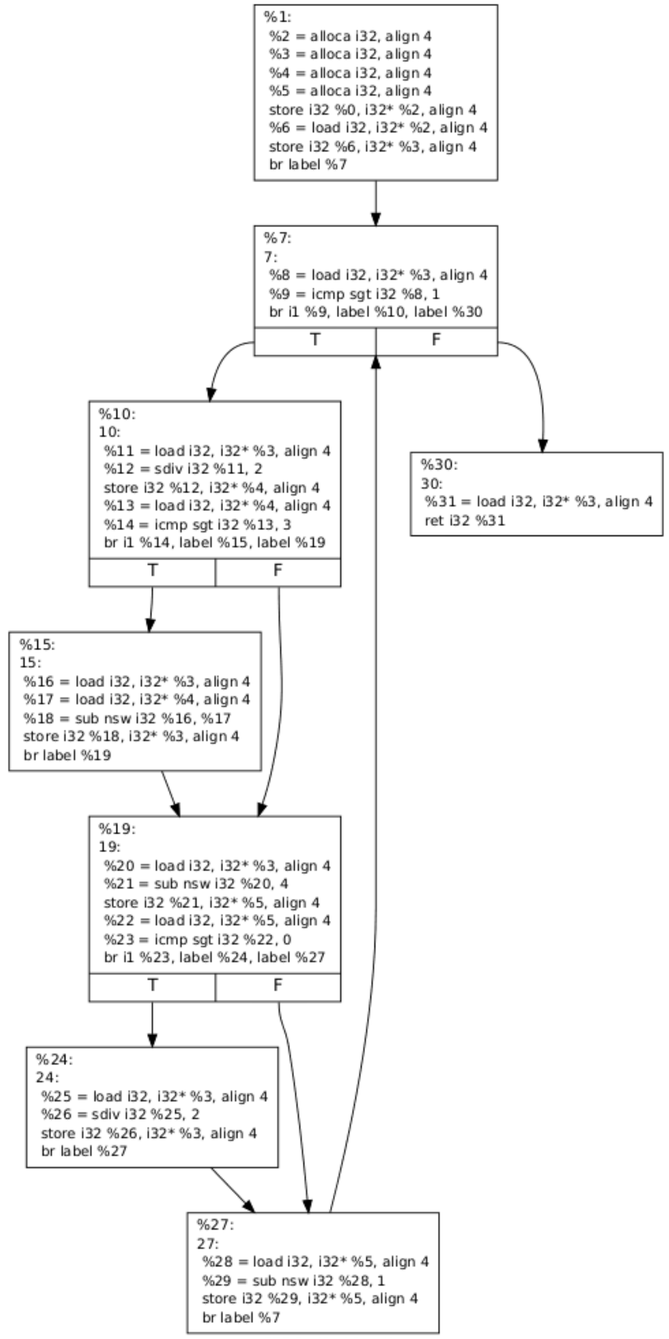
LLVM IR - Control Flow Graphs

- In LLVM's CFG nodes are *basic blocks:*
  groups of statements with sequential control flow

- You can let LLVM generate CFGs using the `dot` tool.

```
opt -dot-cfg foo.o
```

```c
int foo(int input) {
  int x,y,z;
  x = input;
  while (x > 1) {
    y = x / 2;
    if (y > 3) x = x - y;
    z = x - 4;
    if (z > 0) x = x / 2;
    z = z-1;
  }
  return x;
}
```



CFG for 'foo' function

LLVM IR – ϕ nodes vs. load&store instructions

- `load` and `store` instructions model reading and writing from variables which reduces the number of ϕ nodes
- ϕ nodes are still used, e.g. for control flow in individual expressions:

```
int l = y || r;
```

results in:

```
  %6 = load i32, i32* %4, align 4, !dbg !18
  %7 = icmp ne i32 %6, 0, !dbg !18
  br i1 %7, label %11, label %8, !dbg !19

8:
  %9 = load i32, i32* %3, align 4, !dbg !20
  %10 = icmp ne i32 %9, 0, !dbg !20
  br label %11, !dbg !19

11:
  %12 = phi i1 [ true, %2 ], [ %10, %8 ]
  %13 = zext i1 %12 to i32, !dbg !18
  store i32 %13, i32* %5, align 4, !dbg !17
```

References

- Prof. Stephen Chong Compiler Course at Harvard https://www.seas.harvard.edu/courses/cs153/2018fa
- Michael I. Schwartzbach, Lecture Notes on Static Analysis https://itu.dk/people/brabrand/UFPE/Data-Flow-Analysis/static.pdf
- Static Single Assignment Book http://ssabook.gforge.inria.fr/latest/book.pdf
- Online Compiler Explorer https://godbolt.org/