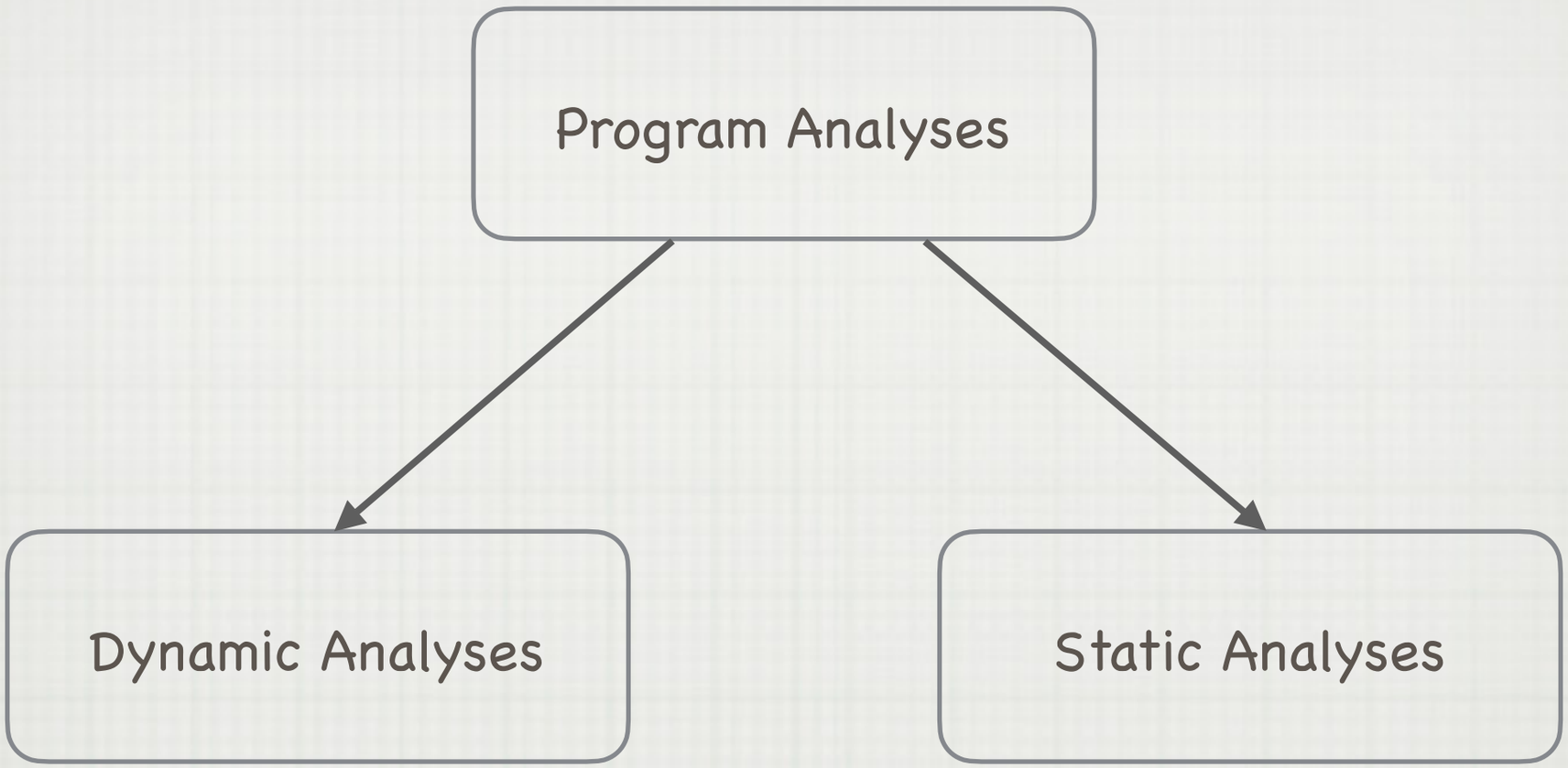
CSC410
Data Flow Analyses

Azadeh Farzan

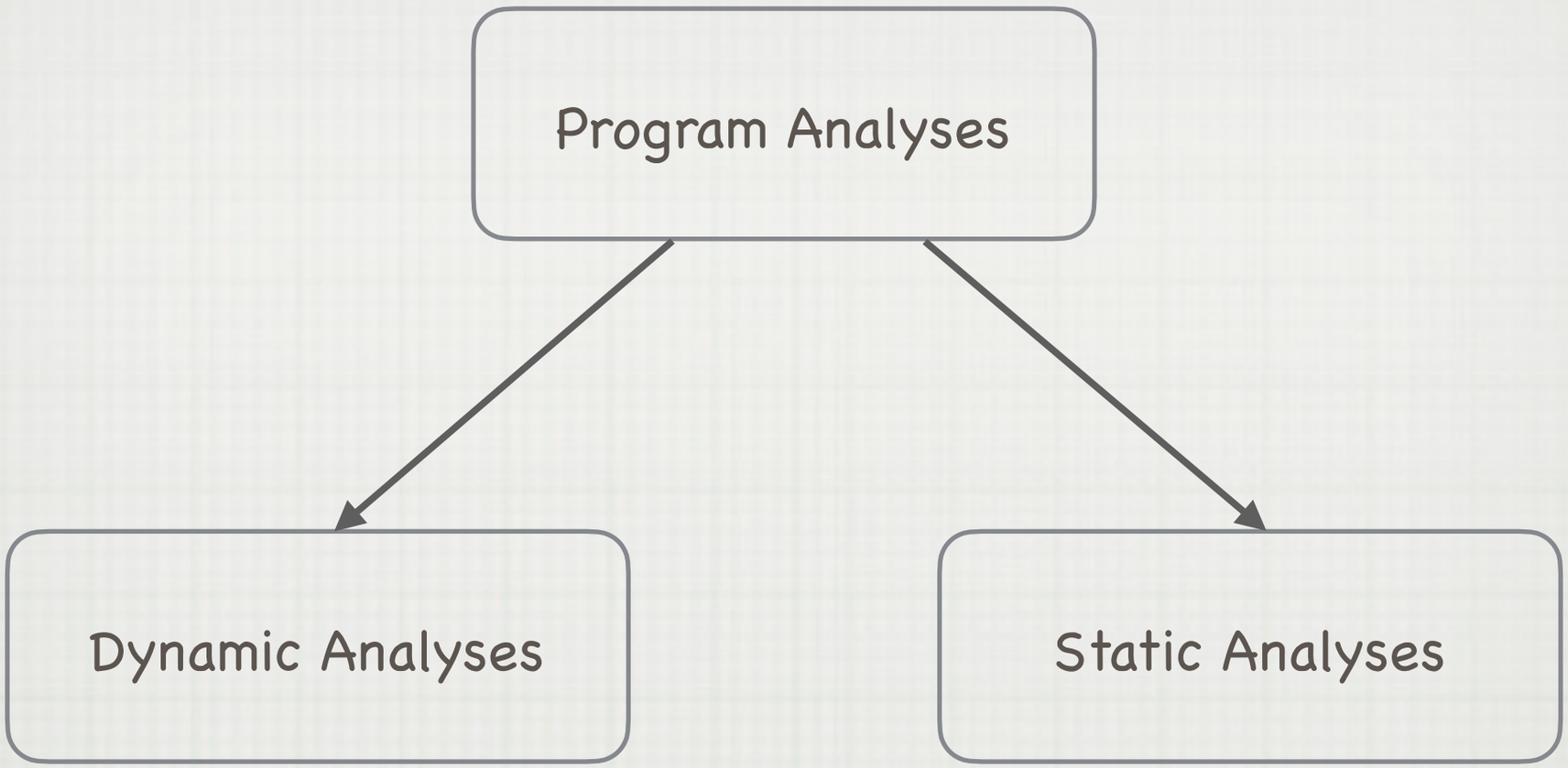
Fall 2026

What is program analysis?

Program Analyses

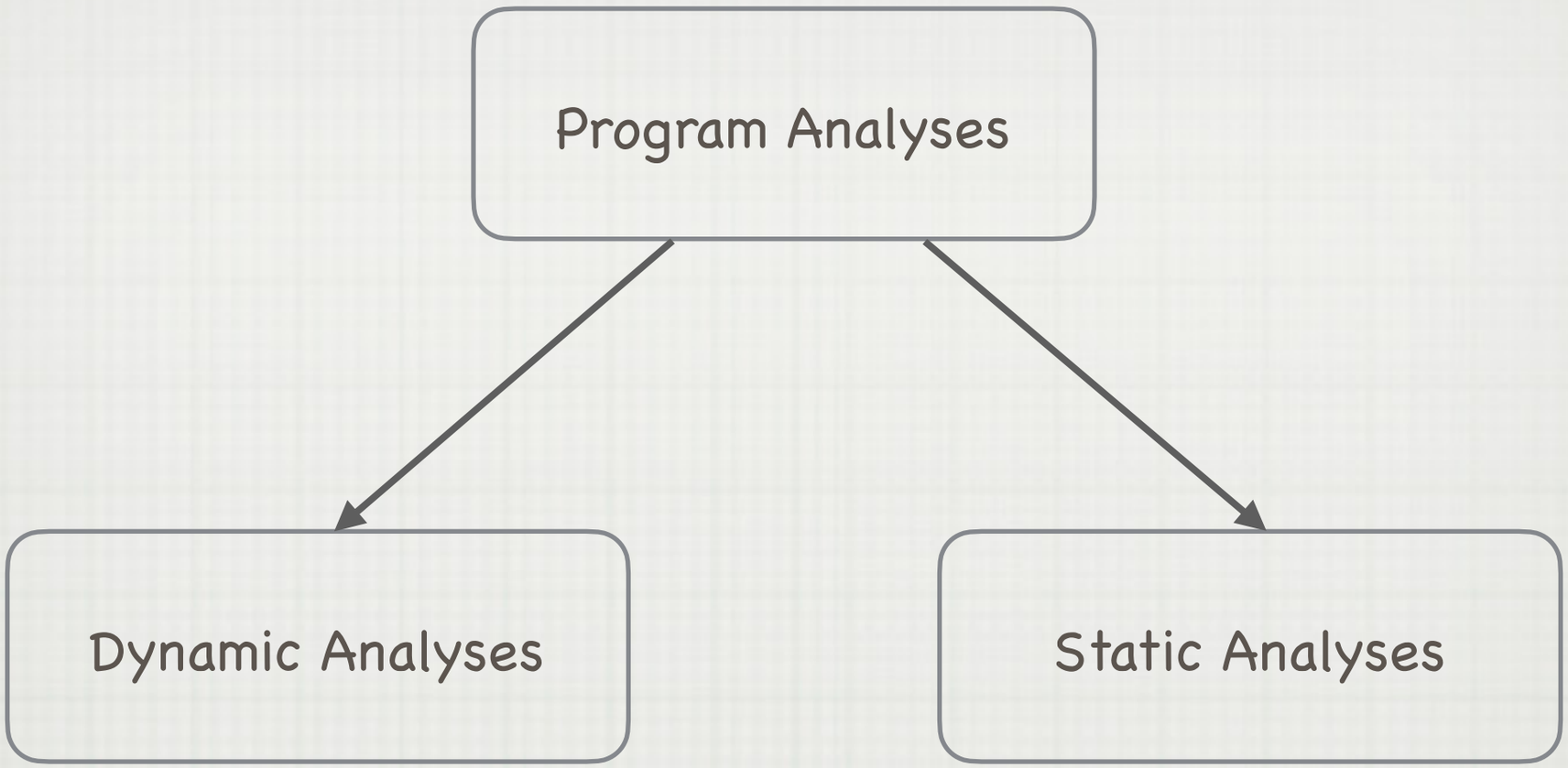


Program Analyses



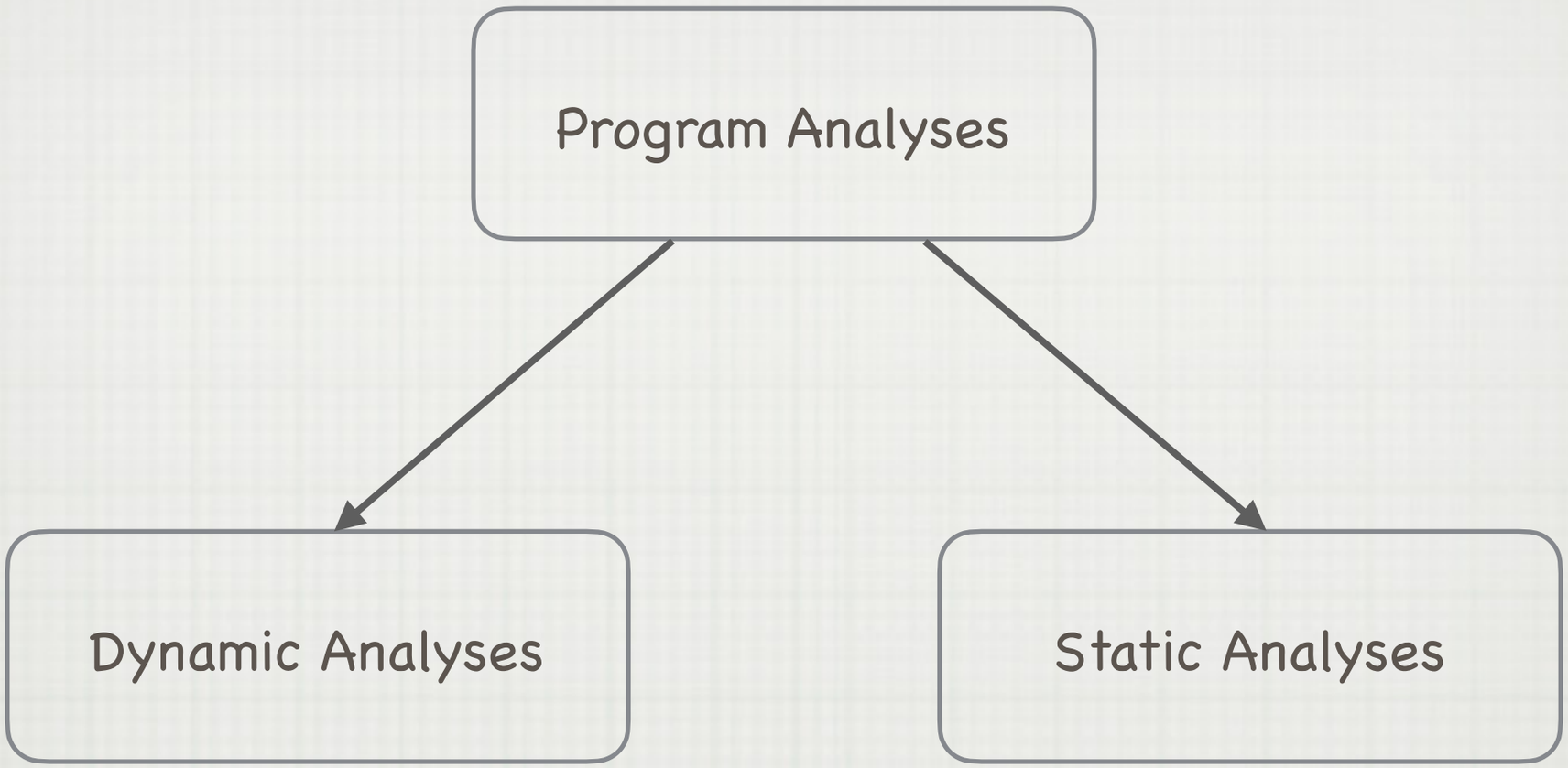
Testing

Program Analyses



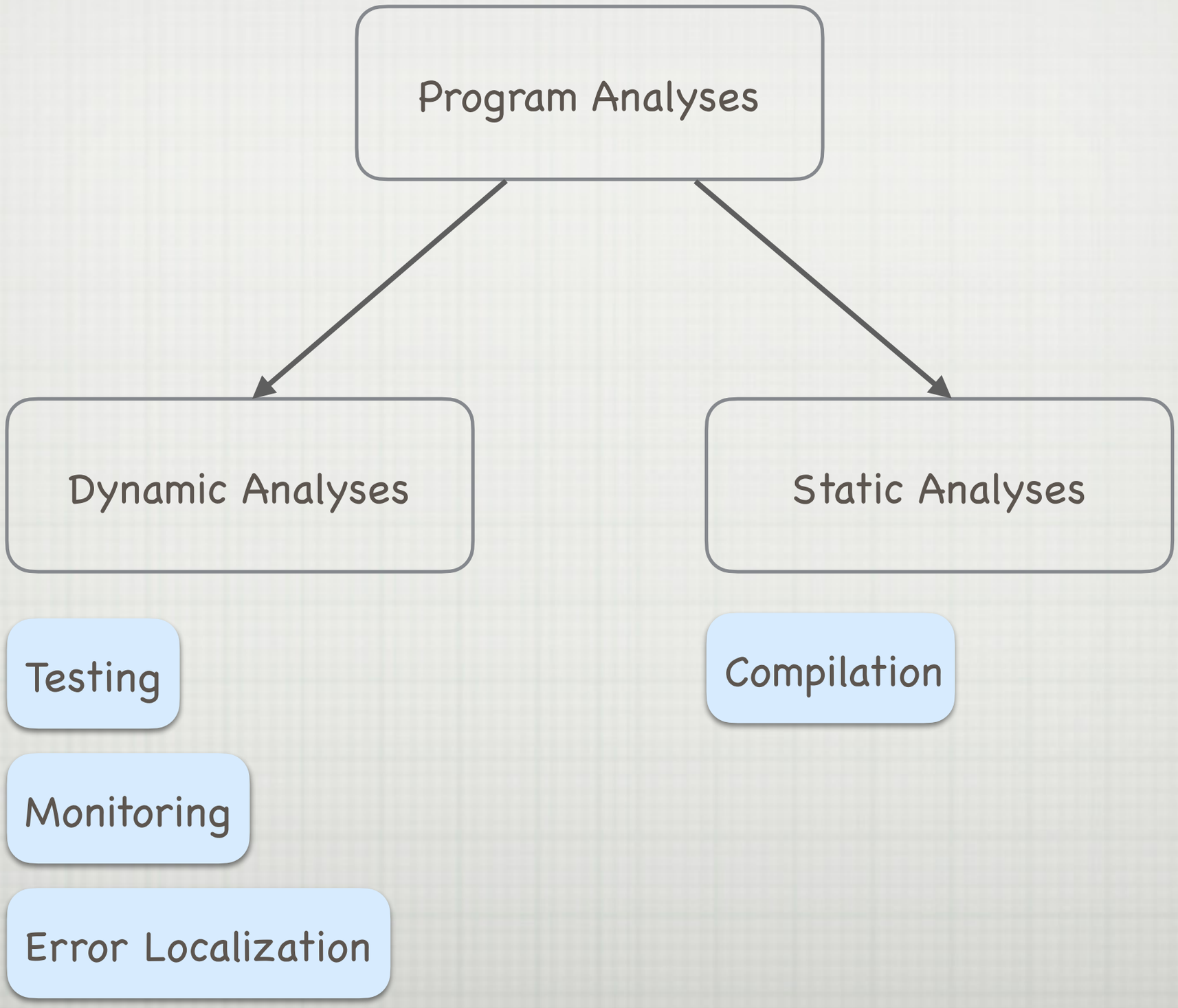
- Testing
- Monitoring

Program Analyses

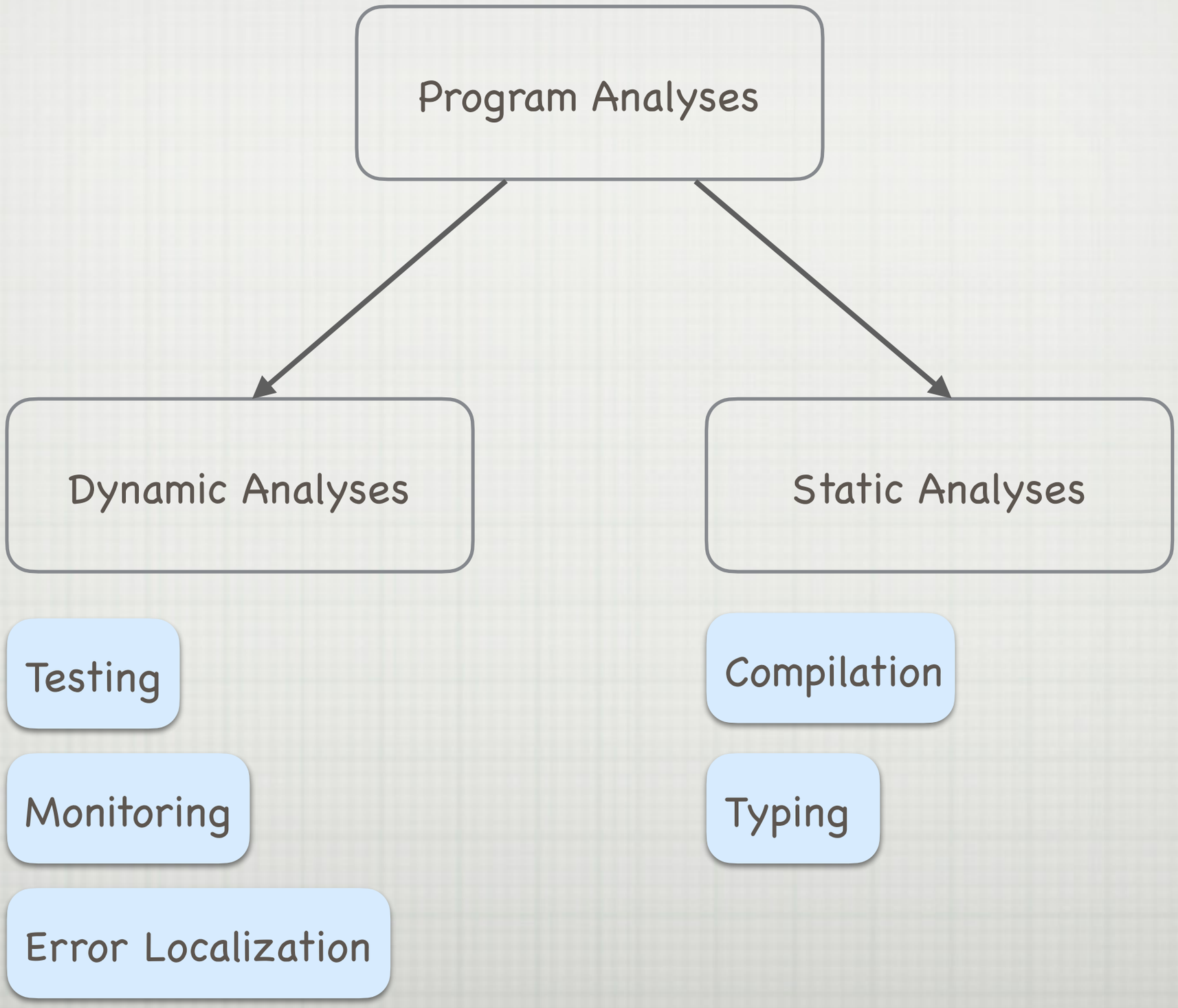


- Testing
- Monitoring
- Error Localization

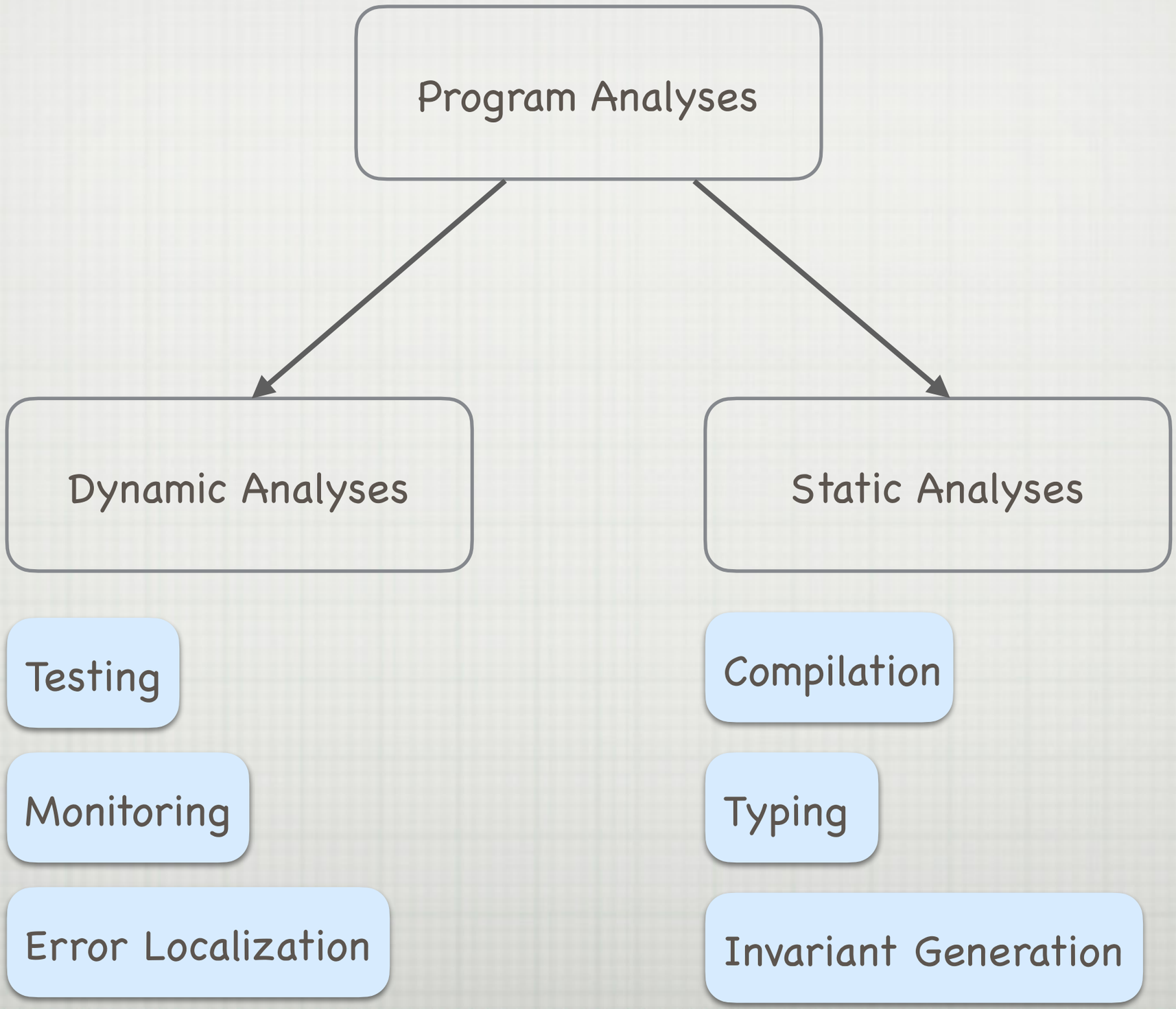
Program Analyses



Program Analyses



Program Analyses

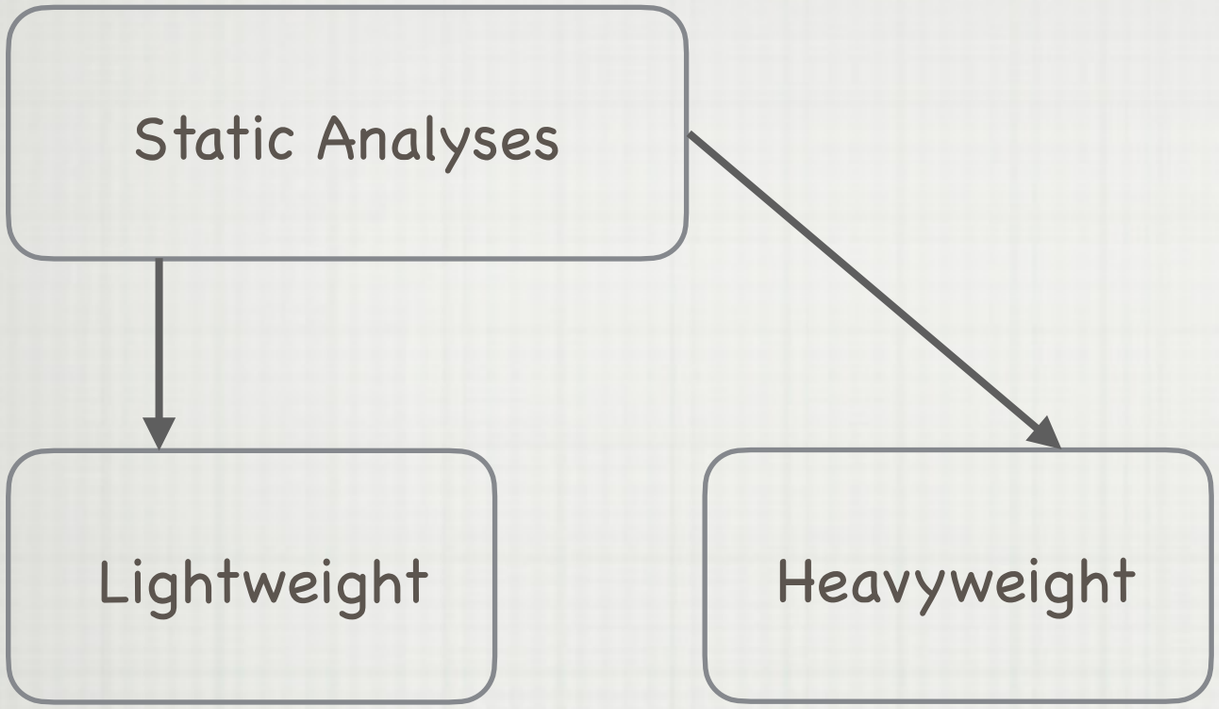


Program Analyses

Static Analyses

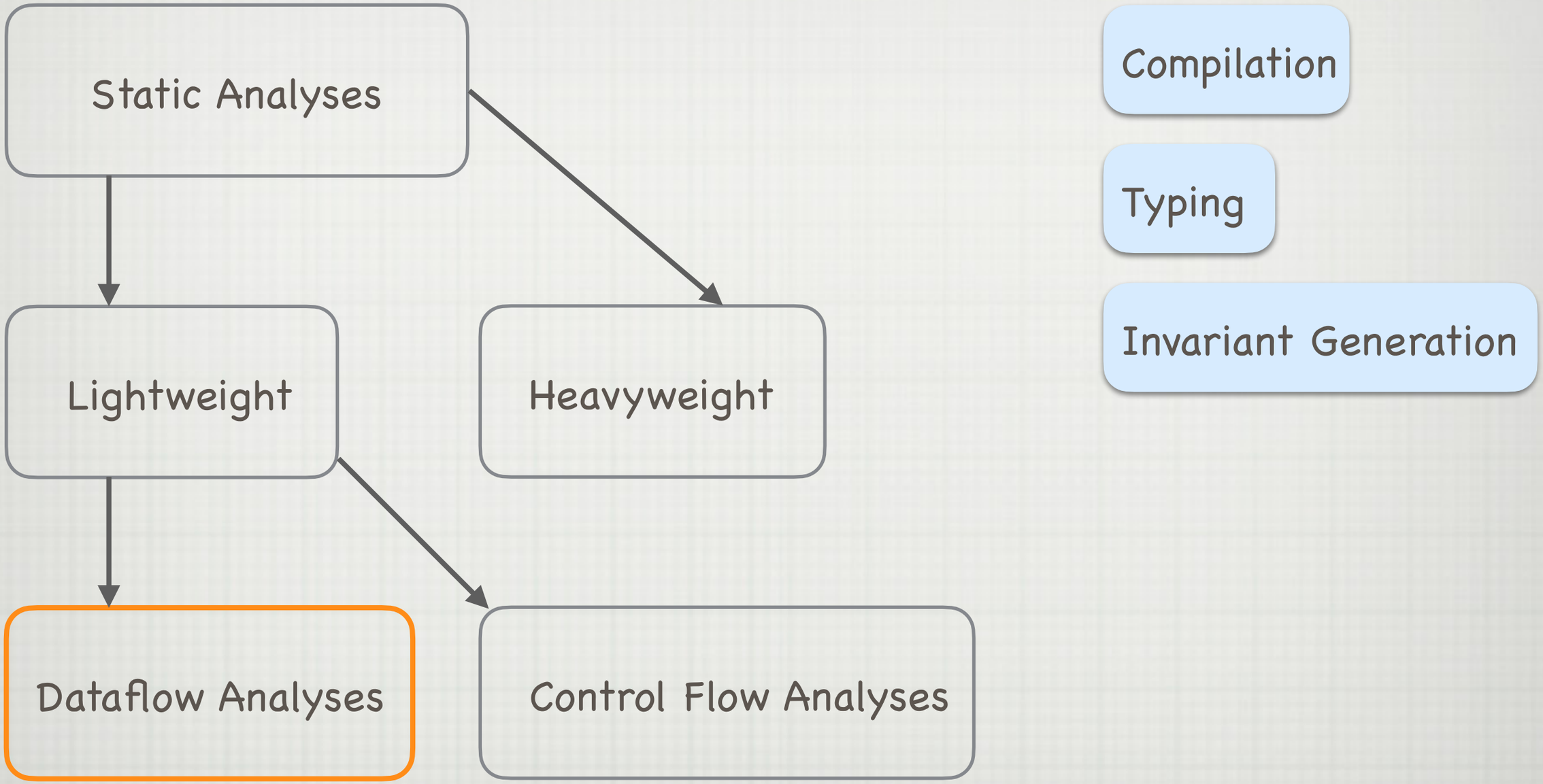
- Compilation
- Typing
- Invariant Generation

Program Analyses

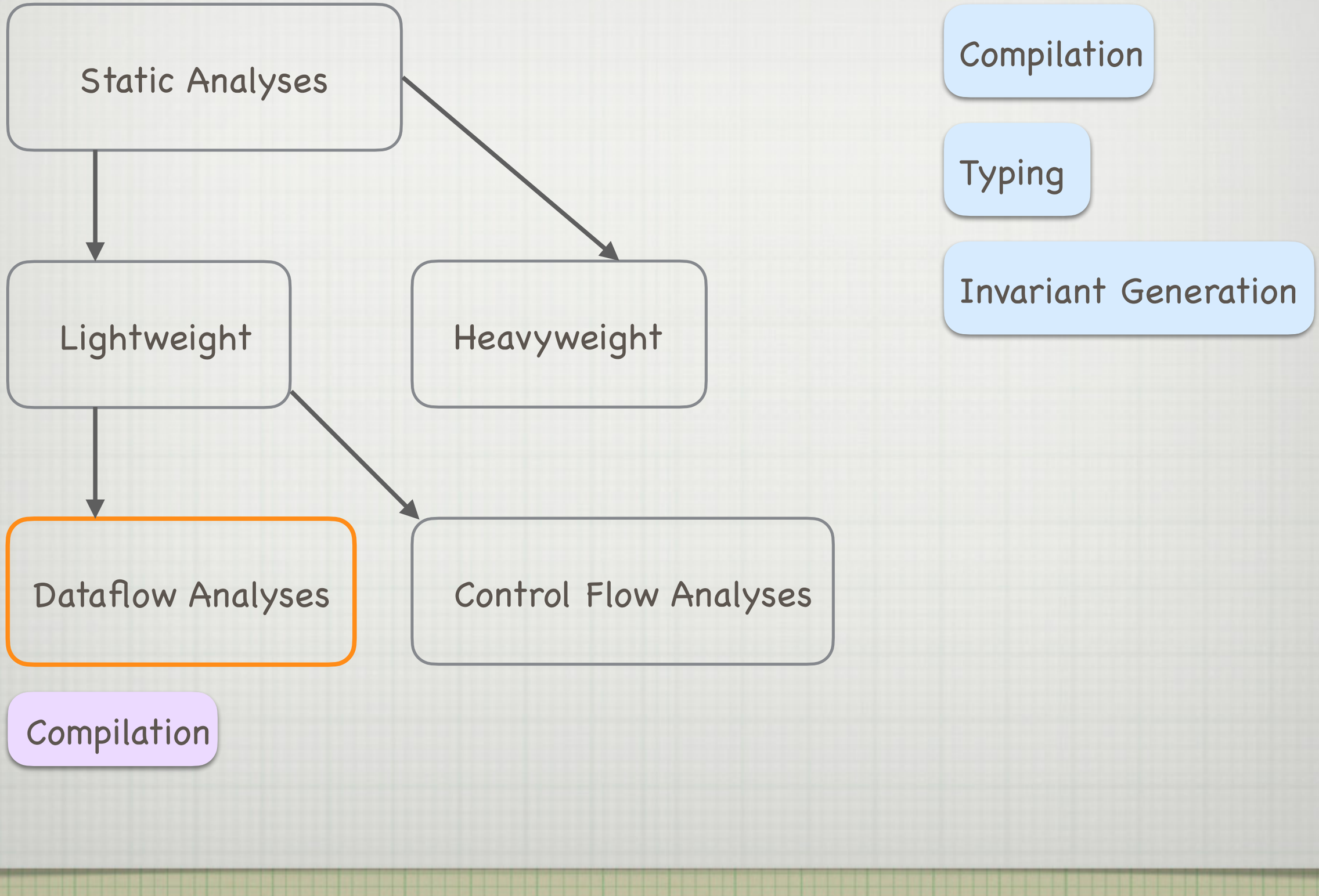


- Compilation
- Typing
- Invariant Generation

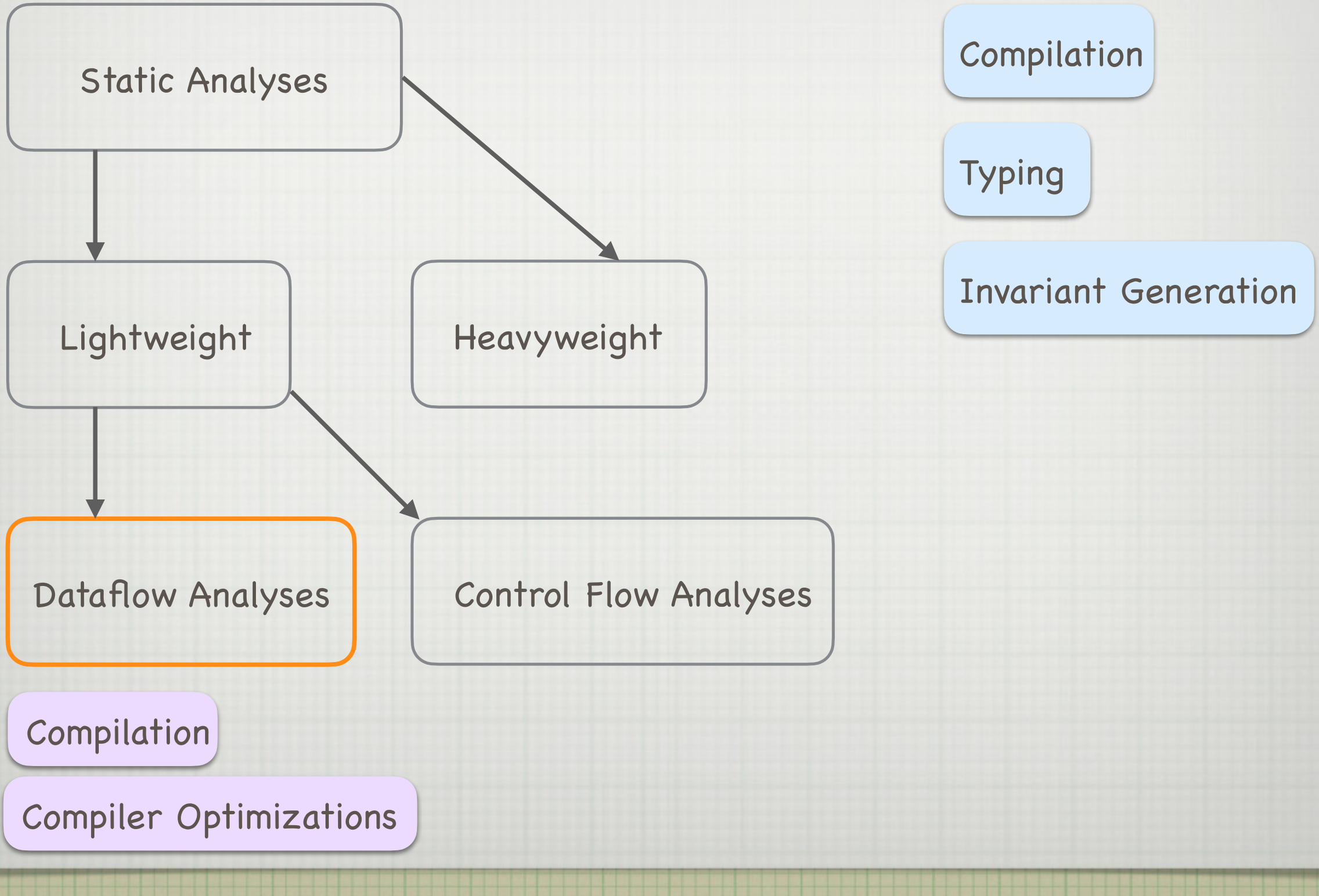
Program Analyses



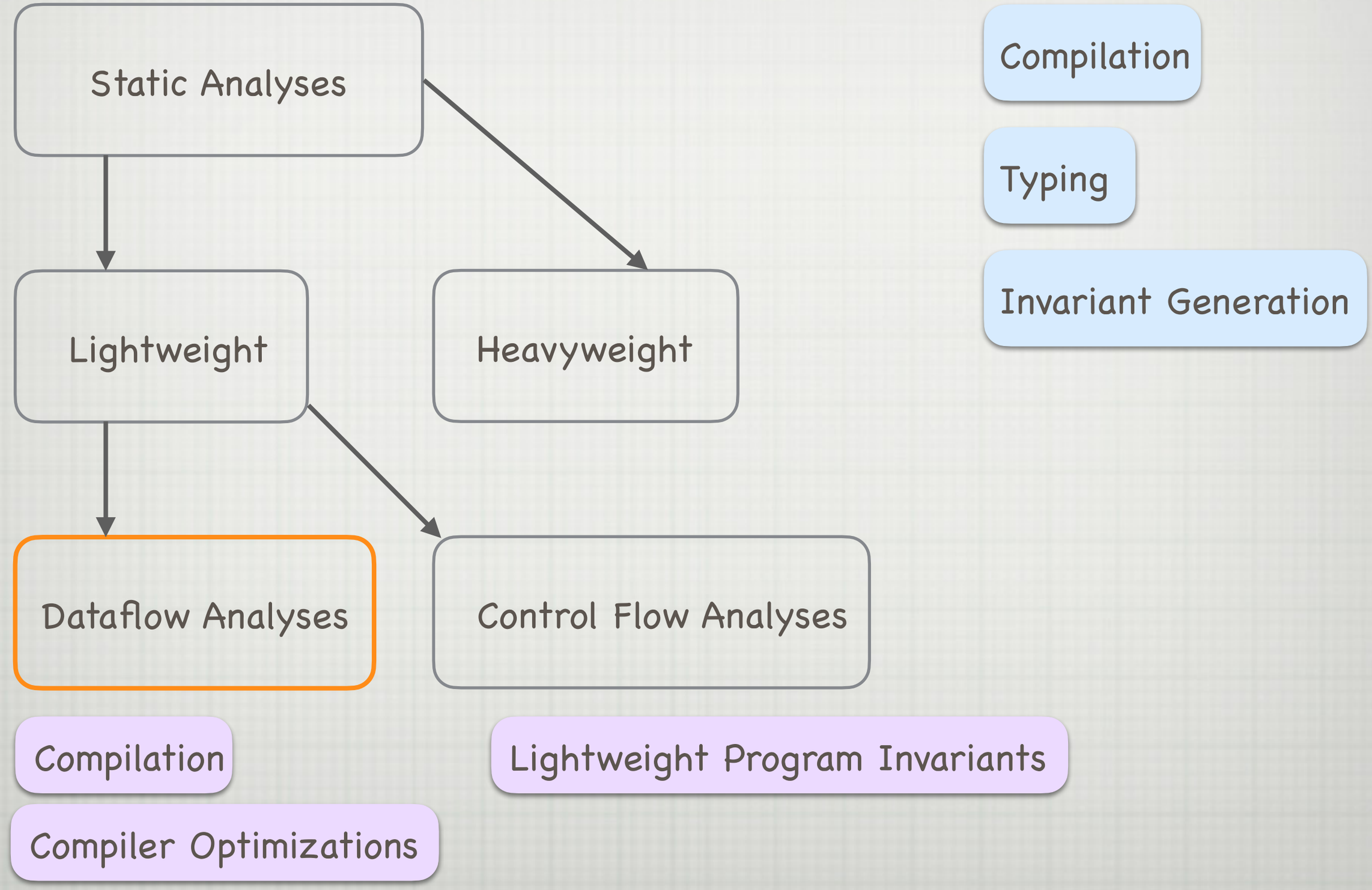
Program Analyses



Program Analyses



Program Analyses



Dataflow Analyses

Dataflow Analyses

Goal and scope:

- Calculate over-approximate information about the behaviours of the program.
- Do it efficiently!

Dataflow Analyses

Goal and scope:

- Calculate over-approximate information about the behaviours of the program.
- Do it efficiently!

Applications:

- Widely used in optimizing compilers
- IDEs
- Verification
- Testing

Let's start with an example...

Dead Code Elimination

Dead Code Elimination

Can part of this program be removed without affecting program behaviour?

```
x := 2
y := 5
x := 1
z := 0
if (x <= 0) {
    z := x + 2
} else {
    z := y * y
}
x := z
```

Dead Code Elimination

Can part of this program be removed without affecting program behaviour?

```
x := 2
y := 5
x := 1
z := 0
if (x <= 0) {
    z := x + 2
} else {
    z := y * y
}
x := z
```

Dead Code Elimination

Can part of this program be removed without affecting program behaviour?

```
x := 2
y := 5
x := 1
z := 0
if (x <= 0) {
    z := x + 2
} else {
    z := y * y
}
x := z
```

Based on the principle that these assignments to x and z can never be read by any other statement.

Live Variable Analysis

Live Variable Analysis

Intuition: A variable v is live at a program point if its value may be read later in some execution of the program. Otherwise, it is dead.

Live Variable Analysis

Intuition: A variable v is live at a program point if its value may be read later in some execution of the program. Otherwise, it is dead.

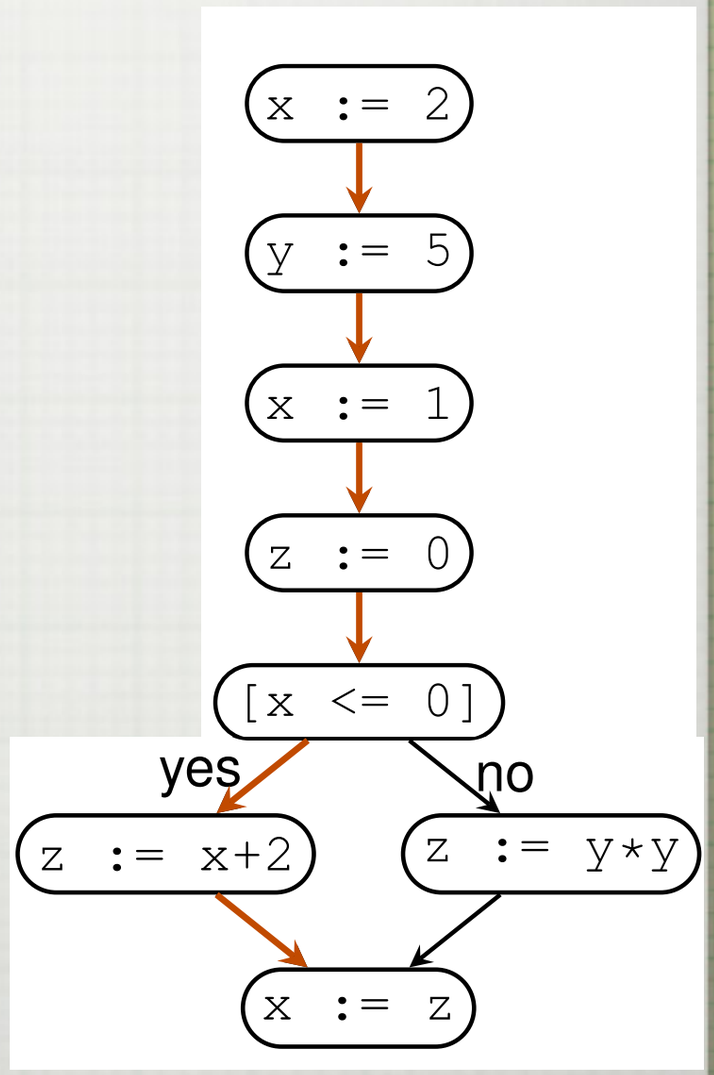
An assignment x := exp can be safely removed from the program (i.e it is dead code) if x is dead (at the exit point of this statement).

How do we calculate this information automatically and efficiently?

Models

Control Flow Graphs

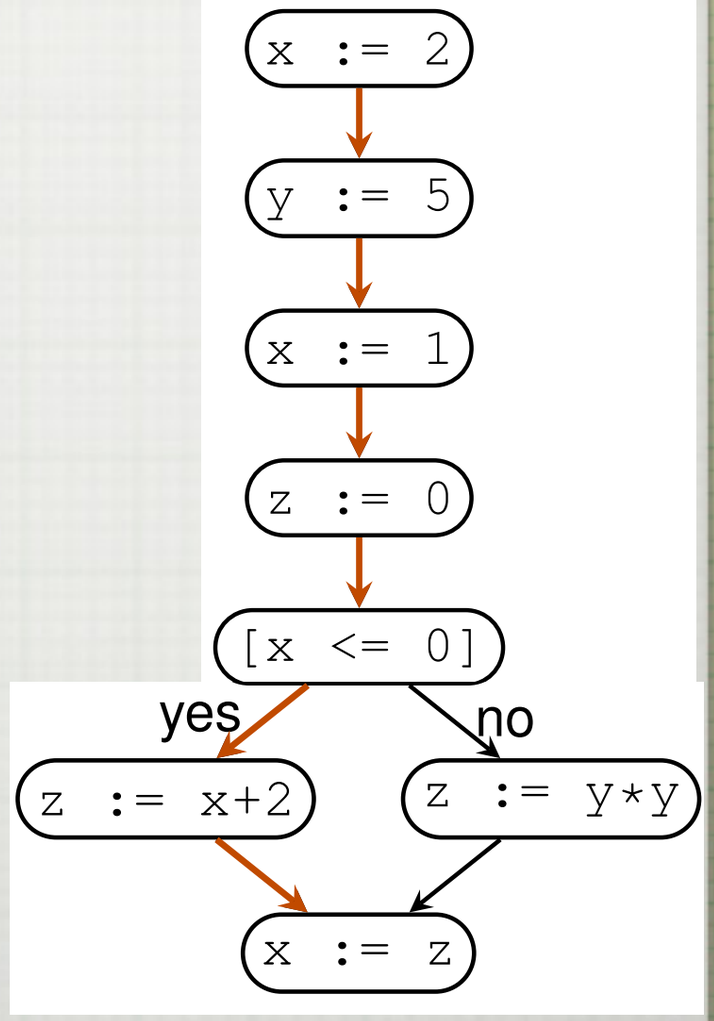
A Control Flow Graph (CFG) is a graphical representation of the executions of a program. Formally, a CFG is a directed graph ⟨V , E⟩, where:



Control Flow Graphs

A Control Flow Graph (CFG) is a graphical representation of the executions of a program. Formally, a CFG is a directed graph $\langle V , E \rangle$, where:

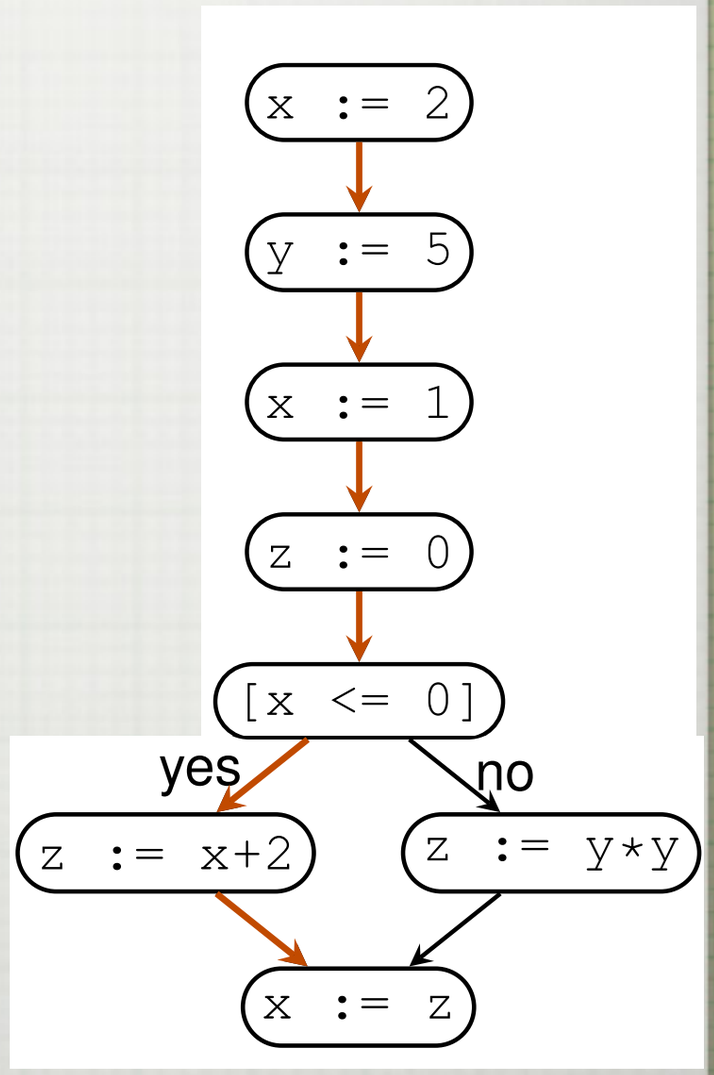
- V is a set of program labels (AKA locations, points)
- E is the control flow successor relation: if v follows u in some execution, then $u \rightarrow v \in E$



Control Flow Graphs

A Control Flow Graph (CFG) is a graphical representation of the executions of a program. Formally, a CFG is a directed graph ⟨V , E⟩, where:

- V is a set of program labels (AKA locations, points)
- E is the control flow successor relation: if v follows u in some execution, then $u \rightarrow v \in E$



Every execution corresponds to a path in the CFG:

- the converse does not hold!
- A CFG represents an over-approximation of the executions of a program

Formal Specification

Live Variable Analysis

Intuition: A variable v is live at a program point if its value may be read later in some execution of the program. Otherwise, it is dead.

Live Variable Analysis

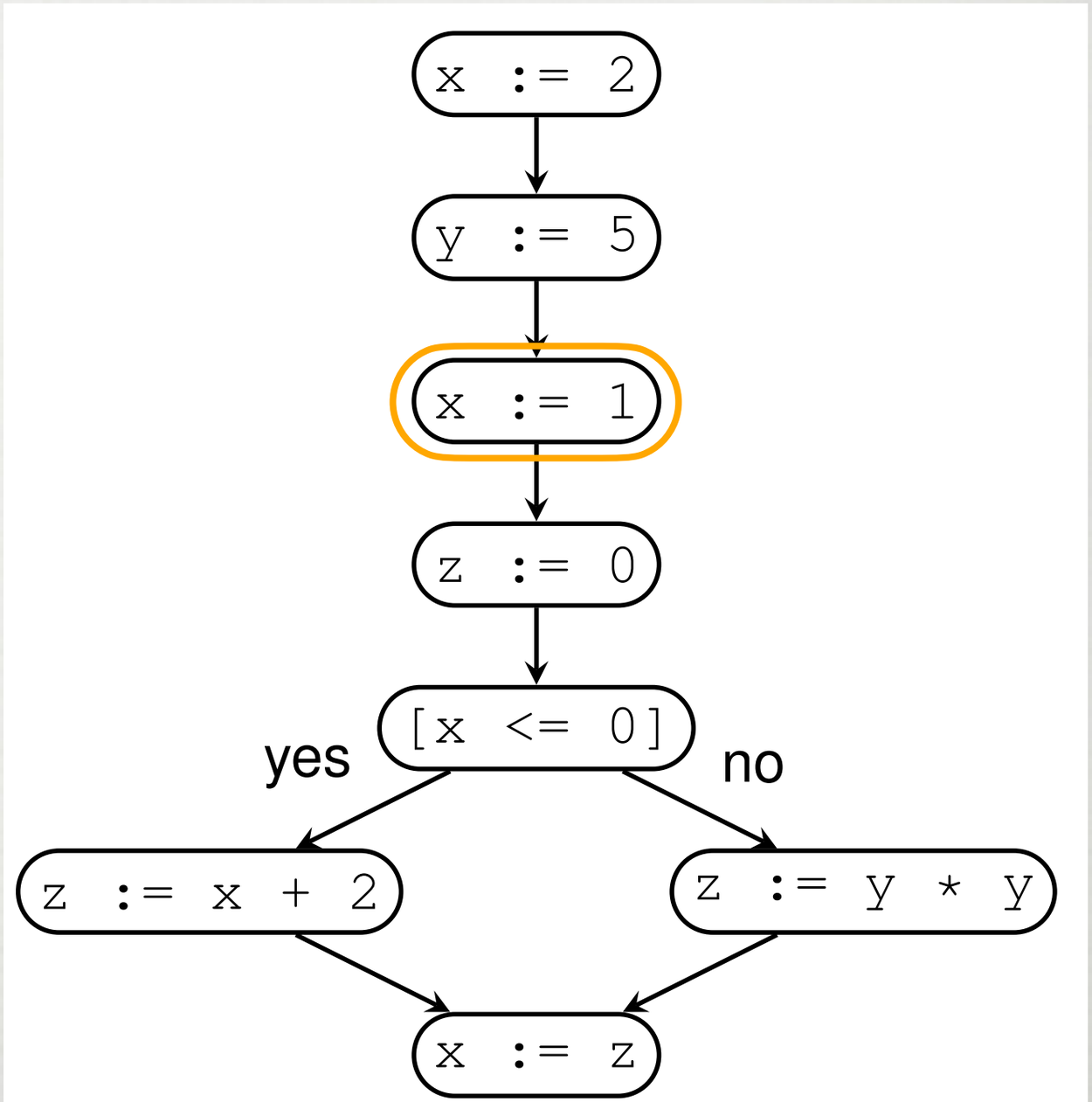
Intuition: A variable v is live at a program point if its value may be read later in some execution of the program. Otherwise, it is dead.

Formal definition: A variable var is live at a label l if there exists a CFG path from l to a label l' such that l' reads var, and no label along that path writes to var. Otherwise it is dead.

Example

At x:=1:

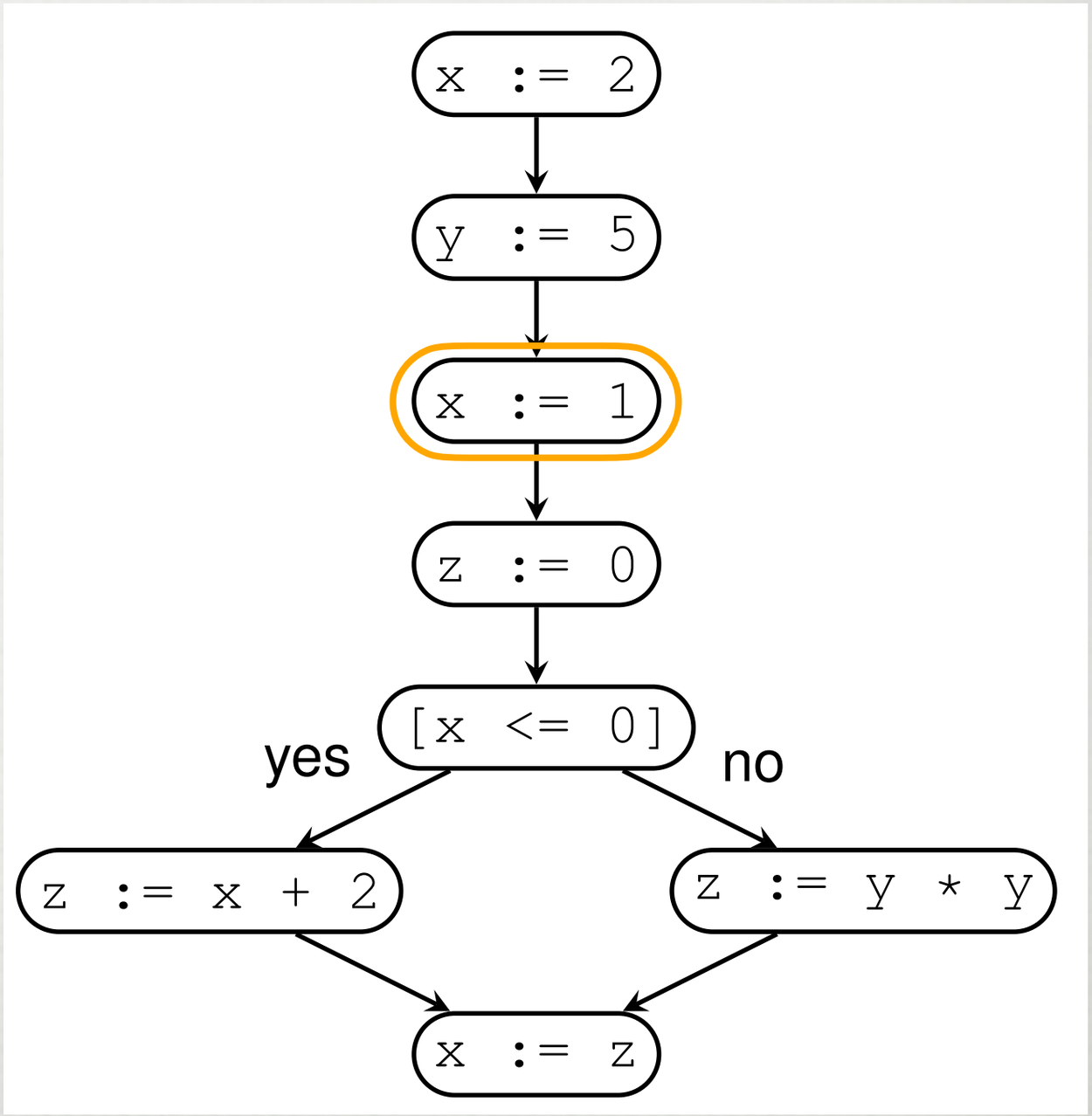
- ☐ x
- ☐ y
- ☐ z



Example

At x:=1:

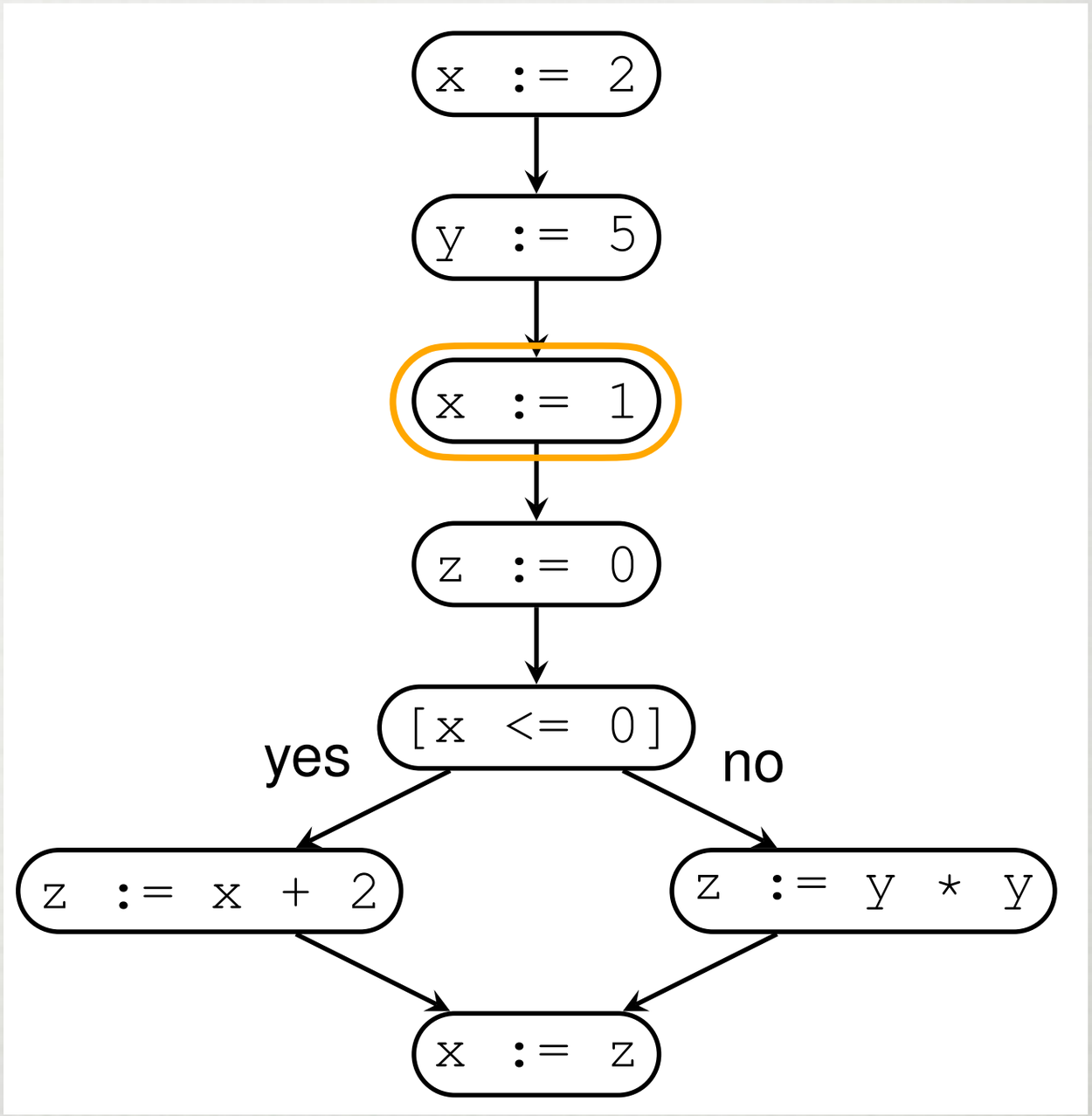
- ☐ x is live
- ☐ y
- ☐ z



Example

At x:=1:

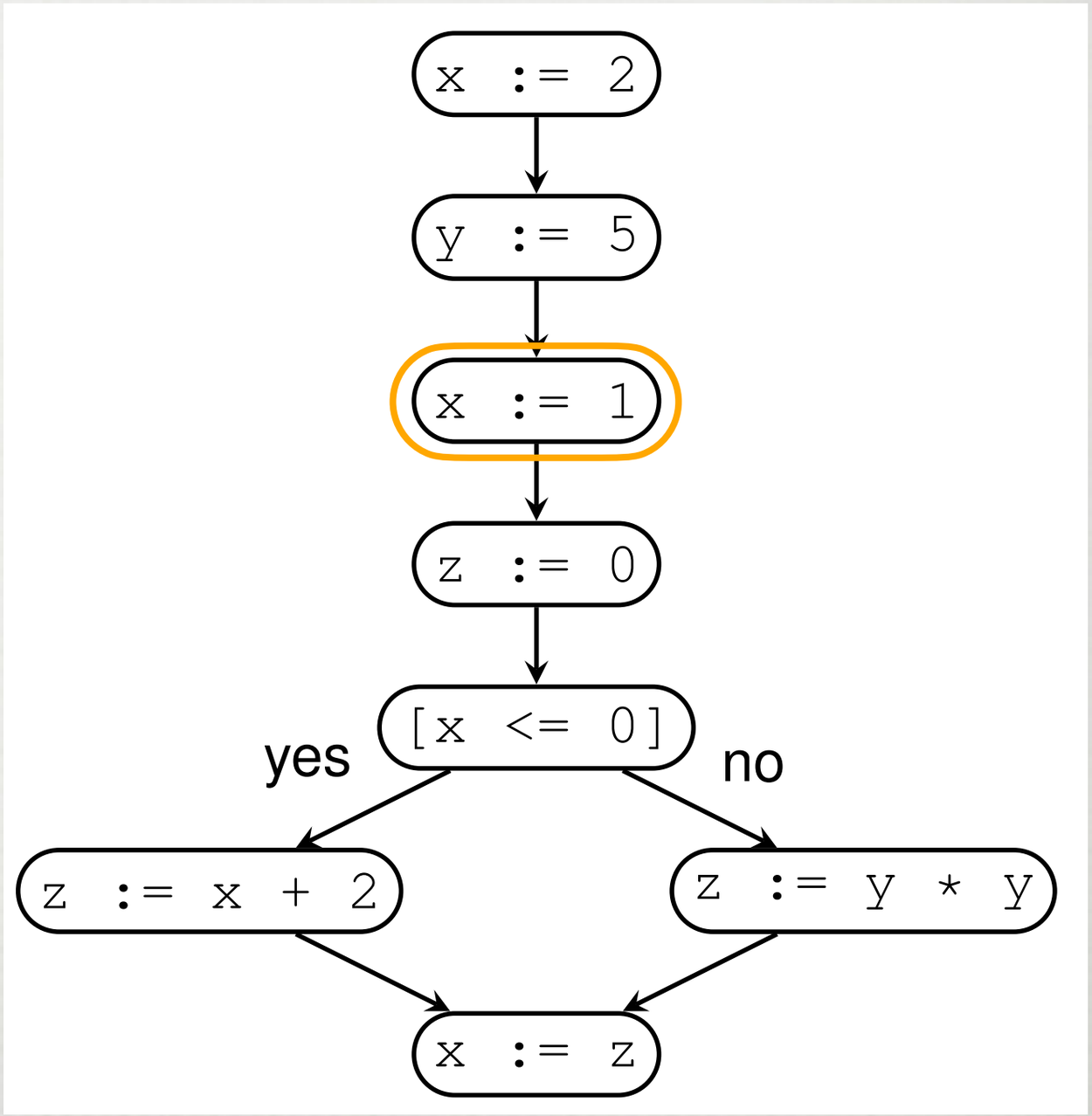
- ☐ x is live
- ☐ y is live
- ☐ z



Example

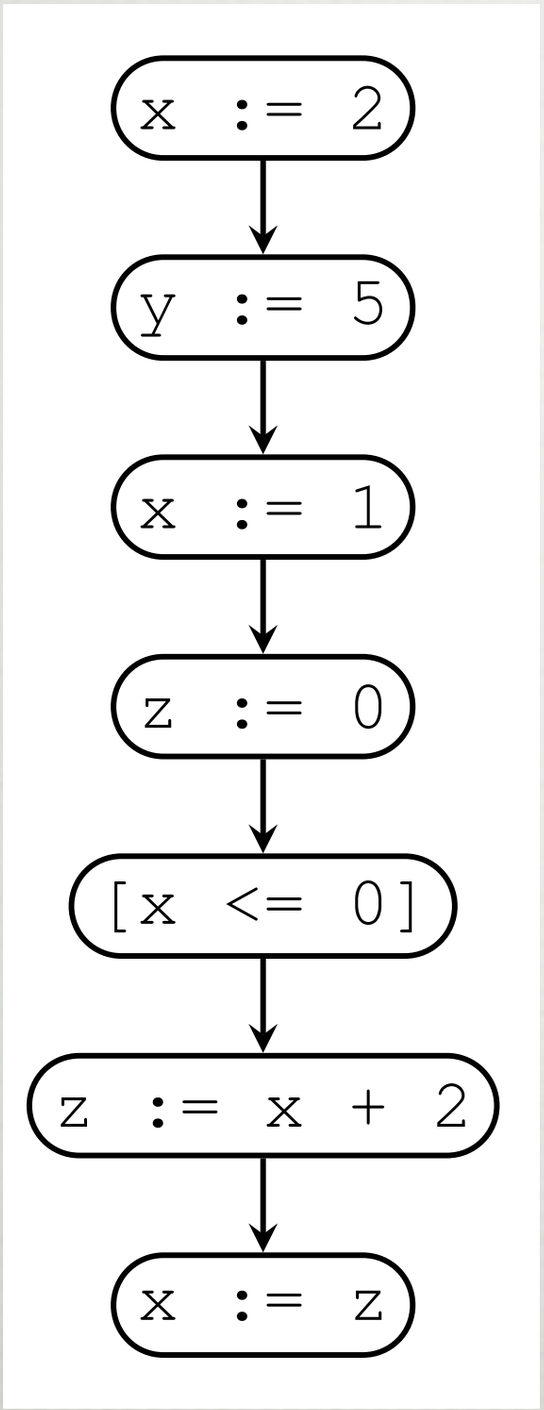
At x:=1:

- ☐ x is live
- ☐ y is live
- ☐ z is dead

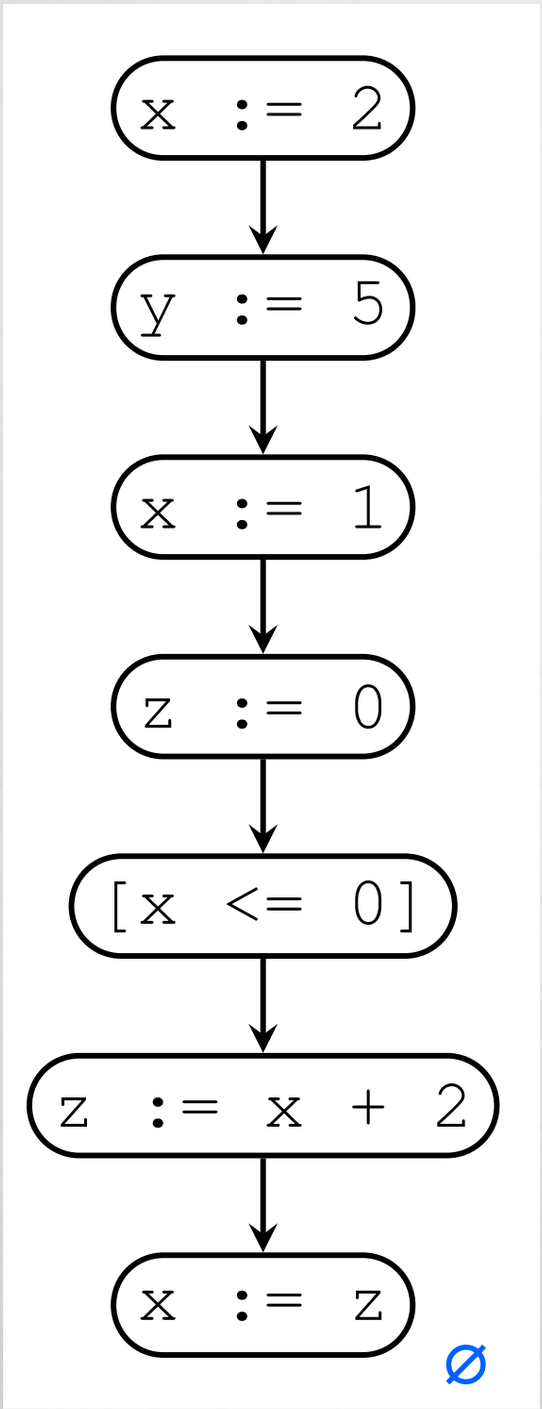


Let's start automating this in the simplest context ...

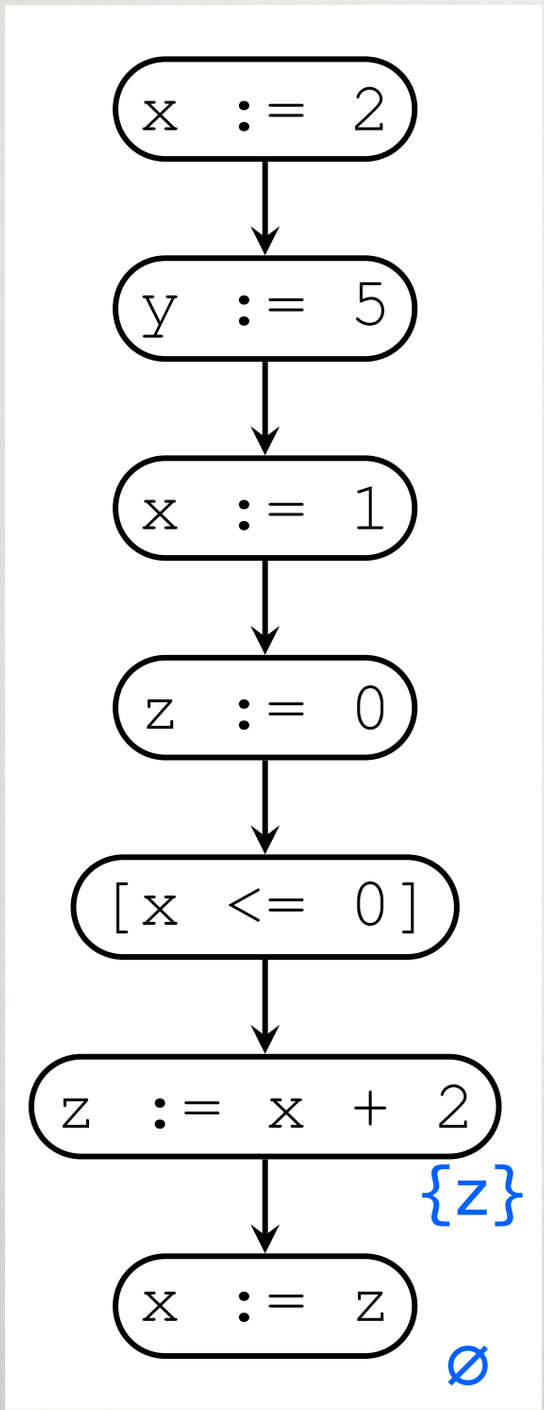
Live Variable Analysis on Paths



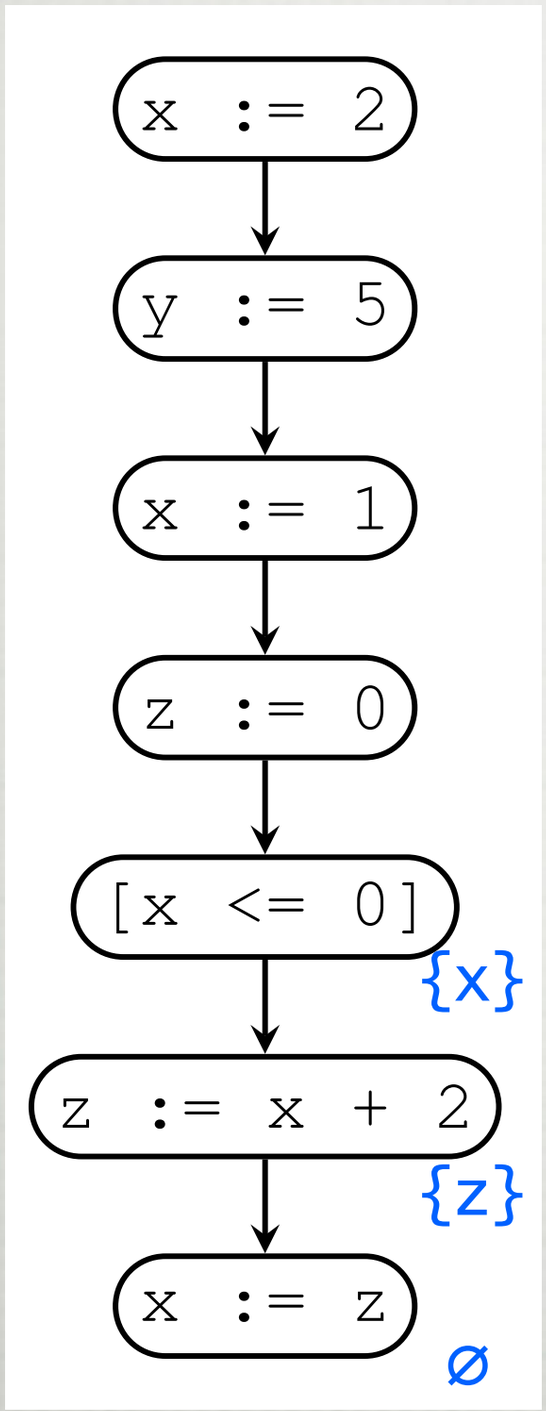
Live Variable Analysis on Paths



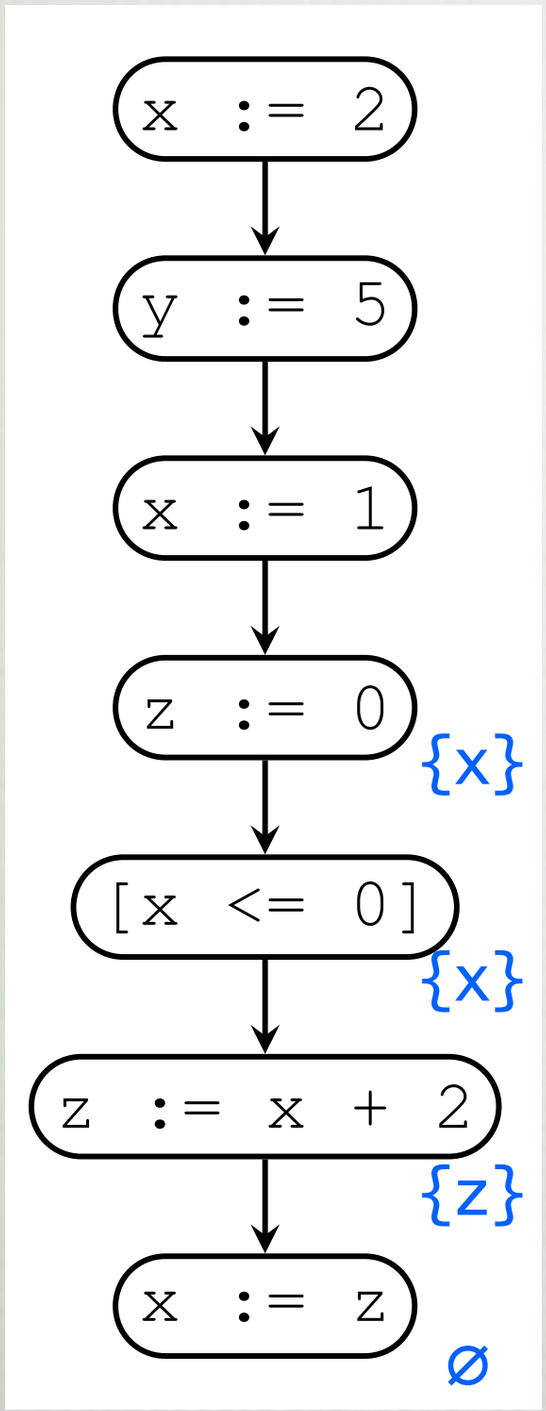
Live Variable Analysis on Paths



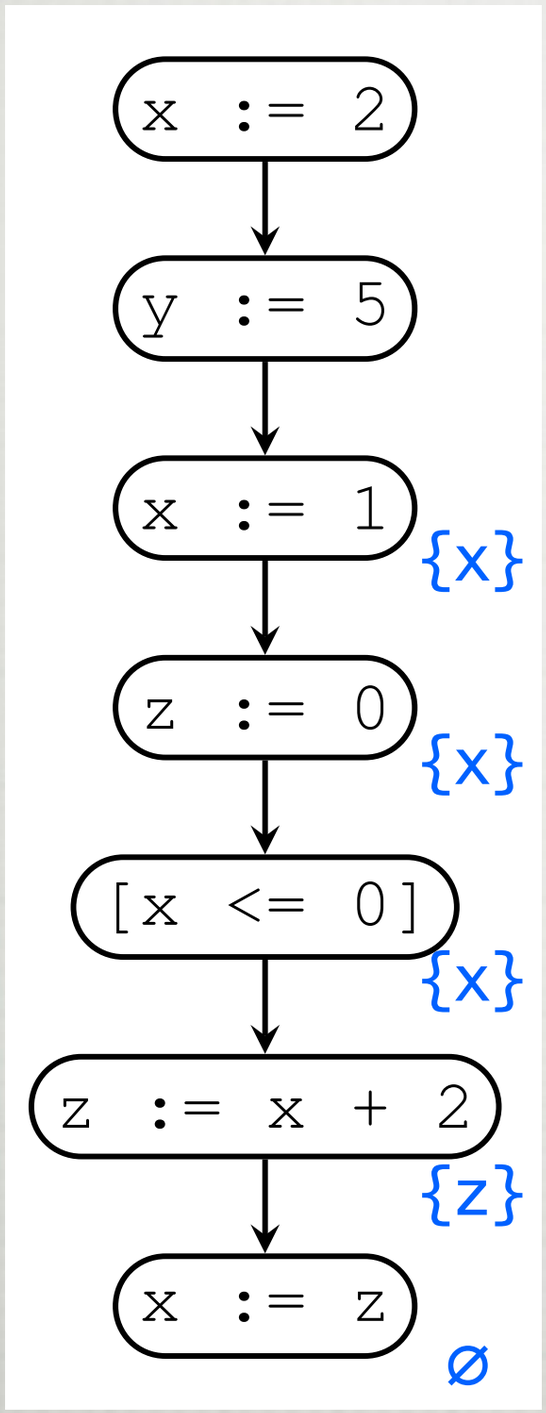
Live Variable Analysis on Paths



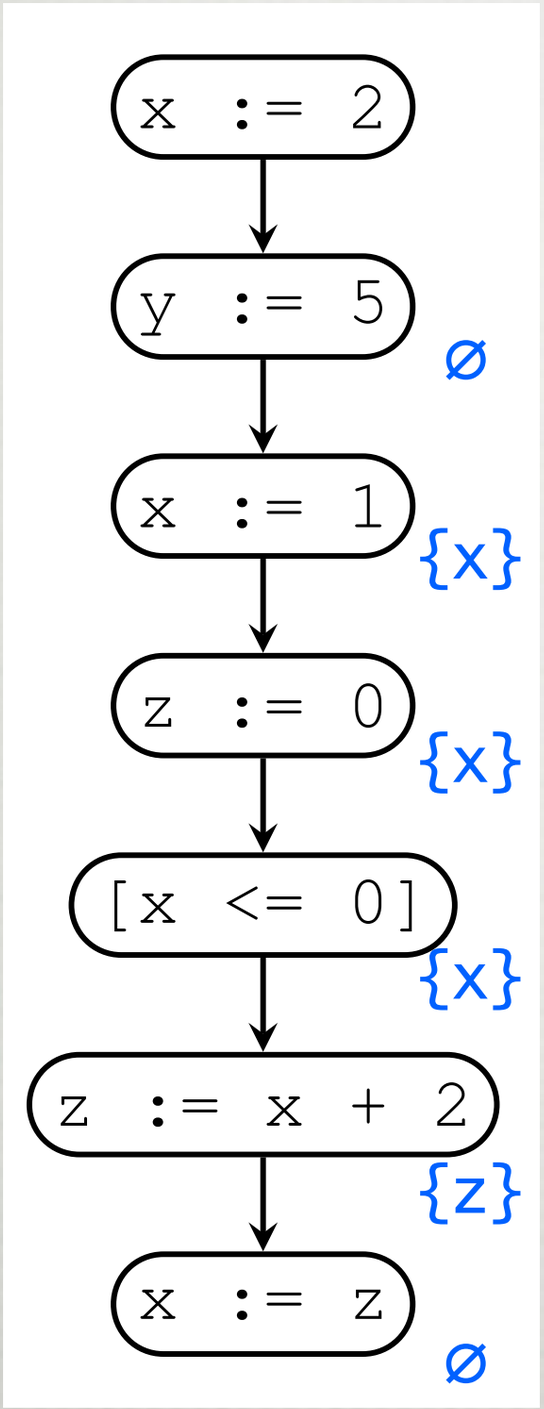
Live Variable Analysis on Paths



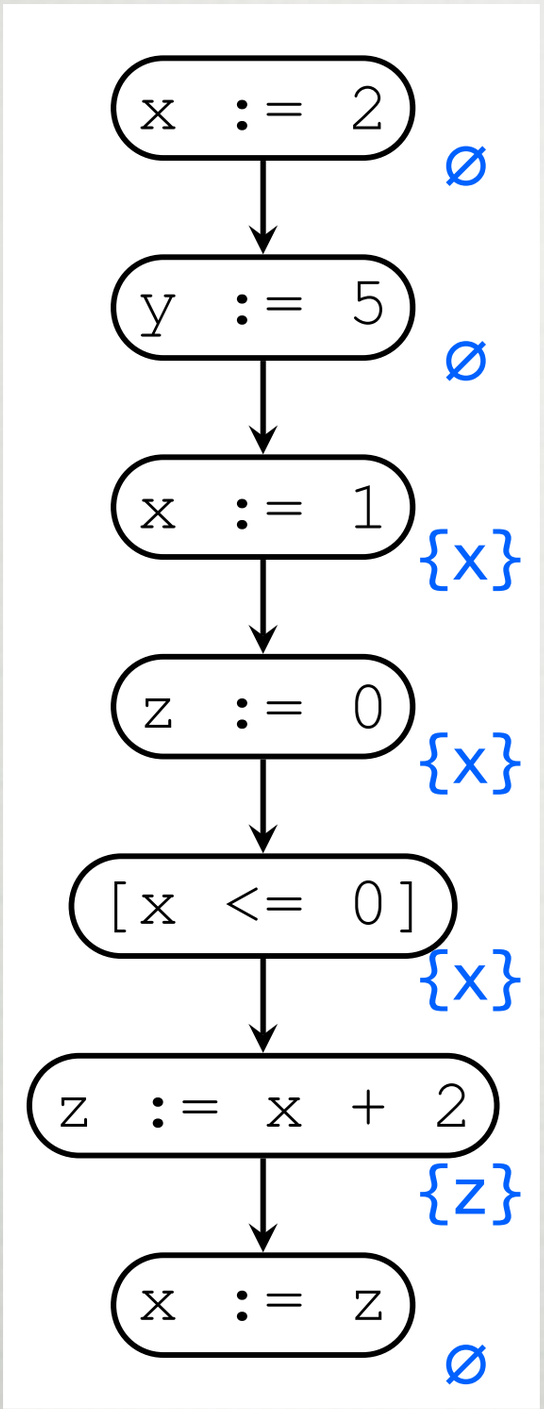
Live Variable Analysis on Paths



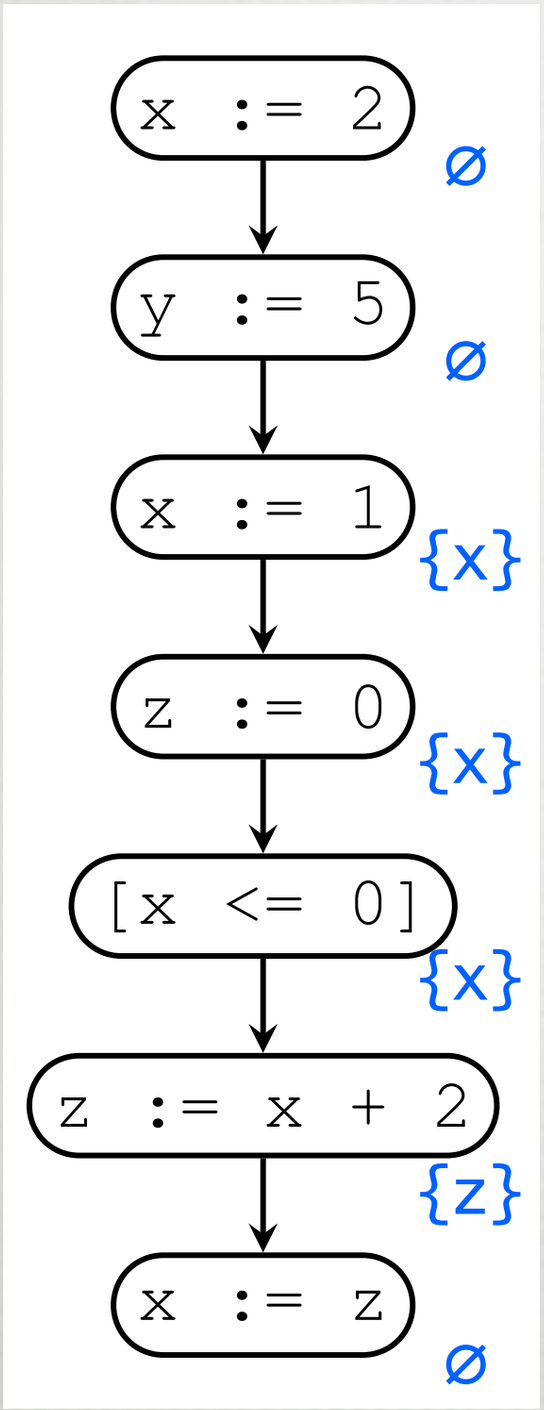
Live Variable Analysis on Paths



Live Variable Analysis on Paths



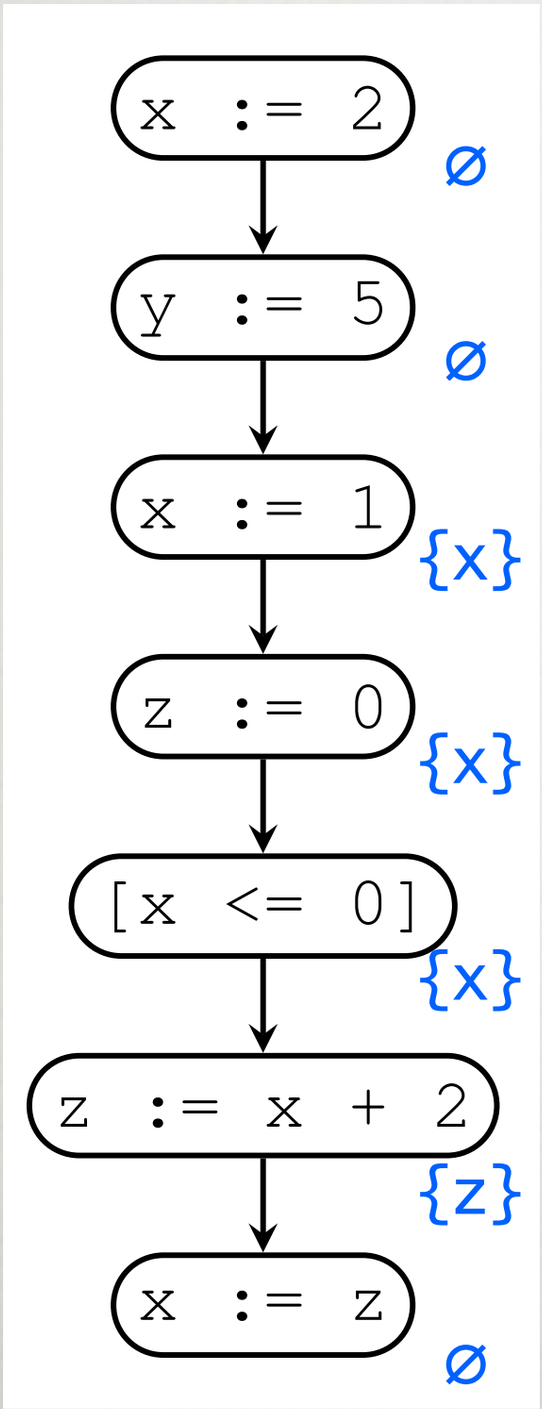
Live Variable Analysis on Paths



Define for each lablel `l`:

- `read(l)`: the set of variables `l` reads from.
- `write(l)`: the set of variables `l` writes to.

Live Variable Analysis on Paths



Define for each lablel l:

- $read(l)$: the set of variables l reads from.
- $write(l)$: the set of variables l writes to.

For a path $l_1 \ldots l_n$:

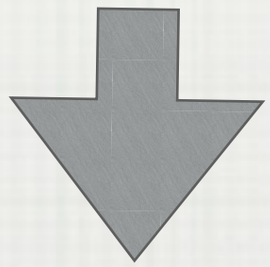
$$
\begin{aligned}
live(\ell_n) &= \emptyset \\
live(\ell_{i-1}) &= live(\ell_i) \setminus write(\ell_i) \cup read(\ell_i)
\end{aligned}
$$

Live Variable Analysis on Paths

$$
\begin{aligned}
live(\ell_n) &= \emptyset \\
live(\ell_{i-1}) &= live(\ell_i) \setminus write(\ell_i) \cup read(\ell_i)
\end{aligned}
$$

Live Variable Analysis on Paths

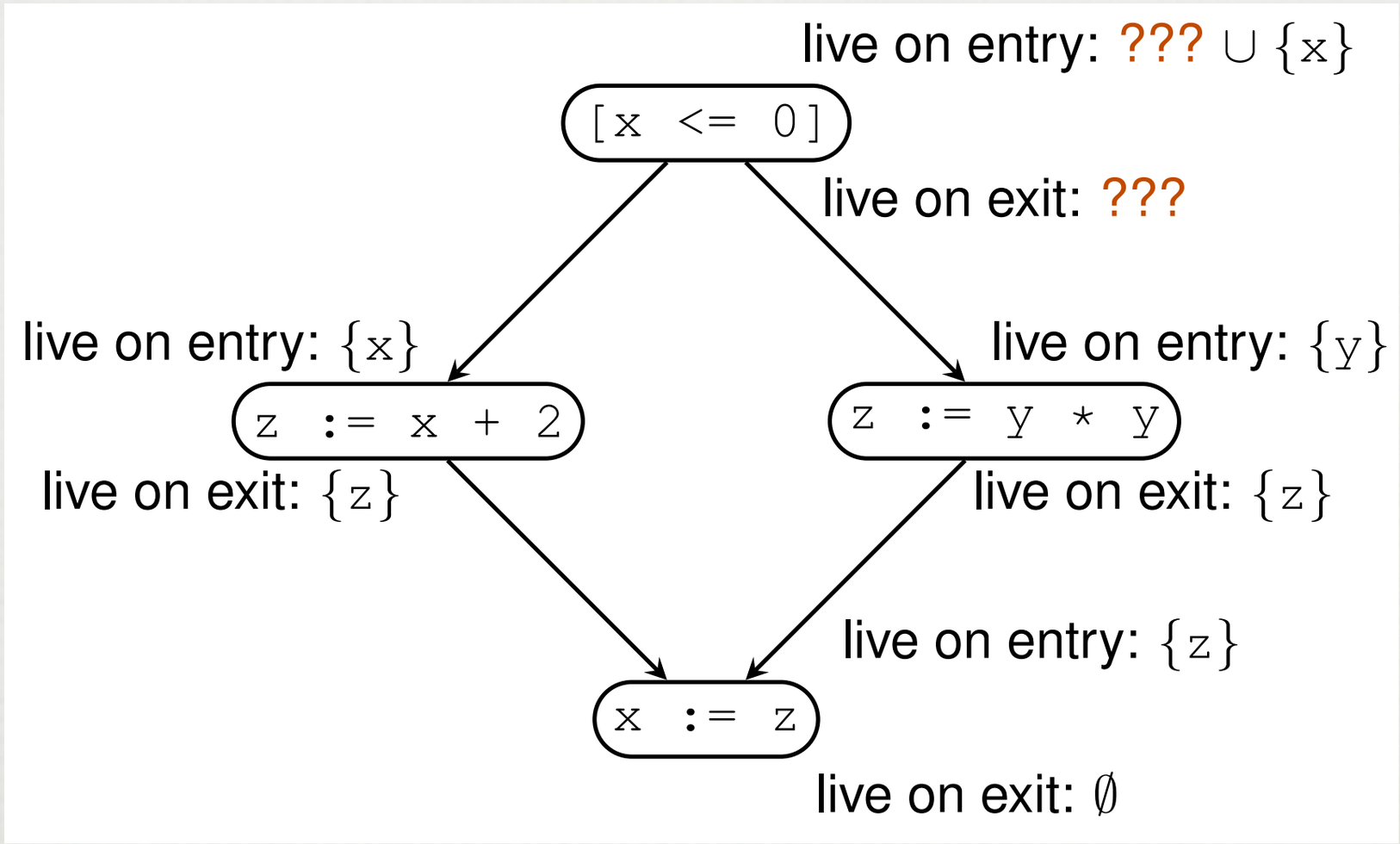
$$
\begin{aligned}
live(\ell_n) &= \emptyset \\
live(\ell_{i-1}) &= live(\ell_i) \setminus write(\ell_i) \cup read(\ell_i)
\end{aligned}
$$



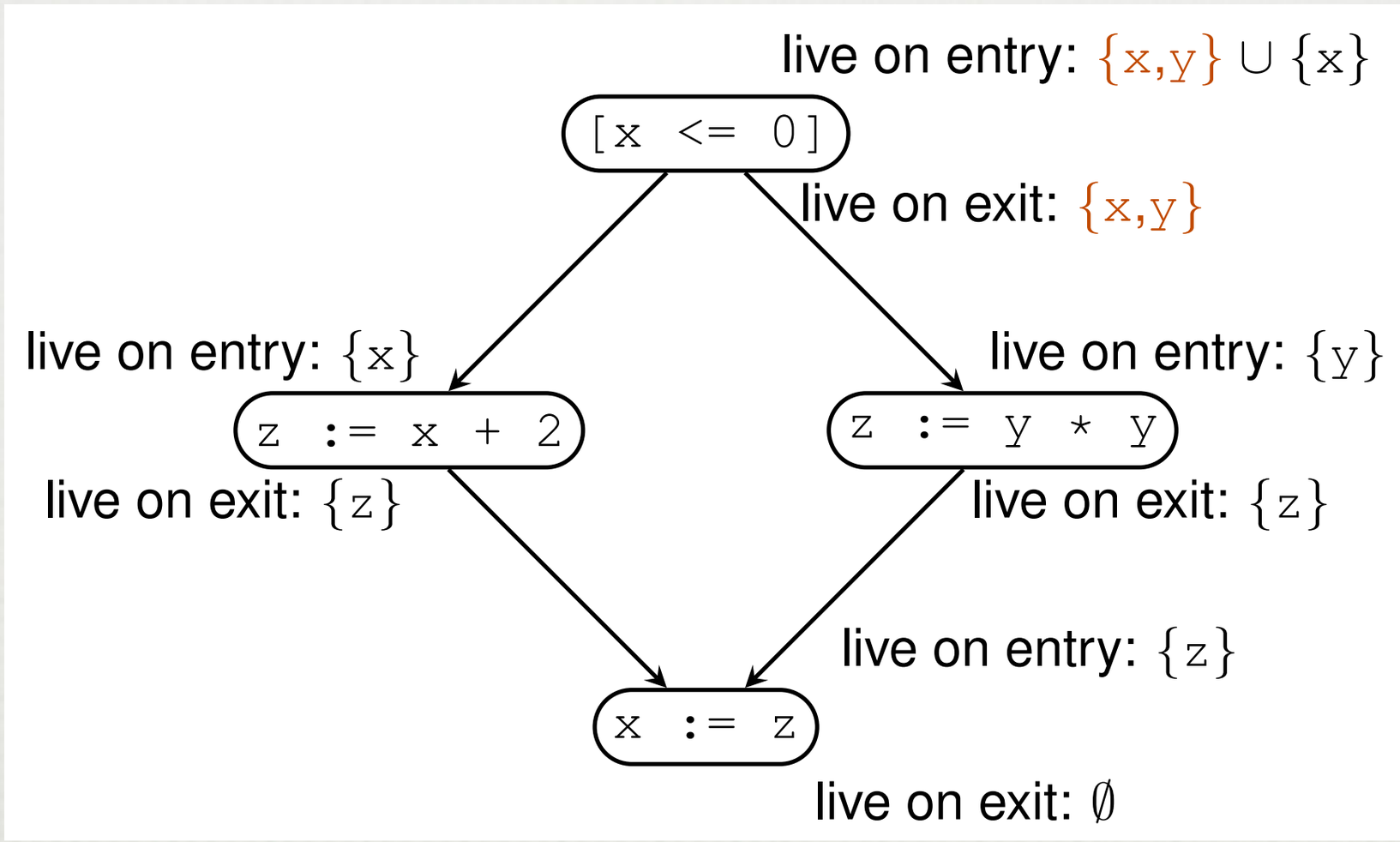
$$
\begin{aligned}
live_{exit}(\ell_n) &= \emptyset \\
live_{entry}(\ell) &= live_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
live_{exit}(\ell_{i-1}) &= live_{entry}(\ell_i)
\end{aligned}
$$

What if we have branches along the way?

Live Variable Analysis on DAGs



Live Variable Analysis on DAGs



Live Variable Analysis on DAGs

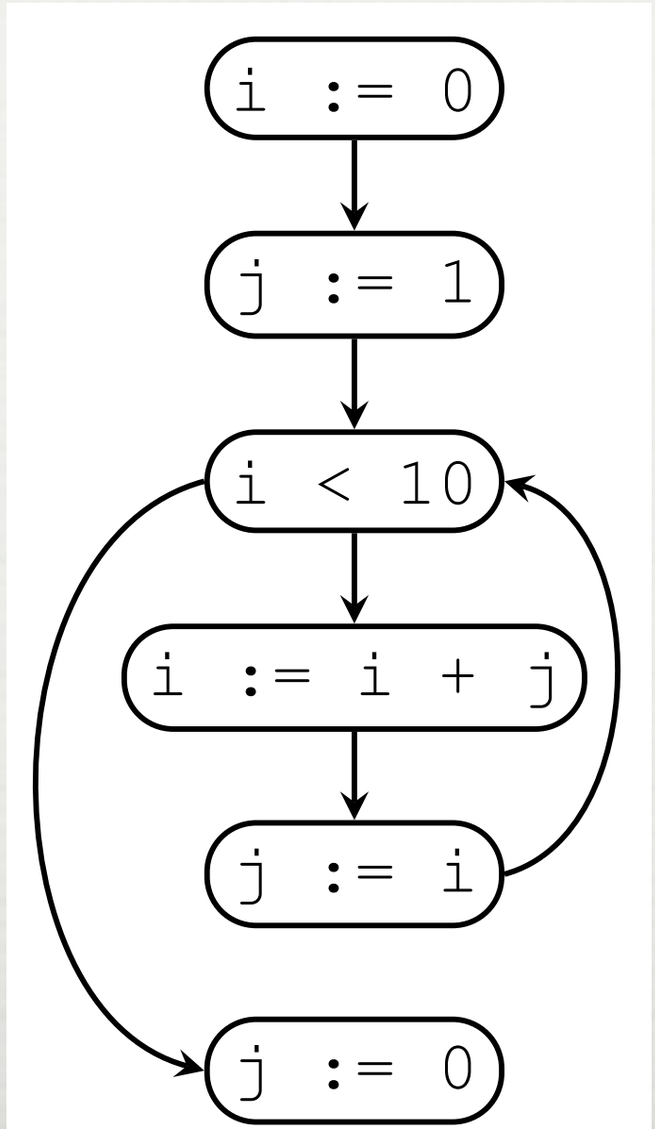
$$
\begin{aligned}
live_{exit}(\ell_n) &= \emptyset \\
live_{entry}(\ell) &= live_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
live_{exit}(\ell_{i-1}) &= live_{entry}(\ell_i)
\end{aligned}
$$

Live Variable Analysis on DAGs

$$
\begin{aligned}
live_{entry}(\ell) &= live_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
live_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} live_{entry}(\ell')
\end{aligned}
$$

What if there are cycles in the graph?

What if there are cycles in the graph?



Live Variable Analysis on Graphs

Consider the equations below as a system of constraints. A a solution is any assignment of live sets that satisfies the equations below:

$$LV_{entry}(\ell) = LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$$

$$LV_{exit}(\ell) = \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$$

Live Variable Analysis on Graphs

Consider the equations below as a system of constraints. A a solution is any assignment of live sets that satisfies the equations below:

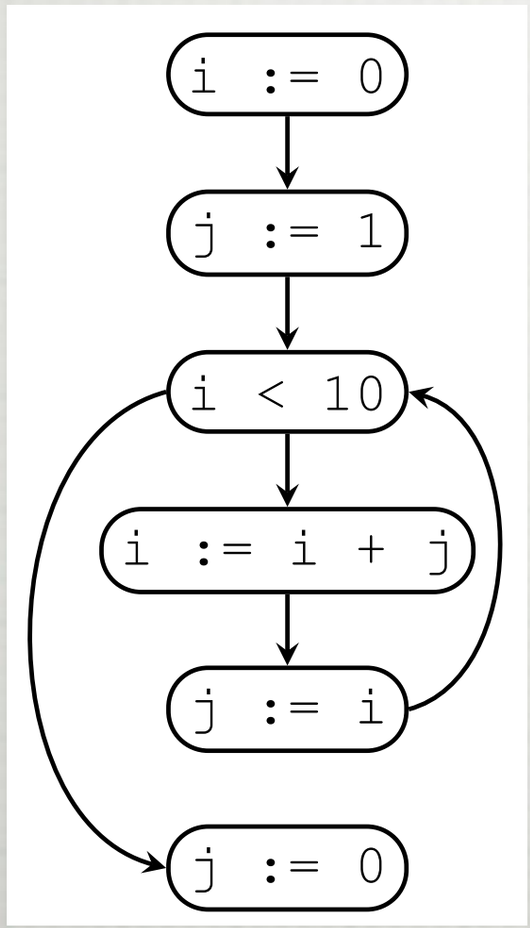
$$LV_{entry}(\ell) = LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$$

$$LV_{exit}(\ell) = \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$$

Our desired live variables information certainly is a solution to this constraint system.

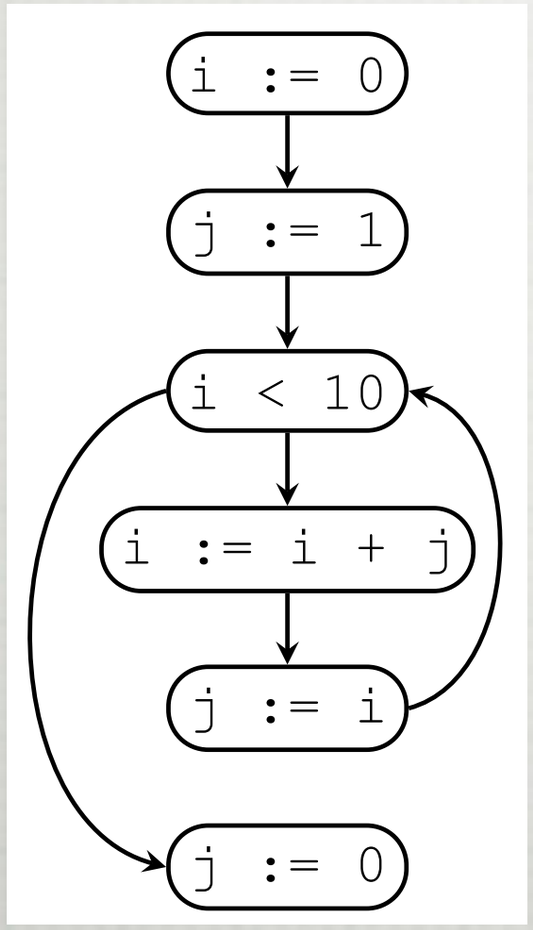
Solution not necessarily unique

$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$



Solution not necessarily unique

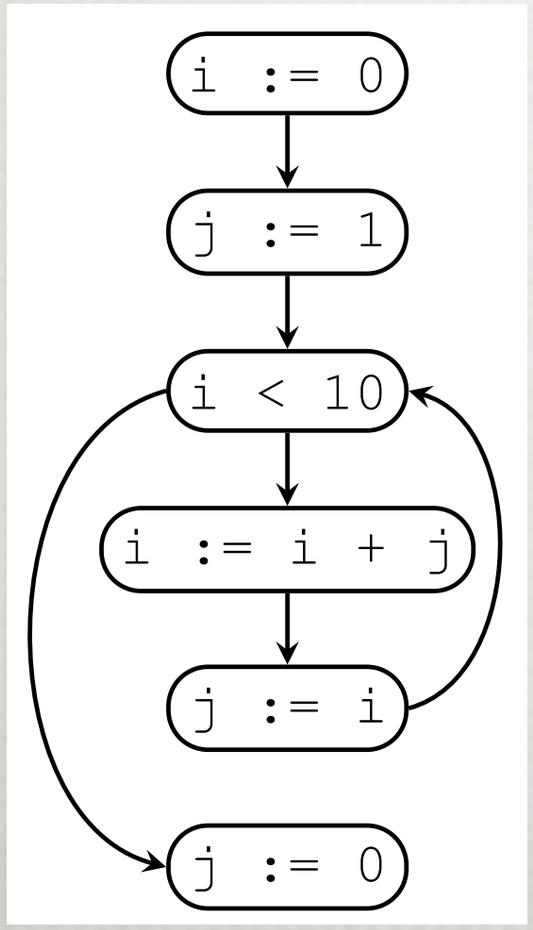
$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$



Label	LV_{exit}	LV_{entry}
`i := 0`	{i}	∅
`j := 1`	{i,j}	{i}
`i < 10`	{i,j}	{i,j}
`i := i + j`	{i}	{i,j}
`j := i`	{i,j}	{i}
`j := 0`	∅	∅

Solution not necessarily unique

$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$



Label	LV_{exit}	LV_{entry}
i := 0	{i,k}	{k}
j := 1	{i,j,k}	{i,k}
i < 10	{i,j,k}	{i,j,k}
i := i + j	{i,k}	{i,j,k}
j := i	{i,j,k}	{i,k}
j := 0	∅	∅

Key Observation

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

Key Observation

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LVe_{exit}(l)$$

Key Observation

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LVe_{exit}(l)$$

The desired solution is the least solution!

Key Observation

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LVe_{exit}(l)$$

The desired solution is the least solution!

Our comparator is the subset relation!

Can we compute the least solution?

The Worklist Algorithm

Algorithm 1 Worklist method

$worklist \leftarrow empty$
for each $\ell \in V$ **do**
  $LV_{exit}(\ell) \leftarrow \emptyset$
  $LV_{entry}(\ell) \leftarrow LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$
  add ℓ to $worklist$
end for
while $worklist$ not empty **do**
  pick ℓ from $worklist$
  $old \leftarrow LV_{entry}(\ell)$
  $LV_{exit}(\ell) \leftarrow \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$
  $LV_{entry}(\ell) \leftarrow LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$
  if $old \neq LV_{entry}(\ell)$ **then**
    Add predecessors of ℓ to $worklist$
  end if
end while

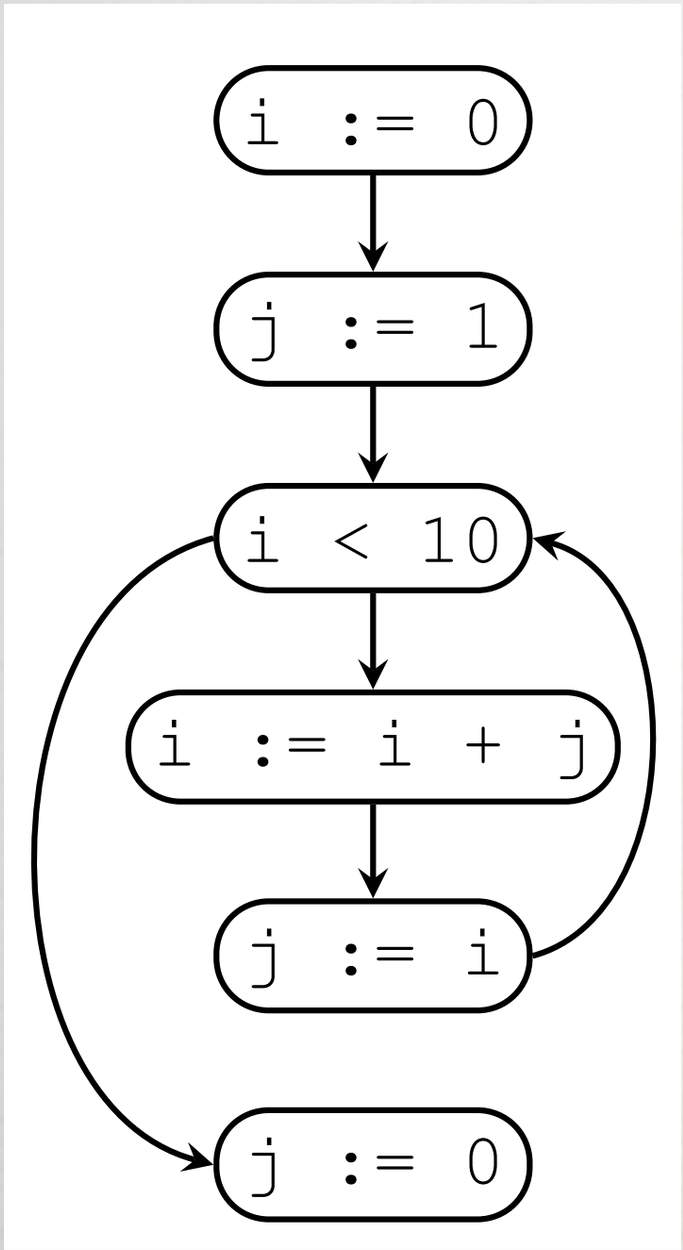
The Worklist Algorithm

Algorithm 1 Worklist method

$worklist \leftarrow empty$
for each $\ell \in V$ **do**
  $LV_{exit}(\ell) \leftarrow \emptyset$
  $LV_{entry}(\ell) \leftarrow LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$
  add ℓ to $worklist$
end for
while $worklist$ not empty **do**
  pick ℓ from $worklist$
  $old \leftarrow LV_{entry}(\ell)$
  $LV_{exit}(\ell) \leftarrow \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$
  $LV_{entry}(\ell) \leftarrow LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$
  if $old \neq LV_{entry}(\ell)$ **then**
    Add predecessors of ℓ to $worklist$
  end if
end while

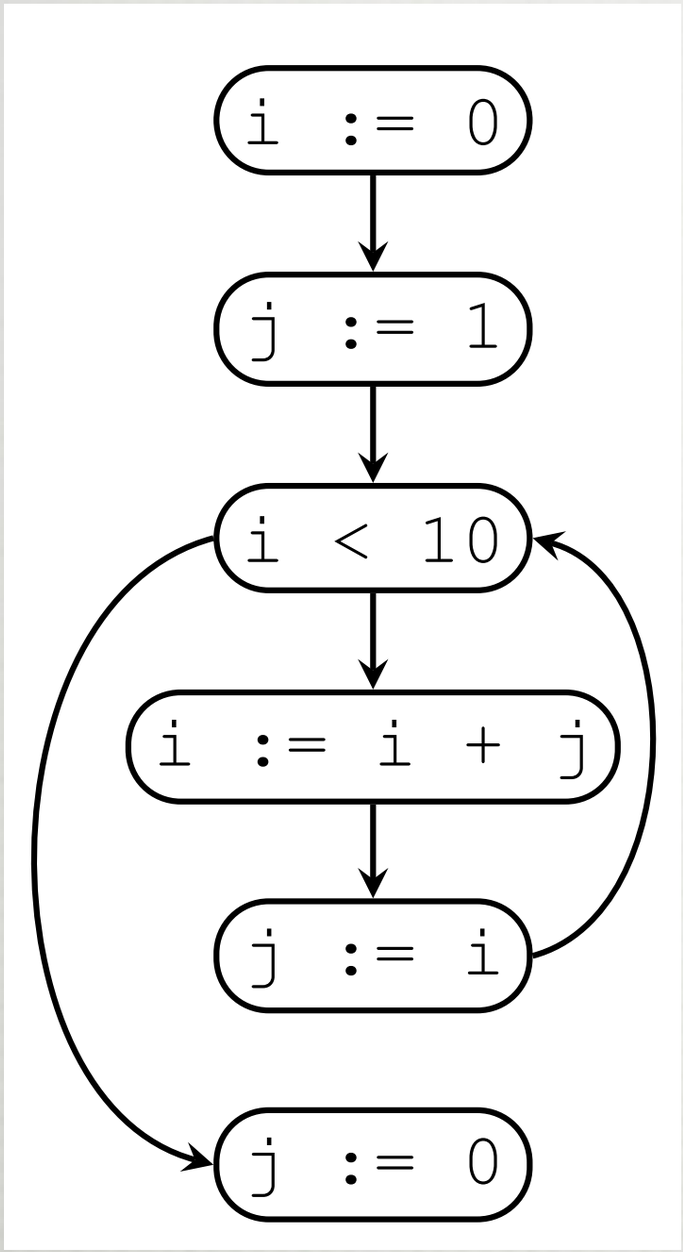
$$\forall \ell. LV_{entry}(\ell) = LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$$
$$LV_{exit}(\ell) \neq \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell') \Rightarrow \ell \in worklist$$

Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

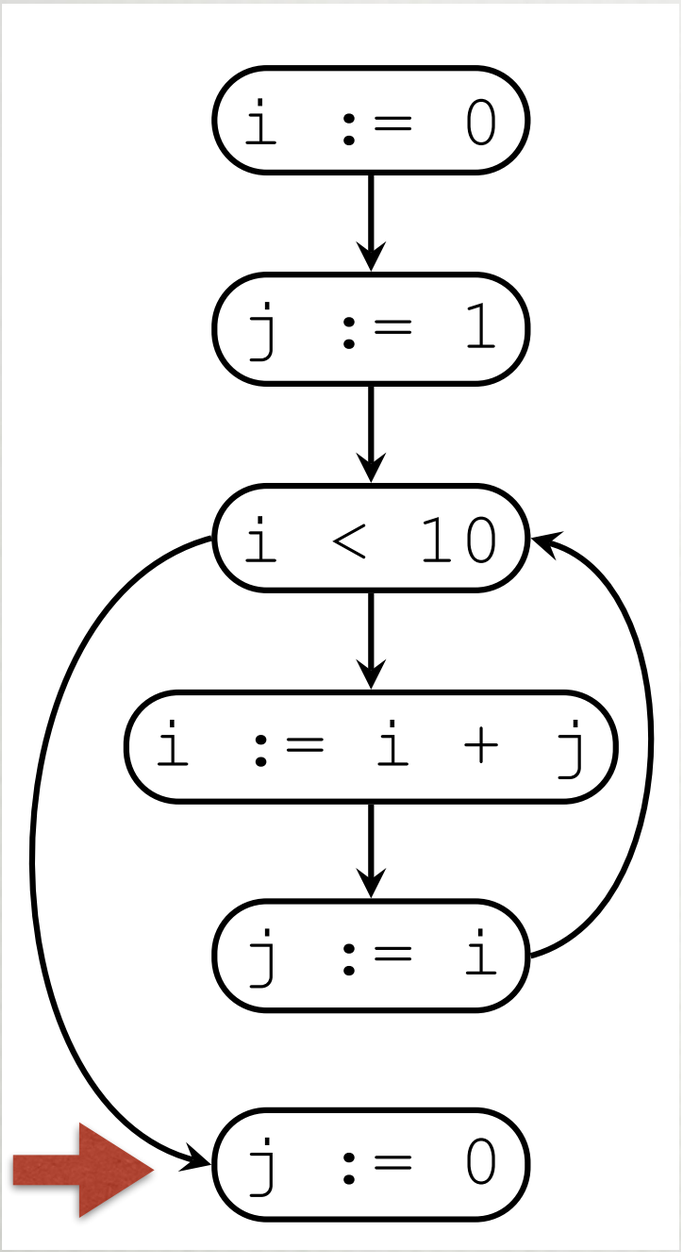
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	$\emptyset$	$\emptyset$	$\surd$
`j := 1`	$\emptyset$	$\emptyset$	$\surd$
`i < 10`	$\emptyset$	{i}	$\surd$
`i := i + j`	$\emptyset$	{i,j}	$\surd$
`j := i`	$\emptyset$	{i}	$\surd$
`j := 0`	$\emptyset$	$\emptyset$	$\surd$

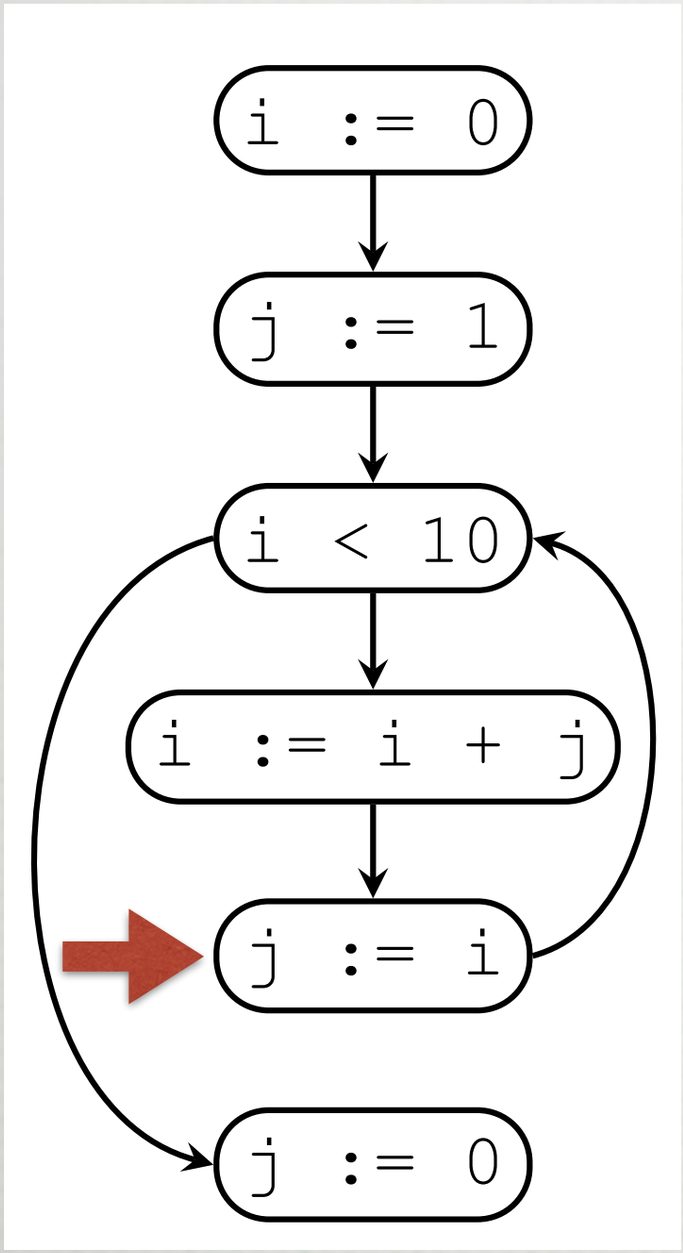
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	$\emptyset$	$\emptyset$	$\surd$
`j := 1`	$\emptyset$	$\emptyset$	$\surd$
`i < 10`	$\emptyset$	{i}	$\surd$
`i := i + j`	$\emptyset$	{i,j}	$\surd$
`j := i`	$\emptyset$	{i}	$\surd$
`j := 0`	$\emptyset$	$\emptyset$	$\surd$

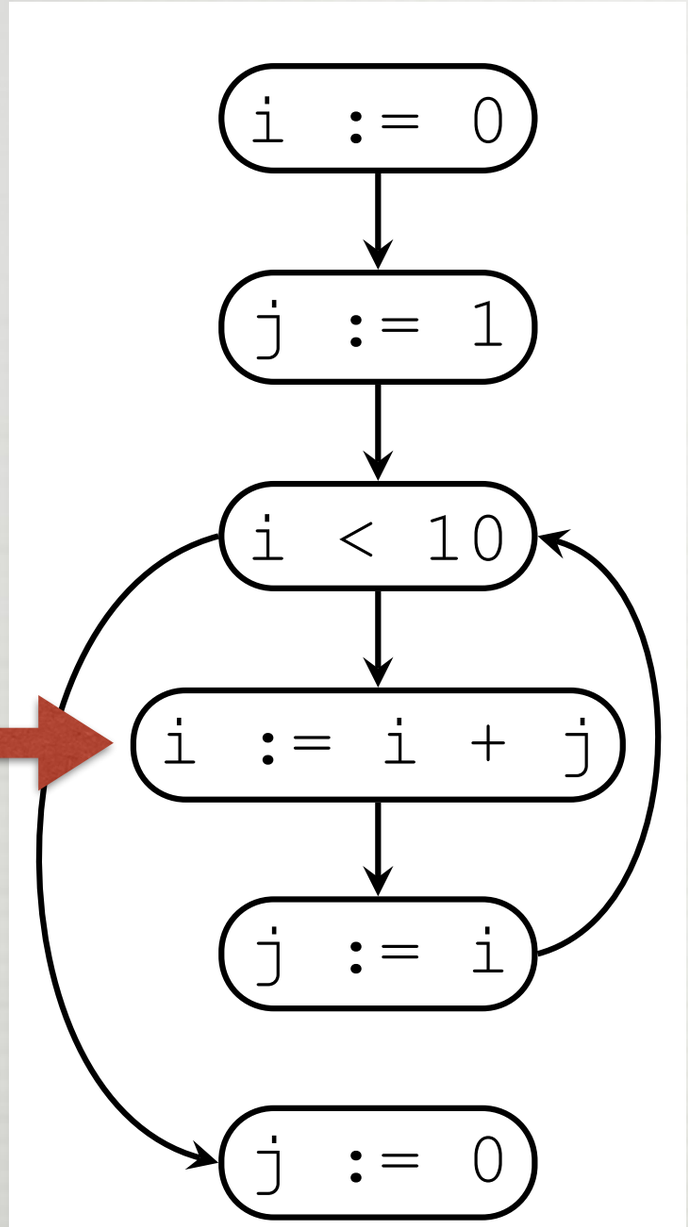
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	$\emptyset$	$\emptyset$	$\surd$
`j := 1`	$\emptyset$	$\emptyset$	$\surd$
`i < 10`	$\emptyset$	{i}	$\surd$
`i := i + j`	$\emptyset$	{i,j}	$\surd$
`j := i`	$\emptyset$	{i}	$\surd$
`j := 0`	$\emptyset$	$\emptyset$	

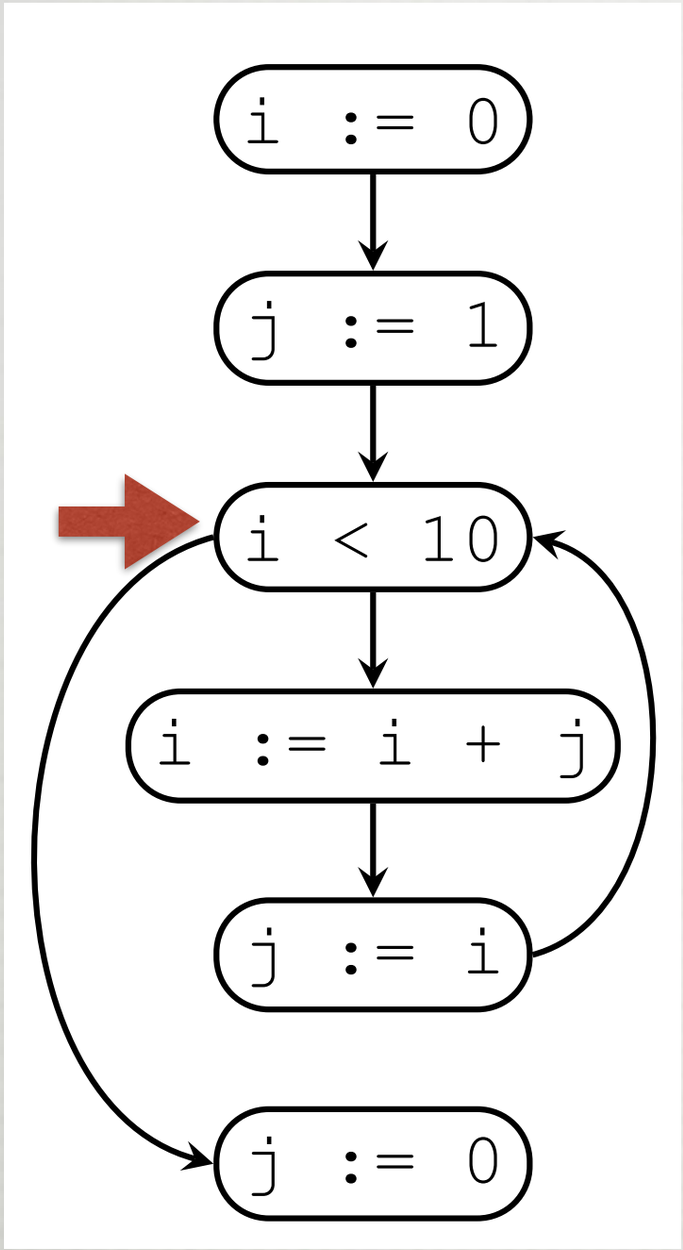
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
i := 0	$\emptyset$	$\emptyset$	$\surd$
j := 1	$\emptyset$	$\emptyset$	$\surd$
i < 10	$\emptyset$	{i}	$\surd$
i := i + j	$\emptyset$	{i,j}	$\surd$
j := i	{i}	{i}	
j := 0	$\emptyset$	$\emptyset$	

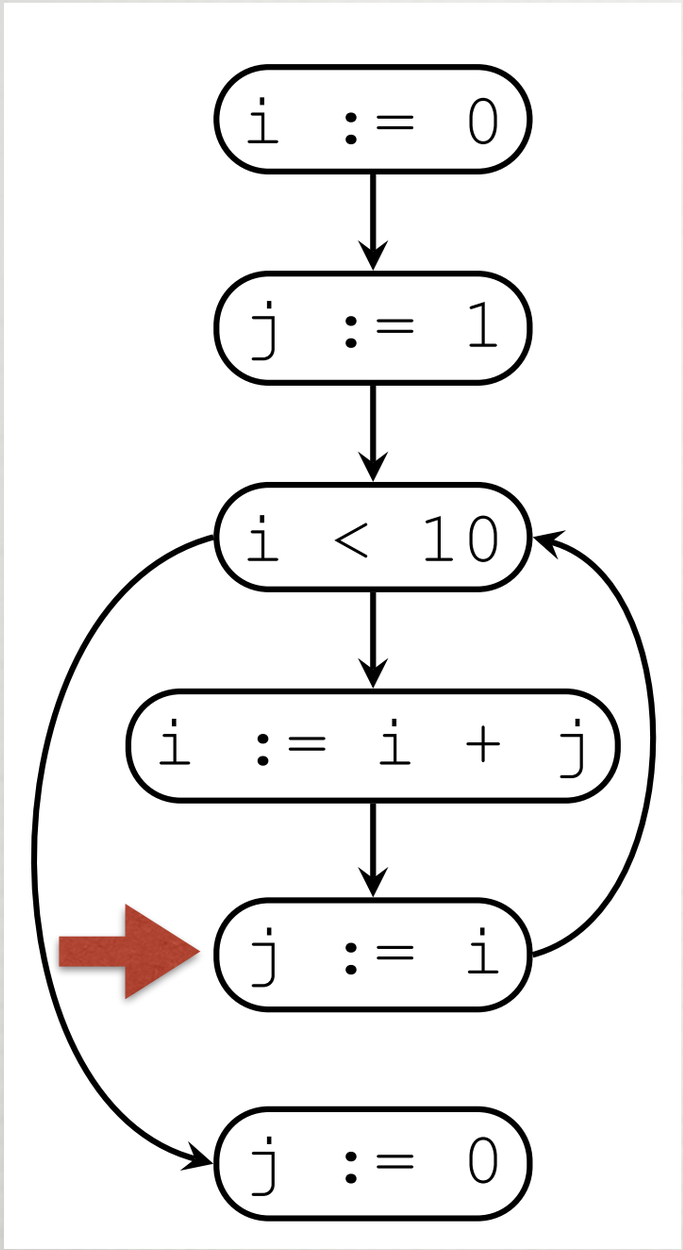
Worklist Algorithm Example



$$LV_{entry}(\ell) = LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$$
$$LV_{exit}(\ell) = \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	$\emptyset$	$\emptyset$	$\surd$
`j := 1`	$\emptyset$	$\emptyset$	$\surd$
`i < 10`	$\emptyset$	{i}	$\surd$
`i := i + j`	{i}	{i,j}	
`j := i`	{i}	{i}	
`j := 0`	$\emptyset$	$\emptyset$	

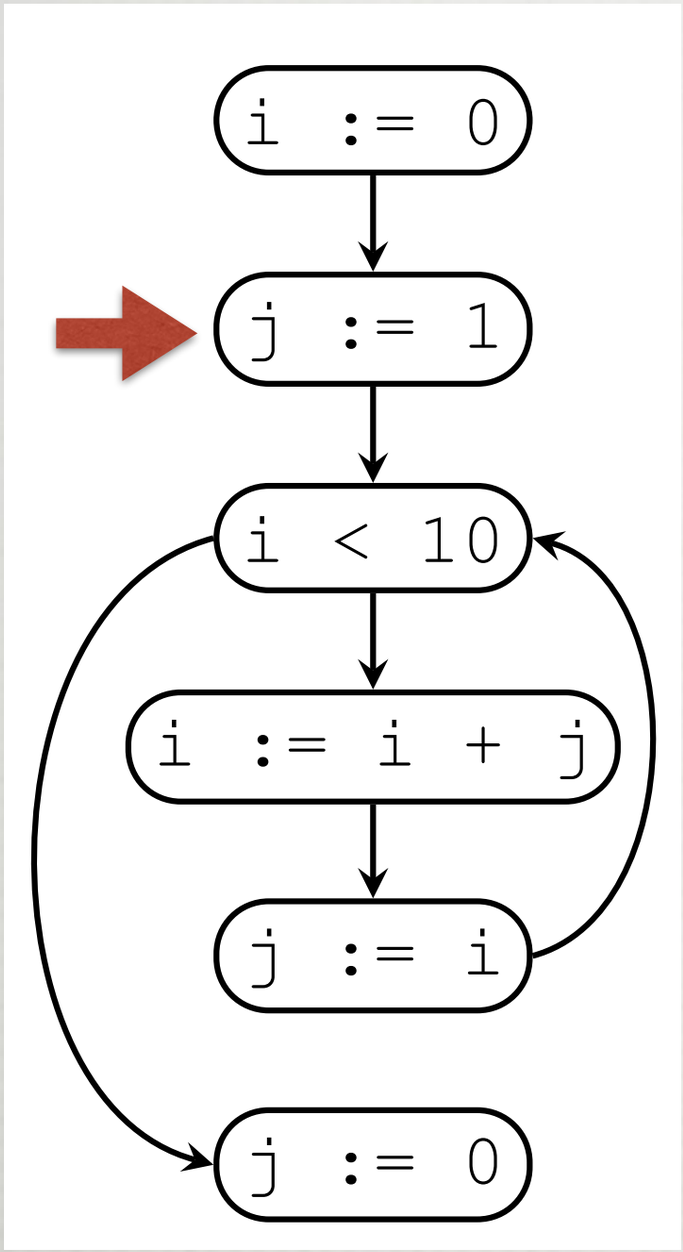
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
i := 0	$\emptyset$	$\emptyset$	$\surd$
j := 1	$\emptyset$	$\emptyset$	$\surd$
i < 10	{i,j}	{i,j}	
i := i + j	{i}	{i,j}	
j := i	{i}	{i}	$\surd$
j := 0	$\emptyset$	$\emptyset$	

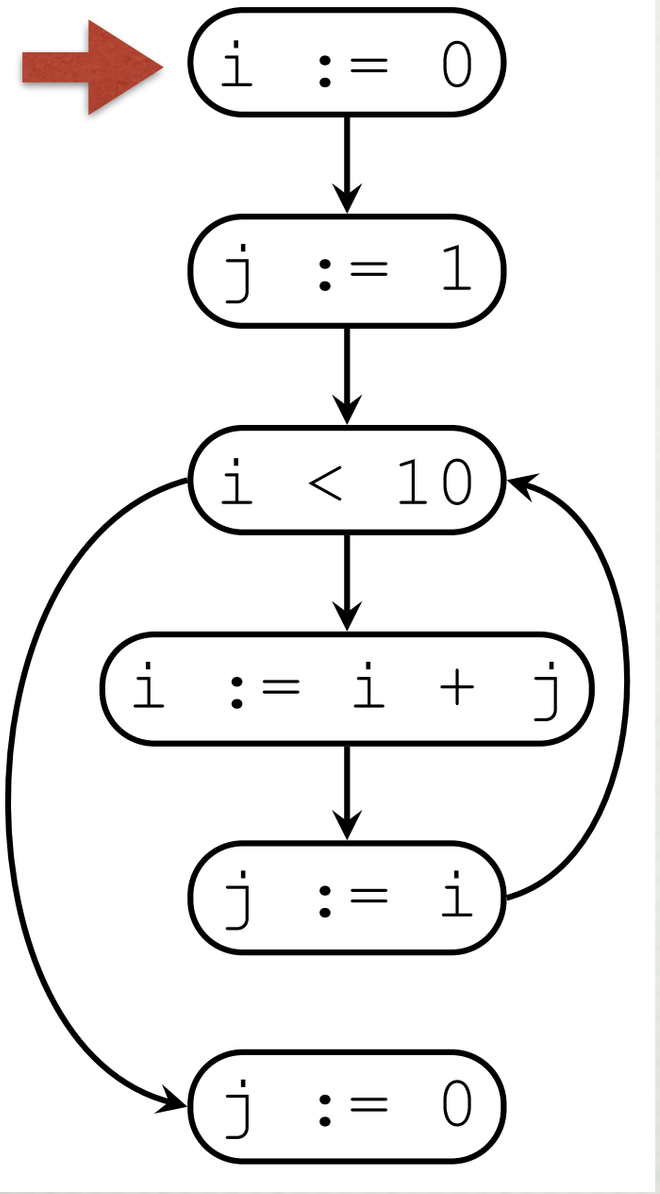
Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
i := 0	$\emptyset$	$\emptyset$	$\surd$
j := 1	$\emptyset$	$\emptyset$	$\surd$
i < 10	{i,j}	{i,j}	
i := i + j	{i}	{i,j}	
j := i	{i,j}	{i}	
j := 0	$\emptyset$	$\emptyset$	

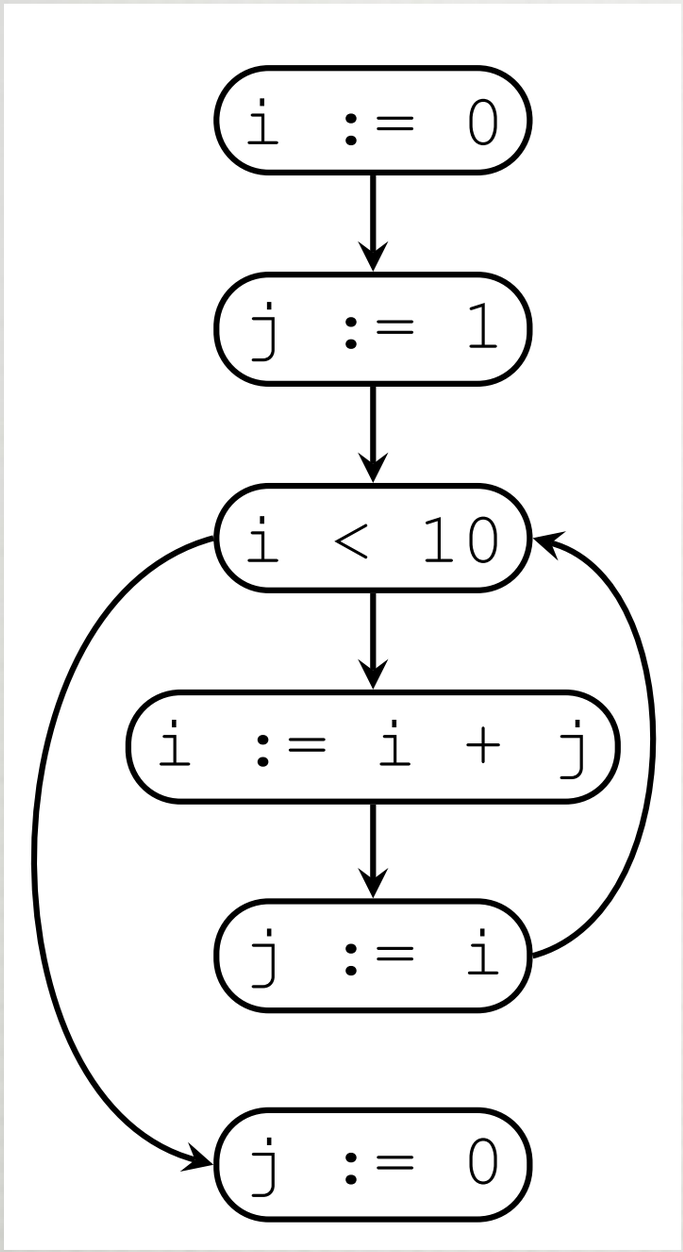
Worklist Algorithm Example



$$LV_{entry}(\ell) = LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell)$$
$$LV_{exit}(\ell) = \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	$\emptyset$	$\emptyset$	$\surd$
`j := 1`	{i,j}	{i}	
`i < 10`	{i,j}	{i,j}	
`i := i + j`	{i}	{i,j}	
`j := i`	{i,j}	{i}	
`j := 0`	$\emptyset$	$\emptyset$	

Worklist Algorithm Example



$$
\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\
LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell')
\end{aligned}
$$

Label	LV_{exit}	LV_{entry}	In WL
`i := 0`	{i}	∅	
`j := 1`	{i,j}	{i}	
`i < 10`	{i,j}	{i,j}	
`i := i + j`	{i}	{i,j}	
`j := i`	{i,j}	{i}	
`j := 0`	∅	∅	

Correctness

Worklist Algorithm Correctness

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.
- ☐ Either the sets increase, or the worklist decreases.

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.
- ☐ Either the sets increase, or the worklist decreases.

Why does the worklist algorithm compute the least solution?

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.
- ☐ Either the sets increase, or the worklist decreases.

Why does the worklist algorithm compute the least solution?

- ☐ Loop invariant for the algorithm:

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.
- ☐ Either the sets increase, or the worklist decreases.

Why does the worklist algorithm compute the least solution?

- ☐ Loop invariant for the algorithm:

$$LV_{exit}(\ell) \subseteq live_{exit}(\ell)$$
$$LV_{entry}(\ell) \subseteq live_{entry}(\ell)$$

Worklist Algorithm Correctness

Why does the worklist algorithm always terminate?

- ☐ $LV_{entry}(l)/LV_{exit}(l)$: always increase.
- ☐ chains in $\mathcal{P}(Vars)$ are finite.
- ☐ Either the sets increase, or the worklist decreases.

Why does the worklist algorithm compute the least solution?

- ☐ Loop invariant for the algorithm:
- ☐ sets are increasing

$$LV_{exit}(\ell) \subseteq live_{exit}(\ell)$$
$$LV_{entry}(\ell) \subseteq live_{entry}(\ell)$$

What about other analyses?

We want to infer program information at program locations.

We need a way of combining information.

Let us use a bit of basic algebra!

First Structure

Semi-Lattices

A (meet) *semi-lattice* $\mathbf{L} = (S, \sqcap)$ is a set S with a binary operation, called *meet* ($\sqcap$), that has the following properties:

(1) For all $x, y \in S$, there exist a unique $z \in S$ such that $x \sqcap y = z$ (CLOSURE).

(2) For all $x, y, z \in S$, we have

$$x \sqcap x = x \quad \text{(idempotence)}$$

$$x \sqcap y = y \sqcap x \quad \text{(commutativity)}$$

$$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z \quad \text{(associativity)}$$

Complete Semi-Lattices

The unit for $\sqcap$ is $\top$:

$$\forall x : \; x \sqcap \top = \top \sqcap x = x$$

Complete Semi-Lattices

The unit for $\sqcap$ is $\top$:

$$\forall x: \; x \sqcap \top = \top \sqcap x = x$$

The meet semi-lattice is called complete if $\top \in \mathbb{L}$

We need a way of comparing information.

Second Structure

Partially Ordered Sets

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

Partially Ordered Sets

Information Order

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

Partially Ordered Sets

Information Order

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

$x \sqsubseteq x$ (reflexivity).

If $x \sqsubseteq y$ and $y \sqsubseteq z$, then $x \sqsubseteq z$ (transitivity).

If $x \sqsubseteq y$ and $y \sqsubseteq x$, then $x = y$ (antisymmetry).

The Connection

Combining should lead to either less (or equal) information.

The Connection Between The Structures

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

The Connection Between The Structures

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

When combining does not give us anything new!

The Connection Between The Structures

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$x \sqsubseteq y$ if and only if $x \sqcap y = x$

When combining does not give us anything new!

$\sqsubseteq$ is provably a partial order relation.

The Connection Between The Structures

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$x \sqsubseteq y$ if and only if $x \sqcap y = x$

When combining does not give us anything new!

$\sqsubseteq$ is provably a partial order relation.

$\sqcap$ is provably the greatest lower bound defined based on $\sqsubseteq$.

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

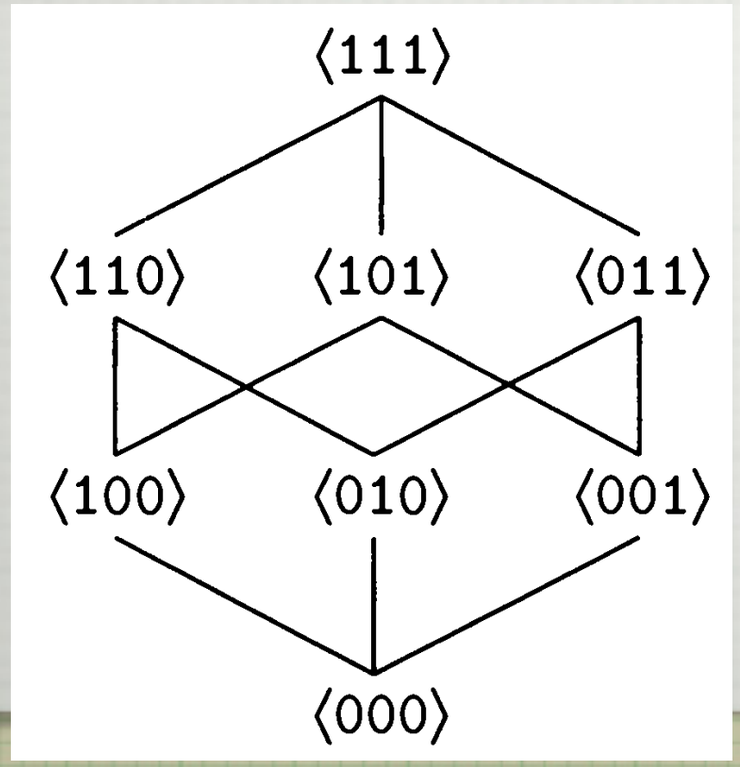
$(S, \sqcap)$ is provably a semi-lattice.

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

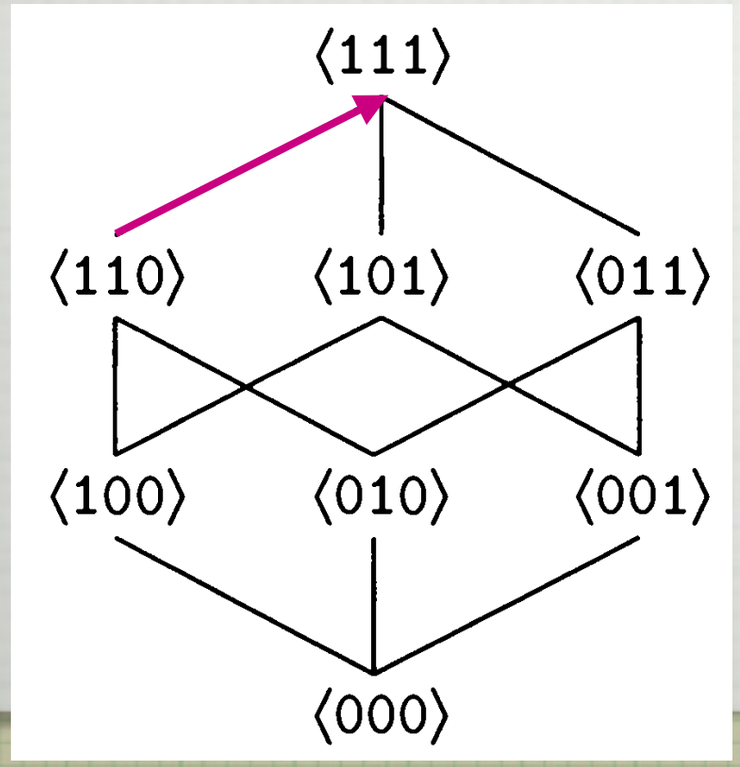


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

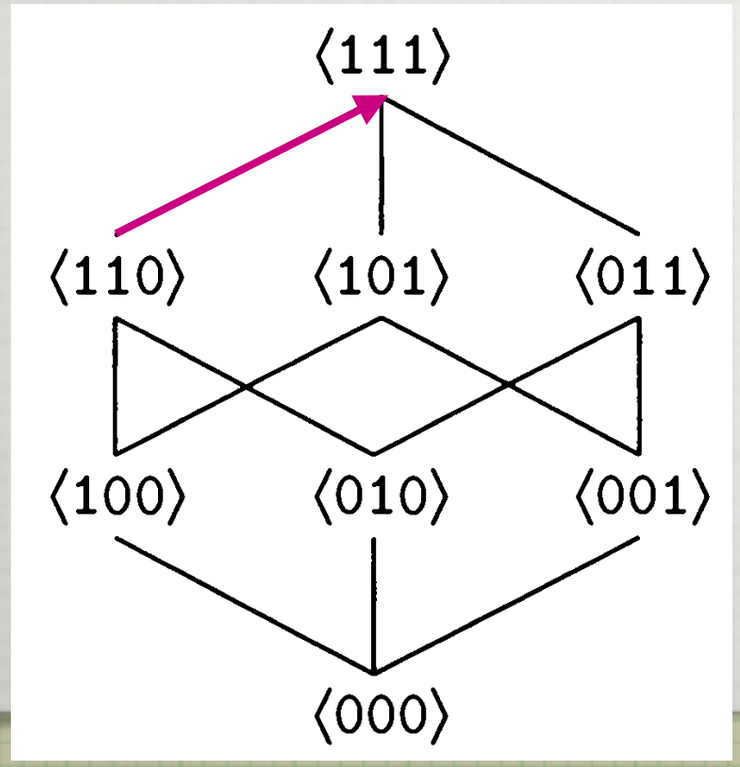


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.



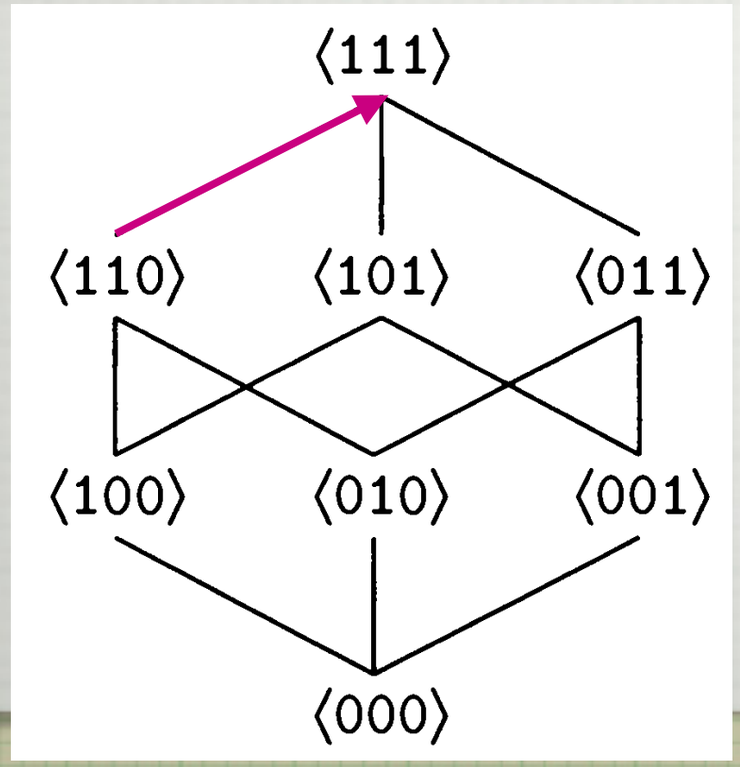
$$\sqcap = \wedge$$

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.



$$\sqcap = \wedge$$

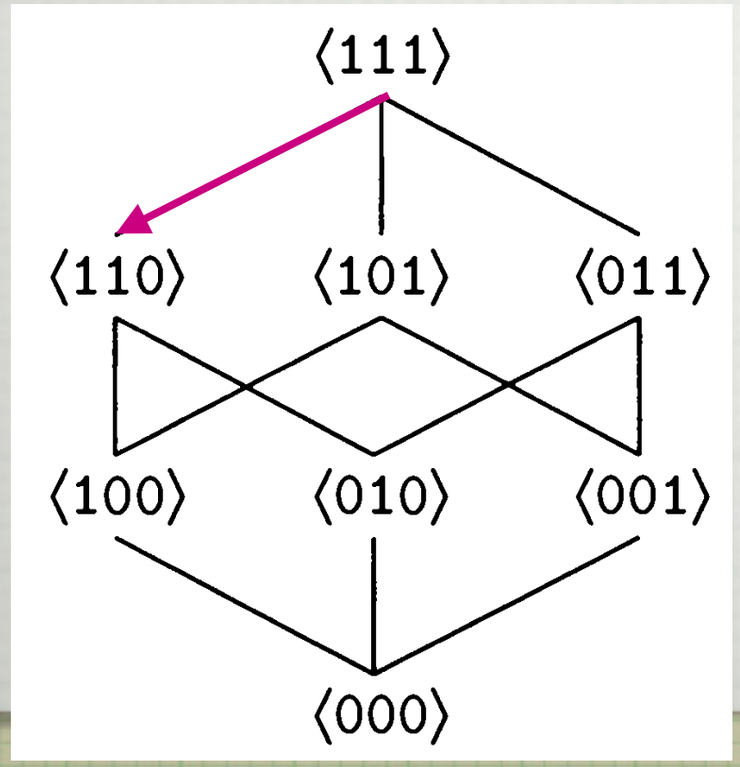
$$\langle 110 \rangle \sqcap \langle 011 \rangle = \langle 110 \rangle \wedge \langle 011 \rangle = \langle 010 \rangle$$

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

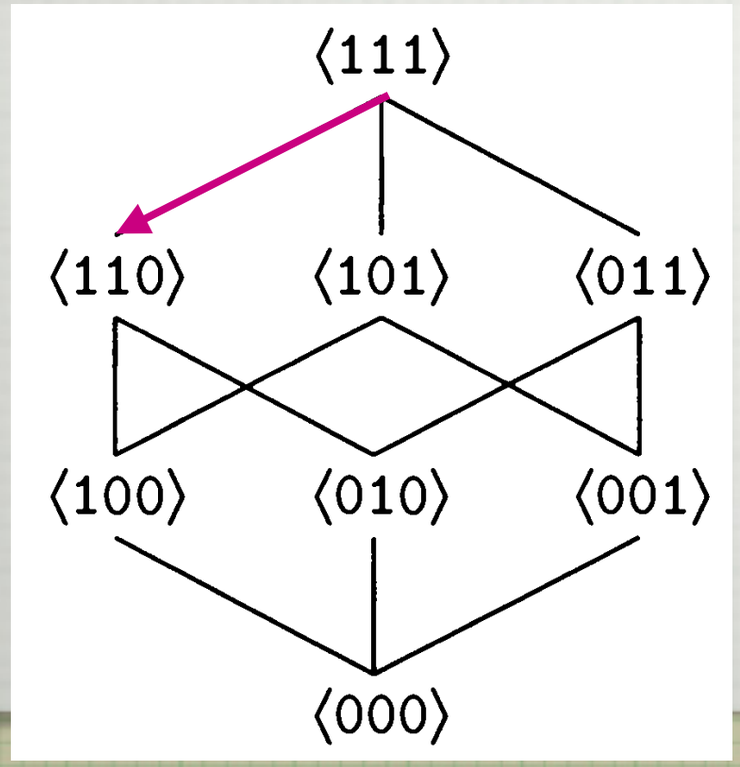


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

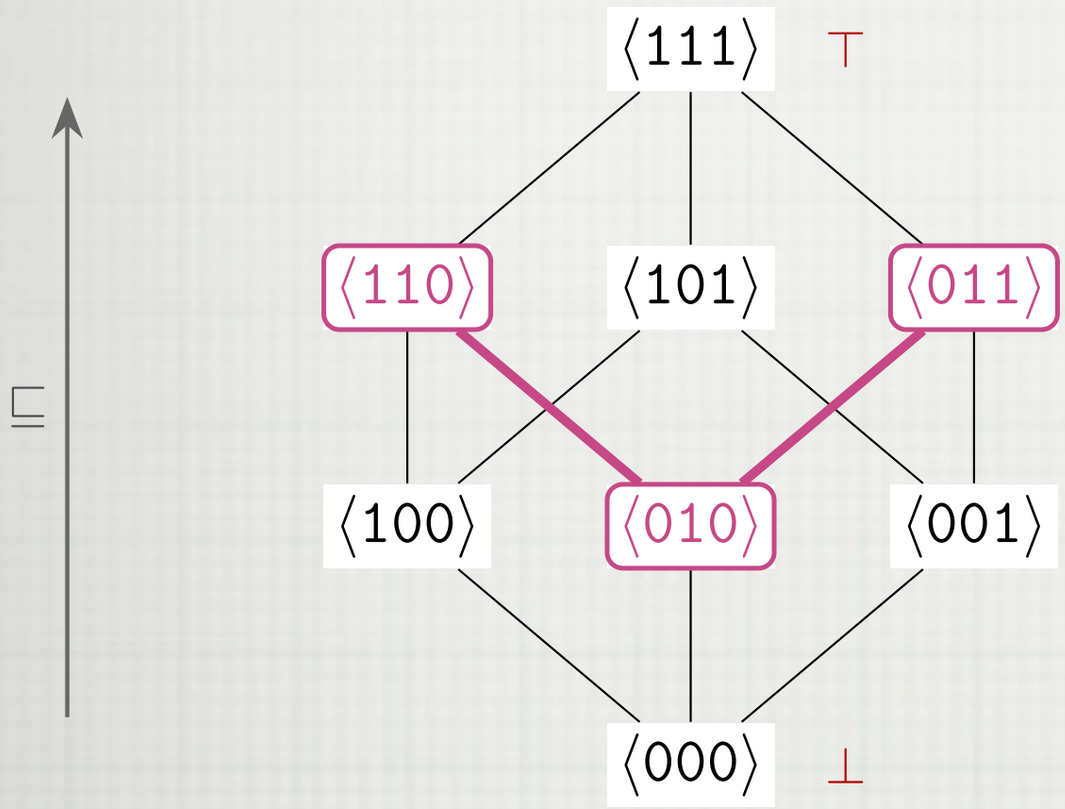
$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

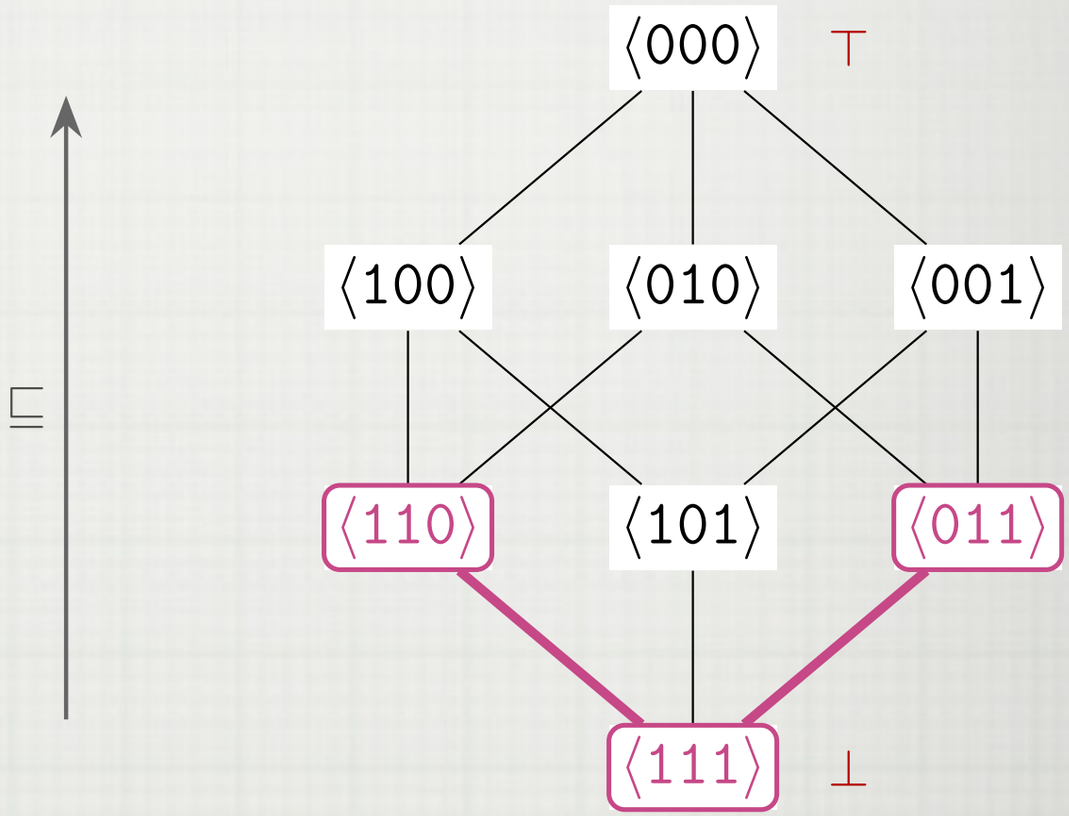


$$\sqcap = \vee$$

$$\langle 110 \rangle \sqcap \langle 011 \rangle = \langle 110 \rangle \vee \langle 011 \rangle = \langle 111 \rangle$$



$\sqcap = \wedge$



$\sqcap = \vee$

Theorem 1

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

$\sqsubseteq$ is provably a partial order relation.

(proof on the board)

Theorem 2

$\sqcap$ is provably the greatest lower bound defined based on $\sqsubseteq$.

(proof on the board)