# *Pointer Analysis: Details*

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

Dec 2017

# Outline

- Pointer Statements Next Topic
- Comparing Points-to and Alias information
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis
- Handling Heap



# Pointer Statements

| Pointer assignments | | Use pointers in expressions |
|---|---|---|
| Addr | $x = \&y$ | *Use x* |
| Copy | $x = y$ | |
| Load | $x = *y$ | |
| | $x = y \rightarrow n$ | |
| Store | $*x = y$ | |
| | $x \rightarrow n = y$ | |

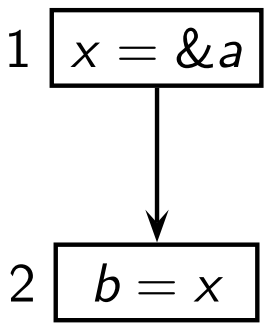
- Field accesses such as $x.n$ are treated as new compile time names
- Containment of $x.n$ within $x$ is recorded in terms of offsets
- Heap will be introduced later

# Outline

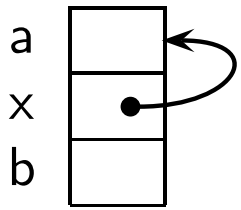
- Pointer Statements
- Comparing Points-to and Alias information Next Topic
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis
- Handling Heap

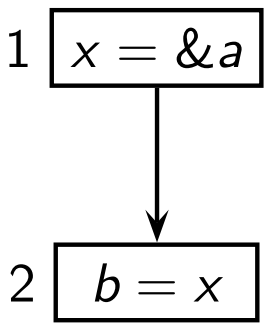


# Alias Information Vs. Points-to Information



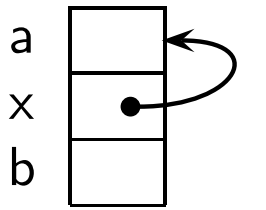
# Alias Information Vs. Points-to Information

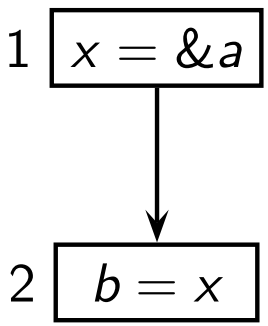


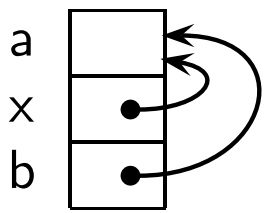

"*x Points-to a*"
denoted $x \rightarrowtail a$

# Alias Information Vs. Points-to Information



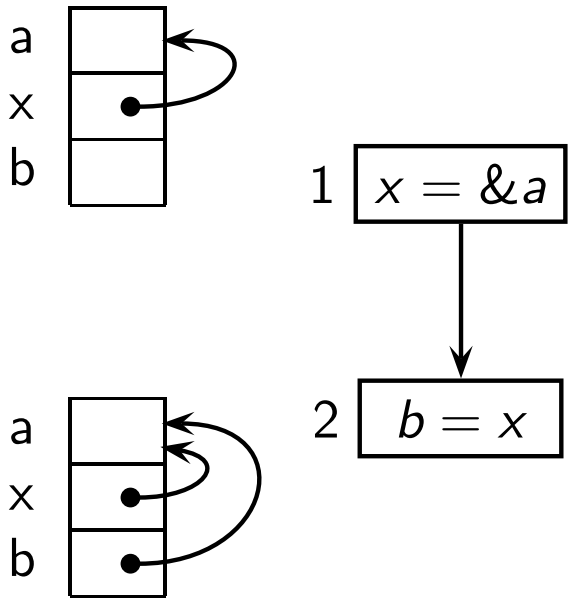




"*x Points-to a*"
denoted $x \rightarrowtail a$




"*x* and *b* are *Aliases*"
denoted $x \doteq b$

# Alias Information Vs. Points-to Information

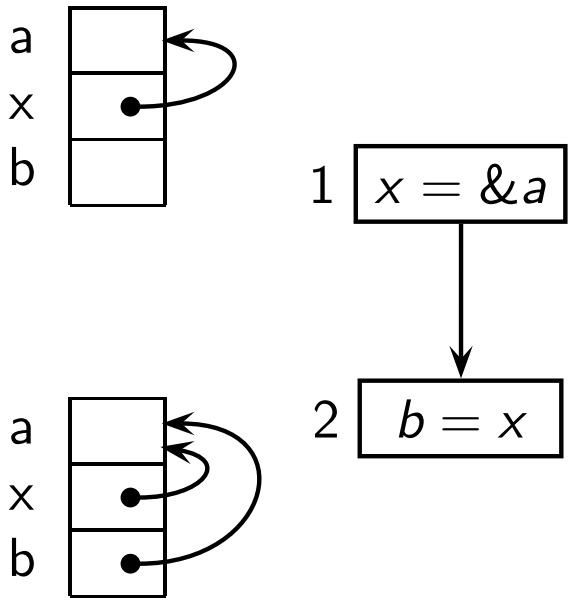



"*x Points-to a*"
denoted $x \rightarrowtail a$

"*x* and *b* are *Aliases*"
denoted $x \doteq b$

Symmetric and Reflexive

# Alias Information Vs. Points-to Information



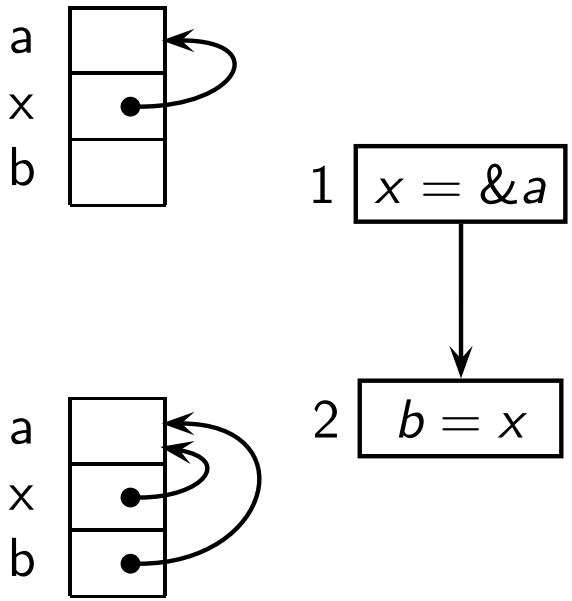

"$x$ *Points-to* $a$"
denoted $x \rightarrowtail a$

Neither Symmetric Nor Reflexive

"$x$ and $b$ are *Aliases*"
denoted $x \stackrel{\circ}{=} b$

Symmetric and Reflexive

# Alias Information Vs. Points-to Information




"*x Points-to a*"
denoted $x \rightarrowtail a$

Neither Symmetric Nor Reflexive

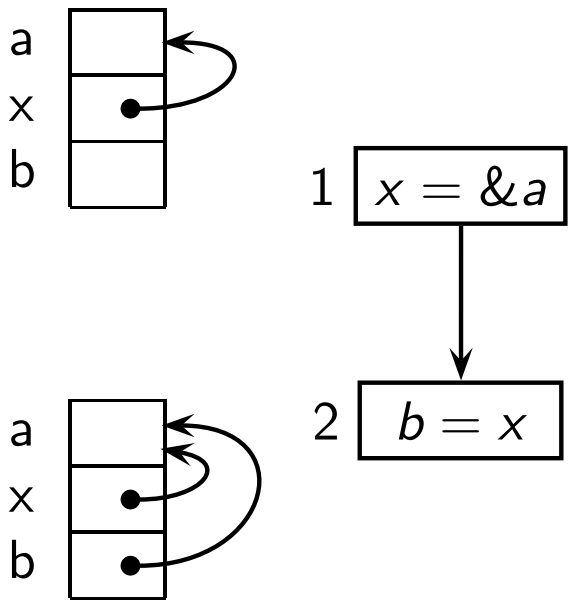
"*x* and *b* are *Aliases*"
denoted $x \doteq b$

Symmetric and Reflexive

- What about transitivity?



# Alias Information Vs. Points-to Information




"$x$ *Points-to* $a$"
denoted $x \rightarrowtail a$

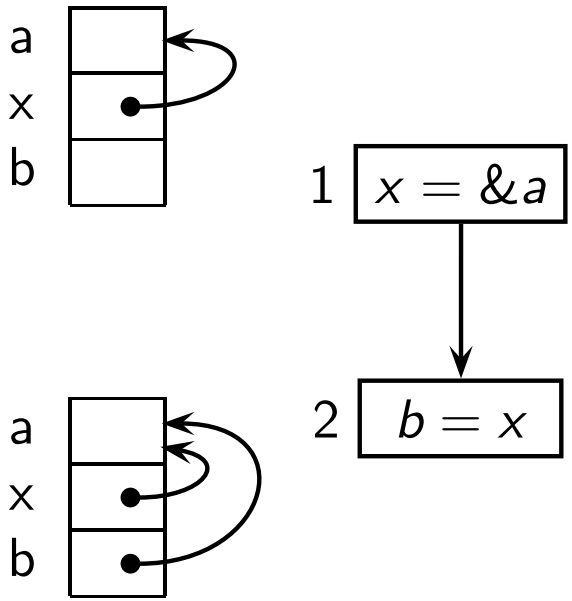
Neither Symmetric Nor Reflexive

"$x$ and $b$ are *Aliases*"
denoted $x \stackrel{\circ}{=} b$

Symmetric and Reflexive

- What about transitivity?
  - Points-to: No

# Alias Information Vs. Points-to Information




"$x$ *Points-to* $a$"
denoted $x \rightarrowtail a$

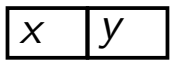
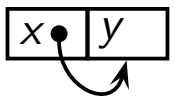
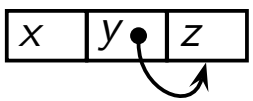
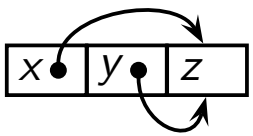
Neither Symmetric Nor Reflexive

"$x$ and $b$ are *Aliases*"
denoted $x \stackrel{\circ}{=} b$

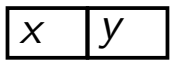
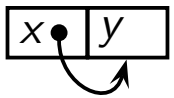
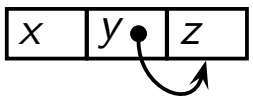
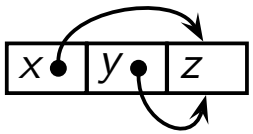
Symmetric and Reflexive

- What about transitivity?
  - Points-to: No
  - Alias: Depends (we will see later)

# Comparing Points-to and Alias Relations (1)

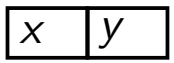
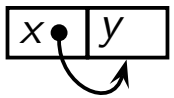
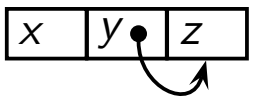
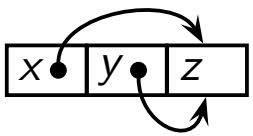
| Statement | Memory | Points-to | | Aliases | |
|---|---|---|---|---|---|
| $x = \&y$ | Before (assume)   | Existing | | Existing | |
| | After   | New | $x \rightarrowtail y$ | New Direct | $x \stackrel{\circ}{=} \&y$ |
| $x = y$ | Before (assume)   | Existing | $y \rightarrowtail z$ | Existing | $y \stackrel{\circ}{=} \&z$ |
| | After   | New | $x \rightarrowtail z$ | New Direct | $x \stackrel{\circ}{=} y$ |
| | | | | New Indirect | $x \stackrel{\circ}{=} \&z$ |

# Comparing Points-to and Alias Relations (1)

| Statement | Memory | Points-to | | Aliases | |
|---|---|---|---|---|---|
| $x = \&y$ | Before (assume)   | Existing | | Existing | |
| | After   | New | $x \rightarrowtail y$ | New Direct | $x \stackrel{\circ}{=} \&y$ |
| $x = y$ | Before (assume)   | Existing | $y \rightarrowtail z$ | Existing | $y \stackrel{\circ}{=} \&z$ |
| | After   | New | $x \rightarrowtail z$ | New Direct | $x \stackrel{\circ}{=} y$ |
| | | | | New Indirect | $x \stackrel{\circ}{=} \&z$ |

- Indirect aliases. Substitute a name by its aliases for transitivity

# Comparing Points-to and Alias Relations (1)

| Statement | Memory | Points-to | | Aliases | |
|---|---|---|---|---|---|
| $x = \&y$ | Before (assume)   | Existing | | Existing | |
| | After   | New | $x \rightarrowtail y$ | New Direct | $x \stackrel{\circ}{=} \&y$ |
| $x = y$ | Before (assume)   | Existing | $y \rightarrowtail z$ | Existing | $y \stackrel{\circ}{=} \&z$ |
| | After   | New | $x \rightarrowtail z$ | New Direct | $x \stackrel{\circ}{=} y$ |
| | | | | New Indirect | $x \stackrel{\circ}{=} \&z$ |

- Indirect aliases. Substitute a name by its aliases for transitivity
- Derived aliases. Apply indirection operator to aliases (ignored here)

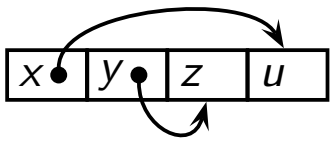
  $x \stackrel{\circ}{=} y \Rightarrow *x \stackrel{\circ}{=} *y$

# Comparing Points-to and Alias Relations (2)

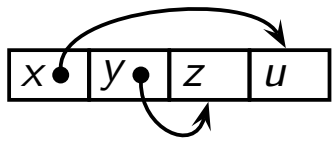
| Statement | Memory | Points-to | Aliases |
|---|---|---|---|
| $*x = y$ | | | |
| $x = *y$ | | | |
| | | | |

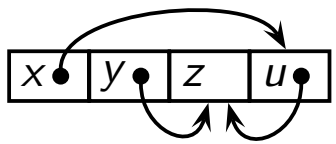


# Comparing Points-to and Alias Relations (2)

| Statement | Memory | Points-to | Aliases |
|---|---|---|---|
| $*x = y$ | Before (assume)   | Existing: $x \rightarrowtail u$, $y \rightarrowtail z$ | Existing: $x \stackrel{\circ}{=} \&u$, $y \stackrel{\circ}{=} \&z$ |
| $x = *y$ | | | |

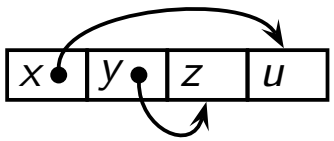
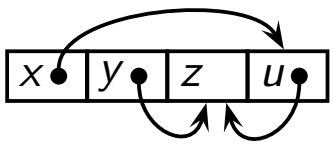
# Comparing Points-to and Alias Relations (2)

<table>
<tr><th>Statement</th><th>Memory</th><th>Points-to</th><th colspan="2">Aliases</th></tr>
<tr><td rowspan="2">$*x = y$</td><td rowspan="2">Before (assume)<br> 

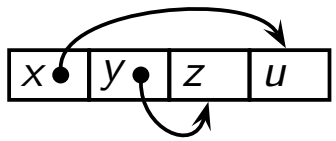
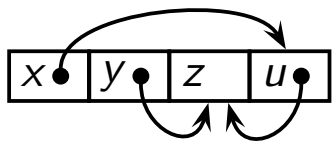
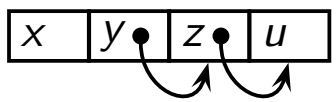
<br>After<br> 
</td><td rowspan="2"><table><tr><td>Existing</td><td>$x \rightarrowtail u$<br>$y \rightarrowtail z$</td></tr><tr><td>New</td><td>$u \rightarrowtail z$</td></tr></table></td><td>Existing</td><td>$x \stackrel{\circ}{=} \&u$<br>$y \stackrel{\circ}{=} \&z$</td></tr>
<tr><td>New Direct</td><td>$*x \stackrel{\circ}{=} y$</td></tr>
<tr><td>$x = *y$</td><td></td><td></td><td colspan="2"></td></tr>
<tr><td colspan="5"></td></tr>
</table>



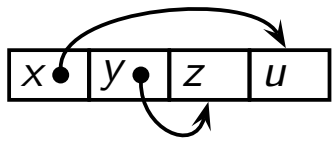
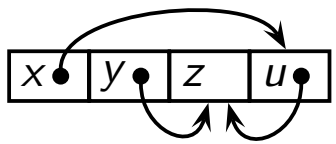
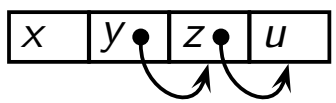
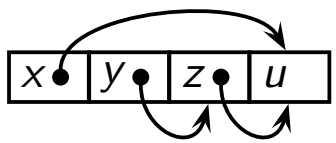
# Comparing Points-to and Alias Relations (2)

<table>
<tr><th>Statement</th><th>Memory</th><th>Points-to</th><th colspan="2">Aliases</th></tr>
<tr><td rowspan="3">$*x = y$</td><td rowspan="3">Before (assume)<br> 
<br>After<br> 
</td><td rowspan="3"><table><tr><td>Existing</td><td>$x \rightarrowtail u$<br>$y \rightarrowtail z$</td></tr><tr><td>New</td><td>$u \rightarrowtail z$</td></tr></table></td><td>Existing</td><td>$x \stackrel{\circ}{=} \&u$<br>$y \stackrel{\circ}{=} \&z$</td></tr>
<tr><td>New Direct</td><td>$*x \stackrel{\circ}{=} y$</td></tr>
<tr><td>New Indirect</td><td>$u \stackrel{\circ}{=} \&z$<br>$y \stackrel{\circ}{=} u$<br>$*x \stackrel{\circ}{=} \&z$</td></tr>
<tr><td>$x = *y$</td><td></td><td></td><td colspan="2"></td></tr>
<tr><td colspan="5"></td></tr>
</table>

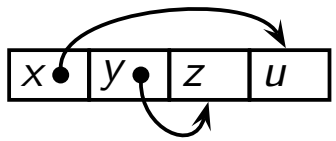
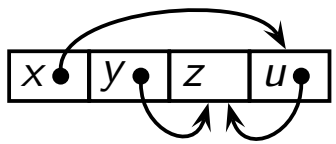
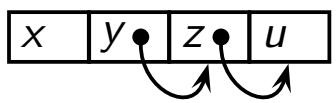
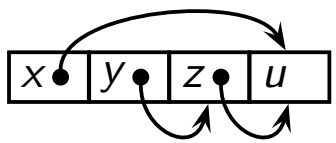
# Comparing Points-to and Alias Relations (2)

<table>
<tr><th>Statement</th><th>Memory</th><th>Points-to</th><th colspan="2">Aliases</th></tr>
<tr><td rowspan="2">$*x = y$</td><td rowspan="2">Before (assume)<br>  <br>After<br>  </td><td rowspan="2"><table><tr><td>Existing</td><td>$x \rightarrowtail u$<br>$y \rightarrowtail z$</td></tr><tr><td>New</td><td>$u \rightarrowtail z$</td></tr></table></td><td>Existing</td><td>$x \doteq \&u$<br>$y \doteq \&z$</td></tr>
<tr><td>New Direct</td><td>$*x \doteq y$</td></tr>
<tr><td rowspan="2">$x = *y$</td><td rowspan="2">Before (assume)<br>  </td><td rowspan="2"><table><tr><td>Existing</td><td>$y \rightarrowtail z$<br>$z \rightarrowtail u$</td></tr></table></td><td>New Indirect</td><td>$u \doteq \&z$<br>$y \doteq u$<br>$*x \doteq \&z$</td></tr>
<tr><td colspan="2"><table><tr><td>Existing</td><td>$y \doteq \&z$<br>$z \doteq \&u$<br>$*y \doteq \&u$</td></tr></table></td></tr>
</table>

# Comparing Points-to and Alias Relations (2)

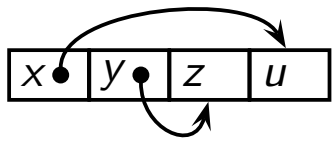
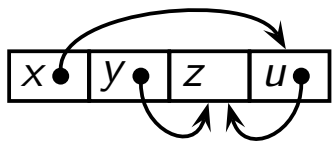
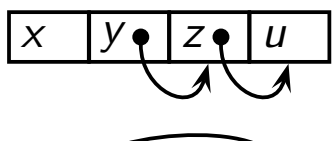
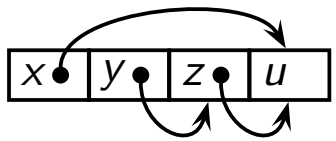
| Statement | Memory | Points-to | Aliases | |
|---|---|---|---|---|
| $*x = y$ | Before (assume)   After   | Existing: $x \rightarrowtail u$, $y \rightarrowtail z$<br>New: $u \rightarrowtail z$ | Existing | $x \doteq \&u$<br>$y \doteq \&z$ |
| | | | New Direct | $*x \doteq y$ |
| | | | New Indirect | $u \doteq \&z$<br>$y \doteq u$<br>$*x \doteq \&z$ |
| $x = *y$ | Before (assume)   After   | Existing: $y \rightarrowtail z$, $z \rightarrowtail u$<br>New: $x \rightarrowtail u$ | Existing | $y \doteq \&z$<br>$z \doteq \&u$<br>$*y \doteq \&u$ |
| | | | New Direct | $x \doteq *y$ |

# Comparing Points-to and Alias Relations (2)

| Statement | Memory | Points-to | Aliases | |
|---|---|---|---|---|
| $*x = y$ | Before (assume)   After   | Existing: $x \rightarrowtail u$, $y \rightarrowtail z$<br>New: $u \rightarrowtail z$ | Existing | $x \doteq \&u$<br>$y \doteq \&z$ |
| | | | New Direct | $*x \doteq y$ |
| | | | New Indirect | $u \doteq \&z$<br>$y \doteq u$<br>$*x \doteq \&z$ |
| $x = *y$ | Before (assume)   After   | Existing: $y \rightarrowtail z$, $z \rightarrowtail u$<br>New: $x \rightarrowtail u$ | Existing | $y \doteq \&z$<br>$z \doteq \&u$<br>$*y \doteq \&u$ |
| | | | New Direct | $x \doteq *y$ |
| | | | New Indirect | $x \doteq \&u$<br>$x \doteq z$ |



# Comparing Points-to and Alias Relations (2)

<table>
<tr><th>Statement</th><th>Memory</th><th>Points-to</th><th colspan="2">Aliases</th></tr>
<tr><td rowspan="4">$*x = y$</td><td rowspan="4">Before (assume)<br>  <br>After<br>  </td><td rowspan="2">Existing</td><td rowspan="2">$x \mapsto u$<br>$y \mapsto z$</td><td>Existing</td><td>$x \doteq \&u$<br>$y \doteq \&z$</td></tr>
<tr><td>New Direct</td><td>$*x \doteq y$</td></tr>
<tr><td>New</td><td>$u \mapsto z$</td><td rowspan="2">New Indirect</td><td rowspan="2">$u \doteq \&z$<br>$y \doteq u$<br>$*x \doteq \&z$</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td rowspan="4">$x = *y$</td><td rowspan="4">Before (assume)<br>  <br>After<br>  </td><td rowspan="2">Existing</td><td rowspan="2">$y \mapsto z$<br>$z \mapsto u$</td><td rowspan="2">Existing</td><td rowspan="2">$y \doteq \&z$<br>$z \doteq \&u$<br>$*y \doteq \&u$</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>New</td><td>$x \mapsto u$</td><td>New Direct</td><td>$x \doteq *y$</td></tr>
<tr><td colspan="2"></td><td>New Indirect</td><td>$x \doteq \&u$<br>$x \doteq z$</td></tr>
<tr><td colspan="5">The resulting memories look similar but are different. In the first case we have $u \mapsto z$ whereas in the second case the arrow direction is opposite (i.e. $z \mapsto u$).</td></tr>
</table>

# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph

- Alias information records paths in the memory graph



# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \stackrel{\circ}{=} \&y$
    $x$ holds the address of $y$

- Alias information records paths in the memory graph
  - paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)



# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \stackrel{\circ}{=} \&y$
    $x$ holds the address of $y$
  - other aliases can be discovered by composing edges

- Alias information records paths in the memory graph
  - paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)



# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \stackrel{\circ}{=} \&y$
    $x$ holds the address of $y$
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

- Alias information records paths in the memory graph
  - paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too



# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \stackrel{\circ}{=} \&y$
    $x$ holds the address of $y$
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

- Alias information records paths in the memory graph
  - paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too
  - if we have $x \stackrel{\circ}{=} y$ then $*x \stackrel{\circ}{=} *y$ is redundant and is not recorded



# Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \stackrel{\circ}{=} \&y$
    $x$ holds the address of $y$
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

  More compact but less general

- Alias information records paths in the memory graph
  - paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too
  - if we have $x \stackrel{\circ}{=} y$ then $*x \stackrel{\circ}{=} *y$ is redundant and is not recorded

  More general and more complex

# An Outline of Pointer Analysis Coverage

- Pointer Statements
- Comparing Points-to and Alias information
- Defining Points-to Analysis Next Topic
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis
- Handling Heap



# Concrete States and Traces for Points-to Analysis

| Notation | Illustration |
|---|---|
| $V$ contains all variables | |
| $P \subseteq V$ contains all pointer variables | |
| $F$ contains all pointer fields in structures (and also "$*$") | |
| Data states $\sigma : V \times F \rightarrow V$ | |
| Traces $\tau$ are sequences of transitions $(n, \sigma) \rightarrow (n', \sigma')$ starting from a given initial $n_0, \sigma_0$ | |

# Concrete States and Traces for Points-to Analysis

| Notation | Illustration |
|---|---|
| $V$ contains all variables | $V = \{a, b, x\}$ |
| $P \subseteq V$ contains all pointer variables | $P = \{ \text{x} \}$ |
| $F$ contains all pointer fields in structures (and also "$*$") | $F = \{ *, \text{f} \}$ |
| Data states $\sigma : V \times F \to V$ | |
| Traces $\tau$ are sequences of transitions $(n, \sigma) \to (n', \sigma')$ starting from a given initial $n_0, \sigma_0$ | |

| Program statement | Corresponding state after each statement |
|---|---|
| 0 : skip | $\sigma_0 = \emptyset$ |
| 1 : $x = \&a$ | $\sigma_1 = \{((x, *) \mapsto a)\}$ |
| 2 : $x \to f = \&b$ | $\sigma_2 = \{((x, *) \mapsto a), ((a, f) \mapsto b)\}$ |

Trace $\tau_1 = (0, \sigma_0) \to (1, \sigma_1) \to (2, \sigma_2)$

# Ideal Points-to Analysis

For a given statement $n$

- Reachable States are the states reaching the statement along all traces
- Ideal May-Points-to analysis computes Points-to information reaching along all traces
- Ideal Must-Points-to analysis computes Points-to information that is common to all traces

$$RS(n) = \{\sigma \mid (n, \sigma) \text{ occurs in some trace } \tau\}$$

$$\text{IdealMayPT}(n) = \bigcup_{\sigma \in RS(n)} \sigma$$

$$\text{IdealMustPT}(n) = \bigcap_{\sigma \in RS(n)} \sigma$$

# Soundness and Precision of Flow-Sensitive May-Points-to Analysis

A flow-sensitive points-to analysis algorithm $A$ computes $S : N \rightarrow V \times F \times V$

- A flow-sensitive points-to analysis algorithm $A$ is sound if

  $$\forall n : \ S(n) \supseteq \text{IdealMayPT}(n)$$

- A flow-sensitive points-to analysis algorithm $A$ is precise if

  $$\forall n : \ S(n) = \text{IdealMayPT}(n)$$

- A flow-sensitive points-to analysis algorithm $A_1$ is more precise than $A_2$ if

  $$\forall n : \ S_2(n) \supseteq S_1(n) \supseteq \text{IdealMayPT}(n)$$

  (Precision is meaningful only for a sound analysis)



# Soundness and Precision of Flow-Insensitive May-Points-to Analysis

A flow-insensitive points-to analysis algorithm $A$ computes $S : V \times F \times V$

- A flow-insensitive points-to analysis algorithm $A$ is sound if

$$S \supseteq \bigcup_{n \in N} \text{IdealMayPT}(n)$$

- A flow-insensitive points-to analysis algorithm $A$ is precise if

$$S = \bigcup_{n \in N} \text{IdealMayPT}(n)$$

- A flow-insensitive points-to analysis algorithm $A_1$ is more precise than $A_2$ if

$$S_2 \supseteq S_1 \supseteq \bigcup_{n \in N} \text{IdealMayPT}(n)$$

(Precision is meaningful only for a sound analysis)

# An Outline of Pointer Analysis Coverage

- Pointer Statements
- Comparing Points-to and Alias information
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis Next Topic
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis
- Handling Heap



# Flow Sensitive Vs. Flow Insensitive Pointer Analysis

- Flow insensitive pointer analysis
  - Inclusion based: Andersen's approach
  - Equality based: Steensgaard's approach
- Flow sensitive pointer analysis
  - May points-to analysis
  - Must points-to analysis



# Notation for Andersen's and Steensgaard's Points-to Analysis

- $P_{x.f}$ denotes the set of pointees of pointer variable $x$ along field $f$
  - $P_{x.*}$ (concisely written as $P_x$) denotes the set of pointees of $x$
  - If $x$ is a structure, $P_x$ is the set of pointees of all fields of $x$
- $Unify(x, y)$ unifies locations $x$ and $y$
  - $x$ and $y$ are treated as equivalent locations
  - the pointees of the unified locations are also unified transitively
- $UnifyPTS(x, y)$ unifies the pointees of $x$ and $y$
  - $x$ and $y$ themselves are not unified
- We use $x.f$ if the pointees of field $f$ of $x$ are to be unified



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

For field $f$ of $x$, we replace $x$ by $x.f$

# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to $y$
- Include $y$ in the points-to set of $x$

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to $y$
- Include $y$ in the points-to set of $x$

Steensgaard's view

- Equivalence between: All pointees of $x$
- Unify $y$ and pointees of $x$

# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of $x$

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of $x$

Steensgaard's view

- Equivalence between: Pointees of $x$ and pointees of $y$
- Unify points-to sets of $x$ and $y$

# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to pointees of pointees of $y$
- Include the pointees of pointees of $y$ in the points-to set of $x$

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- $x$ points to pointees of pointees of $y$
- Include the pointees of pointees of $y$ in the points-to set of $x$

Steensgaard's view

- Equivalence between: Pointees of $x$ and pointees of pointees of $y$
- Unify points-to sets of $x$ and pointees of $y$

# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- Pointees of $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of the pointees of $x$

Steensgaard's view



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Andersen's view

- Pointees of $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of the pointees of $x$

Steensgaard's view

- Equivalence between: Pointees of pointees of $x$ and pointees of $y$
- Unify points-to sets of pointees of $x$ and $y$

# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

Inclusion



# Andersen's and Steensgaard's Points-to Analysis

| Statement | Andersen's Points-to Sets | Steensgaard's Points-to Sets |
|---|---|---|
| $x = \&y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$<br>$\forall z \in P_x, Unify(y, z)$ |
| $x = y$ | $P_x \supseteq P_y$ | $UnifyPTS(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z,\ \forall z \in P_y$ | $\forall z \in P_y,\ UnifyPTS(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y,\ \forall z \in P_x$ | $\forall z \in P_x,\ UnifyPTS(y, z)$ |

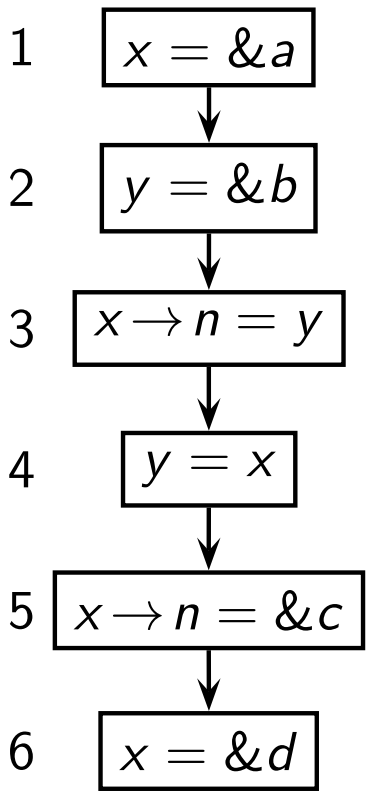
Inclusion

Equality



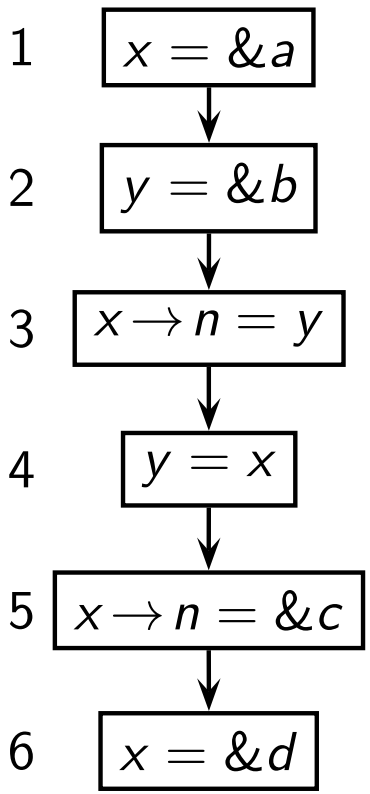
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



# Example of Inclusion Based (aka Andersen's) Points-to Analysis

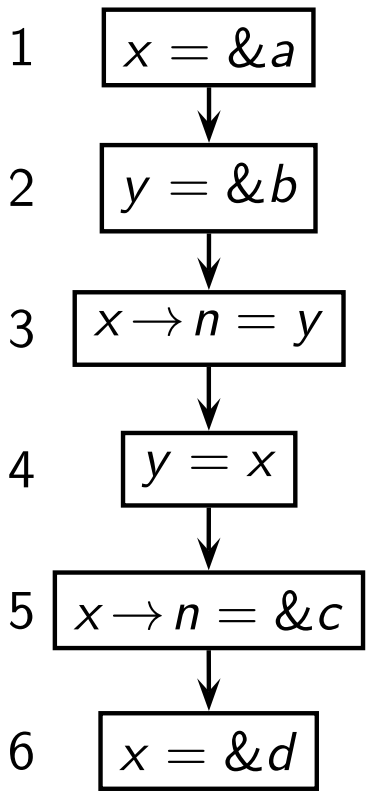
Program




| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

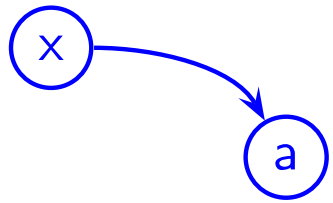
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



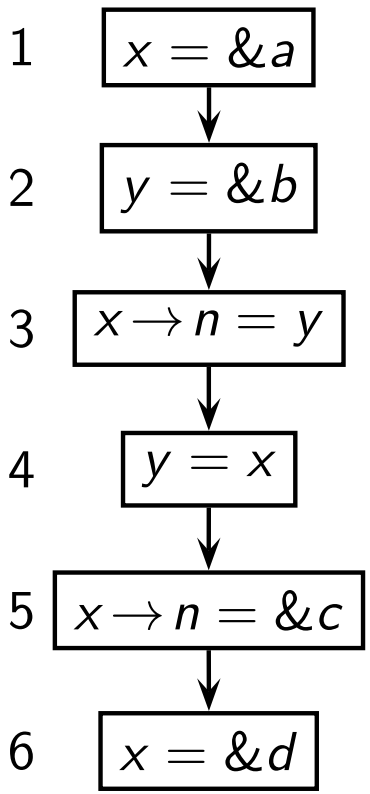

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



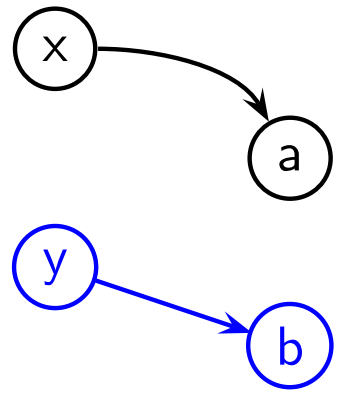
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



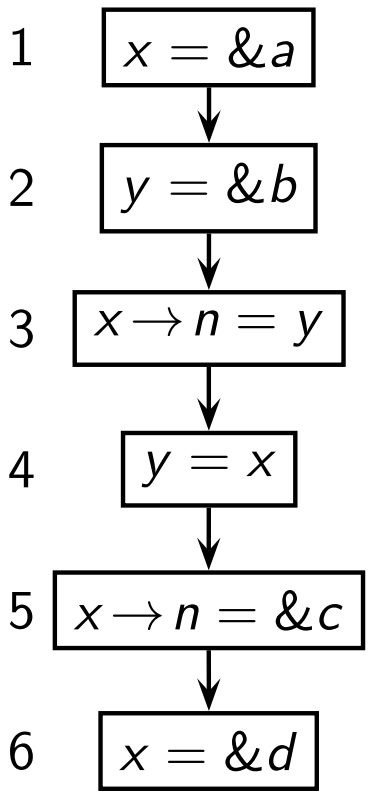

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



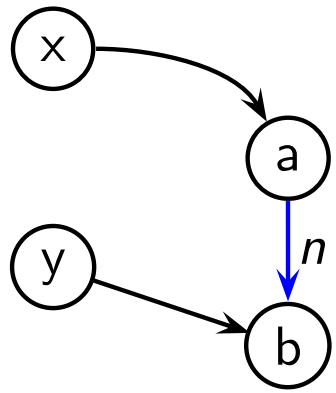
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



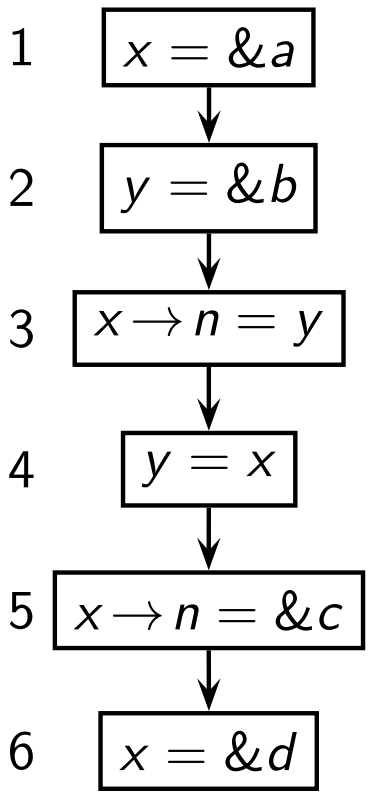

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



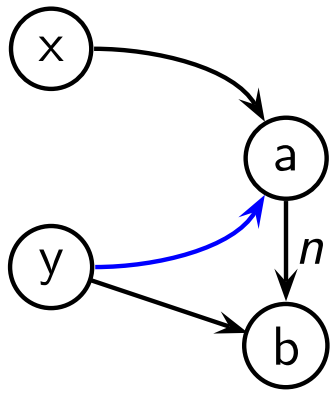
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



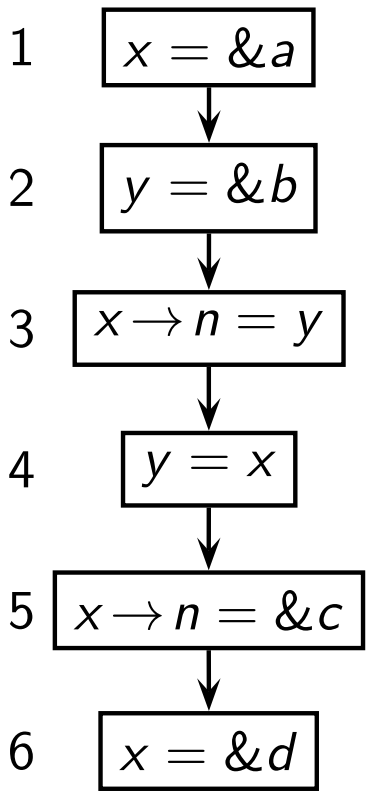

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



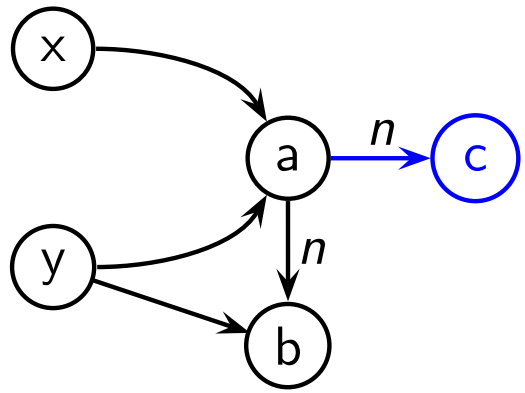
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



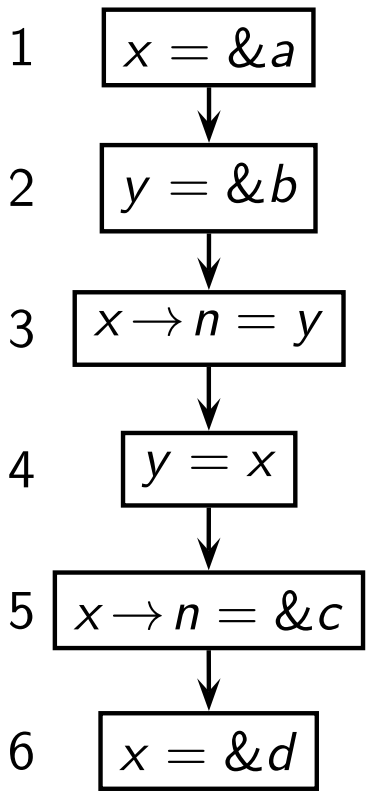

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



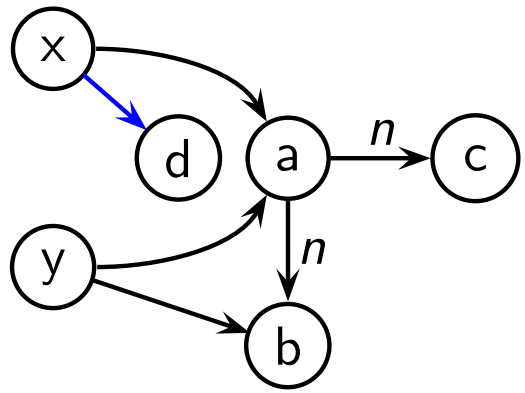
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program



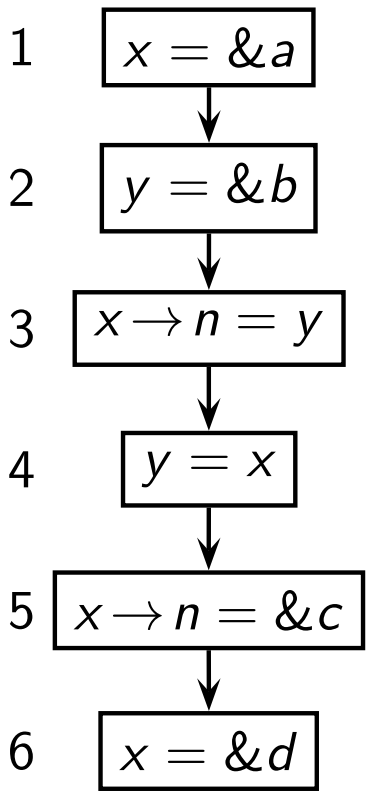

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

Points-to Graph



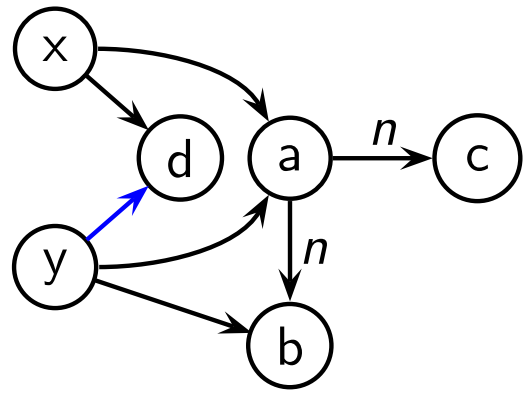
# Example of Inclusion Based (aka Andersen's) Points-to Analysis

Program




| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

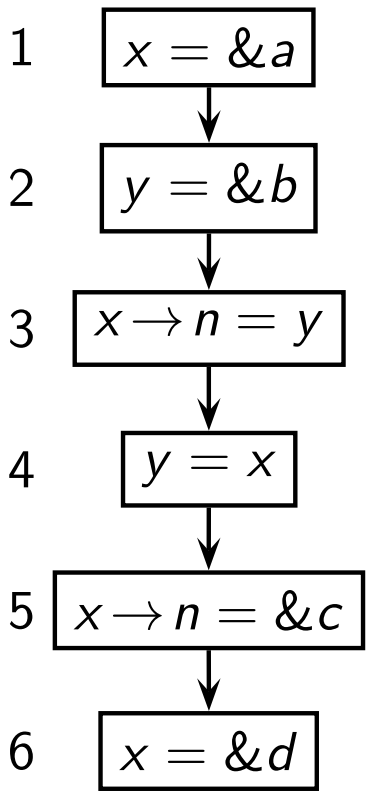
Points-to Graph




- Since $P_x$ has changed, $P_y$ needs to be processed again
- Order of processing the sets influences efficiency significantly
- A plethora of heuristics have been proposed

# Example of Inclusion Based (aka Andersen's) Points-to Analysis

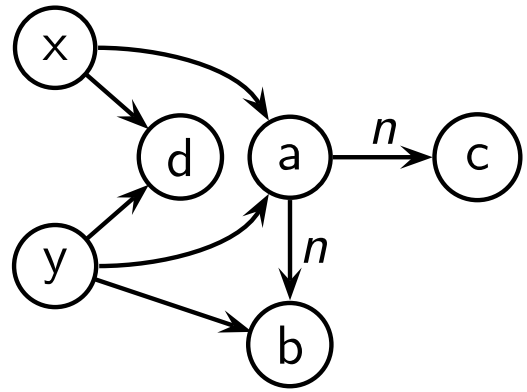
Program

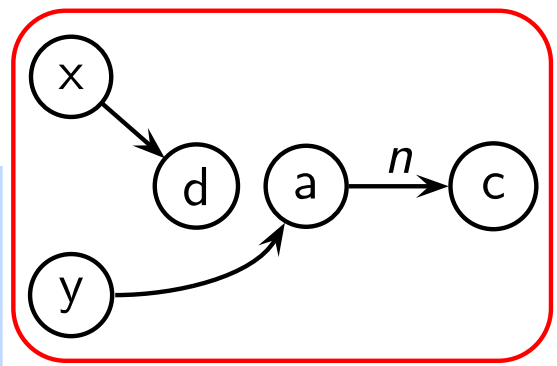



| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$ |
| 2 | $P_y \supseteq \{b\}$ |
| 3 | $\forall z \in P_x, P_{z.n} \supseteq P_y$ |
| 4 | $P_y \supseteq P_x$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$ |
| 6 | $P_x \supseteq \{d\}$ |

- Actual graph after statement 6 is much simpler with many edges killed
- $y$ does not point to $d$ any time in the execution

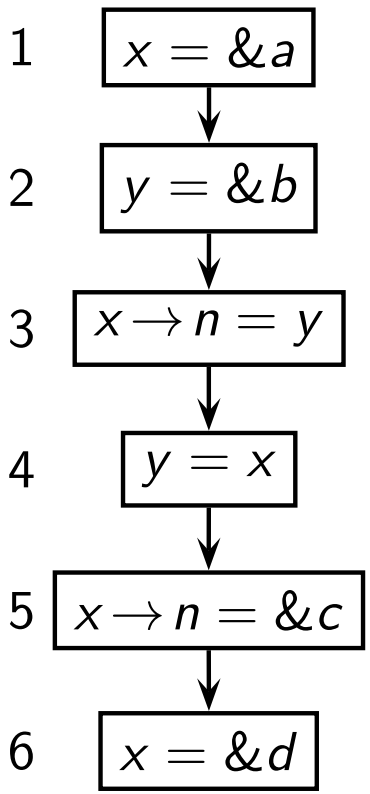
Points-to Graph



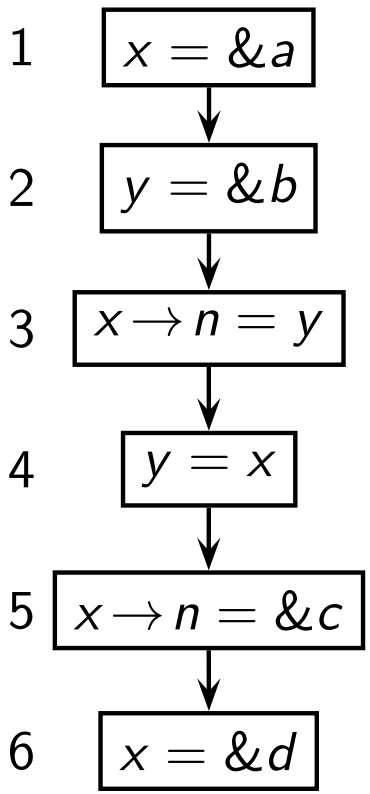
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



# Example of Equality Based (aka Steensgaard's) Points-to Analysis

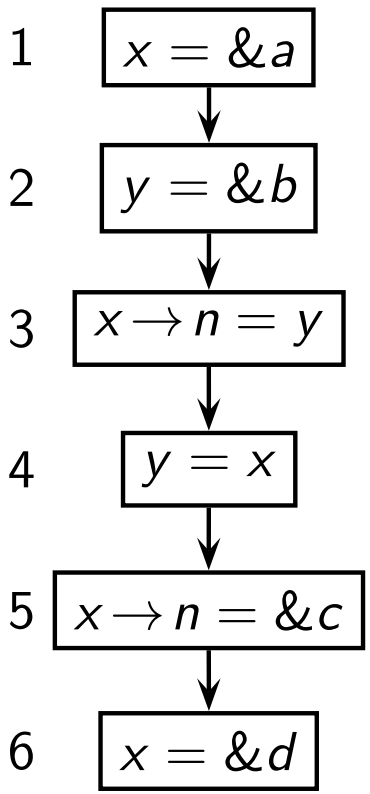
Program




| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

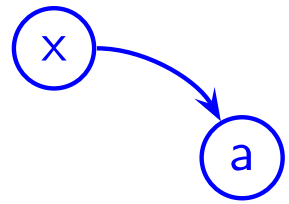
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



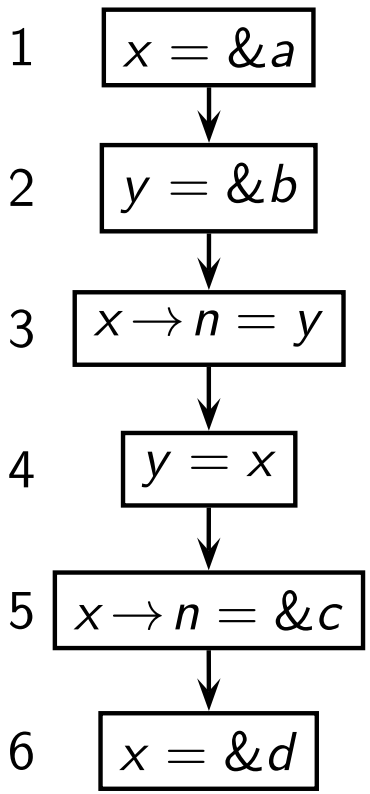

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



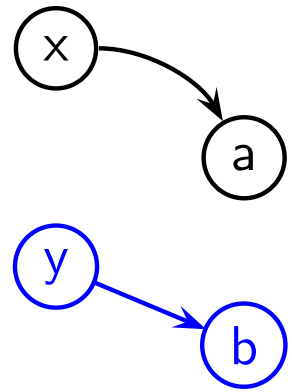
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



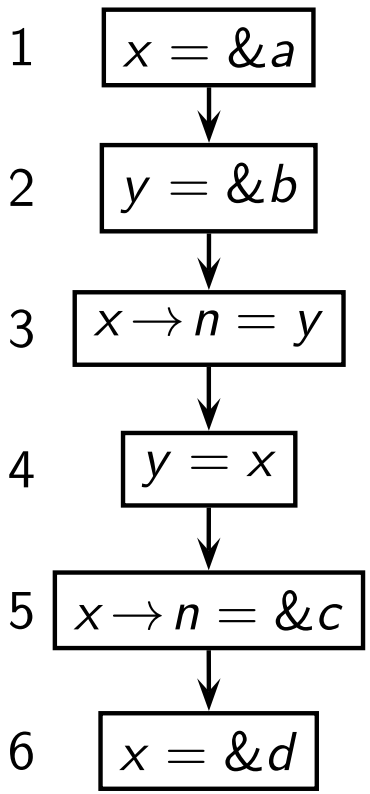

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



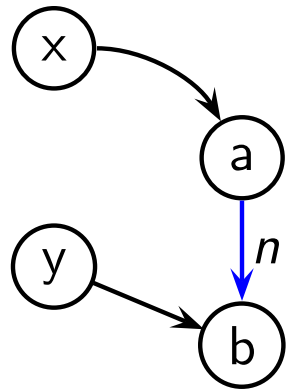
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



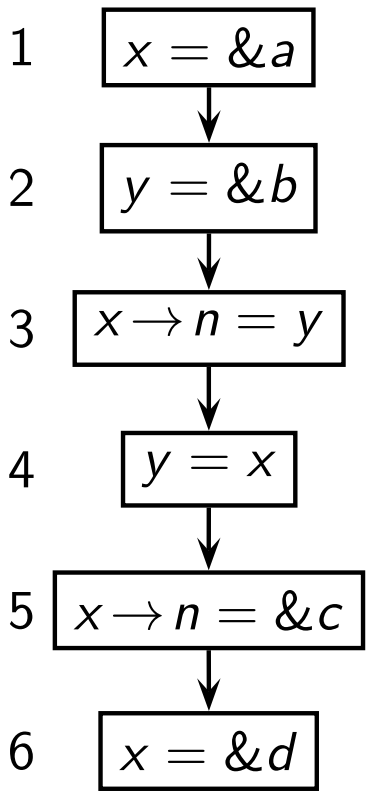

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



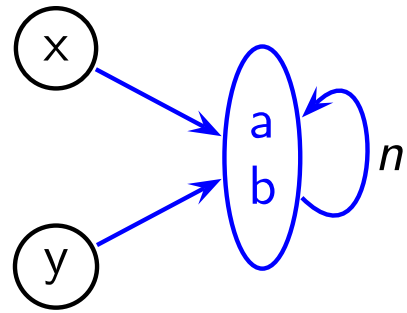
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



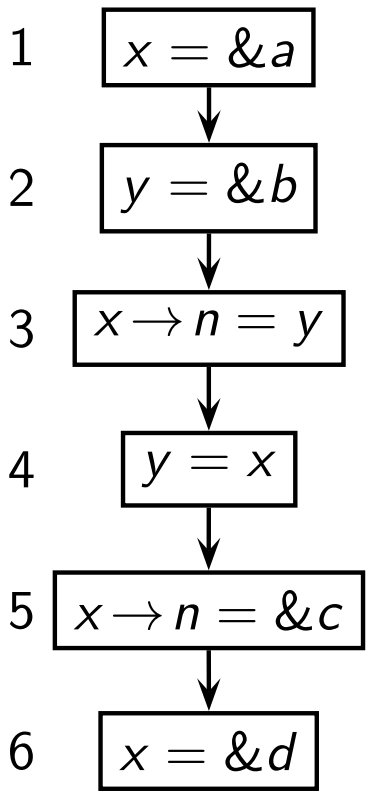

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



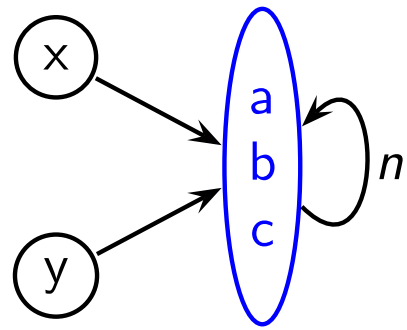
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



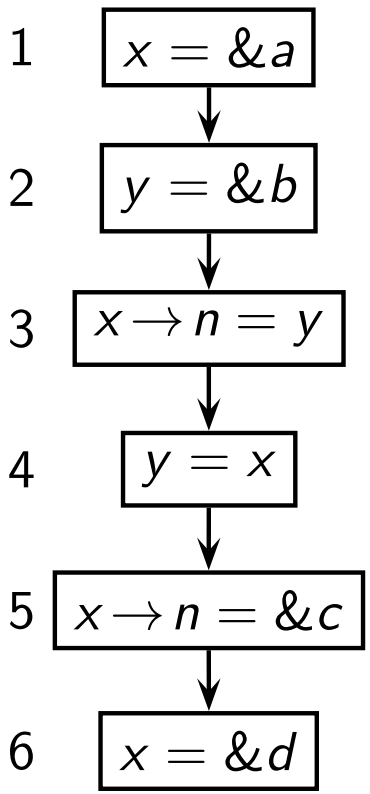

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



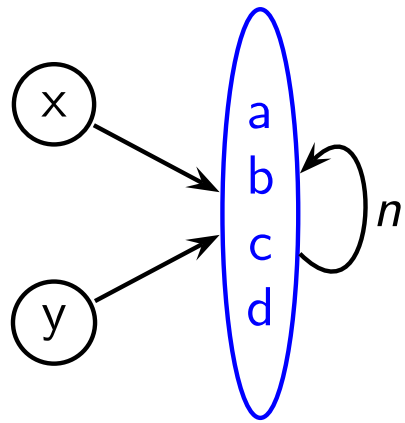
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program



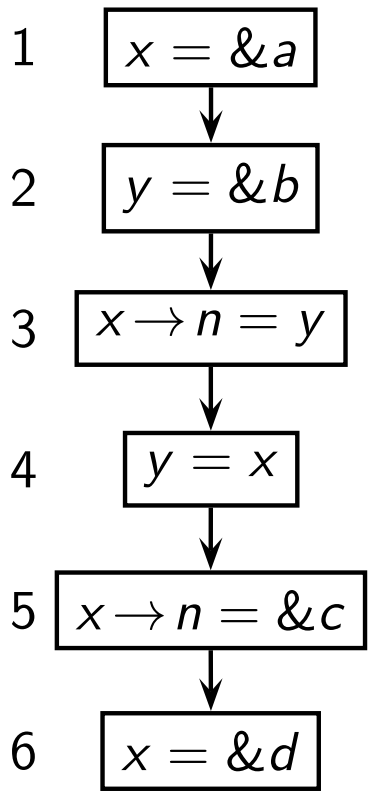

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Points-to Graph



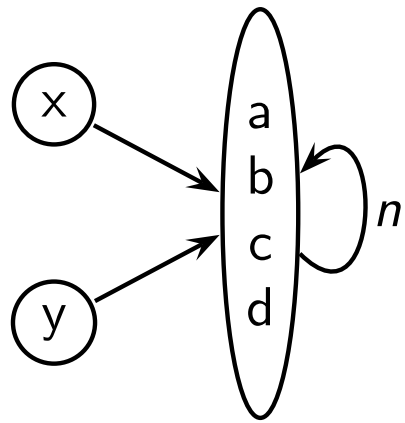
# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program




| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

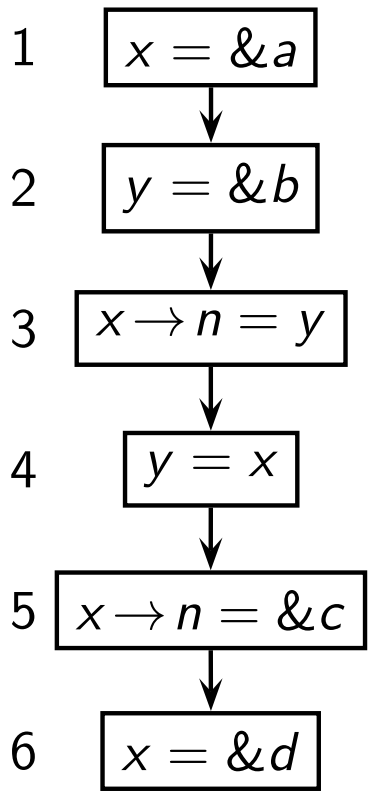
Points-to Graph




No further change

# Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program

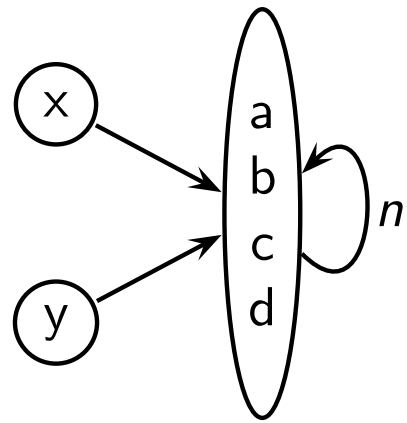


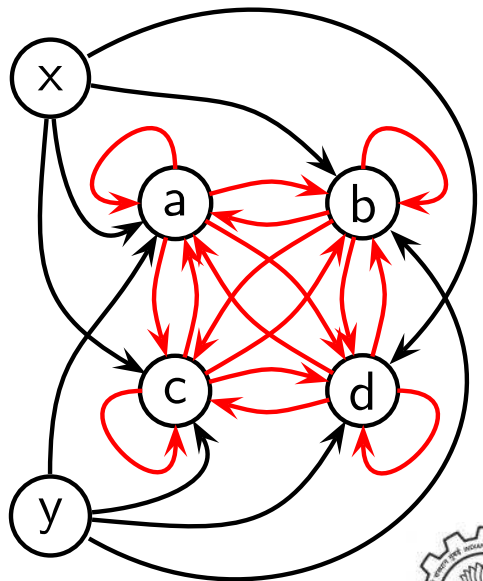

| Node | Constraint |
|---|---|
| 1 | $P_x \supseteq \{a\}$<br>$\forall z \in P_x, Unify(a, z)$ |
| 2 | $P_y \supseteq \{b\}$<br>$\forall z \in P_y, Unify(b, z)$ |
| 3 | $\forall z \in P_x, UnifyPTS(y, z.n)$ |
| 4 | $UnifyPTS(x, y)$ |
| 5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$<br>$\forall w \in P_{z.n}, Unify(w, c)$ |
| 6 | $P_x \supseteq \{d\}$<br>$\forall z \in P_x, Unify(d, z)$ |

Red edges represent field $n$ in the the full blown up graph. It has far more edges than in Andersen's graph

Far more efficient but far less precise

Points-to Graph



# Comparing Equality and Inclusion Based Analyses

- Andersen's algorithm is cubic in number of pointers
- Steensgaard's algorithm is nearly linear in number of pointers
- How can it be more efficient by an orders of magnitude?
  - Andersen's inclusion based wisdom:
    Add edges and let the number of successors increase
  - Steensgaard's equality based wisdom:
    Merge multiple successors and maintain a single successor of any node



# Comparing Equality and Inclusion Based Analyses

- Andersen's algorithm is cubic in number of pointers
- Steensgaard's algorithm is nearly linear in number of pointers
- How can it be more efficient by an orders of magnitude?
  - Andersen's inclusion based wisdom:
    Add edges and let the number of successors increase
  - Steensgaard's equality based wisdom:
    Merge multiple successors and maintain a single successor of any node
- Since a larger number of pointers treated are alike and fewer distinctions are maintained, Steensgaard's points-to graphs (with unified nodes) are much smaller



# Comparing Equality and Inclusion Based Analyses

- Andersen's algorithm is cubic in number of pointers
- Steensgaard's algorithm is nearly linear in number of pointers
- How can it be more efficient by an orders of magnitude?
  - Andersen's inclusion based wisdom:
    Add edges and let the number of successors increase
  - Steensgaard's equality based wisdom:
    Merge multiple successors and maintain a single successor of any node
- Since a larger number of pointers treated are alike and fewer distinctions are maintained, Steensgaard's points-to graphs (with unified nodes) are much smaller
- Efficient *Union-Find* algorithms to merge intersecting subsets

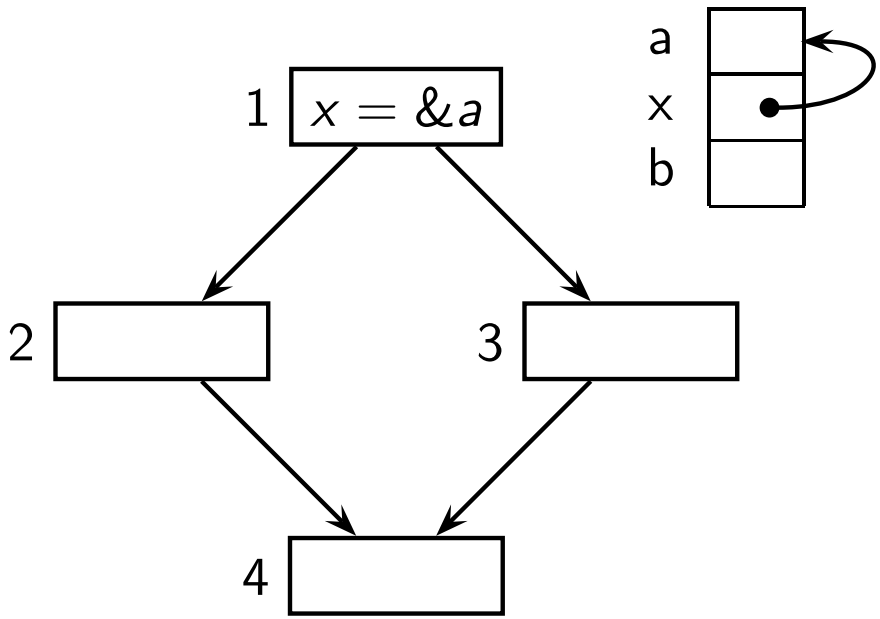


# An Outline of Pointer Analysis Coverage

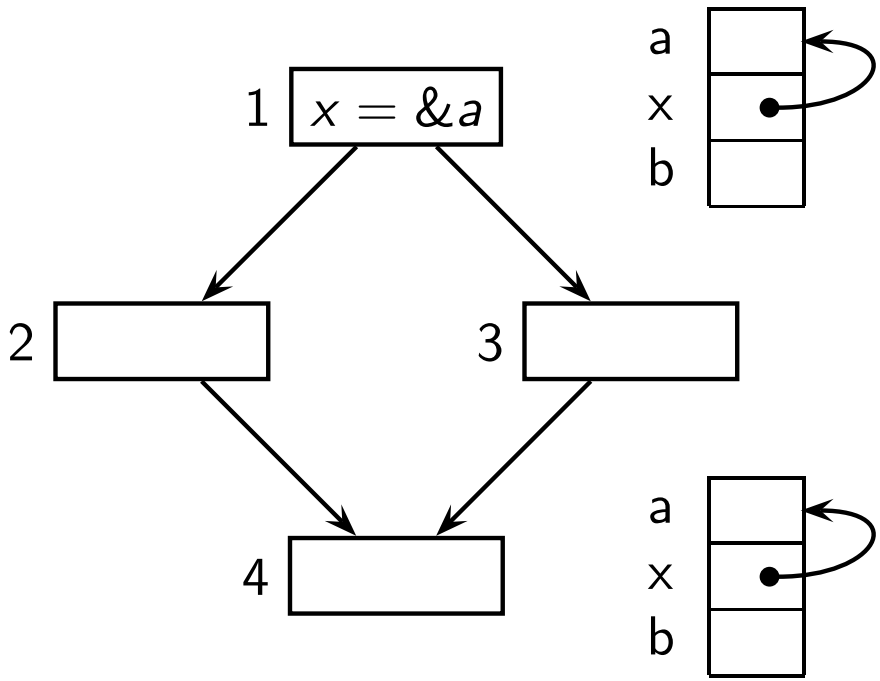
- Pointer Statements
- Comparing Points-to and Alias information
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis Next Topic
- Liveness Based Points-to Analysis
- Handling Heap



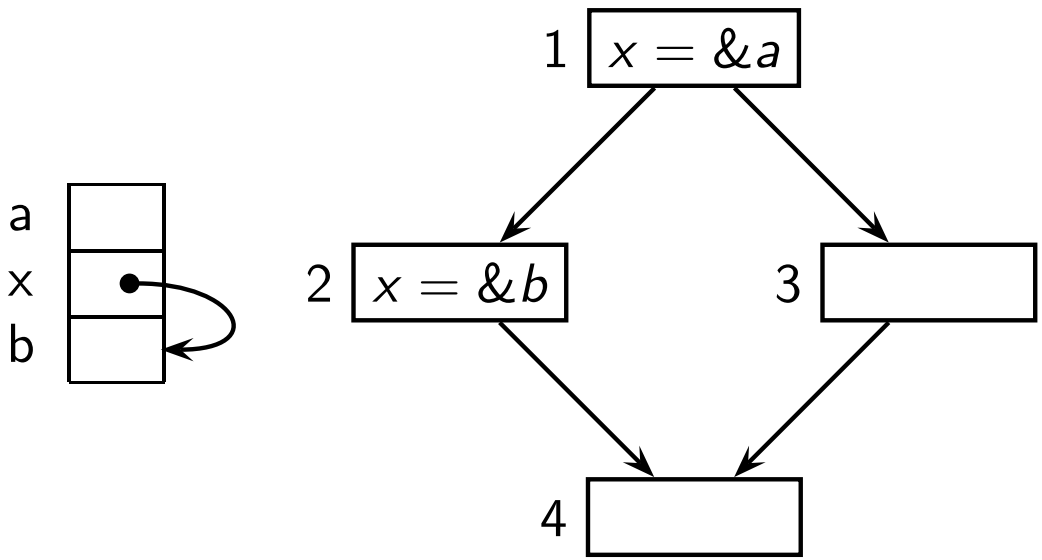
# Must Points-to Information



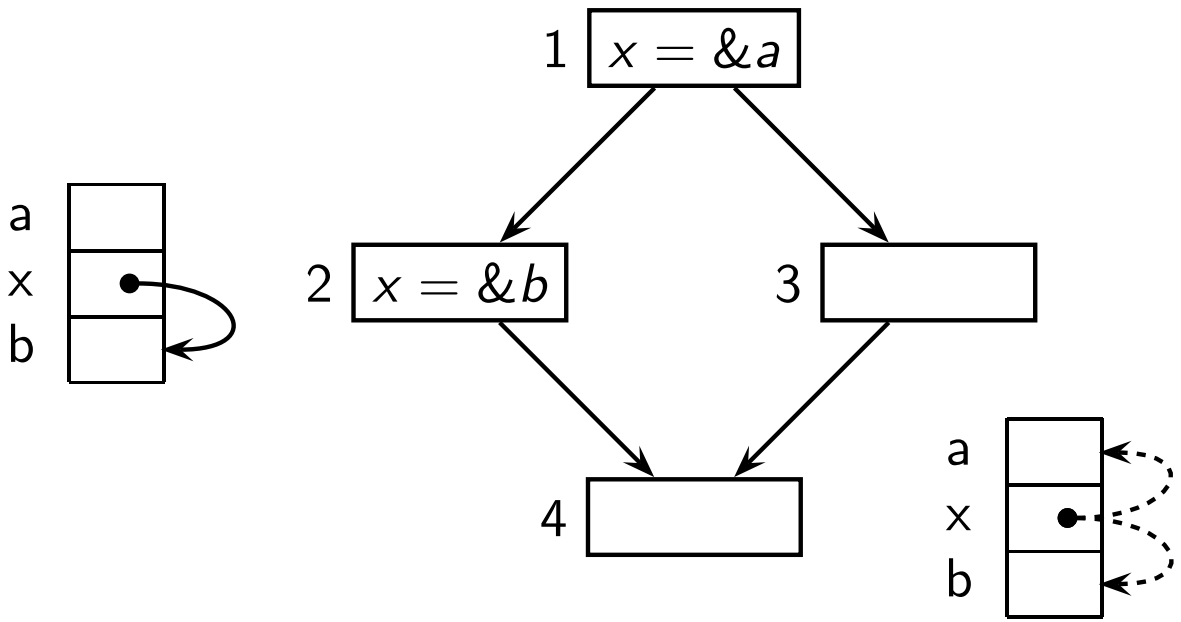
# Must Points-to Information



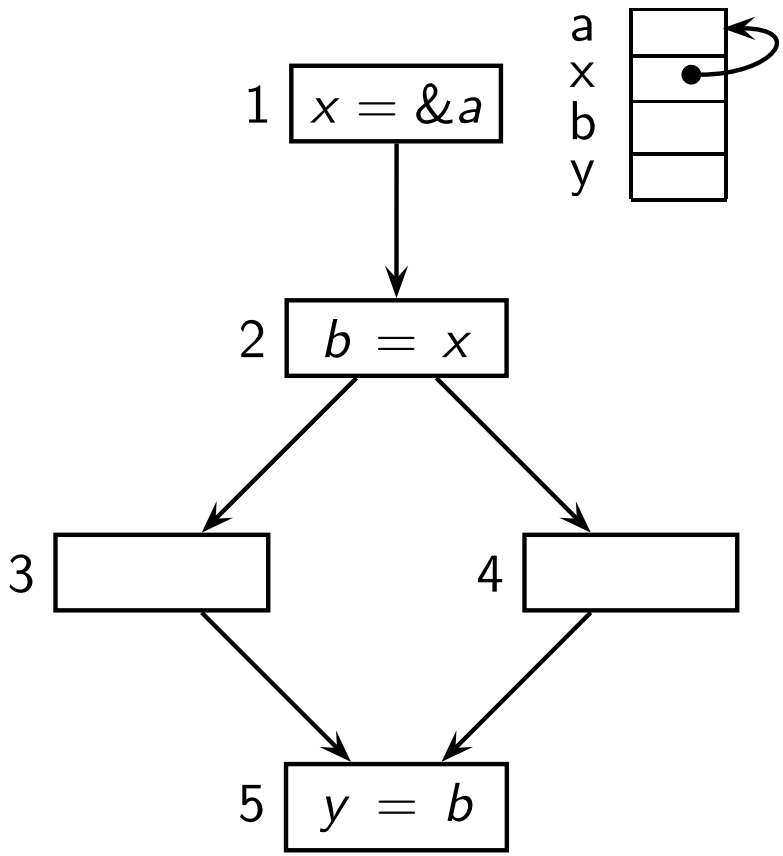
# May Points-to Information



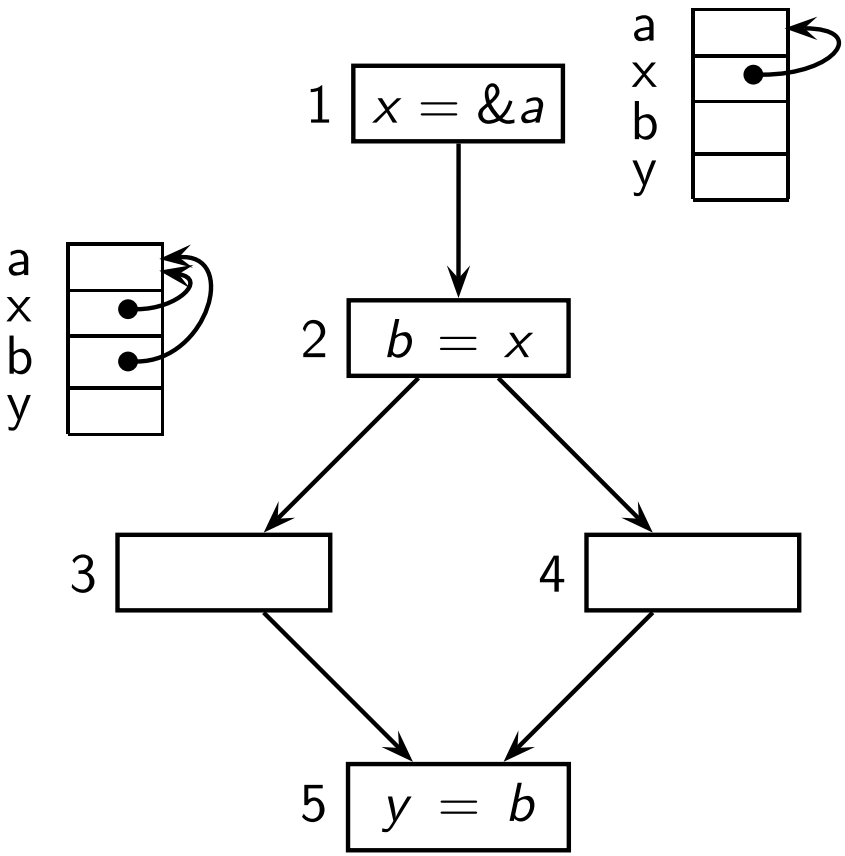
# May Points-to Information



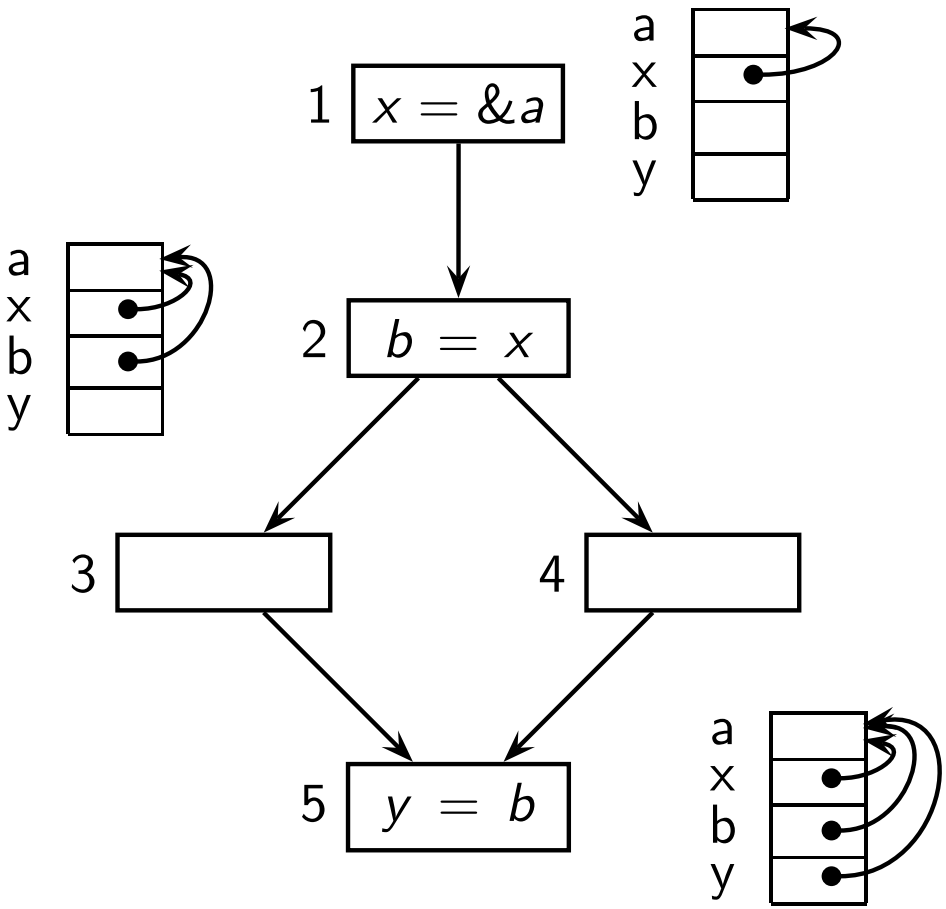
# Must Alias Information



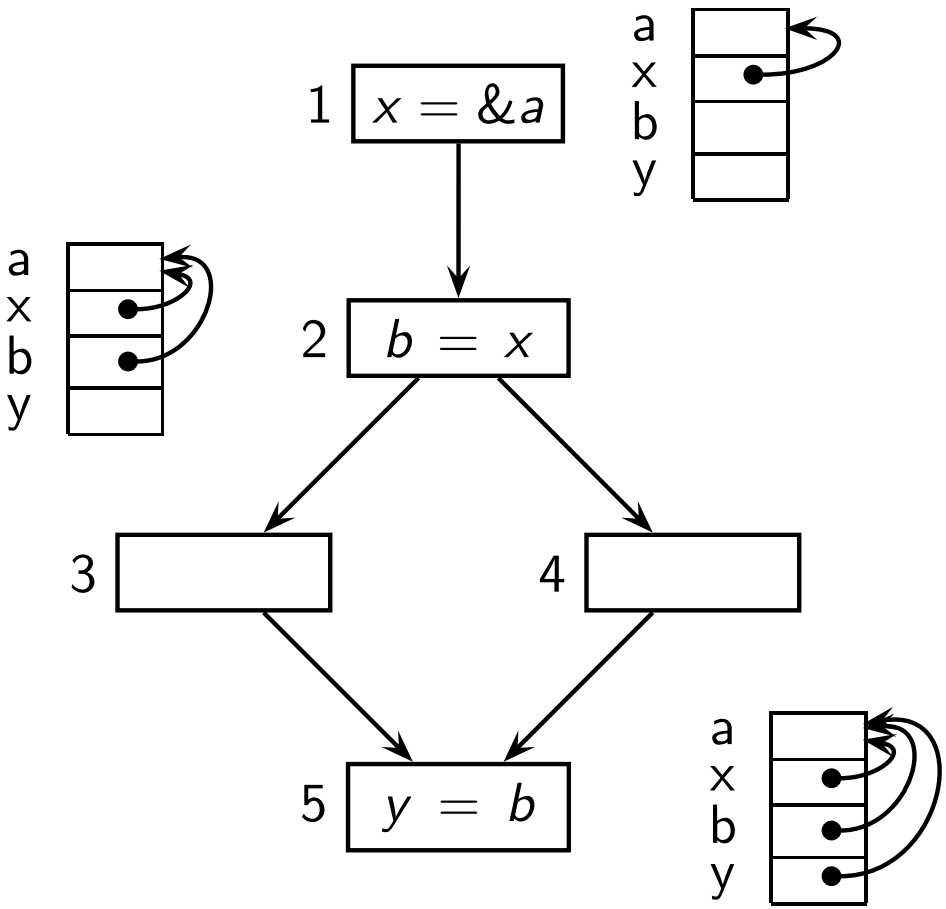
# Must Alias Information



# Must Alias Information

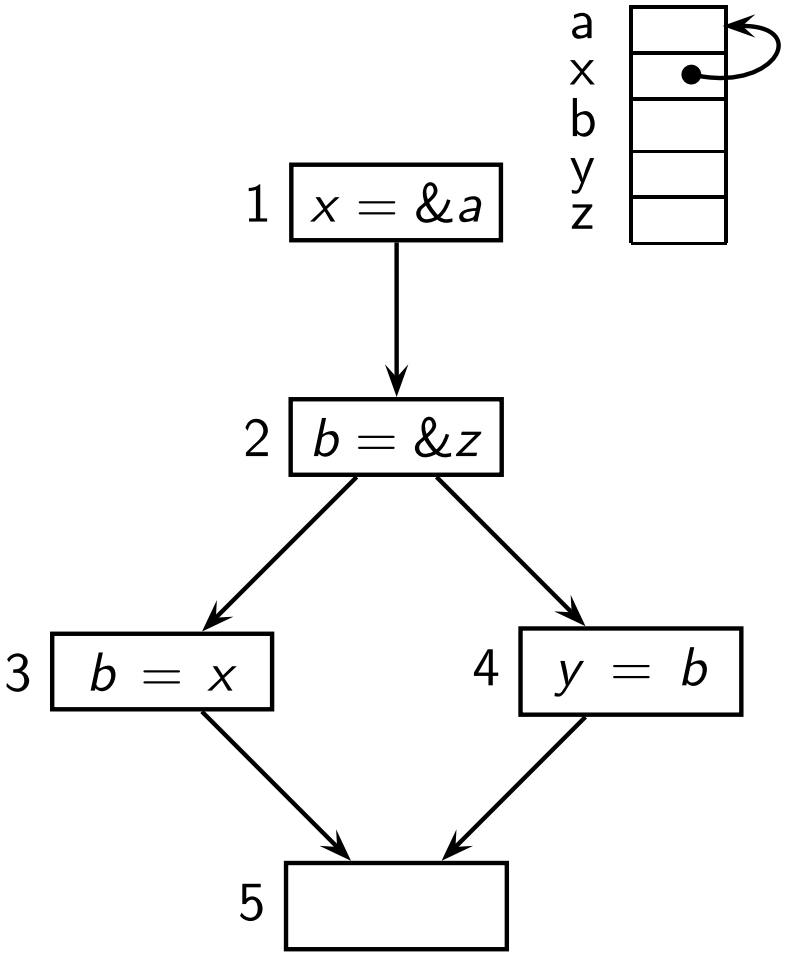


# Must Alias Information

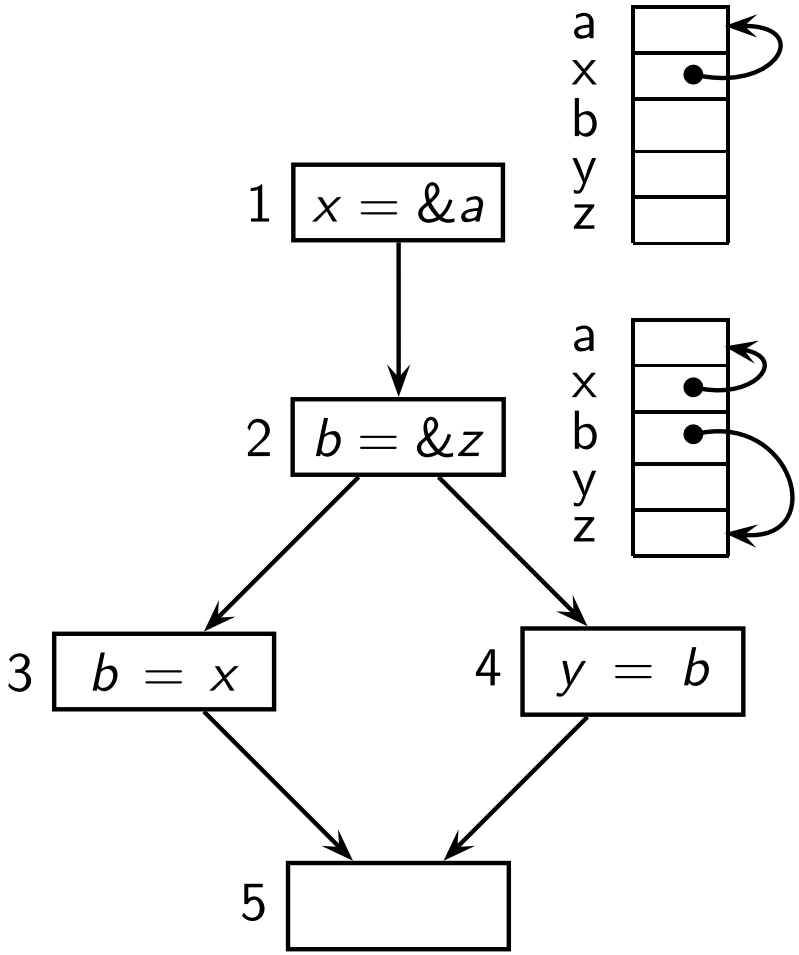



$x \stackrel{\circ}{=} b$ and $b \stackrel{\circ}{=} y \Rightarrow x \stackrel{\circ}{=} y$

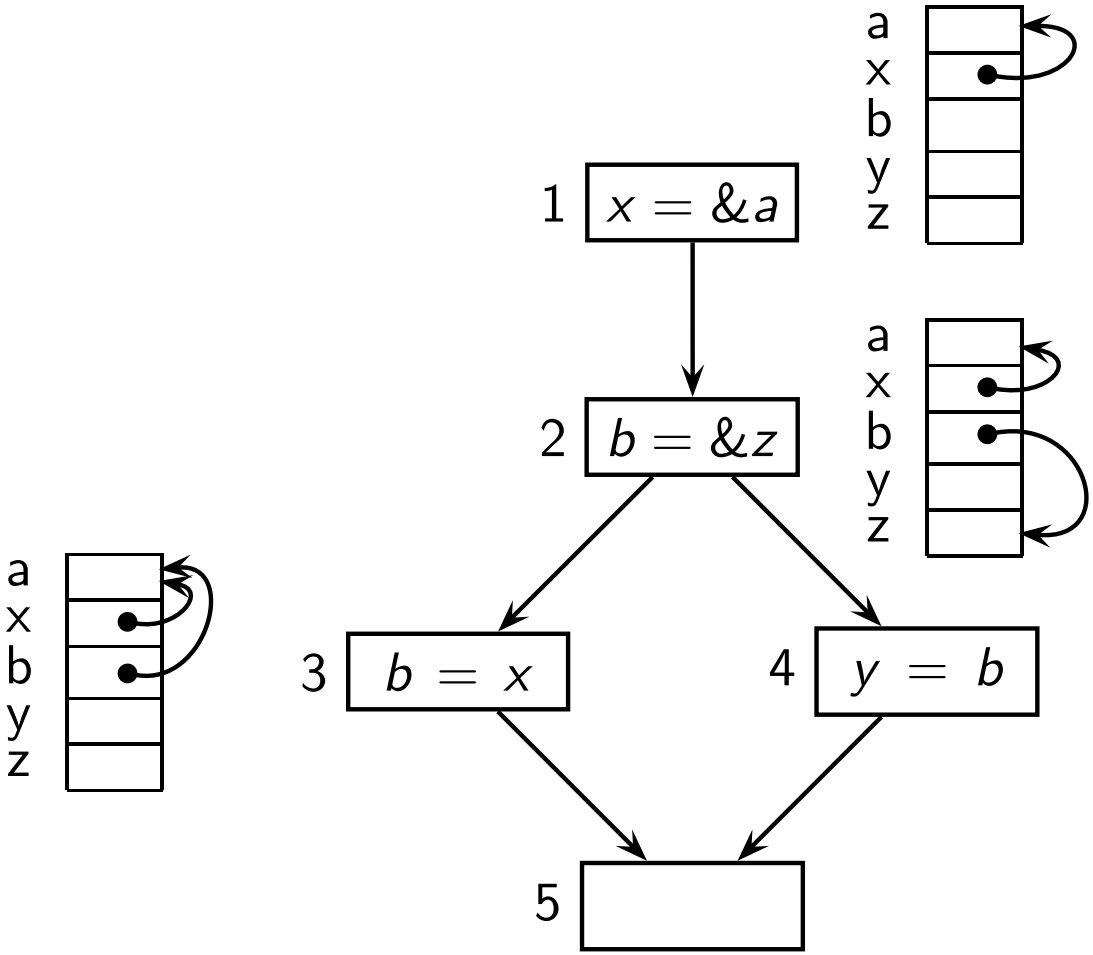
# May Alias Information



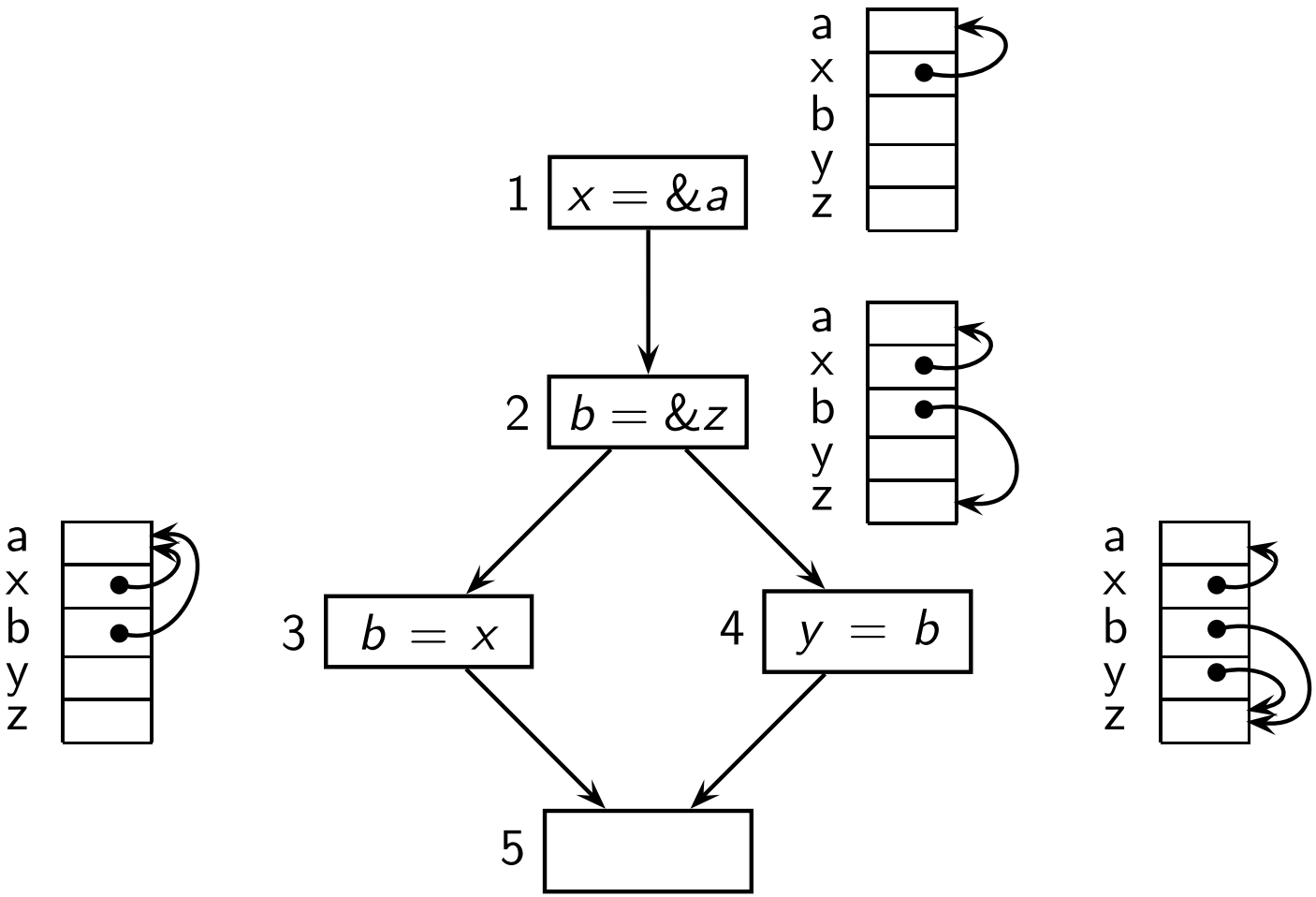
# May Alias Information



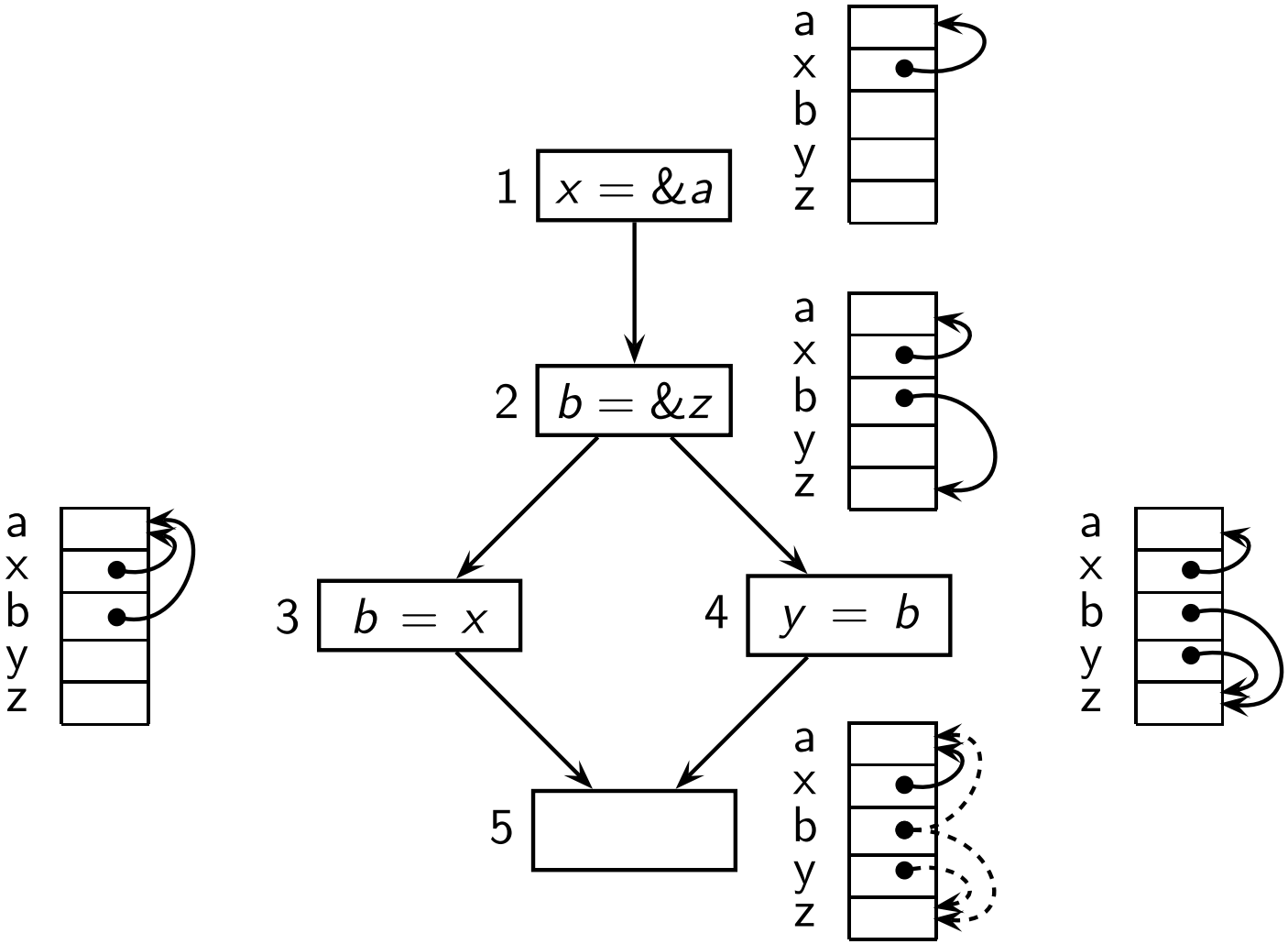
# May Alias Information



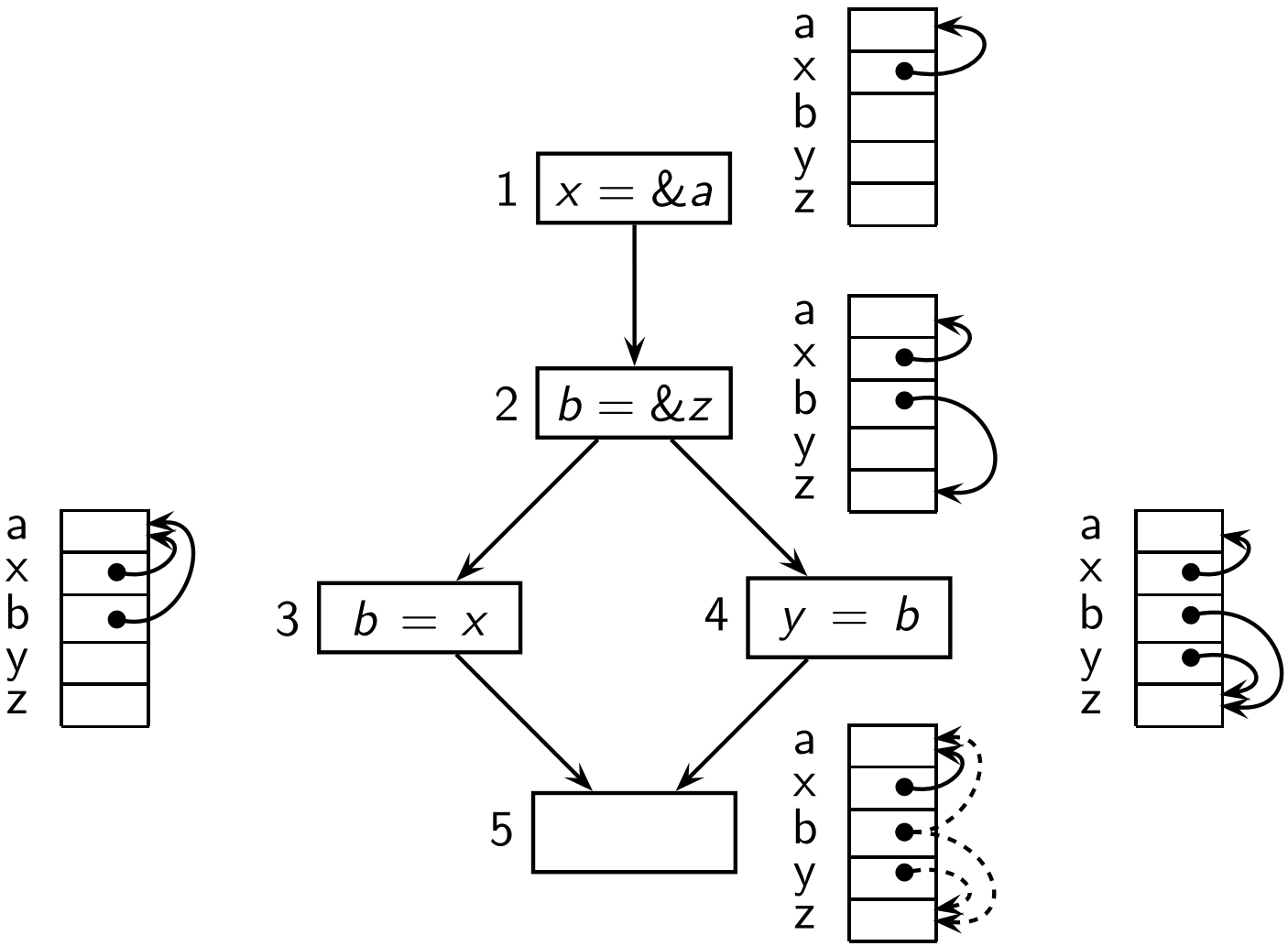
# May Alias Information



# May Alias Information

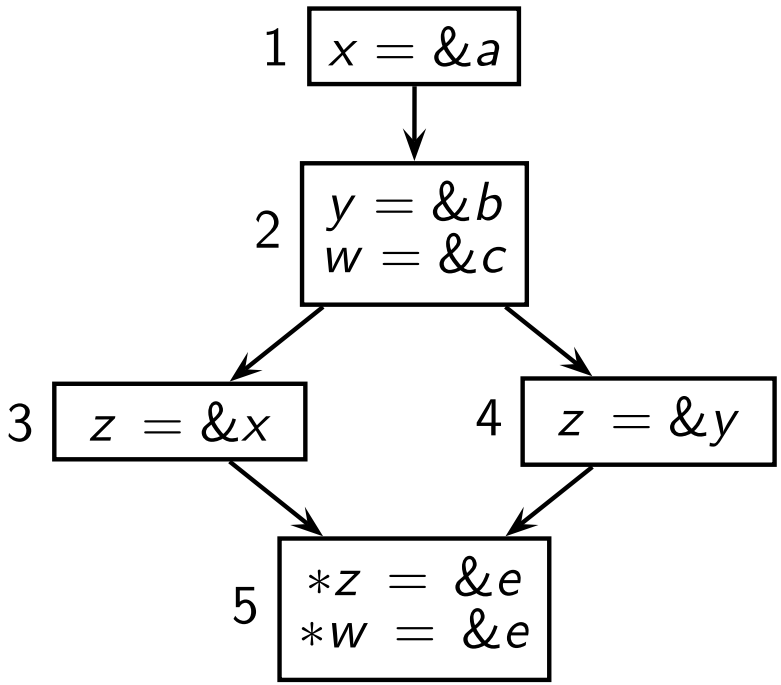


# May Alias Information

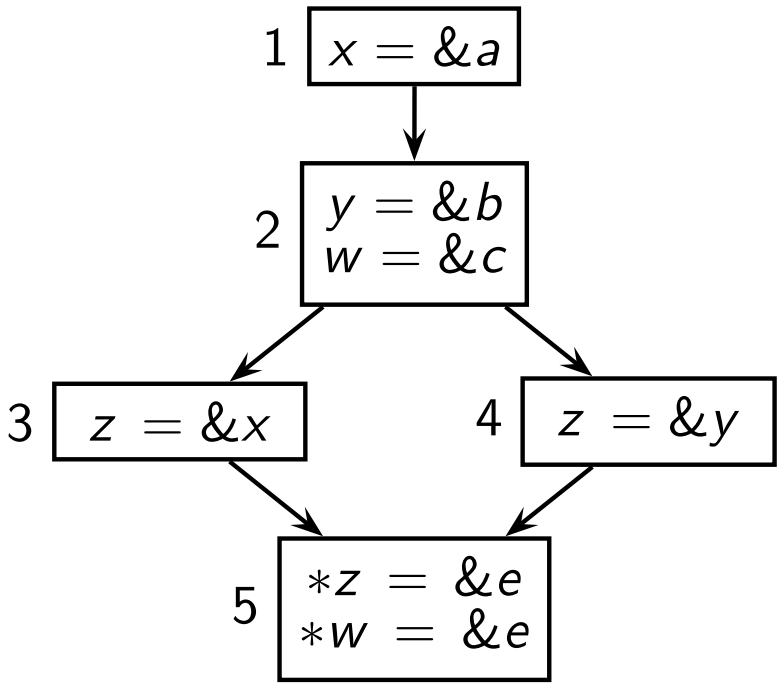



$x \doteq b$ and $b \doteq y \not\Rightarrow x \doteq y$

# Strong and Weak Updates

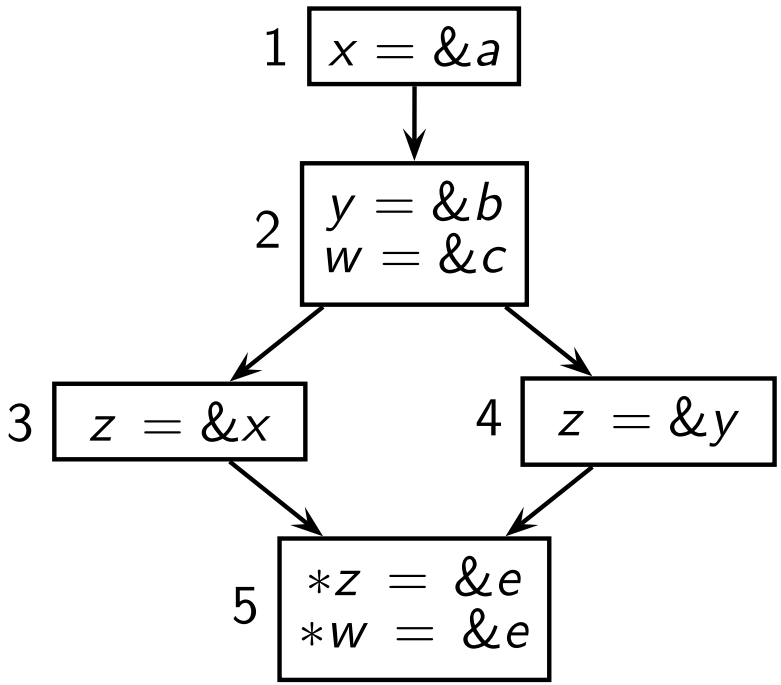


# Strong and Weak Updates




Weak update: Modification of $x$ or $y$ due to $*z$ in block 5

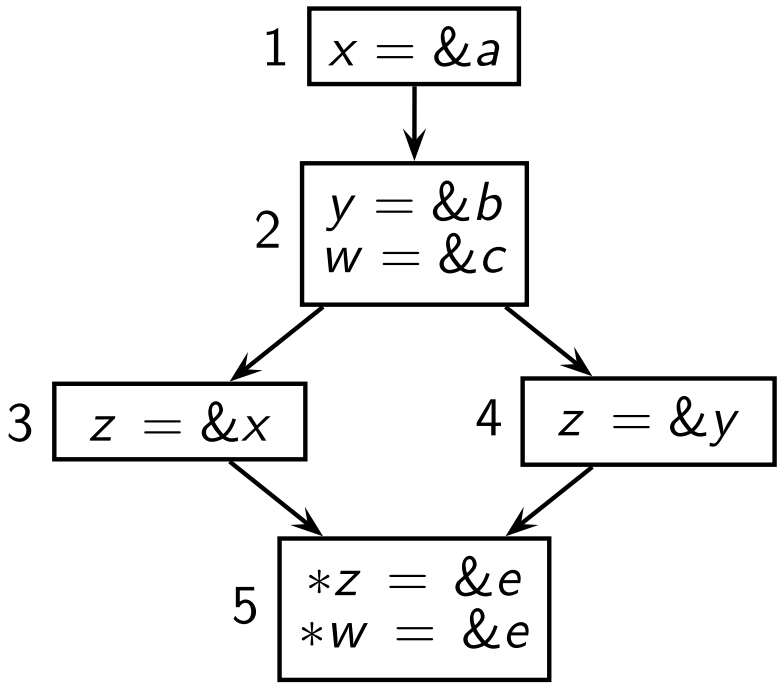
# Strong and Weak Updates




Weak update: Modification of $x$ or $y$ due to $*z$ in block 5

Strong update: Modification of $c$ due to $*w$ in block 5

# Strong and Weak Updates




Weak update: Modification of $x$ or $y$ due to $*z$ in block 5

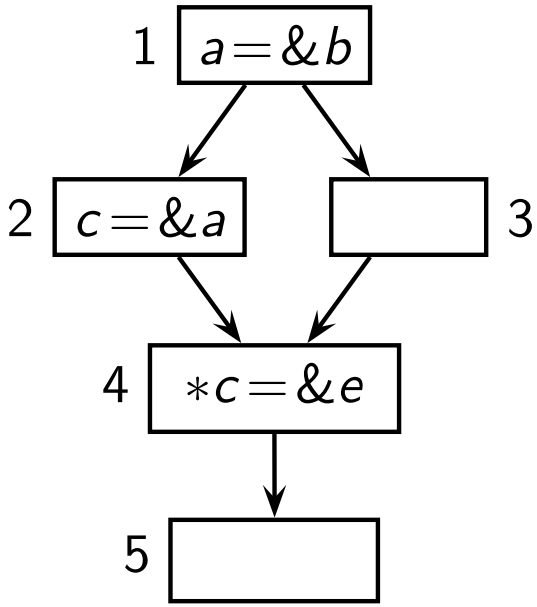
Strong update: Modification of $c$ due to $*w$ in block 5

How is this concept related to May/Must nature of information?

# May and Must Analysis for Killing Points-to Information (1)

*May Points-to Analysis*

*Must Points-to Analysis*



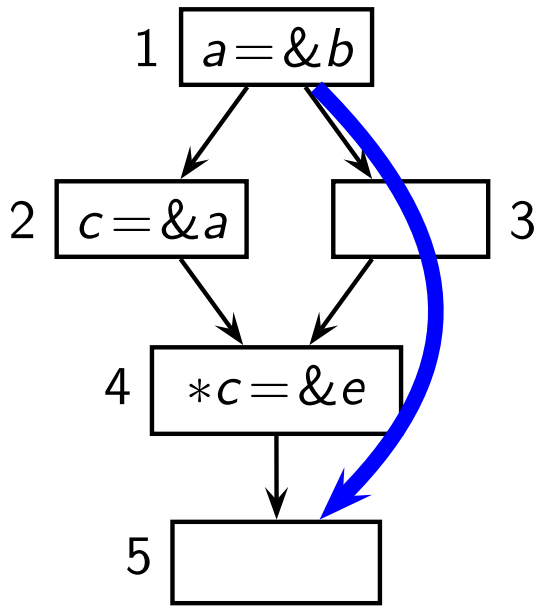
# May and Must Analysis for Killing Points-to Information (1)

*May Points-to Analysis*

- $(a, b)$ should be in $MayIn_5$

  Holds along path 1-3-4
- Block 4 should not kill $(a, b)$
- Possible if pointee set of $c$ is $\emptyset$
- However, $MayIn_4$ contains $(c, a)$

*Must Points-to Analysis*

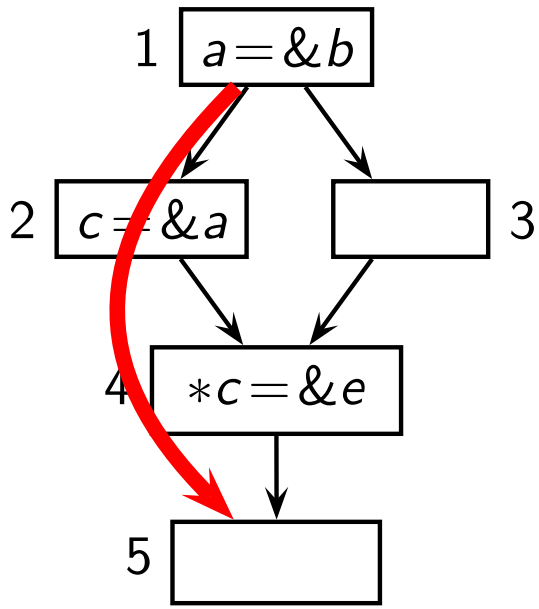


# May and Must Analysis for Killing Points-to Information (1)

## *May Points-to Analysis*

- $(a, b)$ should be in $MayIn_5$

  Holds along path 1-3-4
- Block 4 should not kill $(a, b)$
- Possible if pointee set of $c$ is $\emptyset$
- However, $MayIn_4$ contains $(c, a)$




## *Must Points-to Analysis*

- $(a, b)$ should not be in $MustIn_5$

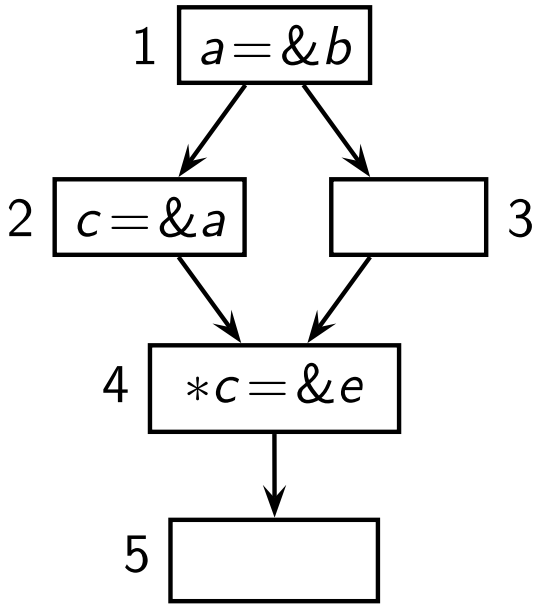
  Does not hold along path 1-2-4
- Block 4 should kill $(a, b)$
- Possible if pointee set of $c$ is $\{a\}$
- However, $MustIn_4$ contains $(a, b)$

# May and Must Analysis for Killing Points-to Information (1)

## *May Points-to Analysis*

- $(a, b)$ should be in $MayIn_5$

  Holds along path 1-3-4
- Block 4 should not kill $(a, b)$
- Possible if pointee set of $c$ is $\emptyset$ (Use $MustIn_4$)
- However, $MayIn_4$ contains $(c, a)$




## *Must Points-to Analysis*

- $(a, b)$ should not be in $MustIn_5$

  Does not hold along path 1-2-4
- Block 4 should kill $(a, b)$
- Possible if pointee set of $c$ is $\{a\}$ (Use $MayIn_4$)
- However, $MustIn_4$ contains $(a, b)$

For killing points-to information through indirection,

- Must points-to analysis should identify pointees of $c$ using $MayIn_4$
- May points-to analysis should identify pointees of $c$ using $MustIn_4$

# May and Must Analysis for Killing Points-to Information (2)

- May Points-to analysis should remove a May points-to pair
  - only if it must be removed along all paths

  Kill should remove only strong updates

  $\Rightarrow$ should use Must Points-to information

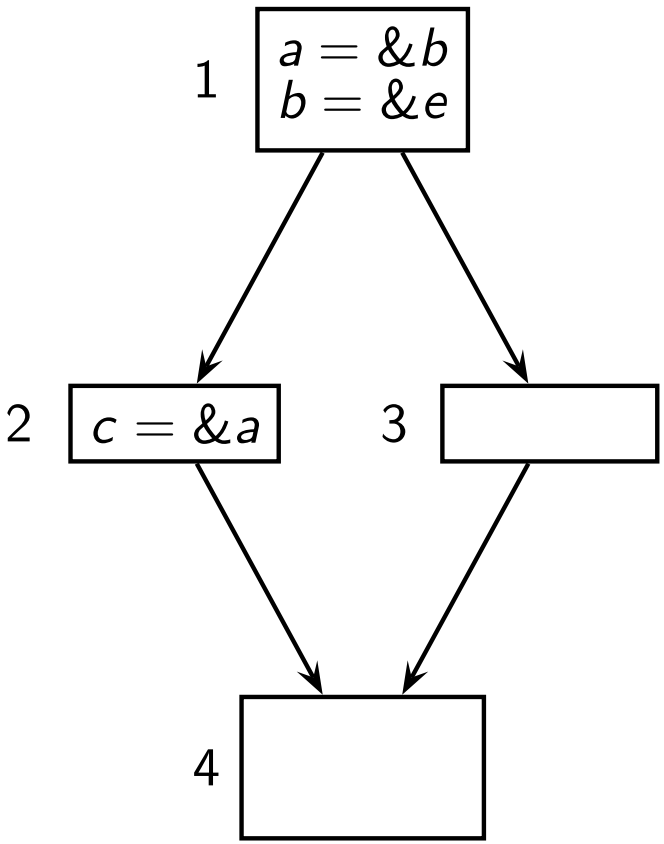
- Must Points-to analysis should remove a Must points-to pair
  - if it can be removed along any path

  Kill should remove all weak updates

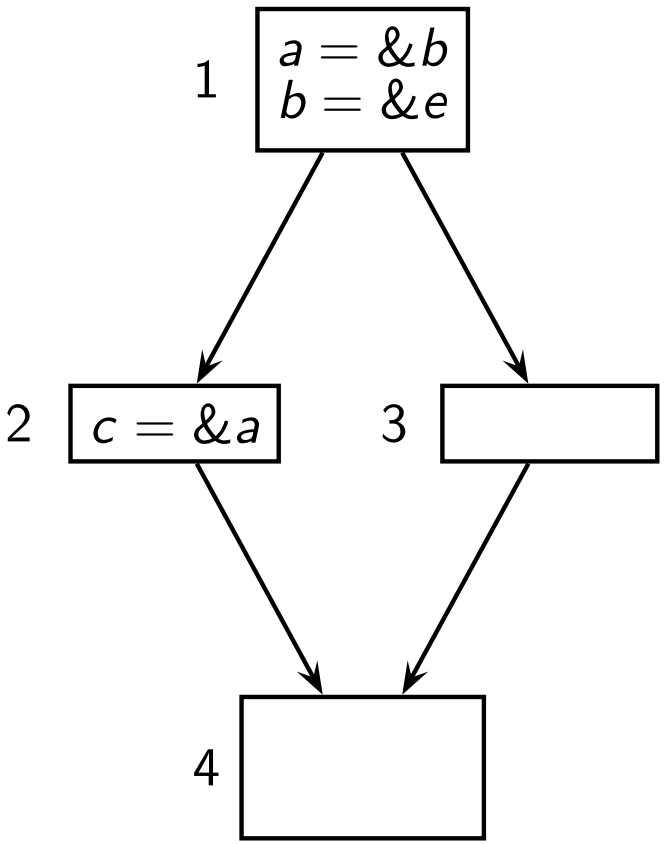
  $\Rightarrow$ should use May Points-to information



# Discovering Must Points-to Information from May Points-to Information



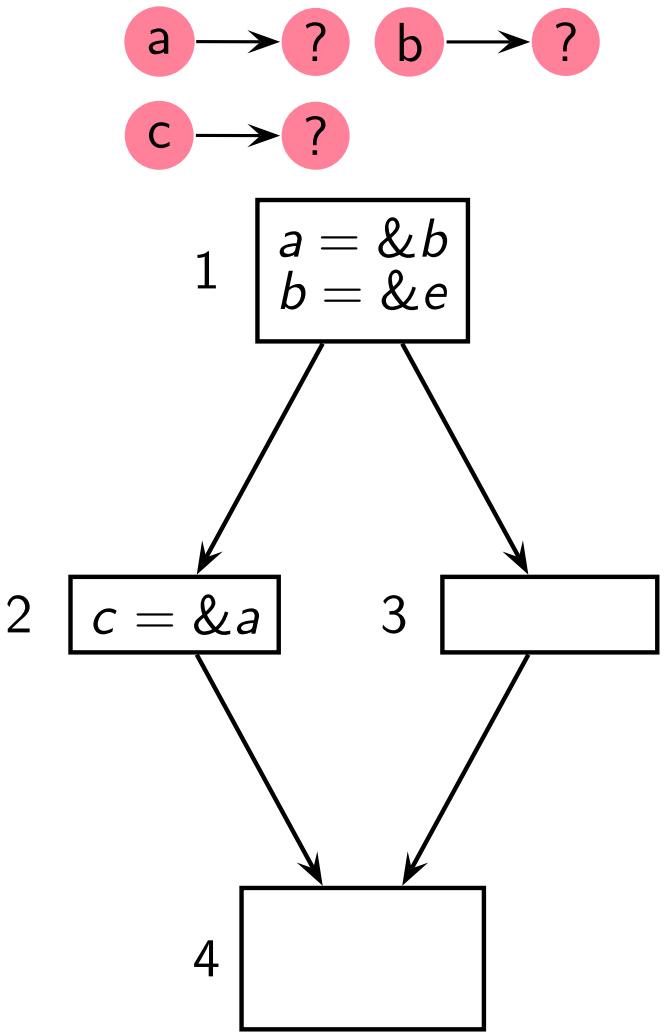
# Discovering Must Points-to Information from May Points-to Information




- *BI*. every pointer points to "?"

  Assume that $e$ is a scalar

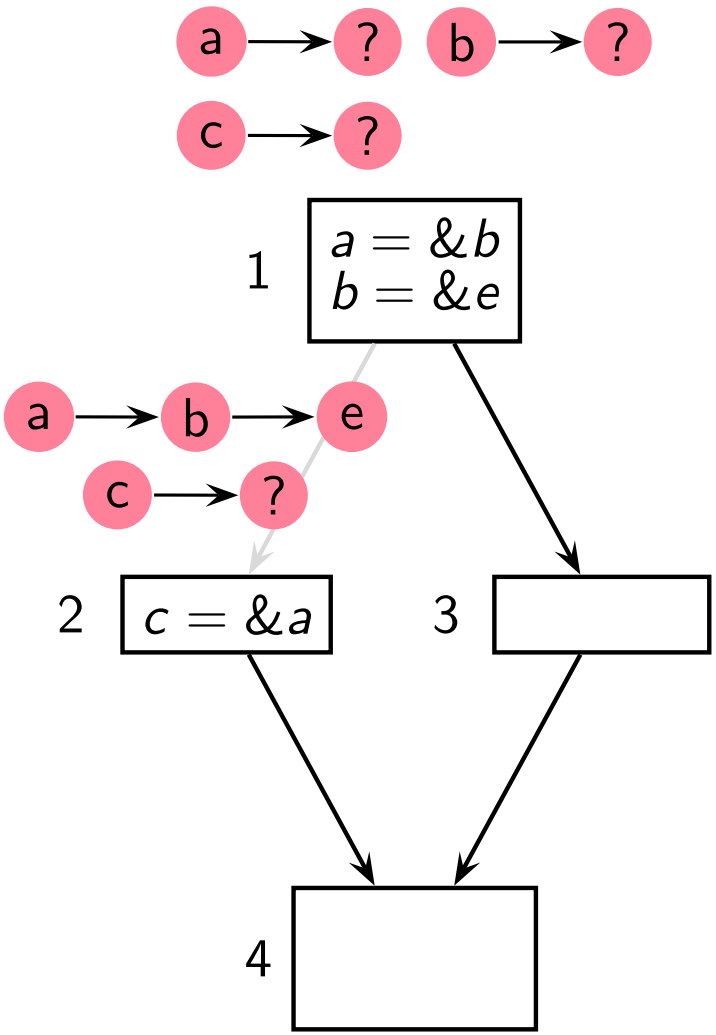
# Discovering Must Points-to Information from May Points-to Information




- *BI*. every pointer points to "?"

  Assume that $e$ is a scalar

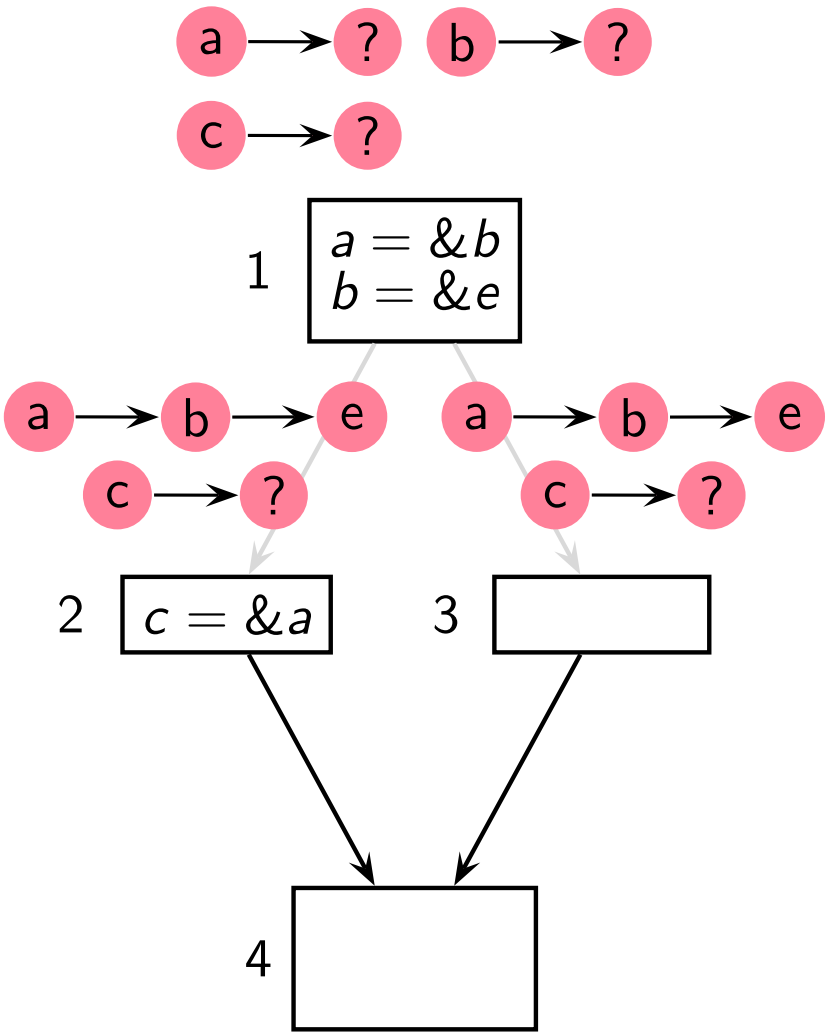
# Discovering Must Points-to Information from May Points-to Information




- *BI.* every pointer points to "?"

  Assume that $e$ is a scalar
- Perform usual may points-to analysis

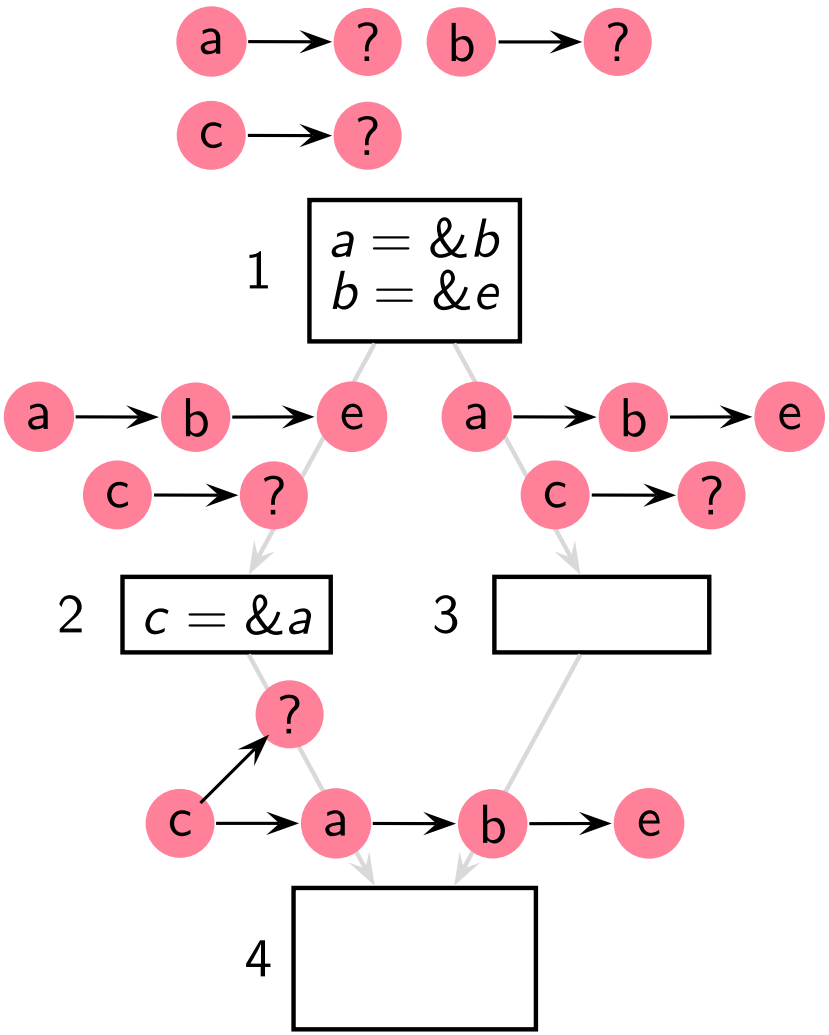
# Discovering Must Points-to Information from May Points-to Information




- *BI.* every pointer points to "?"

  Assume that $e$ is a scalar
- Perform usual may points-to analysis

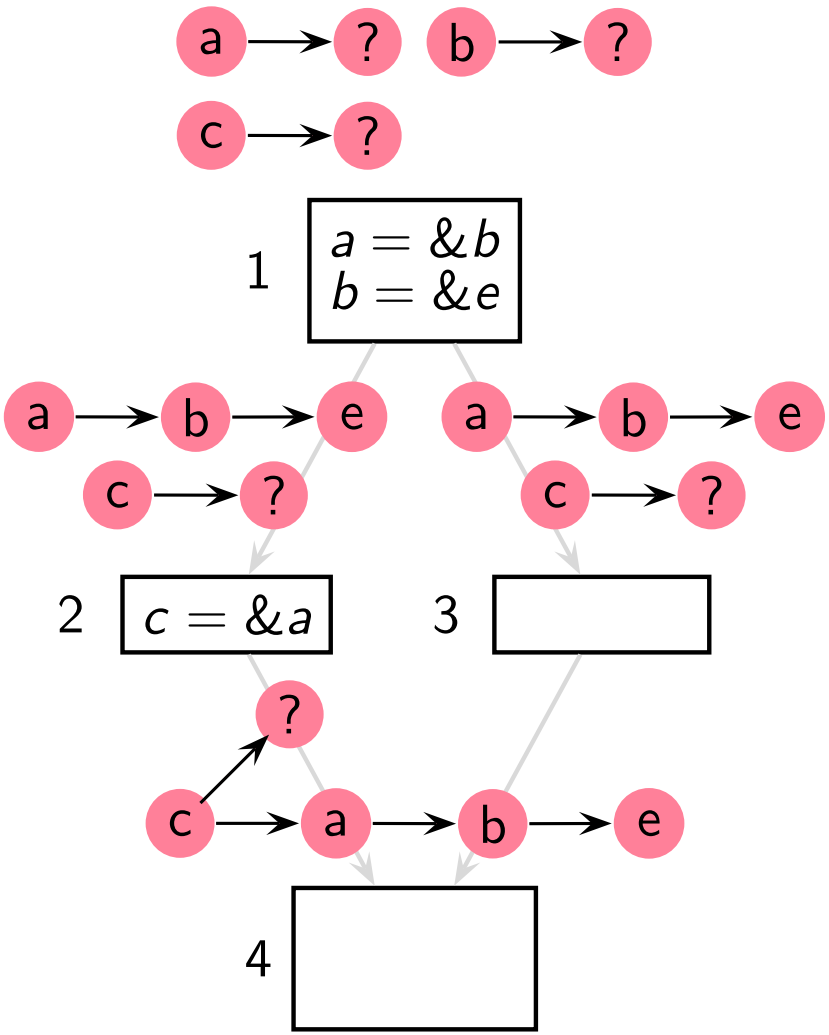
# Discovering Must Points-to Information from May Points-to Information




- *BI.* every pointer points to "?"

  Assume that $e$ is a scalar
- Perform usual may points-to analysis

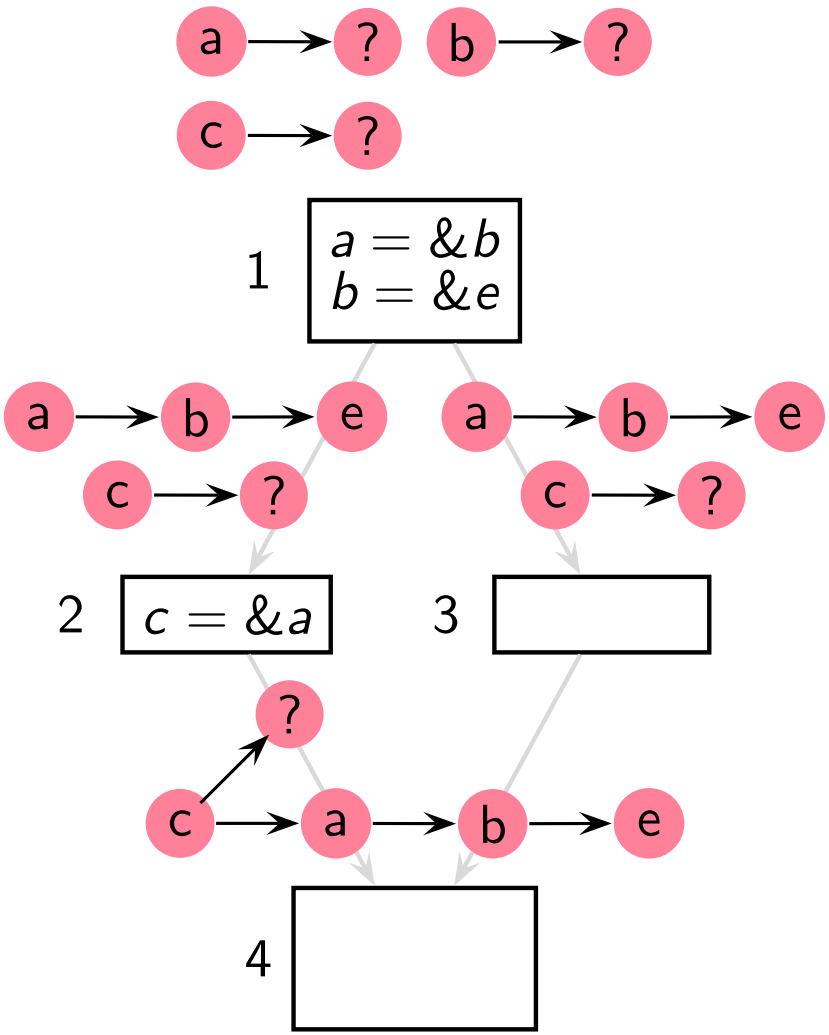
# Discovering Must Points-to Information from May Points-to Information




- *BI.* every pointer points to "?"

  Assume that *e* is a scalar
- Perform usual may points-to analysis
- Since c has multiple pointees, it is a MAY relation

# Discovering Must Points-to Information from May Points-to Information




- *BI.* every pointer points to "?"

  Assume that *e* is a scalar
- Perform usual may points-to analysis
- Since c has multiple pointees, it is a MAY relation
- Since a has a single pointee, it is a MUST relation

# Relevant Algebraic Operations on Relations (1)

- Let $\mathbf{P} \subseteq V$ be the set of pointer variables
- May-points-to information: $\mathcal{A} = \langle 2^{\mathbf{P} \times V}, \supseteq \rangle$
- Standard algebraic operations on points-to relations

  Given relation $R \subseteq \mathbf{P} \times V$ and $X \subseteq \mathbf{P}$,
  - Relation *application* $R\ X = \{v \mid u \in X \wedge (u, v) \in R\}$
  - Relation *restriction* $(R|_X)$ $R|_X = \{(u, v) \in R \mid u \in X\}$



# Relevant Algebraic Operations on Relations (1)

- Let $\mathbf{P} \subseteq V$ be the set of pointer variables
- May-points-to information: $\mathcal{A} = \langle 2^{\mathbf{P} \times V}, \supseteq \rangle$
- Standard algebraic operations on points-to relations

  Given relation $R \subseteq \mathbf{P} \times V$ and $X \subseteq \mathbf{P}$,
  - Relation *application* $R\ X = \{v \mid u \in X \wedge (u, v) \in R\}$
    (Find out the pointees of the pointers contained in $X$)
  - Relation *restriction* $(R|_X)$ $R|_X = \{(u, v) \in R \mid u \in X\}$



# Relevant Algebraic Operations on Relations (1)

- Let $\mathbf{P} \subseteq V$ be the set of pointer variables
- May-points-to information: $\mathcal{A} = \langle 2^{\mathbf{P} \times V}, \supseteq \rangle$
- Standard algebraic operations on points-to relations

  Given relation $R \subseteq \mathbf{P} \times V$ and $X \subseteq \mathbf{P}$,
  - Relation *application* $R\ X = \{v \mid u \in X \wedge (u, v) \in R\}$
    (Find out the pointees of the pointers contained in $X$)
  - Relation *restriction* $(R|_X)$ $R|_X = \{(u, v) \in R \mid u \in X\}$
    (Restrict the relation only to the pointers contained in $X$ by removing points-to information of other pointers)



# Relevant Algebraic Operations on Relations (2)

*Let*

$$\begin{aligned} V &= \{a, b, c, d, e, f, g, ?\} \\ \mathbf{P} &= \{a, b, c, d, e\} \\ R &= \{(a, b), (a, c), (b, d), (c, e), (c, g), (e, ?)\} \\ X &= \{a, c\} \end{aligned}$$

Then,

$$R\,X = \{v \mid u \in X \wedge (u, v) \in R\}$$

$$R|_X = \{(u, v) \in R \mid u \in X\}$$



# Relevant Algebraic Operations on Relations (2)

*Let*

$$\begin{aligned} V &= \{a, b, c, d, e, f, g, ?\} \\ \mathbf{P} &= \{a, b, c, d, e\} \\ R &= \{(a, b), (a, c), (b, d), (c, e), (c, g), (e, ?)\} \\ X &= \{a, c\} \end{aligned}$$

Then,

$$\begin{aligned} R\,X &= \{v \mid u \in X \wedge (u, v) \in R\} \\ &= \{b, c, e, g\} \\ R|_X &= \{(u, v) \in R \mid u \in X\} \end{aligned}$$

# Relevant Algebraic Operations on Relations (2)

*Let*

$$
\begin{aligned}
V &= \{a, b, c, d, e, f, g, ?\} \\
\mathbf{P} &= \{a, b, c, d, e\} \\
R &= \{(a, b), (a, c), (b, d), (c, e), (c, g), (e, ?)\} \\
X &= \{a, c\}
\end{aligned}
$$

Then,

$$
\begin{aligned}
R\,X &= \{v \mid u \in X \wedge (u, v) \in R\} \\
&= \{b, c, e, g\} \\
R|_X &= \{(u, v) \in R \mid u \in X\} \\
&= \{(a, b), (a, c), (c, e), (c, g)\}
\end{aligned}
$$

# Points-to Analysis Data Flow Equations

$$Ain_n = \begin{cases} V \times \{?\} & n \text{ is } Start_p \\ \bigcup_{p \in pred(n)} Aout_p & \text{otherwise} \end{cases}$$

$$Aout_n = \left(Ain_n - \left( Kill_n \times V \right)\right) \cup \left( Def_n \times Pointee_n \right)$$

- $Ain/Aout$: sets of mAy points-to pairs
- $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$



# Points-to Analysis Data Flow Equations

$$Ain_n = \begin{cases} V \times \{?\} & n \text{ is } Start_p \\ \bigcup_{p \in pred(n)} Aout_p & \text{otherwise} \end{cases}$$

$$Aout_n = \left(Ain_n - \left(Kill_n \times V\right)\right) \cup \left(Def_n \times Pointee_n\right)$$

- $Ain/Aout$: sets of mAy points-to pairs
- $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$

Pointers whose points-to relations should be removed



# Points-to Analysis Data Flow Equations

$$Ain_n = \begin{cases} V \times \{?\} & n \text{ is } Start_p \\ \bigcup_{p \in pred(n)} Aout_p & \text{otherwise} \end{cases}$$

$$Aout_n = \left(Ain_n - \left(Kill_n \times V\right)\right) \cup \left(Def_n \times Pointee_n\right)$$

- $Ain/Aout$: sets of mAy points-to pairs
- $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$

Pointers that are defined (i.e. pointers in which addresses are stored)

# Points-to Analysis Data Flow Equations

Pointees (i.e. locations whose addresses are stored)

$$Ain_n = \begin{cases} V \times \{?\} & n \text{ is } Start_p \\ \bigcup_{p \in pred(n)} Aout_p & \text{otherwise} \end{cases}$$

$$Aout_n = \left(Ain_n - \left(Kill_n \times V\right)\right) \cup \left(Def_n \times Pointee_n\right)$$

- $Ain/Aout$: sets of mAy points-to pairs
- $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$



# Points-to Analysis Data Flow Equations

$$Ain_n = \begin{cases} V \times \{?\} & n \text{ is } Start_p \\ \bigcup_{p \in pred(n)} Aout_p & \text{otherwise} \end{cases}$$

$$Aout_n = \left(Ain_n - \left( Kill_n \times V \right)\right) \cup \left( Def_n \times Pointee_n \right)$$

- $Ain/Aout$: sets of mAy points-to pairs
- $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | | | |
| $x = \&a$ | | | |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | | | |
| $x = \&a$ | | | |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |

Pointers that are defined (i.e. pointers in which addresses are stored)

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | | | |
| $x = \&a$ | | | |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |

Pointees (i.e. locations whose addresses are stored)

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | | | |
| $x = \&a$ | | | |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |

Pointers whose points-to relations should be removed

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | | | |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | | | |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | | | |
| $*x = y$ | | | |
| other | | | |

Pointees of $y$ in $Ain_n$ are the targets of defined pointers



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | | | |
| other | | | |

Pointees of those pointees of $y$ in $Ain_n$ which are pointers

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

Pointees of $x$ in $Ain_n$ receive new addresses

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$

Strong update using must-points-to information computed from $Ain_n$

| | $Def_n$ | $Kill_n$ | |
|---|---|---|---|
| *use x* | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$

Strong update using must-points-to information computed from $Ain_n$

| | $Def_n$ | $Kill_n$ | |
|---|---|---|---|
| *use x* | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$

Find out must-pointees of all pointers



# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$

Strong update using must-points-to information computed from $Ain_n$

| | $Def_n$ | $Kill_n$ | |
|---|---|---|---|
| *use x* | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$

$z$ has a single pointee $w$ in must-points-to relation

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$

Strong update using must-points-to information computed from $Ain_n$

| | $Def_n$ | $Kill_n$ | |
|---|---|---|---|
| *use x* | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$

$z$ has no pointee in must-points-to relation

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | | | |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$

Pointees of $y$ in $Ain_n$ are the targets of defined pointers

# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

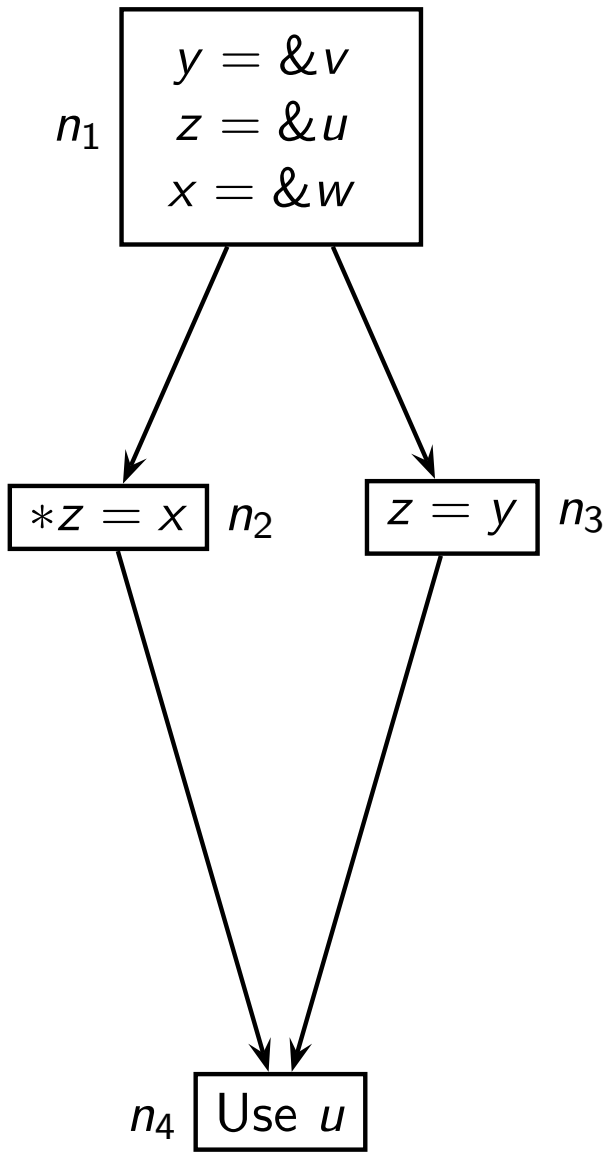
| | $Def_n$ | $Kill_n$ | $Pointee_n$ |
|---|---|---|---|
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ |

$$Must(R) = \bigcup_{z \in \mathbf{P}} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \wedge w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$



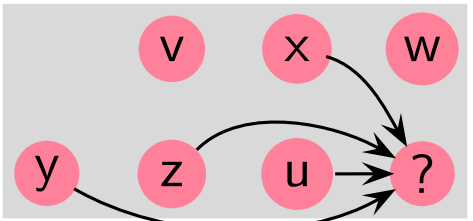
# An Example of Flow Sensitive May Points-to Analysis

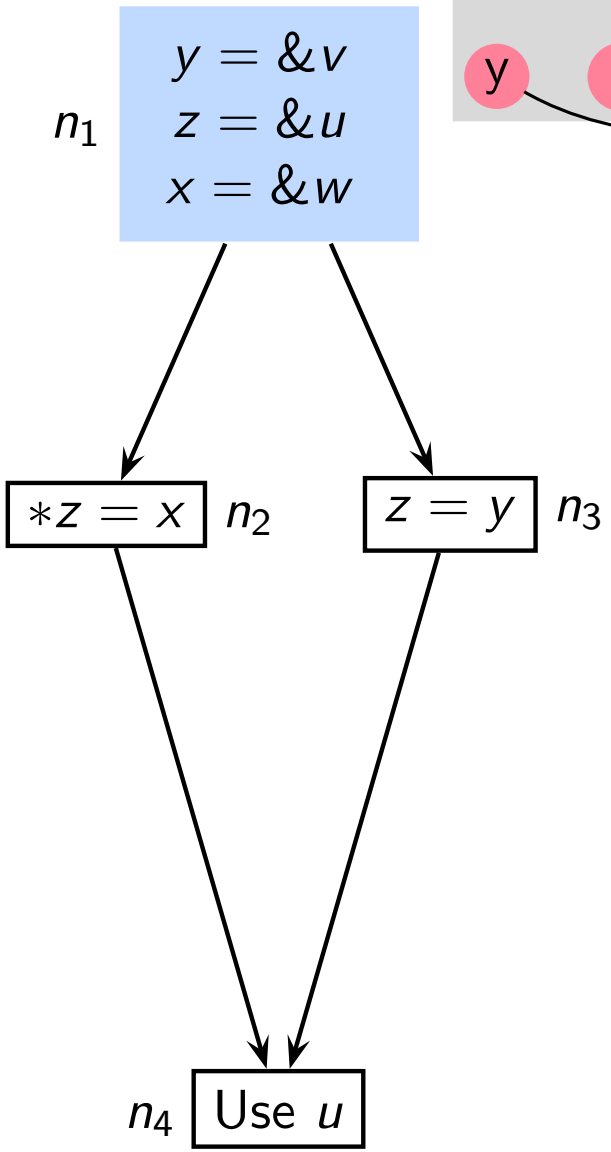
$v$ and $w$ are not pointers



# An Example of Flow Sensitive May Points-to Analysis

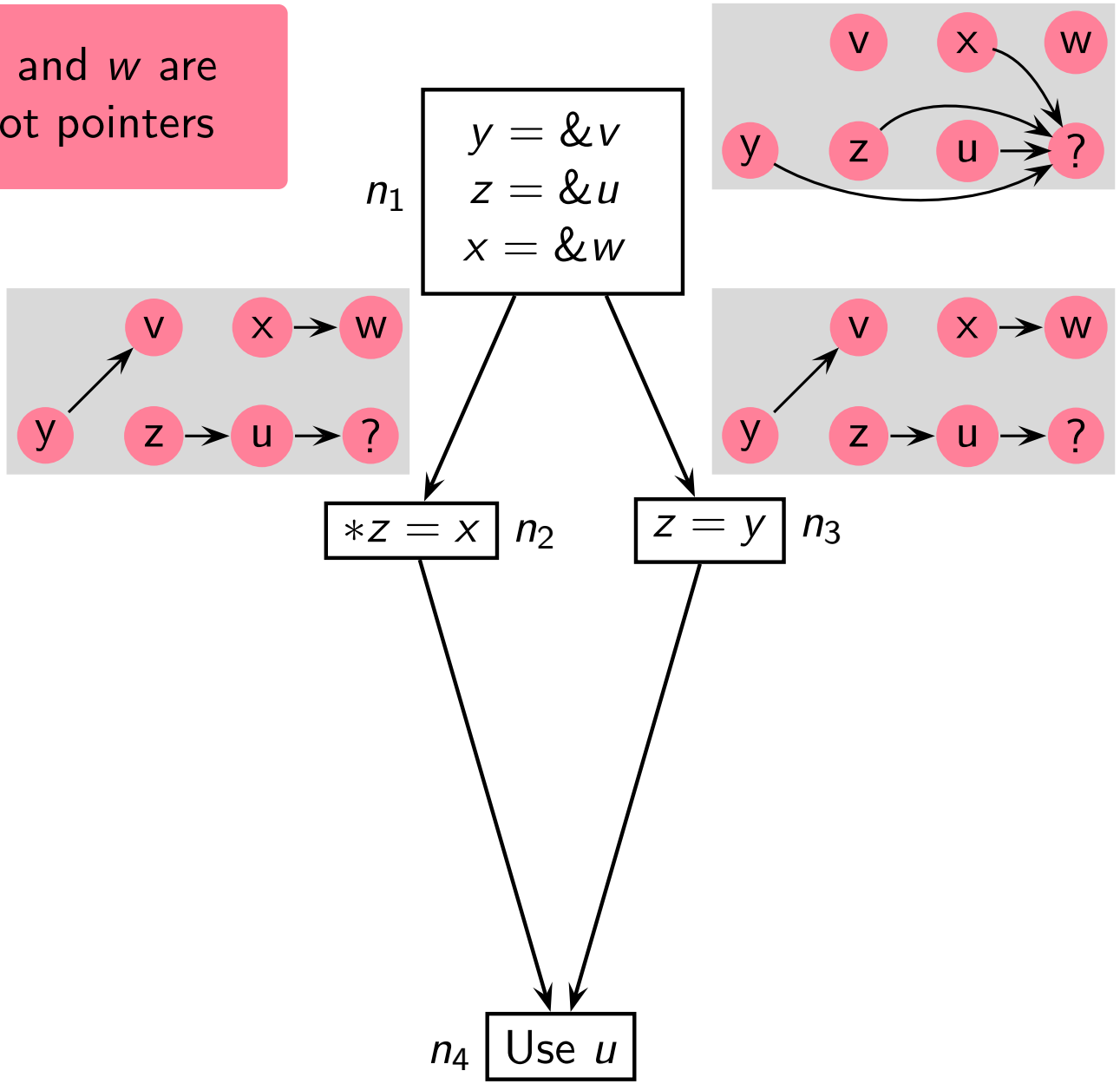
$v$ and $w$ are not pointers



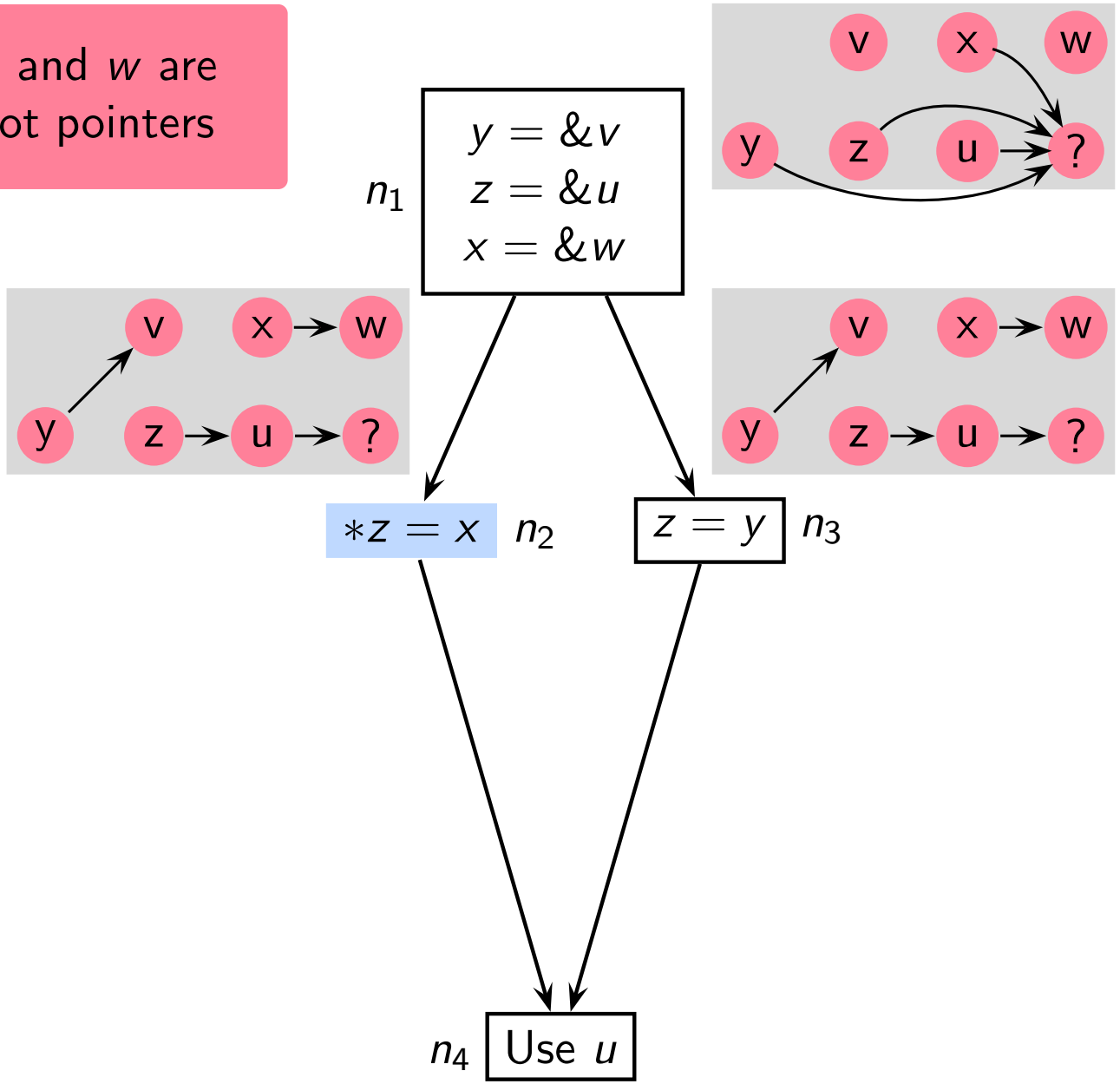
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers



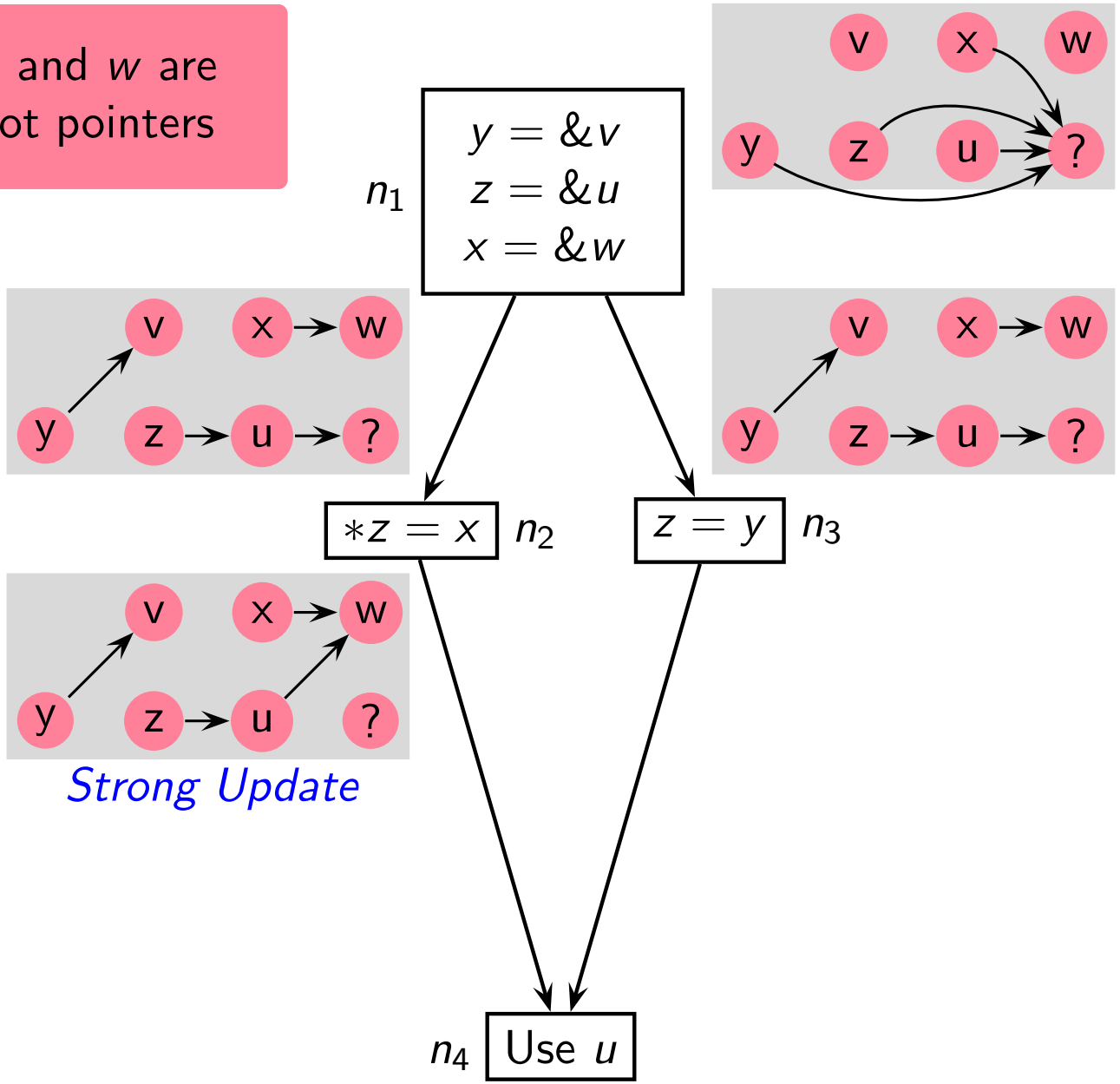
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers



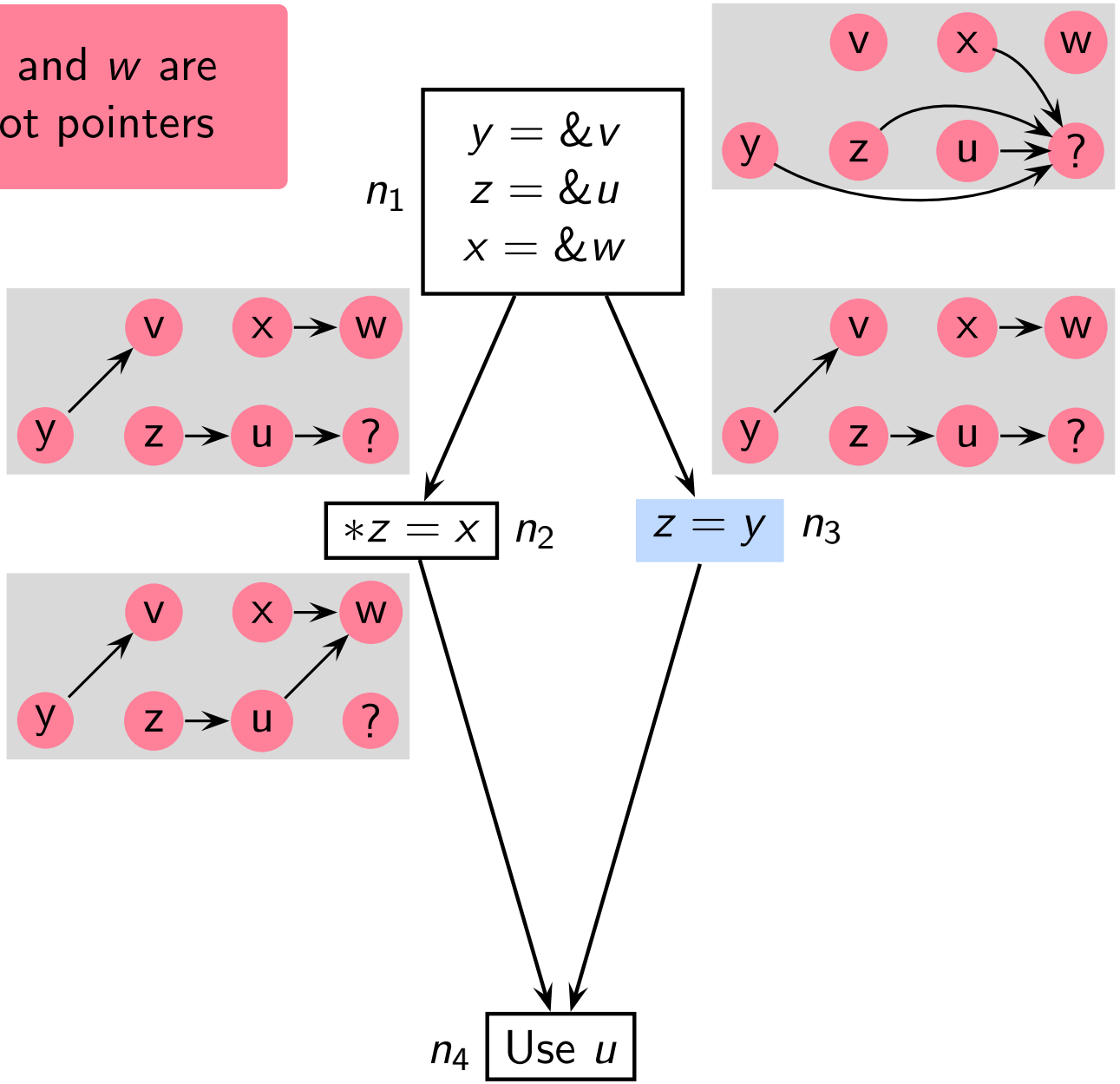
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers



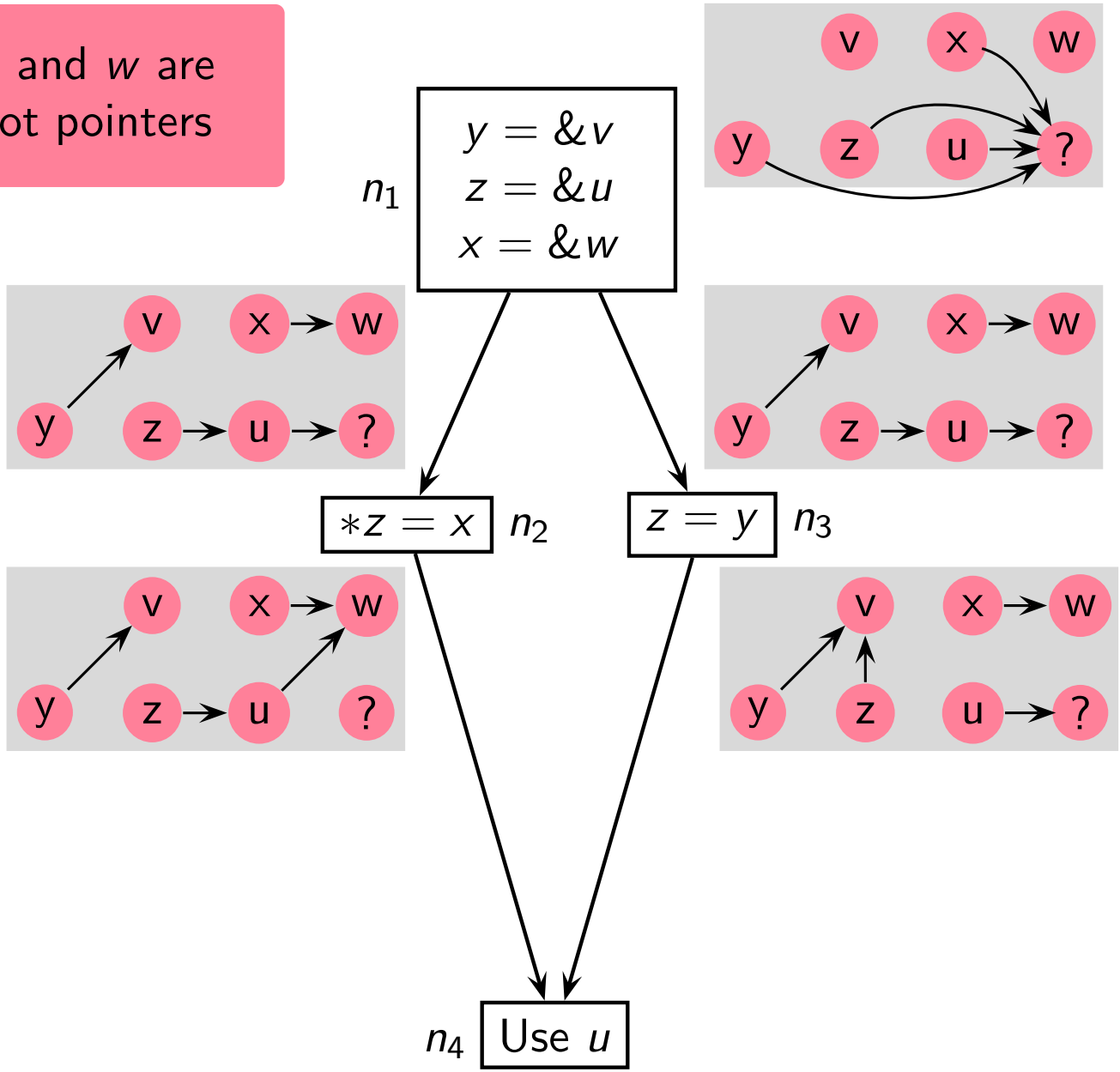
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers



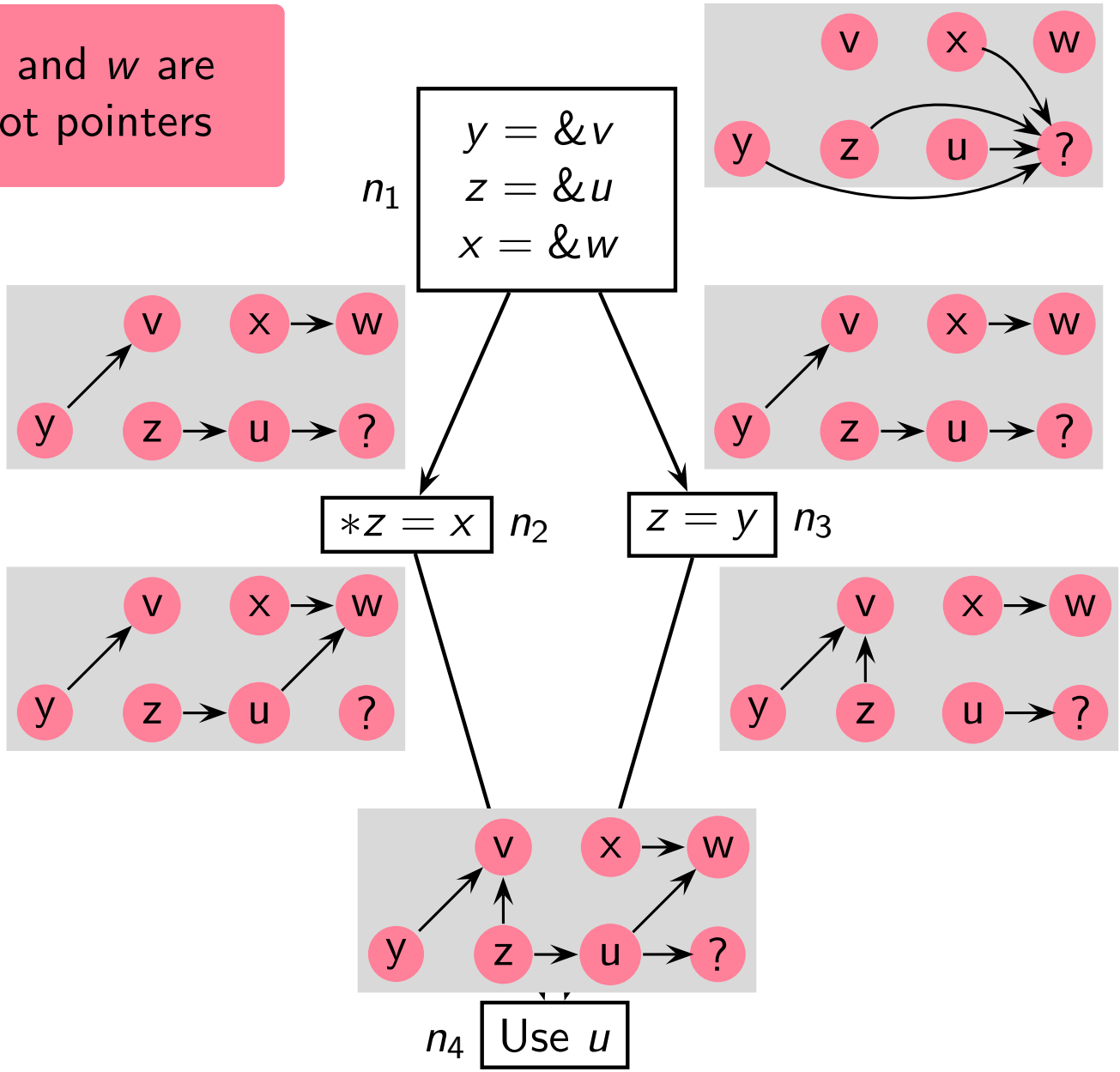
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers



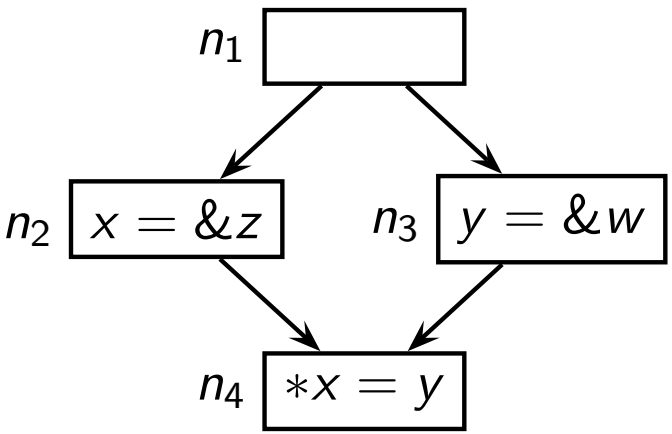
# An Example of Flow Sensitive May Points-to Analysis

$v$ and $w$ are not pointers

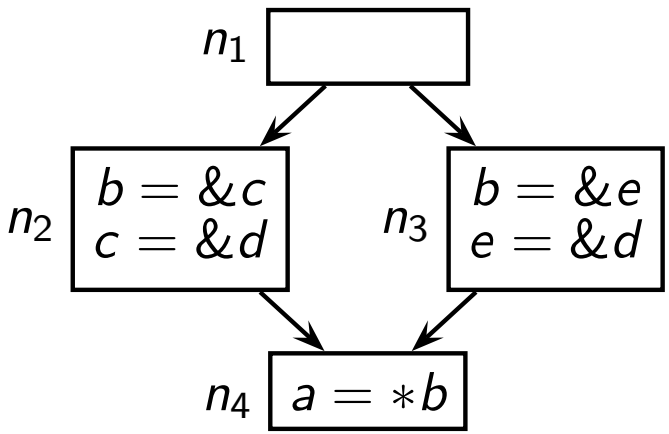


# Non-Distributivity of Points-to Analysis

May Points-to

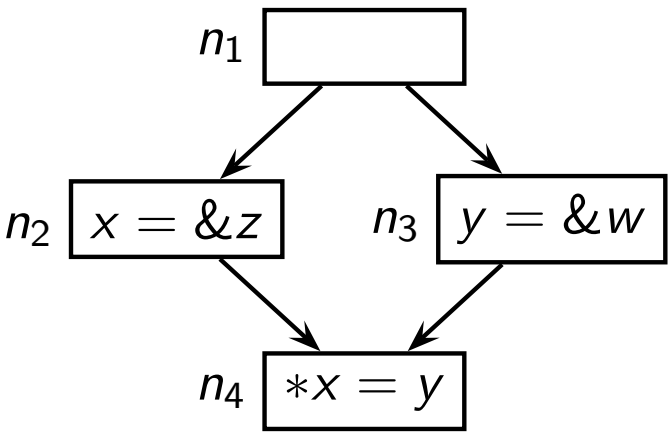



Must Points-to



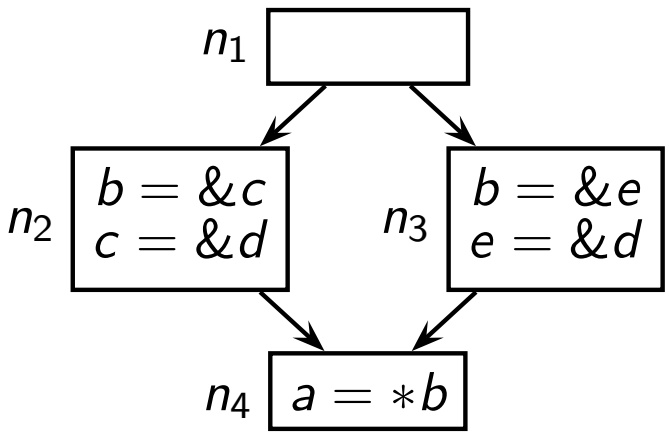
# Non-Distributivity of Points-to Analysis

May Points-to



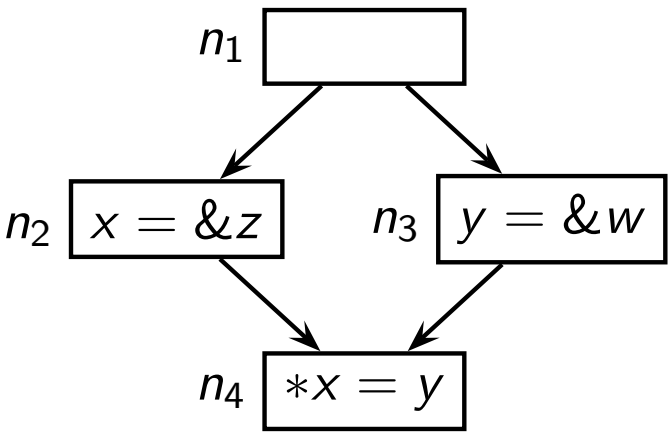

$z \rightarrowtail w$ is spurious

Must Points-to



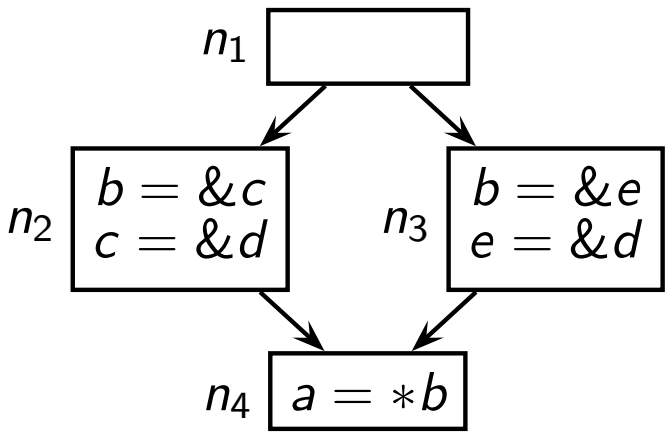
# Non-Distributivity of Points-to Analysis

May Points-to




$z \rightarrowtail w$ is spurious

Must Points-to



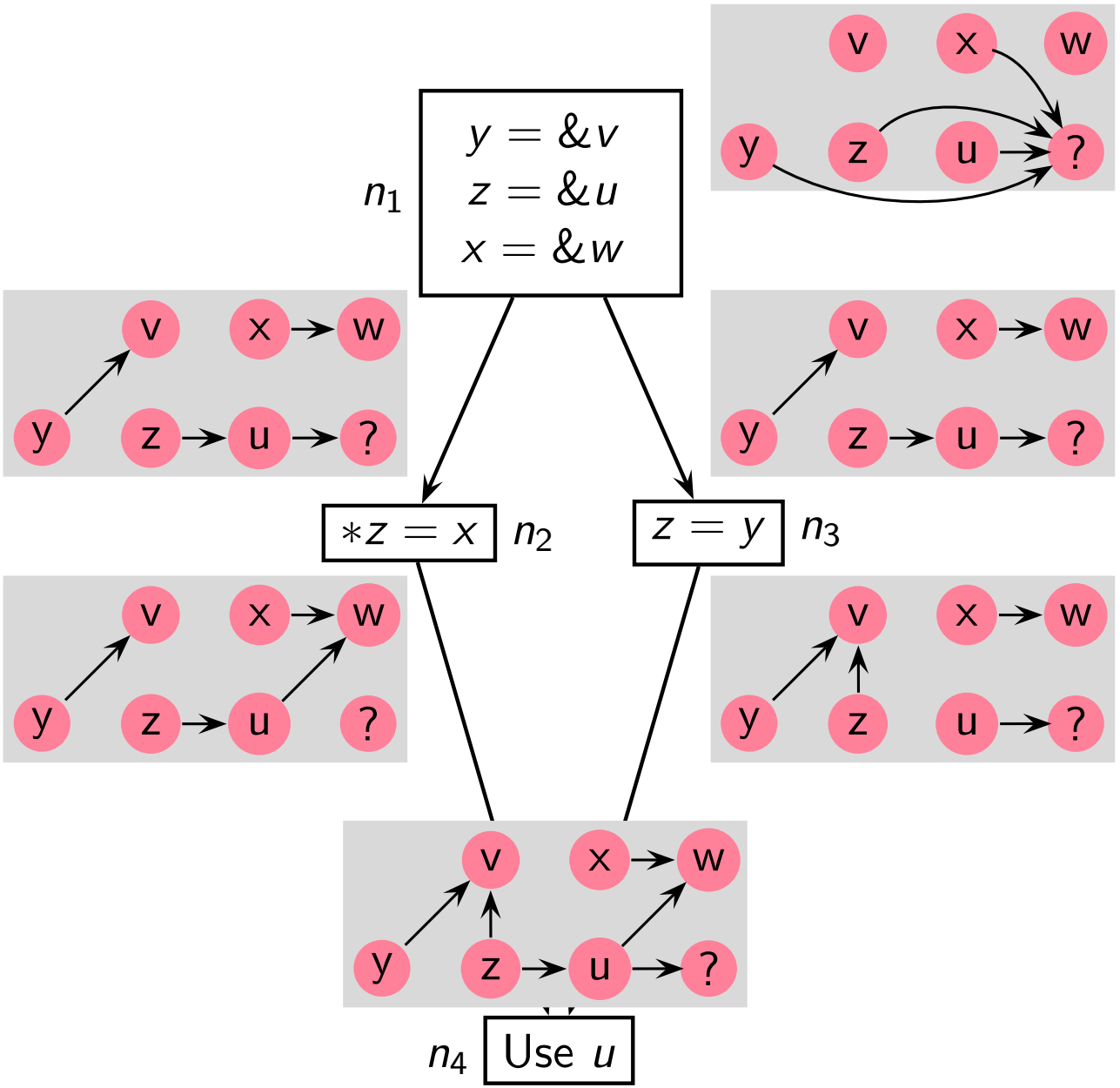

$a \rightarrowtail d$ is missing

# An Outline of Pointer Analysis Coverage

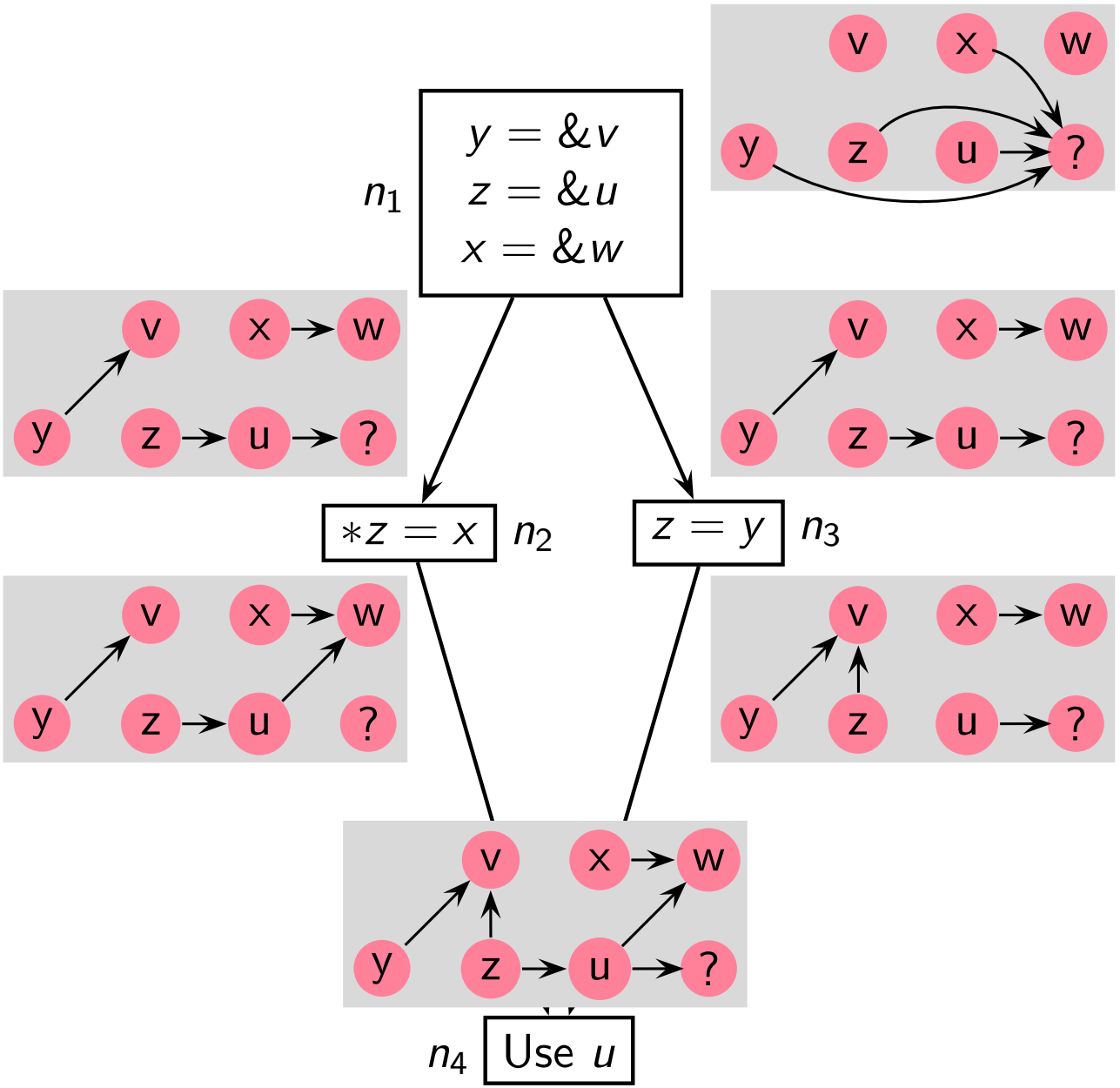
- Pointer Statements
- Comparing Points-to and Alias information
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis Next Topic
- Handling Heap



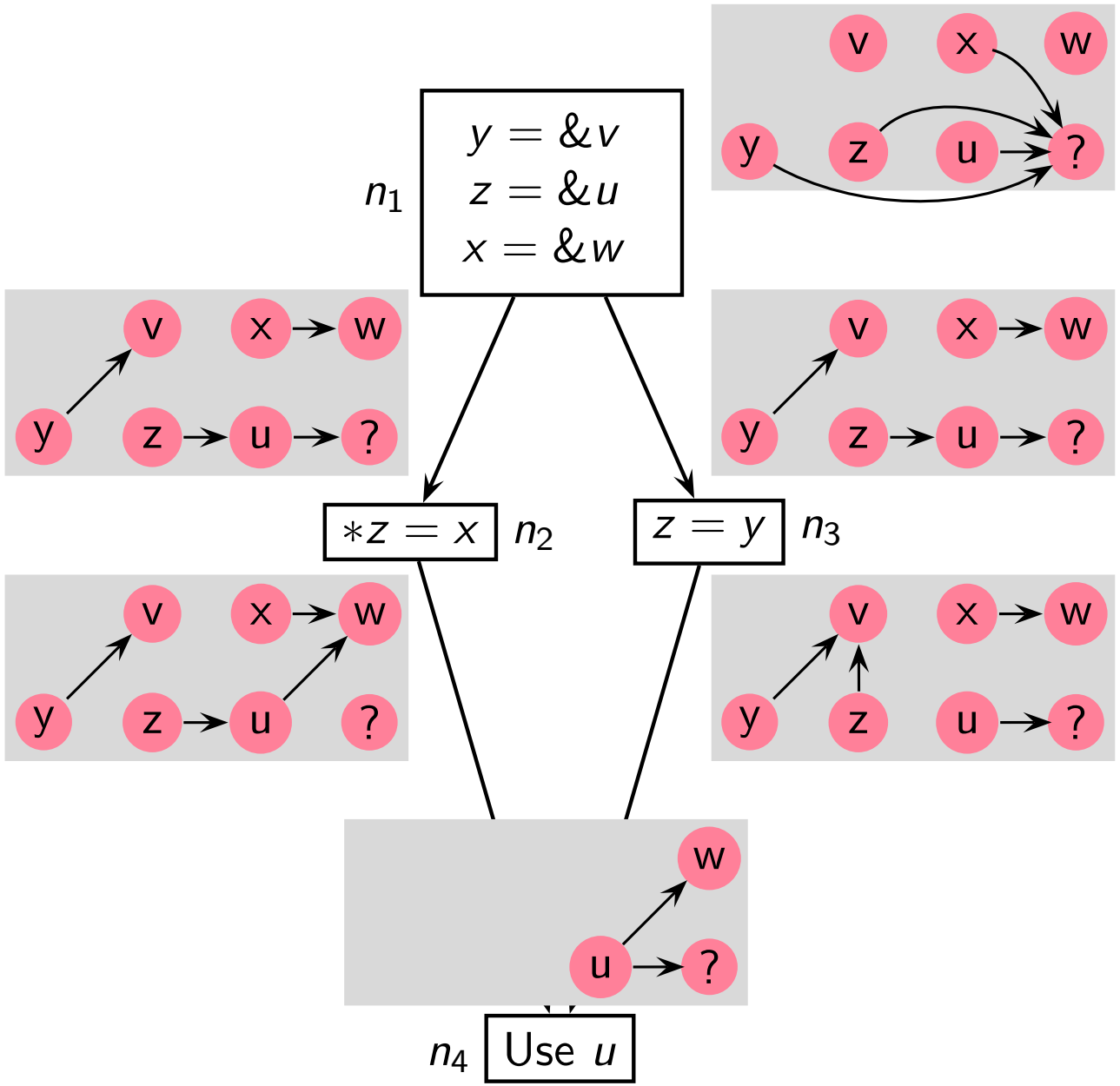
# Our Motivating Example for FCPA



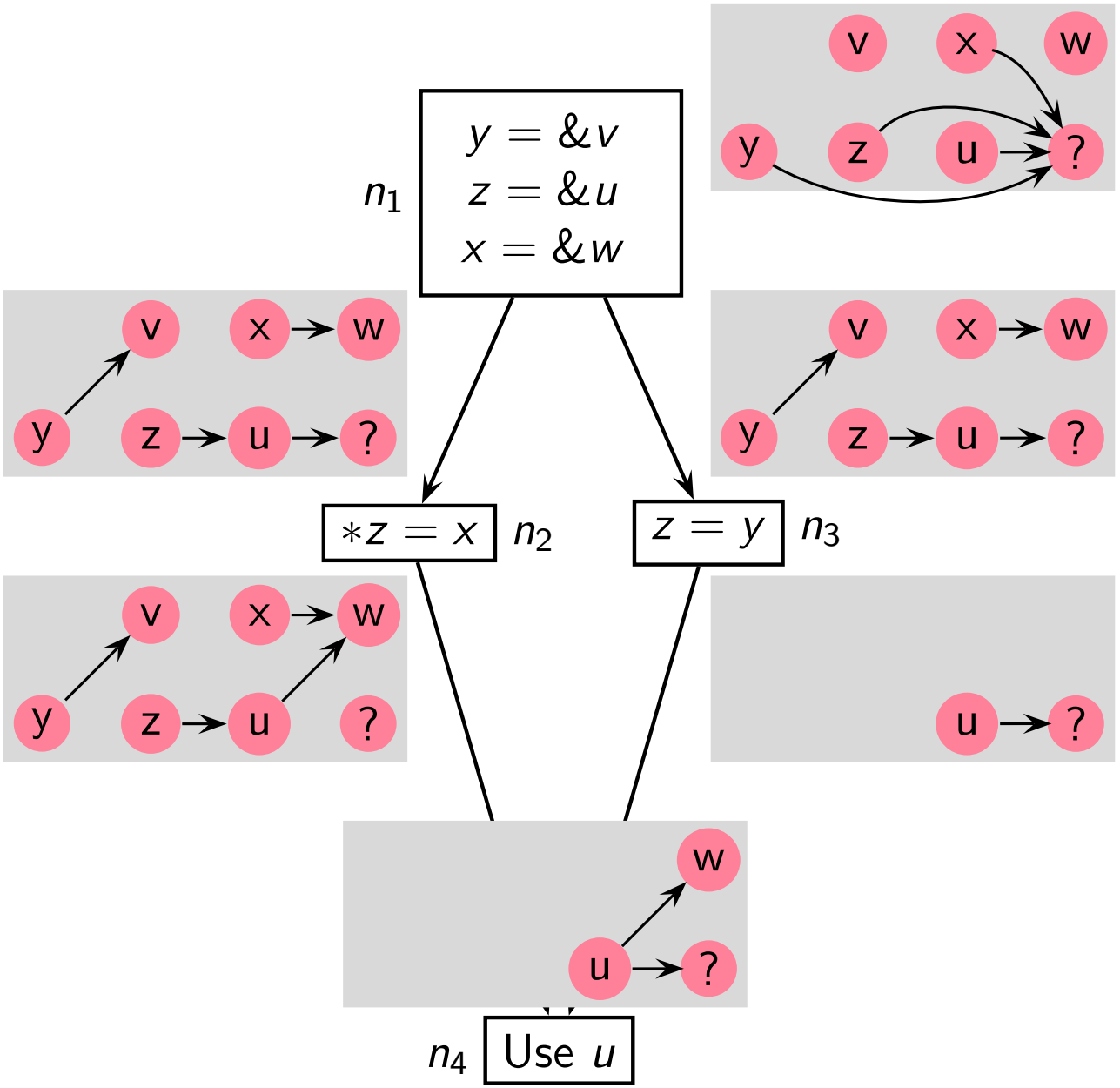
# Is All This Information Useful?



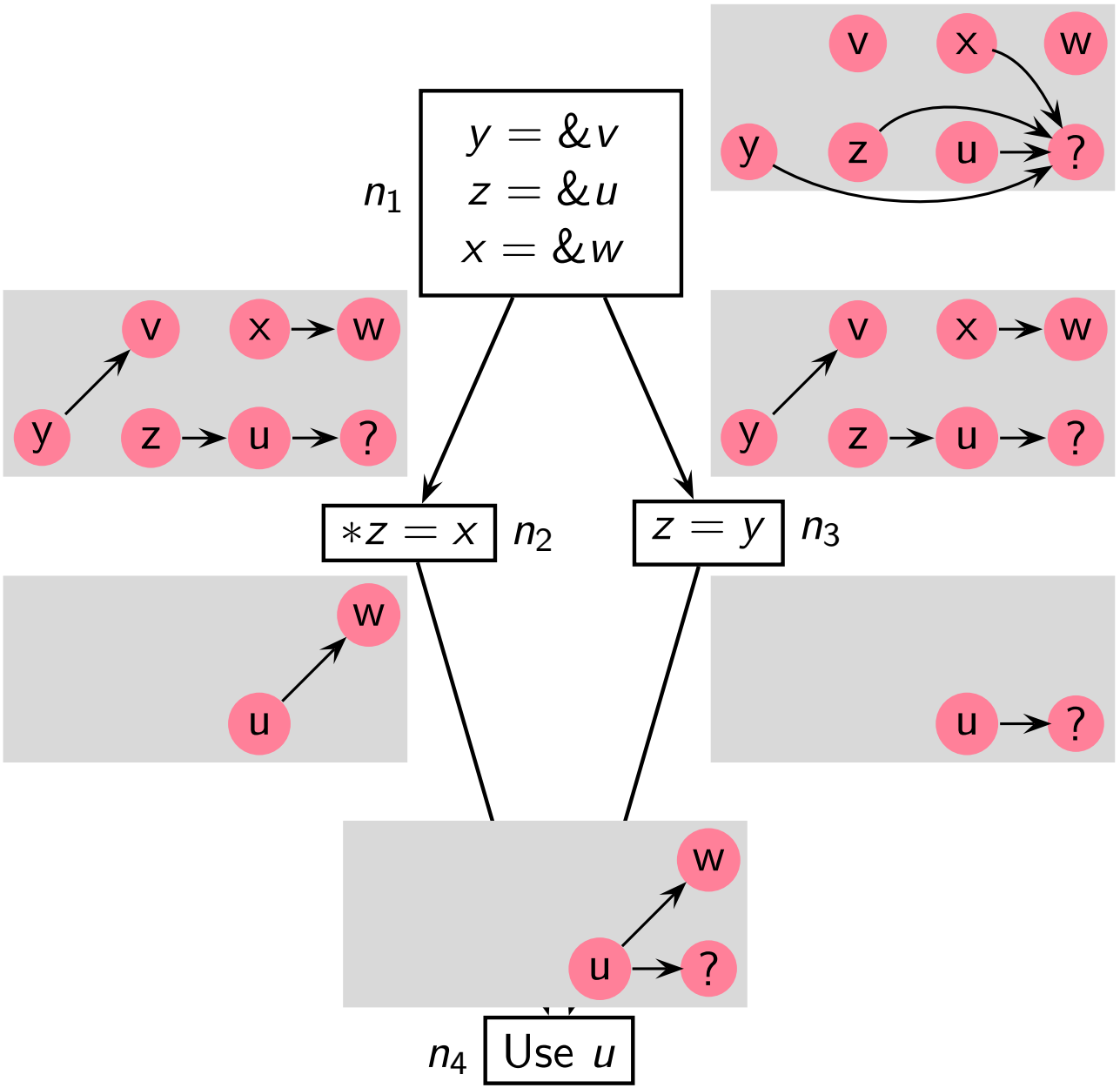
# Is All This Information Useful?



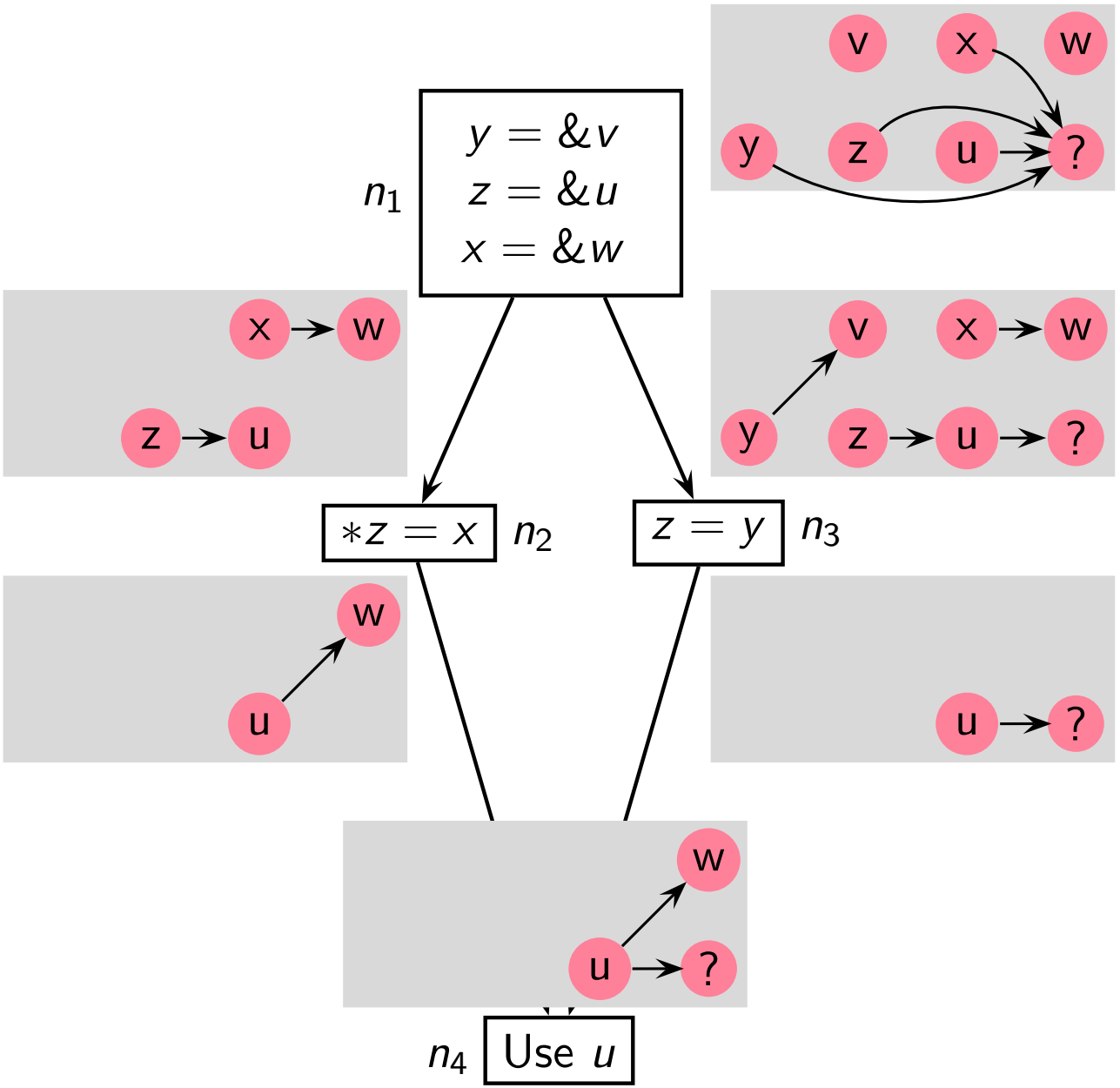
# Is All This Information Useful?



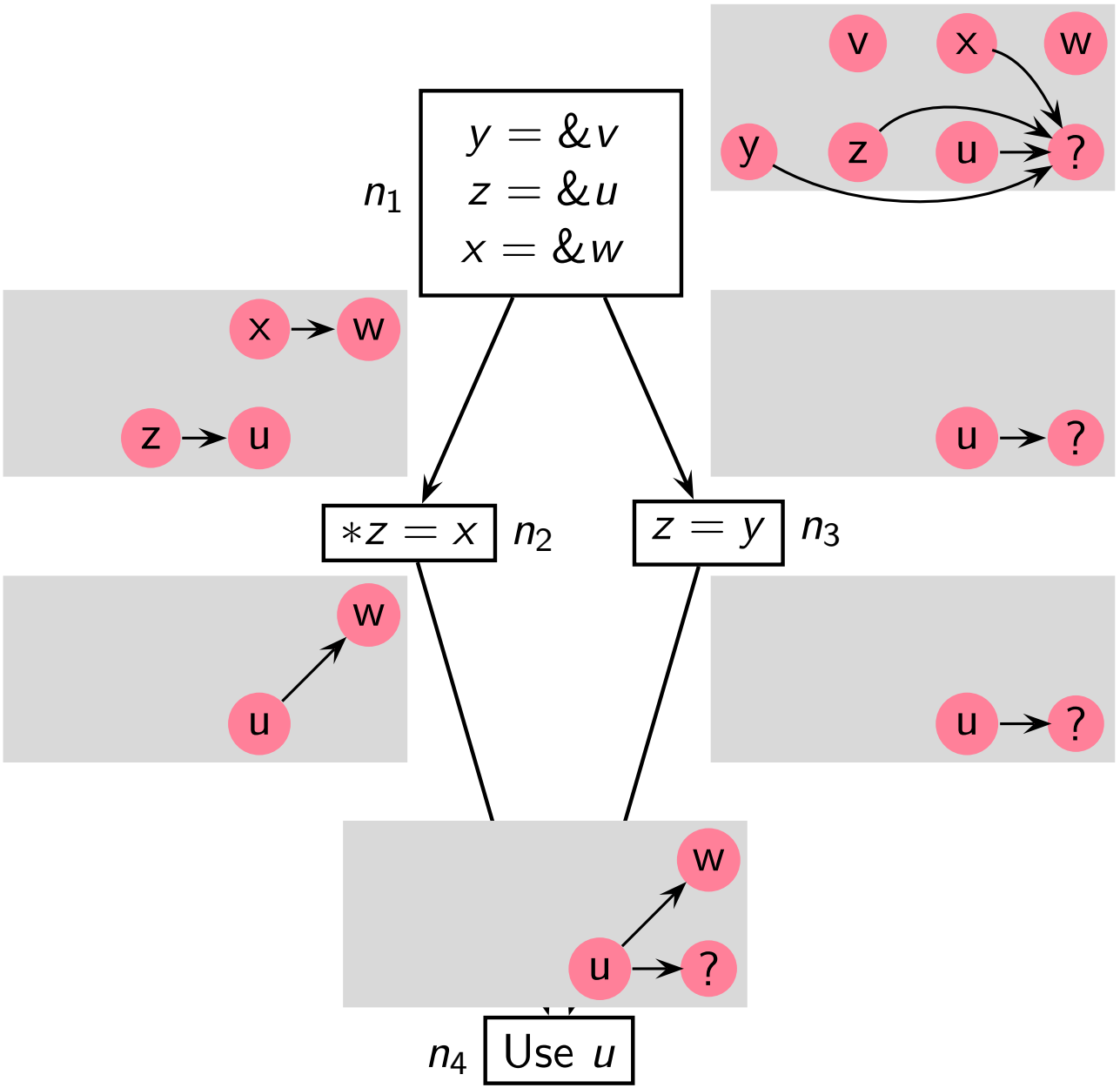
# Is All This Information Useful?



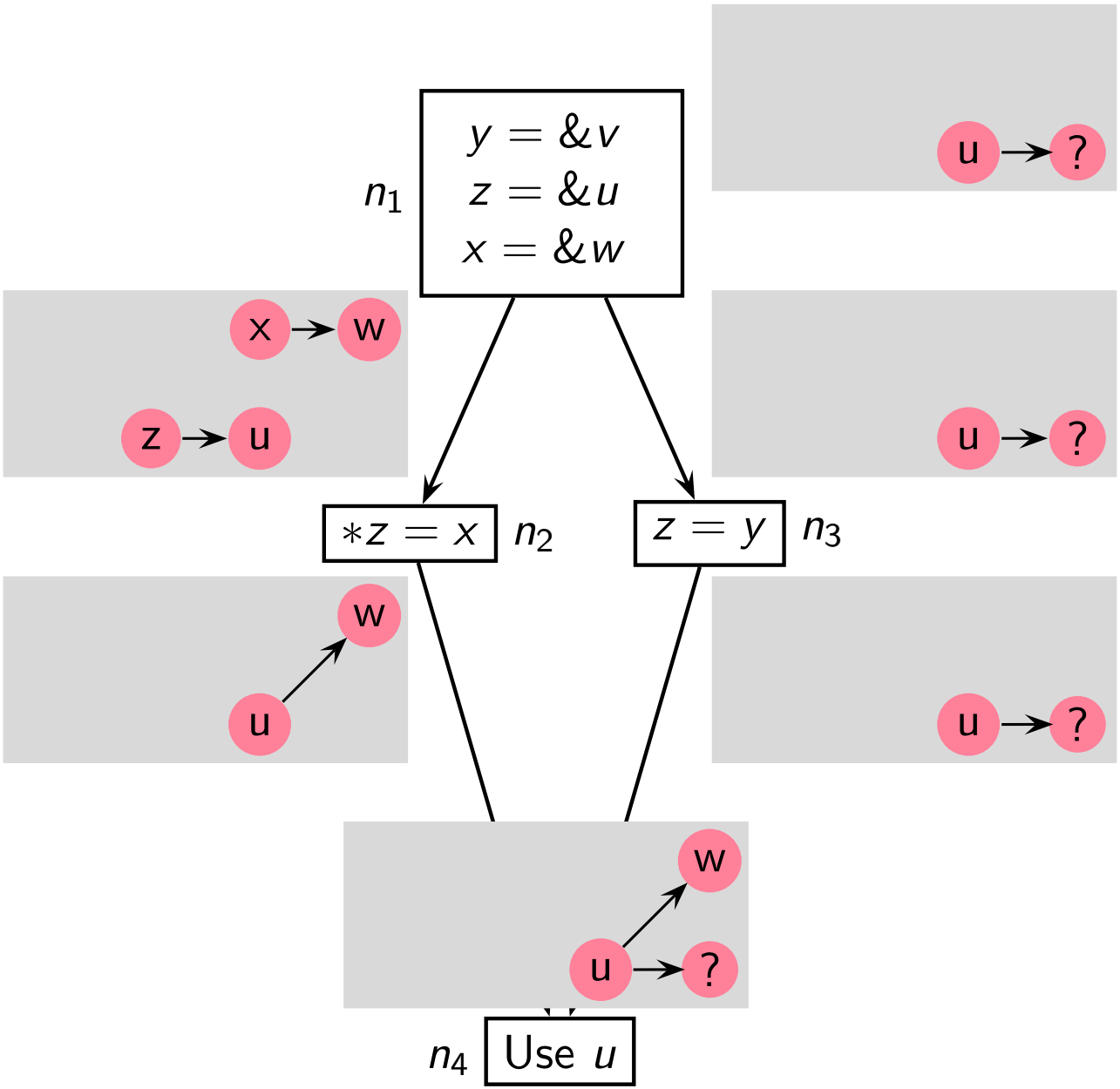
# Is All This Information Useful?



# Is All This Information Useful?



# Is All This Information Useful?



# Liveness-Based Ideal Points-to Analysis

- Let $(n, \sigma)_i \in \tau$, $i \geq 0$, denote the occurrence of program state $(n, \sigma)$ at position $i$ in trace $\tau$
- The liveness-based restriction of $\tau$, denoted $\mathrm{Lv}(\tau)$, is computed by replacing $(n, \sigma)_i \in \tau$ by $(n, \sigma')_i$ where $\sigma' \subseteq \sigma$.

  The state $\sigma'$ is computed by retaining only those $((x, f) \mapsto a) \in \sigma$ for which $(x, f)$ is read subsequently in $\tau$ before being defined

$$\Big(\exists\, j > i \text{ such that } (m, \cdot)_j \in \tau \wedge (x, f) \in \mathit{Ref}(m) \bigwedge$$
$$\big(\nexists\, k, i < k < j \text{ such that } (l, \cdot)_k \in \tau \wedge (x, f) \in \mathit{Def}(l)\big)\Big)$$

  - $\mathit{Def}(n)$ is same as the extractor function $\mathit{Def}_n$ defined before (except that now we use a state $\sigma$ along $\tau$ rather than the points-to information $\mathit{Ain}_n$)
  - $\mathit{Ref}(n)$ will be defined shortly



# Liveness-Based Ideal Points-to Analysis

- Liveness-Based Ideal May-Points-to analysis computes Points-to information reaching along the liveness-based restriction of all traces

$$RLS(n) = \{\sigma \mid (n, \sigma) \text{ occurs in some trace Lv}(\tau)\}$$

$$\text{LvIdealMayPT}(n) = \bigcup_{\sigma \in RLS(n)} \sigma$$

- It is easy to see that

$$\forall n \in N : \quad \text{LvIdealMayPT}(n) \subseteq \text{IdealMayPT}(n)$$

- We redefine the soundness criterion in terms of LvIdealMayPT



# The L and P of LFCPA

Mutual dependence of liveness and points-to information

- Define points-to information only for live pointers
- For pointer indirections, define liveness information using points-to information

# The F and C of LFCPA

- Use call strings method for full flow and context sensitivity
- Use value contexts for efficient interprocedural analysis [Khedker-Karkare-CC-2008, Padhye-Khedker-SOAP-2013]



# Use of Strong Liveness

- Simple liveness considers every use of a variable as useful
- Strong liveness checks the liveness of the result before declaring the operands to be live



# Use of Strong Liveness

- Simple liveness considers every use of a variable as useful
- Strong liveness checks the liveness of the result before declaring the operands to be live
- Strong liveness is more precise than simple liveness



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | | |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs
- $Ref_n$, $Kill_n$, $Def_n$, and $Pointee_n$ are defined in terms of $Ain_n$

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointe_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | | |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

Pointers that become live



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | | |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

Defined pointers must be live at the exit for the read pointers to become live



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | | |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

Some pointers are unconditionally live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | | |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

$x$ is unconditionally live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\boxed{\{y\}}$ | |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

$y$ is live if defined pointers are live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | $\{y\} \cup A\{y\} \cap \mathbf{P}$ | |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

$y$ and its pointees in $Ain_n$ are live if defined pointers are live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | $\{y\} \cup A\{y\} \cap \mathbf{P}$ | $\emptyset$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |



# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | $\{y\} \cup A\{y\} \cap \mathbf{P}$ | $\emptyset$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | $\{x, y\}$ | |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

$y$ is live if defined pointers are live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* — *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | $\{y\} \cup A\{y\} \cap \mathbf{P}$ | $\emptyset$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | $\{x, y\}$ | $\{x\}$ |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |

$x$ is unconditionally live

# Extractor Functions for LFCPA

*Unchanged from earlier points-to analysis* *Generation of strong liveness*

| | $Def_n$ | $Kill_n$ | $Pointee_n$ | $Ref_n$ | |
|---|---|---|---|---|---|
| | | | | $Def_n \cap Lout_n \neq \emptyset$ | *otherwise* |
| *use* $x$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{x\}$ | $\{x\}$ |
| $x = \&a$ | $\{x\}$ | $\{x\}$ | $\{a\}$ | $\emptyset$ | $\emptyset$ |
| $x = y$ | $\{x\}$ | $\{x\}$ | $A\{y\}$ | $\{y\}$ | $\emptyset$ |
| $x = *y$ | $\{x\}$ | $\{x\}$ | $A(A\{y\} \cap \mathbf{P})$ | $\{y\} \cup A\{y\} \cap \mathbf{P}$ | $\emptyset$ |
| $*x = y$ | $A\{x\} \cap \mathbf{P}$ | $Must(A)\{x\} \cap \mathbf{P}$ | $A\{y\}$ | $\{x, y\}$ | $\{x\}$ |
| other | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |



# Deriving *Must* Points-to for LFCPA

For $*x = y$, unless the pointees of $x$ are known

- points-to propagation should be blocked
- liveness propagation should be blocked

to ensure monotonicity

$$Must(R) = \bigcup_{x \in \mathbf{P}} \{x\} \times \begin{cases} V & R\{x\} = \emptyset \vee R\{x\} = \{?\} \\ \{y\} & R\{x\} = \{y\} \wedge y \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$



# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left( Lout_n - Kill_n \right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left( Ain_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times Pointee_n \right) \right) \Big|_{Lout_n}$$

- *Lin*/*Lout*: set of Live pointers
- *Ain*/*Aout*: definitions remain unchanged except for restriction to liveness



# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$Kill_n$ defined in terms of $Ain_n$

$$Lin_n = \left(Lout_n - Kill_n\right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left(Ain_n - \left(Kill_n \times V\right)\right) \cup \left(Def_n \times Pointee_n\right) \right) \Big|_{Lout_n}$$

- $Lin/Lout$: set of Live pointers
- $Ain/Aout$: definitions remain unchanged except for restriction to liveness



# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left( Lout_n - Kill_n \right) \cup \boxed{Ref_n}$$

$Ref_n$ defined in terms of $Ain_n$ and $Lout_n$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left( Ain_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times Pointee_n \right) \right) \Big|_{Lout_n}$$

- $Lin/Lout$: set of Live pointers
- $Ain/Aout$: definitions remain unchanged except for restriction to liveness



# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left( Lout_n - Kill_n \right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left( Ain_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times Pointee_n \right) \right) \Big|_{Lout_n}$$

$Ain_n$ and $Aout_n$ are restricted to $Lin_n$ and $Lout_n$

- $Lin/Lout$: set of Live pointers
- $Ain/Aout$: definitions remain unchanged except for restriction to liveness

# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left( Lout_n - Kill_n \right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left( Ain_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times Pointee_n \right) \right) \Big|_{Lout_n}$$

*BI* restricted to live pointers

- *Lin*/*Lout*: set of Live pointers
- *Ain*/*Aout*: definitions remain unchanged except for restriction to liveness



# LFCPA Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left( Lout_n - Kill_n \right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left( \bigcup_{p \in pred(n)} Aout_p \right) \Big|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left( \left( Ain_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times Pointee_n \right) \right) \Big|_{Lout_n}$$

- *Lin*/*Lout*: set of Live pointers
- *Ain*/*Aout*: definitions remain unchanged except for restriction to liveness

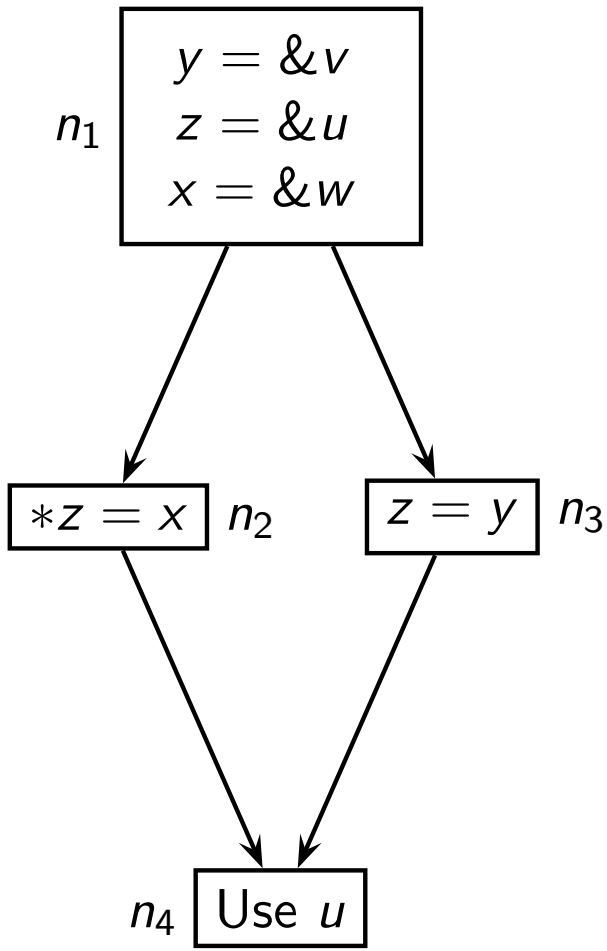


# Motivating Example Revisited

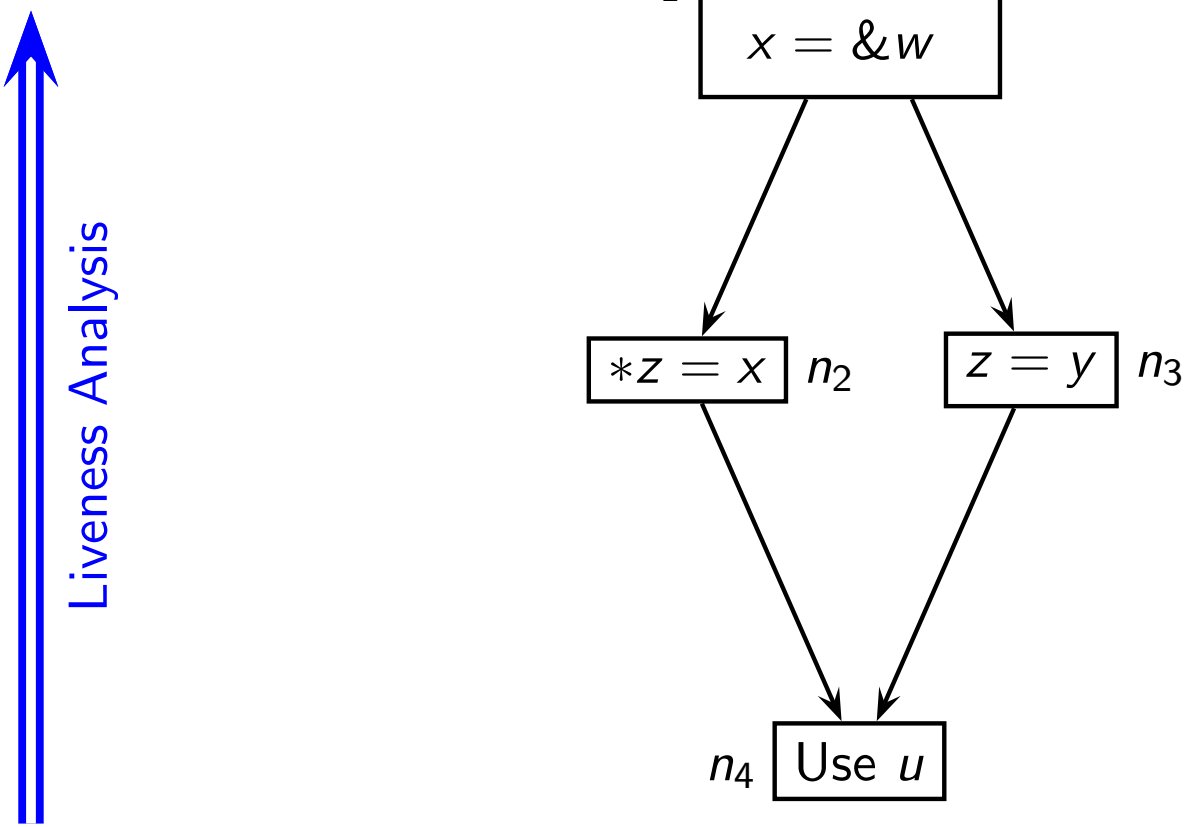
- For convenience, we show complete sweeps of liveness and points-to analysis repeatedly
- This is not required by the computation
- The data flow equations define a single fixed point computation



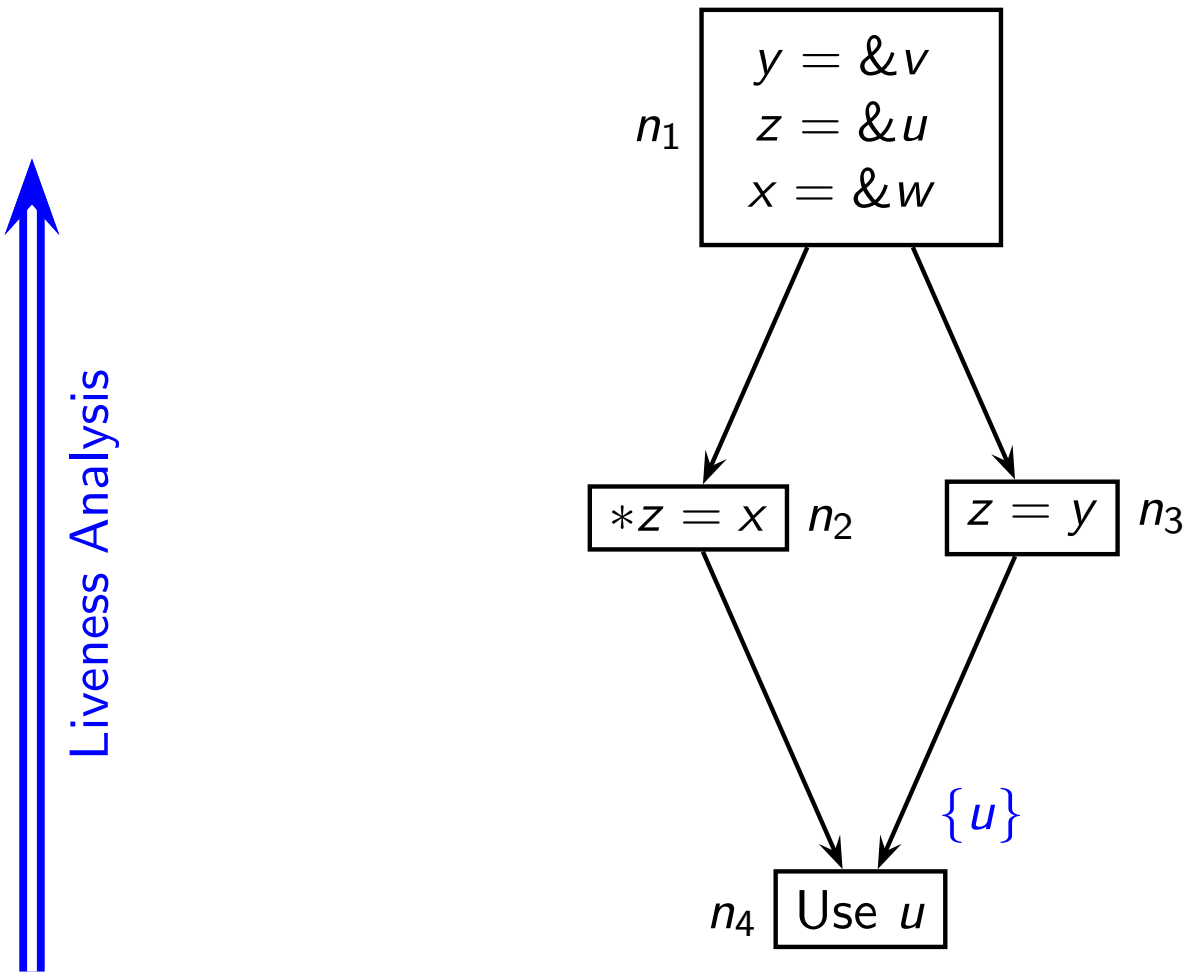
# First Round of Liveness Analysis and Points-to Analysis



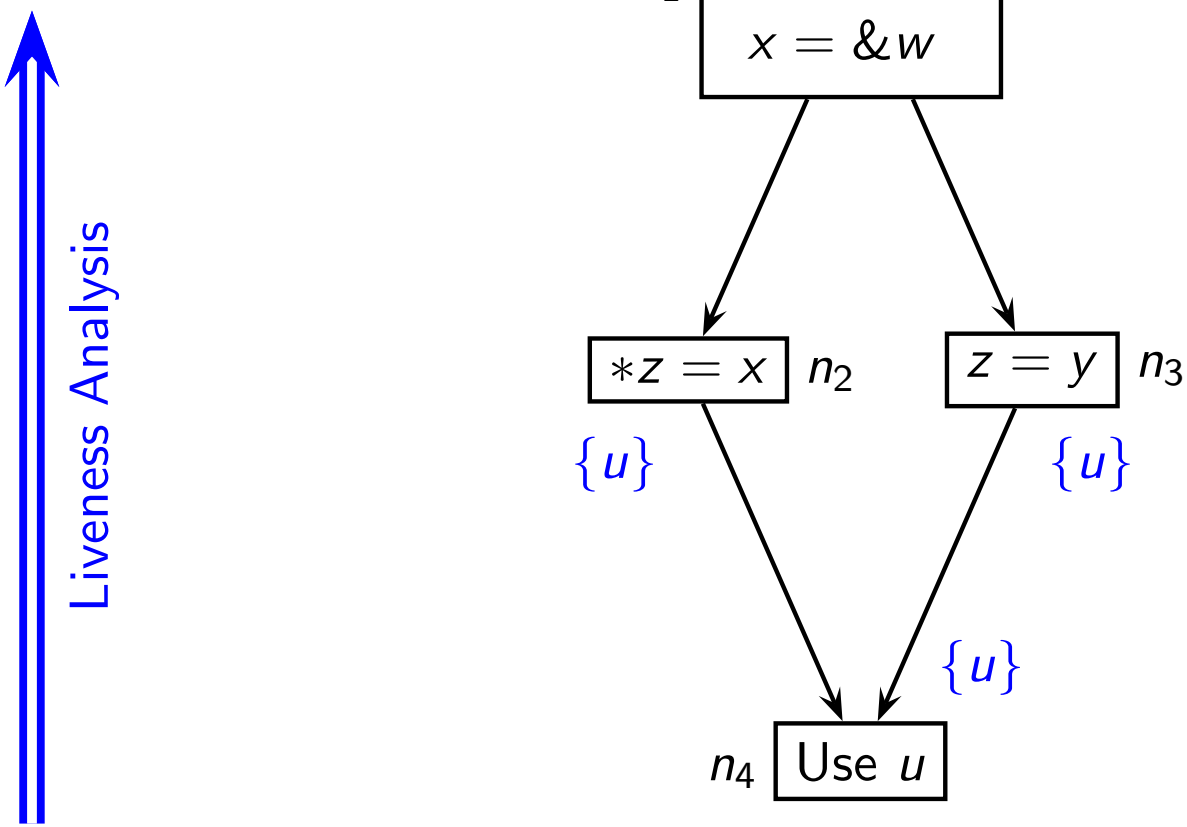
# First Round of Liveness Analysis and Points-to Analysis



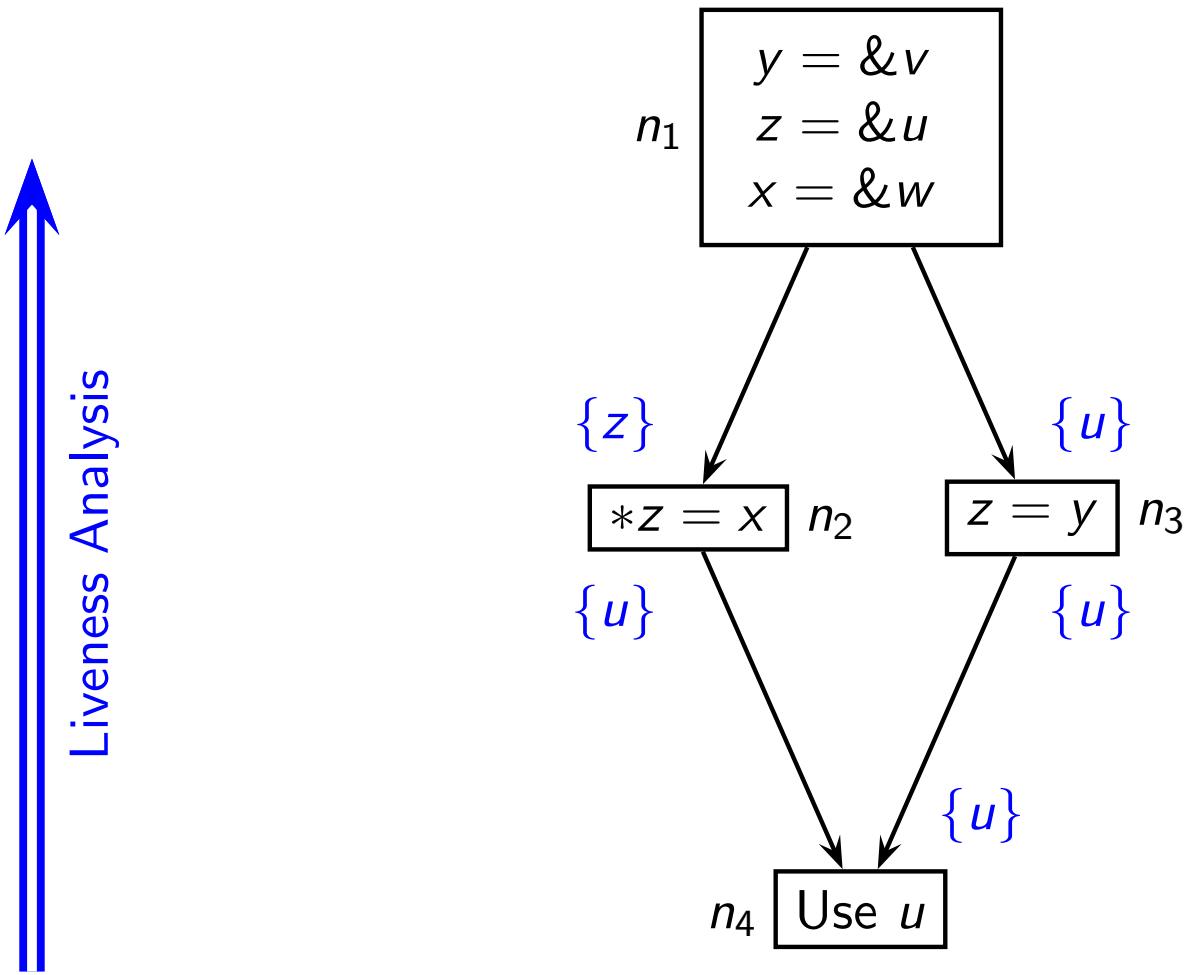
# First Round of Liveness Analysis and Points-to Analysis



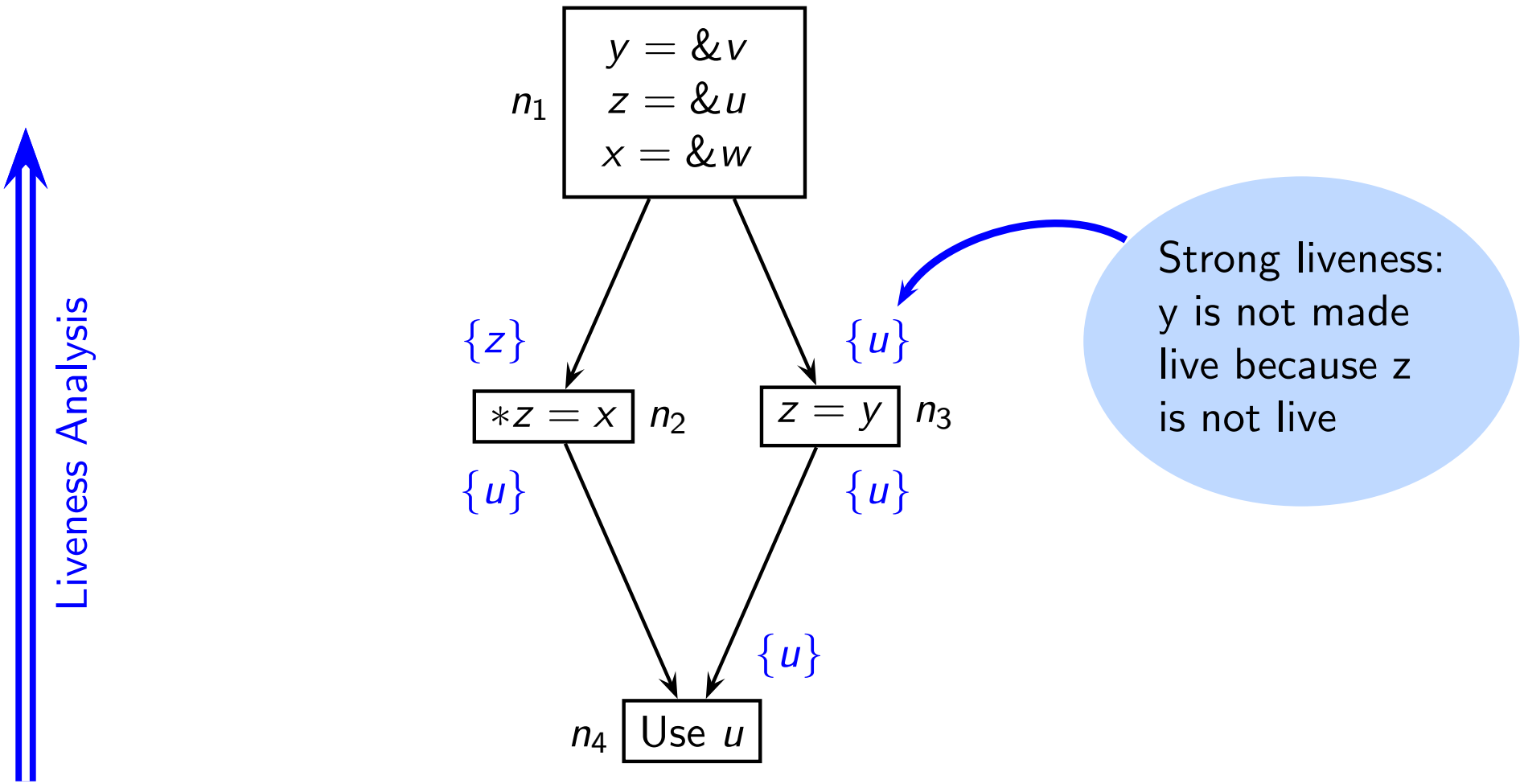
# First Round of Liveness Analysis and Points-to Analysis



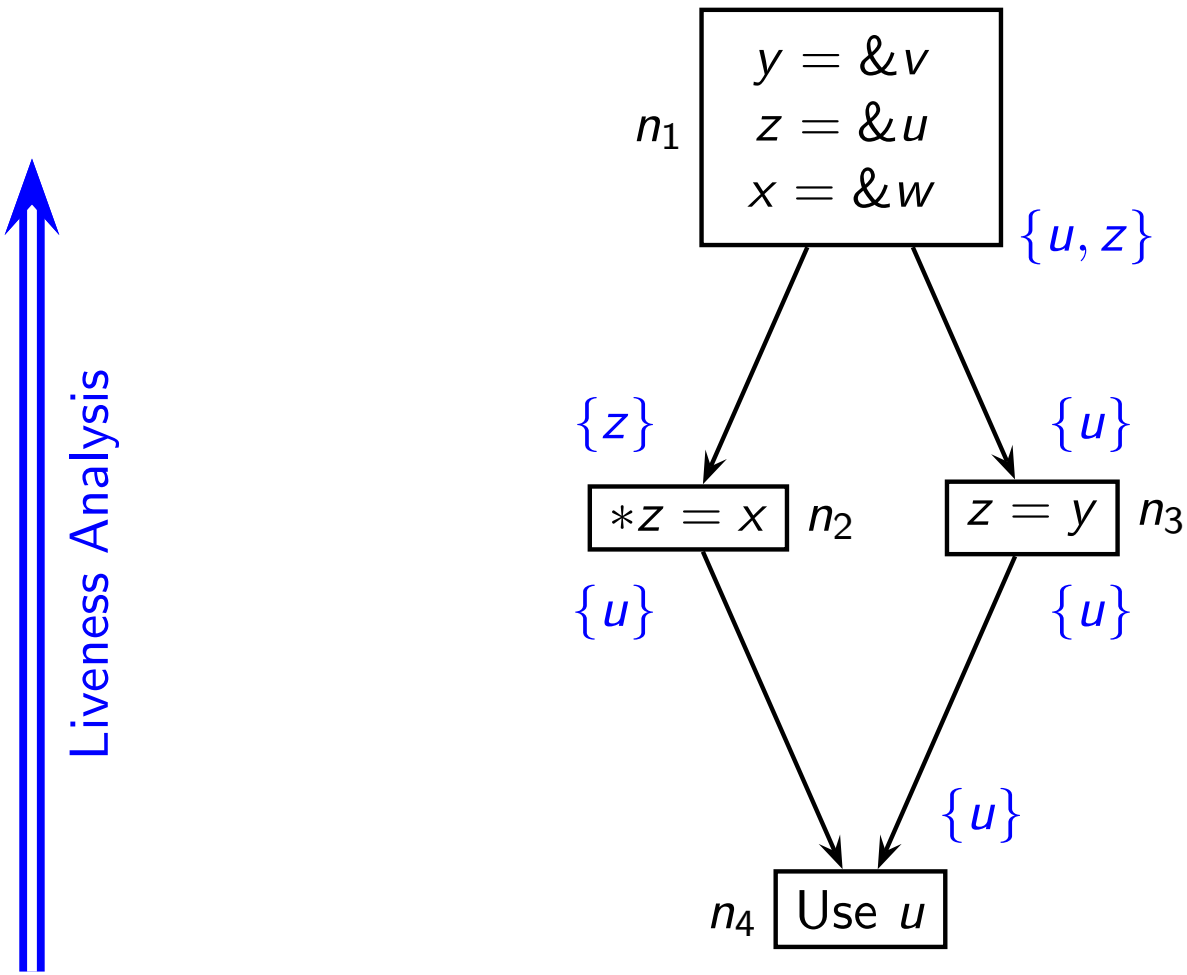
# First Round of Liveness Analysis and Points-to Analysis



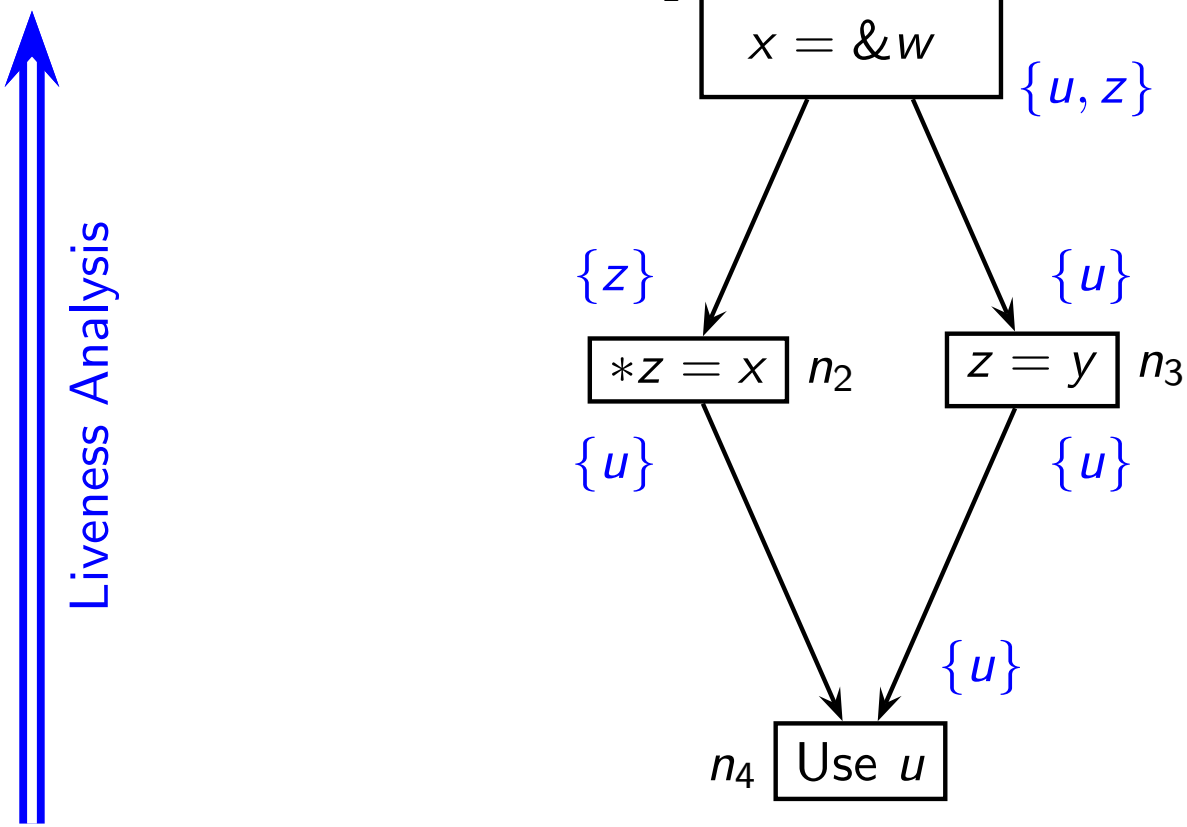
# First Round of Liveness Analysis and Points-to Analysis



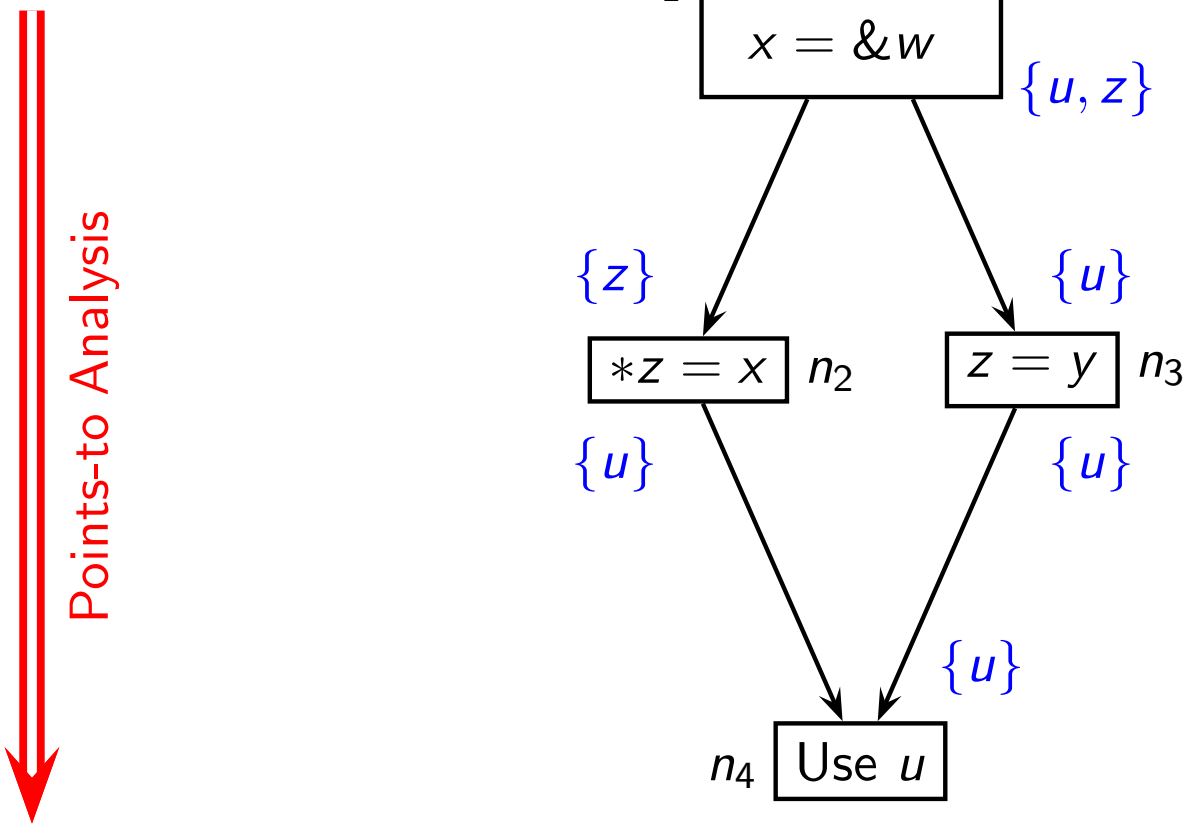
# First Round of Liveness Analysis and Points-to Analysis



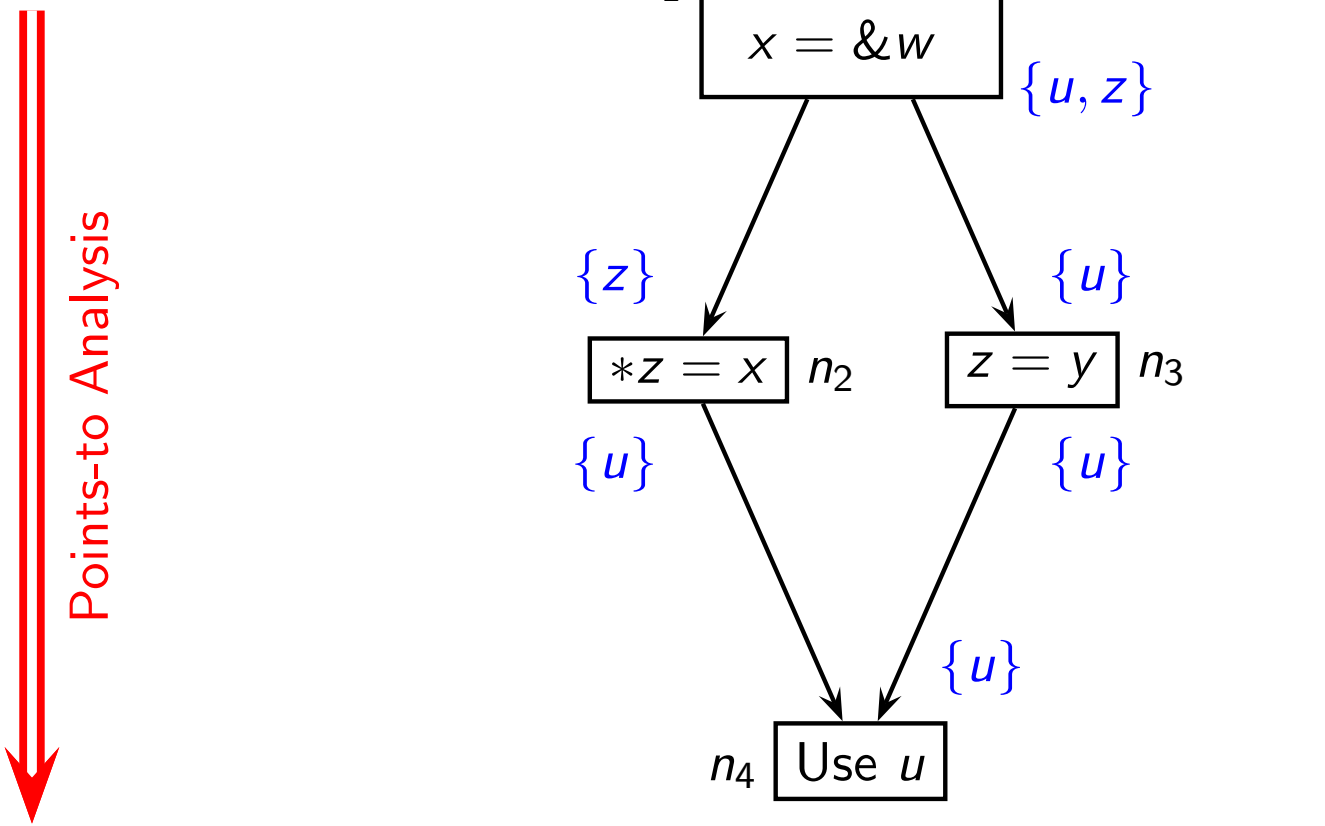
# First Round of Liveness Analysis and Points-to Analysis



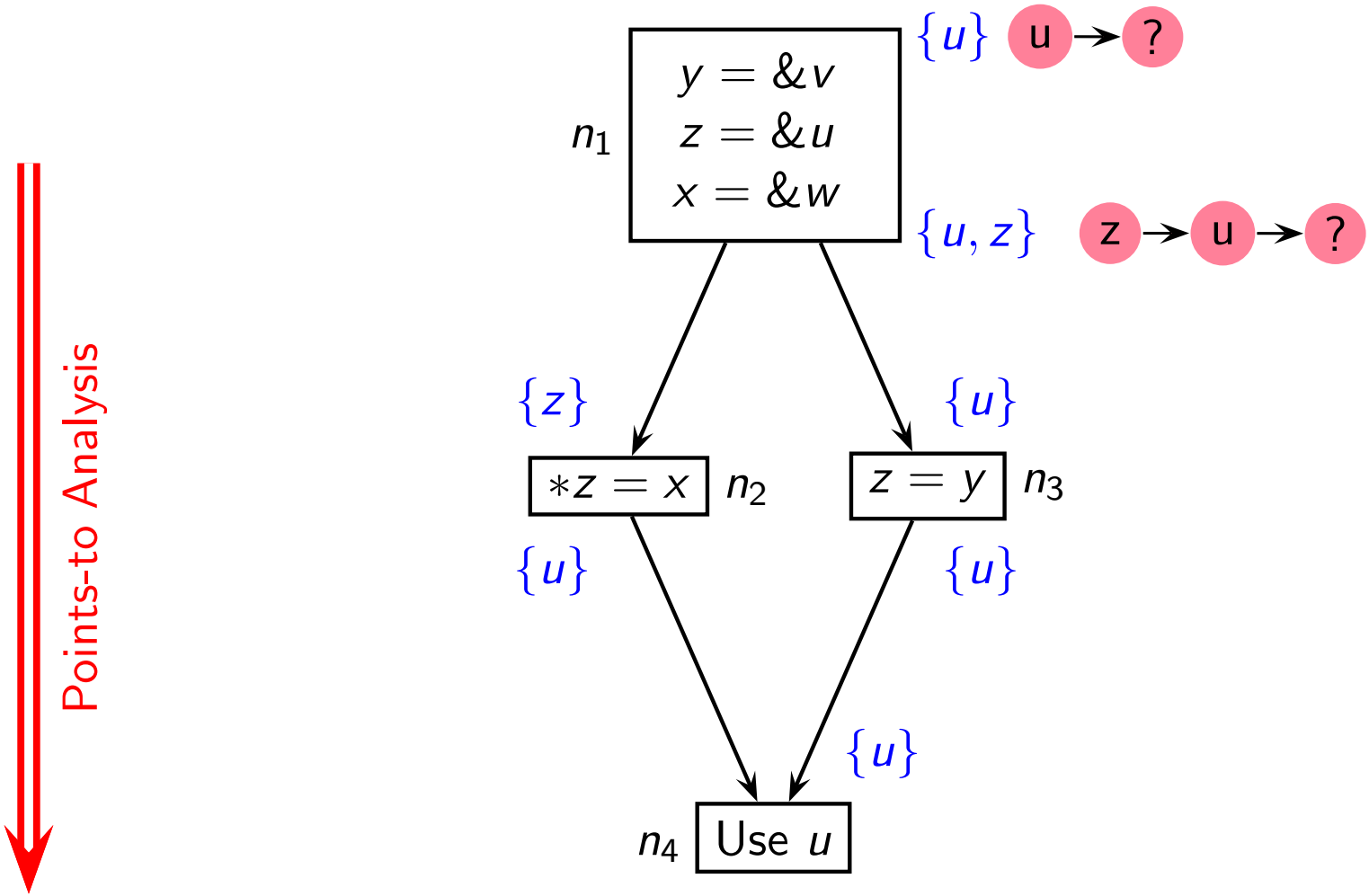
# First Round of Liveness Analysis and Points-to Analysis



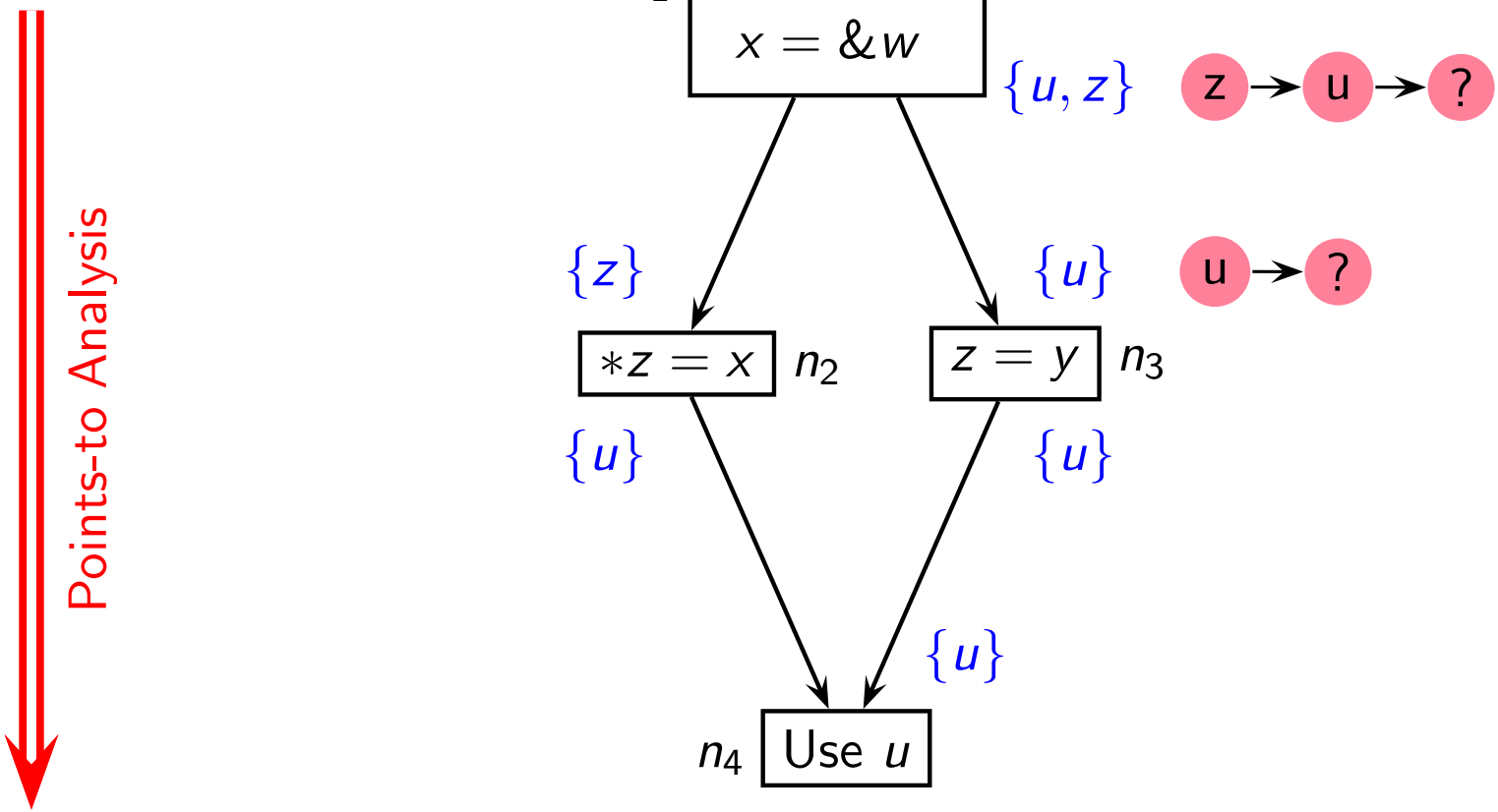
# First Round of Liveness Analysis and Points-to Analysis



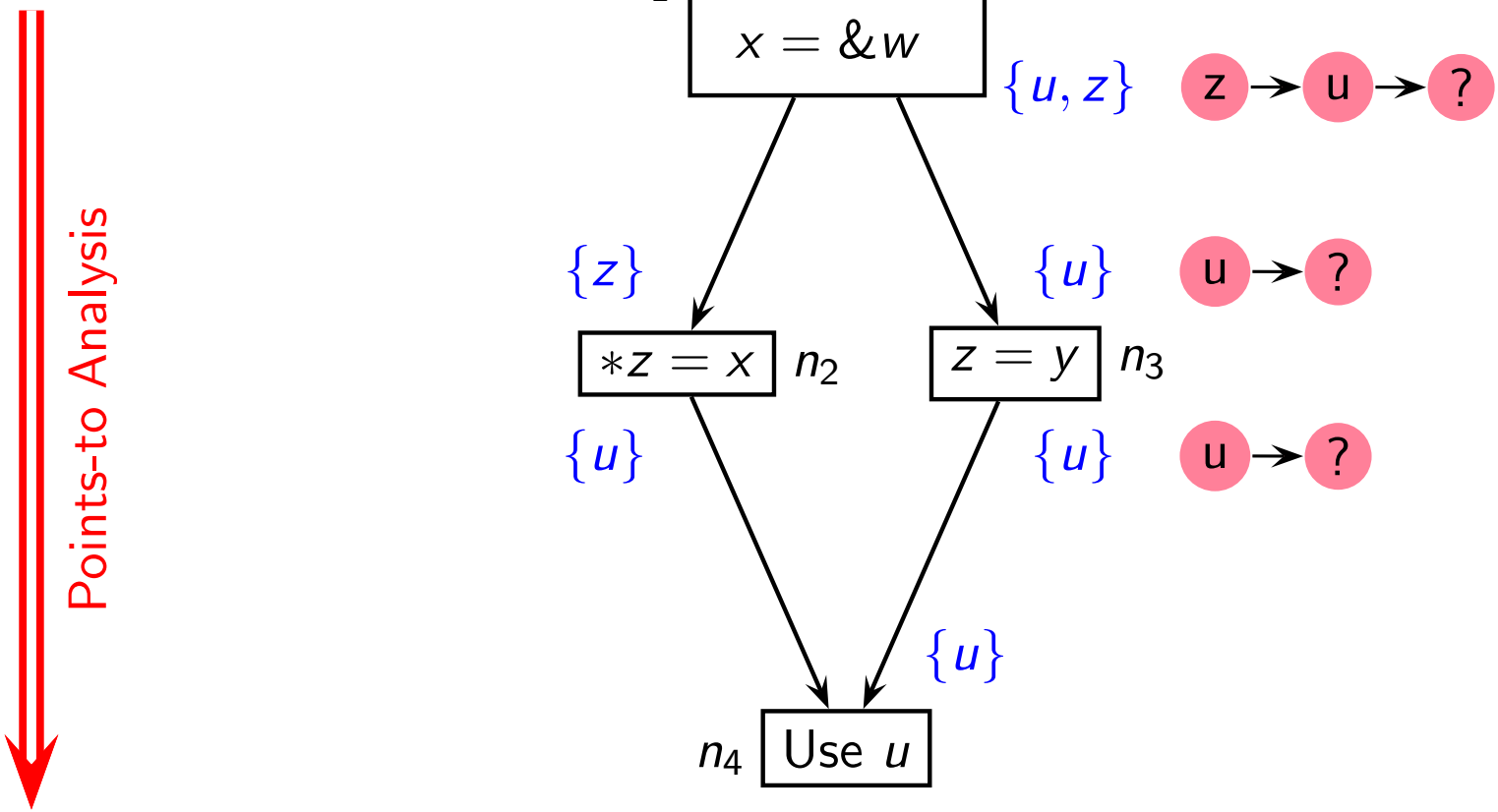
# First Round of Liveness Analysis and Points-to Analysis



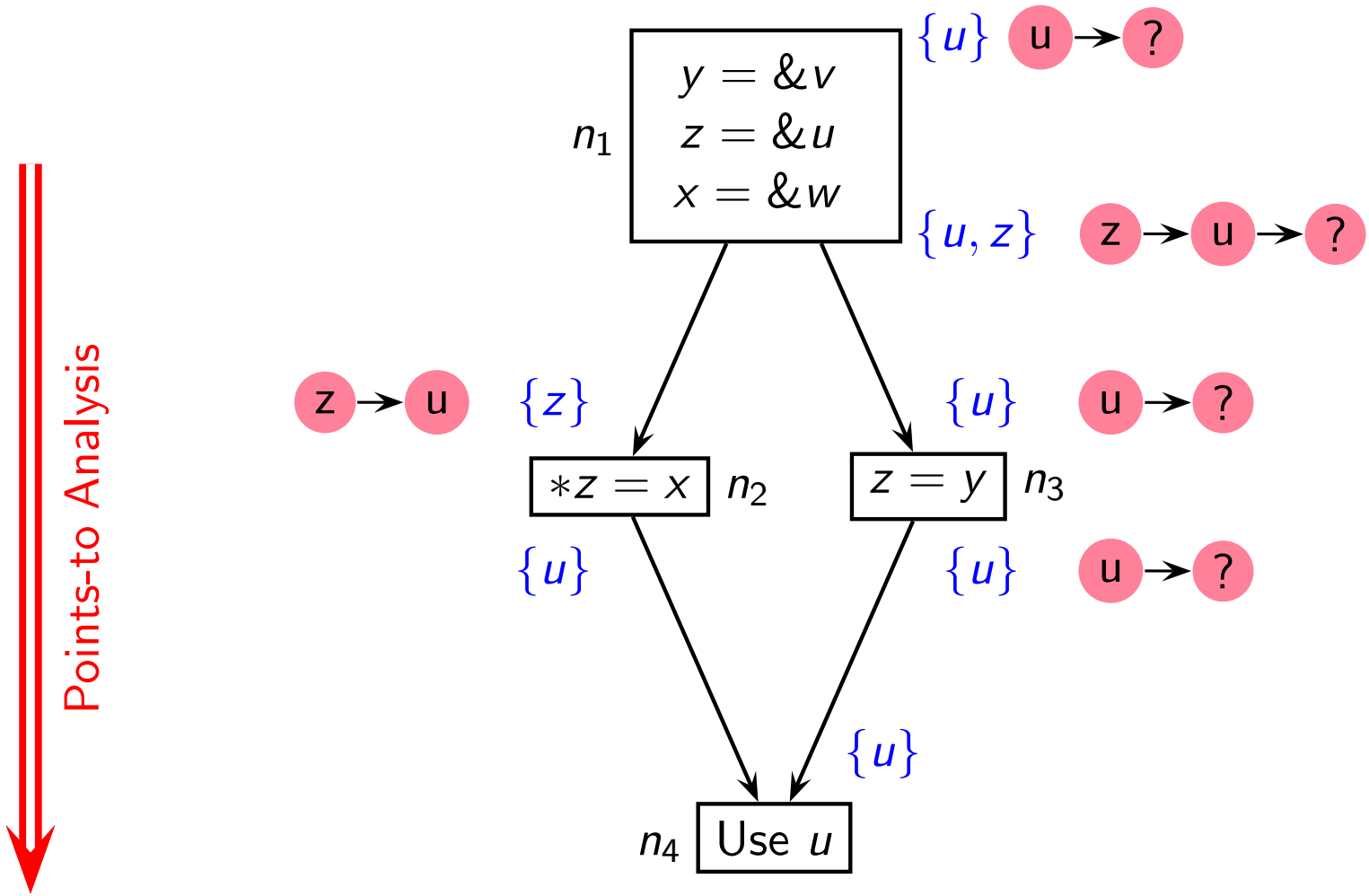
# First Round of Liveness Analysis and Points-to Analysis



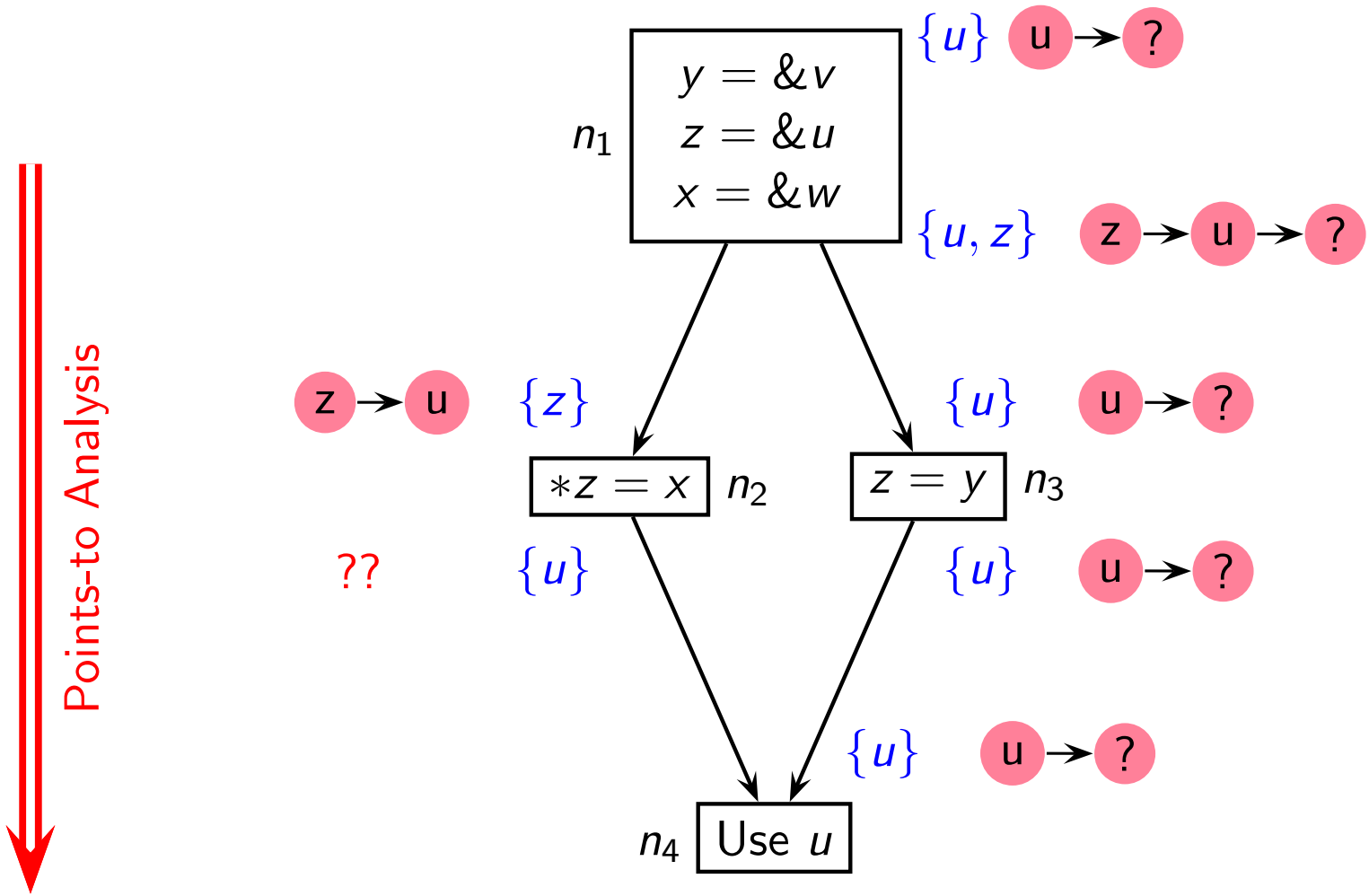
# First Round of Liveness Analysis and Points-to Analysis



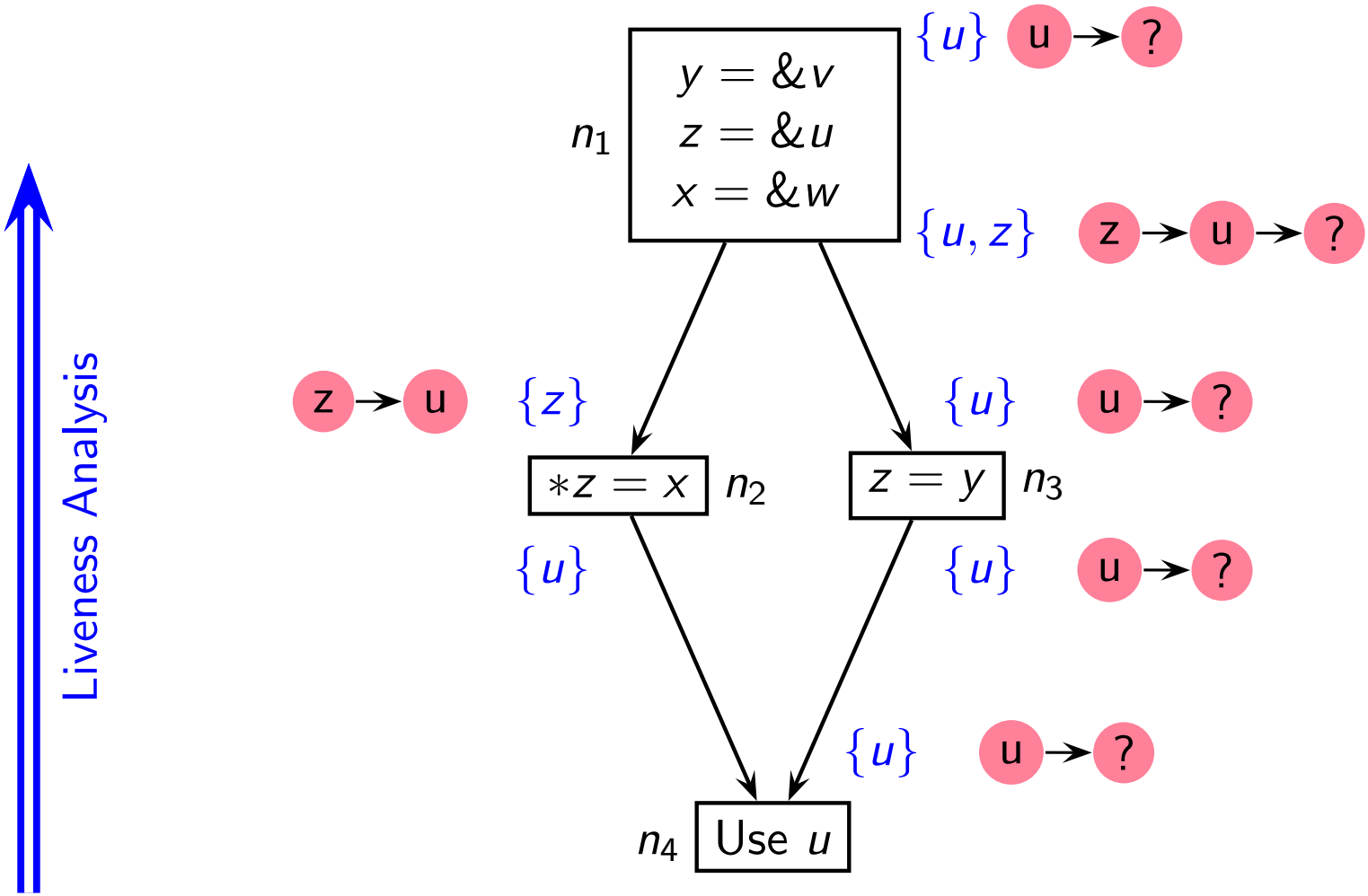
# First Round of Liveness Analysis and Points-to Analysis



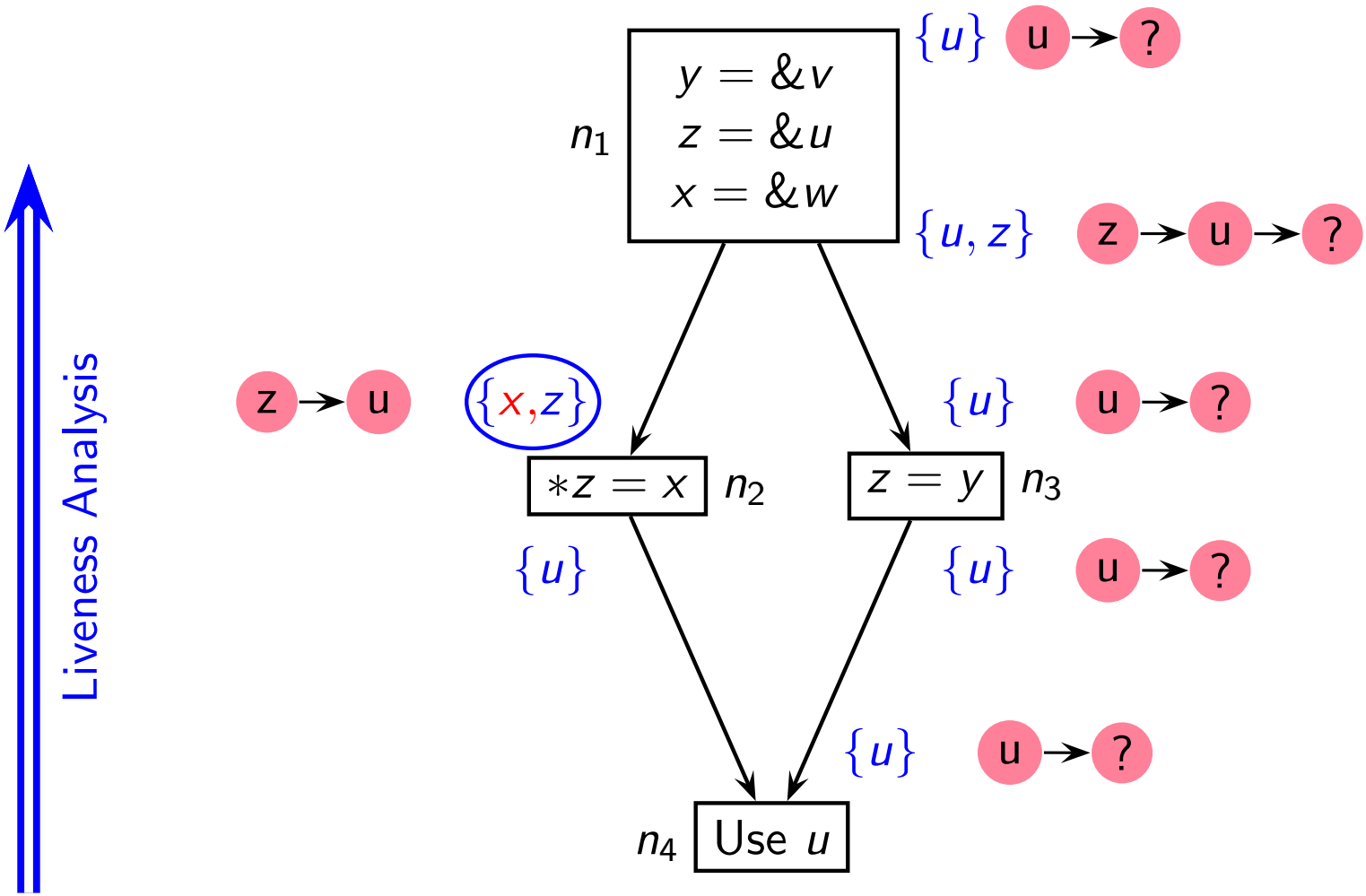
# First Round of Liveness Analysis and Points-to Analysis



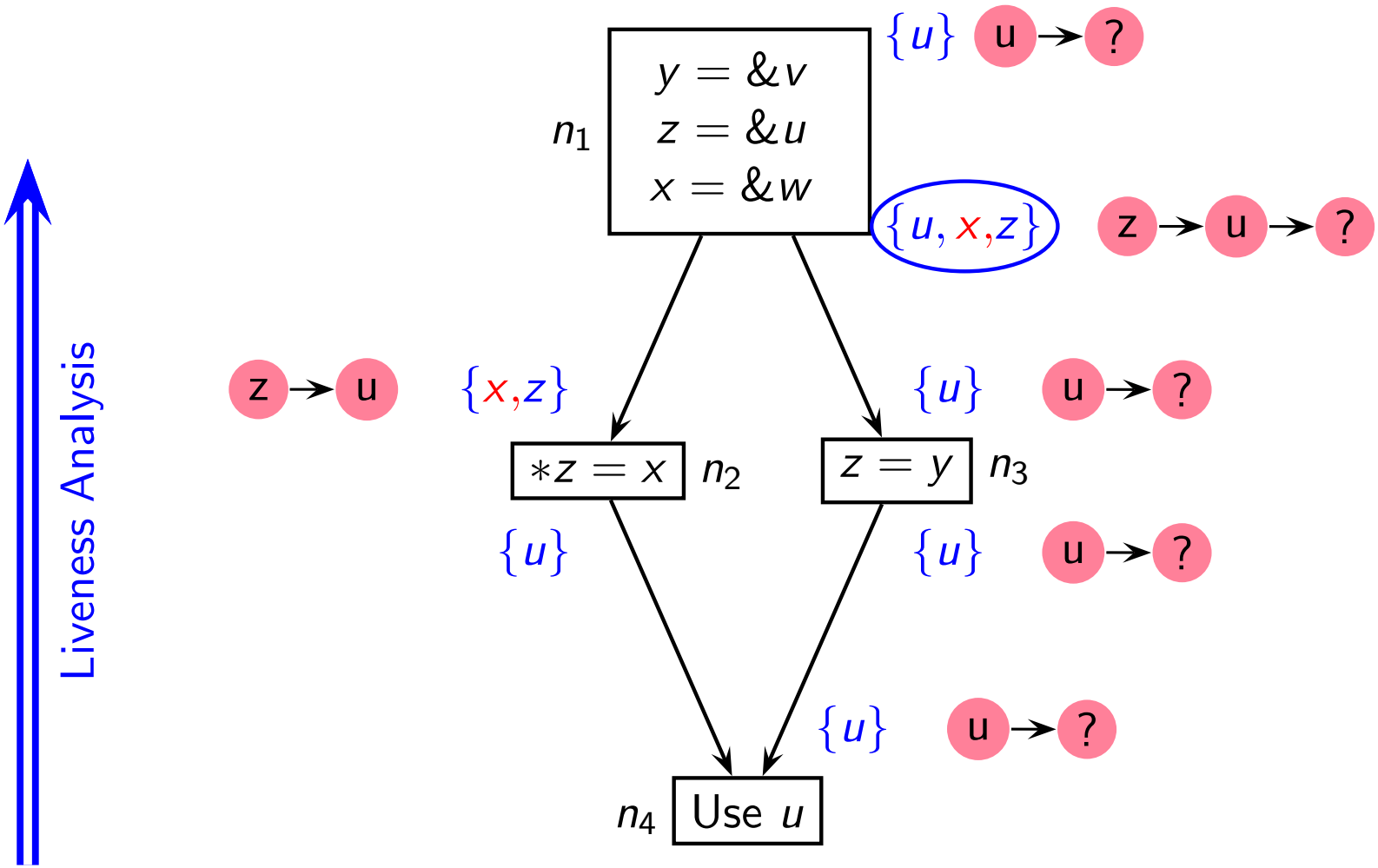
# Second Round of Liveness Analysis and Points-to Analysis



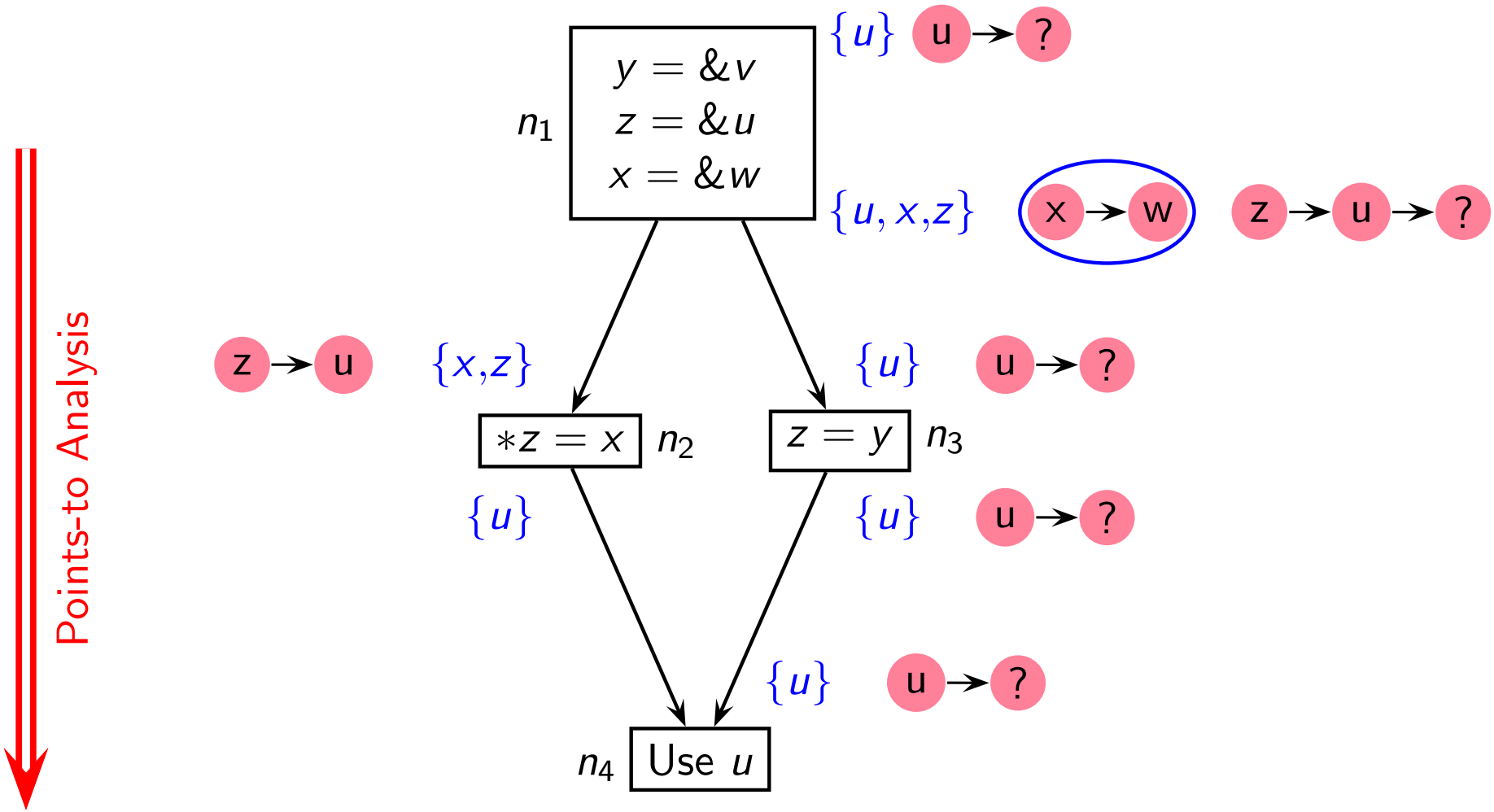
# Second Round of Liveness Analysis and Points-to Analysis



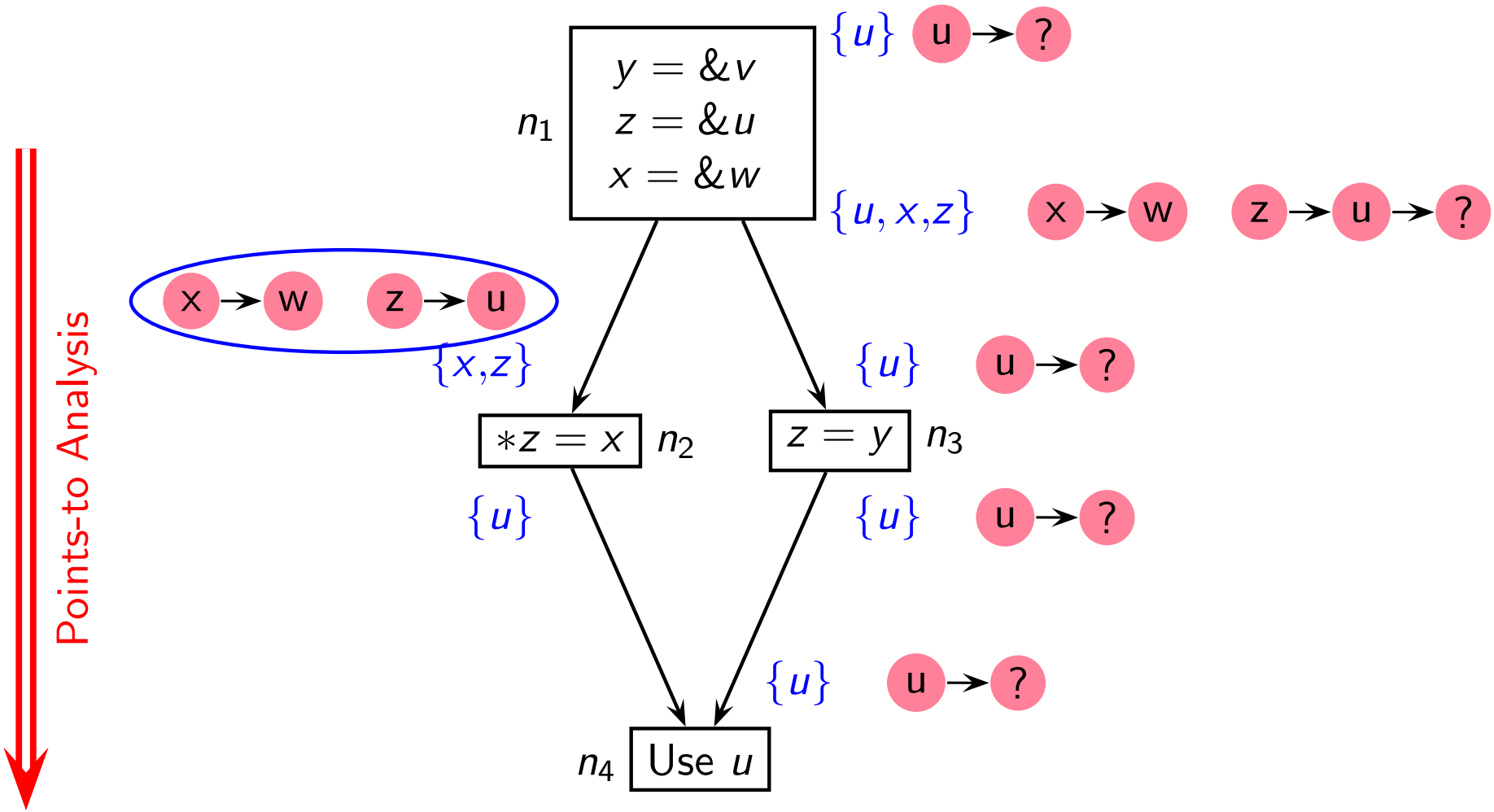
# Second Round of Liveness Analysis and Points-to Analysis



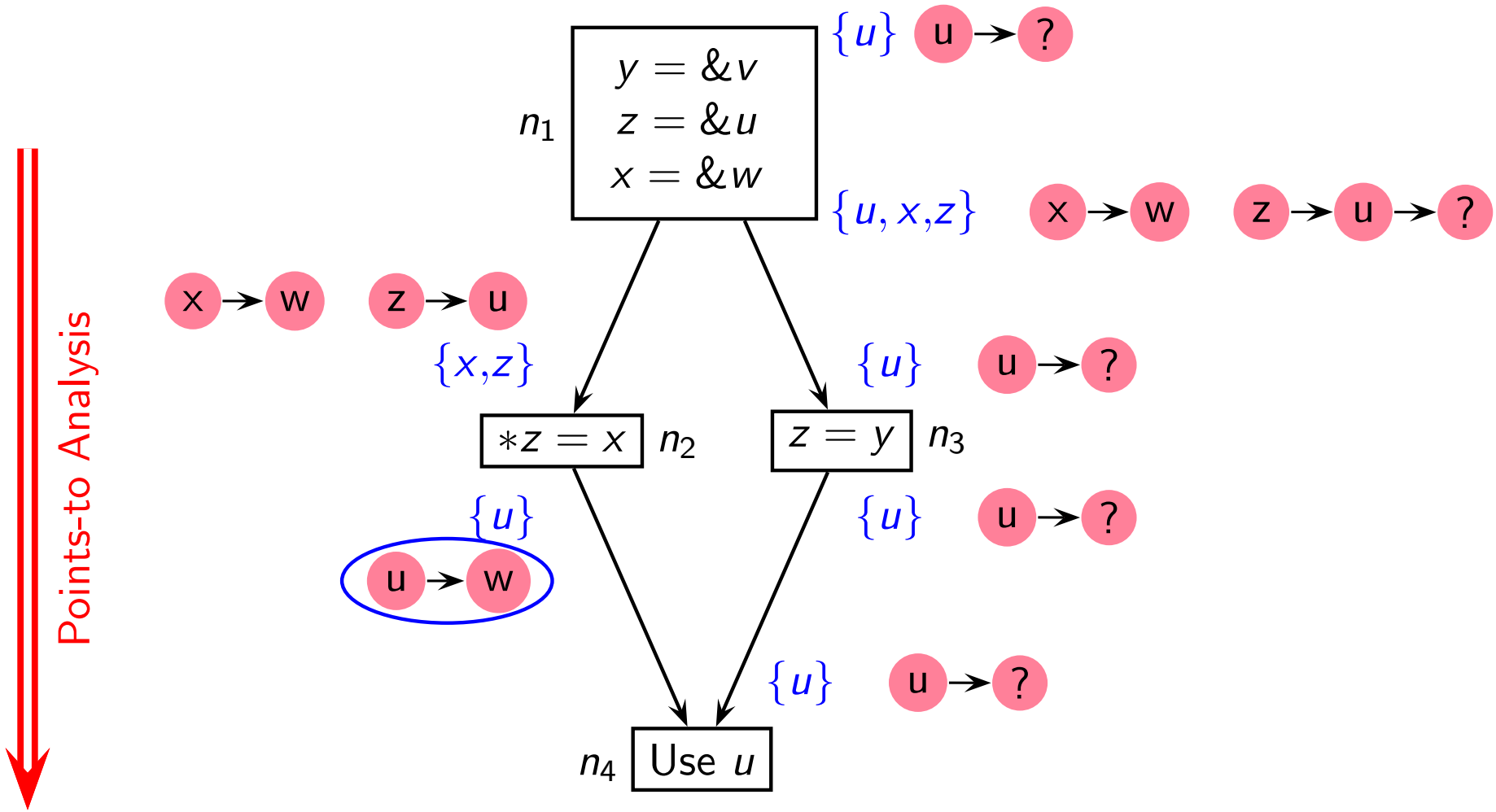
# Second Round of Liveness Analysis and Points-to Analysis



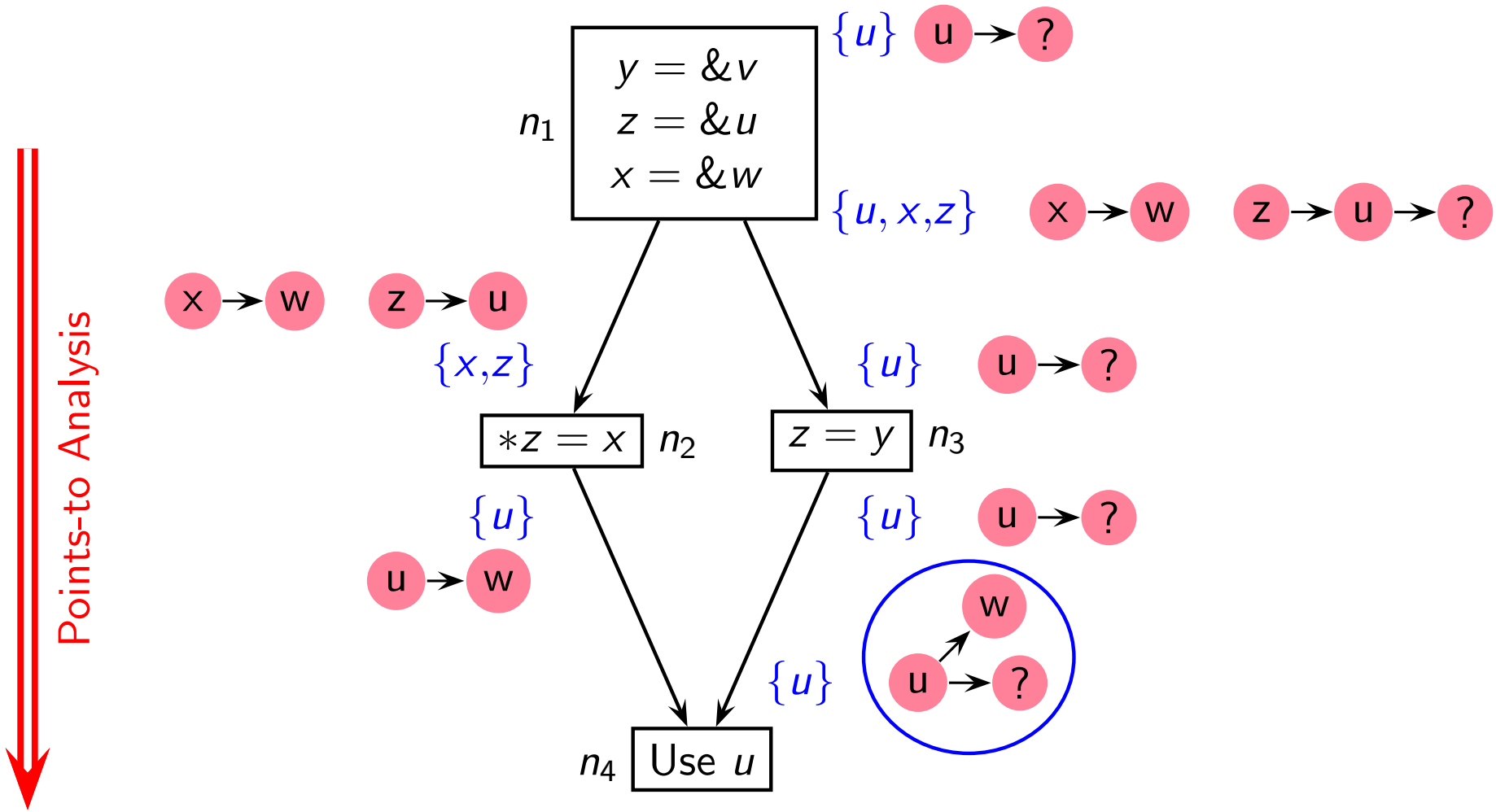
# Second Round of Liveness Analysis and Points-to Analysis



# Second Round of Liveness Analysis and Points-to Analysis



# Second Round of Liveness Analysis and Points-to Analysis



# LFCPA Observations

- Usable pointer information is very small and sparse
- Data flow propagation in real programs seems to involve only a small subset of all possible data flow values
- Earlier approaches reported inefficiency and non-scalability because they computed far more information than the actual usable information



# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation



# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

Maximum Computation

Minimum Computation

# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

Maximum Computation

Minimum Computation

Early Computation

Late Computation

# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

Early Computation

Late Computation

# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

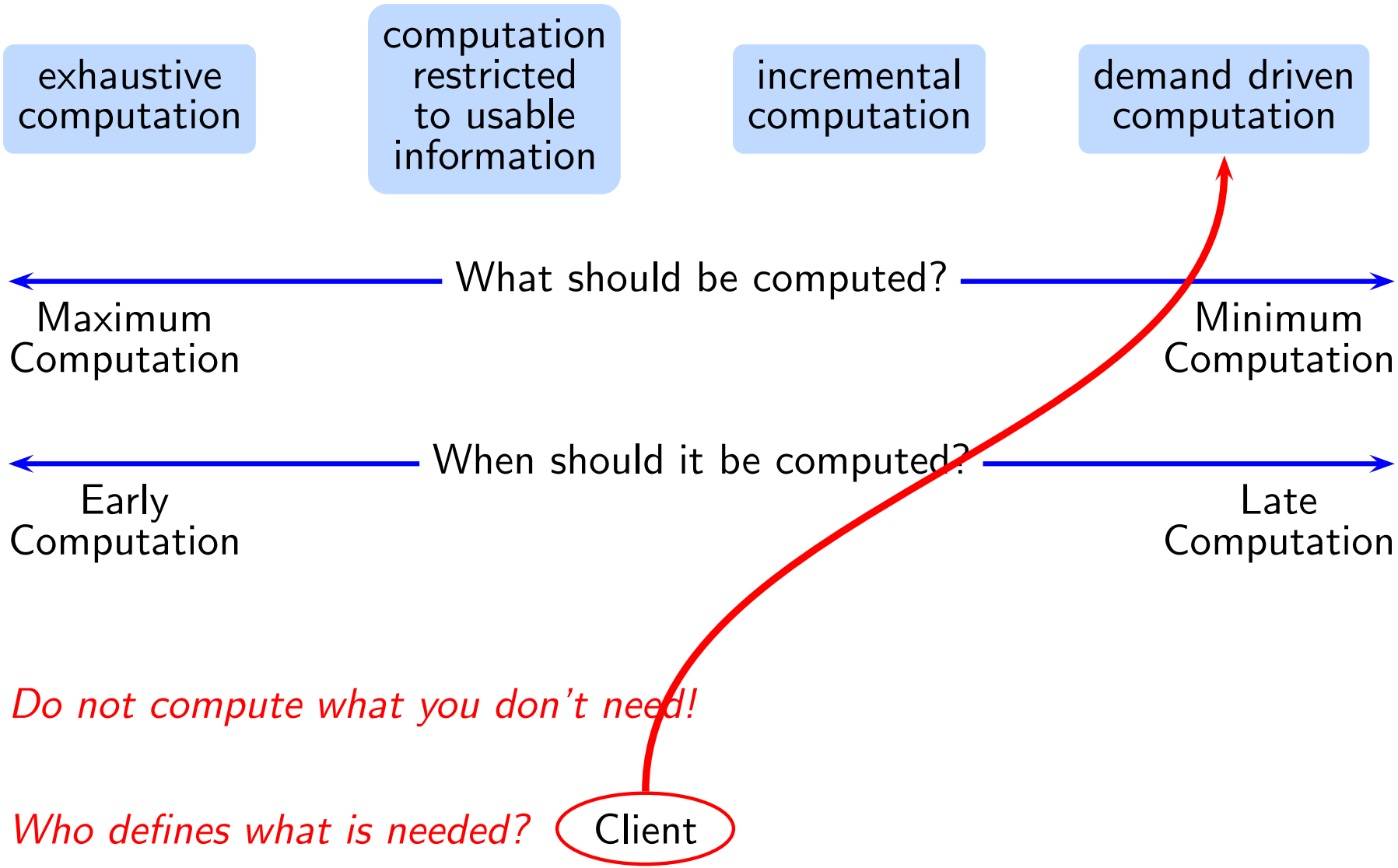
Early Computation

Late Computation

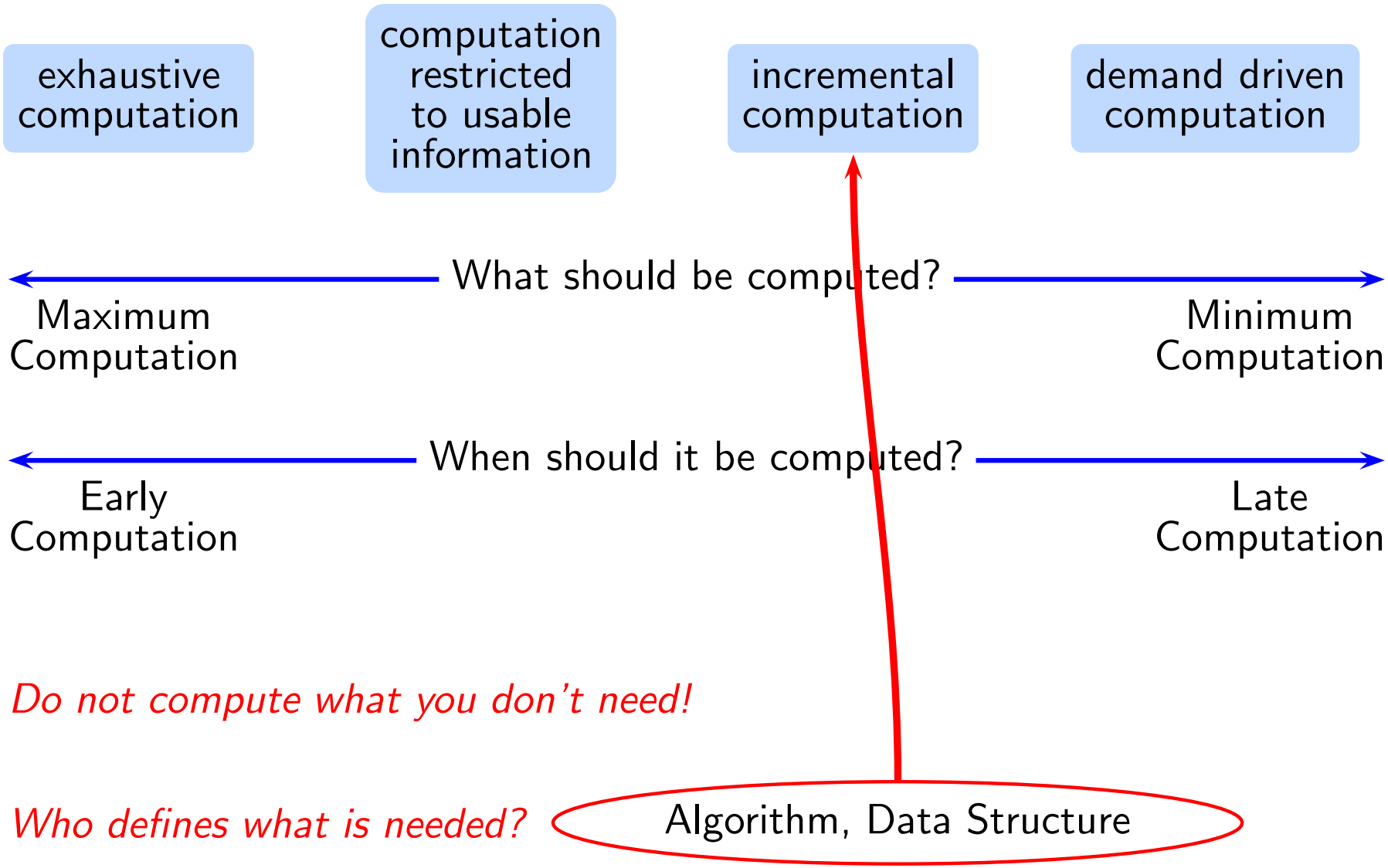
*Do not compute what you don't need!*

*Who defines what is needed?*

# LFCPA Lessons: The Larger Perspective



# LFCPA Lessons: The Larger Perspective



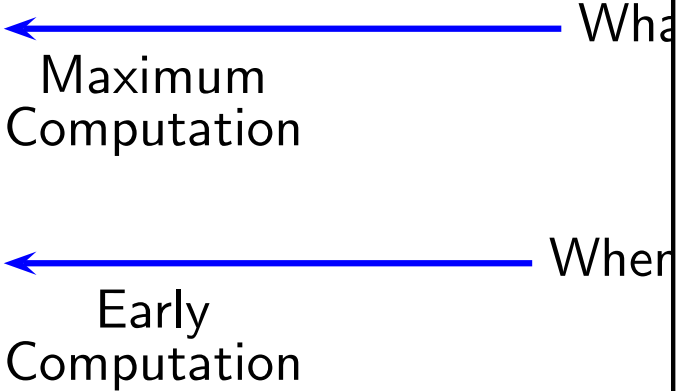
# LFCPA Lessons: The Larger Perspective

exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation




Avoid computing some values because

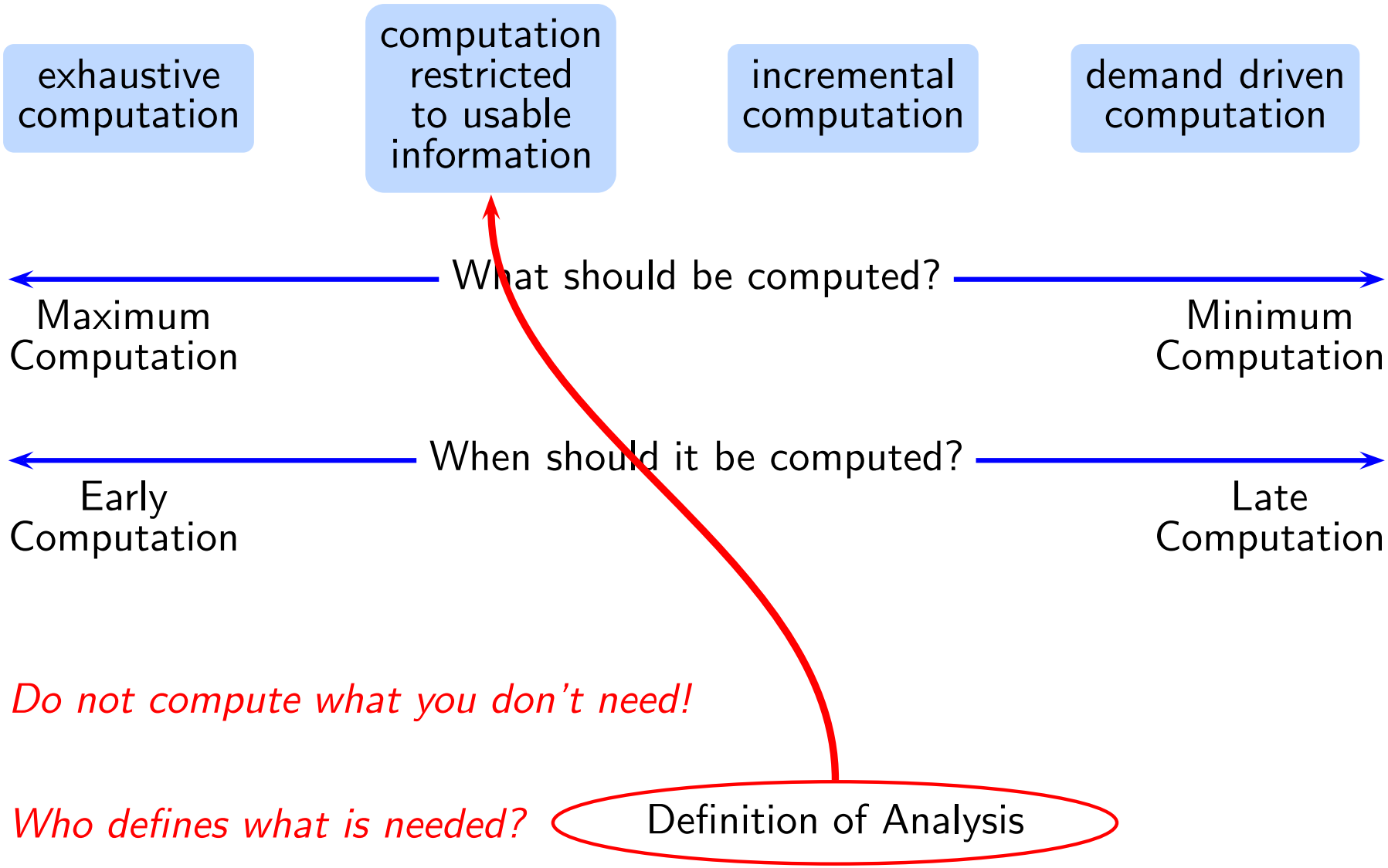
- they have been computed before, or
- they can just be "adjusted", or
- they are equivalent to some other values

E.g. Value based termination of call strings, Work list based methods, BDDs

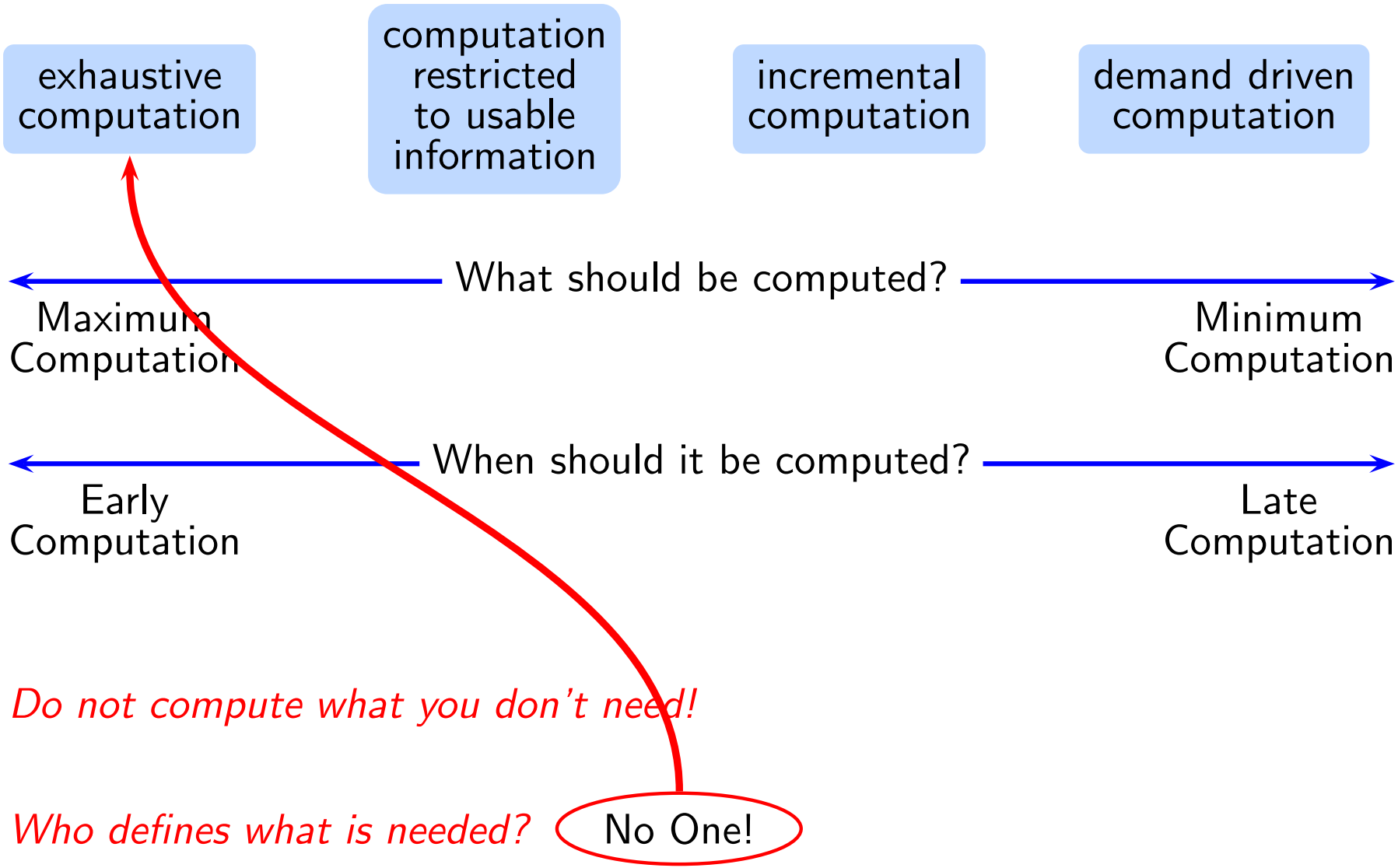
*Do not compute what you don't need!*

*Who defines what is needed?* Algorithm, Data Structure

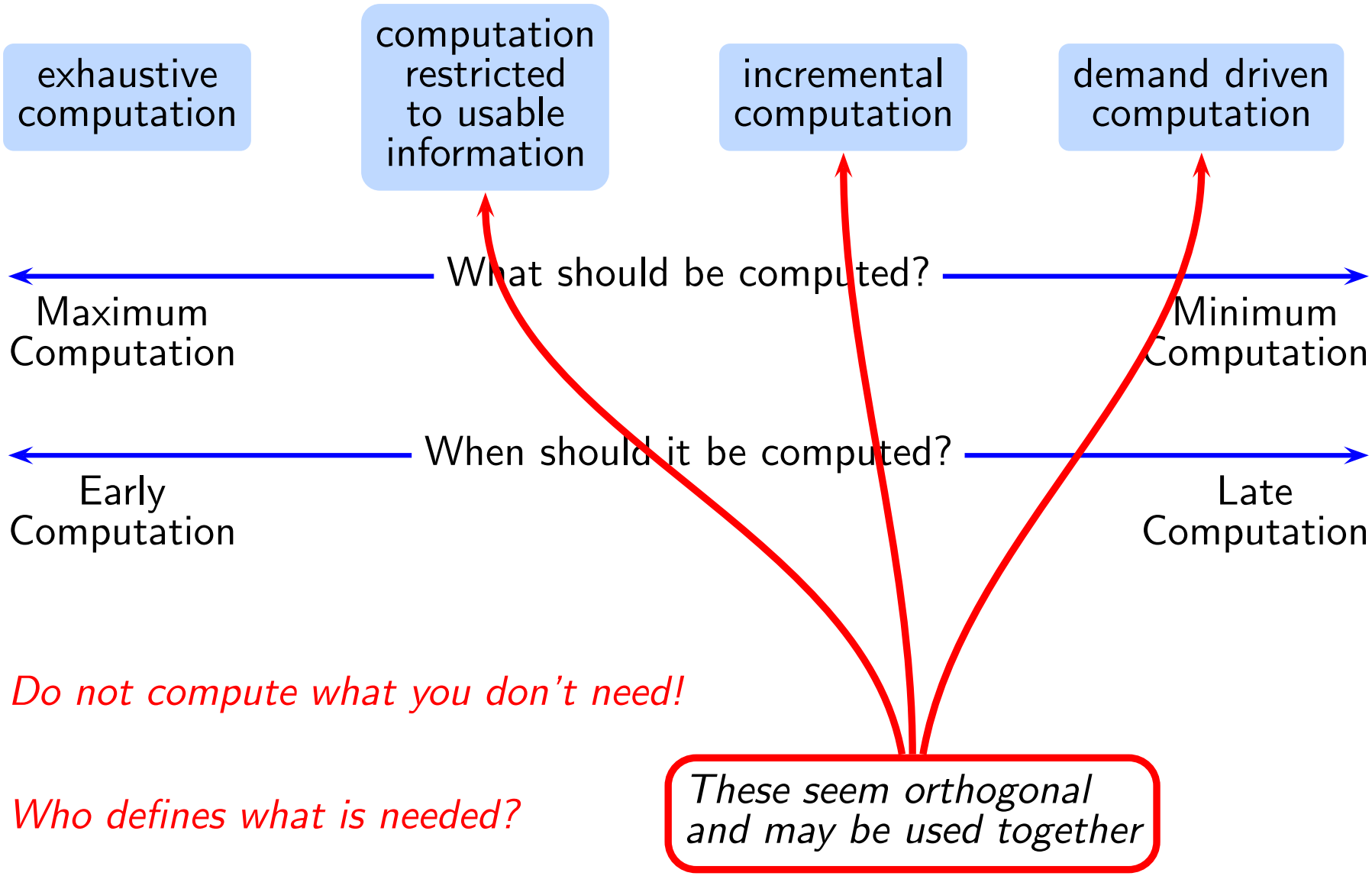
# LFCPA Lessons: The Larger Perspective



# LFCPA Lessons: The Larger Perspective



# LFCPA Lessons: The Larger Perspective



# An Outline of Pointer Analysis Coverage

- Pointer Statements
- Comparing Points-to and Alias information
- Defining Points-to Analysis
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Liveness Based Points-to Analysis
- Handling Heap Next Topic



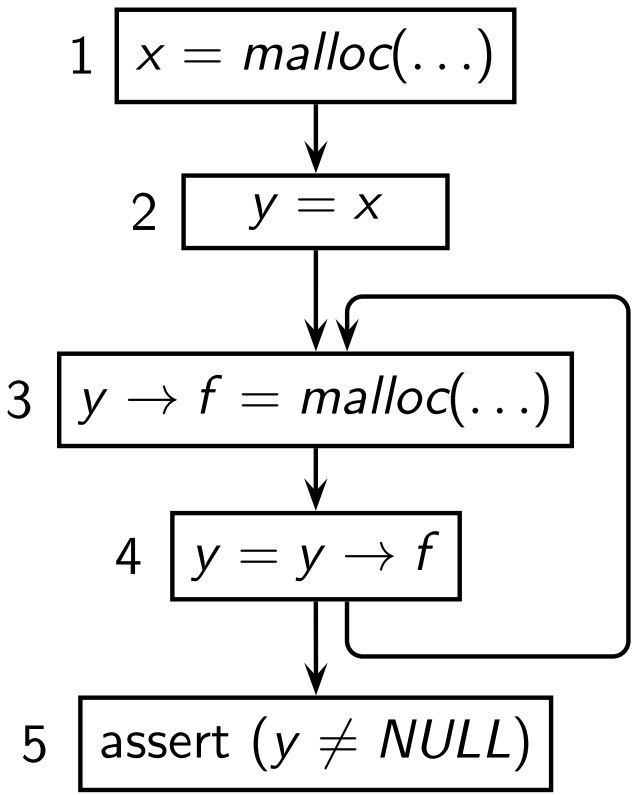
# What About Heap Data?

- Compile time entities, abstract entities, or summarized entities
- Three options:
  - Represent all heap locations by a single abstract heap location
  - Represent all heap locations of a particular type by a single abstract heap location
  - Represent all heap locations allocated at a given memory allocation site by a single abstract heap location
- Summarization of pointer expression: Usually based on the length of pointer expression



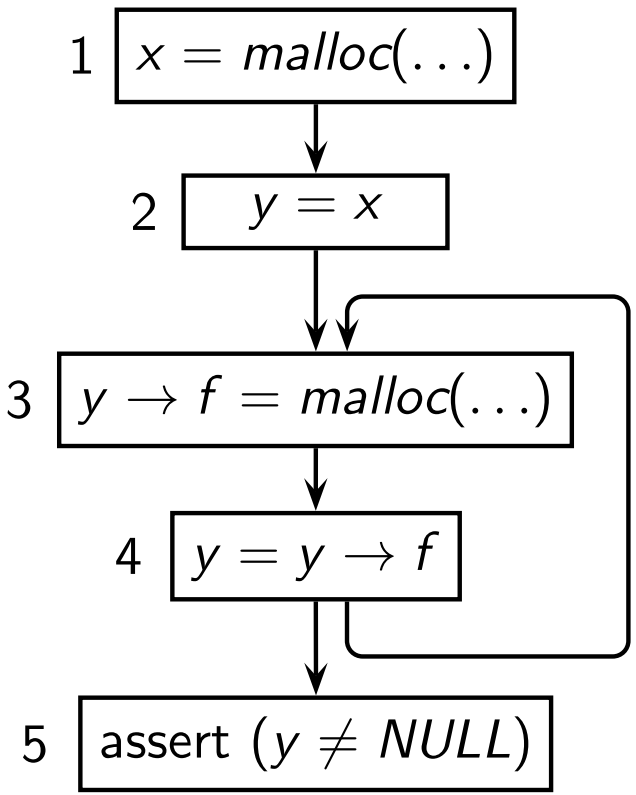
# Allocation Site Based Abstraction of Points-to Graph

Program

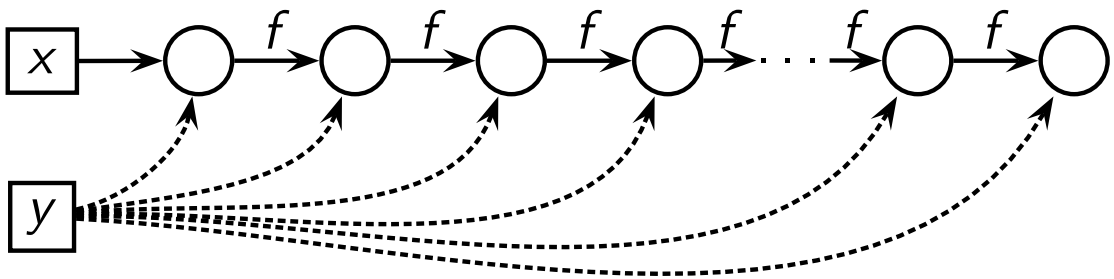


# Allocation Site Based Abstraction of Points-to Graph

Program

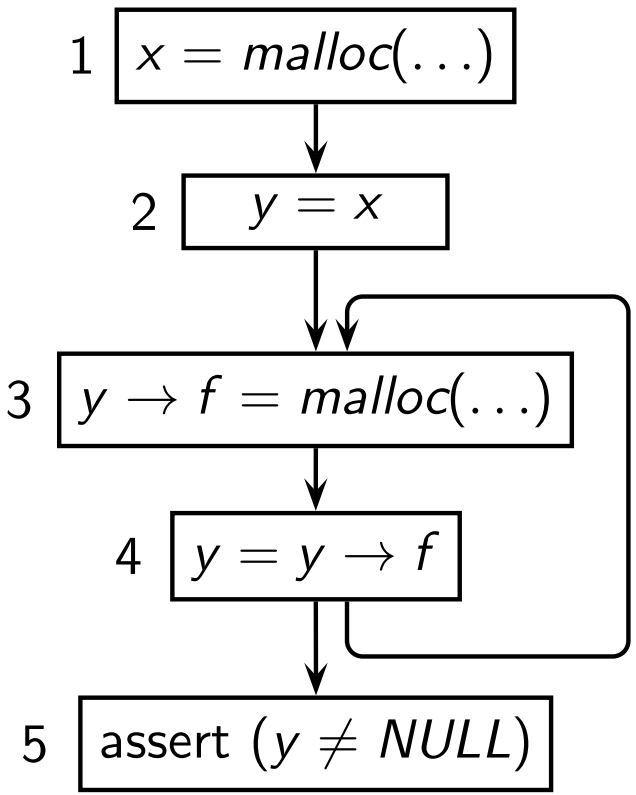



Memory graph representing multiple executions

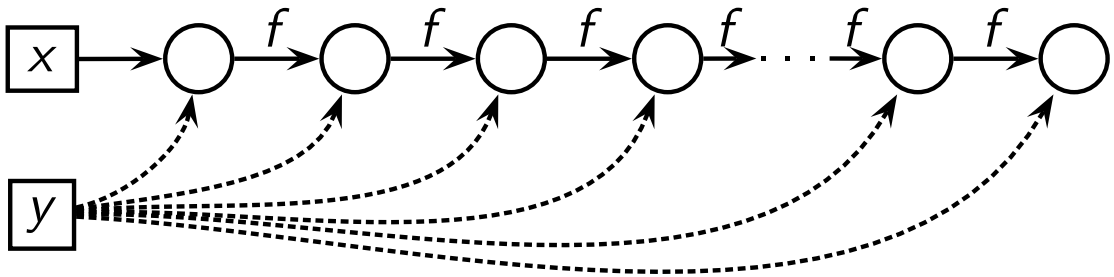


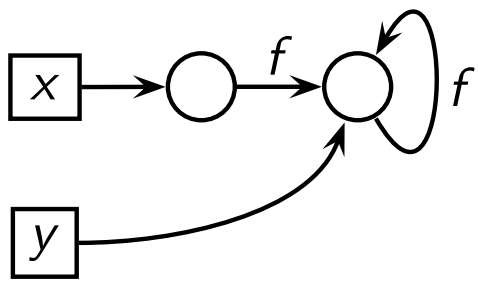
# Allocation Site Based Abstraction of Points-to Graph

Program



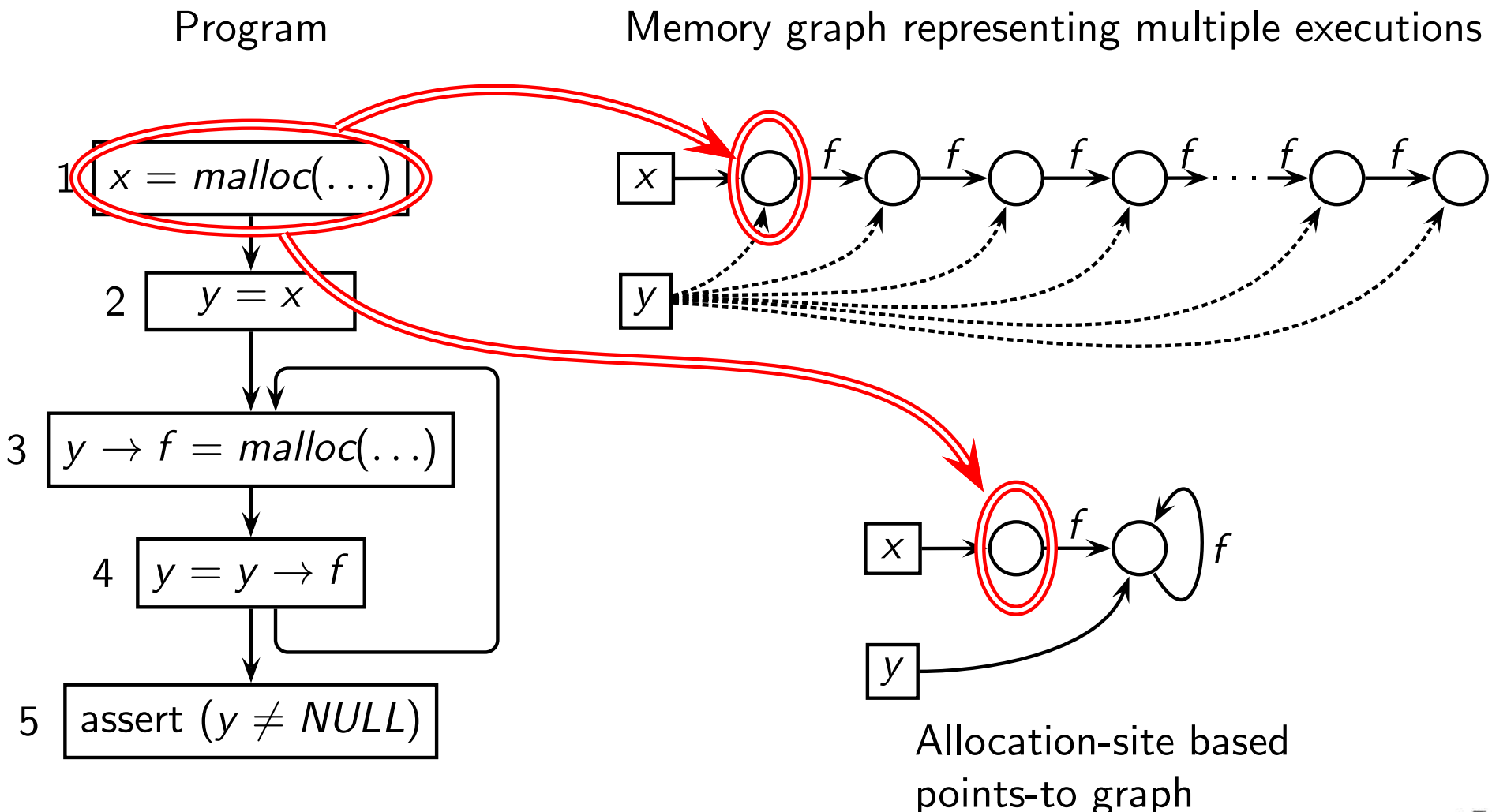

Memory graph representing multiple executions




Allocation-site based points-to graph

# Allocation Site Based Abstraction of Points-to Graph



# Allocation Site Based Abstraction of Points-to Graph