

# Slicing the Static Single Information Form

Jeremy Singer

`jeremy.singer@cl.cam.ac.uk`

`http://www.cl.cam.ac.uk/~jds31`

Computer Laboratory, University of Cambridge

# *Compiler IR Zoo*

# *Compiler IR Zoo*

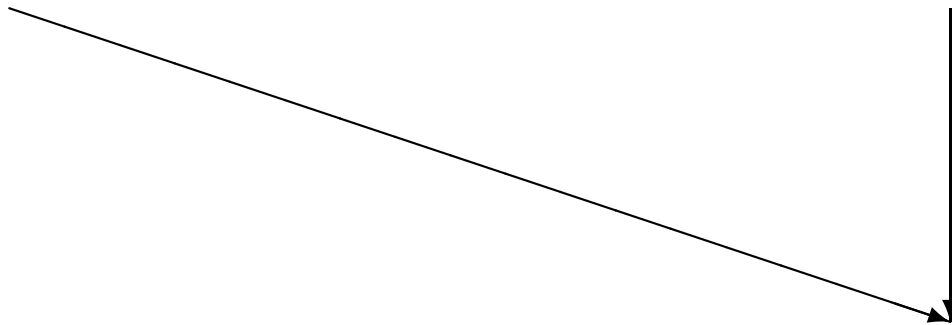
control flow graph  
(CFG)

data dependence graph  
(DDG)

# *Compiler IR Zoo*

control flow graph  
(CFG)

data dependence graph  
(DDG)

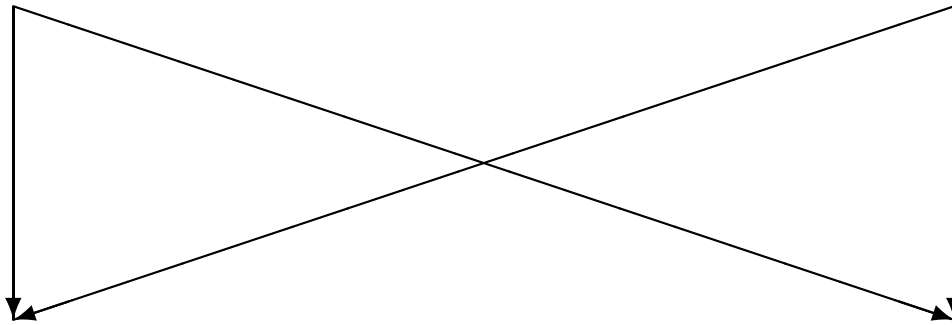


PDG, PDW, VDG, VSDG

# Compiler IR Zoo

control flow graph  
(CFG)

data dependence graph  
(DDG)



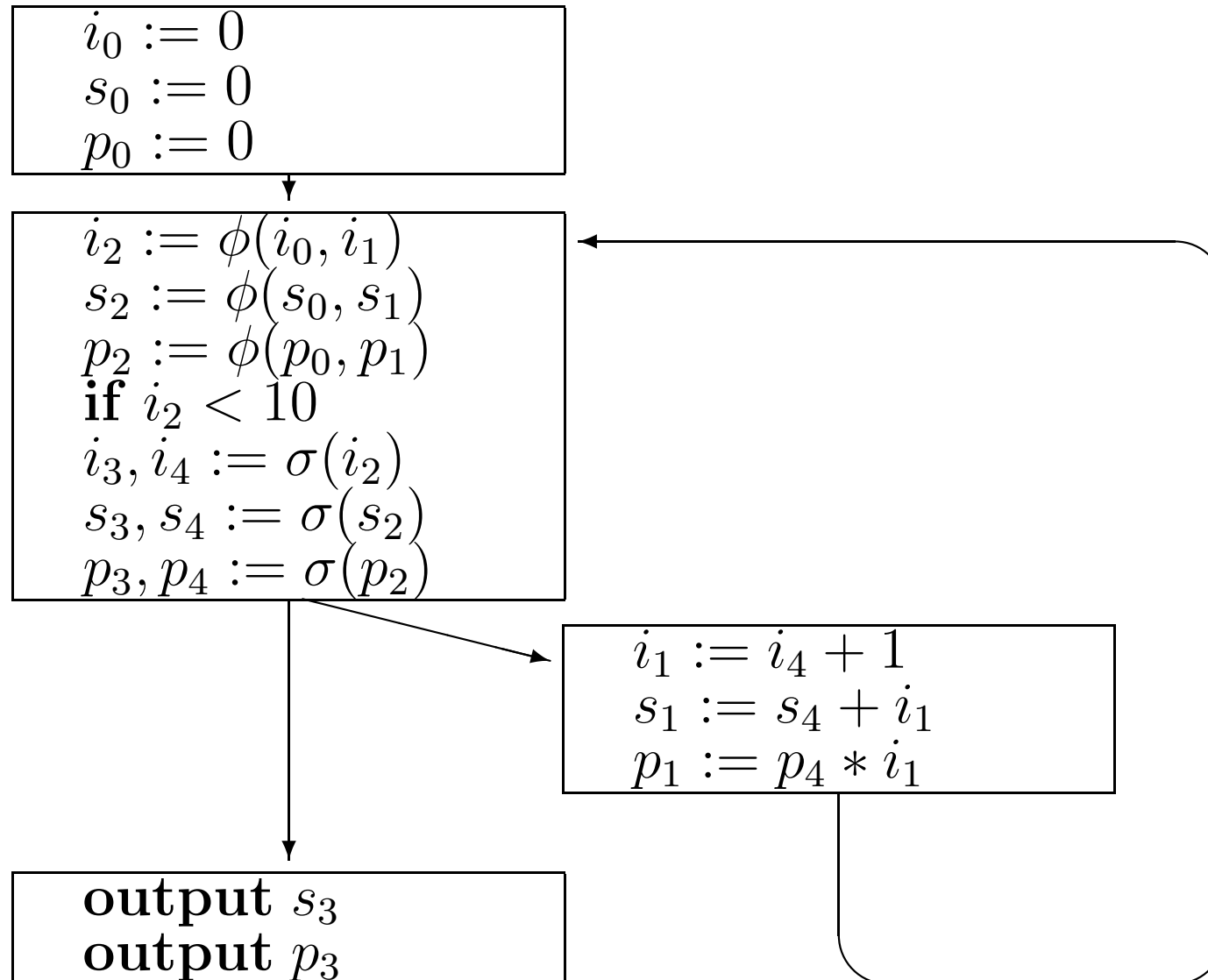
SSA, SSU, SSI

PDG, PDW, VDG, VSDG

# *Static Single Information Form*

- systematic renaming of variables
- dependences implicit in naming scheme
- pseudo-definition functions:  $\phi$  and  $\sigma$

# SSI Example Program



# *Applications of SSI*

- sparse data flow analysis
  - forward analysis e.g. constant propagation
  - predicated analysis e.g. constant propagation
  - backward analysis e.g. live variables
  - bidirectional analysis e.g. type inference
- program slicing

# *SSI Slicing Background*

To be precise:

- backward slicing
- static slicing
- executable slices
- syntax-preserving slices

# *SSI Slicing Background*

To be precise:

- backward slicing
- static slicing
- executable slices
- syntax-preserving slices

Notion of variable dependence:

- data dependence
- control dependence

# *SSI Slicing Algorithm*

Equation for variable dependence:

$$\begin{aligned} \text{vardep}(v : \text{var}, p : \text{prog}) = \\ \text{if } \text{alreadyseen}(v) \text{ return } \{\} \\ \text{else return } \text{ref}(\text{defsite}(v)) \cup \\ \left( \bigcup_{x \in \text{ref}(\text{defsite}(v))} \text{vardep}(x, p) \right) \end{aligned}$$

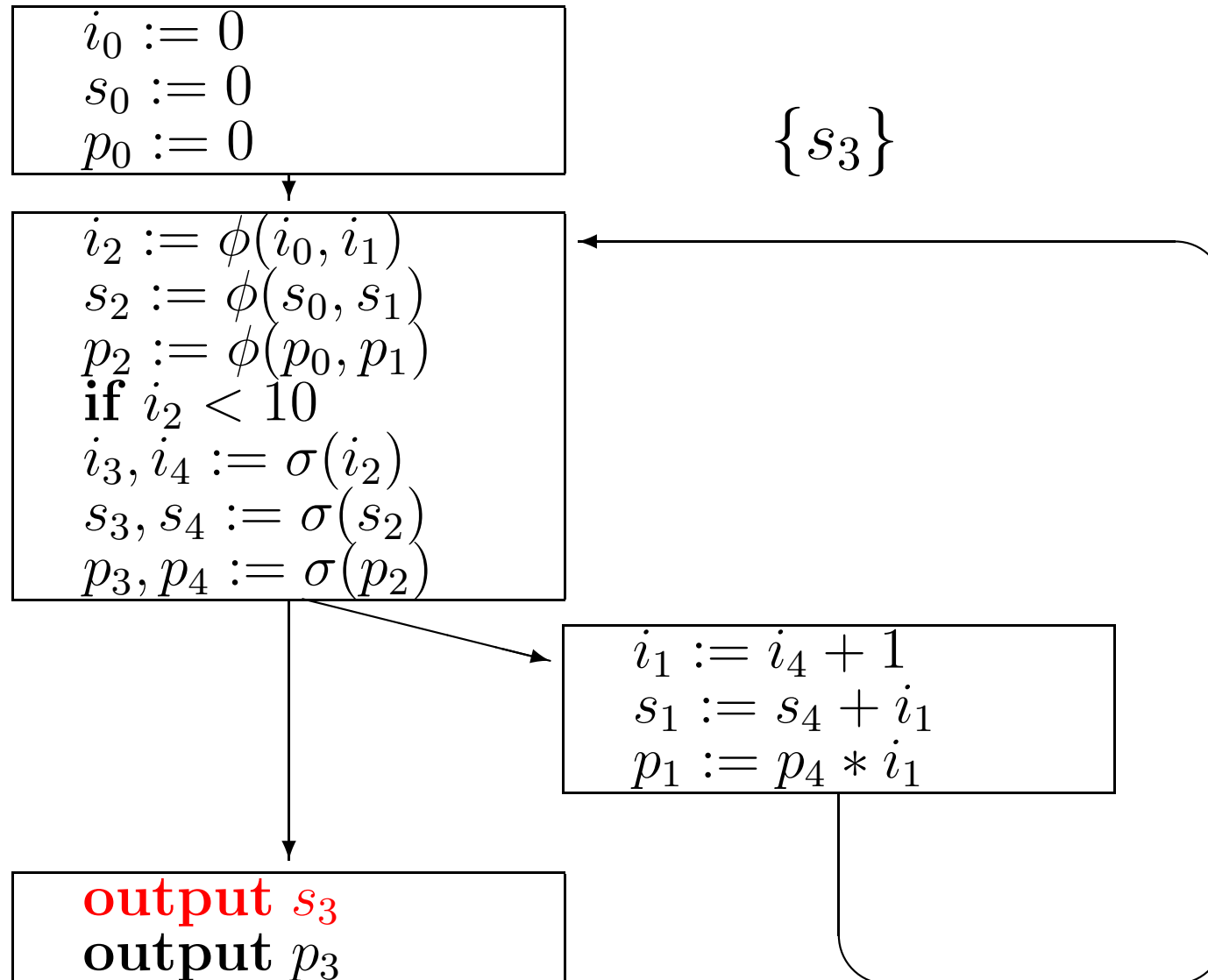
# *SSI Slicing Algorithm*

Equation for variable dependence:

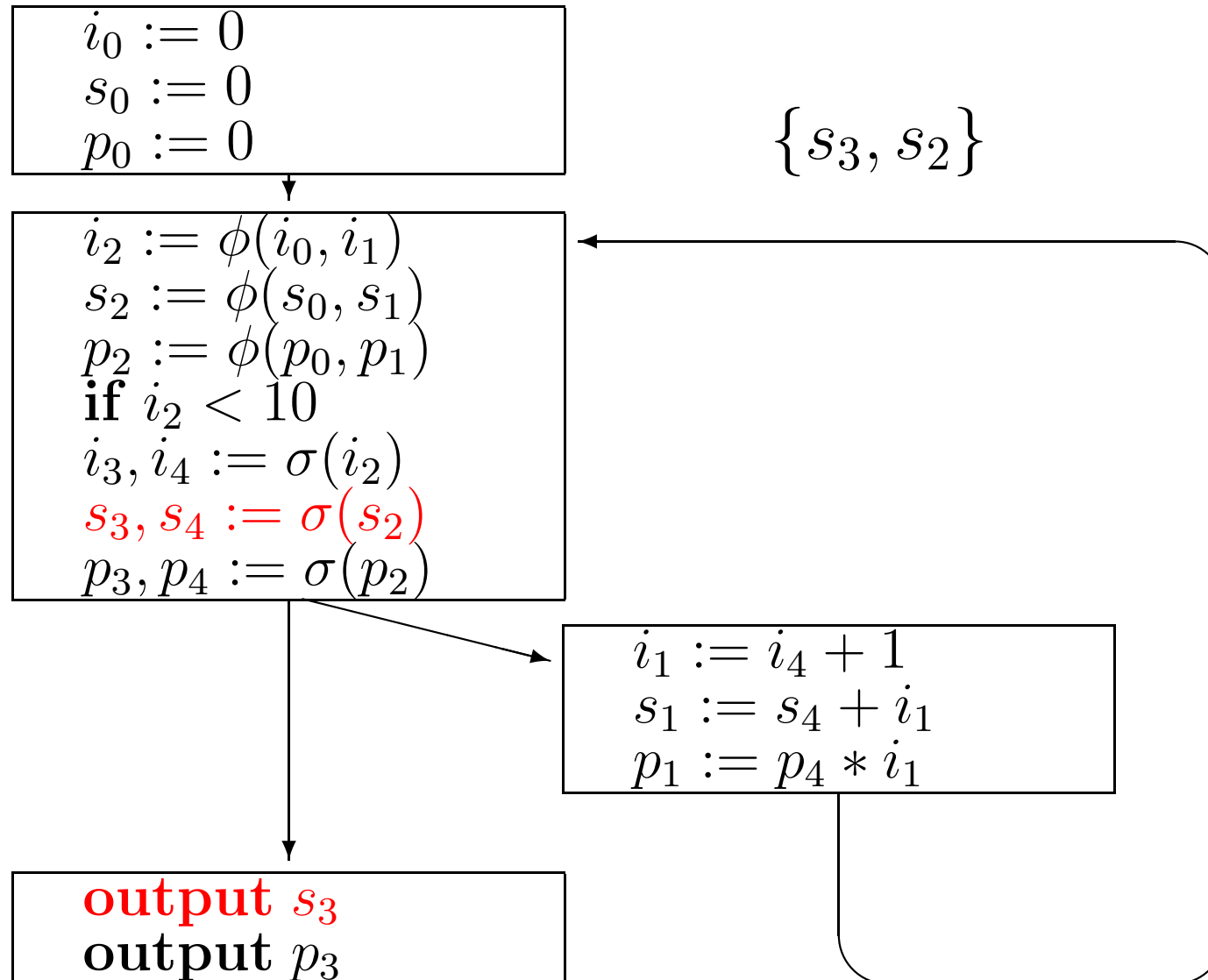
$$\begin{aligned} \text{vardep}(v : \text{var}, p : \text{prog}) = \\ \text{if } \text{alreadyseen}(v) \text{ return } \{\} \\ \text{else return } \text{ref}(\text{defsite}(v)) \cup \\ \left( \bigcup_{x \in \text{ref}(\text{defsite}(v))} \text{vardep}(x, p) \right) \end{aligned}$$

Slice consists of *definitions* of variables that are in vardep set, together with *conditionals* associated with  $\sigma$ -fns in vardep set.

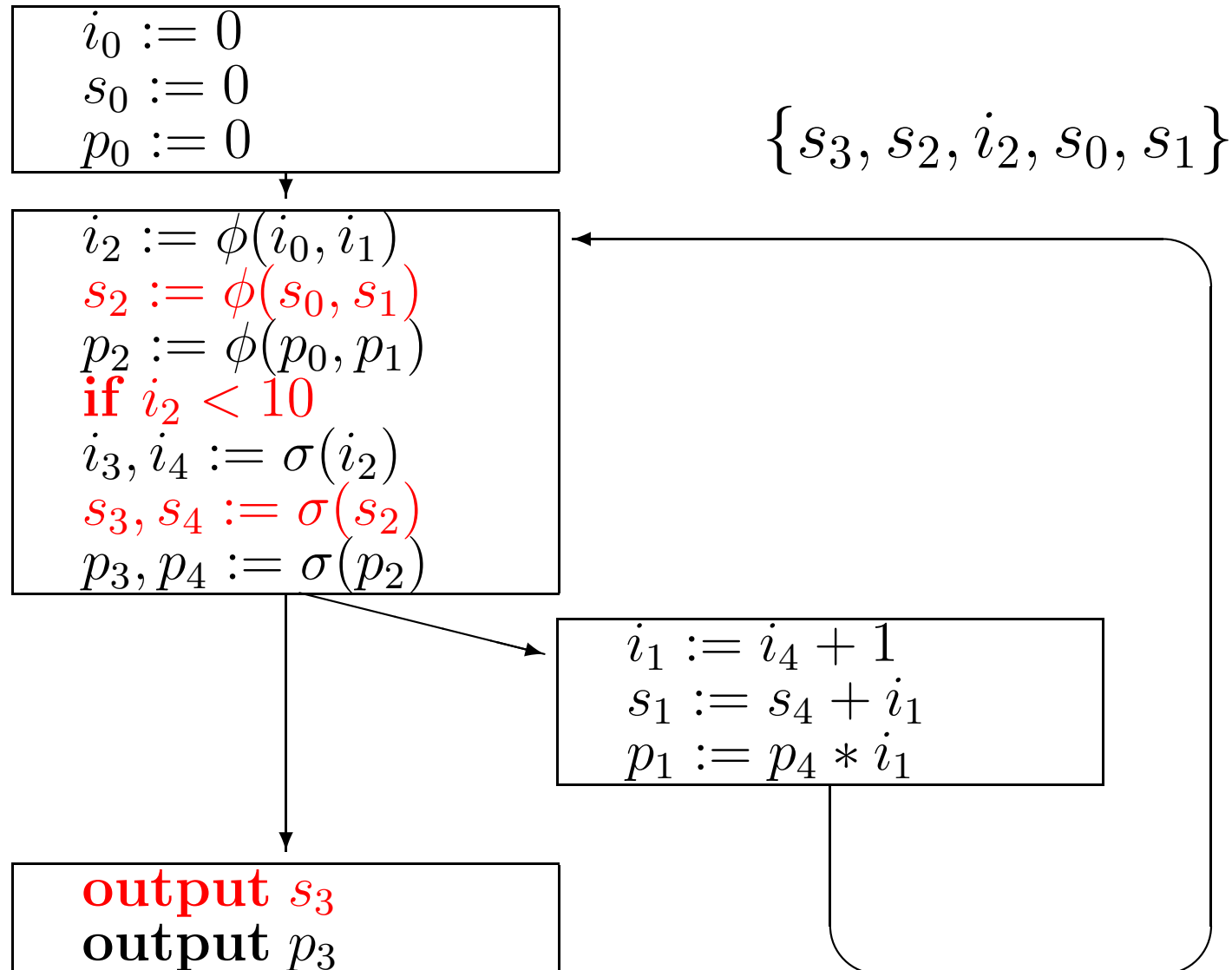
# Workthrough: 1/6



# Workthrough: 2/6



# Workthrough: 3/6



# Workthrough: 4/6

$i_0 := 0$   
 $s_0 := 0$   
 $p_0 := 0$

$\{s_3, s_2, i_2, s_0, s_1, i_0, i_1, s_4\}$

$i_2 := \phi(i_0, i_1)$   
 $s_2 := \phi(s_0, s_1)$   
 $p_2 := \phi(p_0, p_1)$   
**if**  $i_2 < 10$   
 $i_3, i_4 := \sigma(i_2)$   
 $s_3, s_4 := \sigma(s_2)$   
 $p_3, p_4 := \sigma(p_2)$

$i_1 := i_4 + 1$   
 $s_1 := s_4 + i_1$   
 $p_1 := p_4 * i_1$

**output**  $s_3$   
**output**  $p_3$

# Workthrough: 5/6

$i_0 := 0$   
 $s_0 := 0$   
 $p_0 := 0$

$\{s_3, s_2, i_2, s_0, s_1, i_0, i_1, s_4, i_4\}$

$i_2 := \phi(i_0, i_1)$   
 $s_2 := \phi(s_0, s_1)$   
 $p_2 := \phi(p_0, p_1)$   
**if**  $i_2 < 10$   
 $i_3, i_4 := \sigma(i_2)$   
 $s_3, s_4 := \sigma(s_2)$   
 $p_3, p_4 := \sigma(p_2)$

$i_1 := i_4 + 1$   
 $s_1 := s_4 + i_1$   
 $p_1 := p_4 * i_1$

**output**  $s_3$   
**output**  $p_3$

# Workthrough: 6/6

$i_0 := 0$   
 $s_0 := 0$   
 $p_0 := 0$

$\{s_3, s_2, i_2, s_0, s_1, i_0, i_1, s_4, i_4\}$

$i_2 := \phi(i_0, i_1)$   
 $s_2 := \phi(s_0, s_1)$   
 $p_2 := \phi(p_0, p_1)$   
**if**  $i_2 < 10$   
 $i_3, i_4 := \sigma(i_2)$   
 $s_3, s_4 := \sigma(s_2)$   
 $p_3, p_4 := \sigma(p_2)$

$i_1 := i_4 + 1$   
 $s_1 := s_4 + i_1$   
 $p_1 := p_4 * i_1$

**output**  $s_3$   
**output**  $p_3$

# *SSI vs CFG Slicing*

CFG slicing:

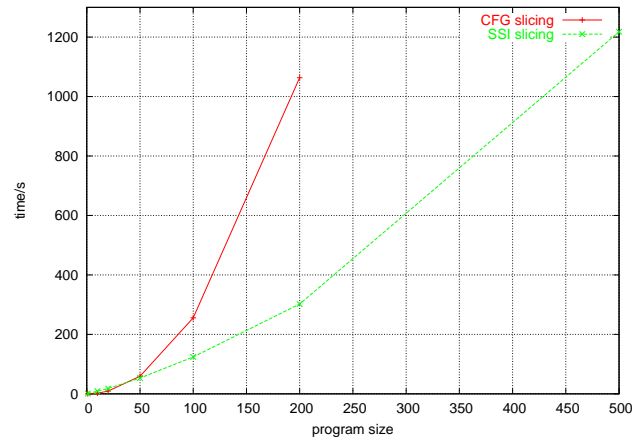
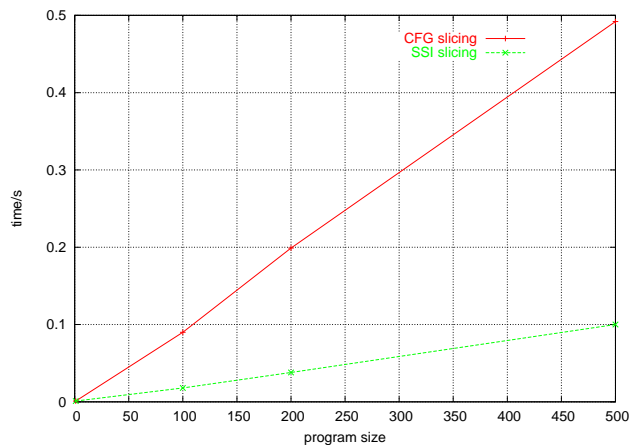
- Weiser-style
- iterative data flow analysis (inefficient)
- control dep much less efficient than data dep

# *SSI vs CFG Slicing*

CFG slicing:

- Weiser-style
- iterative data flow analysis (inefficient)
- control dep much less efficient than data dep

Graphs to show performance difference:



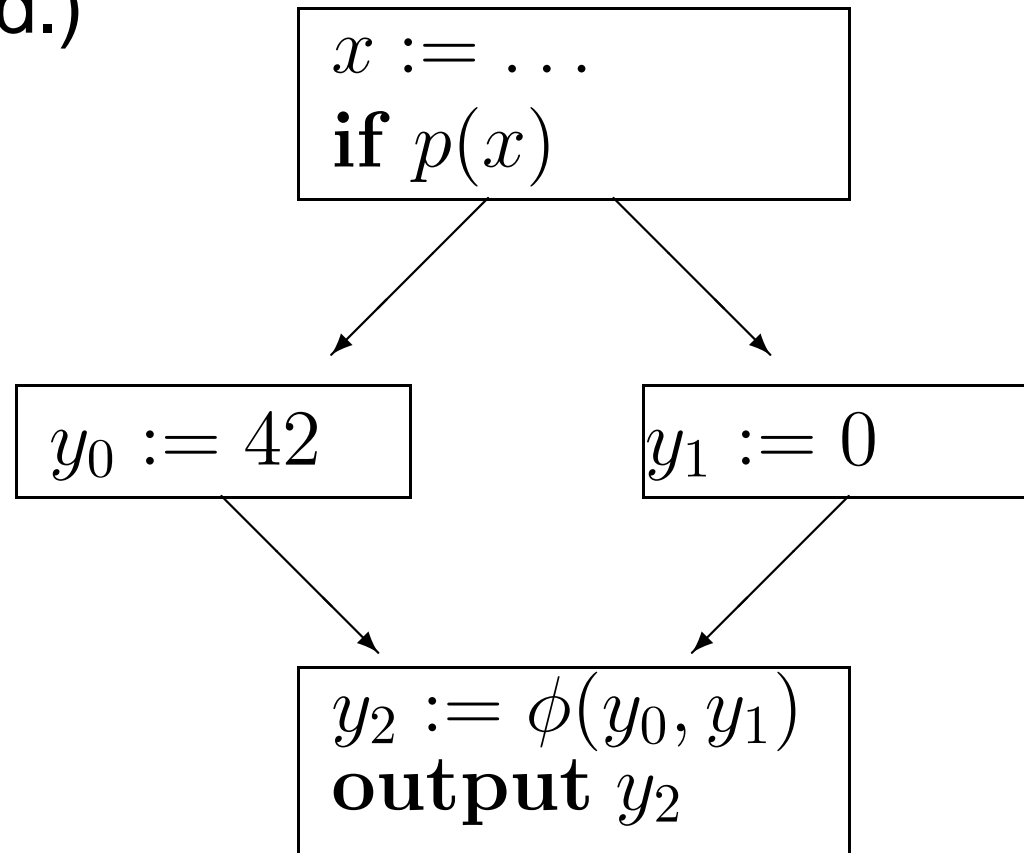
# *SSI vs PDG Slicing*

PDG slicing:

- Similar to SSI. Linear, follows dependence edges.
- More data dependence edges in PDG. SSI has single-assignment property, so eliminates useless data dep edges.
- More control dependence edges in PDG. 1 per instr in conditional context in PDG. 1 per var used in conditional context (effectively) in SSI. Is SSI good enough? Not quite . . .

# *Problem with SSI Slicing*

Not enough control dependence edges!  
(Control dependence of constant operands not represented.)



# *Possible Solutions*

- Explicit control dependence edges for all instrs.  
Same as PDG!

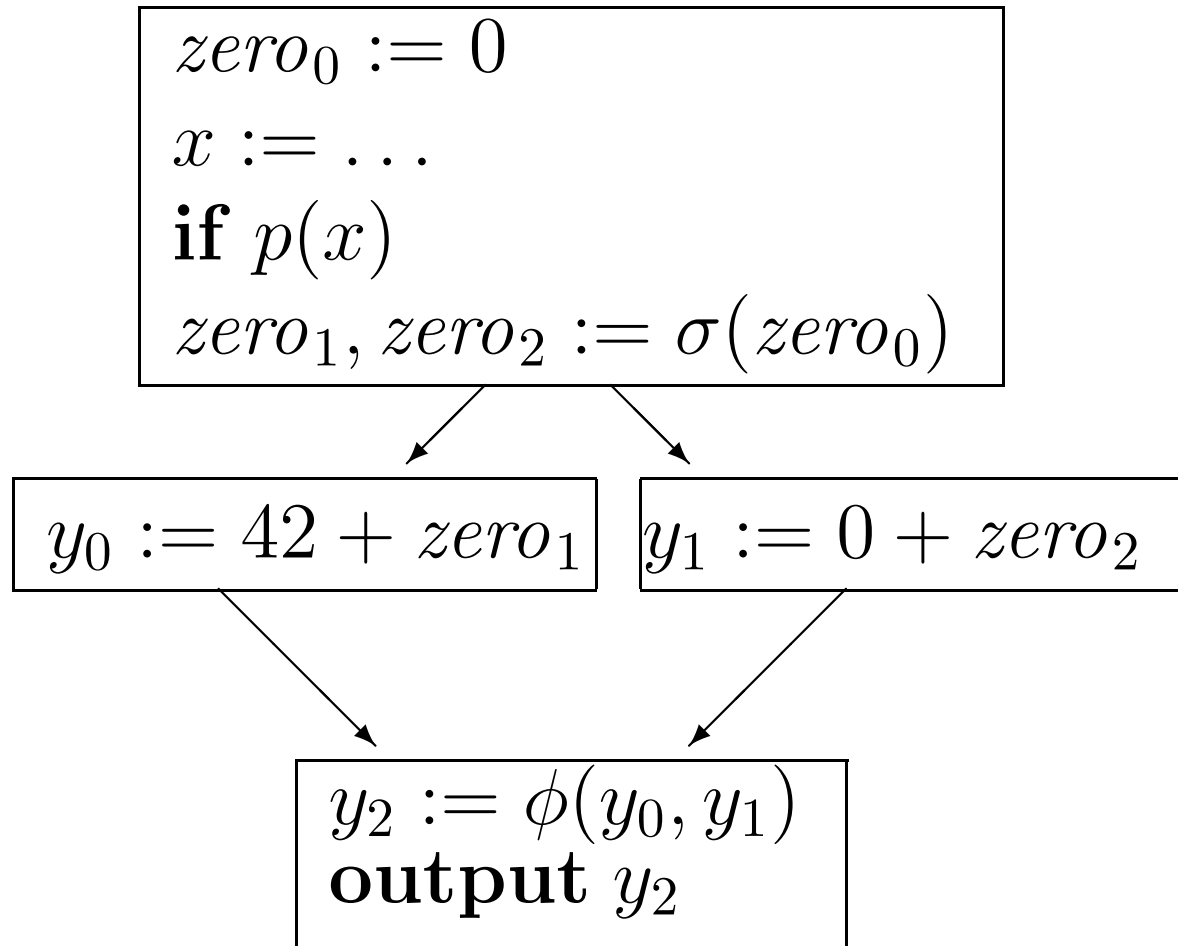
# *Possible Solutions*

- Explicit control dependence edges for all instrs. Same as PDG!
- $\gamma$ -functions instead of  $\phi$ -functions. Have an extra operand to select which source operand's value to assign to dest operand. Same as GSA!

## *Possible Solutions*

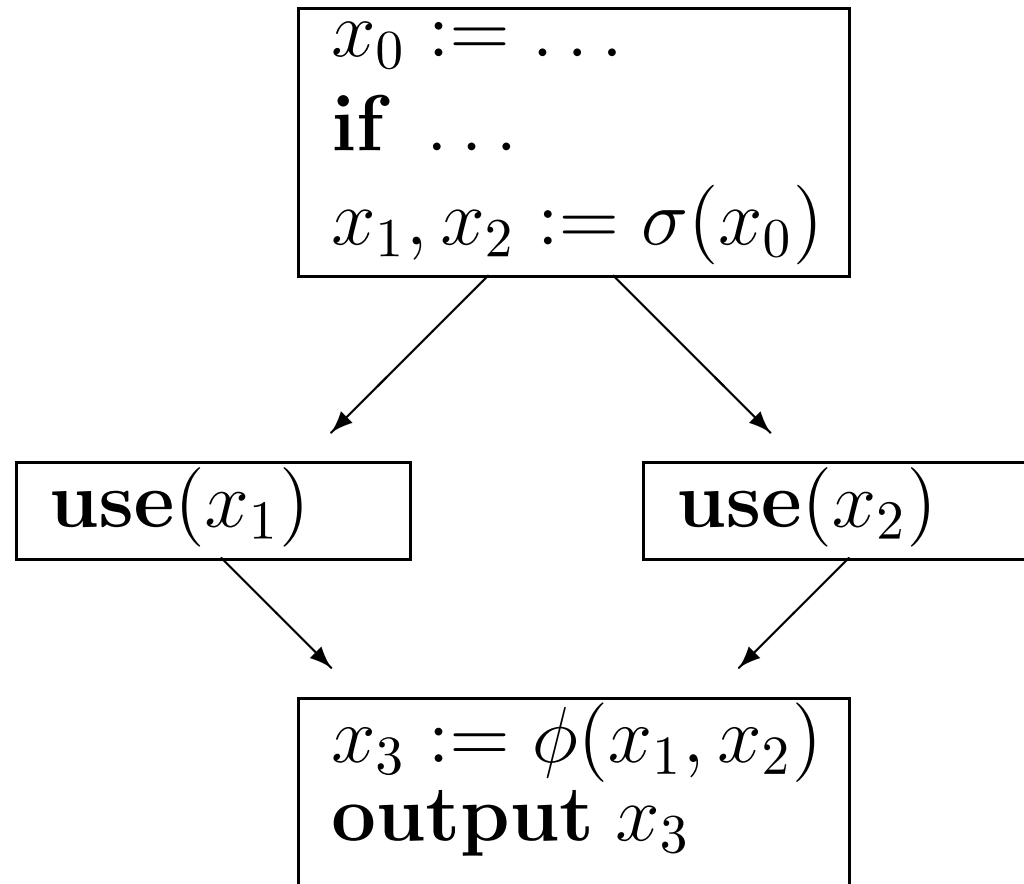
- Explicit control dependence edges for all instrs. Same as PDG!
- $\gamma$ -functions instead of  $\phi$ -functions. Have an extra operand to select which source operand's value to assign to dest operand. Same as GSA!
- Pseudo-variable “zero”. Rewrite program so all assignments of constant values use “zero”. (example)

# Example Program Rewrite



# *Another Problem with SSI Slicing*

Too much control dependence!



Solution: discount some control dep info.

# *Future Directions*

Use SSI for other slicing variants:

# *Future Directions*

Use SSI for other slicing variants:

- Forward slicing. Trace def-use chains, rather than use-def chains.

# *Future Directions*

Use SSI for other slicing variants:

- Forward slicing. Trace def-use chains, rather than use-def chains.
- Amorphous slicing. SSI is part of compiler infrastructure, so apply optimizing compiler transformations on SSI-based slices.  
(before/after slicing?)

# *Future Directions*

Use SSI for other slicing variants:

- Forward slicing. Trace def-use chains, rather than use-def chains.
- Amorphous slicing. SSI is part of compiler infrastructure, so apply optimizing compiler transformations on SSI-based slices.  
(before/after slicing?)
- Type-based slicing: not sure if much work has been done in this area. SSI is good for type inference. So, use SSI slicing for type dependences of vars (rather than value dependencies).