

# Static Program Analysis

Yue Li and Tian Tan



2020 Spring

# Static Program Analysis

## Pointer Analysis Foundations

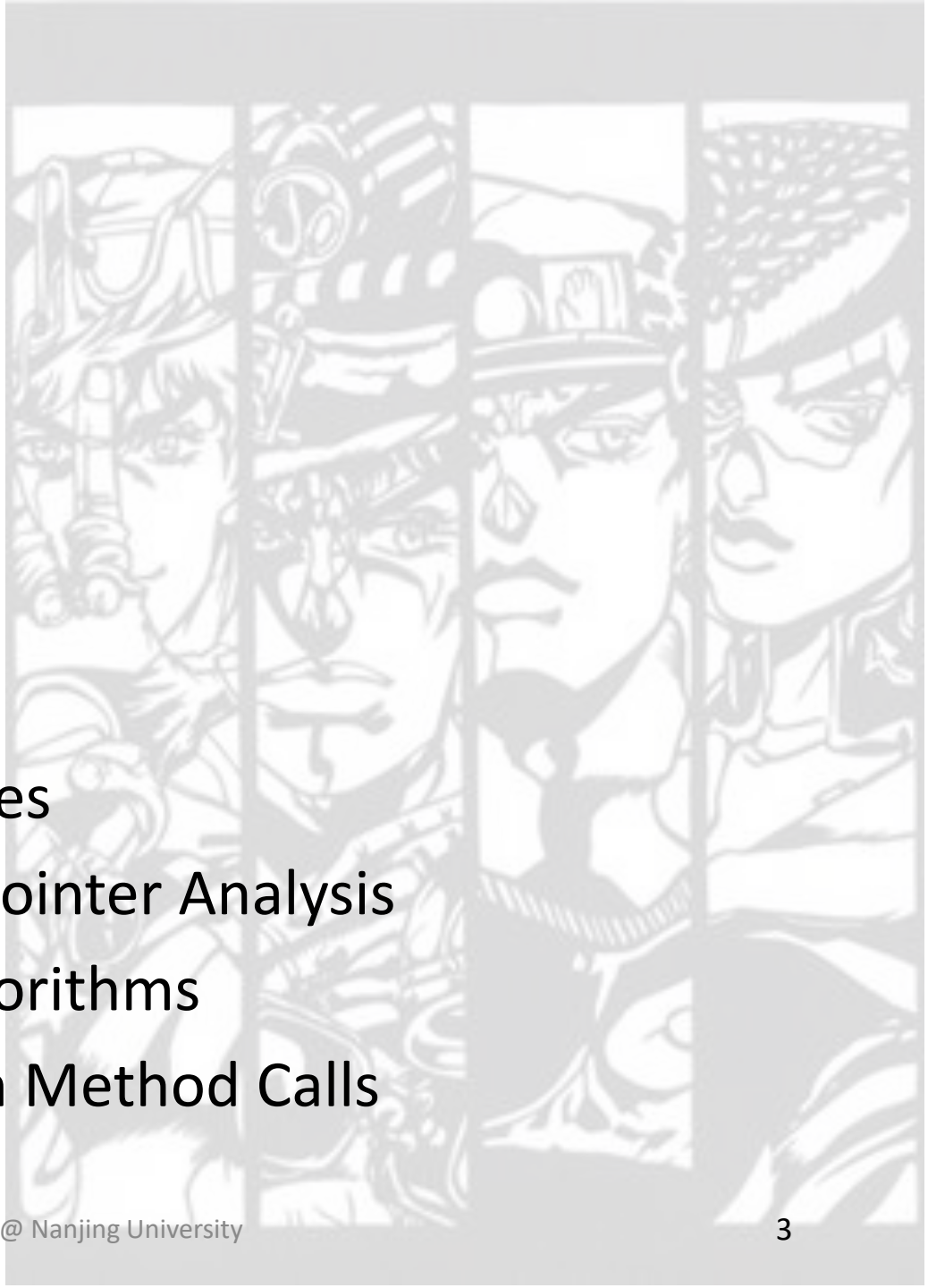
Nanjing University

Tian Tan

2020

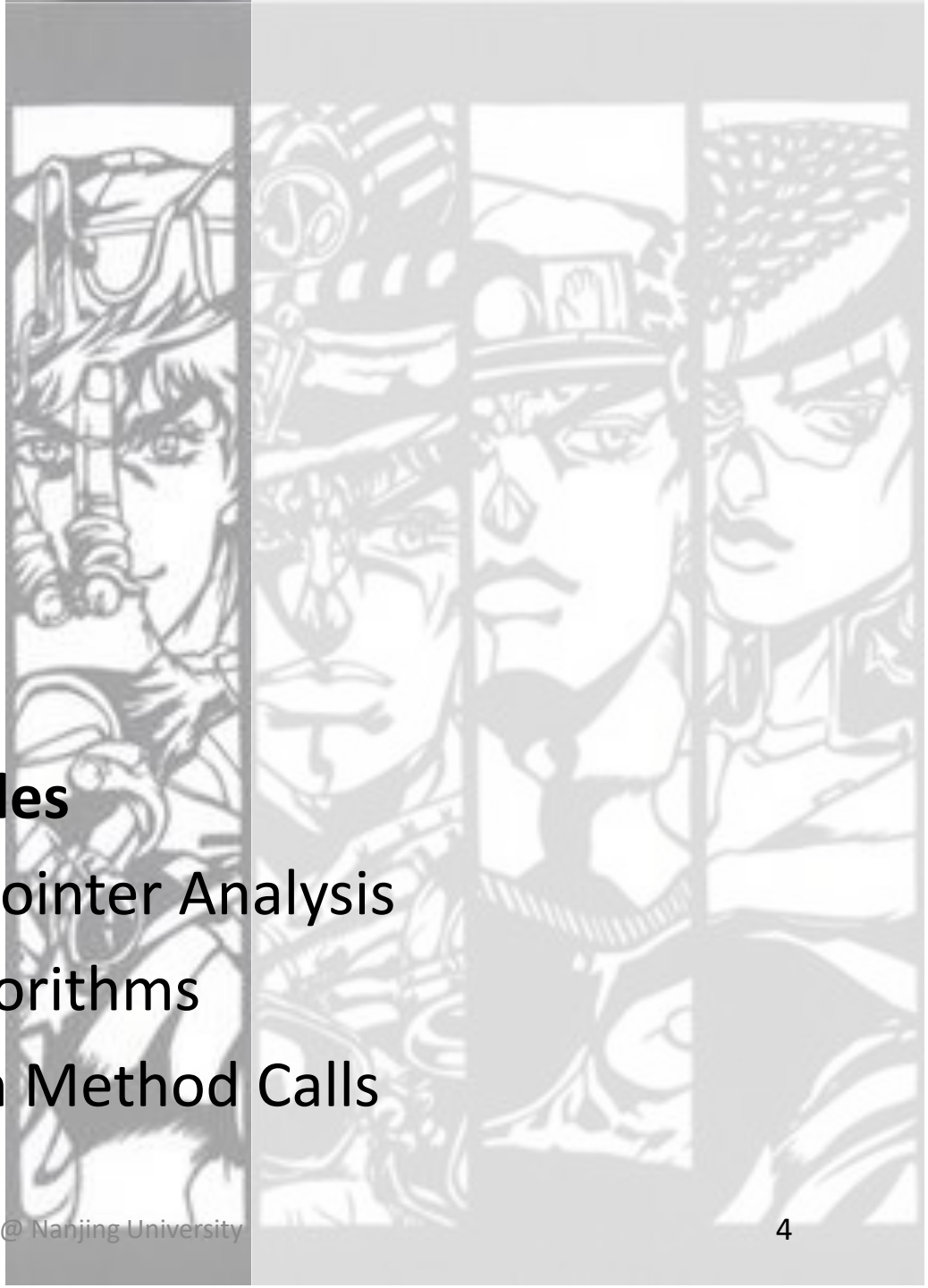
# Contents

1. Pointer Analysis: Rules
2. How to Implement Pointer Analysis
3. Pointer Analysis: Algorithms
4. Pointer Analysis with Method Calls





# Contents

- 1. Pointer Analysis: Rules**
2. How to Implement Pointer Analysis
3. Pointer Analysis: Algorithms
4. Pointer Analysis with Method Calls



# Pointer-Affecting Statements

New	<code>x = new T()</code>	 <p>First focus on these statements (suppose the program has just one method)</p>
Assign	<code>x = y</code>	
Store	<code>x.f = y</code>	
Load	<code>y = x.f</code>	
Call	<code>r = x.k(a, ...)</code>	 <p>Will come back to this in pointer analysis with method calls</p>

# Domains and Notations

**Variables:**  $x, y \in V$

**Fields:**  $f, g \in F$

**Objects:**  $o_i, o_j \in O$

**Instance fields:**  $o_i.f, o_j.g \in O \times F$

**Pointers:**  $\text{Pointer} = V \cup (O \times F)$

**Points-to relations:**  $pt : \text{Pointer} \rightarrow \mathcal{P}(O)$

- $\mathcal{P}(O)$  denotes the powerset of  $O$
- $pt(p)$  denotes the points-to set of  $p$

# Rules

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$

# Rules

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$ ← unconditional
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ ← premises ← conclusion
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$



# Rule: New

$$\overline{o_i \in pt(x)}$$

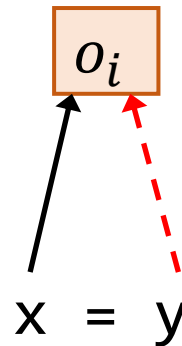
→ Conclusion

$$i: x = \text{new } T() \quad \uparrow \quad o_i$$

# Rule: Assign

$$\frac{o_i \in pt(y)}{o_i \in pt(x)}$$

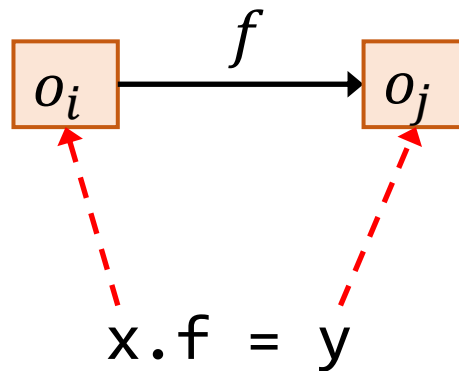
-----> Premises  
-----> Conclusion



# Rule: Store

$$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$$

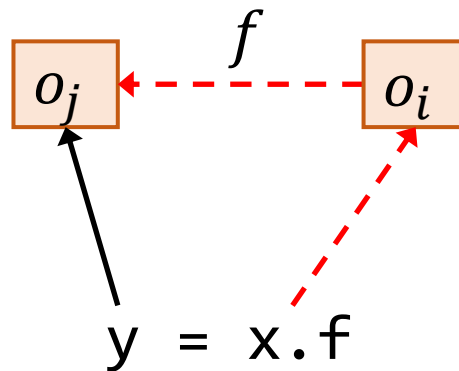
-----> Premises  
-----> Conclusion



# Rule: Load

$$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$$

-----> Premises  
-----> Conclusion



# Rules

-----> Premises

—————> Conclusion

Kind	Rule	Illustration
New	$\frac{}{o_i \in pt(x)}$	<p><math>i: x = \text{new } T()</math></p>
Assign	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	<p><math>x = y</math></p>
Store	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$	<p><math>x.f = y</math></p>
Load	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$	<p><math>y = x.f</math></p>

# Contents




1. Pointer Analysis: Rules
- 2. How to Implement Pointer Analysis**
3. Pointer Analysis: Algorithms
4. Pointer Analysis with Method Calls

# Our Pointer Analysis Algorithms

- A complete whole-program pointer analysis
- Carefully designed for understandability
- Easy to follow and implement

# How to Implement Pointer Analysis?




- Essentially, pointer analysis is to **propagate** points-to information among pointers (variables & fields)

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ 
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$ 
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$ 



# How to Implement Pointer Analysis?

- Essentially, pointer analysis is to **propagate** points-to information among pointers (variables & fields)

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ 
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$ 
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$ 




Pointer analysis as solving a system of **inclusion constraints** for pointers

Referred as *Andersen-style analysis*\*

\* Lars Ole Andersen, 1994. "*Program Analysis and Specialization for the C Programming Language*". Ph.D. Thesis. University of Copenhagen.

# How to Implement Pointer Analysis?

- Essentially, pointer analysis is to **propagate** points-to information among pointers (variables & fields)

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ 
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$ 
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$ 

Key to implementation: when  $pt(x)$  is **changed**,  
**propagate the changed part to the related pointers of  $x$**

# How to Implement Pointer Analysis?

- Essentially, pointer analysis is to **propagate** points-to information among pointers (variables & fields)

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$



## Solution

- We use a **graph** to connect related pointers
- When  $pt(x)$  changes, propagate the changed part to  $x$ 's **successors**

Key to implementation: when  $pt(x)$  is **changed**, propagate the **changed part** to the **related pointers** of  $x$

# Pointer Flow Graph (PFG)

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program.

# Pointer Flow Graph (PFG)

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program.

- Nodes:  $\text{Pointer} = V \cup (O \times F)$




A node  $n$  represents *a variable* or *a field of an abstract object*

- Edges:  $\text{Pointer} \times \text{Pointer}$

An edge  $x \rightarrow y$  means that the objects pointed by pointer  $x$  *may flow to* (and also be pointed to by) pointer  $y$



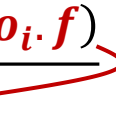
# Pointer Flow Graph: Edges

- PFG edges are added according to the statements of the program and the corresponding rules

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ 
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$ 
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$ 

# Pointer Flow Graph: Edges

- PFG edges are added according to the statements of the program and the corresponding rules

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\frac{}{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$ 	$x \leftarrow y$
Store	$x.f = y$	$\frac{o_i \in pt(x), o_j \in pt(y)}{o_j \in pt(o_i.f)}$ 	$o_i.f \leftarrow y$
Load	$y = x.f$	$\frac{o_i \in pt(x), o_j \in pt(o_i.f)}{o_j \in pt(y)}$ 	$y \leftarrow o_i.f$

# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①

c.f = a;    ②

d = c;      ③

c.f = d;    ④

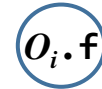
e = d.f;    ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node





# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

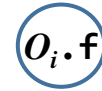
➔ **a = b;** ① ?  
c.f = a; ②  
d = c; ③  
c.f = d; ④  
e = d.f; ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node





# Pointer Flow Graph: An Example

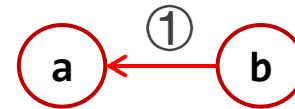
## Program

$(o_i \in pt(c), o_i \in pt(d))$

➔ **a = b;** ①  
c.f = a; ②  
d = c; ③  
c.f = d; ④  
e = d.f; ⑤

## Pointer flow graph

- Variable node 
- Instance field node 





# Pointer Flow Graph: An Example

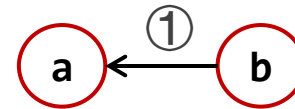
## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①  
➔ c.f = a;    ② ?  
d = c;      ③  
c.f = d;    ④  
e = d.f;    ⑤

## Pointer flow graph

- Variable node      
- Instance field node 





# Pointer Flow Graph: An Example

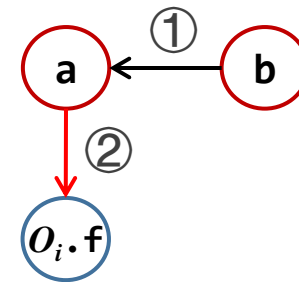
## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①  
➔ c.f = a;    ②  
d = c;      ③  
c.f = d;    ④  
e = d.f;    ⑤

## Pointer flow graph

- Variable node 
- Instance field node 



# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b; ①

c.f = a; ②

➔ d = c; ③

c.f = d; ④

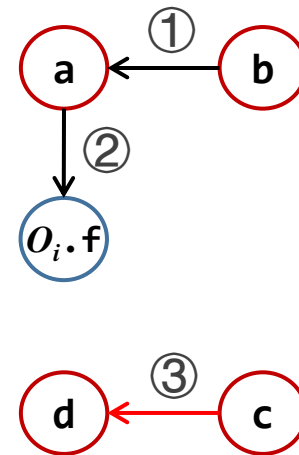
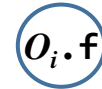
e = d.f; ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b; ①

c.f = a; ②

d = c; ③

➔ c.f = d; ④

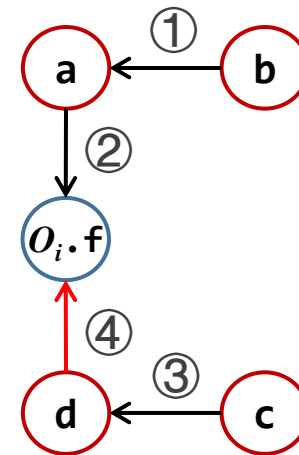
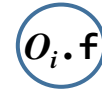
e = d.f; ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①



c.f = a;    ②

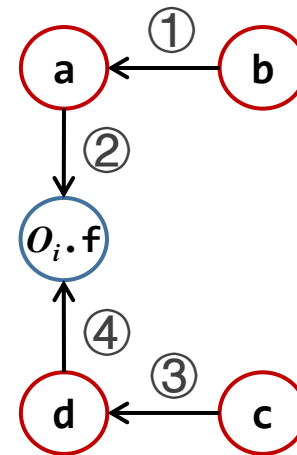
d = c;      ③

c.f = d;    ④

→ e = d.f; ⑤ ?

## Pointer flow graph

- Variable node  v
- Instance field node   $o_i.f$



# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①

c.f = a;    ②

d = c;      ③

c.f = d;    ④

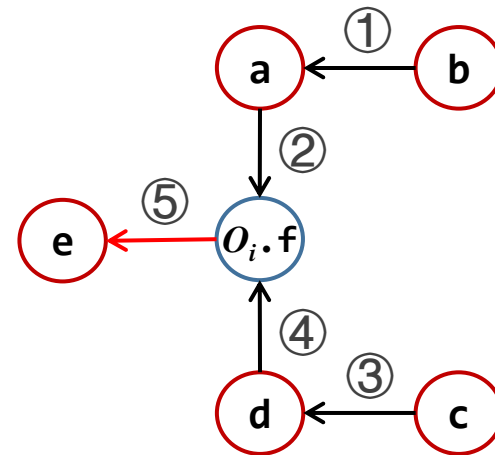
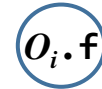
➔ e = d.f;  ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node





# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①

c.f = a;    ②

d = c;      ③

c.f = d;    ④

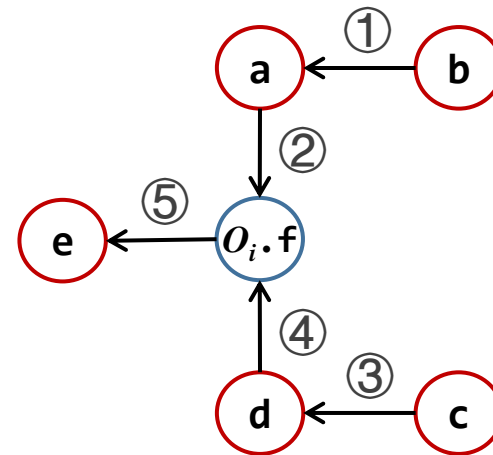
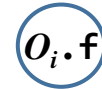
e = d.f;    ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①

c.f = a;    ②

d = c;      ③

c.f = d;    ④

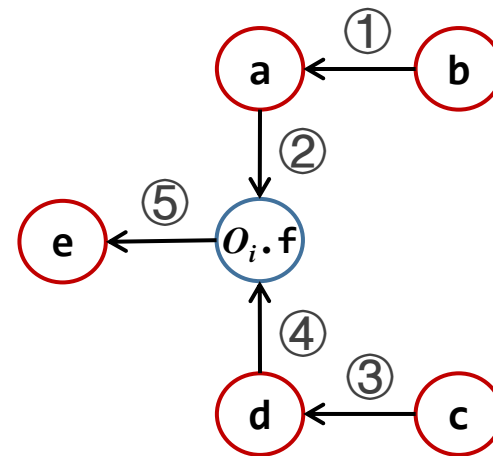
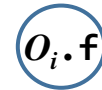
e = d.f;    ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



With PFG, pointer analysis can be solved by computing ***transitive closure*** of the PFG

# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

a = b;      ①

c.f = a;    ②

d = c;      ③

c.f = d;    ④

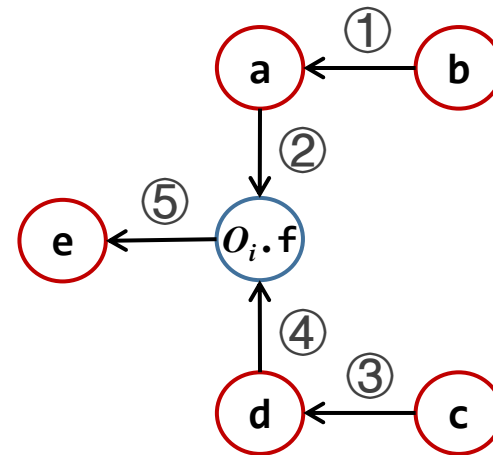
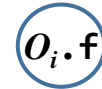
e = d.f;    ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



With PFG, pointer analysis can be solved by computing ***transitive closure*** of the PFG

E.g, **e** is reachable from **b** on the PFG, which means that the objects pointed by **b** may flow to and also be pointed by **e**

# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

➔  $j: b = new\ T();$

$a = b;$  ①

$c.f = a;$  ②

$d = c;$  ③

$c.f = d;$  ④

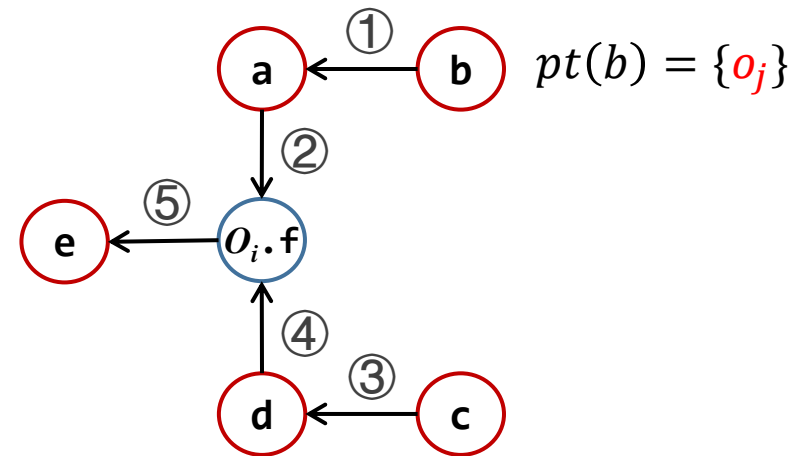
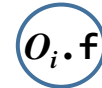
$e = d.f;$  ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



With PFG, pointer analysis can be solved by computing **transitive closure** of the PFG

E.g, **e** is reachable from **b** on the PFG, which means that the objects pointed by **b** may flow to and also be pointed by **e**

# Pointer Flow Graph: An Example

## Program

$(o_i \in pt(c), o_i \in pt(d))$

➔  $j: b = new T();$

$a = b;$  ①

$c.f = a;$  ②

$d = c;$  ③

$c.f = d;$  ④

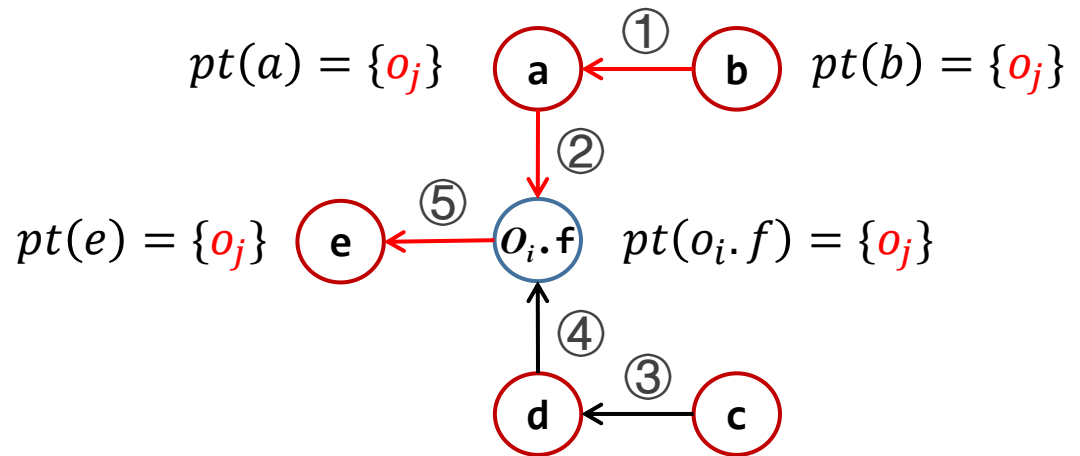
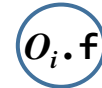
$e = d.f;$  ⑤

## Pointer flow graph

➤ Variable node



➤ Instance field node



With PFG, pointer analysis can be solved by computing **transitive closure** of the PFG

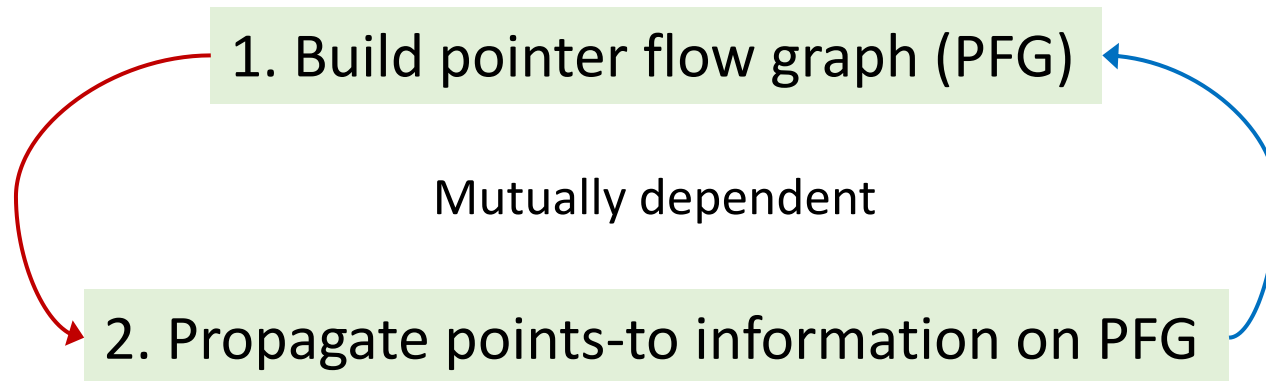
E.g, **e** is reachable from **b** on the PFG, which means that the objects pointed by **b** may flow to and also be pointed by **e**

# Implementing Pointer Analysis

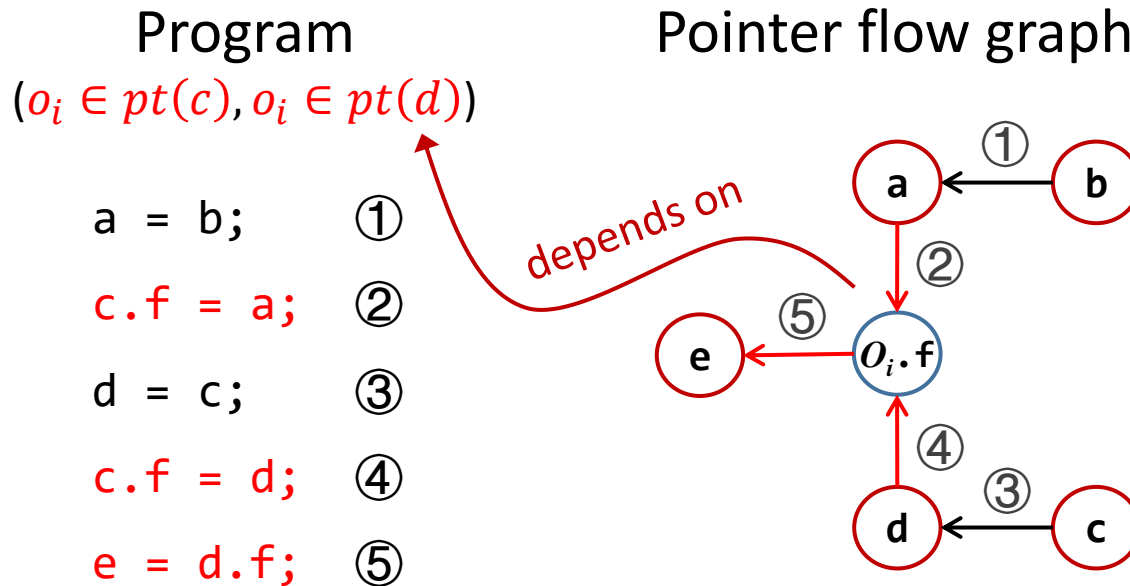
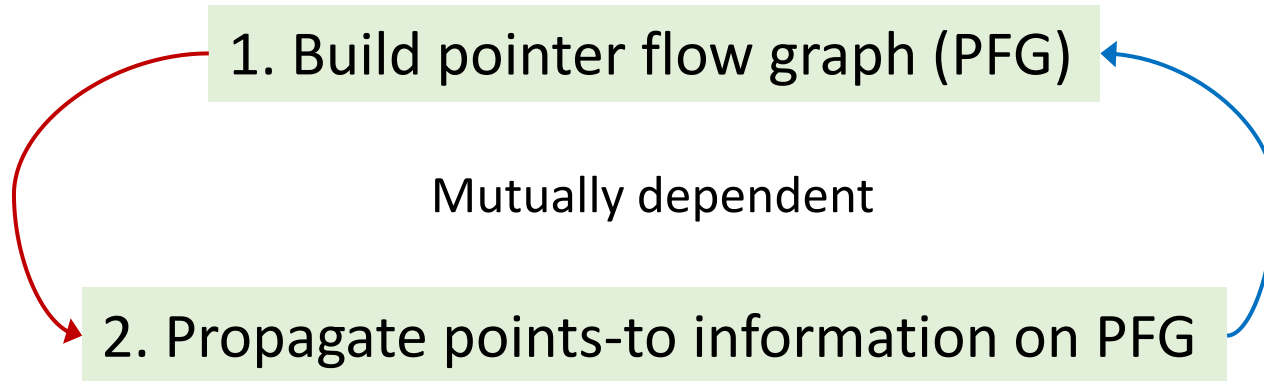
1. Build pointer flow graph (PFG)

2. Propagate points-to information on PFG

# Implementing Pointer Analysis

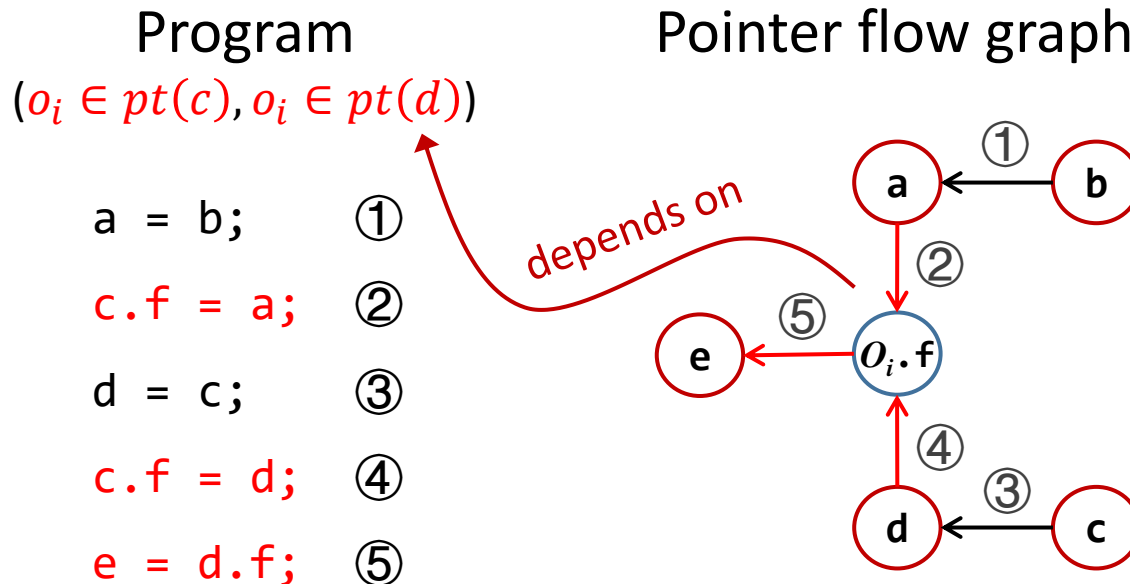
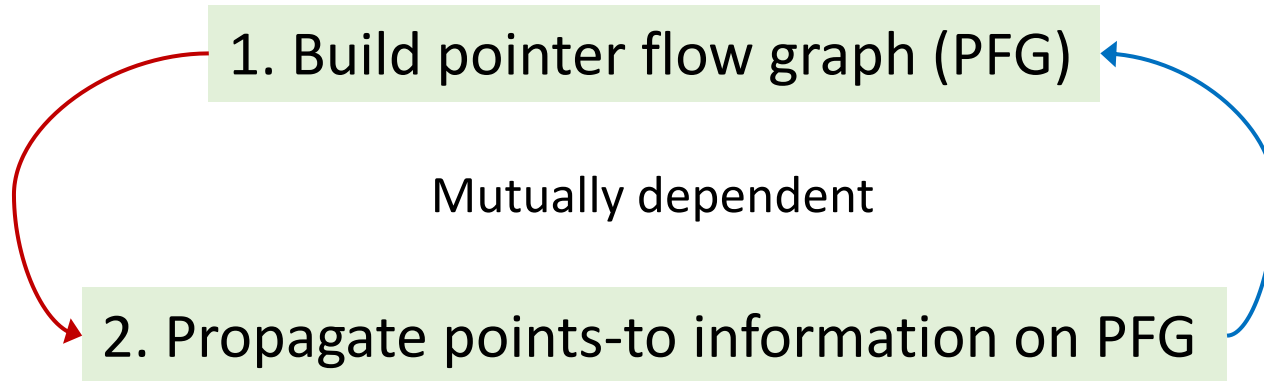


# Implementing Pointer Analysis





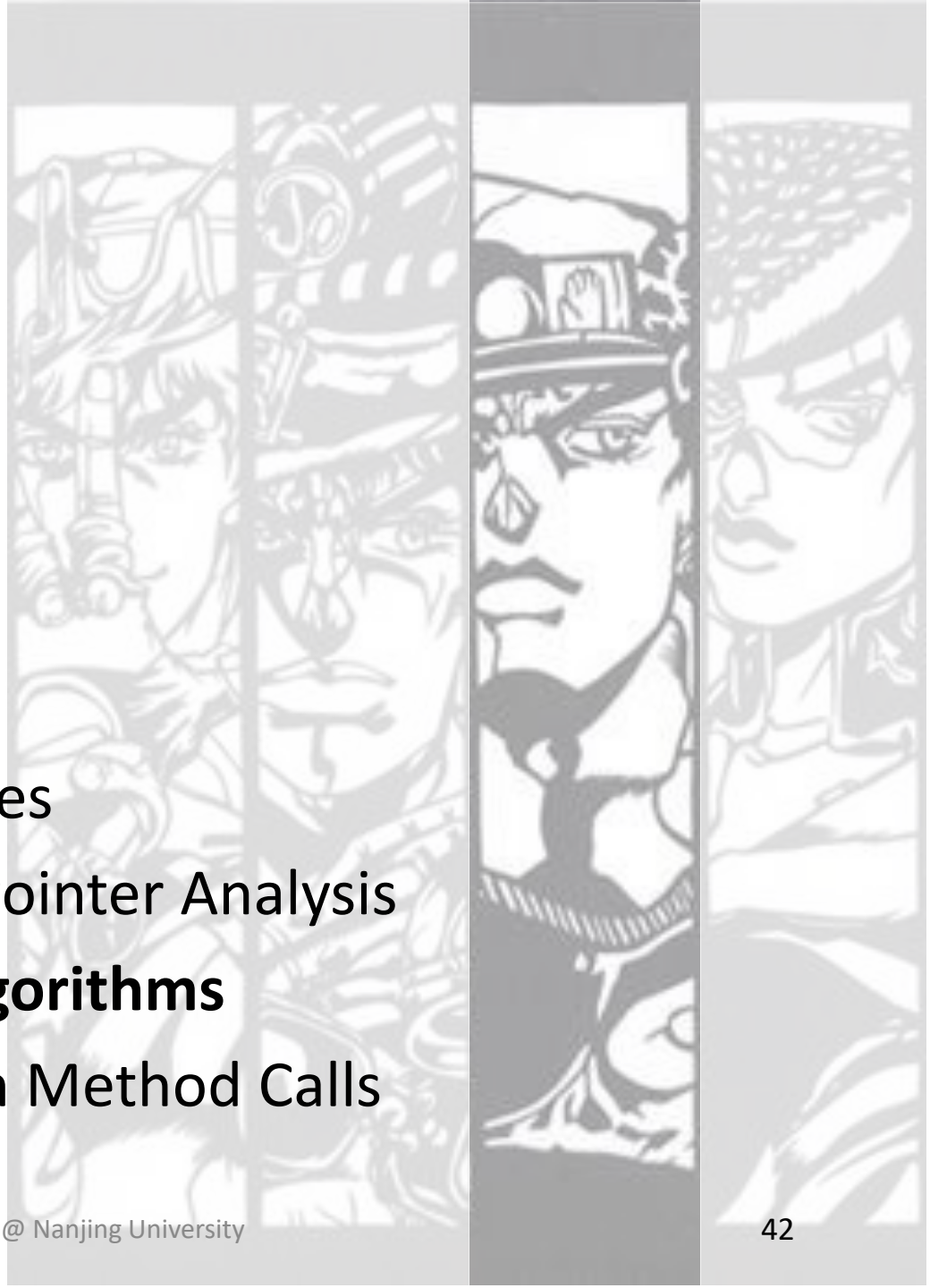
# Implementing Pointer Analysis



PFG is dynamically **updated** during pointer analysis

# Contents

1. Pointer Analysis: Rules
2. How to Implement Pointer Analysis
- 3. Pointer Analysis: Algorithms**
4. Pointer Analysis with Method Calls



# Pointer Analysis: Algorithms

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
    **AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**  
                **AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**  
                **AddEdge**( $o_i.f, y$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**

        add  $s \rightarrow t$  to  $PFG$

**if**  $pt(s)$  is not empty **then**  
        add  $\langle t, pt(s) \rangle$  to  $WL$

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**

$pt(n) \cup = pts$

**foreach**  $n \rightarrow s \in PFG$  **do**  
        add  $\langle s, pts \rangle$  to  $WL$

$S$      Set of statements of  
          the input program

$WL$     Work list

$PFG$    Pointer flow graph

# Pointer Analysis: Algorithms

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T()$   $\in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y$   $\in S$  **do**  
    **AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    **Propagate**( $n, \Delta$ )  
    **if**  $n$  represents a variable  $x$  **then**  
        **foreach**  $o_i \in \Delta$  **do**  
            **foreach**  $x.f = y$   $\in S$  **do**  
                **AddEdge**( $y, o_i.f$ )  
            **foreach**  $y = x.f$   $\in S$  **do**  
                **AddEdge**( $o_i.f, y$ )

main  
algorithm

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
    add  $s \rightarrow t$  to  $PFG$   
    **if**  $pt(s)$  is not empty **then**  
        add  $\langle t, pt(s) \rangle$  to  $WL$

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
     $pt(n) \cup = pts$   
    **foreach**  $n \rightarrow s \in PFG$  **do**  
        add  $\langle s, pts \rangle$  to  $WL$

$S$	Set of statements of the input program
$WL$	Work list
$PFG$	Pointer flow graph

# Pointer Analysis: Algorithms

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
    **AddEdge**( $y, x$ )

→ **while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    **Propagate**( $n, \Delta$ )  
    **if**  $n$  represents a variable  $x$  **then**  
        **foreach**  $o_i \in \Delta$  **do**  
            **foreach**  $x.f = y \in S$  **do**  
                **AddEdge**( $y, o_i.f$ )  
            **foreach**  $y = x.f \in S$  **do**  
                **AddEdge**( $o_i.f, y$ )

main  
algorithm

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
    add  $s \rightarrow t$  to  $PFG$   
    **if**  $pt(s)$  is not empty **then**  
        add  $\langle t, pt(s) \rangle$  to  $WL$

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
     $pt(n) \cup = pts$   
    **foreach**  $n \rightarrow s \in PFG$  **do**  
        add  $\langle s, pts \rangle$  to  $WL$

$S$	Set of statements of the input program
$WL$	Work list
$PFG$	Pointer flow graph

# Worklist ( $WL$ )

- Worklist contains the points-to information **to be processed**
  - $WL \subseteq \langle \text{Pointer}, \mathcal{P}(O) \rangle^*$
- Each worklist entry  $\langle n, pts \rangle$  is **a pair of pointer  $n$  and points-to set  $pts$** , which means that  $pts$  should be propagated to  $pt(n)$ 
  - E.g.,  $[\langle x, \{o_i\} \rangle, \langle y, \{o_j, o_k\} \rangle, \langle o_j \cdot f, \{o_l\} \rangle \dots]$

# Main Algorithm

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

Initialize the analysis

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$



# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

Initialize the analysis

Add assign edges to PFG

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then** ①  
 add  $s \rightarrow t$  to  $PFG$   
**if**  $pt(s)$  is not empty **then**  
 add  $\langle t, pt(s) \rangle$  to  $WL$

① Do nothing if  $s \rightarrow t$  is already in  $PFG$

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**

add  $s \rightarrow t$  to  $PFG$

**if**  $pt(s)$  is not empty **then**

add  $\langle t, pt(s) \rangle$  to  $WL$

①

②

① Do nothing if  $s \rightarrow t$  is already in PFG

② **Add PFG edge**

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

## Solve( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
 AddEdge( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
 Propagate( $n, \Delta$ )

## AddEdge( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then** ①  
 add  $s \rightarrow t$  to  $PFG$  ②  
**if**  $pt(s)$  is not empty **then** ③  
 add  $\langle t, pt(s) \rangle$  to  $WL$

- ① Do nothing if  $s \rightarrow t$  is already in PFG
- ② Add PFG edge
- ③ Ensure every object pointed by  $s$  is also pointed by  $t$

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\overline{o_i \in pt(y)}$ $\overline{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

Process the entries in  $WL$

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

remove  $\langle x, \{o_1, o_3\} \rangle$  from  $WL$

$pt(x) = \{o_1, o_2\}$

$\Delta = pts - pt(x)$

$= \{o_1, o_3\} - \{o_1, o_2\}$

$= \{o_3\}$

**Propagate**( $x, \{o_3\}$ )

Process the entries in  $WL$

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then** ①

$pt(n) \cup = pts$

**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

① Do nothing if  $pts$  is empty

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then** ①

$pt(n) \cup = pts$  ②

**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

① Do nothing if  $pts$  is empty

② Propagate  $pts$  to points-to set of  $n$

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$



# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
 add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then** ①

$pt(n) \cup = pts$  ②

**foreach**  $n \rightarrow s \in PFG$  **do** ③  
 add  $\langle s, pts \rangle$  to  $WL$

- ① Do nothing if  $pts$  is empty
- ② **Propagate  $pts$  to points-to set of  $n$**
- ③ **Propagate  $pts$  (the changed part) to  $n$ 's successors on PFG**

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\overline{o_i \in pt(y)}$ $\overline{o_i \in pt(x)}$	$x \leftarrow y$

# Differential Propagation

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  **Propagate**( $n, \Delta$ )

...

Why?

# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

...

**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
    Propagate( $n, \Delta$ )

...

# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

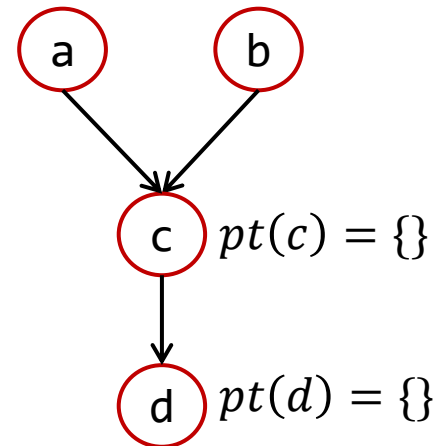
...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  Propagate( $n, \Delta$ )

...

$$pt(a) = \{o_1, o_2, o_3\}$$

$$pt(b) = \{o_1, o_3, o_5\}$$



PFG

# Differential Propagation

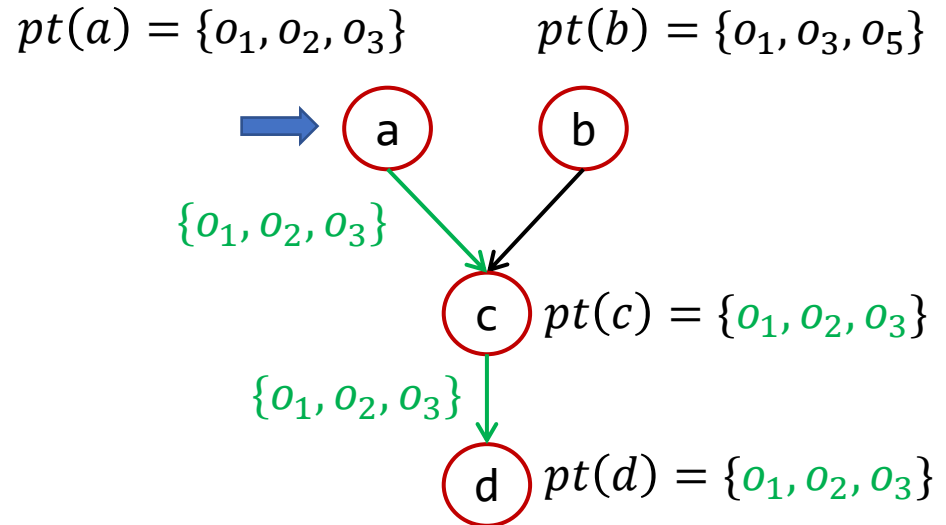
- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  Propagate( $n, \Delta$ )

...



PFG

# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

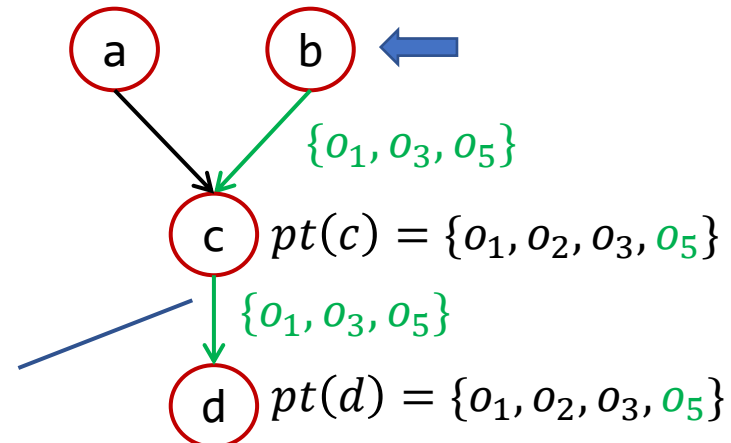
...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  Propagate( $n, \Delta$ )

...

$pt(a) = \{o_1, o_2, o_3\}$

$pt(b) = \{o_1, o_3, o_5\}$



Direct propagation:

PFG

# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

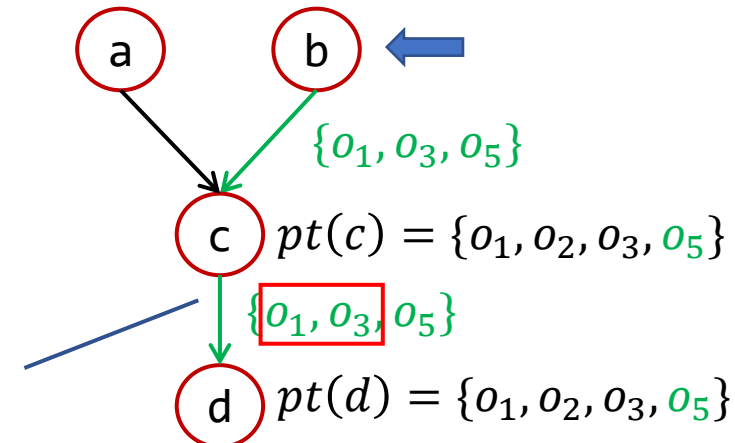
...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  Propagate( $n, \Delta$ )

...

$pt(a) = \{o_1, o_2, o_3\}$

$pt(b) = \{o_1, o_3, o_5\}$



Direct propagation:  
**redundant**

PFG

# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

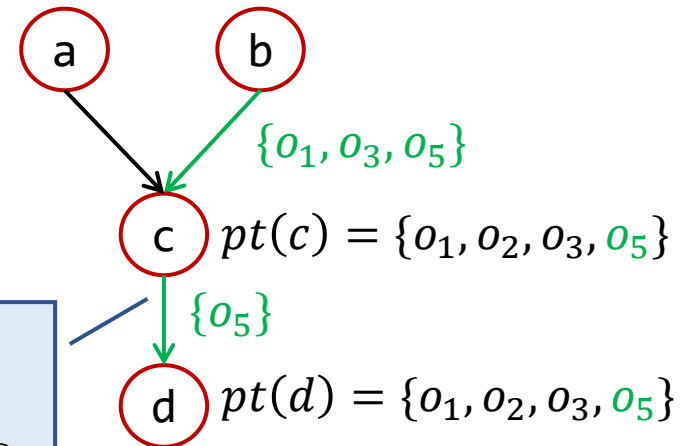
...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  **Propagate**( $n, \Delta$ )

...

$$pt(a) = \{o_1, o_2, o_3\}$$

$$pt(b) = \{o_1, o_3, o_5\}$$



PFG

In practice,  $\Delta$  is usually small compared with the original set, so propagating only the new points-to information ( $\Delta$ ) improves efficiency

Differential propagation:

$$\begin{aligned}\Delta &= pts - pt(c) \\ &= \{o_1, o_3, o_5\} - \{o_1, o_2, o_3\} \\ &= \{o_5\}\end{aligned}$$



# Differential Propagation

- **Differential propagation** is employed to avoid propagation and processing of redundant points-to information
- **Insight:** existing points-to information in  $pt(n)$  have already been propagated to  $n$ 's successors, and **no need** to be propagated again

Solve( $S$ )

...  
**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$   
  **Propagate**( $n, \Delta$ )

...

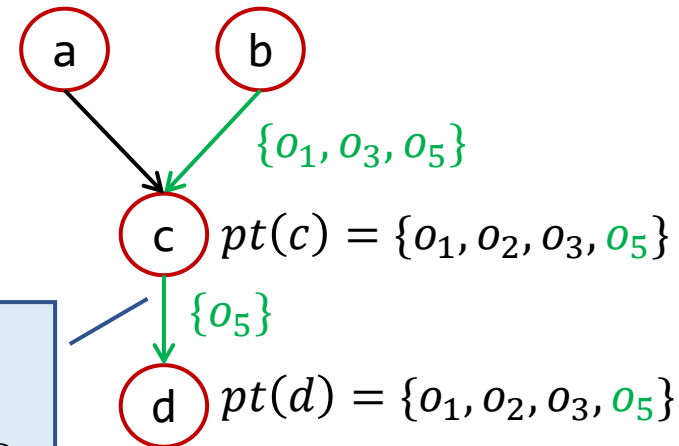
Besides,  $\Delta$  is also important for efficiency when handling stores, loads, and method calls, as explained later

Differential propagation:

$$\begin{aligned}\Delta &= pts - pt(c) \\ &= \{o_1, o_3, o_5\} - \{o_1, o_2, o_3\} \\ &= \{o_5\}\end{aligned}$$

$$pt(a) = \{o_1, o_2, o_3\}$$

$$pt(b) = \{o_1, o_3, o_5\}$$



PFG

# Handling of New and Assign

**Solve**( $S$ )

$WL = [], PFG = \{ \}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

Kind	Statement	Rule	PFG Edge
New	$i: x = \text{new } T()$	$\overline{o_i \in pt(x)}$	N/A
Assign	$x = y$	$\frac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$

# Handling of Store and Load

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**  
**foreach**  $o_i \in \Delta$  **do**  
**foreach**  $x.f = y \in S$  **do**  
**AddEdge**( $y, o_i.f$ )  
**foreach**  $y = x.f \in S$  **do**  
**AddEdge**( $o_i.f, y$ )

Kind	Statement	Rule	PFG Edge
Store	$x.f = y$	$\frac{o_i \in pt(\mathbf{x}), o_j \in pt(\mathbf{y})}{o_j \in pt(\mathbf{o_i.f})}$	$o_i.f \leftarrow y$
Load	$y = x.f$	$\frac{o_i \in pt(\mathbf{x}), o_j \in pt(\mathbf{o_i.f})}{o_j \in pt(\mathbf{y})}$	$y \leftarrow o_i.f$

# Handling of Store and Load

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**  
**foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
         **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
         **AddEdge**( $o_i.f, y$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
 add  $s \rightarrow t$  to  $PFG$   
**if**  $pt(s)$  is not empty **then**  
 add  $\langle t, pt(s) \rangle$  to  $WL$

New points-to information  
 may introduce new PFG edges

Kind	Statement	Rule	PFG Edge
Store	$x.f = y$	$\frac{o_i \in pt(\mathbf{x}), o_j \in pt(\mathbf{y})}{o_j \in pt(\mathbf{o_i.f})}$	$o_i.f \leftarrow y$
Load	$y = x.f$	$\frac{o_i \in pt(\mathbf{x}), o_j \in pt(\mathbf{o_i.f})}{o_j \in pt(\mathbf{y})}$	$y \leftarrow o_i.f$

# Algorithms: Review

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
    **AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    **Propagate**( $n, \Delta$ )  
    **if**  $n$  represents a variable  $x$  **then**  
        **foreach**  $o_i \in \Delta$  **do**  
            **foreach**  $x.f = y \in S$  **do**  
                **AddEdge**( $y, o_i.f$ )  
            **foreach**  $y = x.f \in S$  **do**  
                **AddEdge**( $o_i.f, y$ )

main  
algorithm

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
    add  $s \rightarrow t$  to  $PFG$   
    **if**  $pt(s)$  is not empty **then**  
        add  $\langle t, pt(s) \rangle$  to  $WL$

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
     $pt(n) \cup = pts$   
    **foreach**  $n \rightarrow s \in PFG$  **do**  
        add  $\langle s, pts \rangle$  to  $WL$

$S$	Set of statements of the input program
$WL$	Work list
$PFG$	Pointer flow graph

# An Example

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
    **AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**  
        **foreach**  $x.f = y \in S$  **do**  
            **AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**  
            **AddEdge**( $o_i.f, y$ )

$S$ :

```
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
```

$WL$ : []

$PFG$ :

# An Example

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

→ 1  $b = \text{new } C();$

2  $a = b;$

→ 3  $c = \text{new } C();$

$S:$  4  $c.f = a;$

5  $d = c;$

6  $c.f = d;$

7  $e = d.f;$

$WL: [\langle b, \{o_1\} \rangle, \langle c, \{o_3\} \rangle]$

$PFG:$

# An Example

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**

**AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

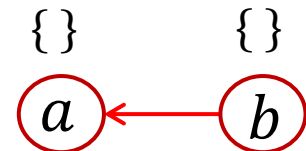
**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

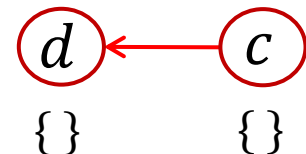
$S:$

```
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
```

$WL: [\langle b, \{o_1\} \rangle, \langle c, \{o_3\} \rangle]$



$PFG:$





# An Example

**Solve**( $S$ )

...

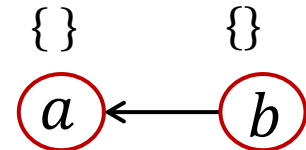
**while**  $WL$  is not empty **do**  
remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
  **foreach**  $o_i \in \Delta$  **do**  
    **foreach**  $x.f = y \in S$  **do**  
      **AddEdge**( $y, o_i.f$ )  
    **foreach**  $y = x.f \in S$  **do**  
      **AddEdge**( $o_i.f, y$ )

$S$ :

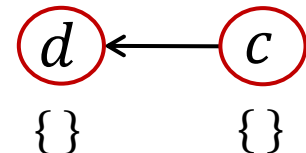
```
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
```

$WL$ :  $[\langle c, \{o_3\} \rangle]$

**Processing:**  $\langle b, \{o_1\} \rangle$



$PFG$ :



# An Example

**Solve**(*S*)

...

**while** *WL* is not empty **do**  
 remove  $\langle n, pts \rangle$  from *WL*  
 $\Delta = pts - pt(n)$   
**Propagate**(*n*,  $\Delta$ )  
**if** *n* represents a variable *x* **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**(*y*,  $o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f$ , *y*)

**Propagate**(*n*, *pts*)

**if** *pts* is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to *WL*

*S*:

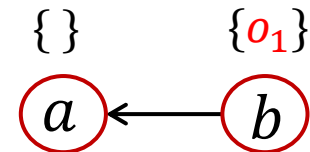
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

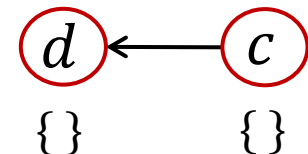
```

*WL*:  $[\langle c, \{o_3\} \rangle]$

**Processing**:  $\langle b, \{o_1\} \rangle$



*PFG*:



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
   add  $\langle s, pts \rangle$  to  $WL$

$S$ :

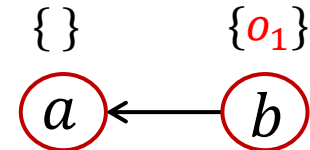
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

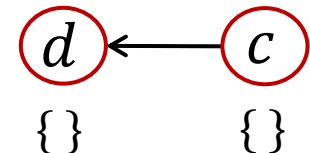
```

$WL$ : [ $\langle c, \{o_3\} \rangle, \langle a, \{o_1\} \rangle$ ]

**Processing**:  $\langle b, \{o_1\} \rangle$



$PFG$ :



# An Example

**Solve**(*S*)

...

**while** *WL* is not empty **do**  
 remove  $\langle n, pts \rangle$  from *WL*  
 $\Delta = pts - pt(n)$   
**Propagate**(*n*,  $\Delta$ )  
**if** *n* represents a variable *x* **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**(*y*,  $o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f$ , *y*)

**Propagate**(*n*, *pts*)

**if** *pts* is not empty **then**  
    $pt(n) \cup = pts$   
   **foreach**  $n \rightarrow s \in PFG$  **do**  
     add  $\langle s, pts \rangle$  to *WL*

*S*:

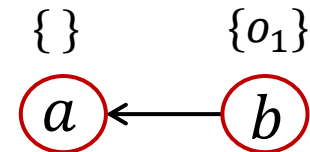
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

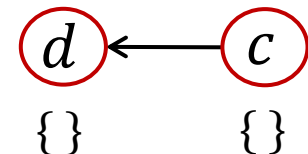
```

*WL*:  $[\langle a, \{o_1\} \rangle]$

**Processing:**  $\langle c, \{o_3\} \rangle$



*PFG*:



# An Example

**Solve**(*S*)

...

**while** *WL* is not empty **do**  
 remove  $\langle n, pts \rangle$  from *WL*  
 $\Delta = pts - pt(n)$   
**Propagate**(*n*,  $\Delta$ )  
**if** *n* represents a variable *x* **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**(*y*,  $o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f$ , *y*)

**Propagate**(*n*, *pts*)

**if** *pts* is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to *WL*

*S*:

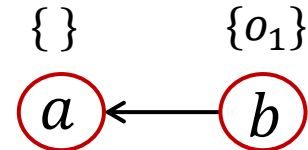
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

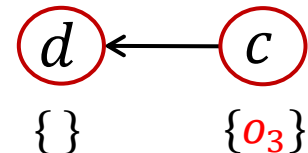
```

*WL*:  $\langle a, \{o_1\} \rangle$

**Processing**:  $\langle c, \{o_3\} \rangle$



*PFG*:



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
   add  $\langle s, pts \rangle$  to  $WL$

$S$ :

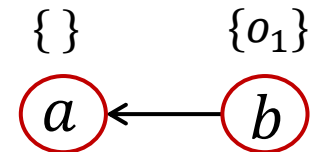
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

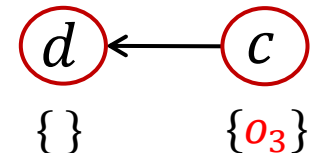
```

$WL$ :  $[\langle a, \{o_1\} \rangle, \langle d, \{o_3\} \rangle]$

**Processing**:  $\langle c, \{o_3\} \rangle$



$PFG$ :



# An Example

**Solve**( $S$ )

...

```
while  $WL$  is not empty do  
  remove  $\langle n, pts \rangle$  from  $WL$   
   $\Delta = pts - pt(n)$   
  Propagate( $n, \Delta$ )  
  if  $n$  represents a variable  $x$  then  
    foreach  $o_i \in \Delta$  do  
      foreach  $x.f = y \in S$  do  
        AddEdge( $y, o_i.f$ )  
      foreach  $y = x.f \in S$  do  
        AddEdge( $o_i.f, y$ )
```

**Propagate**( $n, pts$ )

```
if  $pts$  is not empty then  
   $pt(n) \cup = pts$   
  foreach  $n \rightarrow s \in PFG$  do  
    add  $\langle s, pts \rangle$  to  $WL$ 
```

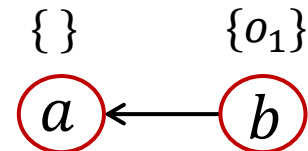
$S$ :

```
1  $b = \text{new } C();$   
2  $a = b;$   
3  $c = \text{new } C();$   
4  $c.f = a;$   
5  $d = c;$   
6  $c.f = d;$   
7  $e = d.f;$ 
```

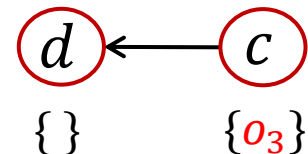
$WL$ :  $[\langle a, \{o_1\} \rangle, \langle d, \{o_3\} \rangle]$

**Processing**:  $\langle c, \{o_3\} \rangle$

What next?



$PFG$ :



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
 add  $s \rightarrow t$  to  $PFG$   
**if**  $pt(s)$  is not empty **then**  
 add  $\langle t, pt(s) \rangle$  to  $WL$

```

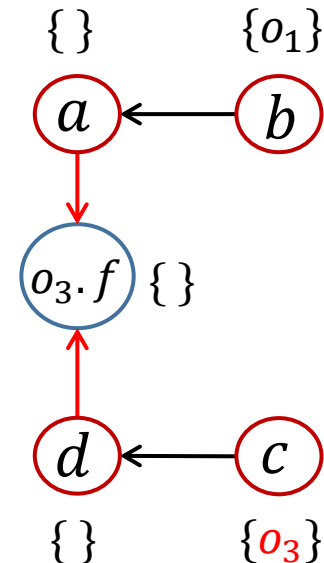
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
  
```

$S$ :  $\longrightarrow$

$\longrightarrow$

$WL$ :  $[\langle a, \{o_1\} \rangle, \langle d, \{o_3\} \rangle]$

**Processing**:  $\langle c, \{o_3\} \rangle$



$PFG$ :



# An Example

**Solve**(*S*)

...

**while** *WL* is not empty **do**  
 remove  $\langle n, pts \rangle$  from *WL*  
 $\Delta = pts - pt(n)$   
**Propagate**(*n*,  $\Delta$ )  
**if** *n* represents a variable *x* **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**(*y*,  $o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f$ , *y*)

**Propagate**(*n*, *pts*)

**if** *pts* is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to *WL*

*S*:

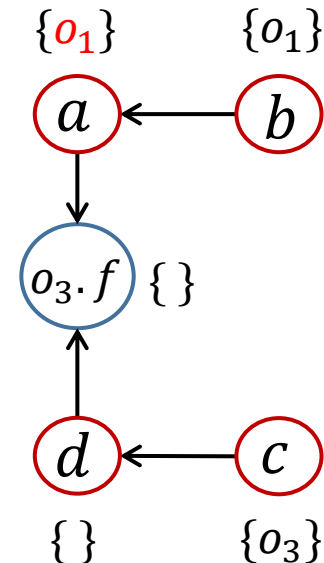
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

*WL*:  $[\langle d, \{o_3\} \rangle]$

**Processing**:  $\langle a, \{o_1\} \rangle$

*PFG*:



# An Example

**Solve**(*S*)

...

**while** *WL* is not empty **do**  
 remove  $\langle n, pts \rangle$  from *WL*  
 $\Delta = pts - pt(n)$   
**Propagate**(*n*,  $\Delta$ )  
**if** *n* represents a variable *x* **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**(*y*,  $o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f$ , *y*)

**Propagate**(*n*, *pts*)

**if** *pts* is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
   add  $\langle s, pts \rangle$  to *WL*

*S*:

```

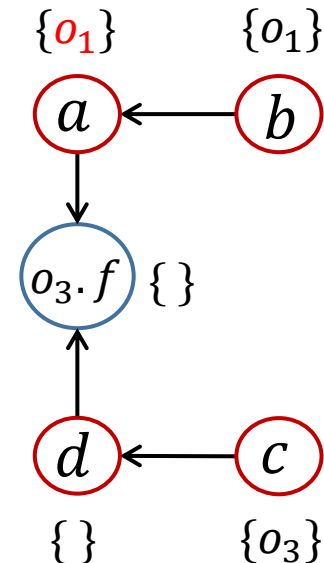
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

```

*WL*:  $[\langle d, \{o_3\} \rangle, \langle o_3.f, \{o_1\} \rangle]$

**Processing**:  $\langle a, \{o_1\} \rangle$

*PFG*:



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

$S$ :

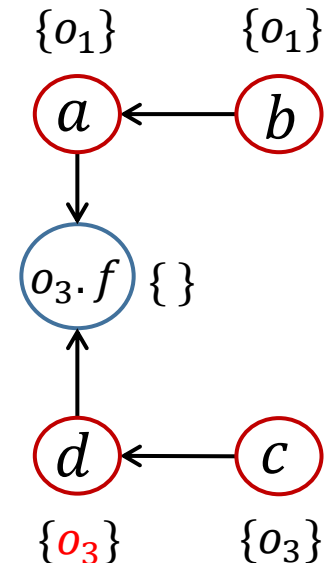
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

$WL$ :  $[\langle o_3.f, \{o_1\} \rangle]$

**Processing**:  $\langle d, \{o_3\} \rangle$

$PFG$ :



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

$S$ :

```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

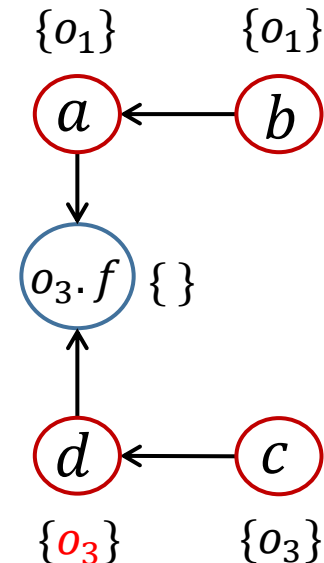
```

$WL$ :  $[\langle o_3.f, \{o_1\} \rangle]$

**Processing**:  $\langle d, \{o_3\} \rangle$

**What next?**

$PF$ G:



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
  remove  $\langle n, pts \rangle$  from  $WL$   
   $\Delta = pts - pt(n)$   
  **Propagate**( $n, \Delta$ )  
  **if**  $n$  represents a variable  $x$  **then**  
    **foreach**  $o_i \in \Delta$  **do**  
      **foreach**  $x.f = y \in S$  **do**  
        **AddEdge**( $y, o_i.f$ )  
      **foreach**  $y = x.f \in S$  **do**  
        **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
   $pt(n) \cup = pts$   
  **foreach**  $n \rightarrow s \in PFG$  **do**  
    add  $\langle s, pts \rangle$  to  $WL$

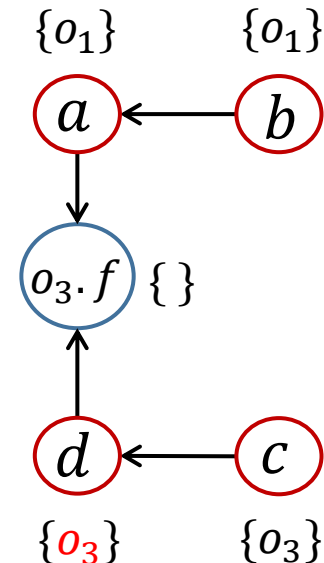
$S$ :

```
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
```

$WL$ :  $[\langle o_3.f, \{o_1\} \rangle, \langle o_3.f, \{o_3\} \rangle]$

**Processing**:  $\langle d, \{o_3\} \rangle$

$PFG$ :



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
   add  $\langle s, pts \rangle$  to  $WL$

$S$ :

```

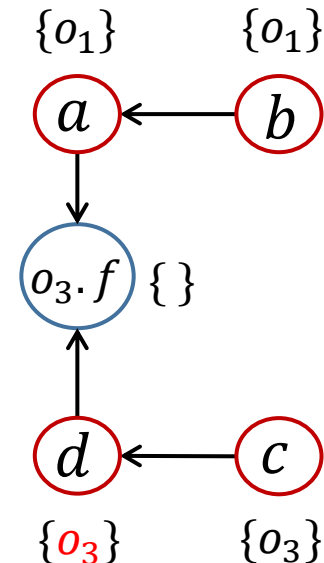
1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

$WL$ :  $[\langle o_3.f, \{o_1\} \rangle, \langle o_3.f, \{o_3\} \rangle]$

**Processing:**  $\langle d, \{o_3\} \rangle$

**What next?**

$PF$ G:



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
 add  $s \rightarrow t$  to  $PFG$   
**if**  $pt(s)$  is not empty **then**  
 add  $\langle t, pt(s) \rangle$  to  $WL$

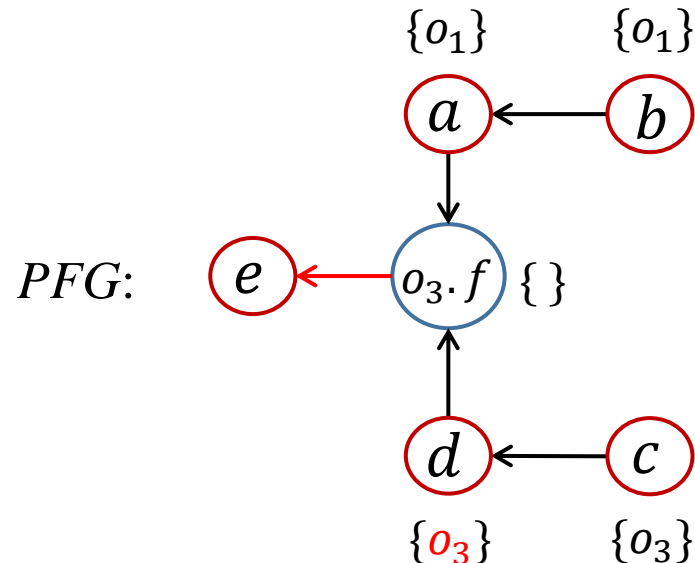
$S$ :

```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

$WL$ :  $\langle \langle o_3.f, \{o_1\} \rangle, \langle o_3.f, \{o_3\} \rangle \rangle$

**Processing**:  $\langle d, \{o_3\} \rangle$



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**

$pt(n) \cup = pts$

**foreach**  $n \rightarrow s \in PFG$  **do**

add  $\langle s, pts \rangle$  to  $WL$

$S$ :

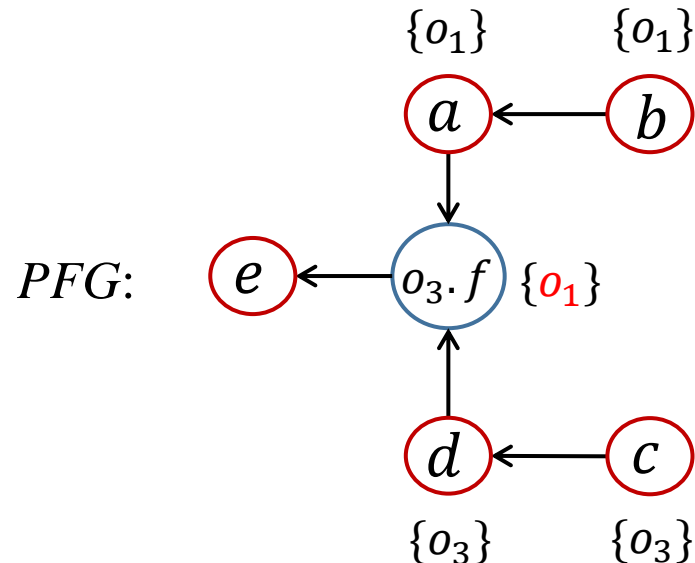
```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;

```

$WL$ :  $[\langle o_3.f, \{o_3\} \rangle, \langle e, \{o_1\} \rangle]$

**Processing**:  $\langle o_3.f, \{o_1\} \rangle$





# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

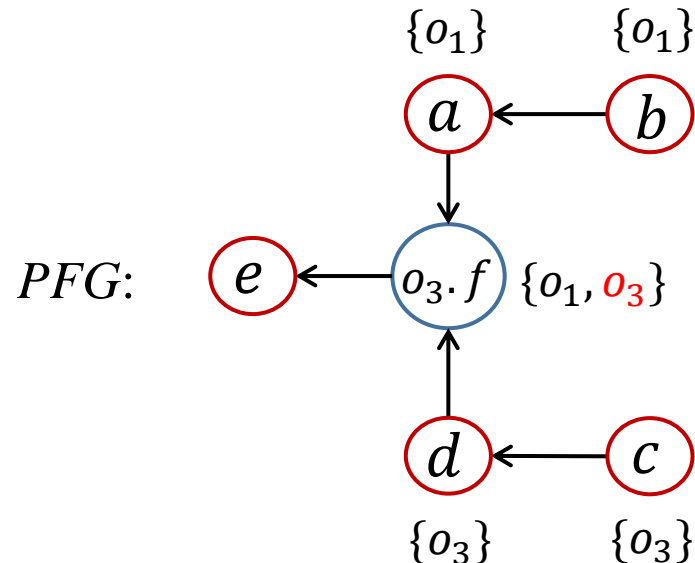
$S$ :

```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

$WL$ :  $\langle e, \{o_1\} \rangle, \langle e, \{o_3\} \rangle$

**Processing**:  $\langle o_3.f, \{o_3\} \rangle$



# An Example

**Solve**( $S$ )

...

**while**  $WL$  is not empty **do**  
 remove  $\langle n, pts \rangle$  from  $WL$   
 $\Delta = pts - pt(n)$   
**Propagate**( $n, \Delta$ )  
**if**  $n$  represents a variable  $x$  **then**  
   **foreach**  $o_i \in \Delta$  **do**  
     **foreach**  $x.f = y \in S$  **do**  
       **AddEdge**( $y, o_i.f$ )  
     **foreach**  $y = x.f \in S$  **do**  
       **AddEdge**( $o_i.f, y$ )

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
 $pt(n) \cup = pts$   
**foreach**  $n \rightarrow s \in PFG$  **do**  
 add  $\langle s, pts \rangle$  to  $WL$

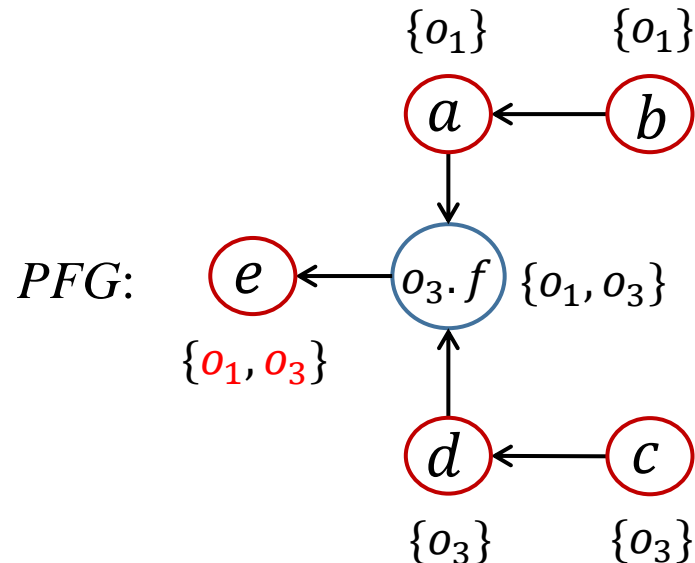
$S$ :

```

1 b = new C();
2 a = b;
3 c = new C();
4 c.f = a;
5 d = c;
6 c.f = d;
7 e = d.f;
    
```

$WL$ : []

**Processing:**  $\langle e, \{o_1\} \rangle, \langle e, \{o_3\} \rangle$



# An Example

**Solve**( $S$ )

...

```

while  $WL$  is not empty do
  remove  $\langle n, pts \rangle$  from  $WL$ 
   $\Delta = pts - pt(n)$ 
  Propagate( $n, \Delta$ )
  if  $n$  represents a variable  $x$  then
    foreach  $o_i \in \Delta$  do
      foreach  $x.f = y \in S$  do
        AddEdge( $y, o_i.f$ )
      foreach  $y = x.f \in S$  do
        AddEdge( $o_i.f, y$ )
  
```

**Propagate**( $n, pts$ )

```

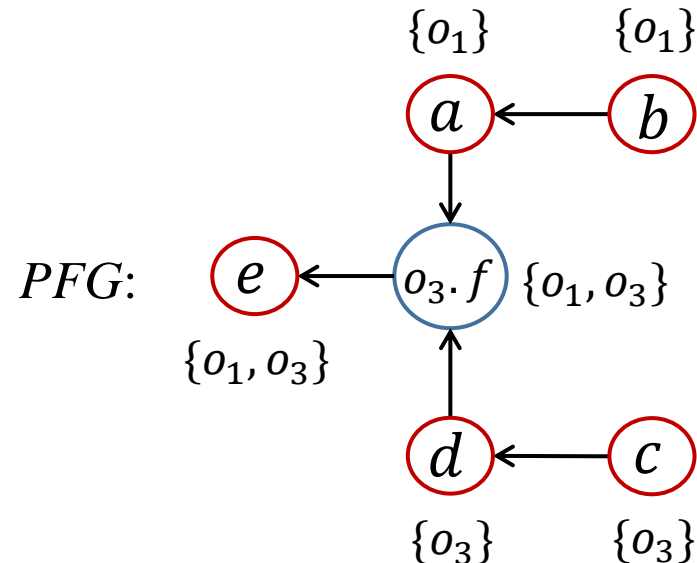
if  $pts$  is not empty then
   $pt(n) \cup = pts$ 
  foreach  $n \rightarrow s \in PFG$  do
    add  $\langle s, pts \rangle$  to  $WL$ 
  
```

$S$ :

```

1  $b = \text{new } C();$ 
2  $a = b;$ 
3  $c = \text{new } C();$ 
4  $c.f = a;$ 
5  $d = c;$ 
6  $c.f = d;$ 
7  $e = d.f;$ 
  
```

$WL$ : []



# Algorithms: Review

**Solve**( $S$ )

$WL = [], PFG = \{\}$

**foreach**  $i: x = \text{new } T() \in S$  **do**  
    add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S$  **do**  
    **AddEdge**( $y, x$ )

**while**  $WL$  is not empty **do**  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    **Propagate**( $n, \Delta$ )  
    **if**  $n$  represents a variable  $x$  **then**  
        **foreach**  $o_i \in \Delta$  **do**  
            **foreach**  $x.f = y \in S$  **do**  
                **AddEdge**( $y, o_i.f$ )  
            **foreach**  $y = x.f \in S$  **do**  
                **AddEdge**( $o_i.f, y$ )

main  
algorithm

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**  
    add  $s \rightarrow t$  to  $PFG$   
    **if**  $pt(s)$  is not empty **then**  
        add  $\langle t, pt(s) \rangle$  to  $WL$

**Propagate**( $n, pts$ )

**if**  $pts$  is not empty **then**  
     $pt(n) \cup = pts$   
    **foreach**  $n \rightarrow s \in PFG$  **do**  
        add  $\langle s, pts \rangle$  to  $WL$

$S$	Set of statements of the input program
$WL$	Work list
$PFG$	Pointer flow graph

# The X You Need To Understand in This Lecture

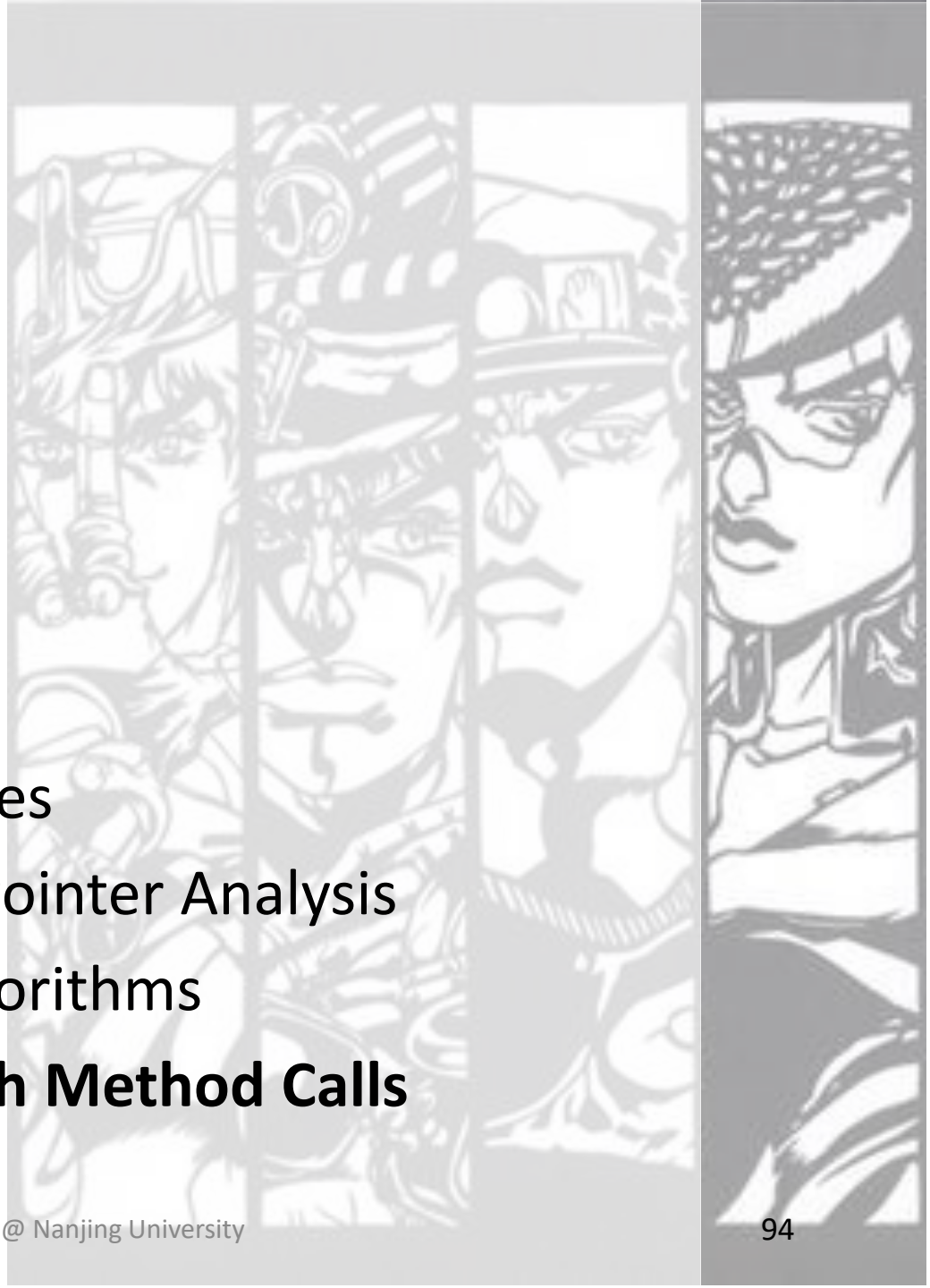
- Understand pointer analysis rules
- Understand pointer flow graph
- Understand pointer analysis algorithms

注意注意!  
划重点了!



# Contents

1. Pointer Analysis: Rules
2. How to Implement Pointer Analysis
3. Pointer Analysis: Algorithms
4. **Pointer Analysis with Method Calls**



# Pointer Analysis in the Presence of Method Invocations

- Inter-procedural pointer analysis requires call graph

```
void foo(A a) {  
    ...  
    b = a.bar();  
    ...  
}
```

*pt(a) = ???*  
*pt(b) = ???*

# Pointer Analysis in the Presence of Method Invocations

- Inter-procedural pointer analysis requires call graph

**CHA:** resolve call targets  
based on **declared type of a**

```
void foo(A a) {  
    ...  
    b = a.bar();  
    ...  
}
```

*pt(a) = ???*  
*pt(b) = ???*

- Call graph construction
  - CHA: imprecise, introduce spurious call graph edges and points-to relations



# Pointer Analysis in the Presence of Method Invocations

- Inter-procedural pointer analysis requires call graph

**CHA:** resolve call targets based on **declared type of a**

**Pointer analysis:** resolve call targets based on  $pt(a)$

```
void foo(A a) {  
    ...  
    b = a.bar();  
    ...  
}
```

$pt(a) = ???$

$pt(b) = ???$

- Call graph construction

- CHA: imprecise, introduce spurious call graph edges and points-to relations
- Pointer analysis: more precise than CHA, both for call graph and points-to relations

a.k.a **on-the-fly**  
call graph construction

# Rule: Call

Kind	Statement	Rule
Call	$L: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\  o_u \in pt(a_j), 1 \leq j \leq n \\  o_v \in pt(m_{ret}) \\  \hline  o_i \in pt(m_{this}) \\  o_u \in pt(m_{pj}), 1 \leq j \leq n \\  o_v \in pt(r)  \end{array}  $

# Rule: Call

Kind	Statement	Rule
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(\mathbf{x}), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$

- **Dispatch**( $o_i, k$ ): resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )

# Rule: Call

Kind	Statement	Rule
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(\mathbf{x}), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(\mathbf{m}_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$

# Rule: Call

Kind	Statement	Rule
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(\mathbf{x}), m = \text{Dispatch}(o_i, k)$ $o_u \in pt(a_j), 1 \leq j \leq n$ $o_v \in pt(m_{ret})}{o_i \in pt(\mathbf{m}_{this})}$ $o_u \in pt(m_{pj}), 1 \leq j \leq n$ $o_v \in pt(r)$

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$

```
C x = new T();
```

```
...
```

```
r = x.foo(a1, a2);
```

```
class T ... { ...
  B foo(A p1, A p2) {
    this...
    return ret;
  }}

```

# Rule: Call

Kind	Statement	Rule
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(\mathbf{a}_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this}) \quad o_u \in pt(\mathbf{m}_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)}$

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$
- $m_{pj}$ : the  **$j$ -th parameter** of  $m$

```
C x = new T();
```

```
...
r = x.foo(a1, a2);
```

```
class T ... { ...
  B foo(A p1, A p2) {
    this...
    return ret;
  }}

```

# Rule: Call

Kind	Statement	Rule	PFGE Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this}) \quad o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)}$	$a_1 \rightarrow m_{p1}$ $\dots$ $a_n \rightarrow m_{pn}$

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$
- $m_{pj}$ : the  **$j$ -th parameter** of  $m$

```
C x = new T();
```

```
...
r = x.foo(a1, a2);
```

```
class T ... {
  B foo(A p1, A p2) {
    this...
    return ret;
  }
}
```

# Rule: Call

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this}) \quad o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)}$	$a_1 \rightarrow m_{p1}$ $\dots$ $a_n \rightarrow m_{pn}$

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$
- $m_{pj}$ : the  **$j$ -th parameter** of  $m$
- $m_{ret}$ : the variable that holds the **return value** of  $m$

```
C x = new T();
```

```
...
r = x.foo(a1, a2);
```

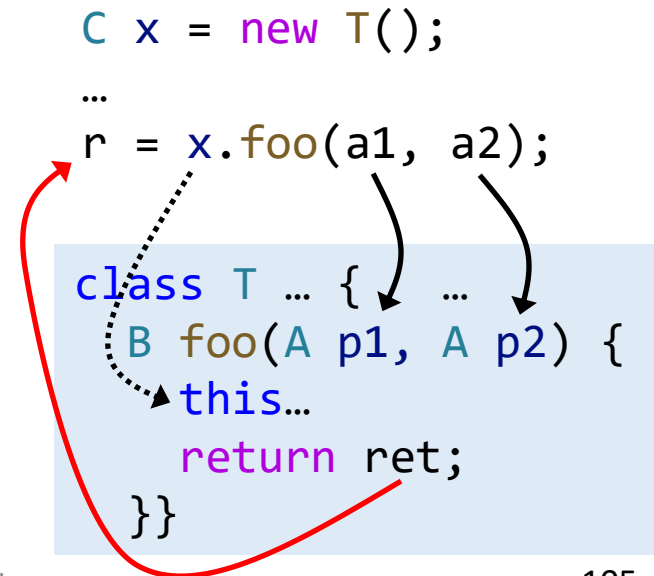
```
class T ... {
  B foo(A p1, A p2) {
    this...
    return ret;
  }
}
```



# Rule: Call

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this}) \quad o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)}$	$a_1 \rightarrow m_{p1} \quad \dots \quad a_n \rightarrow m_{pn} \quad r \leftarrow m_{ret}$

- **Dispatch**( $o_i, k$ ): resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$
- $m_{pj}$ : the  **$j$ -th parameter** of  $m$
- $m_{ret}$ : the variable that holds the **return value** of  $m$

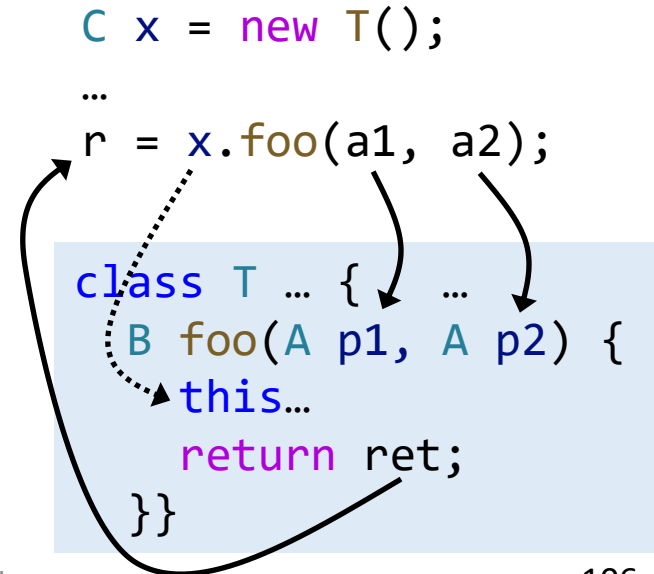


# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this}) \quad o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)}$	$a_1 \rightarrow m_{p1} \dots a_n \rightarrow m_{pn} \quad r \leftarrow m_{ret}$

- **Dispatch**( $o_i, k$ ): resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $m_{this}$ : **this variable** of  $m$
- $m_{pj}$ : the  **$j$ -th parameter** of  $m$
- $m_{ret}$ : the variable that holds the **return value** of  $m$



# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$L: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

$pt(x) = \{$   
     new A,  
     new B,  
     new C  
 $\}$

$x.foo();$

```
class A {
  T foo() {
    this...
  }
}
```

```
class B extends A {
  T foo() {
    this...?
  }
}
```

```
class C extends A {
  T foo() {
    this...
  }
}
```

# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$L: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

$pt(x) = \{$   
new A,  
new B,  
new C  
 $\}$

$x.foo();$

```
class A {
  T foo() {
    this... ?
  }
}
```

```
class B extends A {
  T foo() {
    this...
  }
}
```

```
class C extends A {
  T foo() {
    this...
  }
}
```

# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$  ?

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

Receiver object should only flow to **this** variable of the **corresponding** target method

PFG edge  $x \rightarrow m_{this}$  would introduce **spurious** points-to relations for **this** variables

```

{ new A,
pt(x) = new B,
  new C }

```

```
x.foo();
```

```

class A {
  T foo() {
    this...
  }
}

```

```

class B extends A {
  T foo() {
    this...
  }
}

```

```

class C extends A {
  T foo() {
    this...
  }
}

```

# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

Receiver object should only flow to **this** variable of the **corresponding** target method

PFG edge  $x \rightarrow m_{this}$  would introduce **spurious** points-to relations for **this** variables

$pt(x) = \{ \text{new A}, \text{new B}, \text{new C} \}$

$x.foo();$

→  
With  $x \rightarrow m_{this}$

```
class A {
  T foo() {
    this...
  }
}
```

```
class B extends A {
  T foo() {
    this...
  }
}
```

```
class C extends A {
  T foo() {
    this...
  }
}
```

# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$L: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

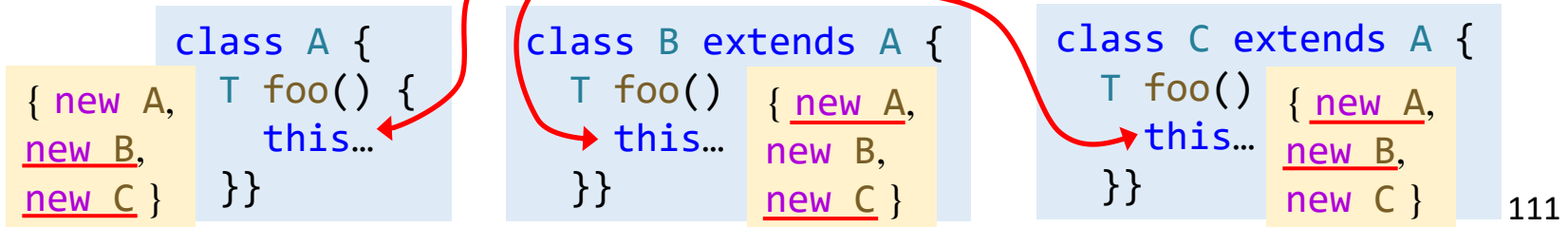
Receiver object should only flow to **this** variable of the **corresponding** target method

PFG edge  $x \rightarrow m_{this}$  would introduce **spurious** points-to relations for **this** variables

$pt(x) = \{ \text{new A}, \text{new B}, \text{new C} \}$

$x.foo();$

With  $x \rightarrow m_{this}$



# Rule: Call

Why not add PFG edge  $x \rightarrow m_{this}$ ?

Kind	Statement	Rule	PFG Edge
Call	$L: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} o_i \in pt(x), m = \text{Dispatch}(o_i, k) \\ o_u \in pt(a_j), 1 \leq j \leq n \\ o_v \in pt(m_{ret}) \end{array}}{\begin{array}{l} o_i \in pt(m_{this}) \\ o_u \in pt(m_{pj}), 1 \leq j \leq n \\ o_v \in pt(r) \end{array}}$	$\begin{array}{l} a_1 \rightarrow m_{p1} \\ \dots \\ a_n \rightarrow m_{pn} \\ r \leftarrow m_{ret} \end{array}$

Receiver object should only flow to **this** variable of the **corresponding** target method

PFG edge  $x \rightarrow m_{this}$  would introduce **spurious** points-to relations for **this** variables

$pt(x) = \{ \text{new A}, \text{new B}, \text{new C} \}$

$x.foo();$



Without  $x \rightarrow m_{this}$



```
class A {
  T foo() {
    { new A } this...
  }
}
```

```
class B extends A {
  T foo() {
    this... { new B }
  }
}
```

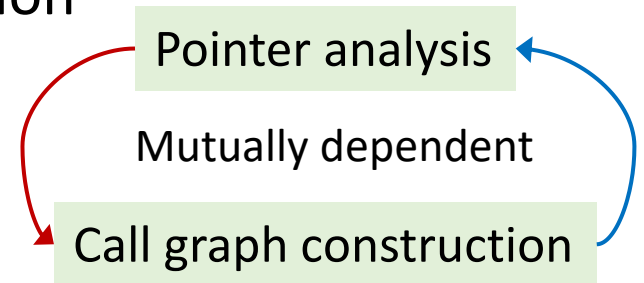
```
class C extends A {
  T foo() {
    this... { new C }
  }
}
```



# Interprocedural Pointer Analysis

- Run **together with** call graph construction

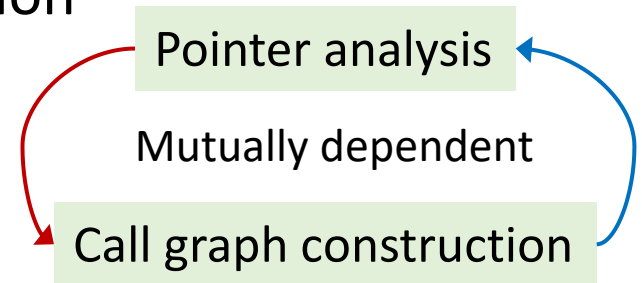
```
void foo(A a) {  
    ...  
    b = a.bar();  
    ...  
}
```



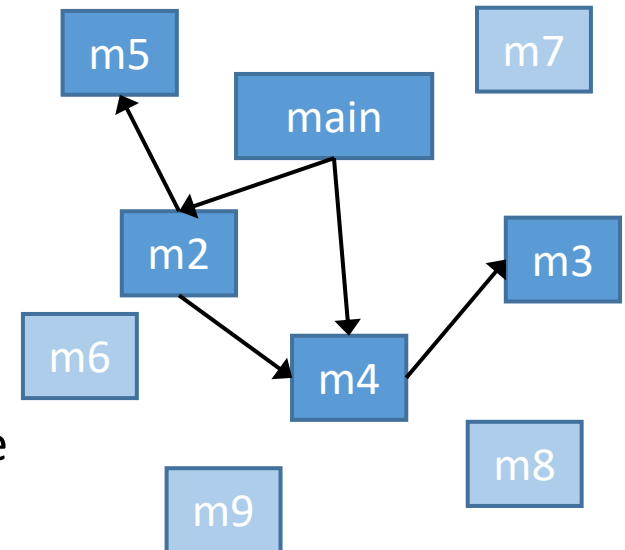
# Interprocedural Pointer Analysis

- Run **together with** call graph construction

```
void foo(A a) {  
    ...  
    b = a.bar();  
    ...  
}
```



- Call graph forms a “**reachable world**”
  - **Entry methods** (e.g., the main method) are reachable from the beginning
  - The other reachable methods are **gradually discovered** during analysis
  - Only **reachable** methods and statements are analyzed



# Algorithms

New part

**Solve**( $m^{entry}$ )

$WL=[ ], PFG=\{ \}, S=\{ \}, RM=\{ \}, CG=\{ \}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method</b> $m$
$RM$	Set of <b>reachable</b> methods
$CG$	Call graph <b>edges</b>

# AddReachable( $m$ )

- Expand the “**reachable world**”, called
  - at the beginning for entry methods
  - when new call graph edge is discovered

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

$S$  Set of **reachable** statements

$S_m$  Set of **statements in method**  $m$

$RM$  Set of **reachable** methods

# AddReachable( $m$ )

- Expand the “**reachable world**”, called
  - at the beginning for entry methods
  - when new call graph edge is discovered

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

Add new reachable method and statements

$S$  Set of **reachable** statements

$S_m$  Set of **statements in method**  $m$

$RM$  Set of **reachable** methods

# AddReachable( $m$ )

- Expand the “**reachable world**”, called
  - at the beginning for entry methods
  - when new call graph edge is discovered

**AddEdge**( $s, t$ )

**if**  $s \rightarrow t \notin PFG$  **then**

add  $s \rightarrow t$  to  $PFG$

**if**  $pt(s)$  is not empty **then**

add  $\langle t, pt(s) \rangle$  to  $WL$

(Same as before)

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

Add new reachable method and statements

Update worklist and PFG for new discovered statements

$S$  Set of **reachable** statements

$S_m$  Set of **statements in method  $m$**

$RM$  Set of **reachable** methods

# Algorithms

New part

**Solve**( $m^{entry}$ )

$WL=[ ], PFG=\{ \}, S=\{ \}, RM=\{ \}, CG=\{ \}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

$S$  Set of **reachable** statements

$S_m$  Set of **statements in method**  $m$

$RM$  Set of **reachable** methods

$CG$  Call graph **edges**

# ProcessCall( $x, o_i$ )

```

ProcessCall( $x, o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
    add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$ 
    if  $l \rightarrow m \notin CG$  then
      add  $l \rightarrow m$  to  $CG$ 
      AddReachable( $m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $a_i, p_i$ )
      AddEdge( $m_{ret}, r$ )
  
```

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this})}$ $o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)$	$a_1 \rightarrow m_{p1}$ $\dots$ $a_n \rightarrow m_{pn}$ $r \leftarrow m_{ret}$



# ProcessCall( $x, o_i$ )

```

ProcessCall( $x, o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
    add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$ 
    if  $l \rightarrow m \notin CG$  then
      add  $l \rightarrow m$  to  $CG$ 
      AddReachable( $m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $a_i, p_i$ )
        AddEdge( $m_{ret}, r$ )
  
```

Pass receiver object to *this* variable

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this})}$ $o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)$	$a_1 \rightarrow m_{p_1}$ $\dots$ $a_n \rightarrow m_{p_n}$ $r \leftarrow m_{ret}$

# ProcessCall( $x, o_i$ )

ProcessCall( $x, o_i$ )

foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

if  $l \rightarrow m \notin CG$  then

add  $l \rightarrow m$  to  $CG$

AddReachable( $m$ )

foreach parameter  $p_i$  of  $m$  do

AddEdge( $a_i, p_i$ )

AddEdge( $m_{ret}, r$ )

Pass receiver object to *this* variable

Construct call graph on the fly

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this})}$ $o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)$	$a_1 \rightarrow m_{p1}$ $\dots$ $a_n \rightarrow m_{pn}$ $r \leftarrow m_{ret}$

# ProcessCall( $x, o_i$ )

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

Pass receiver object to *this* variable

Construct call graph on the fly

Pass arguments

Pass return values

Kind	Statement	Rule	PFG Edge
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{o_i \in pt(x), m = \text{Dispatch}(o_i, k) \quad o_u \in pt(a_j), 1 \leq j \leq n \quad o_v \in pt(m_{ret})}{o_i \in pt(m_{this})}$ $o_u \in pt(m_{pj}), 1 \leq j \leq n \quad o_v \in pt(r)$	$a_1 \rightarrow m_{p1}$ $\dots$ $a_n \rightarrow m_{pn}$ $r \leftarrow m_{ret}$

# Algorithms Output:

Points-to Relations (*pt*)  
Call Graph (*CG*)

**Solve**( $m^{entry}$ )

$WL=[ ], PFG=\{ \}, S=\{ \}, RM=\{ \}, CG=\{ \}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

$S$  Set of **reachable** statements

$S_m$  Set of **statements in method**  $m$

$RM$  Set of **reachable** methods

$CG$  Call graph **edges**

# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

# An Example

**Solve**( $m^{entry}$ )

→  $WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

$WL:$  []

$RM:$  {}

$CG:$  {}

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$PFG:$

# An Example

**Solve**( $m^{entry}$ )

$WL=[]$ ,  $PFG=\{\}$ ,  $S=\{\}$ ,  $RM=\{\}$ ,  $CG=\{\}$

→ **AddReachable**( $m^{entry}$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

$WL:$  []

$RM:$  {}

$CG:$  {}

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$PFG:$

# An Example

**Solve**( $m^{entry}$ )

$WL=[]$ ,  $PFG=\{\}$ ,  $S=\{\}$ ,  $RM=\{\}$ ,  $CG=\{\}$

**AddReachable**( $m^{entry}$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

→ **add**  $m$  **to**  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

**add**  $\langle x, \{o_i\} \rangle$  **to**  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$WL:$  []

$RM:$  { **A.main()** }

$CG:$  {}

$PFG:$



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

→ add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

**WL:**  $[\langle a, \{o_3\} \rangle, \langle b, \{o_4\} \rangle]$

**RM:**  $\{ A.main() \}$

**CG:**  $\{\}$

**PFG:**

# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

→ remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $[\langle b, \{o_4\} \rangle]$

**RM:**  $\{ A.main() \}$

**Processing:**

$\langle a, \{o_3\} \rangle$

**CG:**  $\{\}$

**PFG:**

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $[\langle b, \{o_4\} \rangle]$

**RM:**  $\{ A.main() \}$

**Processing:**

$\langle a, \{o_3\} \rangle$

**CG:**  $\{\}$

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```

$\{o_3\}$

$\textcircled{a}$

# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

→ remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5          A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

$\{o_3\}$

$a$

**WL:**  $[\ ]$

**RM:**  $\{ A.main() \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{\ \}$

**PFG:**

# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$\{o_4\}$      $\{o_3\}$   
 $\textcircled{b}$      $\textcircled{a}$

**WL:** []

**RM:** { A.main() }

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:** {}

**PFG:**

# An Example

**Solve**( $m^{entry}$ )

$WL=[]$ ,  $PFG=\{\}$ ,  $S=\{\}$ ,  $RM=\{\}$ ,  $CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

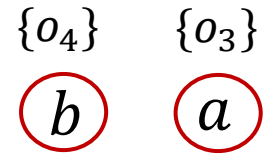
**ProcessCall**( $x, o_i$ )

What next?

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



**WL:** []

**RM:** { A.main() }

**Processing:**  
 $\langle b, \{o_4\} \rangle$

**CG:** {}

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )


**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )



**AddEdge**( $m_{ret}, r$ )

 **ProcessCall**( $x, o_i$ )

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5   A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

$\{o_4\}$      $\{o_3\}$   
    

**WL:** []

**RM:** { A.main() }

**Processing:**  
 $\langle b, \{o_4\} \rangle$

**CG:** {}

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$\rightarrow m = \text{Dispatch}(o_i, k)$   $m = ?$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5     }
6     A foo(A x) { ... }
7 }
8
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$\{o_4\}$      $\{o_3\}$   
b    a

**WL:** []

**RM:** { A.main() }

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:** {}

**PFG:**



# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$\rightarrow m = \text{Dispatch}(o_i, k)$   $m = B.foo(A)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

**WL:** []

**RM:** { A.main() }

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:** {}

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5     }
6     A foo(A x) { ... }
7 }
8
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```

$\{o_4\}$      $\{o_3\}$   
b    a

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

→ add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```
1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5     → A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }
```

$\{o_4\}$

$\{o_3\}$

$\textcircled{b}$

$\textcircled{a}$

**WL:** [ $\langle B.foo/this, \{o_4\} \rangle$ ]

**RM:** { A.main() }

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:** { }

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

→ add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5  →   A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

$\{o_4\}$

$\{o_3\}$

$b$

$a$

**WL:**  $\langle \langle B.\text{foo}/\text{this}, \{o_4\} \rangle \rangle$

**RM:**  $\{ A.\text{main}() \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.\text{foo}(A) \}$

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

→ add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )


**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5  →   A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

$\{o_4\}$      $\{o_3\}$   


**WL:**  $\langle \langle B.\text{foo}/\text{this}, \{o_4\} \rangle \rangle$

**RM:**  $\{ A.\text{main}() \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.\text{foo}(A) \}$

**PFG:**

**CHA:**  $\{ 5 \rightarrow B.\text{foo}(A), \underline{5 \rightarrow A.\text{foo}(A)} \}$

Spurious call edge

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

→ **AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

What change?

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```

$\{o_4\}$

$\{o_3\}$

$\circ b$

$\circ a$

**WL:**  $[\langle B.\text{foo}/\text{this}, \{o_4\} \rangle]$

**RM:**  $\{ A.\text{main} \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.\text{foo}(A) \}$

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

→ **AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $\langle \text{B.foo/this}, \{o_4\} \rangle$

**RM:**  $\{ \text{A.main} \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow \text{B.foo(A)} \}$

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5  →   A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

→ **AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $\langle \text{B.foo/this}, \{o_4\} \rangle$

**RM:**  $\{ \text{A.main() B.foo(A)} \}$

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow \text{B.foo(A)} \}$

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5     → A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

→ add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new T}() \in S_m$  **do**

add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

→ **AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $\langle \langle B.foo/this, \{o_4\} \rangle, \langle r, \{o_{11}\} \rangle \rangle$

**RM:**  $\{ A.main() B.foo(A) \}$  **foreach**  $i: x = \text{new } T() \in S_m$  **do**

→ add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )

**Processing:**

$\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5     → A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```

**AddReachable**( $m$ )

**if**  $m \notin RM$  **then**

add  $m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

→ add  $\langle x, \{o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $y, x$ )



# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

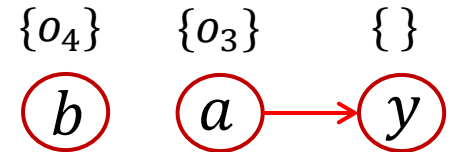
**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```

1  class A {
2      static void main() {
3          A a = new A();
4          A b = new B();
5  →   A c = b.foo(a);
6      }
7      A foo(A x) { ... }
8  }
9  class B extends A {
10     A foo(A y) {
11         A r = new A();
12         return r;
13     }
14 }

```



**WL:**  $[\langle B.foo/this, \{o_4\} \rangle,$   
 $\langle r, \{o_{11}\} \rangle, \langle y, \{o_3\} \rangle]$

**RM:**  $\{ A.main() B.foo(A) \}$

**Processing:**  
 $\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

**PFG:**

# An Example

**ProcessCall**( $x, o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

add  $\langle m_{this}, \{o_i\} \rangle$  to  $WL$

**if**  $l \rightarrow m \notin CG$  **then**

add  $l \rightarrow m$  to  $CG$

**AddReachable**( $m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $a_i, p_i$ )

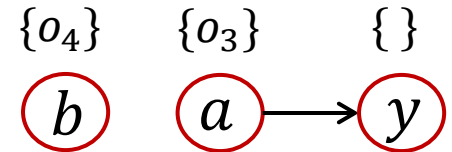
**AddEdge**( $m_{ret}, r$ )

**ProcessCall**( $x, o_i$ )

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



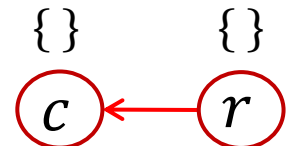
**WL:**  $[\langle B.foo/this, \{o_4\} \rangle, \langle r, \{o_{11}\} \rangle, \langle y, \{o_3\} \rangle]$

**RM:**  $\{ A.main() B.foo(A) \}$

**Processing:**  
 $\langle b, \{o_4\} \rangle$

**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

**PFG:**



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $[\langle r, \{o_{11}\} \rangle, \langle y, \{o_3\} \rangle]$

**RM:**  $\{ A.main() B.foo(A) \}$

**Processing:**

$\langle B.foo/this, \{o_4\} \rangle$

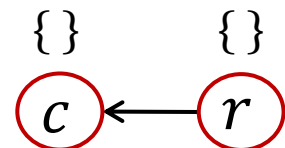
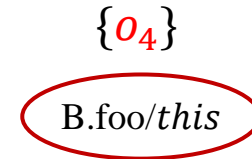
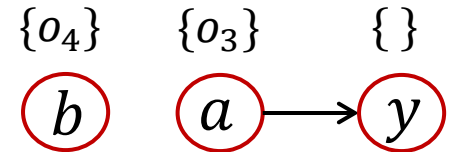
**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $[\langle y, \{o_3\} \rangle, \langle c, \{o_{11}\} \rangle]$

**RM:**  $\{ A.main() B.foo(A) \}$

**Processing:**

$\langle r, \{o_{11}\} \rangle$

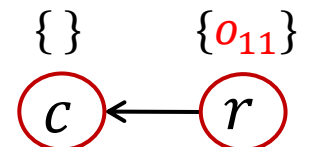
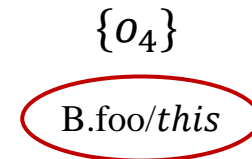
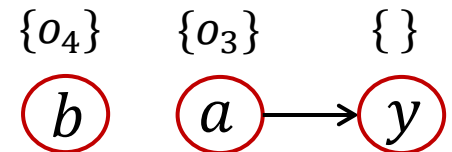
**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:**  $\langle c, \{o_{11}\} \rangle$

**RM:**  $\{ A.main() B.foo(A) \}$

**Processing:**

$\langle y, \{o_3\} \rangle$

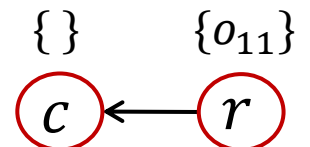
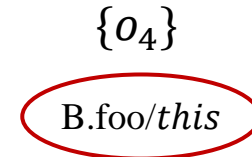
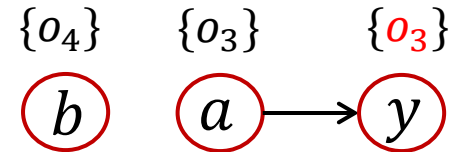
**CG:**  $\{ 5 \rightarrow B.foo(A) \}$

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

**ProcessCall**( $x, o_i$ )

**WL:** []

**RM:** { A.main() B.foo(A) }

**Processing:**

$\langle c, \{o_{11}\} \rangle$

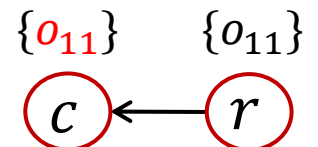
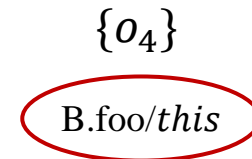
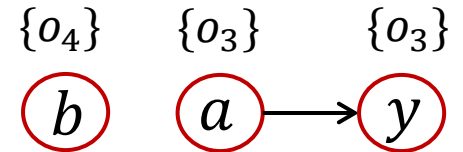
**CG:** { 5  $\rightarrow$  B.foo(A) }

**PFG:**

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```



# An Example

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $m^{entry}$ )

**while**  $WL$  is not empty **do**

remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $x$  **then**

**foreach**  $o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $y, o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $o_i.f, y$ )

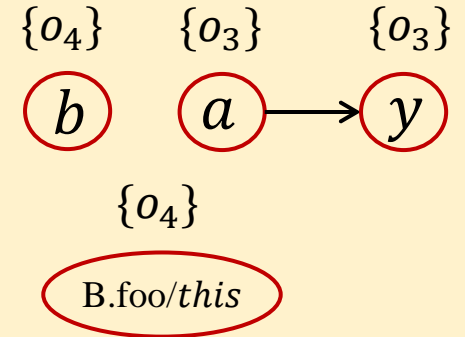
**ProcessCall**( $x, o_i$ )

```

1 class A {
2     static void main() {
3         A a = new A();
4         A b = new B();
5         A c = b.foo(a);
6     }
7     A foo(A x) { ... }
8 }
9 class B extends A {
10    A foo(A y) {
11        A r = new A();
12        return r;
13    }
14 }

```

**Algorithm finishes**



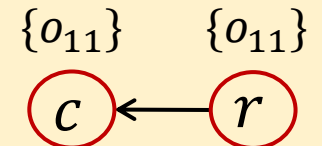
**WL:** []

**RM:** { A.main() B.foo(A) }

**Processing:**

**CG:** { 5  $\rightarrow$  B.foo(A) }

**PFG:**



**Final results**

# The X You Need To Understand in This Lecture

- Understand pointer analysis rule for method call
- Understand inter-procedural pointer analysis algorithm
- Understand on-the-fly call graph construction

注意注意!  
划重点了!





# 软件分析

南京大学

计算机科学与技术系

程序设计语言与

静态分析研究组

李棣  
谭添