



THE UNIVERSITY
OF BRITISH COLUMBIA

MANDOLINE: *Dynamic Slicing of Android Applications* *with Trace-Based Alias Analysis*

ICST, 2021



Khaled Ahmed

khaledea@ece.ubc.ca



Mieszko Lis

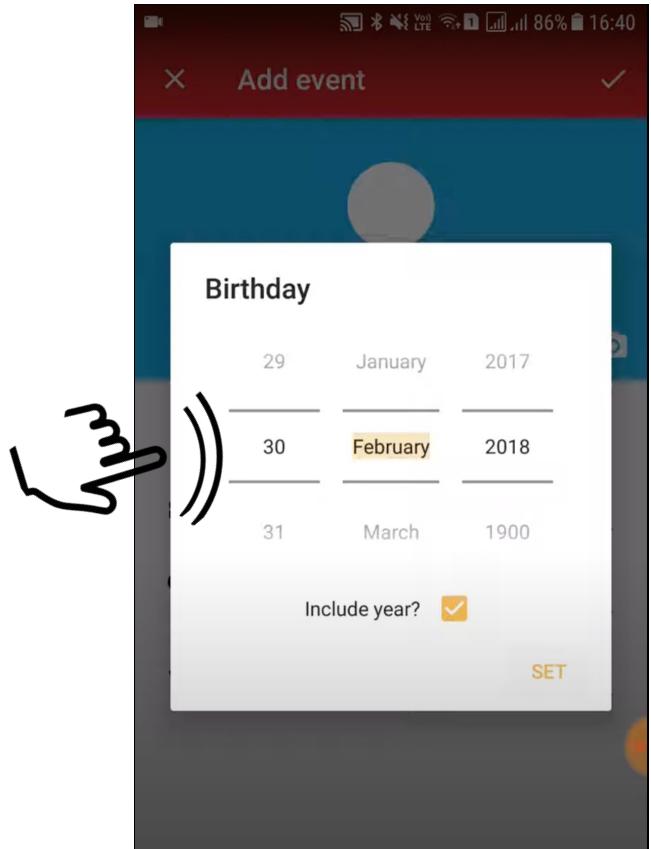
mieszko@ece.ubc.ca



Julia Rubin

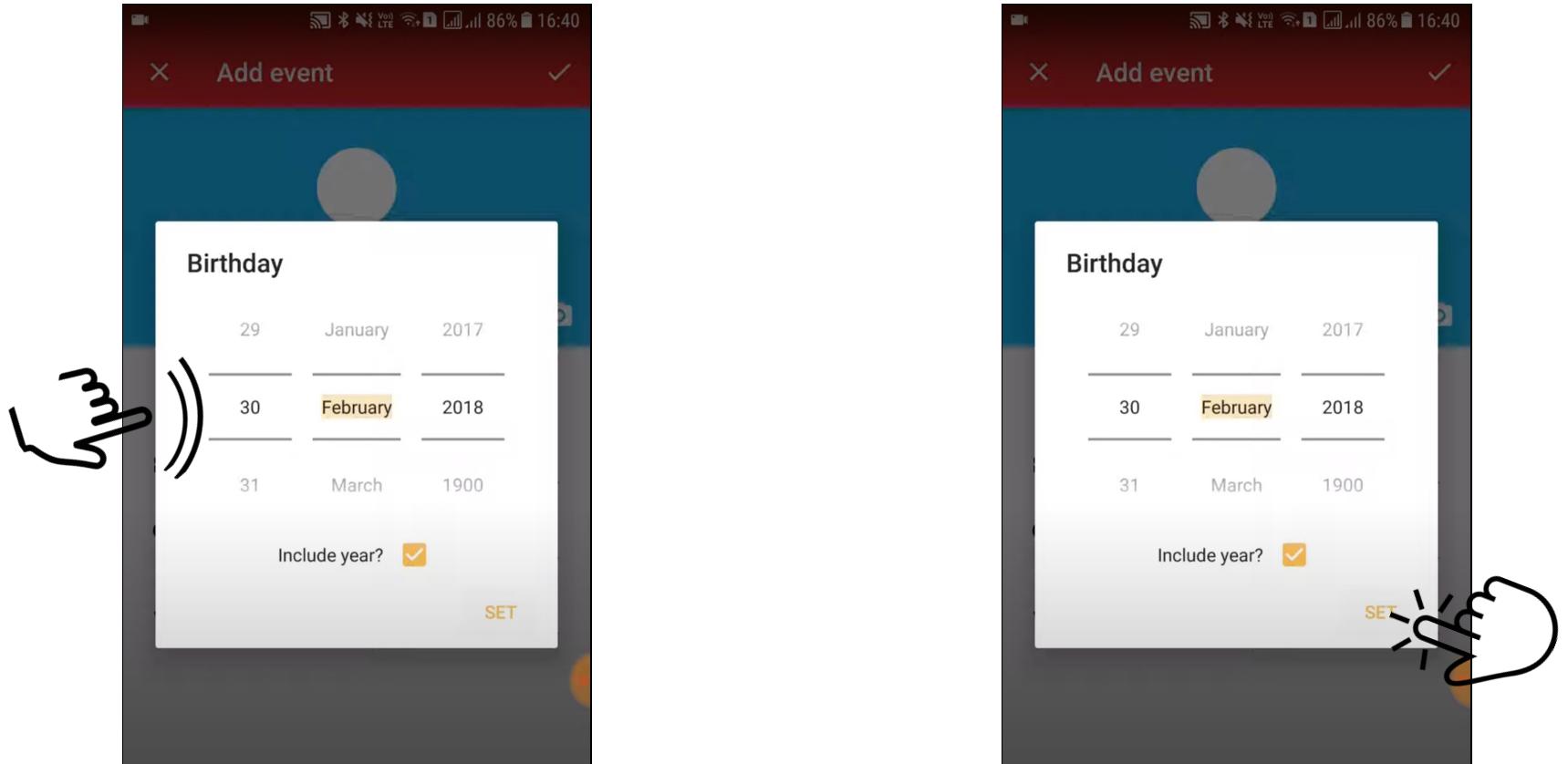
mjulia@ece.ubc.ca

Buggy Android App



User selects monthly recurring event day

Buggy Android App



The image consists of two side-by-side screenshots of a mobile application's date picker interface. Both screenshots show a "Add event" screen with a "Birthday" title. The date picker displays the following data:

Date	Month	Year
29	January	2017
30	February	2018
31	March	1900

An "Include year?" checkbox is checked, and a "SET" button is visible at the bottom right of the picker.

Hand-drawn annotations are present in both screenshots:

- In the left screenshot, a hand icon with three curved lines is pointing towards the date "30" in the February row.
- In the right screenshot, a hand icon with a glowing energy effect is pointing towards the "SET" button.

Below the screenshots are two callout boxes:

- A yellow callout box on the left contains the text: "User selects monthly recurring event day".
- A yellow callout box on the right contains the text: "Crash! There's no February 30th!" in red text.



Debugging

```
Date date = new Date(day,month,year);
```

Debugging

```
int day = userInput.d;  
Date date = new Date(day,month,year);
```



Debugging

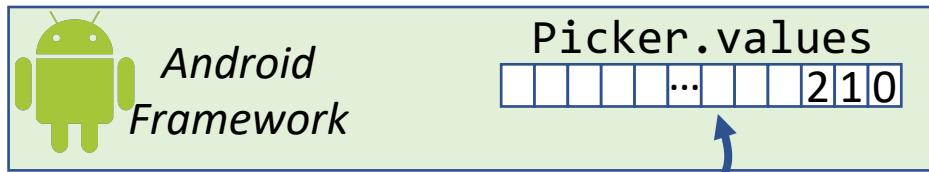
```
bar() {  
  
    foo(selected);  
}  
  
foo(Object userInput) {  
  
    int day = userInput.d;  
  
    Date date = new Date(day,month,year);  
}
```

Debugging

```
bar() {  
  
    selected.d = picker.getValue();  
    ←  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    ←  
    Date date = new Date(day,month,year);  
}
```

Debugging

```
bar() {
```



```
    selected.d = picker.getValue();
```

```
    foo(selected);
```

```
}
```

```
foo(Object userInput) {
```

```
    int day = userInput.d;
```

```
    Date date = new Date(day,month,year);
```

```
}
```

Debugging

```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework Picker.values  
    [     ...     2 | 10 ]  
    ↑  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Debugging

```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework Picker.values  
    [     ...     2 | 10 ]  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Automated by slicing
slice << code size

Debugging with Slicing

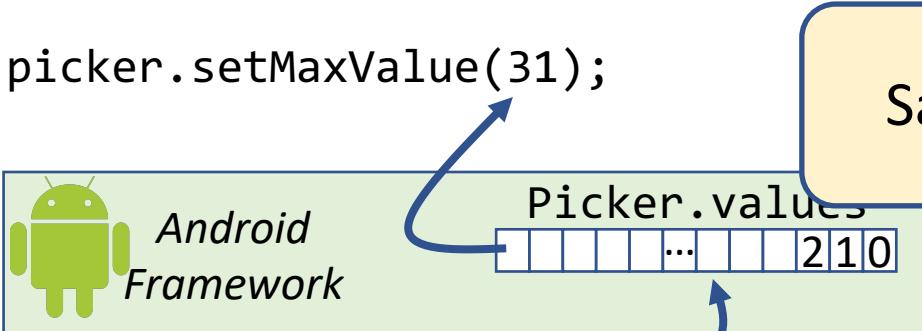
```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework Picker.values  
    [     ...     2 | 10 ]  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

= Data-flow

Challenge 1: Fields

```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework Picker.values  
    [     ...     2 | 10 ]  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Challenge 1: Fields

```
bar() {  
    picker.setMaxValue(31);  
}  


Same field, different names



Android Framework



Picker.values



selected.d = picker.getValue();



foo(selected);



}



foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}


```

Challenge 1: Fields

```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework  
    Picker.values  
    [ 1 2 3 ... 210 ]  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
  
foo(Object userInput) {  
    int day = userInput.d;  
  
    Date date = new Date(day,month,year);  
}
```

Same field, different names

Simple solution:
record memory address
= slow for Android

Challenge 1: Fields

```
bar() {  
    picker.setMaxValue(31);  
  
    Android Framework  
    Picker.values  
    [ ... ] 210  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
  
foo(Object userInput) {  
    int day = userInput.d;  
  
    Date date = new Date(day,month,year);  
}
```

Same field, different names

Simple solution:
record memory address
= slow for Android

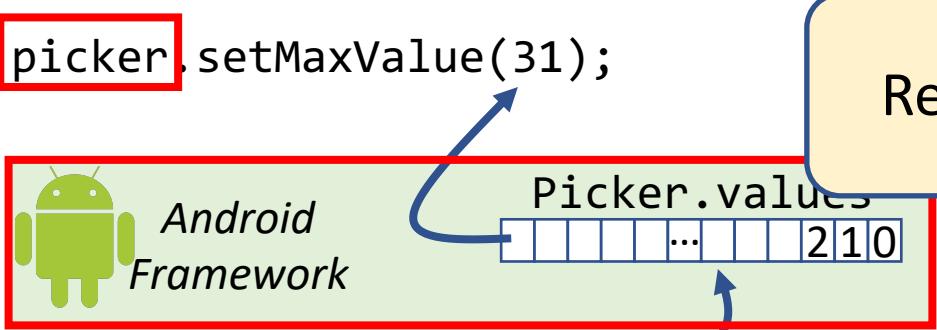
Android kills slow apps

Challenge2: Framework

```
bar() {  
    picker.setMaxValue(31);  
  
     Android Framework Picker.values  
    ... 2 1 0  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Challenge2: Framework

```
bar() {  
    picker.setMaxValue(31);  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```



Rely on framework methods

Challenge2: Framework

```
bar() {  
    picker.setMaxValue(31);  
}  
  
Android Framework  
  
Picker.values  
...  
210  
  
selected.d = picker.getValue();  
foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Rely on framework methods

Simple solution:
Instrument entire framework

Challenge2: Framework

```
bar() {  
    picker.setMaxValue(31);  
  
    Android Framework  
    Picker.values  
    [ ... ] 210  
  
    selected.d = picker.getValue();  
    foo(selected);  
}  
foo(Object userInput) {  
    int day = userInput.d;  
    Date date = new Date(day,month,year);  
}
```

Rely on framework methods

Simple solution:
Instrument entire framework

Not practical

 *Our Insight*



Our Insight



selected.d

userInput.d



Light-weight Instrumentation on-device

+

field data-flow analysis on the trace

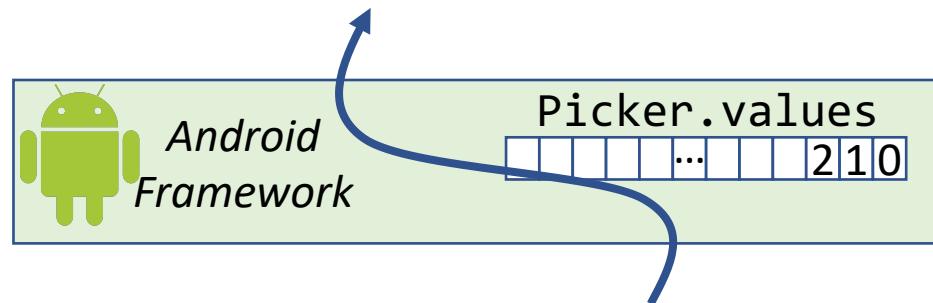


Our Insight



selected.d
↑
userInput.d

Light-weight Instrumentation on-device
+
field data-flow analysis on the trace

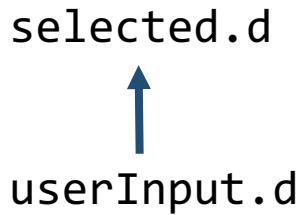


framework modeling

Trace-Based Field Analysis



Goal: Field data-flows w/o object addresses

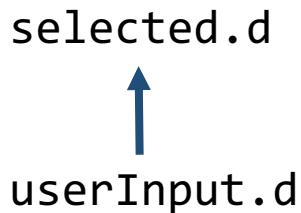


Trace-Based Field Analysis



Goal: Field data-flows w/o object addresses

- How to produce a trace with light-weight instrumentation?



Trace-Based Field Analysis



Goal: Field data-flows w/o object addresses

selected.d
↑
userInput.d

- How to produce a trace with light-weight instrumentation?
- What information to use in the trace instead of object addresses?

Light-Weight Instrumentation

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age
```

Light-Weight Instrumentation

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age
```

Record executed statement IDs

Field Data-Flow Analysis

```
1  jane.age = 15  
2  john        = jane  
3  john.age = 25  
4  res = jane.age
```

Field Data-Flow Analysis

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age      use
```

Field Data-Flow Analysis

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age
```



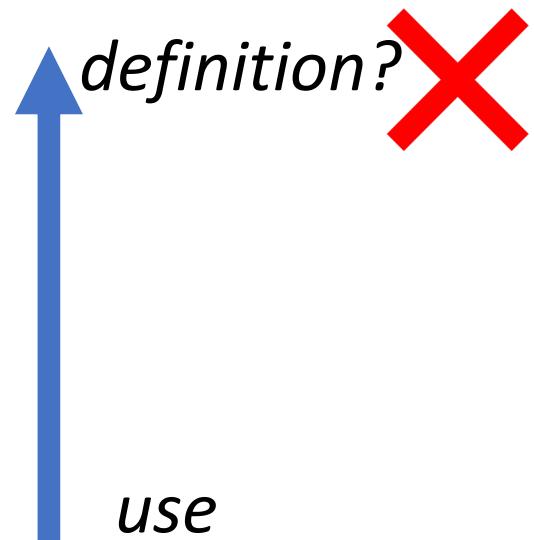
use

Field Data-Flow Analysis

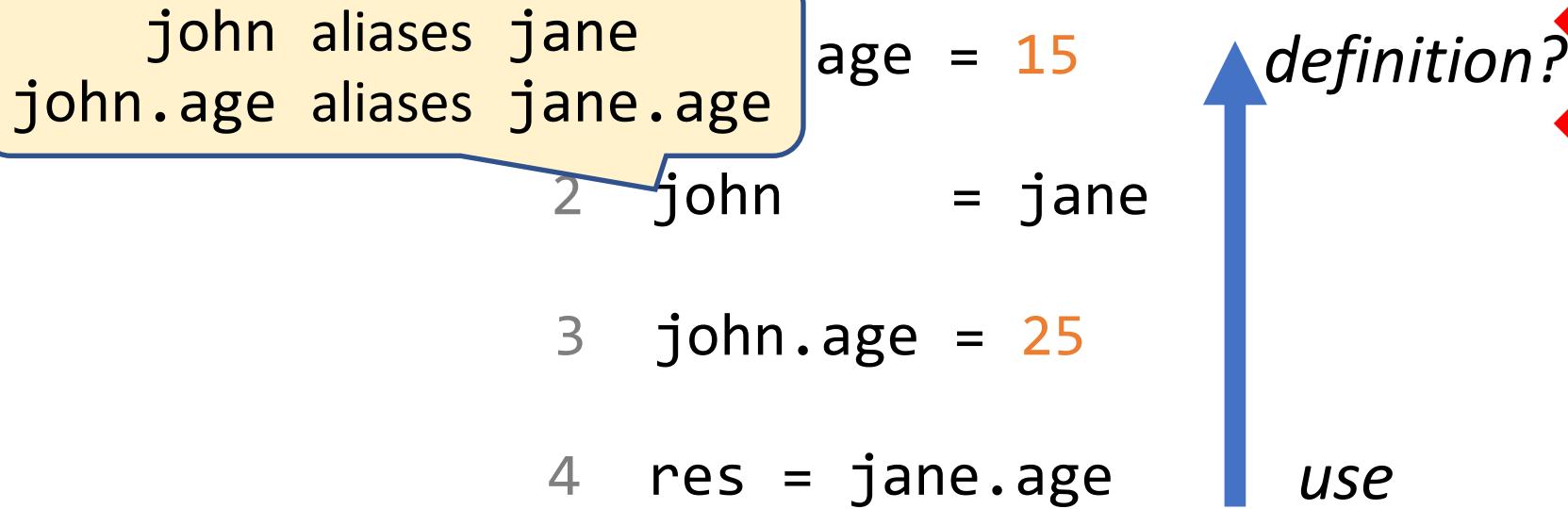
```
1  jane.age = 15      ↑ definition?  
2  john          = jane  
3  john.age = 25  
4  res = jane.age    ↓ use
```

Field Data-Flow Analysis

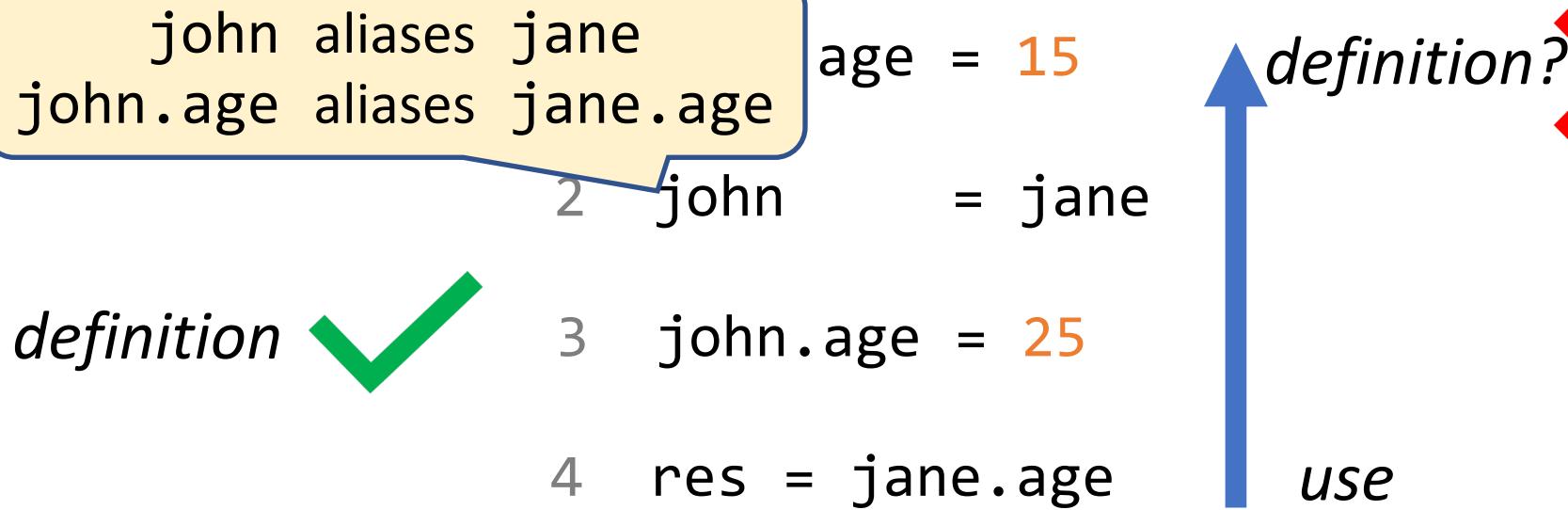
```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res      = jane.age
```



Field Data-Flow Analysis



Field Data-Flow Analysis



Field Alias Analysis

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age
```

Field Alias Analysis

```
1  jane.age = 15  
2  john      = jane  
3  john.age = 25  
4  res = jane.age      use
```

Field Alias Analysis

1 jane.age = 15

2 john = jane

3 john.age = 25

4 res = jane.age



use

Field Alias Analysis

1 jane.age = 15

2 john = jane

3 john.age = 25

4 res = jane.age



use

Field Alias Analysis

Search for `john.age` too

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`



use

Field Alias Analysis

Search for `john.age` too

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`



use

Field Alias Analysis

Search for `john.age` too

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`

use

Field Alias Analysis

Search for `john.age` too

definition

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`

use

Field Alias Analysis

Search for `john.age` too

definition

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`

use

Backward + forward analysis on trace

Field Alias Analysis

```

1 Input: ICDG,  $t$ ,  $v$ 
2 Output:  $T'$ 
3 begin
4    $V \leftarrow \{v\}$                                 ▷ Aliases of the variable  $v$ 
5    $S \leftarrow \text{BackwardAnalysis(ICDG, } V, t)$     ▷ Alias statements
6   return  $\text{LastDefined}(S, t)$                   ▷ Definitions of  $v$  and its fields
7
8 Procedure BackwardAnalysis(ICDG,  $V$ ,  $t$ )
9   begin
10   $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in inverse execution order
11  foreach  $t' s.t. (t', t) \in ICDG$  do
12    if  $\text{method}(t) \neq \text{method}(t')$  then
13       $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
14
15    if  $\exists v \in V s.t. v$  and  $\text{LHS}(t')$  have a common prefix then
16       $t'$  is a definition of a variable in  $V$ 
17       $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
18
19       $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\uparrow, V, t')$ 
20       $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
21       $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
22
23  return  $S$ 
24
25 Procedure ForwardAnalysis(ICDG,  $V$ ,  $t$ )
26   begin
27      $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in execution order
28     foreach  $t' s.t. (t, t') \in ICDG$  do
29       if  $\text{method}(t) \neq \text{method}(t')$  then
30          $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
31
32       if  $\exists v \in V s.t. v$  is a prefix of  $\text{LHS}(t')$  then
33          $t'$  is a definition of a field of a variable in  $V$ 
34          $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
35
36        $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\downarrow, V, t')$ 
37        $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
38        $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
39
40   return  $S$ 

```

```

30 Procedure AliasAnalysis( $d$ ,  $V$ ,  $t'$ )
31   ▷ Initialize with the original set of variables
32   if  $d = \uparrow$  then
33      $V_b \leftarrow V; V_f \leftarrow \emptyset$ 
34
35   else if  $d = \downarrow$  then
36      $V_b \leftarrow \emptyset; V_f \leftarrow V$ 
37
38   foreach  $v \in V$  s.t.  $v$  and  $\text{RHS}(t')$  have a common prefix do
39     ▷  $\text{LHS}(t')$  is a new alias for  $v$ 
40      $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{LHS}(t'))$            ▷ Follow it forward
41
42   foreach  $v \in V$  s.t.  $v$  is a prefix of  $\text{LHS}(t')$  do
43     ▷  $t'$  is a re-definition of a field of  $v$ 
44     ▷ Follow the assigned variable both backward and forward
45      $V_b \leftarrow V_b \cup \text{RHS}(t')$ 
46      $V_f \leftarrow V_f \cup \text{RHS}(t')$ 
47
48   foreach  $v \in V$  s.t.  $\text{LHS}(t')$  is a prefix or equal to  $v$  do
49     ▷  $t'$  is a full re-definition of  $v$ 
50     if  $d = \uparrow$  then
51        $V_b \leftarrow V_b \setminus \{v\}$           ▷ Do not search before the definition
52       ▷ Follow the assigned variable both backward and forward
53        $V_b \leftarrow V_b \cup \text{ExtendFields}(\text{RHS}(t'))$ 
54        $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{RHS}(t'))$ 
55
56     else if  $d = \downarrow$  then
57        $V_f \leftarrow V_f \setminus \{v\}$            ▷ Do not search for new variables
58
59   return  $\langle V_b, V_f \rangle$ 

```

Field Alias Analysis

```

1  Input: ICDG,  $t$ ,  $v$ 
2  Output:  $T'$ 
3  begin
4       $V \leftarrow \{v\}$                                 ▷ Aliases of the variable  $v$ 
5       $S \leftarrow \text{BackwardAnalysis(ICDG, } V, t)$     ▷ Alias statements
6      return  $\text{LastDefined}(S, t)$                   ▷ Definitions of  $v$  and its fields
7
8  Procedure BackwardAnalysis(ICDG,  $V$ ,  $t$ )
9  begin
10      $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in inverse execution order
11     foreach  $t' s.t. (t', t) \in ICDG$  do
12         if  $\text{method}(t) \neq \text{method}(t')$  then
13              $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
14             if  $\exists v \in V s.t. v$  and  $\text{LHS}(t')$  have a common prefix then
15                  $t'$  is a definition of a variable in  $V$ 
16                  $S \leftarrow S \cup \{t'\}$                       ▷ Add the definition statement
17                  $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\uparrow, V, t')$ 
18                  $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
19                  $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
20
21     return  $S$ 
22
23  Procedure ForwardAnalysis(ICDG,  $V$ ,  $t$ )
24  begin
25       $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in execution order
26      foreach  $t' s.t. (t, t') \in ICDG$  do
27          if  $\text{method}(t) \neq \text{method}(t')$  then
28               $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
29              if  $\exists v \in V s.t. v$  is a prefix of  $\text{LHS}(t')$  then
30                   $t'$  is a definition of a field of a variable in  $V$ 
31                   $S \leftarrow S \cup \{t'\}$                       ▷ Add the definition statement
32
33                   $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\downarrow, V, t')$ 
34                   $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
35                   $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
36
37     return  $S$ 

```

```

30  Procedure AliasAnalysis( $d$ ,  $V$ ,  $t'$ )
31      ▷ Initialize with the original set of variables
32      if  $d = \uparrow$  then
33           $V_b \leftarrow V; V_f \leftarrow \emptyset$ 
34      else if  $d = \downarrow$  then
35           $V_b \leftarrow \emptyset; V_f \leftarrow V$ 
36
37      foreach  $v \in V$  s.t.  $v$  and  $\text{RHS}(t')$  have a common prefix do
38          ▷  $\text{LHS}(t')$  is a new alias for  $v$ 
39           $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{LHS}(t'))$            ▷ Follow it forward
40
41      foreach  $v \in V$  s.t.  $v$  is a prefix of  $\text{LHS}(t')$  do
42          ▷  $t'$  is a re-definition of a field of  $v$ 
43          ▷ Follow the assigned variable both backward and forward
44           $V_b \leftarrow V_b \cup \text{RHS}(t')$ 
45           $V_f \leftarrow V_f \cup \text{RHS}(t')$ 
46
47      foreach  $v \in V$  s.t.  $\text{LHS}(t')$  is a prefix or equal to  $v$  do
48          ▷  $t'$  is a full re-definition of  $v$ 
49          if  $d = \uparrow$  then
50               $V_b \leftarrow V_b \setminus \{v\}$            ▷ Do not search before the definition
51              ▷ Follow the assigned variable both backward and forward
52               $V_b \leftarrow V_b \cup \text{ExtendFields}(\text{RHS}(t'))$ 
53               $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{RHS}(t'))$ 
54
55          else if  $d = \downarrow$  then
56               $V_f \leftarrow V_f \setminus \{v\}$            ▷ Do not search for new variables
57
58      return  $\langle V_b, V_f \rangle$ 

```

Search for field definitions backward

Field Alias Analysis

```

1 Input: ICDG,  $t$ ,  $v$ 
2 Output:  $T'$ 
3 begin
4    $V \leftarrow \{v\}$                                 ▷ Aliases of the variable  $v$ 
5    $S \leftarrow \text{BackwardAnalysis(ICDG, } V, t)$     ▷ Alias statements
6   return  $\text{LastDefined}(S, t)$                   ▷ Definitions of  $v$  and its fields
7
8 Procedure BackwardAnalysis(ICDG,  $V$ ,  $t$ )
9   begin
10   $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in inverse execution order
11  foreach  $t' s.t. (t', t) \in ICDG$  do
12    if  $\text{method}(t) \neq \text{method}(t')$  then
13       $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
14
15    if  $\exists v \in V s.t. v$  and  $\text{LHS}(t')$  have a common prefix then
16       $t'$  is a definition of a variable in  $V$ 
17       $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
18
19     $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\uparrow, V, t')$ 
20     $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
21     $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
22
23  return  $S$ 
24
25 Procedure ForwardAnalysis(ICDG,  $V$ ,  $t$ )
26   begin
27      $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in execution order
28     foreach  $t' s.t. (t, t') \in ICDG$  do
29       if  $\text{method}(t) \neq \text{method}(t')$  then
30          $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
31
32       if  $\exists v \in V s.t. v$  is a prefix of  $\text{LHS}(t')$  then
33          $t'$  is a definition of a field of a variable in  $V$ 
34          $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
35
36        $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\downarrow, V, t')$ 
37        $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
38        $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
39
40   return  $S$ 
41
42
43
44
45
46
47

```

```

Procedure AliasAnalysis( $d$ ,  $V$ ,  $t'$ )
  ▷ Initialize with the original set of variables
  if  $d = \uparrow$  then
     $V_b \leftarrow V; V_f \leftarrow \emptyset$ 
  else if  $d = \downarrow$  then
     $V_b \leftarrow \emptyset; V_f \leftarrow V$ 
  foreach  $v \in V$  s.t.  $v$  and  $\text{RHS}(t')$  have a common prefix do
    ▷  $\text{LHS}(t')$  is a new alias for  $v$ 
     $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{LHS}(t'))$  ▷ Follow it forward
  foreach  $v \in V$  s.t.  $v$  is a prefix of  $\text{LHS}(t')$  do
    ▷  $t'$  is a re-definition of a field of  $v$ 
    ▷ Follow the assigned variable both backward and forward
     $V_b \leftarrow V_b \cup \text{RHS}(t')$ 
     $V_f \leftarrow V_f \cup \text{RHS}(t')$ 
  foreach  $v \in V$  s.t.  $\text{LHS}(t')$  is a prefix or equal to  $v$  do
    ▷  $t'$  is a full re-definition of  $v$ 
    if  $d = \uparrow$  then
       $V_b \leftarrow V_b \setminus \{v\}$  ▷ Do not search before the definition
      ▷ Follow the assigned variable both backward and forward
       $V_b \leftarrow V_b \cup \text{ExtendFields}(\text{RHS}(t'))$ 
       $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{RHS}(t'))$ 
    else if  $d = \downarrow$  then
       $V_f \leftarrow V_f \setminus \{v\}$  ▷ Do not search for new variables
  return  $\langle V_b, V_f \rangle$ 

```

Search for field definitions forward

Field Alias Analysis

```

1 Input: ICDG,  $t$ ,  $v$ 
2 Output:  $T'$ 
3 begin
4    $V \leftarrow \{v\}$                                 ▷ Aliases of the variable  $v$ 
5    $S \leftarrow \text{BackwardAnalysis(ICDG, } V, t)$     ▷ Alias statements
6   return  $\text{LastDefined}(S, t)$                   ▷ Definitions of  $v$  and its fields
7
8 Procedure BackwardAnalysis(ICDG,  $V$ ,  $t$ )
9   begin
10     $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in inverse execution order
11    foreach  $t' s.t. (t', t) \in ICDG$  do
12      if  $\text{method}(t) \neq \text{method}(t')$  then
13         $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
14
15      if  $\exists v \in V s.t. v$  and  $\text{LHS}(t')$  have a common prefix then
16         $t'$  is a definition of a variable in  $V$ 
17         $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
18
19         $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\uparrow, V, t')$ 
20         $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
21         $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
22
23    return  $S$ 
24
25 Procedure ForwardAnalysis(ICDG,  $V$ ,  $t$ )
26   begin
27     $S \leftarrow \emptyset$                             ▷ Traverse the ICDG in execution order
28    foreach  $t' s.t. (t, t') \in ICDG$  do
29      if  $\text{method}(t) \neq \text{method}(t')$  then
30         $V \leftarrow \text{ChangeScope}(V, \text{method}(t), \text{method}(t'))$ 
31
32      if  $\exists v \in V s.t. v$  is a prefix of  $\text{LHS}(t')$  then
33         $t'$  is a definition of a field of a variable in  $V$ 
34         $S \leftarrow S \cup \{t'\}$                          ▷ Add the definition statement
35
36         $\langle V_b, V_f \rangle \leftarrow \text{AliasAnalysis}(\downarrow, V, t')$ 
37         $S \leftarrow S \cup \text{BackwardAnalysis}(ICDG, V_b, t')$ 
38         $S \leftarrow S \cup \text{ForwardAnalysis}(ICDG, V_f, t')$ 
39
40    return  $S$ 
41
42
43
44
45
46
47

```

```

Procedure AliasAnalysis( $d$ ,  $V$ ,  $t'$ )
  ▷ Initialize with the original set of variables
  if  $d = \uparrow$  then
     $V_b \leftarrow V$ ;  $V_f \leftarrow \emptyset$ 
  else if  $d = \downarrow$  then
     $V_b \leftarrow \emptyset$ ;  $V_f \leftarrow V$ 
  foreach  $v \in V$  s.t.  $v$  and  $\text{RHS}(t')$  have a common prefix do
    ▷  $\text{LHS}(t')$  is a new alias for  $v$ 
     $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{LHS}(t'))$                                 ▷ Follow it forward
  foreach  $v \in V$  s.t.  $v$  is a prefix of  $\text{LHS}(t')$  do
    ▷  $t'$  is a re-definition of a field of  $v$ 
    ▷ Follow the assigned variable both backward and forward
     $V_b \leftarrow V_b \cup \text{RHS}(t')$ 
     $V_f \leftarrow V_f \cup \text{RHS}(t')$ 
  foreach  $v \in V$  s.t.  $\text{LHS}(t')$  is a prefix or equal to  $v$  do
    ▷  $t'$  is a full re-definition of  $v$ 
    if  $d = \uparrow$  then
       $V_b \leftarrow V_b \setminus \{v\}$           ▷ Do not search before the definition
      ▷ Follow the assigned variable both backward and forward
       $V_b \leftarrow V_b \cup \text{ExtendFields}(\text{RHS}(t'))$ 
       $V_f \leftarrow V_f \cup \text{ExtendFields}(\text{RHS}(t'))$ 
    else if  $d = \downarrow$  then
       $V_f \leftarrow V_f \setminus \{v\}$           ▷ Do not search for new variables
  return  $\langle V_b, V_f \rangle$ 

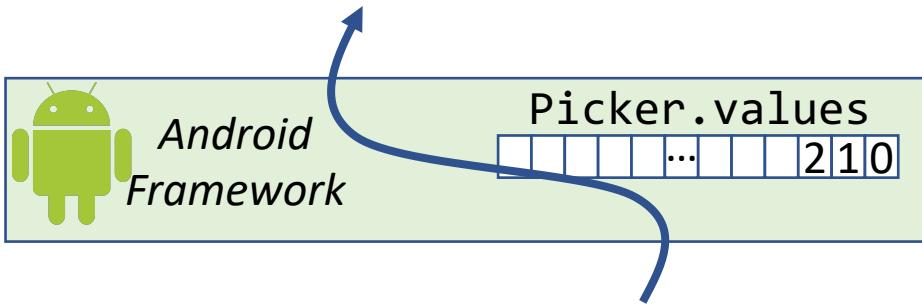
```

Decide on variables to search for and the direction

Framework Data-Flows



Goal: Data-flows inside the Framework



How to find data-flows inside framework w/o instrumentation?

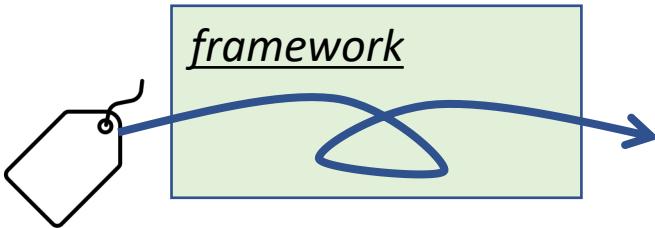


Taint-Propagation Models

StubDroid [1] & FlowDroid [2]

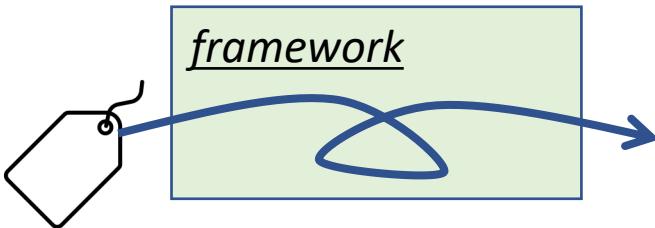
Taint-Propagation Models

StubDroid [1] & FlowDroid [2]



Taint-Propagation Models

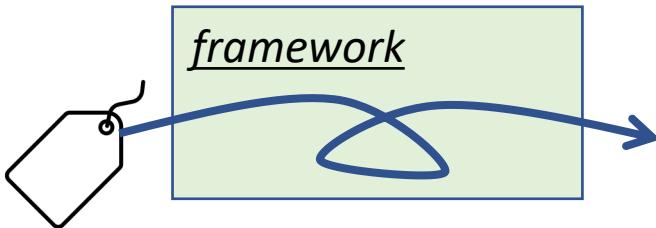
StubDroid [1] & FlowDroid [2]



Express how taint propagates

Taint-Propagation Models

StubDroid [1] & FlowDroid [2]



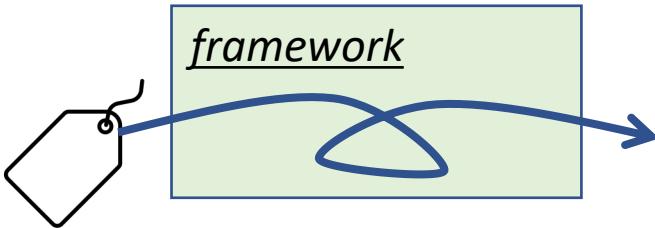
Express how taint propagates

From:

- Parameters
 - Receiver
- } + Fields

Taint-Propagation Models

StubDroid [1] & FlowDroid [2]



Express how taint propagates

From:

- Parameters
 - Receiver
- } + Fields

To:

- Parameters
 - Receiver
 - Return
- } + Fields



Taint-Propagation Models

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

Taint-Propagation Models

Taint-propagation:

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

value → this.inA

Taint-Propagation Models

Taint-propagation:

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

value	→	this.inA
index	✗	
this	✗	

Taint-Propagation Models

Taint-propagation:

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

value	→	this.inA
index	✗	
this	✗	

```
Log::info(value) {  
    print(value);  
}
```

Taint-Propagation Models

Taint-propagation:

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

value → this.inA
index ✗
this ✗

```
Log::info(value) {  
    print(value);  
}
```

value ✗

Taint-Propagation Models

Taint-propagation:

```
Array::set(index, value) {  
    this.inA[index] = value;  
}
```

value → this.inA
index ✗
this ✗

```
Log::info(value) {  
    print(value);  
}
```

value ✗

Accurate taint propagation rules
for Java and Android methods

Modeling w/ Implicit Assignments



value → this.inA

Modeling w/ Implicit Assignments



value → this.inA

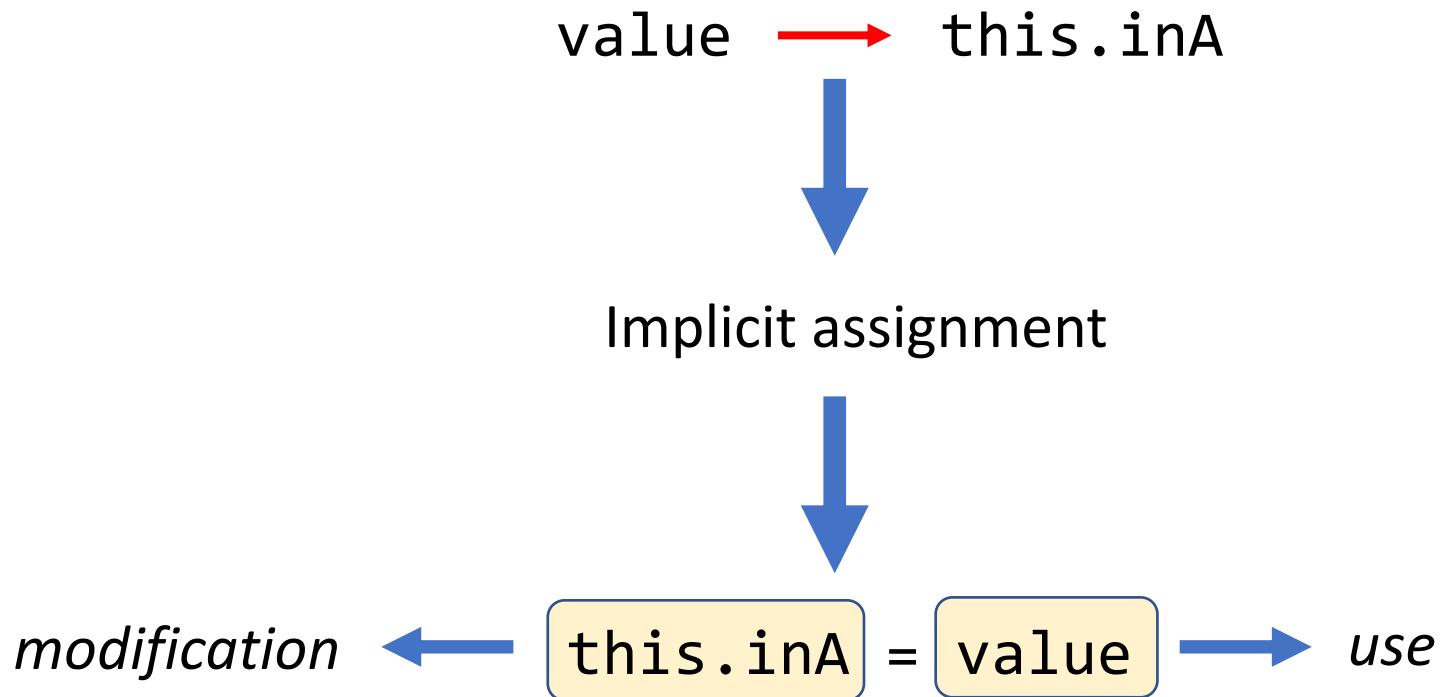


Implicit assignment



this.inA = value

Modeling w/ Implicit Assignments



Framework Modeling: Example



Code

```
x = 5
```

```
array.set(1, x)
```

```
y = array.get(1)
```

Model

```
y = array.inA
```

Framework Modeling: Example



Code

```
x = 5
```

```
array.set(1, x)
```

```
y = array.get(1)
```

Model

```
y = array.inA
```



Framework Modeling: Example



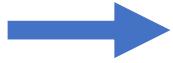
Code

x = 5

array.set(1, x)

y = array.get(1)

Model



array.inA = x

y = array.inA

Framework Modeling: Example



Code

```
x = 5
```

```
array.set(1, x)
```

```
y = array.get(1)
```



Model

```
array.inA = x
```

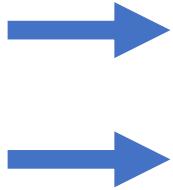
```
y = array.inA
```

Framework Modeling: Example



Code

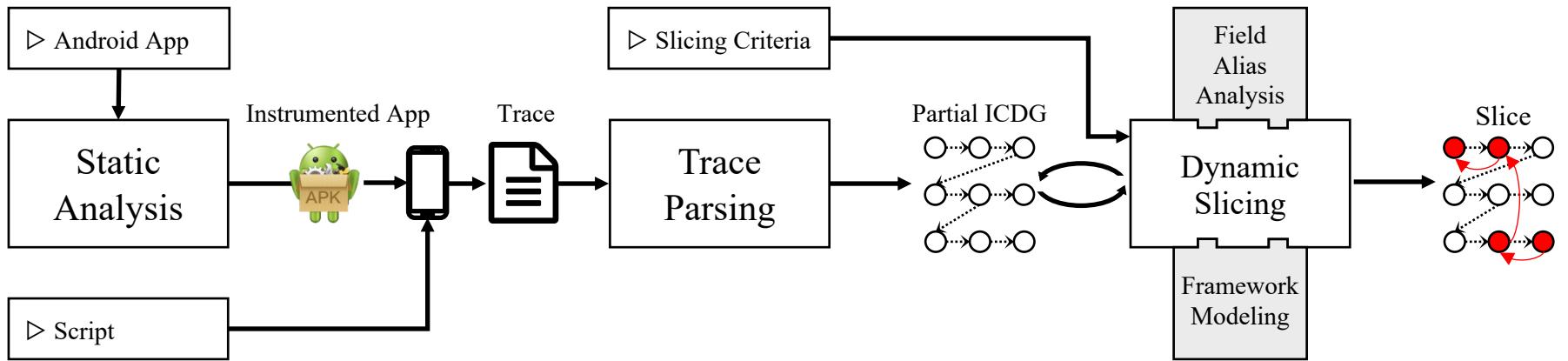
```
x = 5  
array.set(1, x)  
y = array.get(1)
```



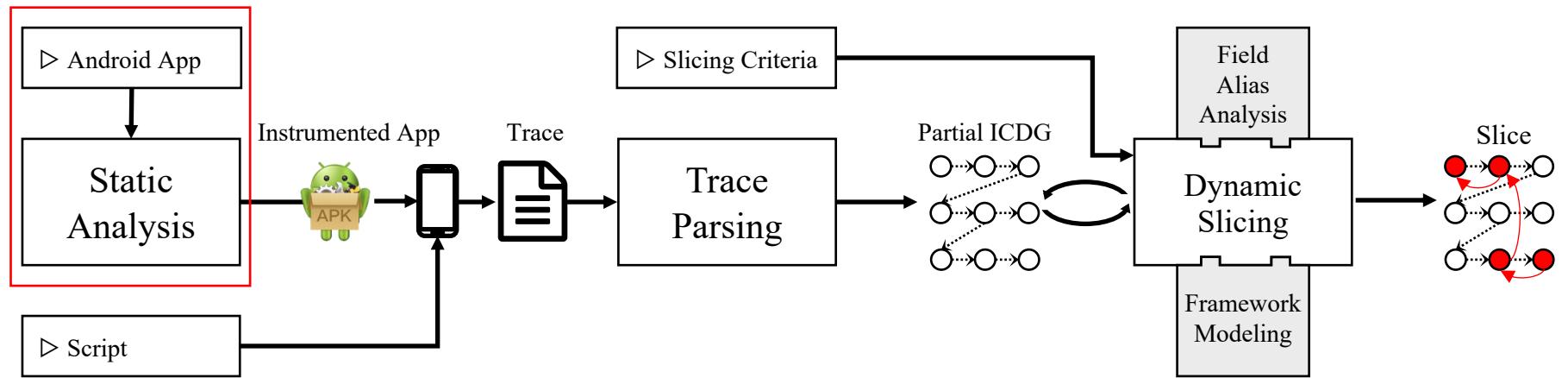
Model

```
array.inA = x  
y = array.inA
```

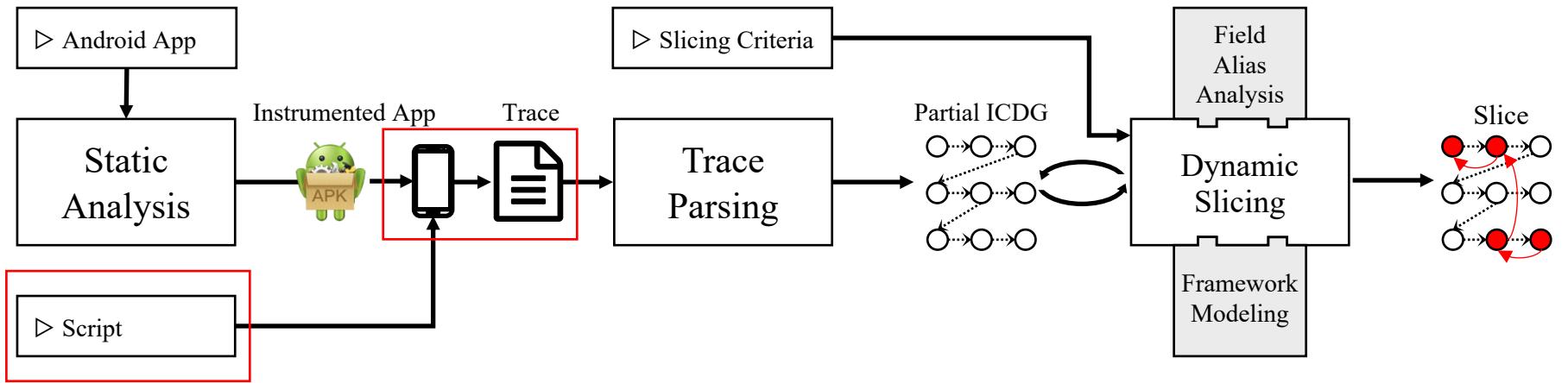
Our Solution: MANDOLINE



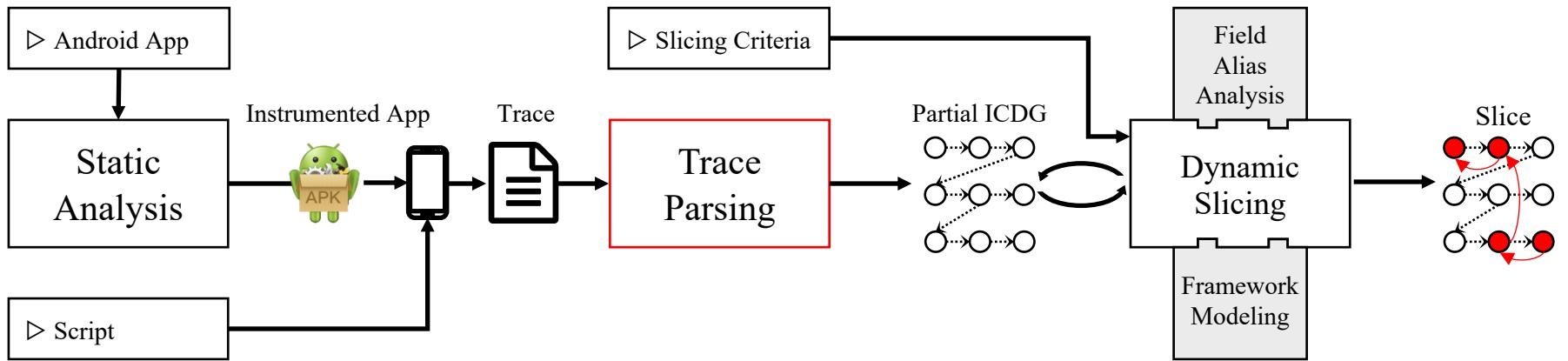
Our Solution: MANDOLINE



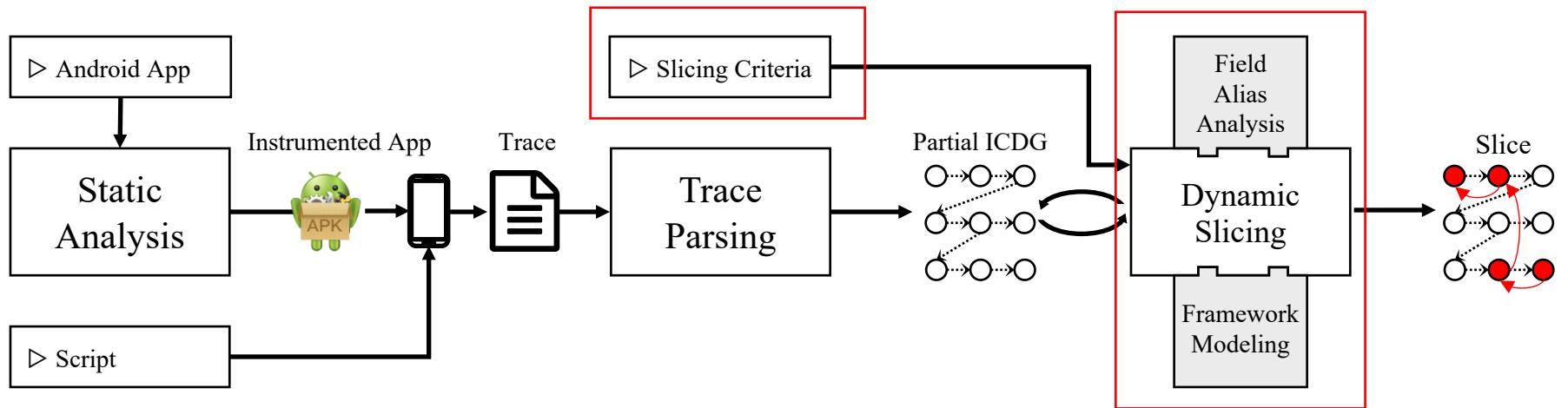
Our Solution: MANDOLINE



Our Solution: MANDOLINE



Our Solution: MANDOLINE



Evaluation



RQ1: Accuracy

vs. state-of-the-art

RQ2: Performance



Subjects

12 Apps from DroixBench [3] and ReCDroid [4]

- Faulty apps w/ crashes
- Reproducible crashes
- Crashes = slicing criteria

Baseline

AndroidSlicer [5] (state-of-the-art)

- First slicer for Android apps
- Inter-callback data-flows
- Misses data-flows in fields and framework

AndroidSlicer++:

- Fixed implementation issues
- Improved instrumentation
- Allows fair comparison

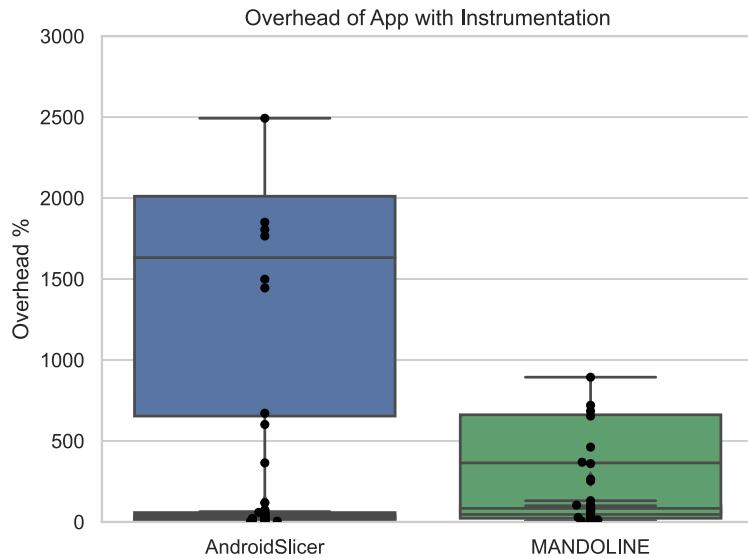


Benchmark

Manually produced slices

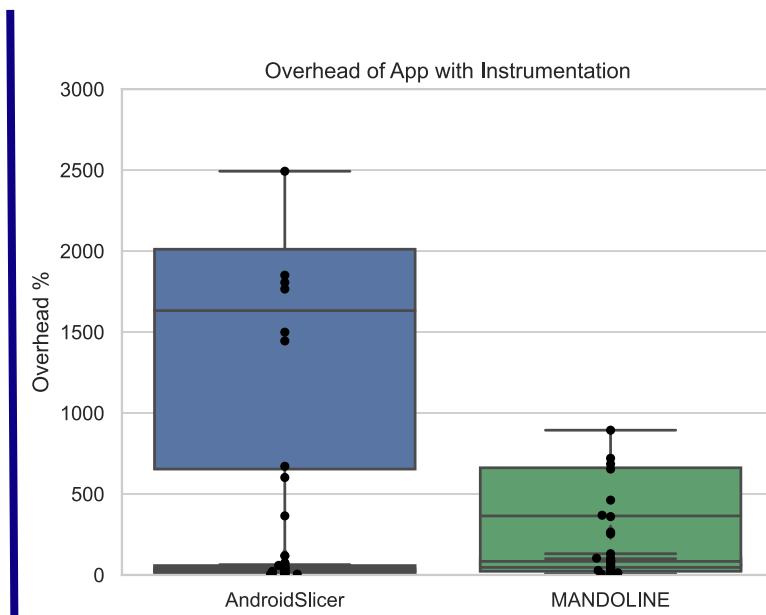
- Cross validated by two researchers
- First Android slices benchmark
- State-of-the-art [5] only compared slice to trace sizes

Accuracy and Performance



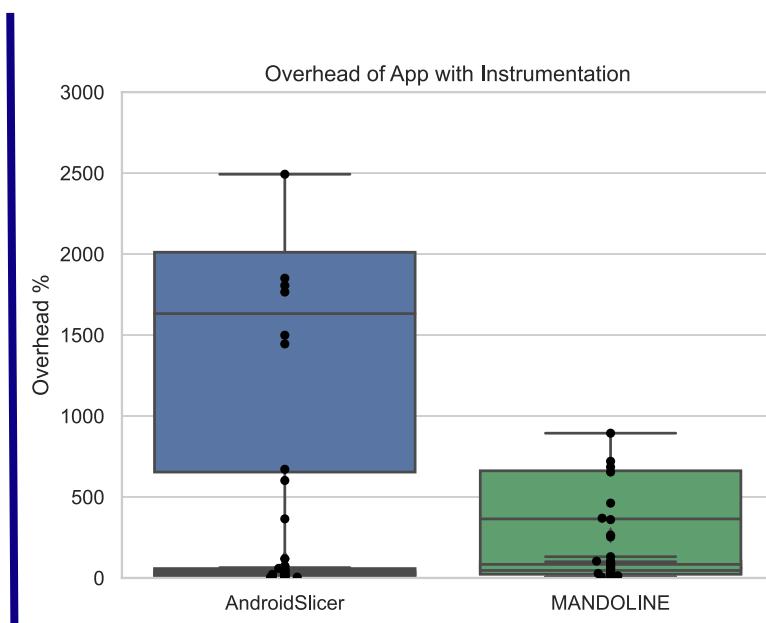
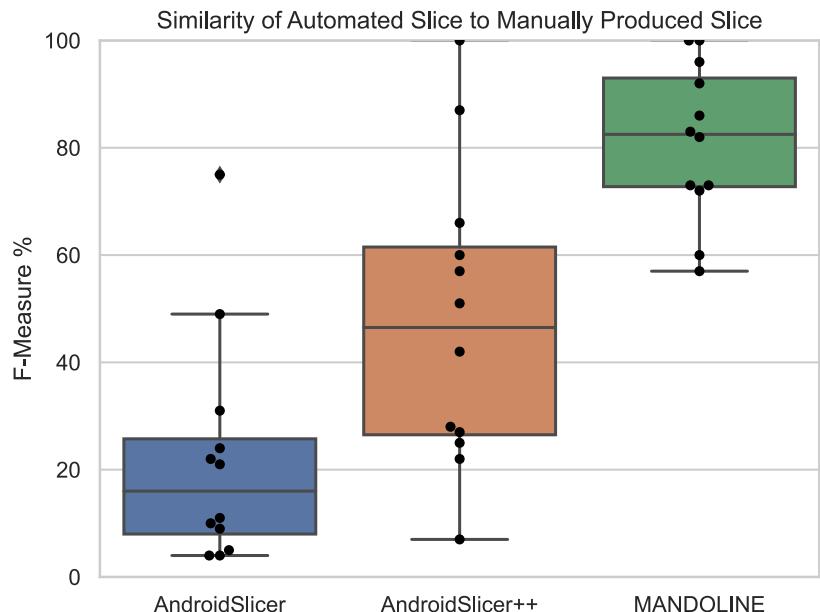
Accuracy and Performance

RQ1



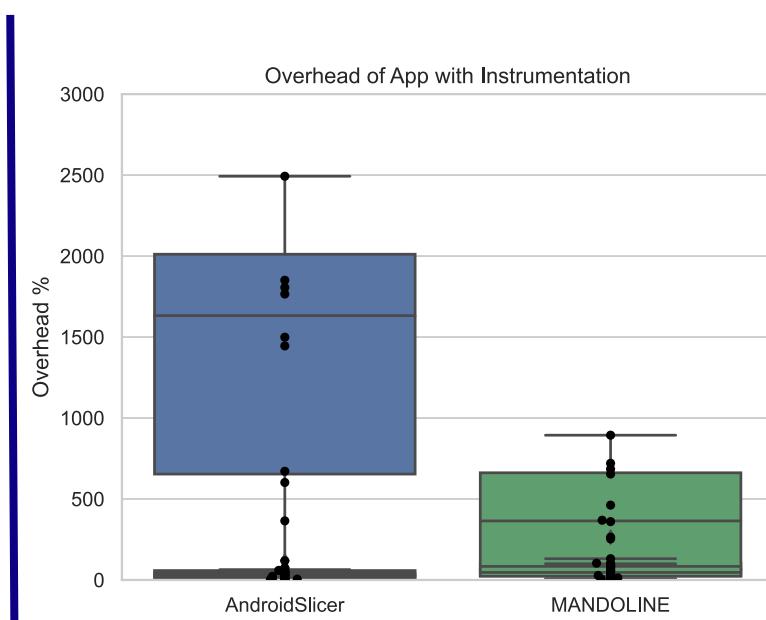
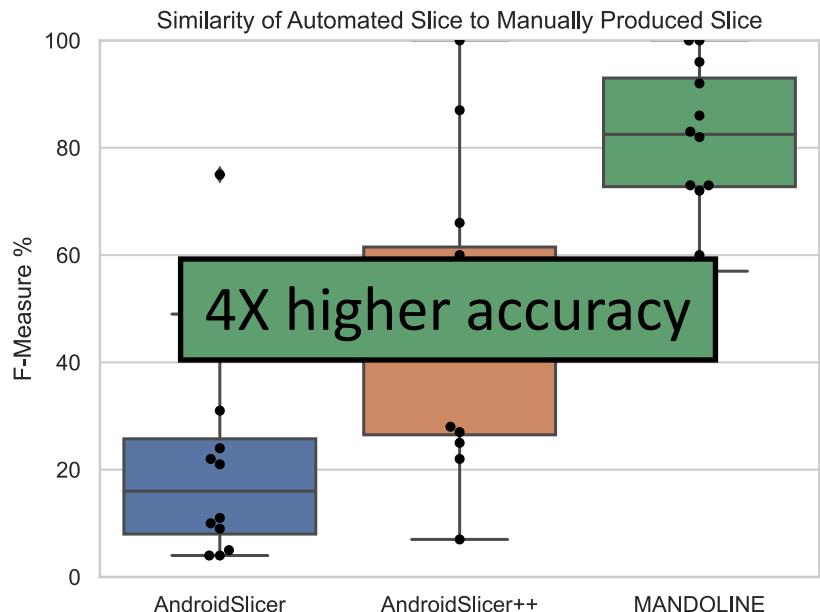
Accuracy and Performance

RQ1



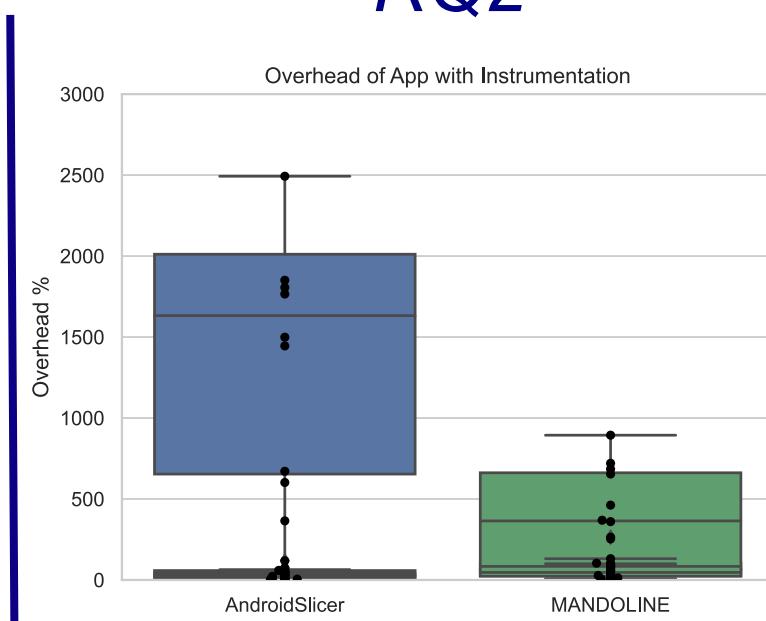
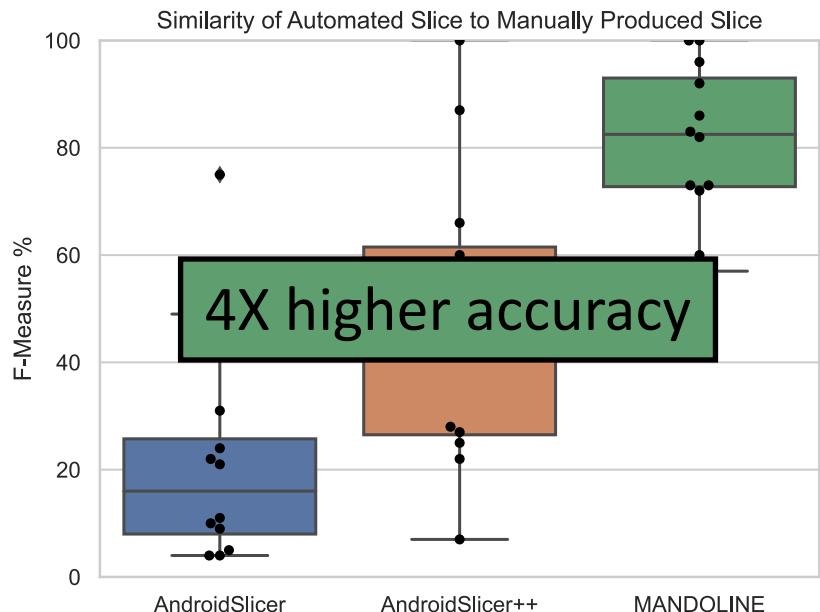
Accuracy and Performance

RQ1



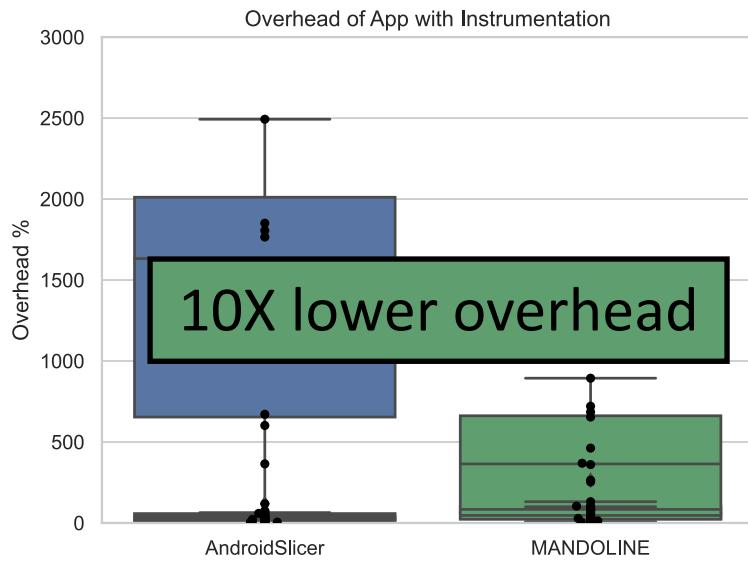
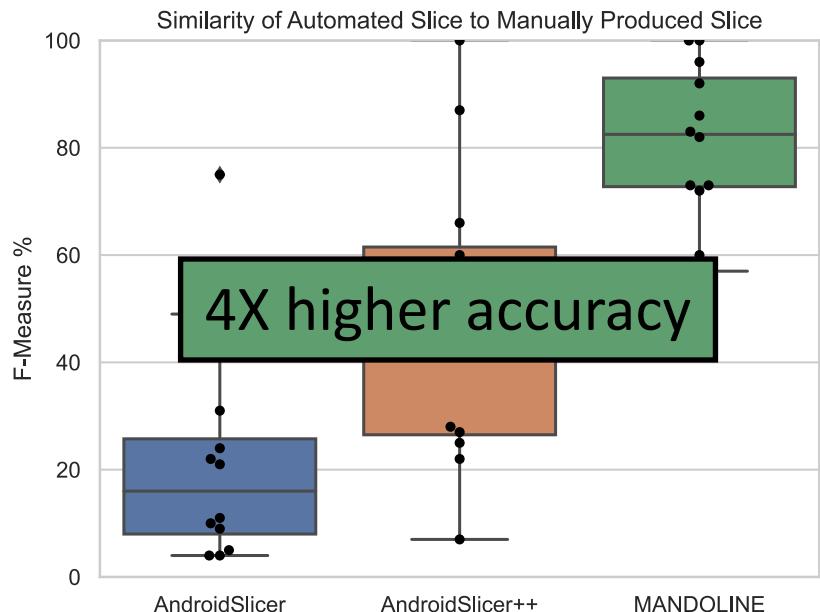
Accuracy and Performance

RQ1 RQ2



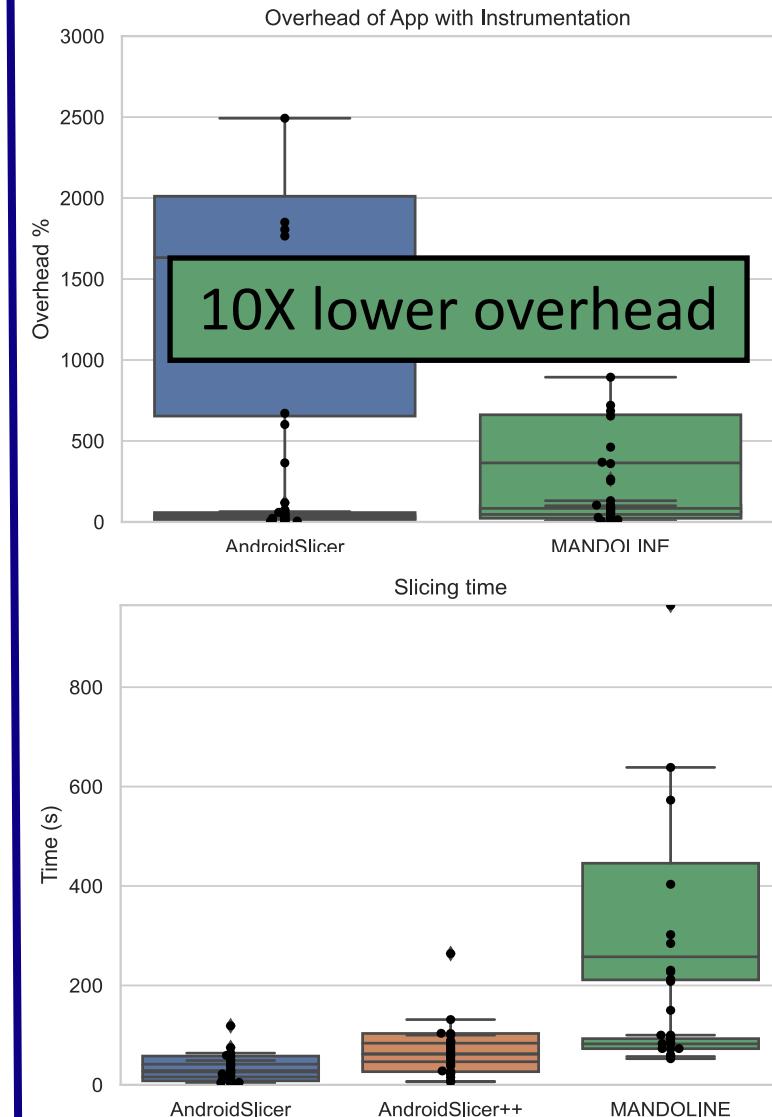
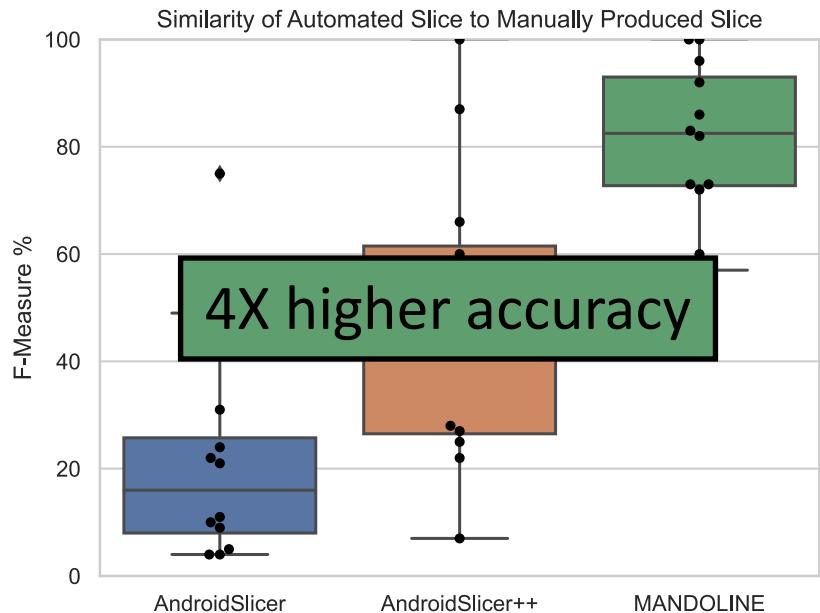
Accuracy and Performance

RQ1 RQ2



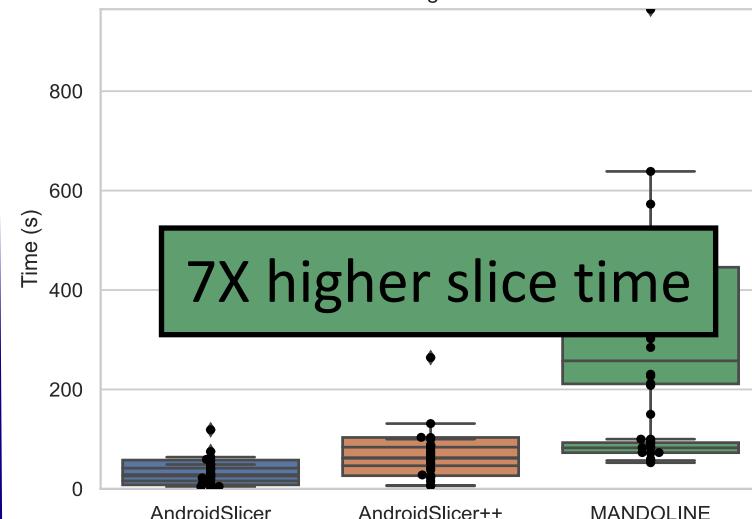
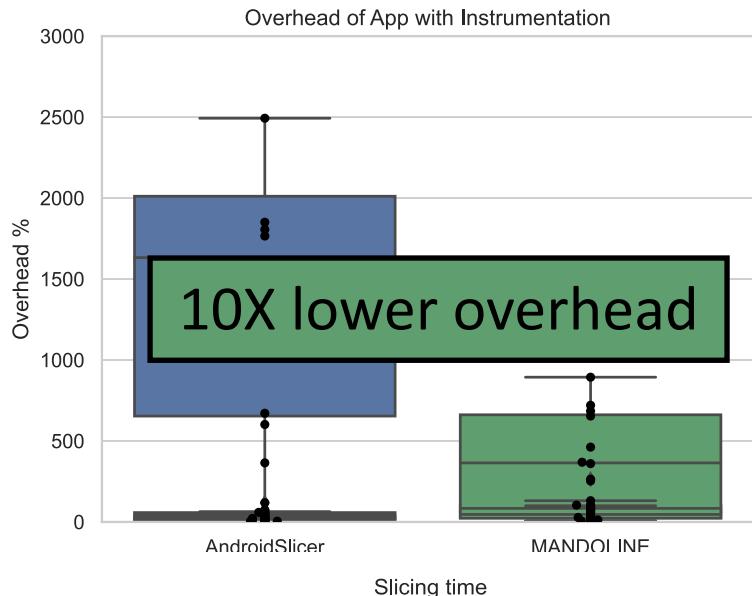
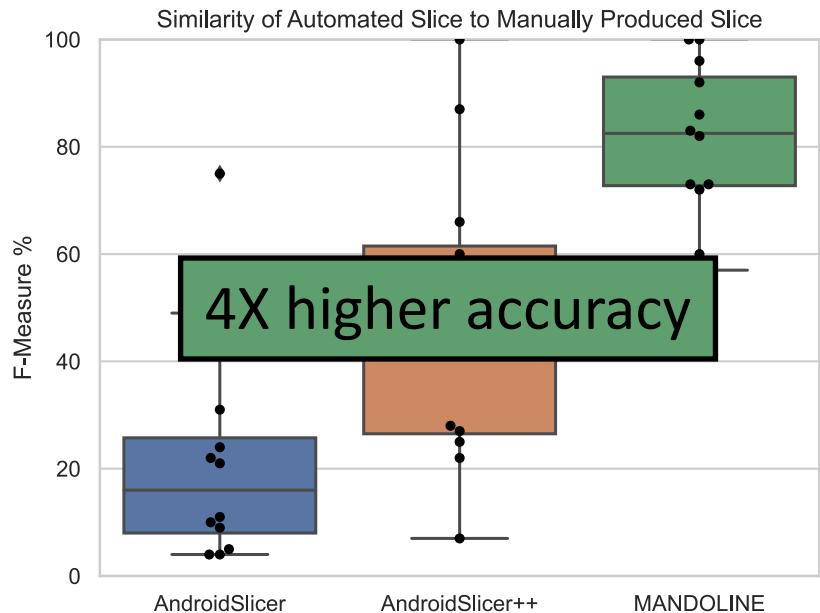
Accuracy and Performance

RQ1 RQ2



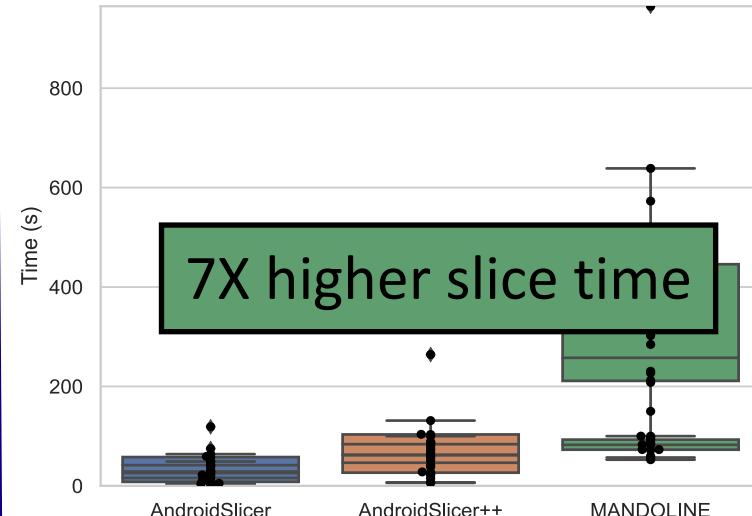
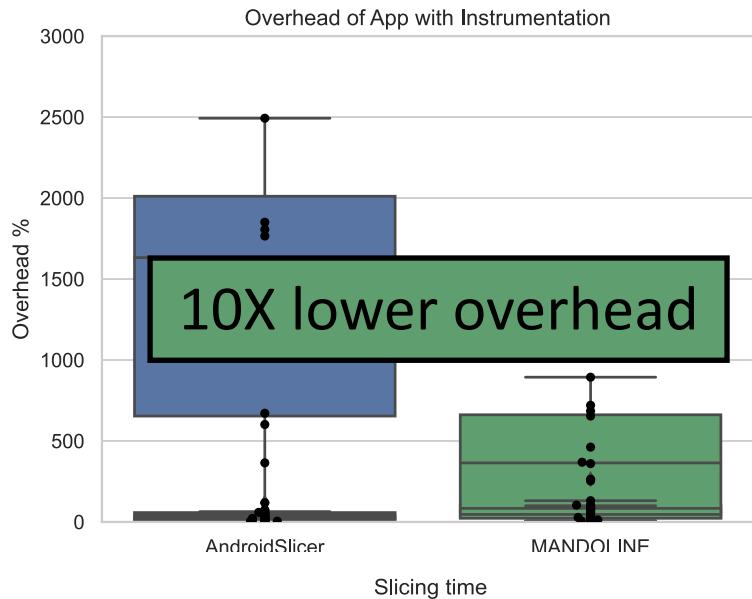
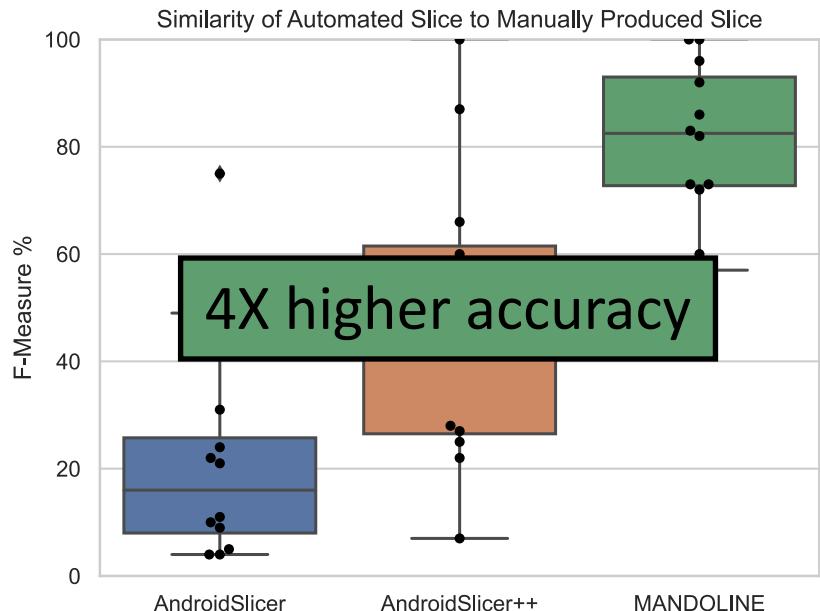
Accuracy and Performance

RQ1 RQ2



Accuracy and Performance

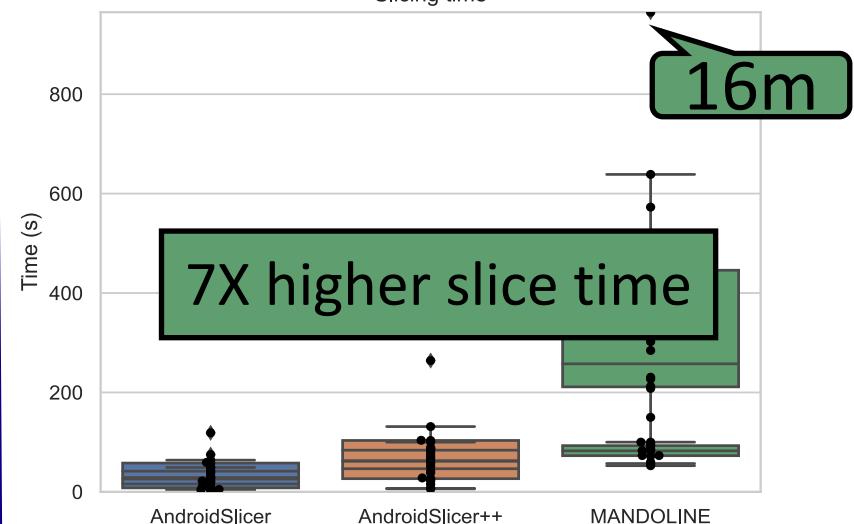
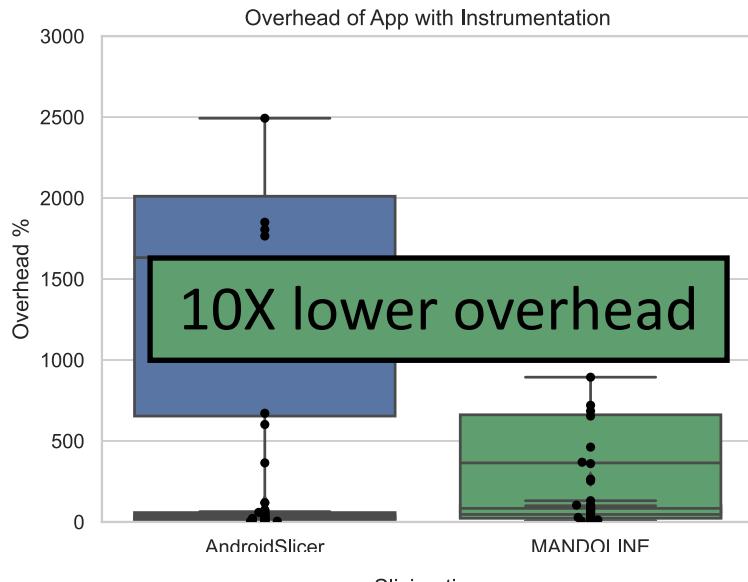
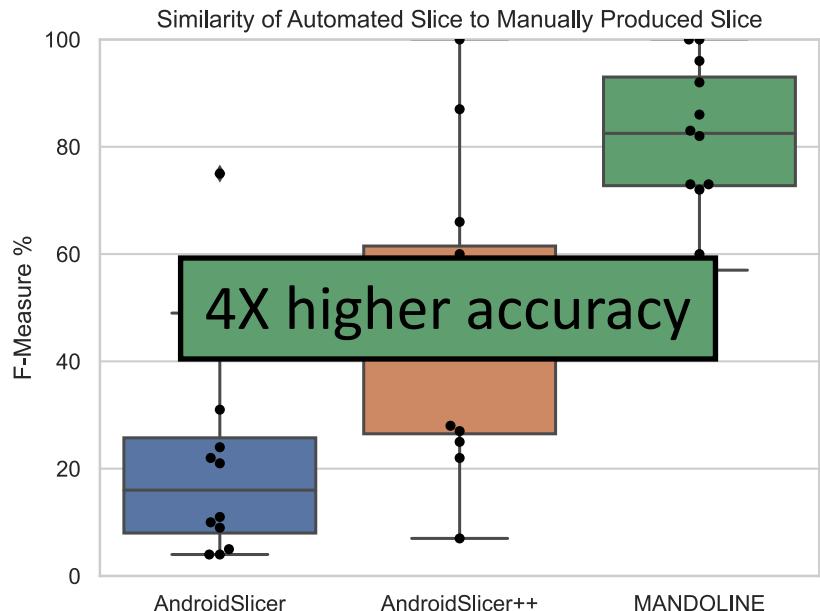
RQ1 RQ2



Acceptable slice time given huge accuracy improvement

Accuracy and Performance

RQ1 RQ2



Acceptable slice time given huge accuracy improvement

Summary

Insights

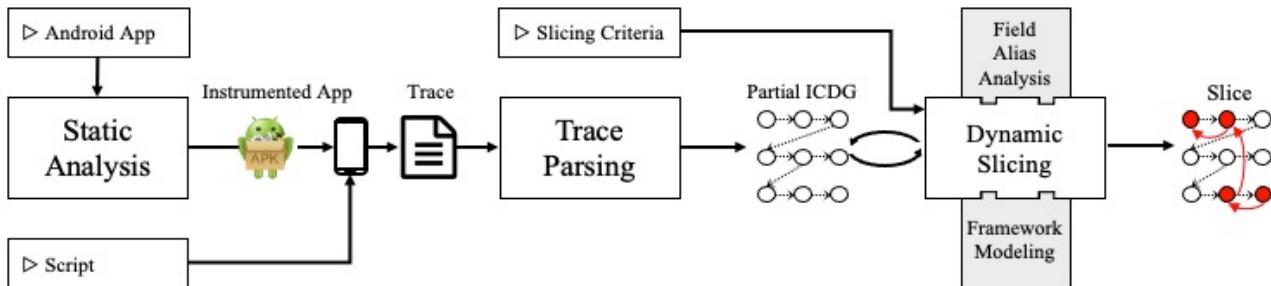


Extract field data flows from the trace using alias analysis

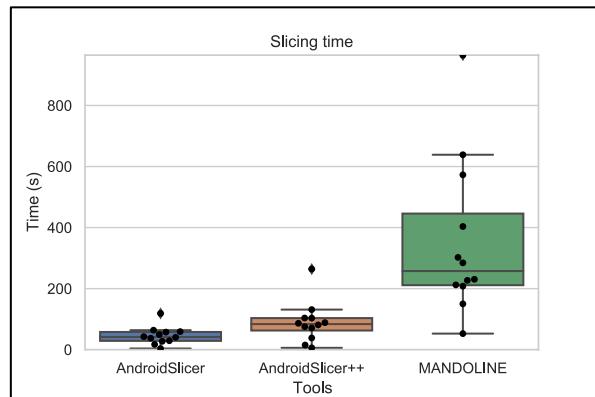
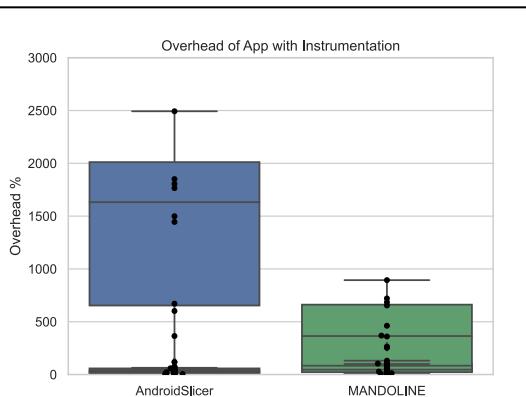
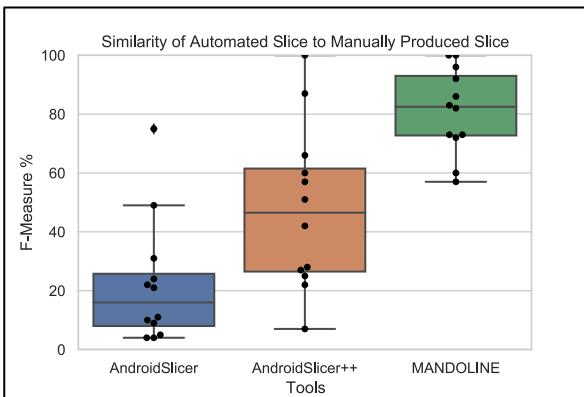


Use framework models from static analysis

Approach



Results



Code + Benchmark suite: <https://resess.github.io/PaperAppendices/Mandoline/>