Eventually Sound Points-To Analysis with Specifications

Osbert Bastani, Rahul Sharma, Lazaro Clapp, Saswat Anand, and Alex Aiken

Android Malware Detection

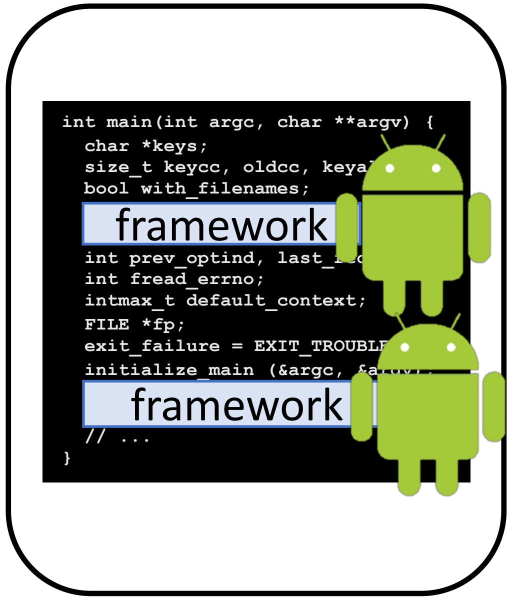
- Characterize Android malware using source to sink taint flows

Information leak:	location	flows to	Internet
SMS Fraud:	phone #	used in	SMS send
Ransomware:	network packets	encrypt	files

- Use a static taint analysis to find these flows
  - Example of a client for our eventually sound static points-to analysis

Taint Analysis for Android Apps

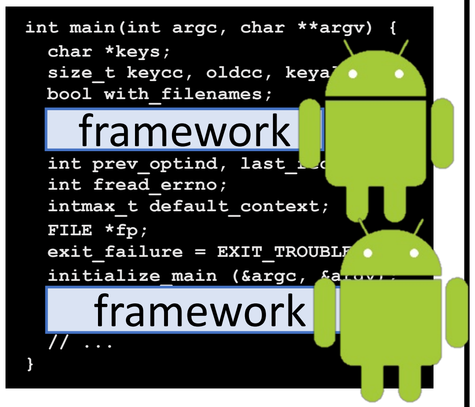
Taint Analysis for Android Apps



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keya
  bool with_filenames;
  framework
  int prev_optind, last_
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE
  initialize_main (&argc, &a
  framework
  // ...
}
```

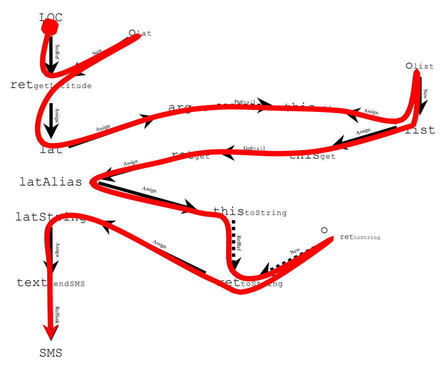
Input: Android app and framework

Taint Analysis for Android Apps



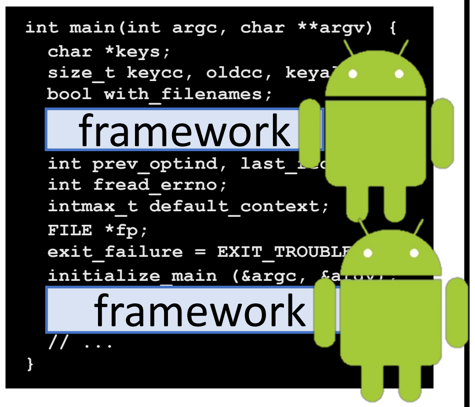
```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keya
  bool with_filenames;
    framework
  int prev_optind, last_
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE
  initialize_main (&argc, &a
    framework
  // ...
}
```

Input: Android app and framework

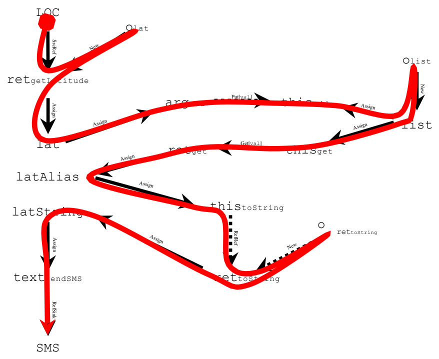


Step 1: Static taint analysis

Taint Analysis for Android Apps



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keya
  bool with_filenames;
  framework
  int prev_optind, last_
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE
  initialize_main (&argc,
  framework
  // ...
}
```



location → Internet
SMS → Internet

Input: Android app and framework

Step 1: Static taint analysis

Output: Malicious behavior

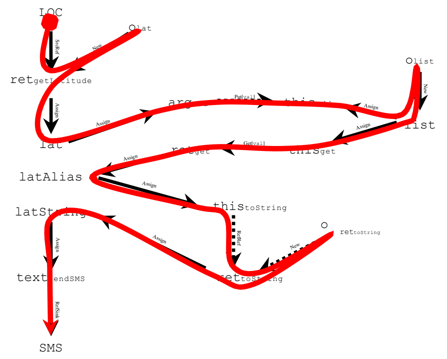
Taint Analysis for Android Apps

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keya
  bool with_filenames;
```

```
  // ...
}
```

- Native code
- Java reflection

Input: Android app and framework

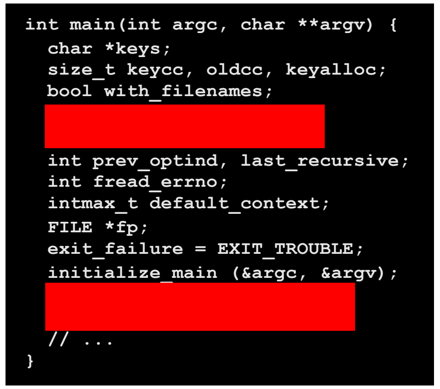


Step 1: Static taint analysis

location → Internet
SMS → Internet

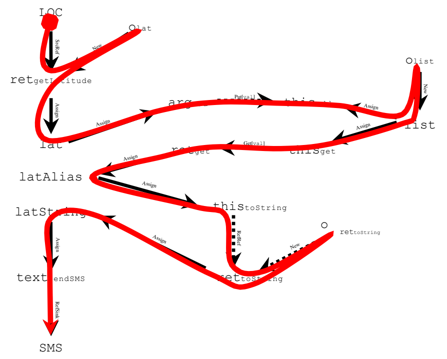
Output: Malicious behavior

Taint Analysis for Android Apps



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);

  // ...
}
```

Input: Android app and framework

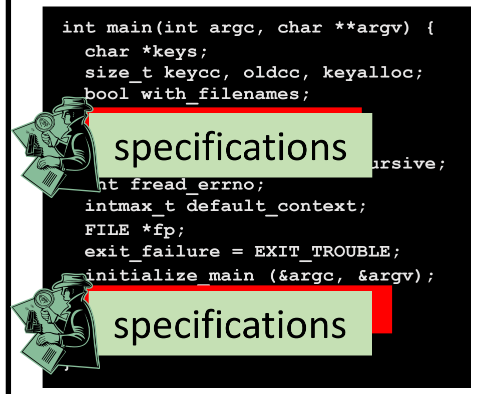


Step 1: Static taint analysis

location → Internet

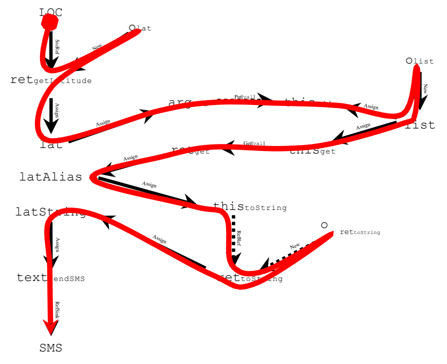
Output: Partial malicious behavior

Android Framework Specifications



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
                                    ursive;
  nt fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

Input: Android app and specifications

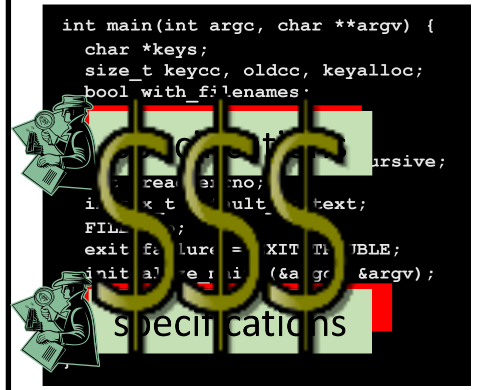


Step 1: Static taint analysis

location → Internet
SMS → Internet

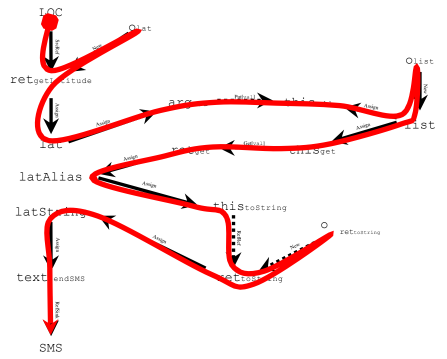
Output: Malicious behavior

Android Framework Specifications



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [illegible]ursive;
  [illegible] read[illegible]no;
  i[illegible]x_t [illegible]ult_[illegible]text;
  FIL[illegible];
  exit_fa[illegible]lure = [illegible]XIT_T[illegible]UBLE;
  init[illegible]al[illegible]e_[illegible] (&a[illegible]gc[illegible] &argv);
}
```

Input: Android app and specifications

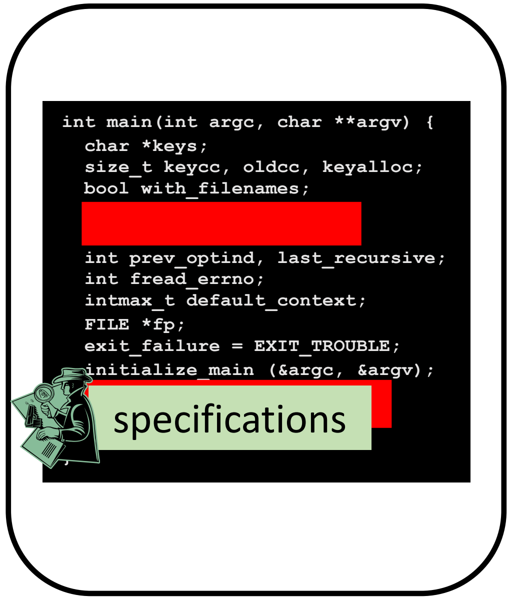


Step 1: Static taint analysis

location → Internet
SMS → Internet

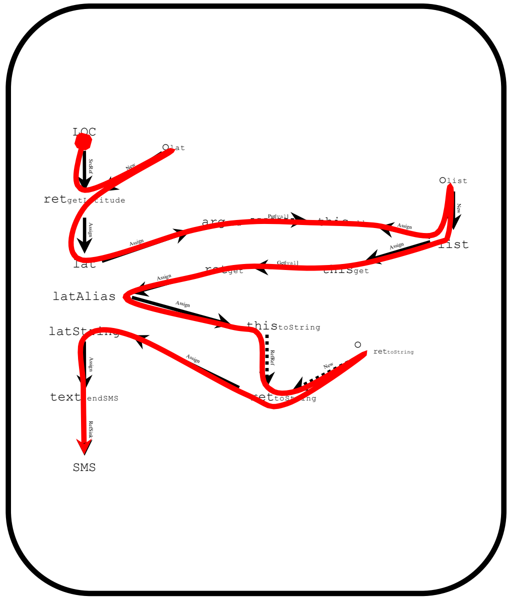
Output: Malicious behavior

Android Framework Specifications

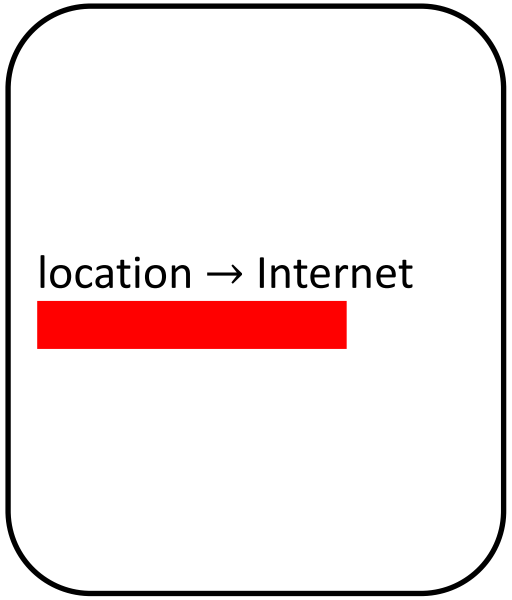


```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

Input: Android app and partial specifications



Step 1: Static taint analysis



location → Internet

Output: Partial malicious behavior

Our Goal

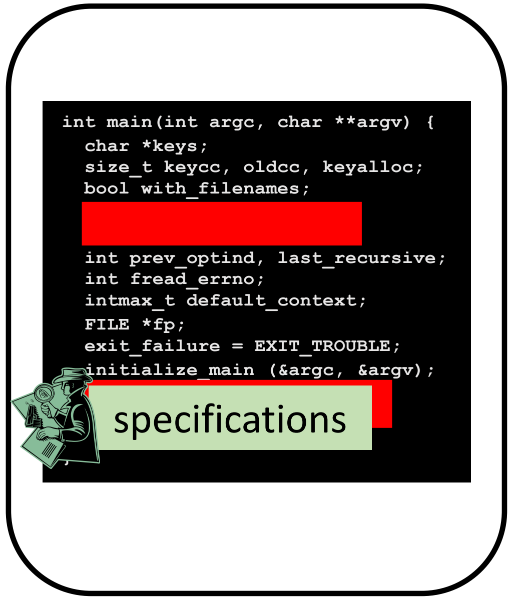
- Ensure soundness despite missing specifications

- Using runtime checks (dynamic soundness)
  - Dynamic policy enforcement (Enck 2010)
  - Debugging (Liblit 2005, Jin 2012)
  - …

- Improve soundness of static analysis over time

Our Approach: Eventual Soundness

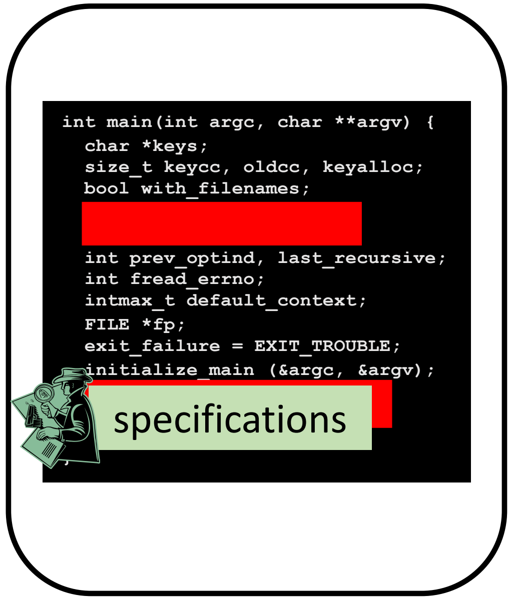
Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

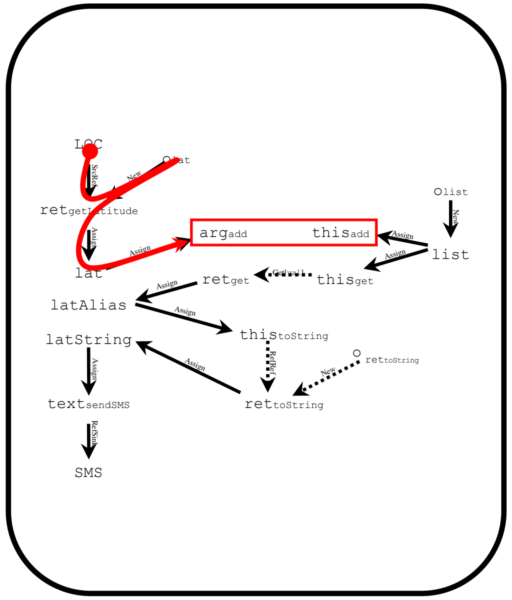
Input: Android app

Our Approach: Eventual Soundness



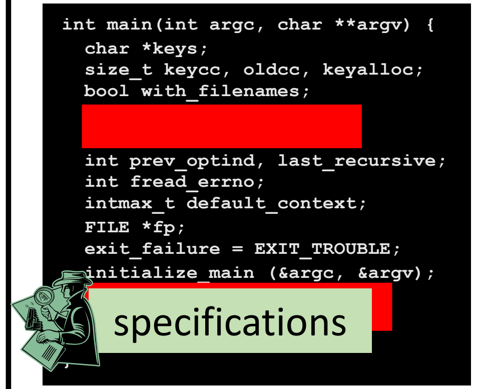
```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  [redacted]
```

Input: Android app



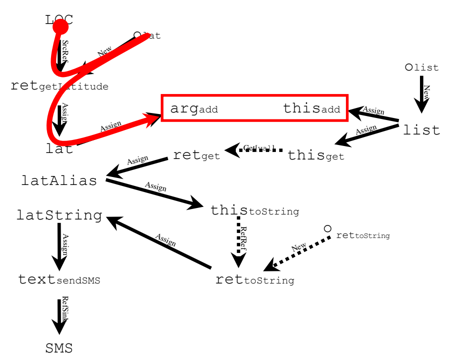
Step 1: Optimistic static analysis

Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

Input: Android app

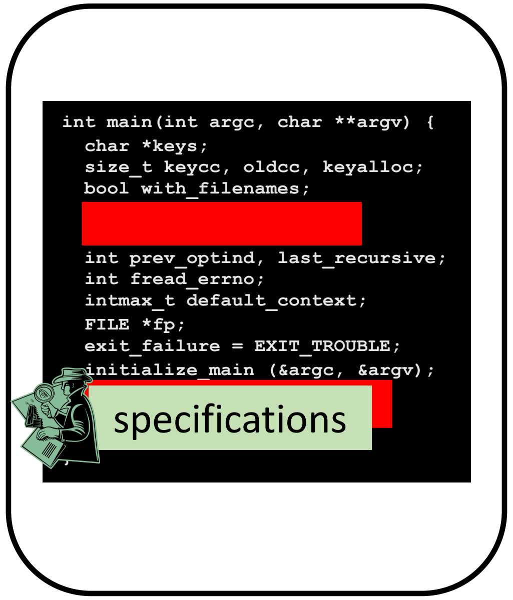


Step 1: Optimistic static analysis

location → Internet

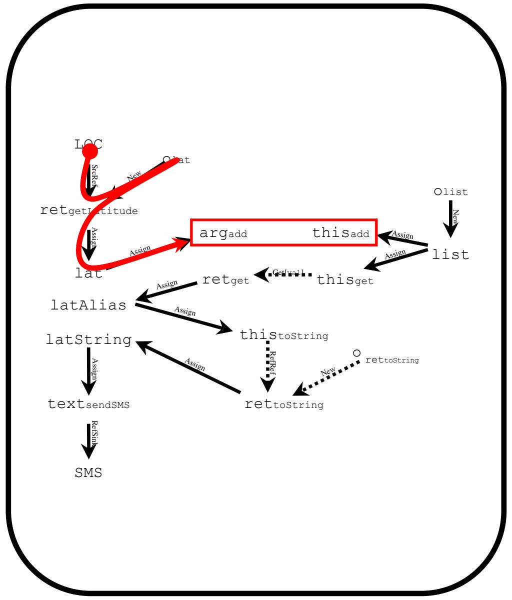
Output: Partial malicious behaviors

Our Approach: Eventual Soundness

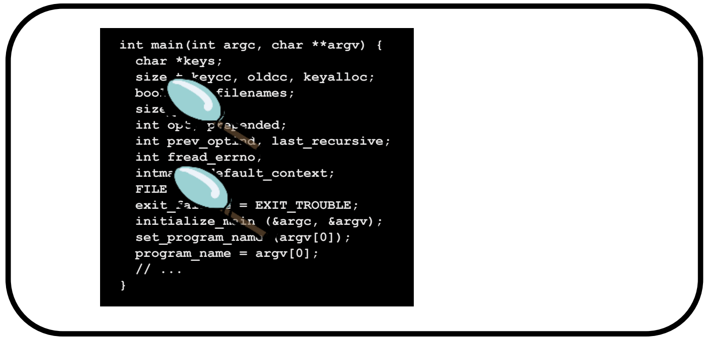


```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  [redacted]
```

Input: Android app



Step 1: Optimistic static analysis



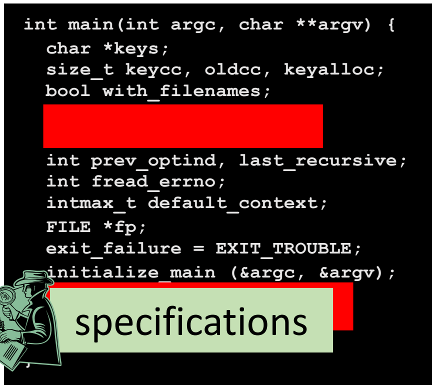
```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  boo[illegible]filenames;
  size[illegible]
  int opt, prepended;
  int prev_optind, last_recursive;
  int fread_errno;
  intma[illegible]default_context;
  FILE
  exit_fa[illegible] = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Step 2: Monitor for counter-examples

location → Internet
[redacted]

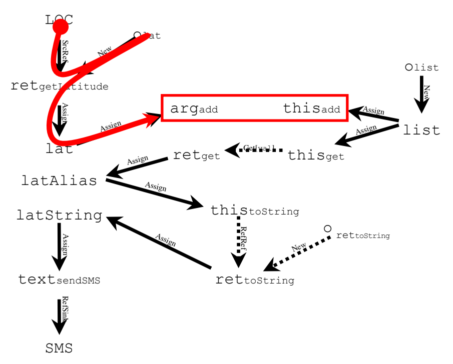
Output: Partial malicious behaviors

Our Approach: Eventual Soundness

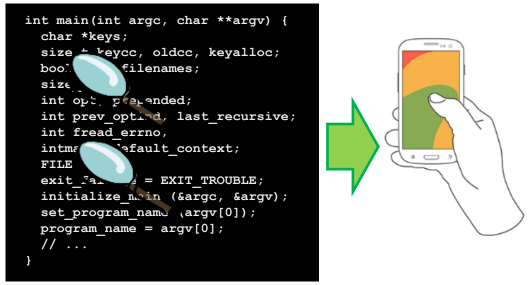


```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  [redacted]
```

Input: Android app



Step 1: Optimistic static analysis



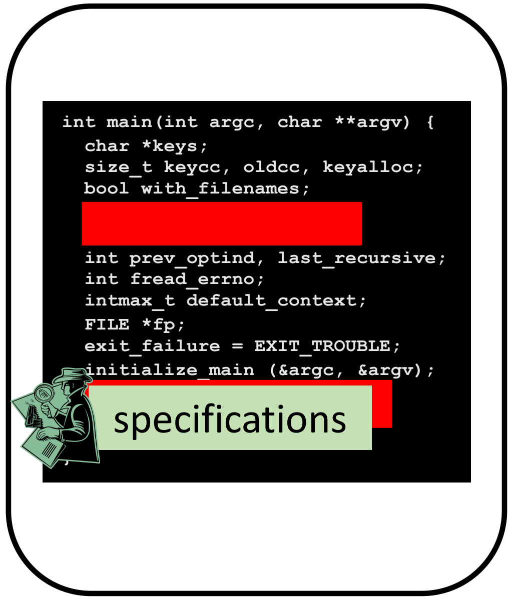
```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  boo[illegible] filenames;
  size[illegible]
  int opt, prepended;
  int prev_optind, last_recursive;
  int fread_errno;
  intma[illegible]default_context;
  FILE
  exit_fa[illegible] = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Step 2: Monitor for counter-examples

location → Internet
[redacted]

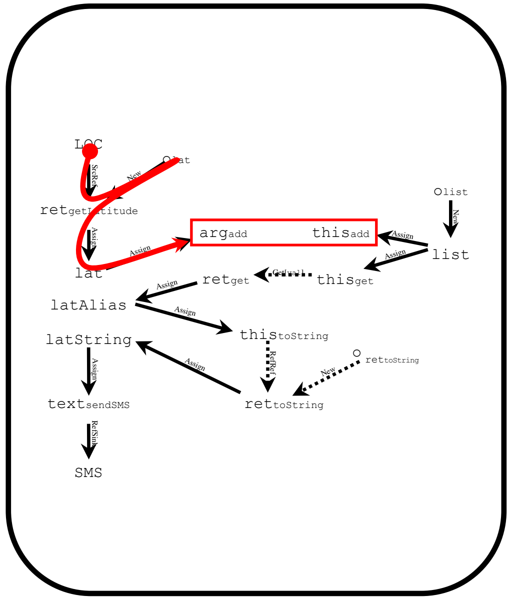
Output: Partial malicious behaviors

Our Approach: Eventual Soundness

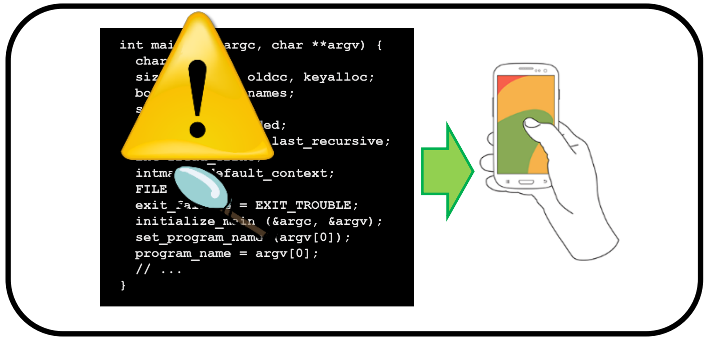


```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  [redacted]
```

Input: Android app



Step 1: Optimistic static analysis



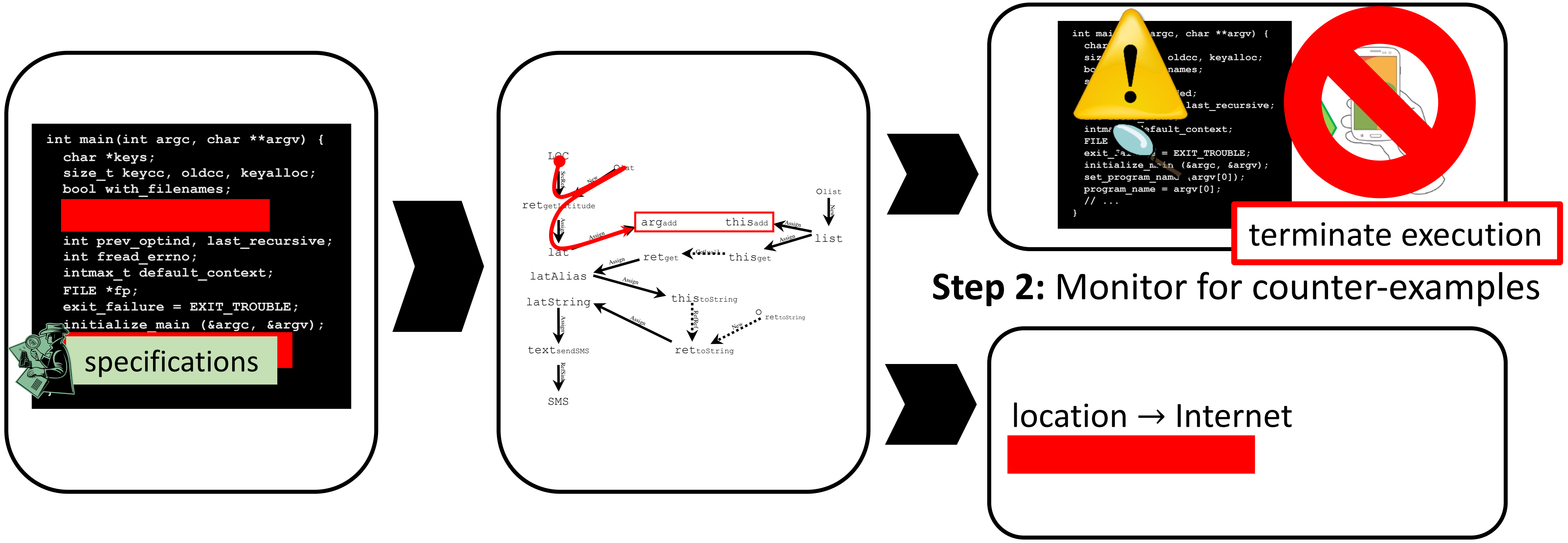
```
int mai    argc, char **argv) {
  char
  siz        oldcc, keyalloc;
  bo         names;
  s
             ed;
             last_recursive;

  intma   efault_context;
  FILE
  exit_fa    = EXIT_TROUBLE;
  initialize_m  n (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Step 2: Monitor for counter-examples

location → Internet
[redacted]

Output: Partial malicious behaviors

Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

```
int main(int argc, char **argv) {
  char
  siz        oldcc, keyalloc;
  bo         names;
  s
             ed;
             last_recursive;

  intma    default_context;
  FILE
  exit_fa    = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

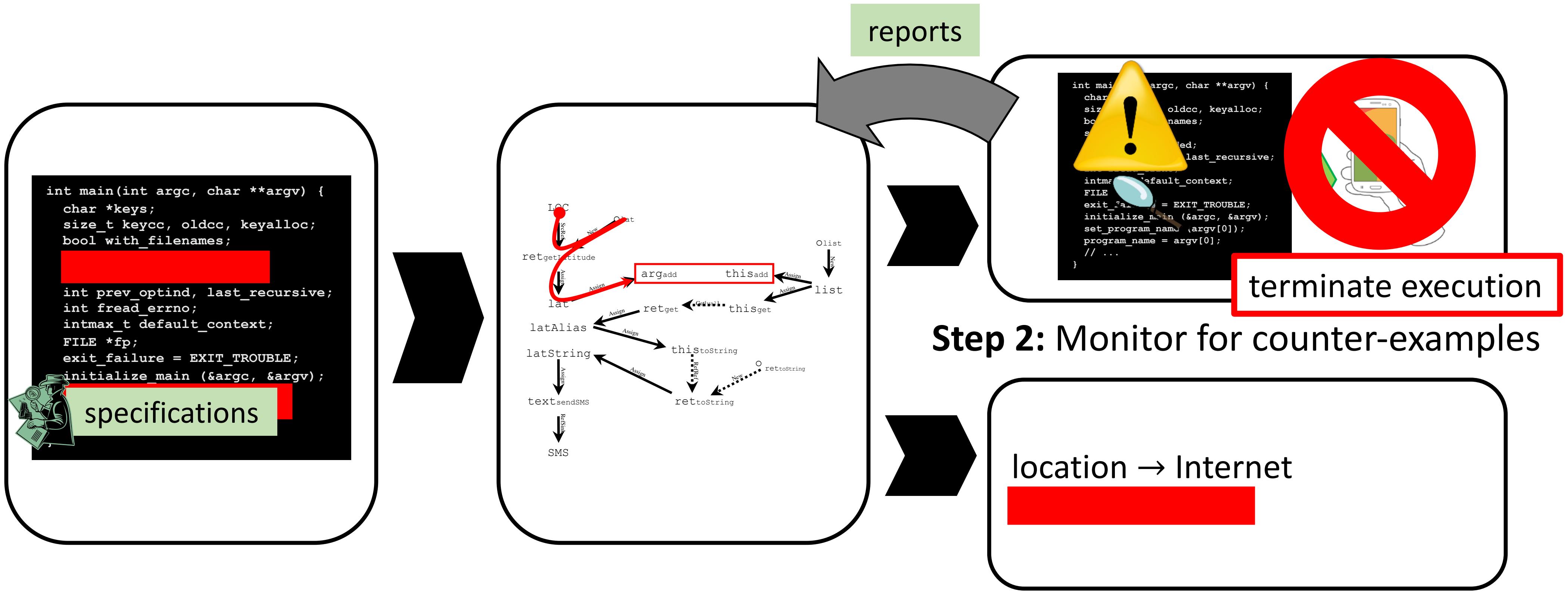
Step 2: Monitor for counter-examples

Input: Android app

Step 1: Optimistic static analysis

Output: Partial malicious behaviors

Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

specifications
LOC
$\mathrm{ret}_{\mathrm{getLatitude}}$
lat
latAlias
latString
$\mathrm{text}_{\mathrm{sendSMS}}$
SMS
$\mathrm{arg}_{\mathrm{add}}$
$\mathrm{this}_{\mathrm{add}}$
O_{list}
list
$\mathrm{ret}_{\mathrm{get}}$
$\mathrm{this}_{\mathrm{get}}$
$\mathrm{this}_{\mathrm{toString}}$
$\mathrm{ret}_{\mathrm{toString}}$
reports
terminate execution
location → Internet

[redacted]

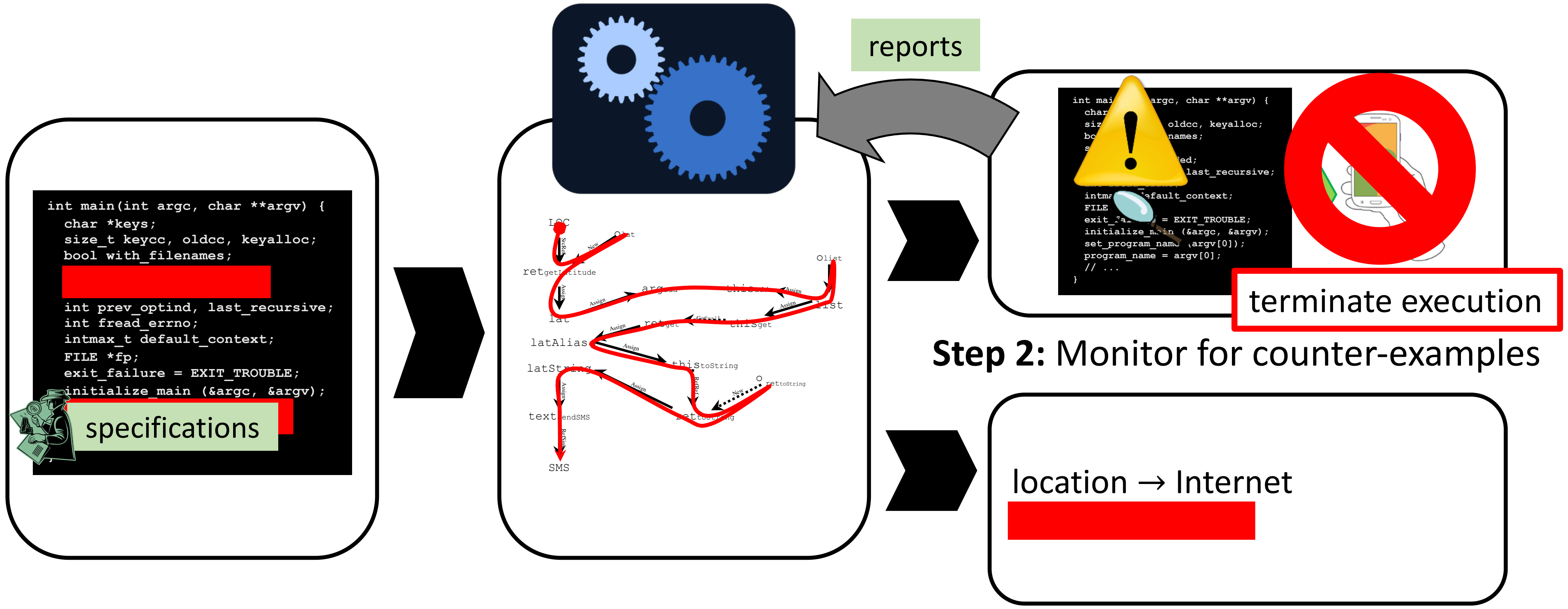
Step 2: Monitor for counter-examples

Input: Android app

Step 1: Optimistic static analysis

Output: Partial malicious behaviors

Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  [redacted]
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  [redacted]
```

```
int mai      argc, char **argv) {
  char
  siz          oldcc, keyalloc;
  bo           names;
  s              ed;
               last_recursive;
  intma   efault_context;
  FILE
  exit_fai     = EXIT_TROUBLE;
  initialize_m  in (&argc, &argv);
  set_program_nam  (argv[0]);
  program_name = argv[0];
  // ...
}
```

[redacted]

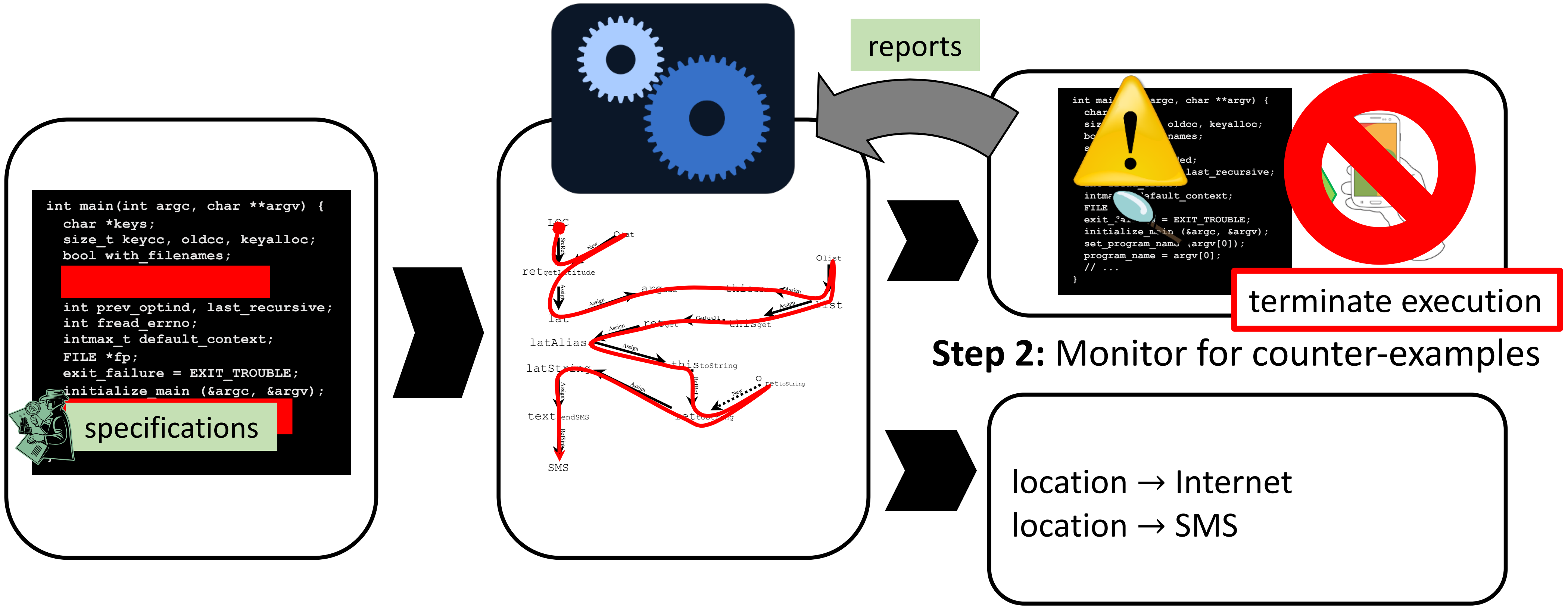
Input: Android app

Step 1: Updated static analysis

Step 2: Monitor for counter-examples

Output: Partial malicious behaviors

Our Approach: Eventual Soundness



```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;

  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
```

Step 2: Monitor for counter-examples

Input: Android app

Step 1: Updated static analysis

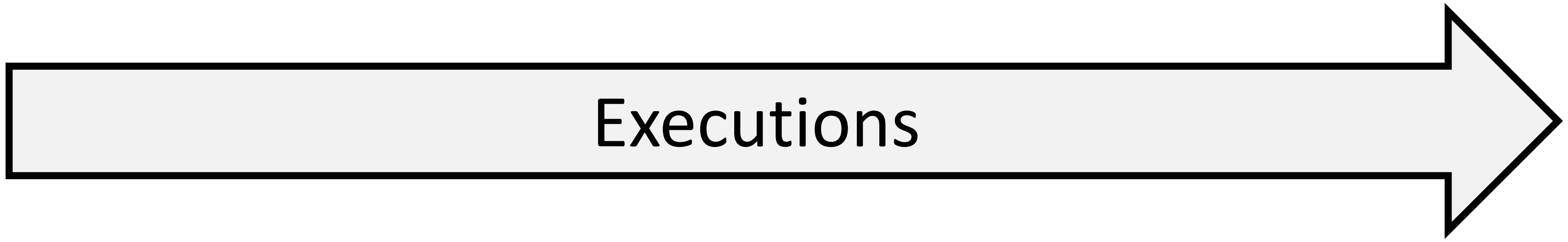
Output: Malicious behaviors

Guarantees

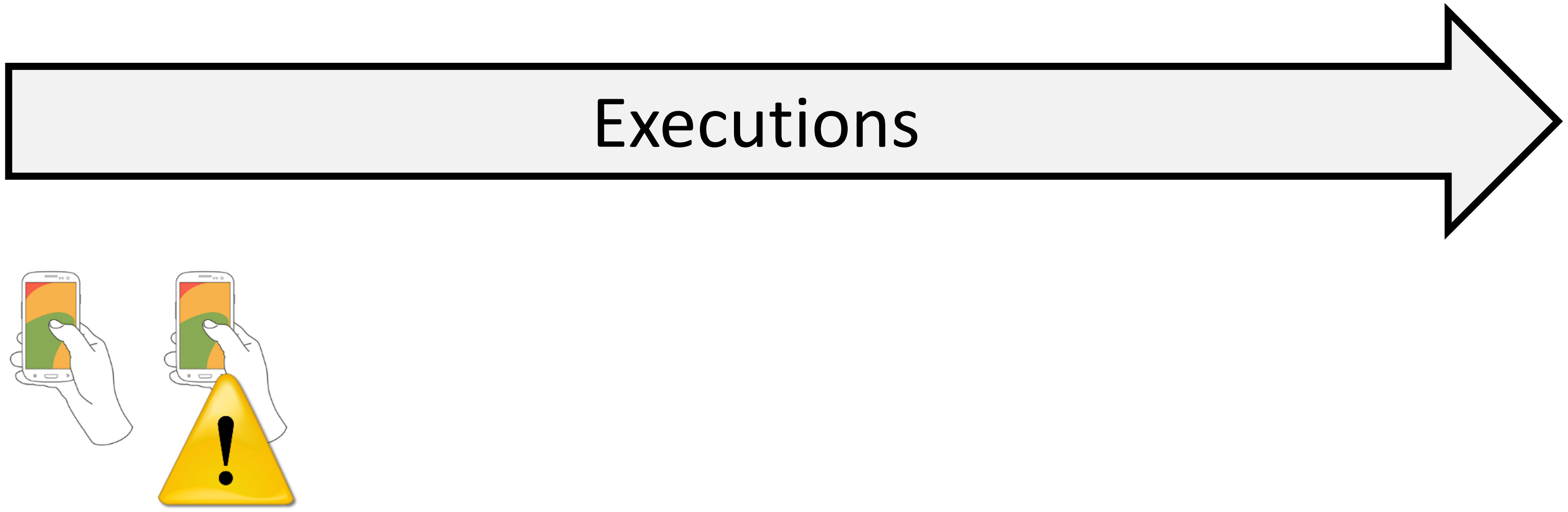
- **Eventual soundness**
  - Dynamic soundness via runtime checks
  - Eventually, the static analysis results are sound for all subsequent executions

Eventual Soundness

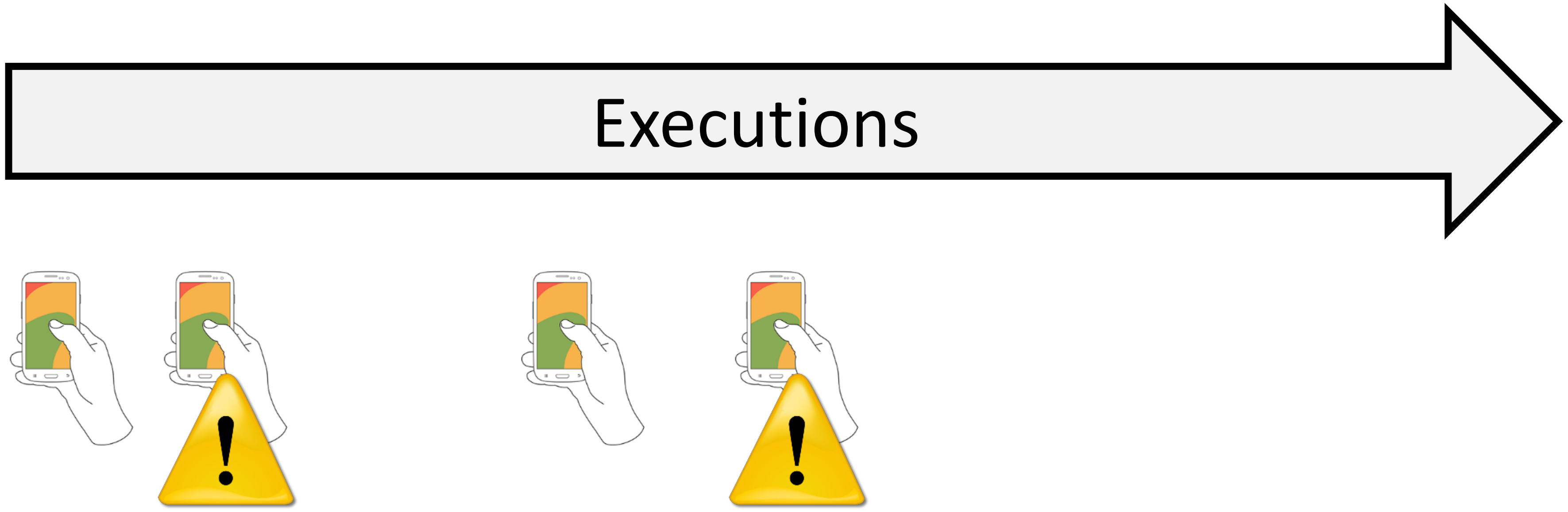
Eventual Soundness



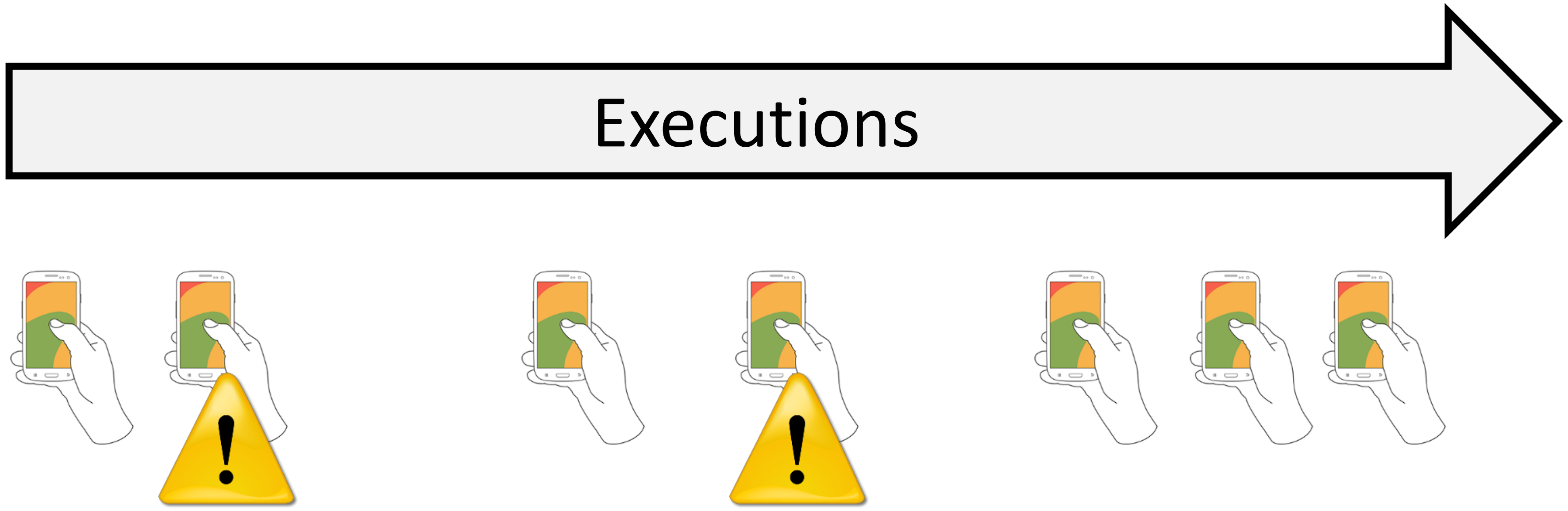
Eventual Soundness



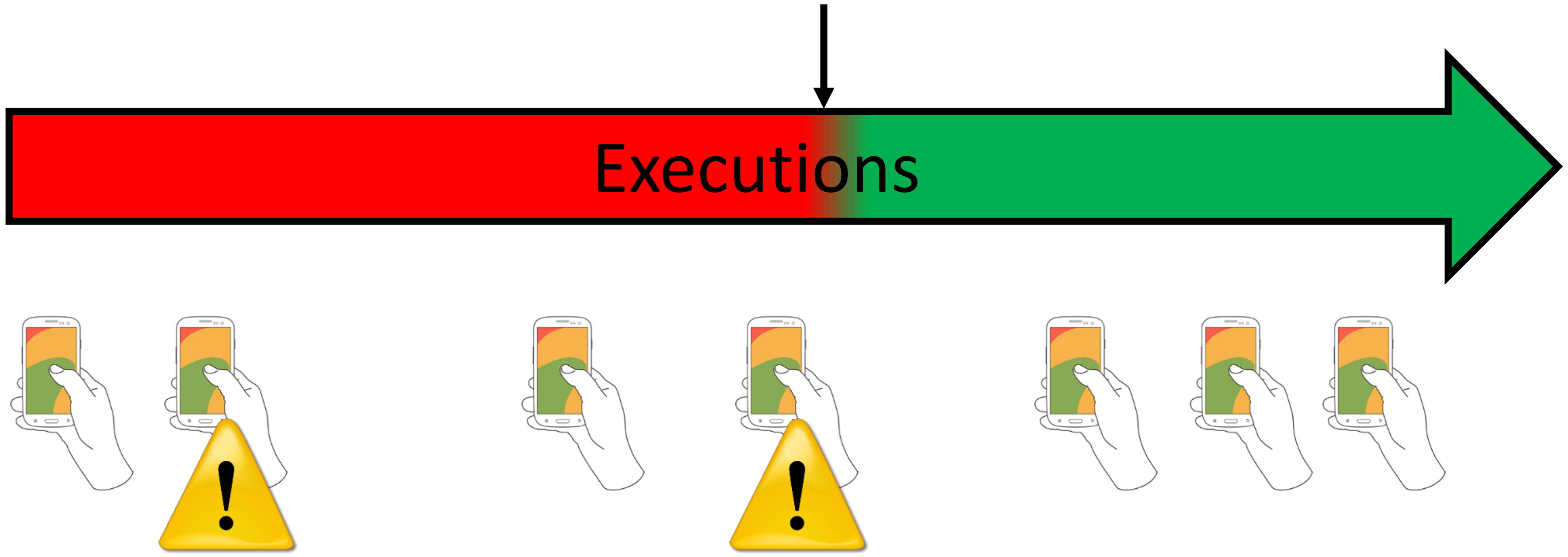
Eventual Soundness



Eventual Soundness



Eventual Soundness



Guarantees

- **Eventual soundness**
  - Dynamic soundness via runtime checks
  - Eventually, the static analysis results are sound for all subsequent executions

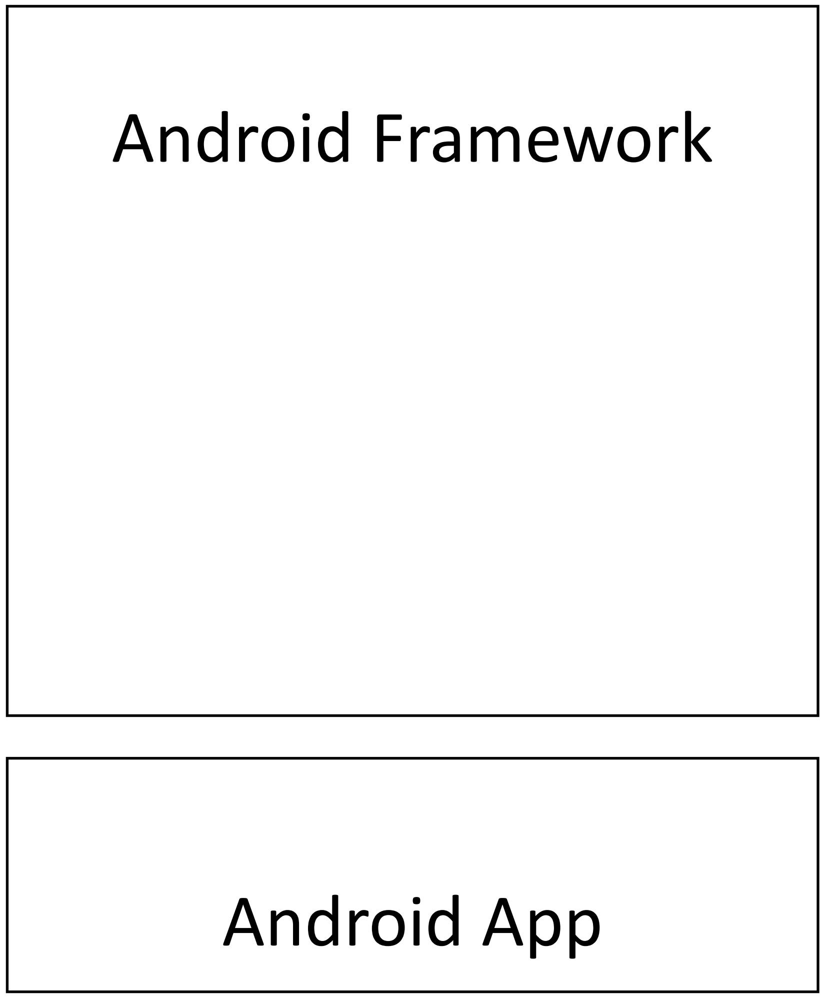
- **Precise**
  - Relative to knowing all specifications beforehand

Eventually Sound Points-To Analysis

Taint Analysis for Android Apps

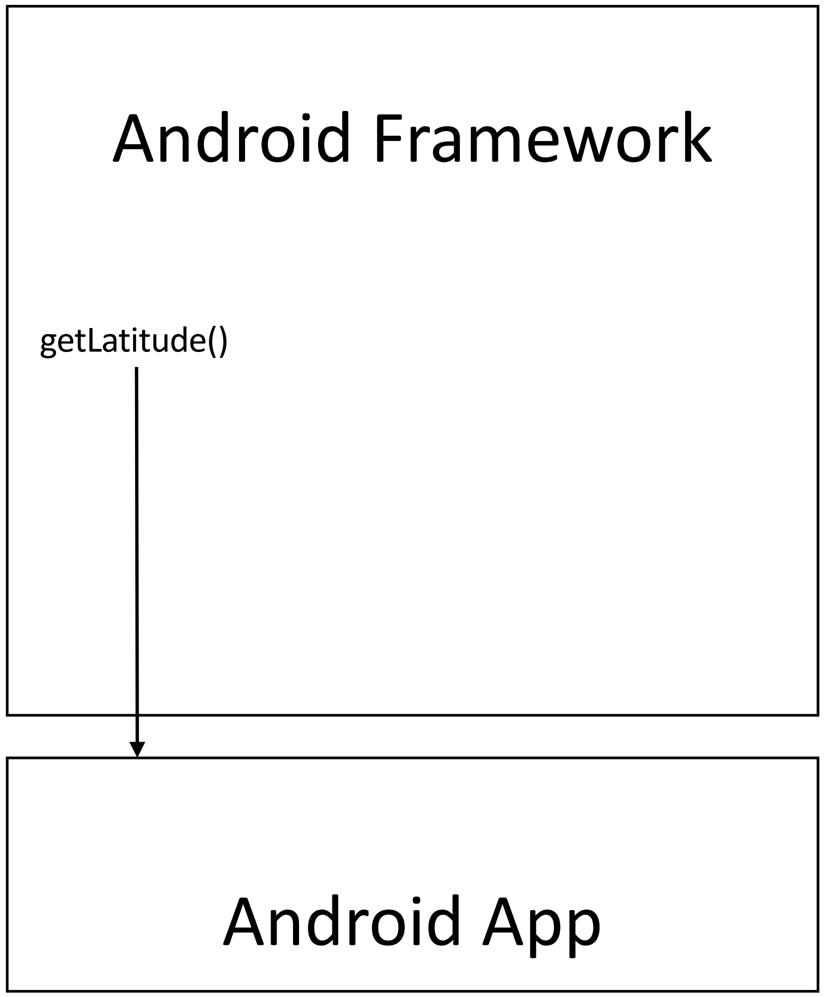
Taint Analysis for Android Apps

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```



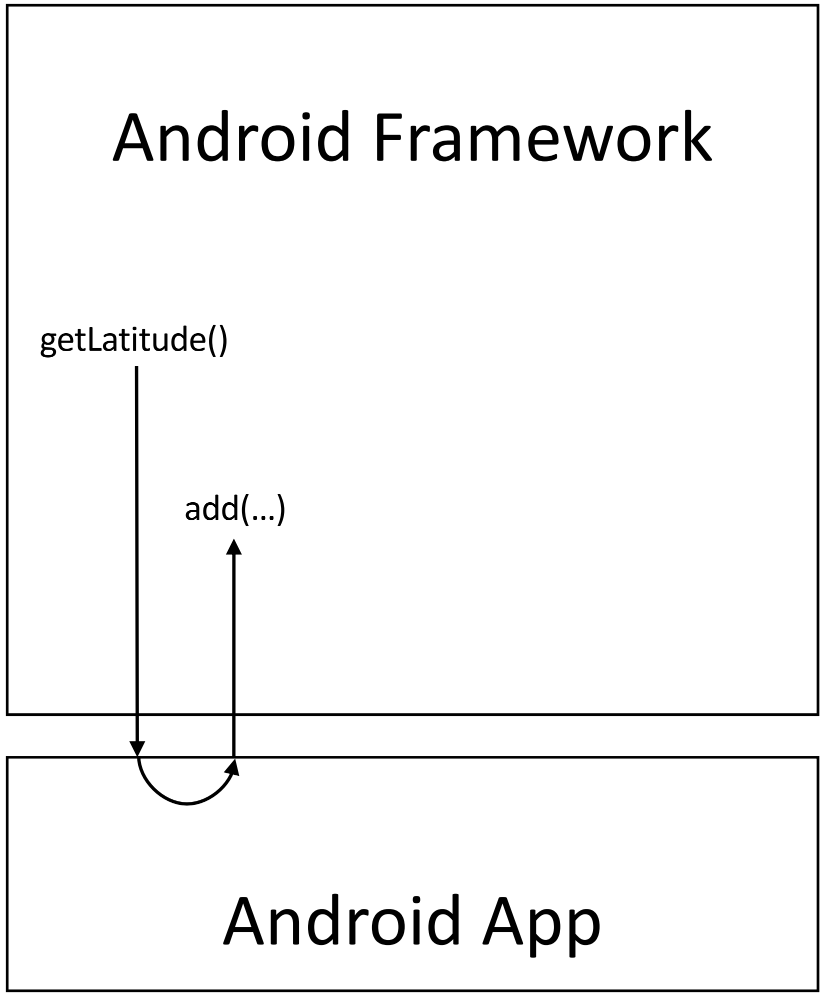
Taint Analysis for Android Apps

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```



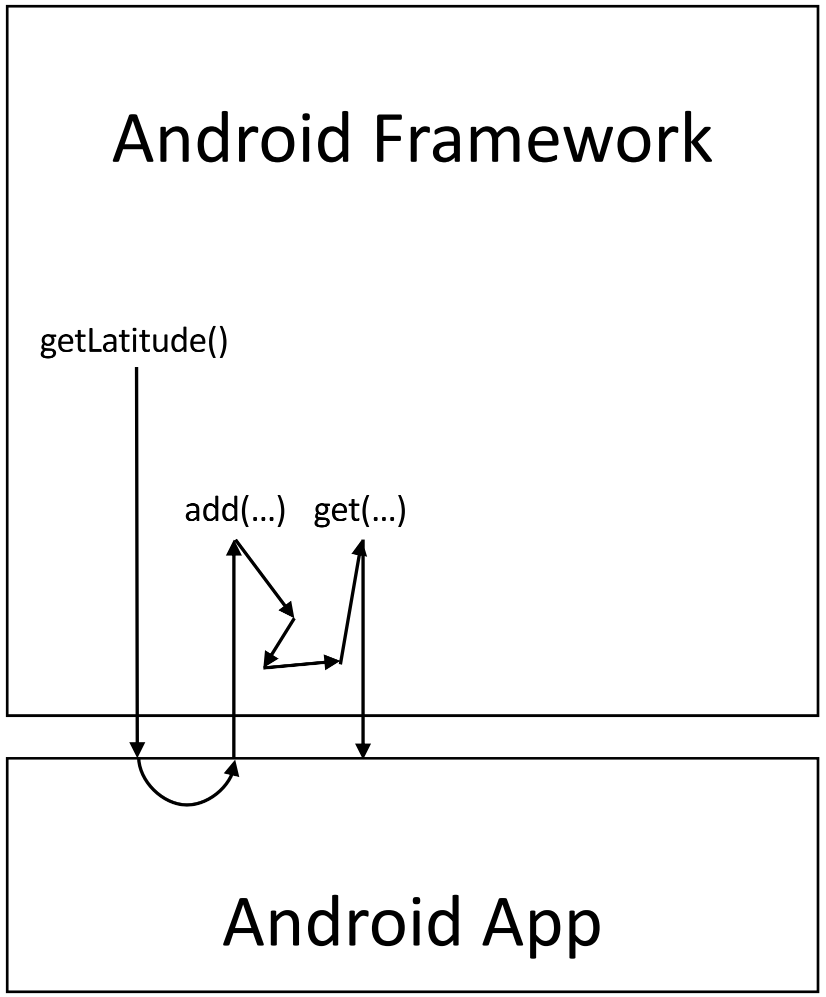
Taint Analysis for Android Apps

```
1. Double latitude = getLatitude();
2. List list = new List();
3. list.add(latitude);
4. Double data = list.get(0);
5. Double dataDup = data;
6. sendSMS(dataDup);
```



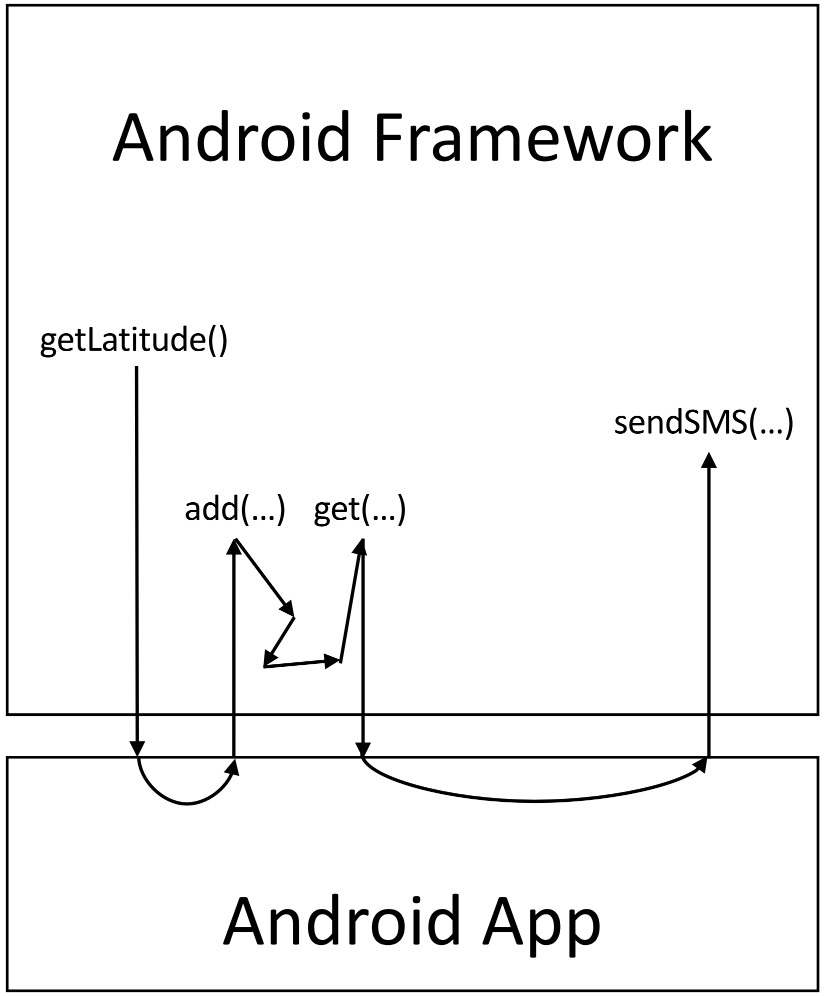
Taint Analysis for Android Apps

```
1. Double latitude = getLatitude();
2. List list = new List();
3. list.add(latitude);
4. Double data = list.get(0);
5. Double dataDup = data;
6. sendSMS(dataDup);
```



Taint Analysis for Android Apps

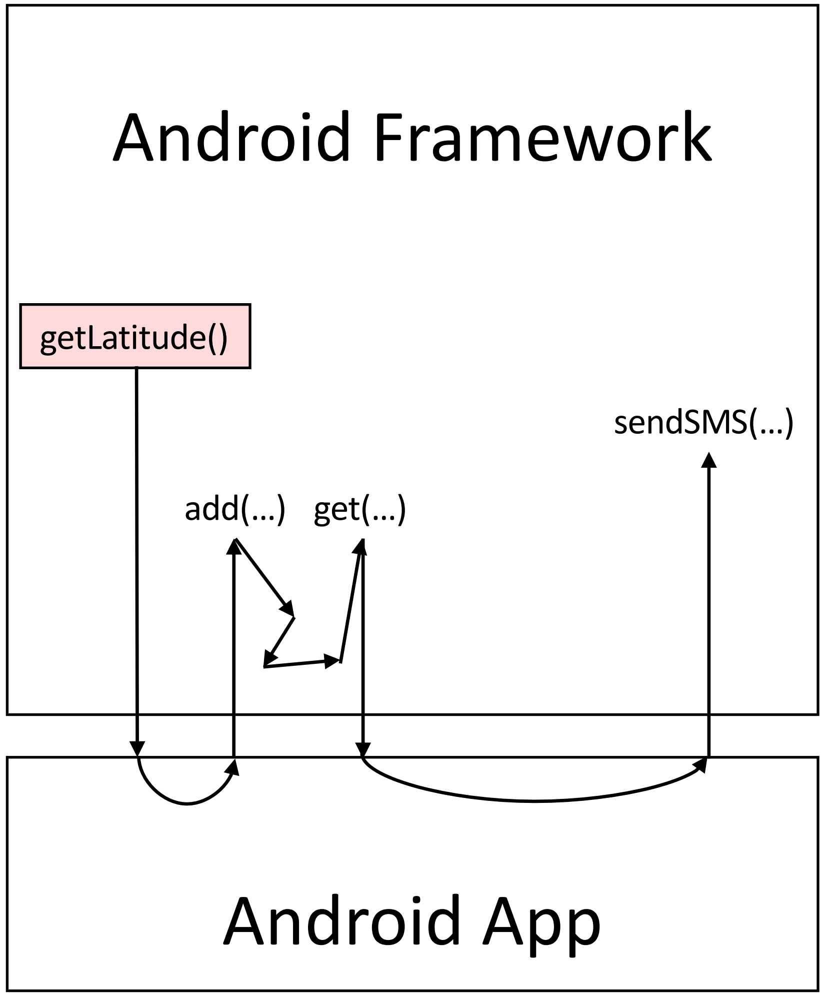
```
1. Double latitude = getLatitude();
2. List list = new List();
3. list.add(latitude);
4. Double data = list.get(0);
5. Double dataDup = data;
6. sendSMS(dataDup);
```



Taint Analysis for Android Apps

```
1.  Double latitude = getLatitude();
2.  List list = new List();
3.  list.add(latitude);
4.  Double data = list.get(0);
5.  Double dataDup = data;
6.  sendSMS(dataDup);
```

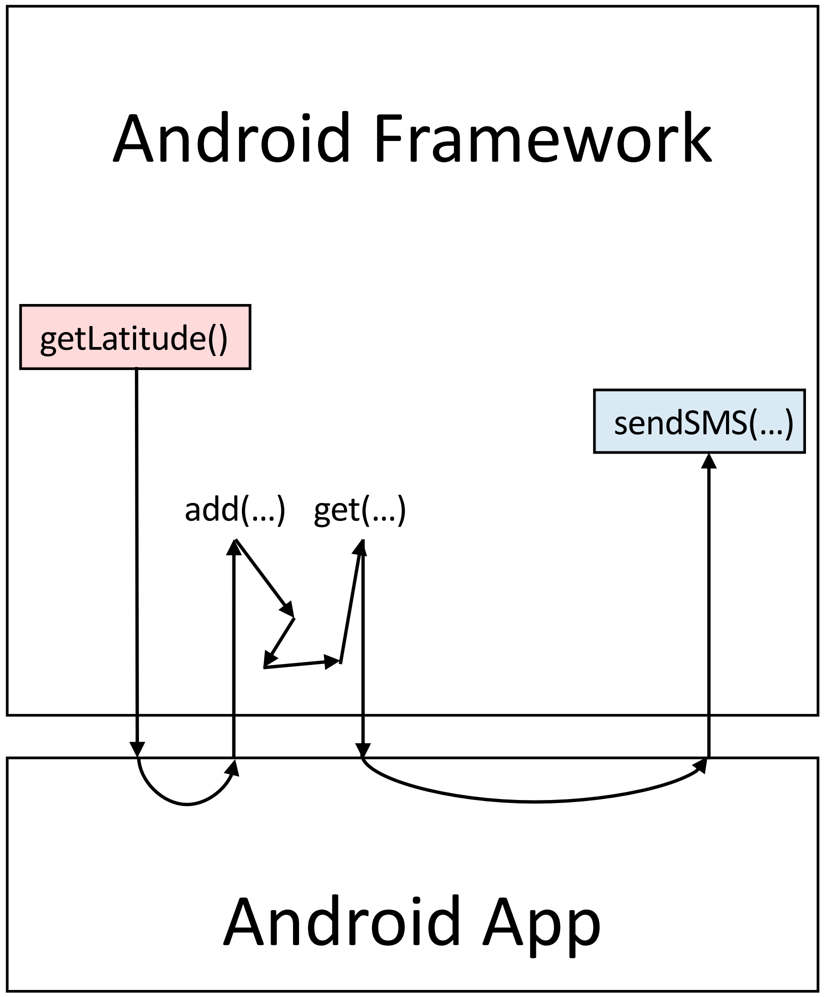
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() { … }
```



Taint Analysis for Android Apps

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

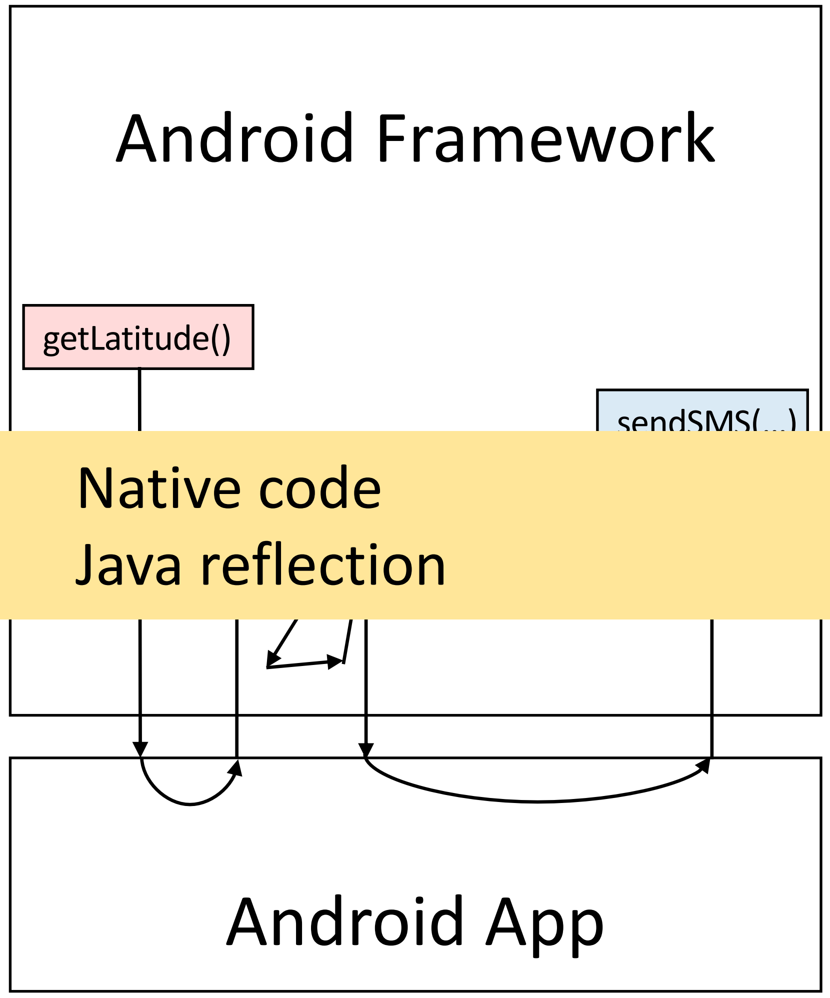
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() { … }
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) { … }
```



Taint Analysis for Android Apps

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() { … }
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) { … }
```

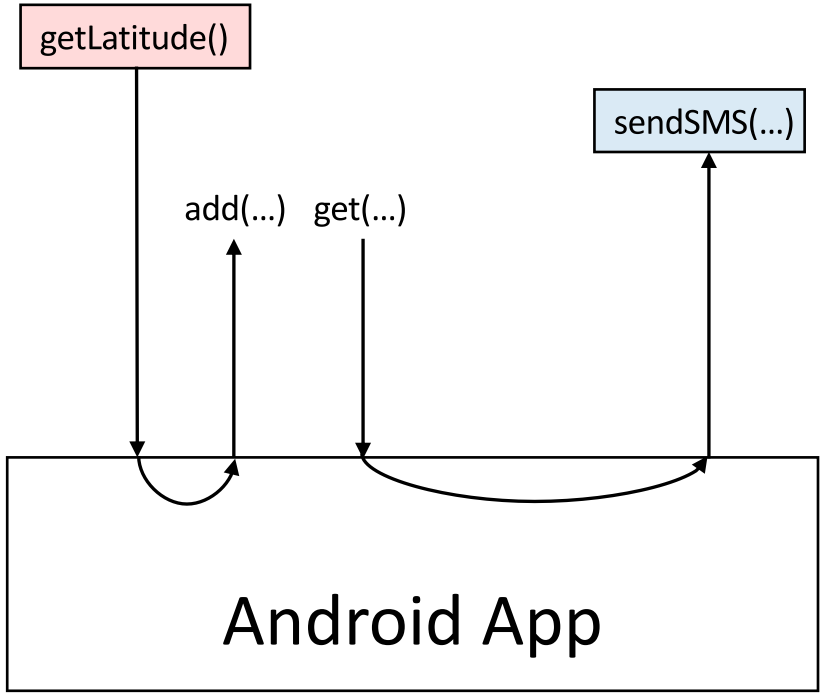


- Native code
- Java reflection

Points-To Specifications

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Points-To Specifications

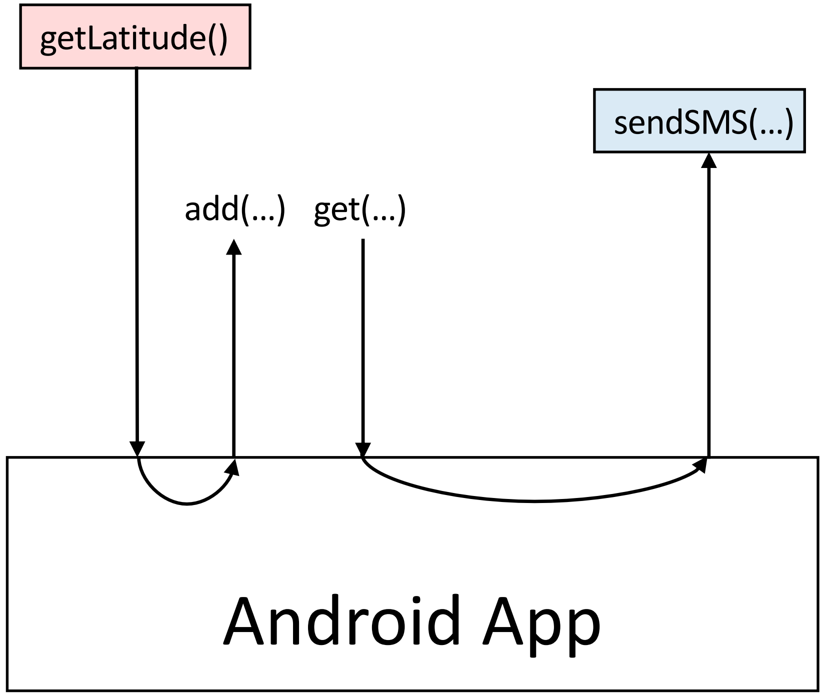
- Describes aliasing
  - @Alias(x, y) means "x may alias y"

-
  ```
  @Alias(add.arg, get.return)
  class List:
      void add(Object arg) {}
      Object get(Integer index) {}
  ```

Points-To Specifications

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

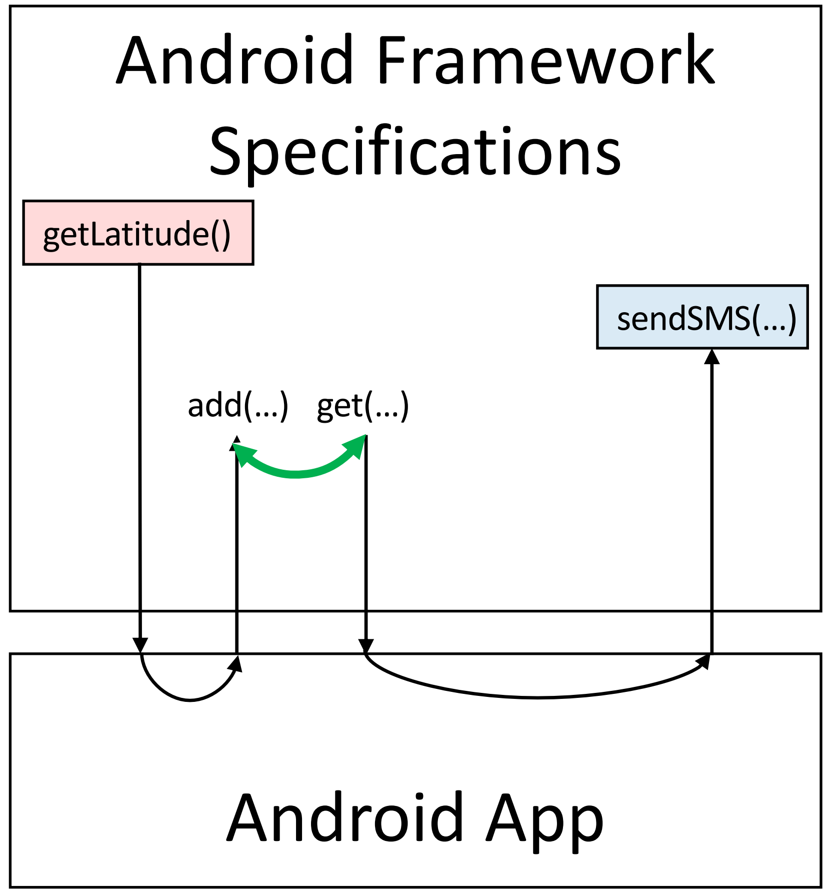
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Points-To Specifications

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

```
@Alias(add.arg, get.return)
class List:
    void add(Object arg) {}
    Object get(Integer index) {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Missing Specifications

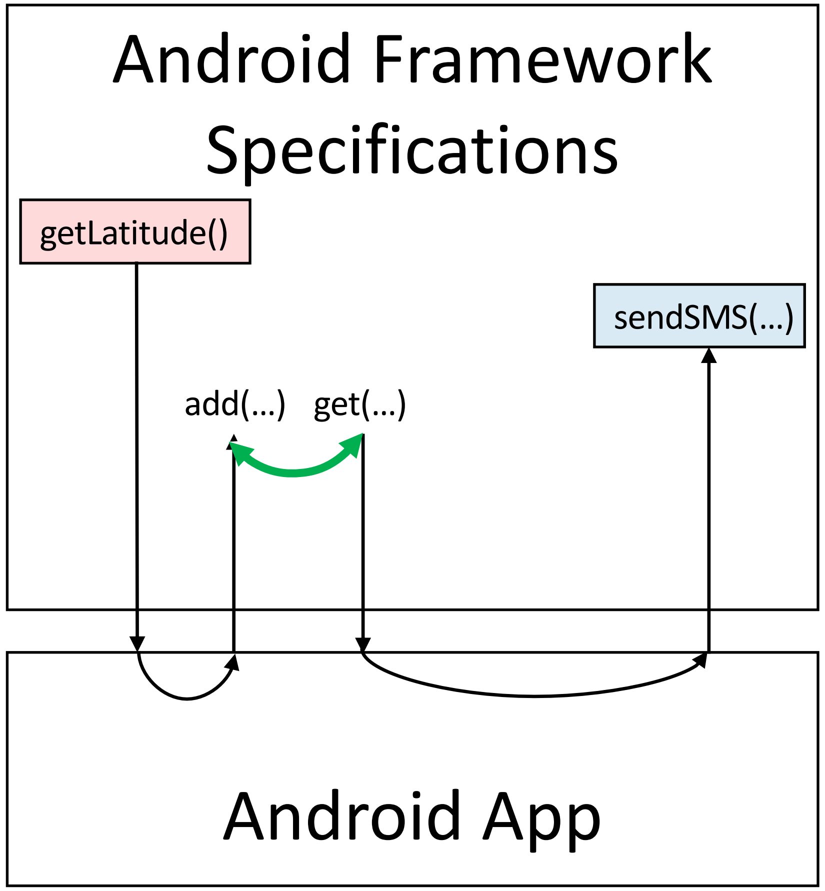
- Expensive to write specifications for every framework method
  - $\approx 4{,}000$ framework classes
  - A given app may use hundreds of classes

- Specifications typically written as needed
  - For a given app, only a few classes are relevant for finding taint flows
  - Our experience: specifications for ≈ 175 classes over course of a few years

Missing Specifications

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

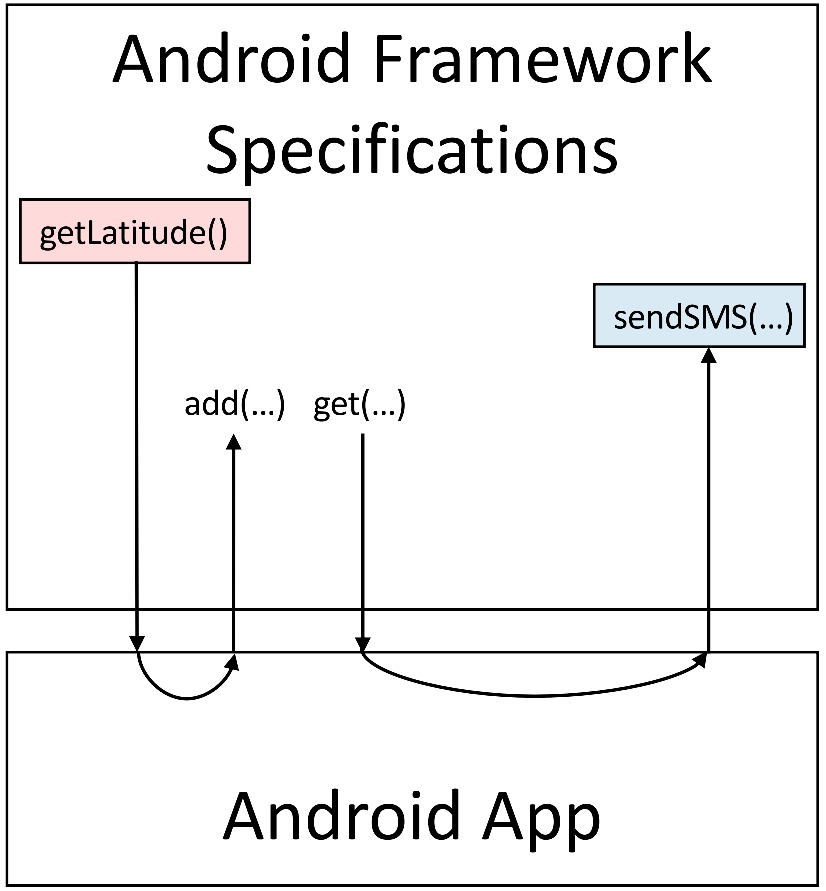
```
@Alias(add.arg, get.return)
class List:
    void add(Object arg) {}
    Object get(Integer index) {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Missing Specifications

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



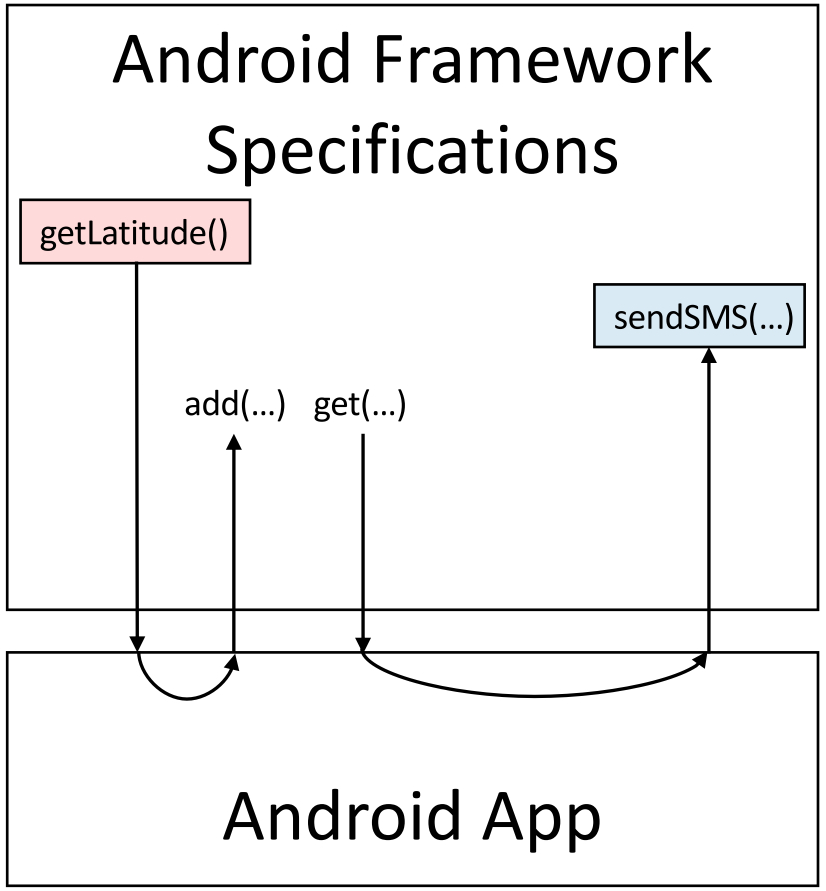
Our Approach: Eventual Soundness

- **Step 1:** Optimistic static analysis
- **Step 2:** Monitor for counter-examples
  - Report counter-examples detected during execution to static analysis
- **Step 3:** Update static analysis
  - Take into account detected counter-examples

Step 1: Optimistic Static Analysis

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

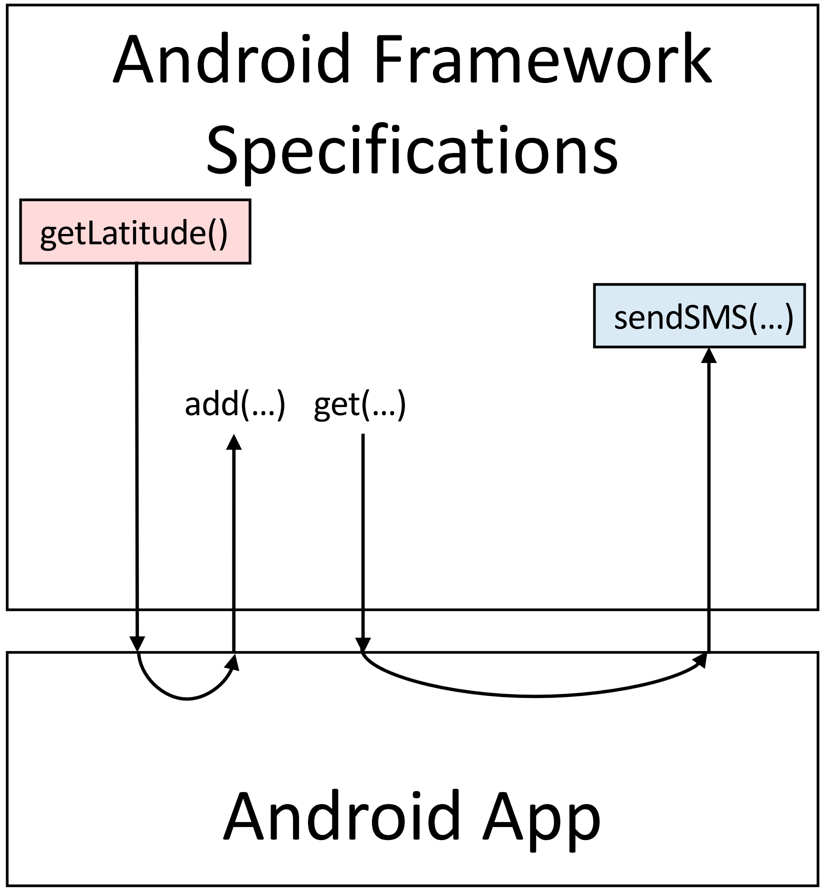
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step 2: Monitor for Counter-Examples

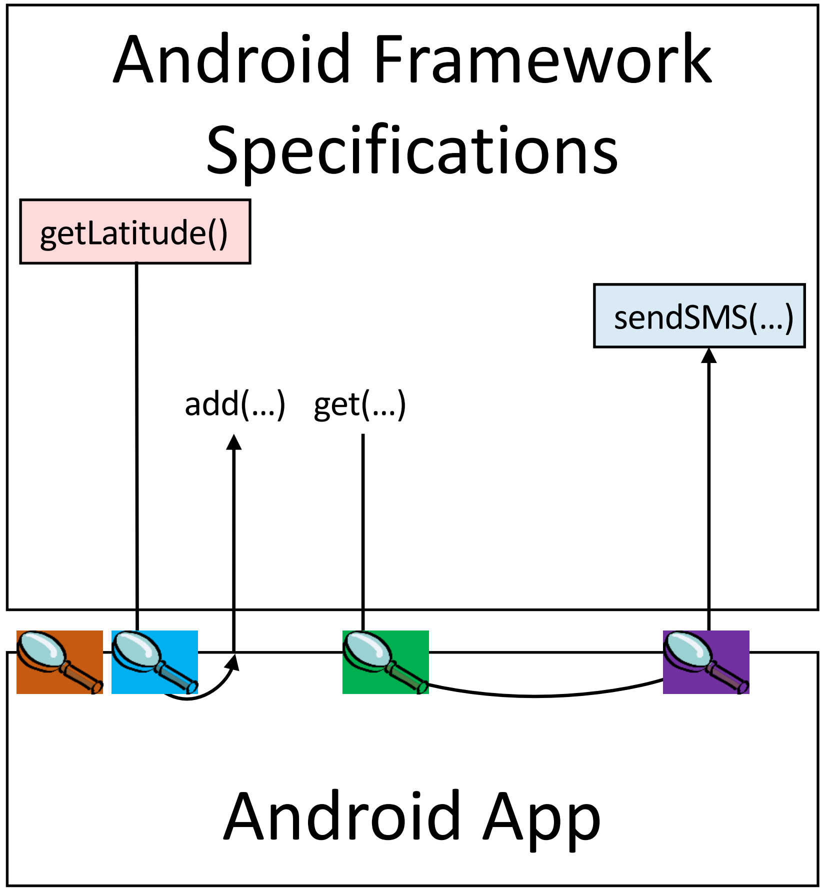
```
1. Double latitude = getLatitude();
2. List list = new List();
3. list.add(latitude);
4. Double data = list.get(0);
5. Double dataDup = data;
6. sendSMS(dataDup);
```

```
5. class LocationManager:
6.     @Flow(LOC, return)
7.     static String getLatitude() {}
8. class SMS:
9.     @Flow(text, SMS)
10.    static void sendSMS(String text) {}
```



Step 2: Monitor for Counter-Examples

```
   Double latitude = getLatitude();
   List list = new List();
3. list.add(latitude);
   Double data = list.get(0);
   Double dataDup = data;
6. sendSMS(dataDup);
```

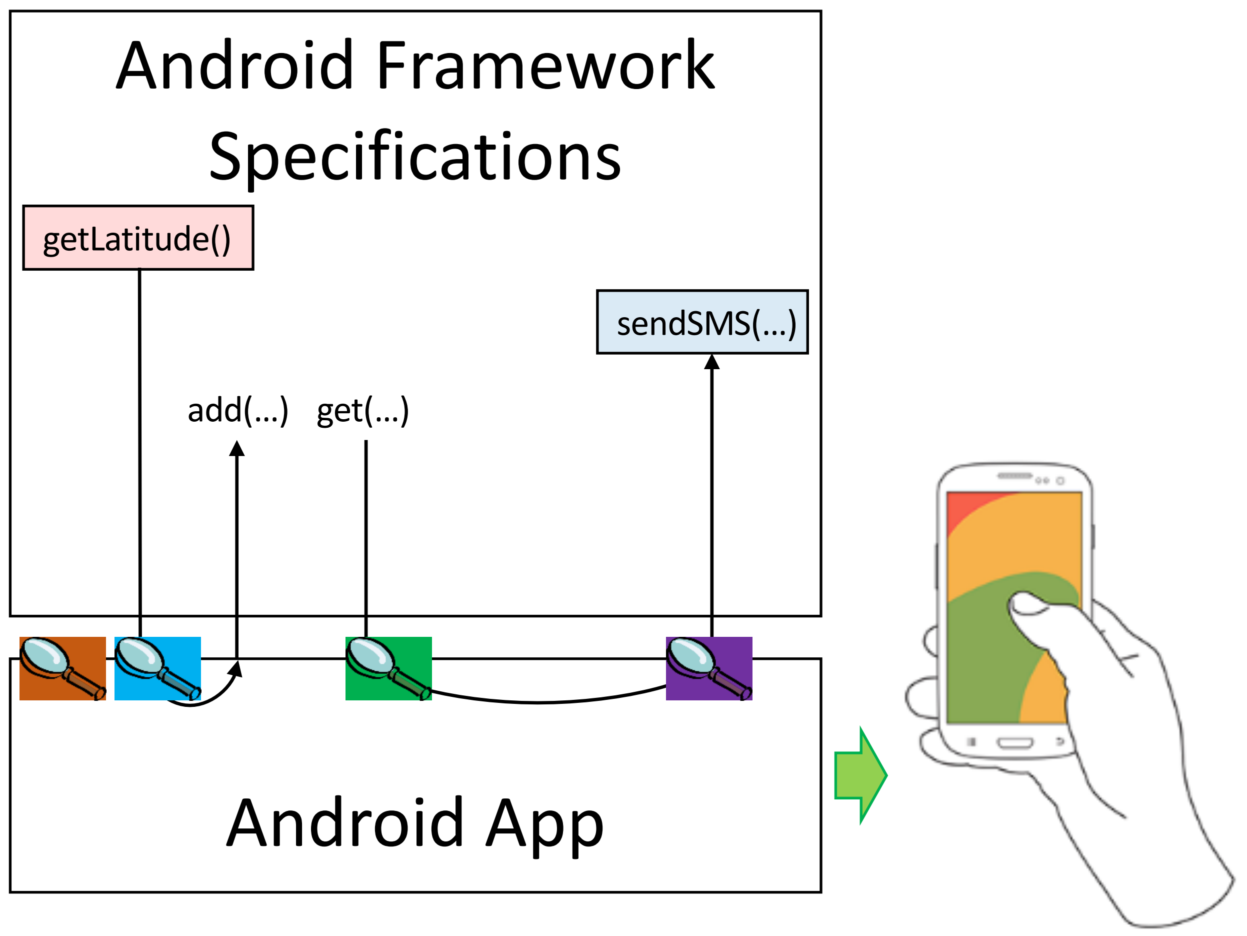


```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```

Step 2: Monitor for Counter-Examples

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

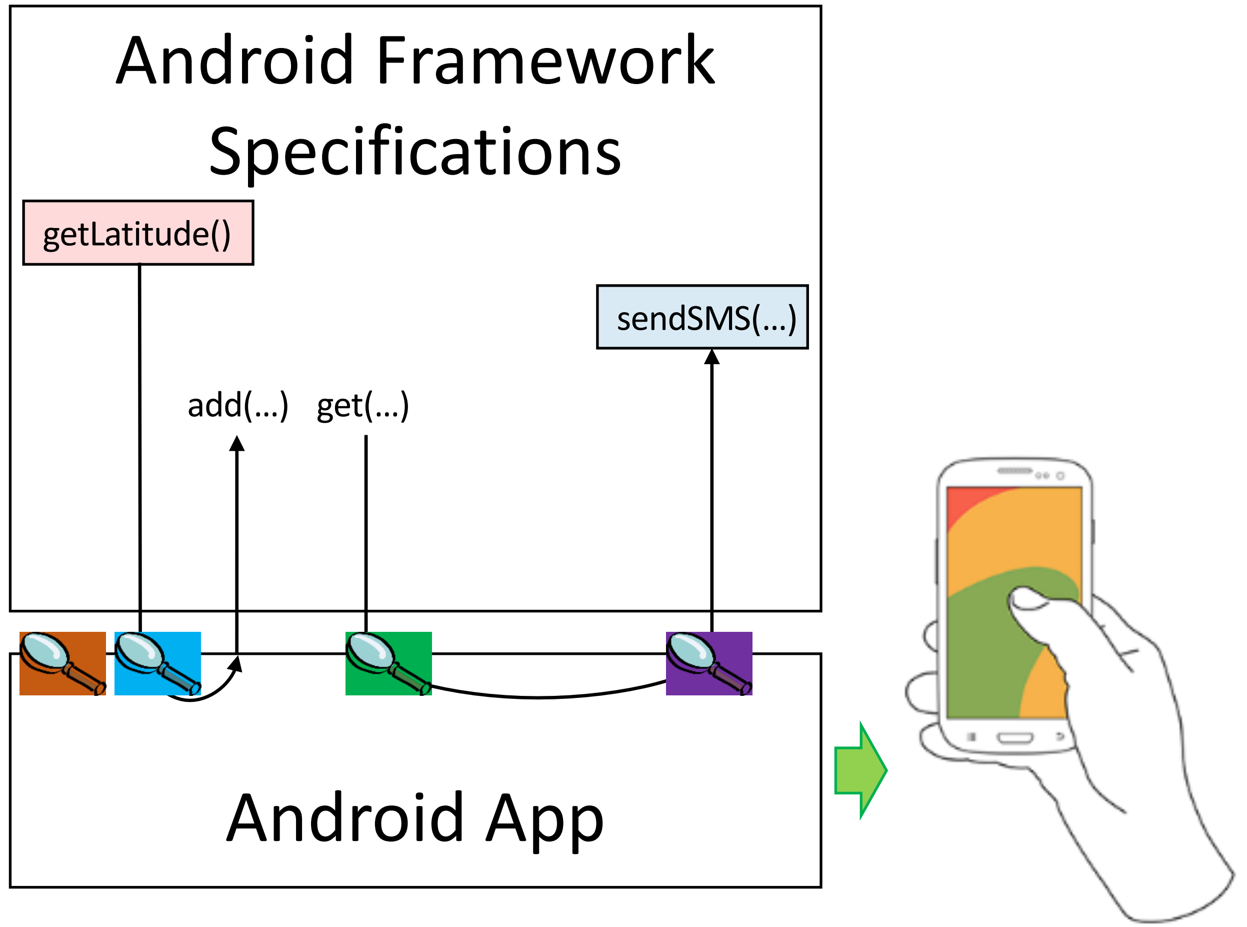
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step 2: Monitor for Counter-Examples

```
0xAAAAA  Double latitude = getLatitude();
         List list = new List();
         list.add(latitude);
         Double data = list.get(0);
         Double dataDup = data;
         sendSMS(dataDup);
```

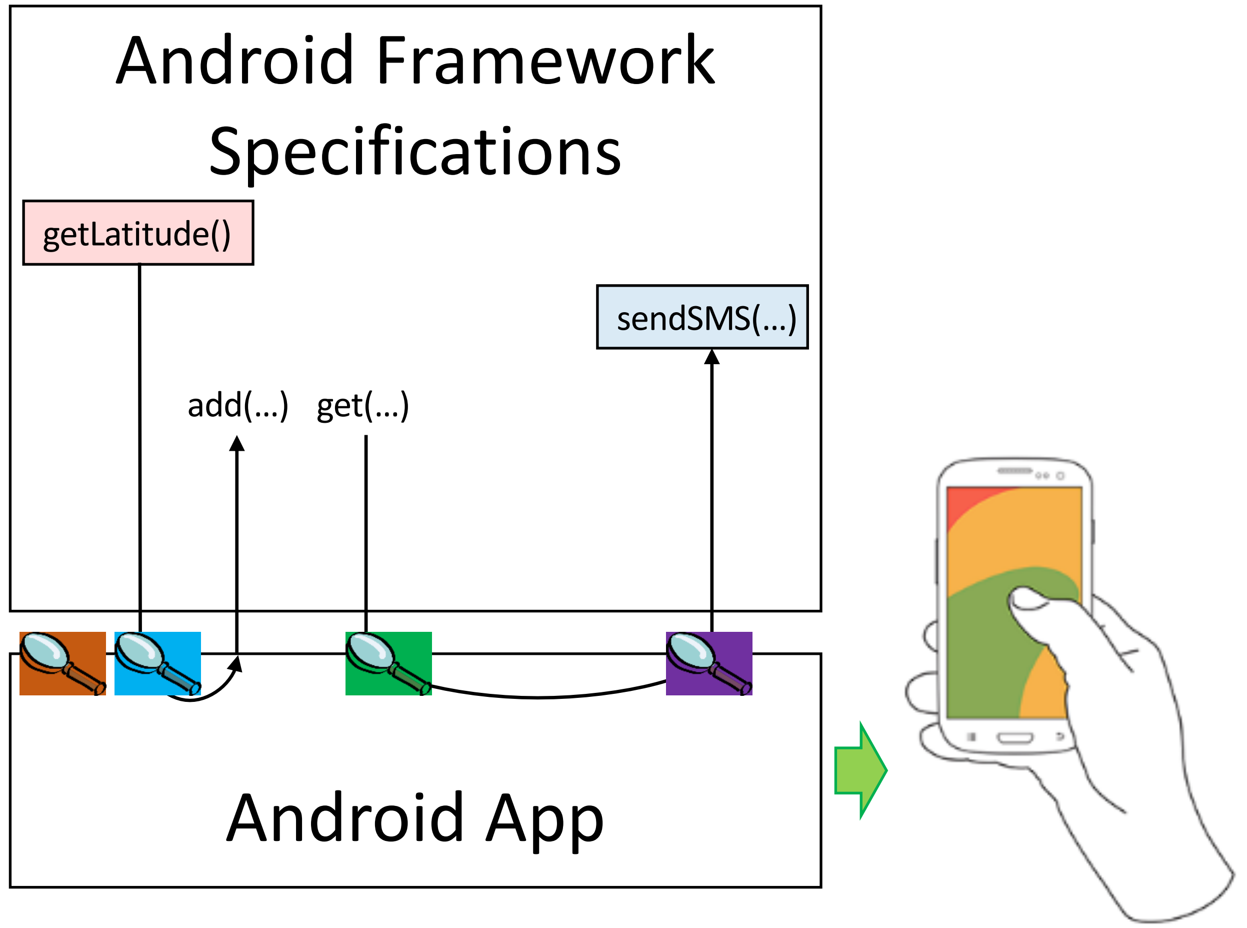
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step 2: Monitor for Counter-Examples

```
0xAAAAA  Double latitude = getLatitude();
0x00000  List list = new List();
         list.add(latitude);
         Double data = list.get(0);
         Double dataDup = data;
         sendSMS(dataDup);
```

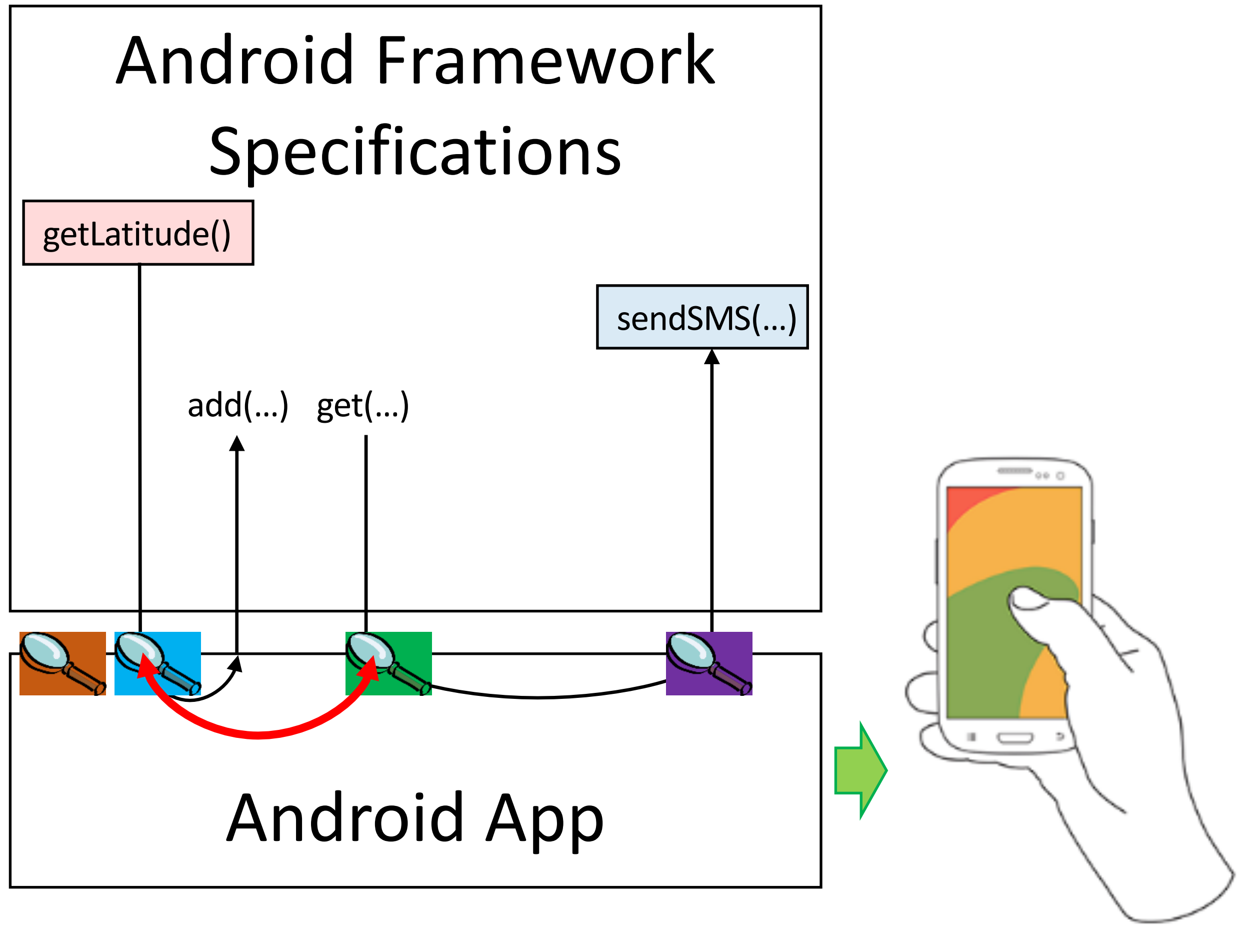
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step 2: Monitor for Counter-Examples

```
0xAAAAA 🟦 Double latitude = getLatitude();
0x00000 🟧 List list = new List();
           list.add(latitude);
0xAAAAA 🟩 Double data = list.get(0);
        🟪 Double dataDup = data;
           sendSMS(dataDup);
```

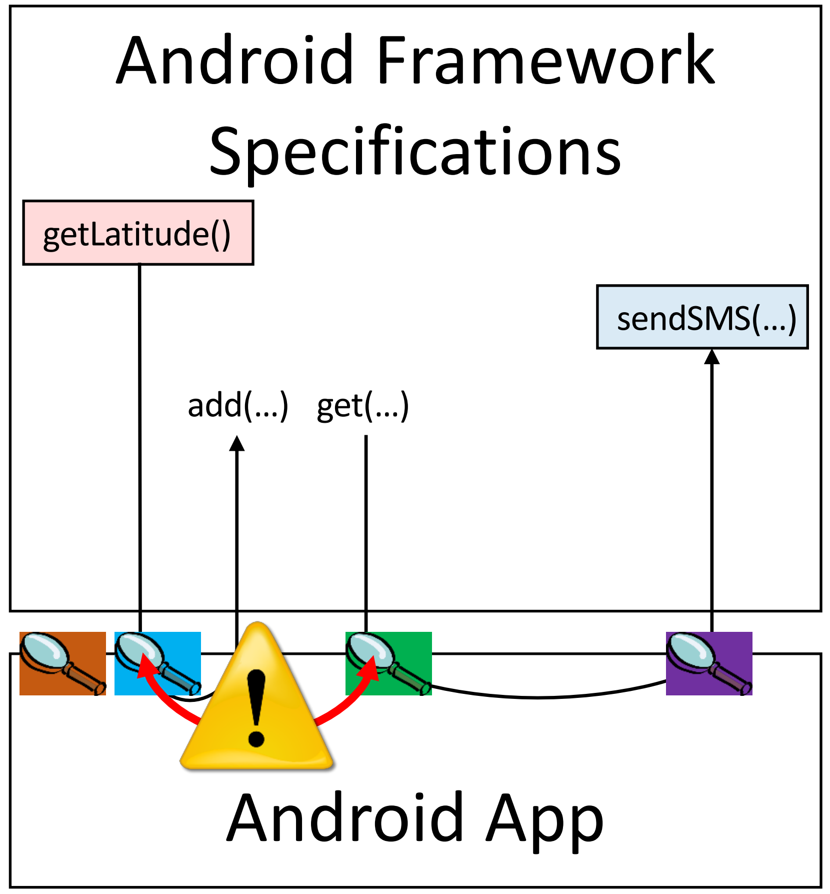
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```

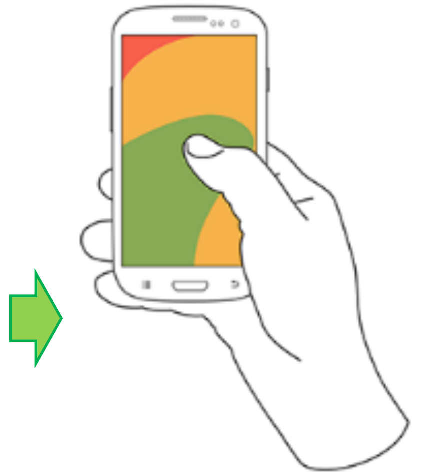


Step 2: Monitor for Counter-Examples

```
0xAAAAA      Double latitude = getLatitude();
0x00000      List list = new List();
         3.  list.add(latitude);
0xAAAAA      Double data = list.get(0);
             Double dataDup = data;
         6.  sendSMS(dataDup);
```

```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```

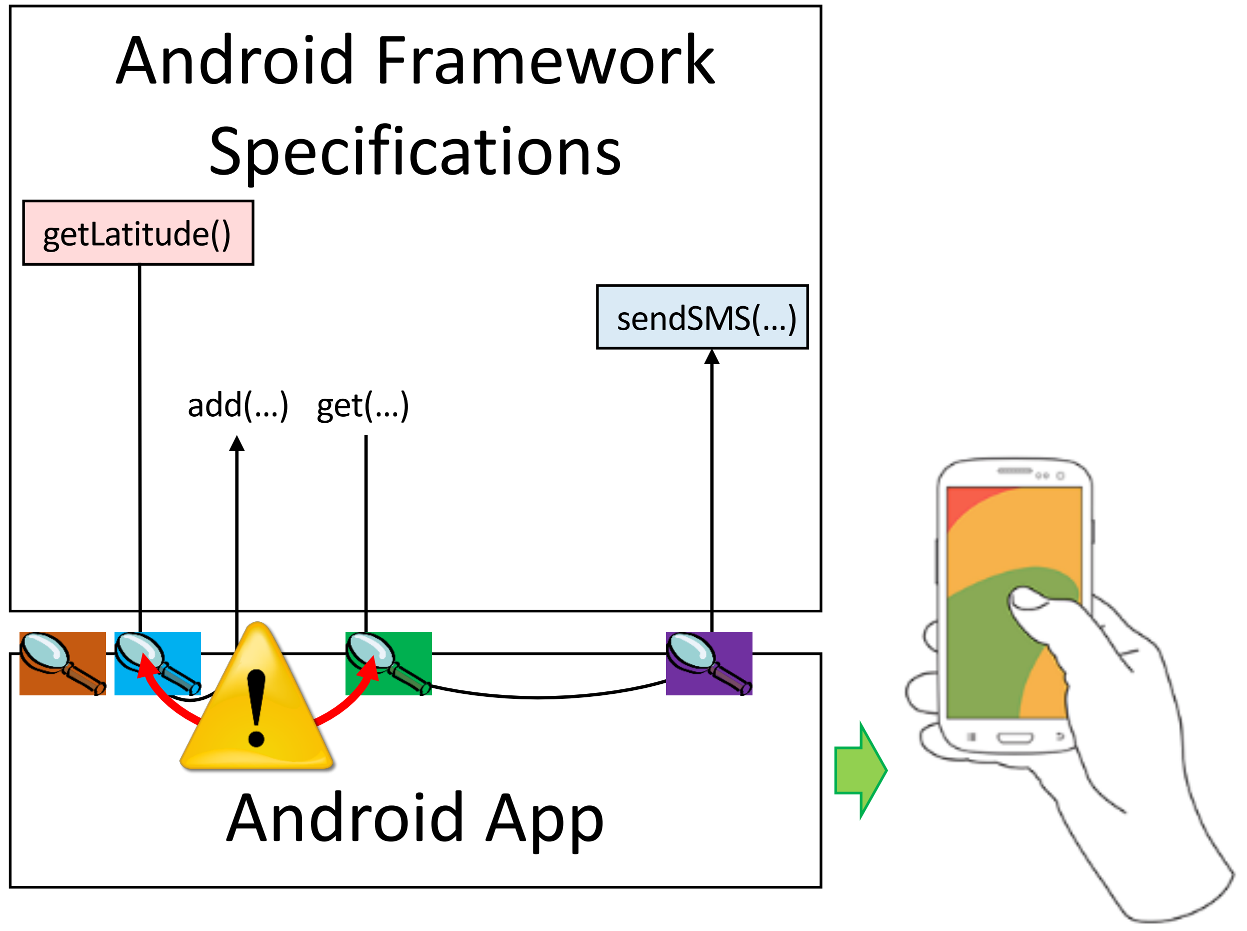




Step 3: Add Reported Counter-Examples

```
0xAAAAA    Double latitude = getLatitude();
0x00000    List list = new List();
           list.add(latitude);
0xAAAAA    Double data = list.get(0);
           Double dataDup = data;
           sendSMS(dataDup);
```

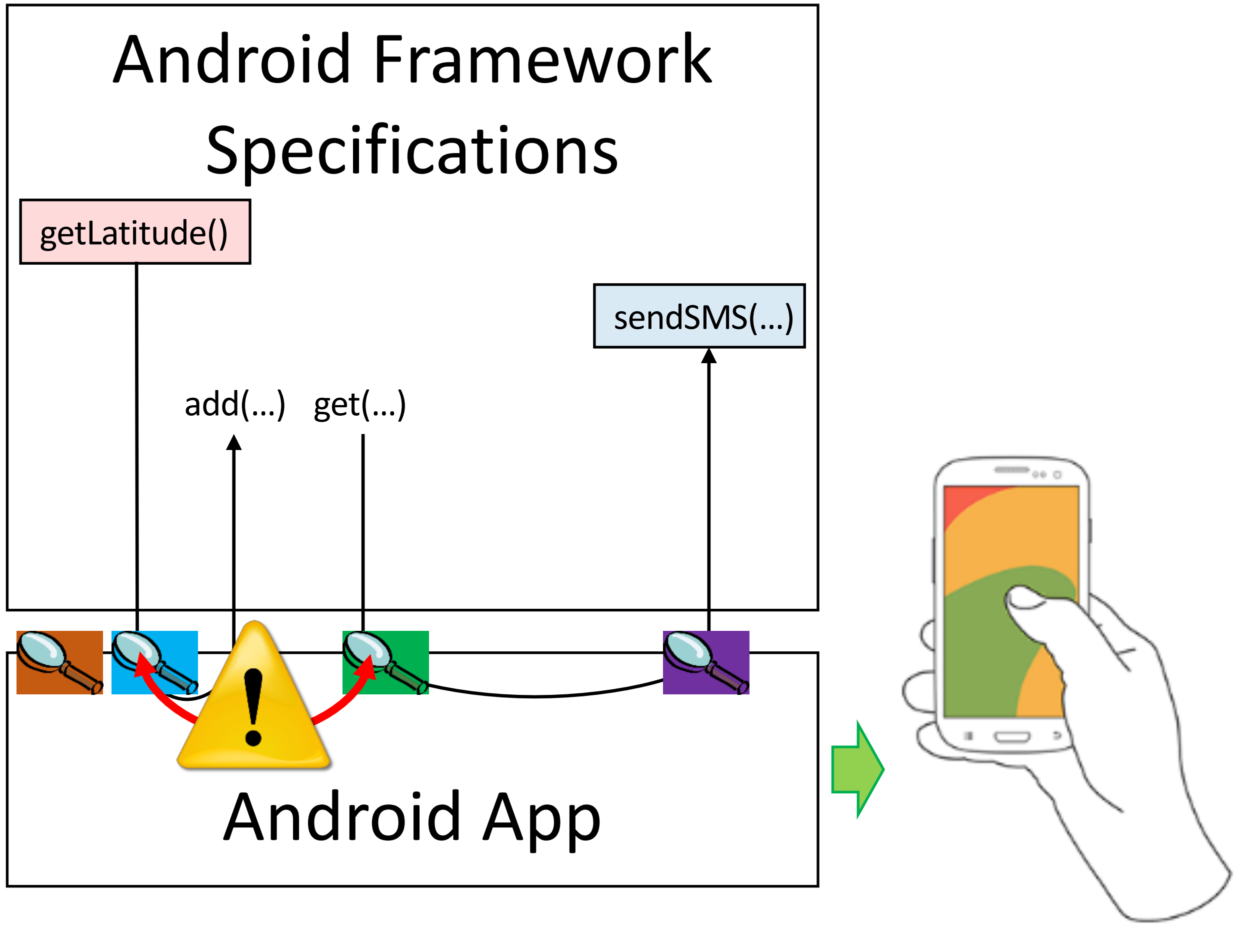
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step 3: Add Reported Counter-Examples

```
              @Alias(latitude, data)
0xAAAAA       Double latitude = getLatitude();
0x00000       List list = new List();
        3.    list.add(latitude);
0xAAAAA       Double data = list.get(0);
              Double dataDup = data;
        6.    sendSMS(dataDup);
```

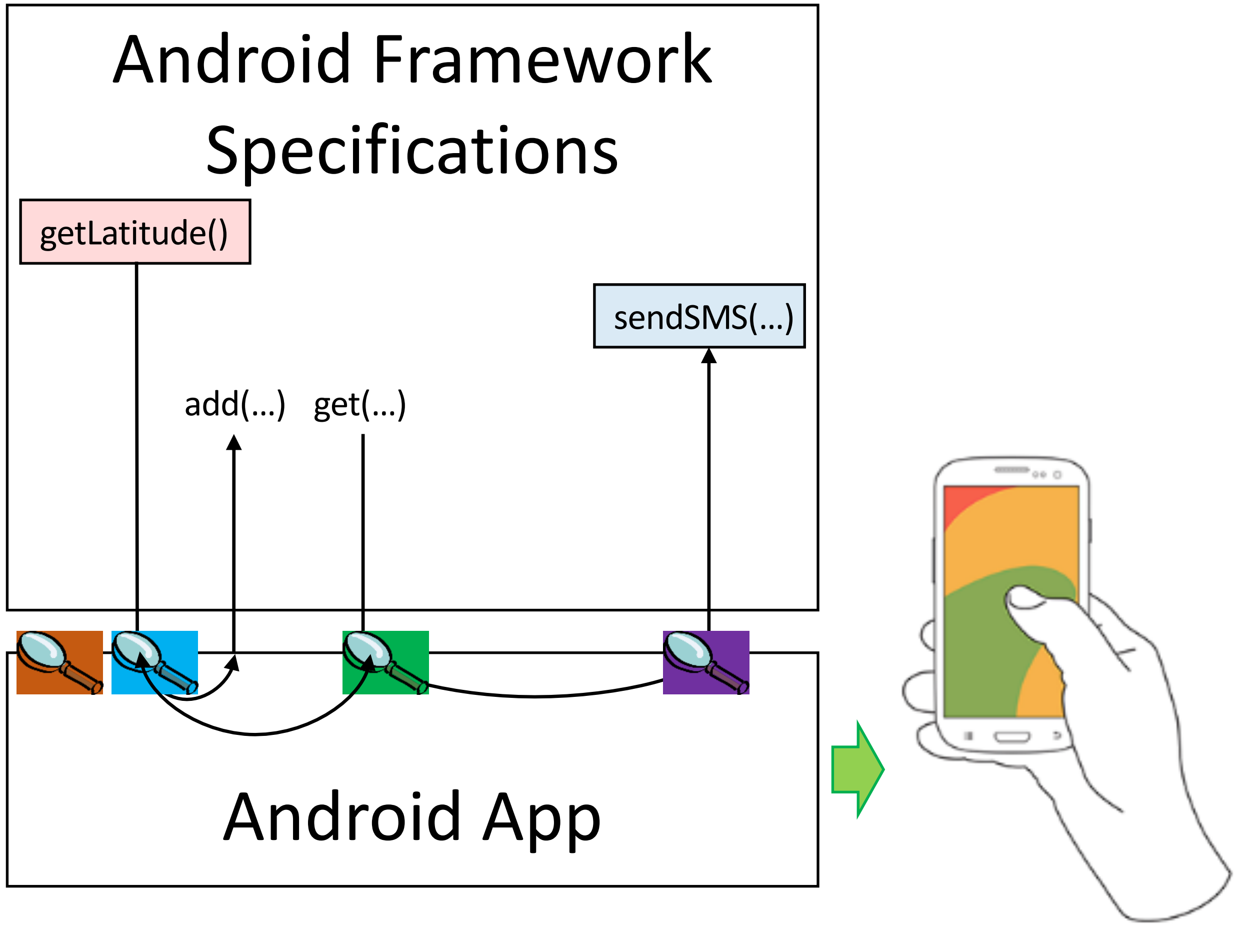
```
5.   class LocationManager:
6.       @Flow(LOC, return)
7.       static String getLatitude() {}
8.   class SMS:
9.       @Flow(text, SMS)
10.      static void sendSMS(String text) {}
```



Step 3: Add Reported Counter-Examples

```
@Alias(latitude, data)
0xAAAAA      Double latitude = getLatitude();
0x00000      List list = new List();
          3. list.add(latitude);
0xAAAAA      Double data = list.get(0);
             Double dataDup = data;
          6. sendSMS(dataDup);
```

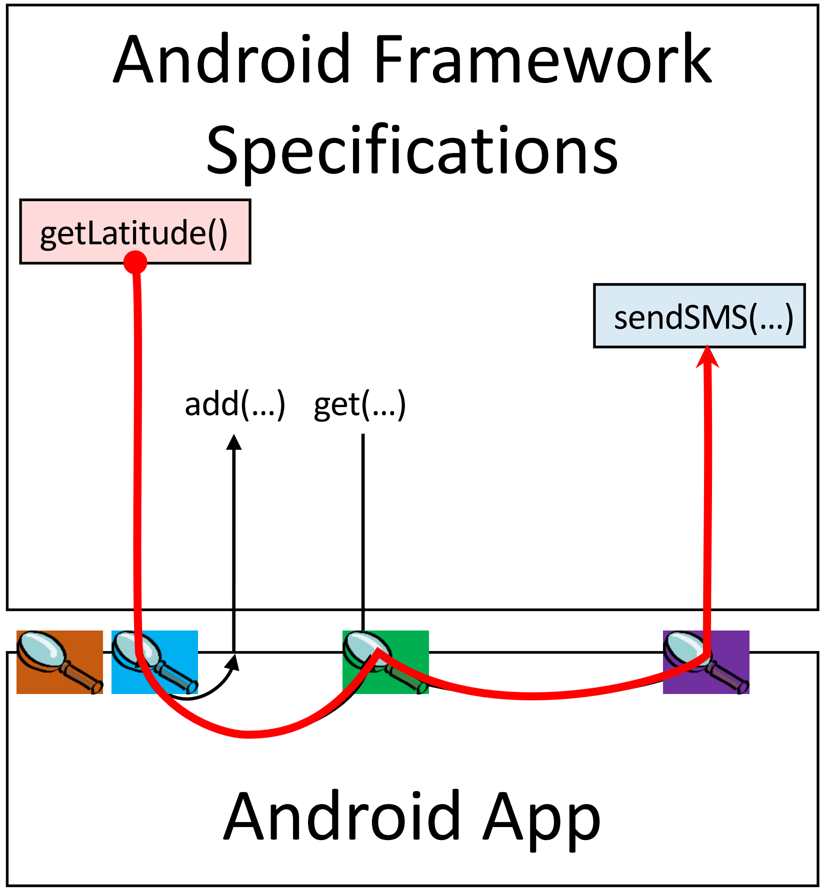
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```

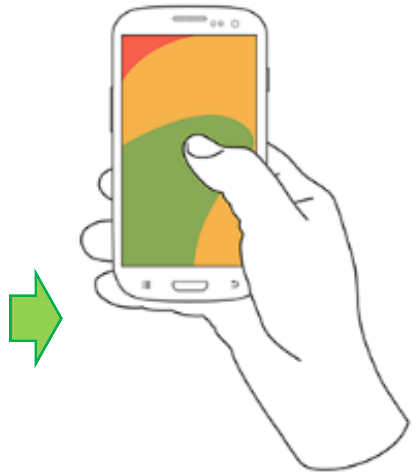


Step 3: Add Reported Counter-Examples

```
@Alias(latitude, data)

0xAAAAA     Double latitude = getLatitude();
0x00000     List list = new List();
         3. list.add(latitude);
0xAAAAA     Double data = list.get(0);
            Double dataDup = data;
         6. sendSMS(dataDup);
```

```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```





Naïve Monitoring

- **Monitor:**
  - Every allocation, assignment, and field load/store

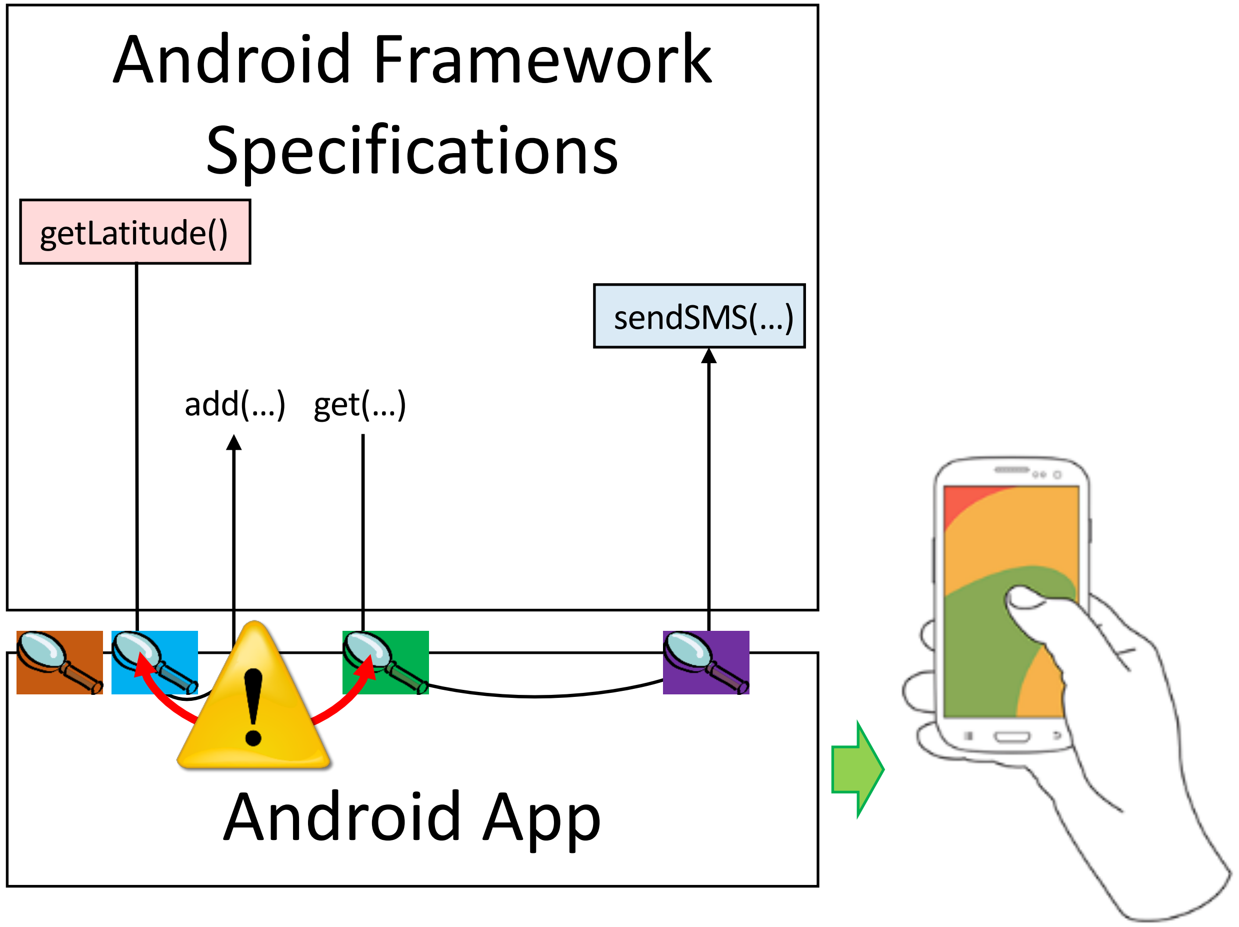
- **Theorem (easy):**
  - Instrumentation scheme is dynamically sound

Optimized Monitoring

Optimized Monitoring

```
0xAAAAA  🔍 Double latitude = getLatitude();
0x00000  🔍 List list = new List();
         3. list.add(latitude);
0xAAAAA  🔍 Double data = list.get(0);
         🔍 Double dataDup = data;
         6. sendSMS(dataDup);
```

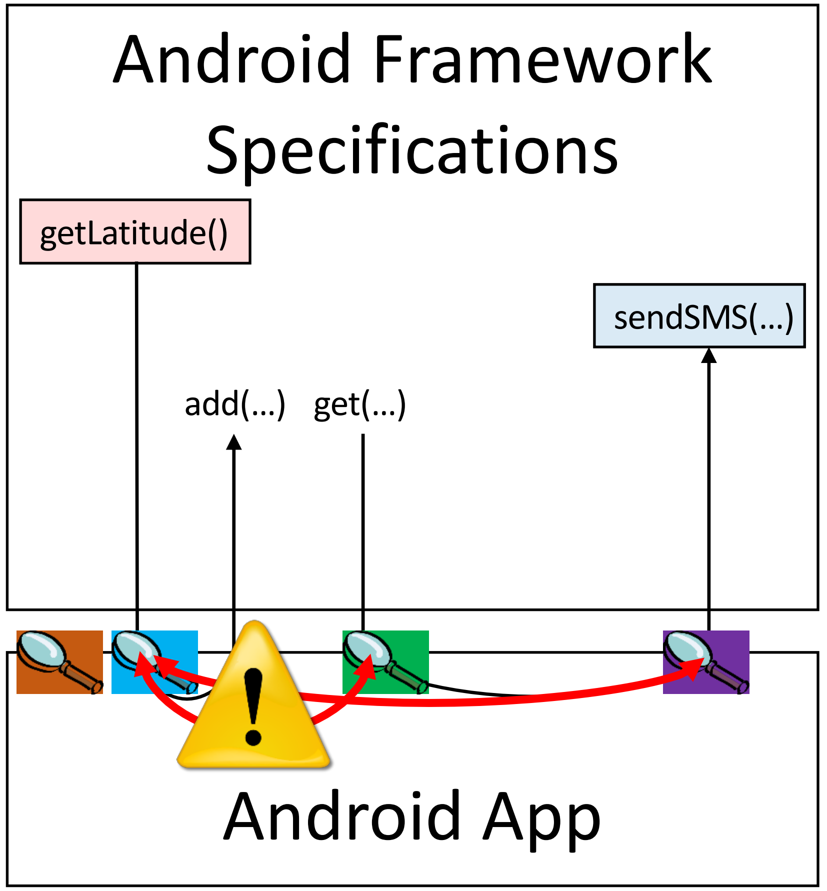
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```

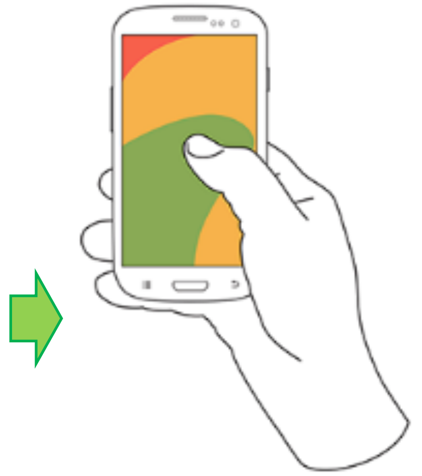


Optimized Monitoring

```
0xAAAAA  Double latitude = getLatitude();
0x00000  List list = new List();
         list.add(latitude);
0xAAAAA  Double data = list.get(0);
0xAAAAA  Double dataDup = data;
         sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```

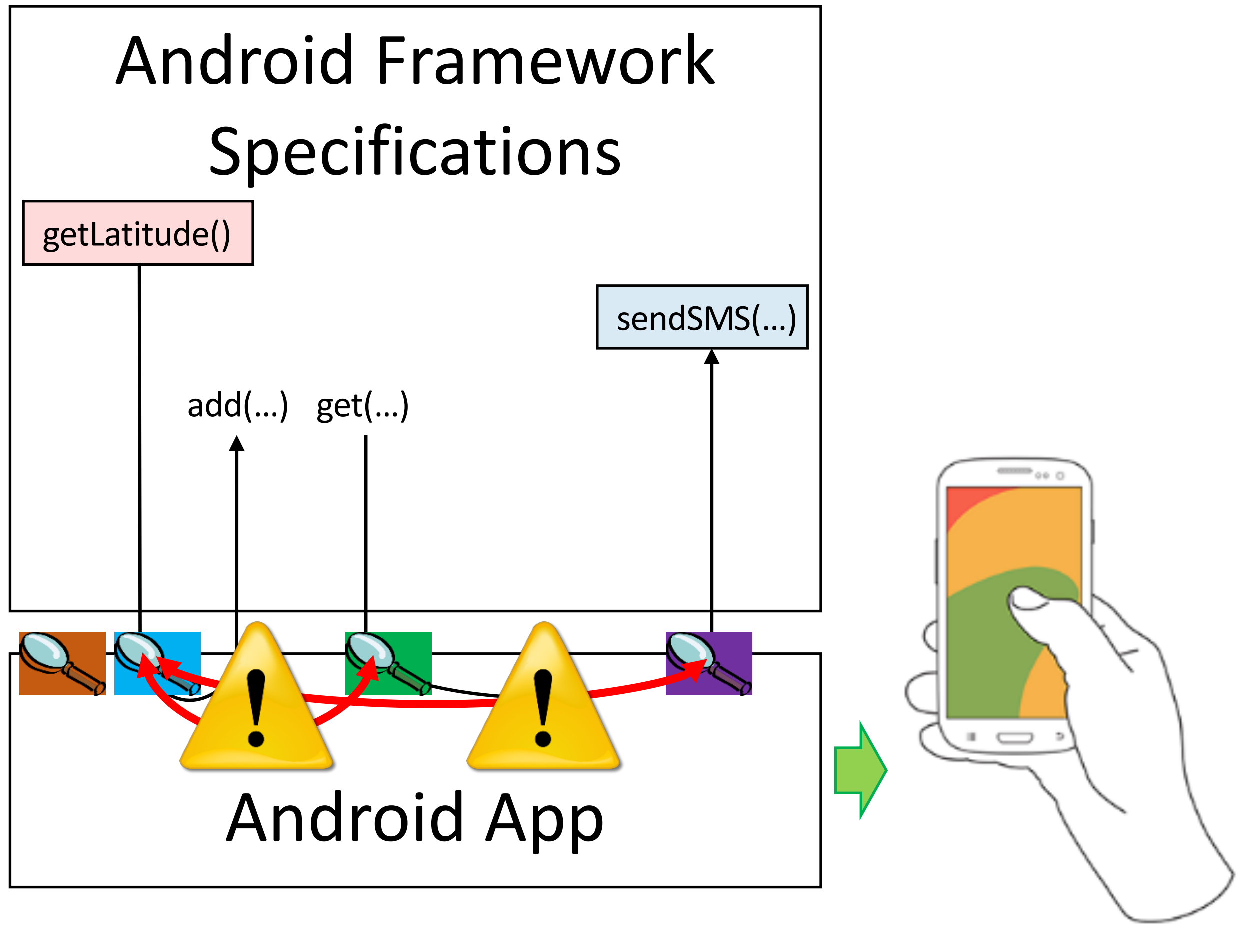




Optimized Monitoring

```
0xAAAAA   Double latitude = getLatitude();
0x00000   List list = new List();
          list.add(latitude);
0xAAAAA   Double data = list.get(0);
0xAAAAA   Double dataDup = data;
          sendSMS(dataDup);
```

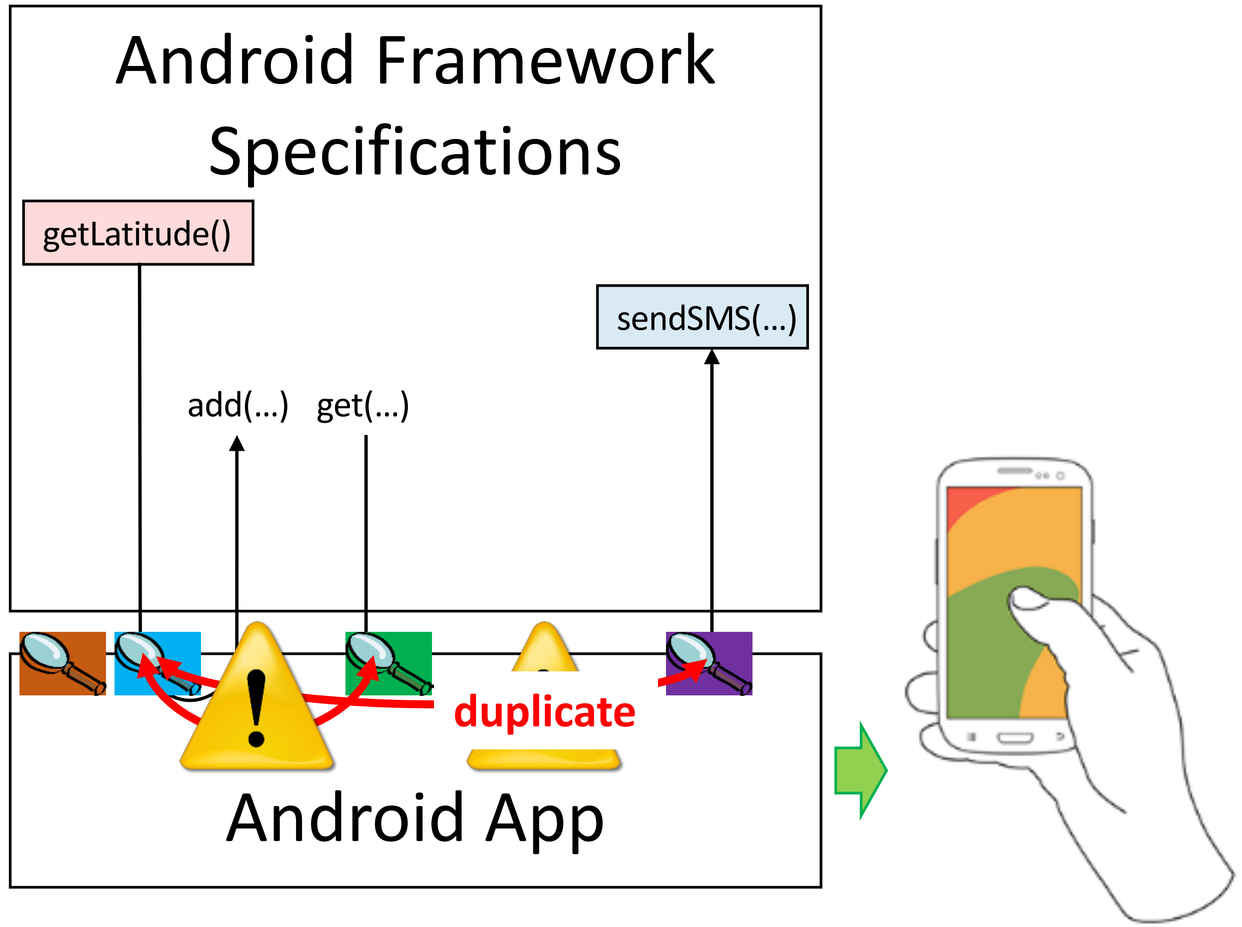
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Optimized Monitoring

```
0xAAAAA   Double latitude = getLatitude();
0x00000   List list = new List();
          list.add(latitude);
0xAAAAA   Double data = list.get(0);
0xAAAAA   Double dataDup = data;
          sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Optimized Monitoring

- **Assumption:** The framework does not access app fields
  - Holds in practice

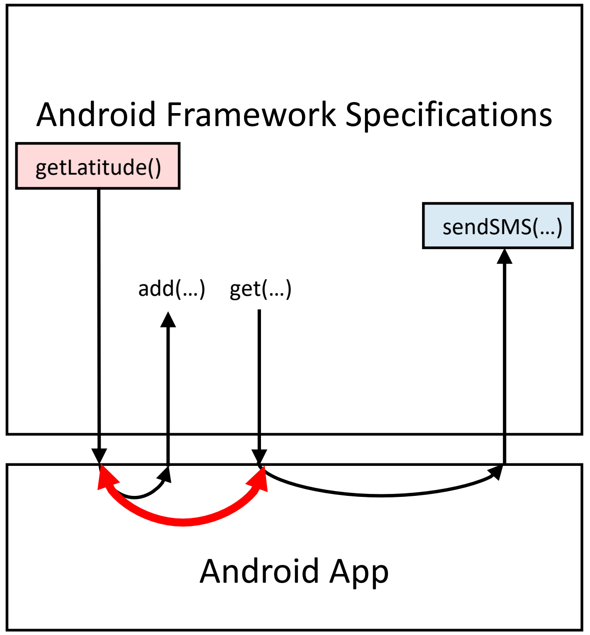
- **Monitor:**
  - Allocations $x \leftarrow \text{new}\, X()$ that may leak to the framework
  - Return values $x \leftarrow m(y)$ of framework methods
  - App accesses $x \leftarrow y.f$ to framework fields

- **Theorem:**
  - Instrumentation scheme is dynamically sound

Beyond a Single App

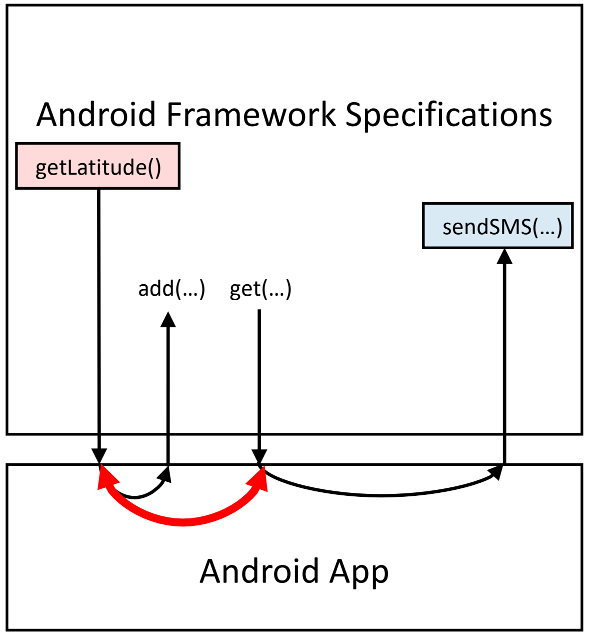
Beyond a Single App

- Counter-examples are used to improve results for a **single** app
  - Alias(latitude, data)

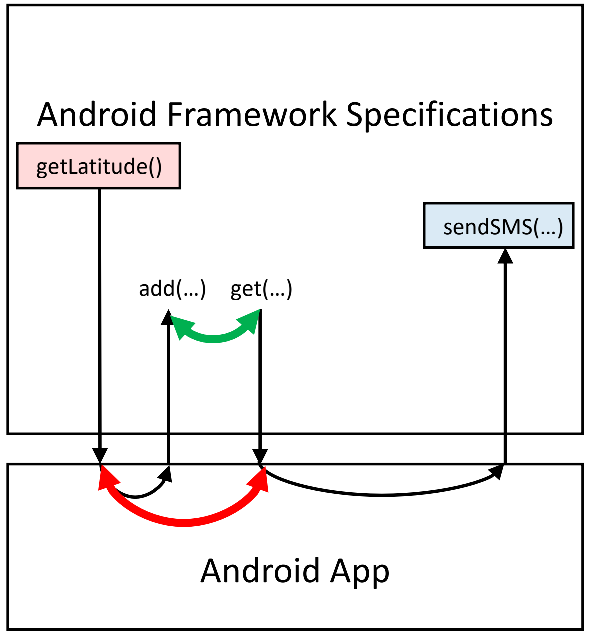


Beyond a Single App

- Counter-examples are used to improve results for a **single** app
  - Alias(latitude, data)



- Infer specifications for the **framework** and use across apps
  - Alias(add.arg, get.return)
    ⇒ Alias(latitude, data)



Summary

- Eventually sound points-to analysis
  - **Step 1:** Optimistic static analysis
  - **Step 2:** Instrument app to monitor for counter-examples
  - **Step 3:** Update static analysis to address detected counter-examples

Evaluation

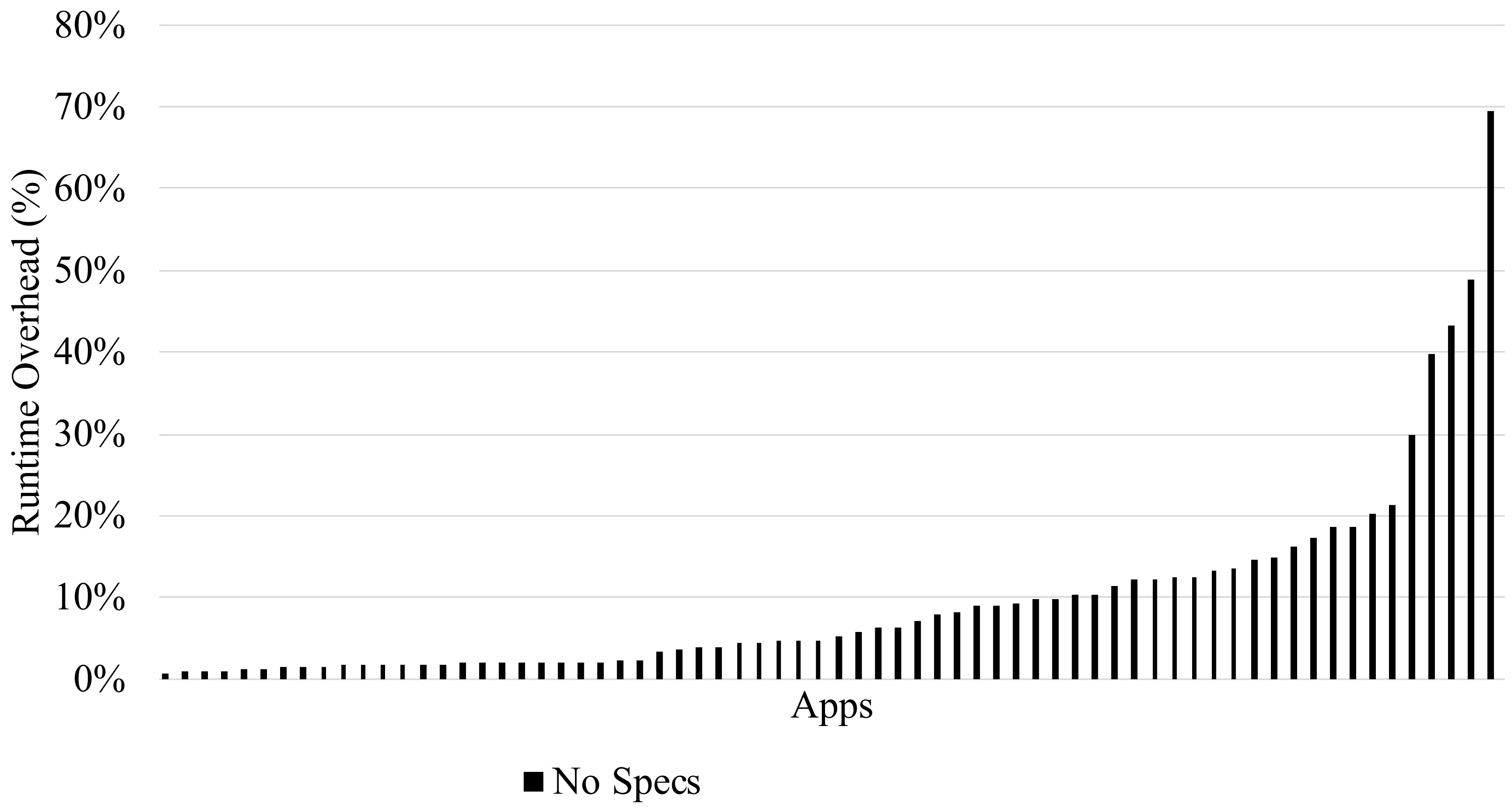
Evaluation

- Instrumented 72 Android apps

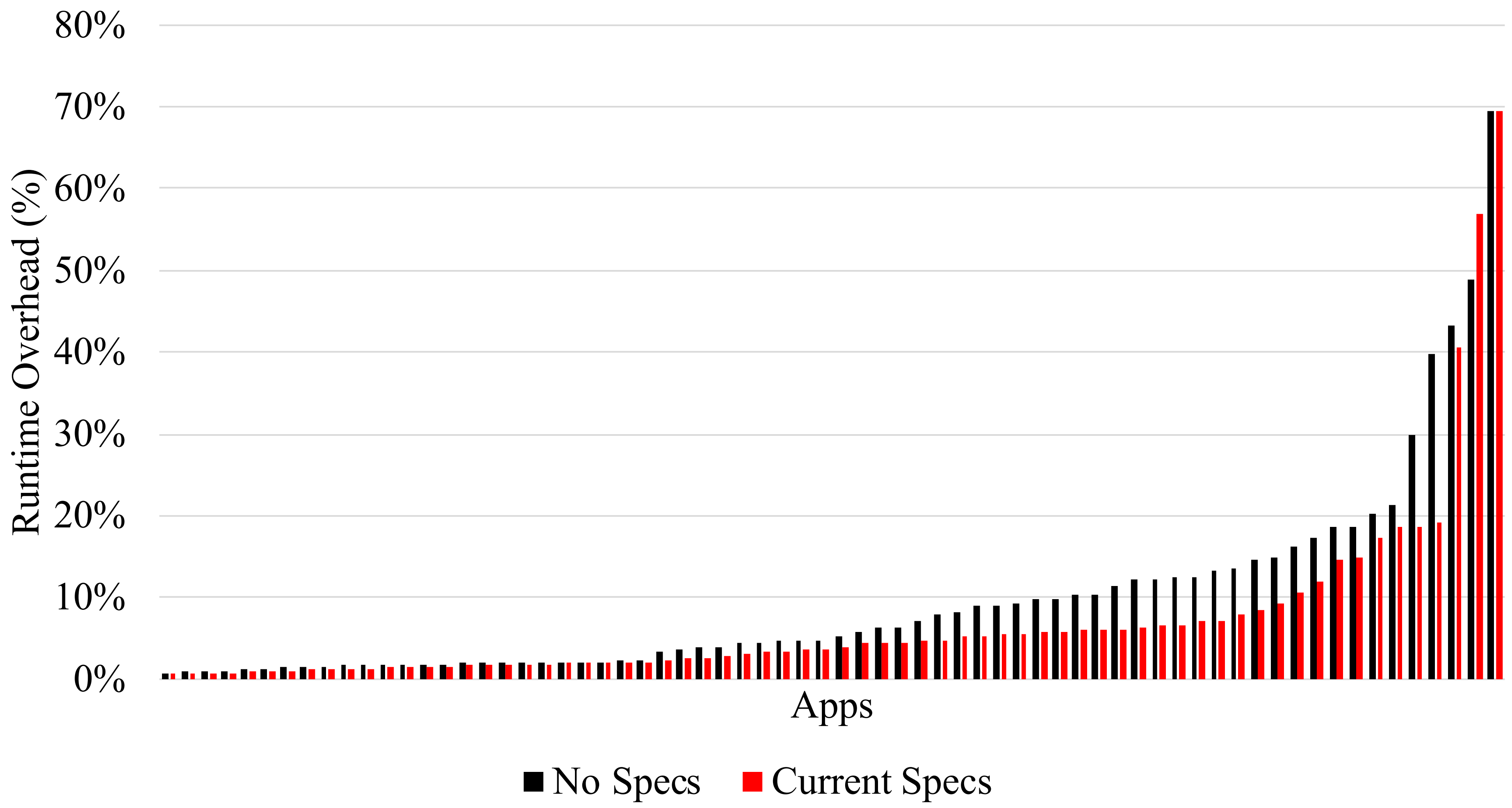
- Executed apps
  - In Android emulator
  - 1 hour each
  - Used Monkey to generate random events

Overhead (Running Time)

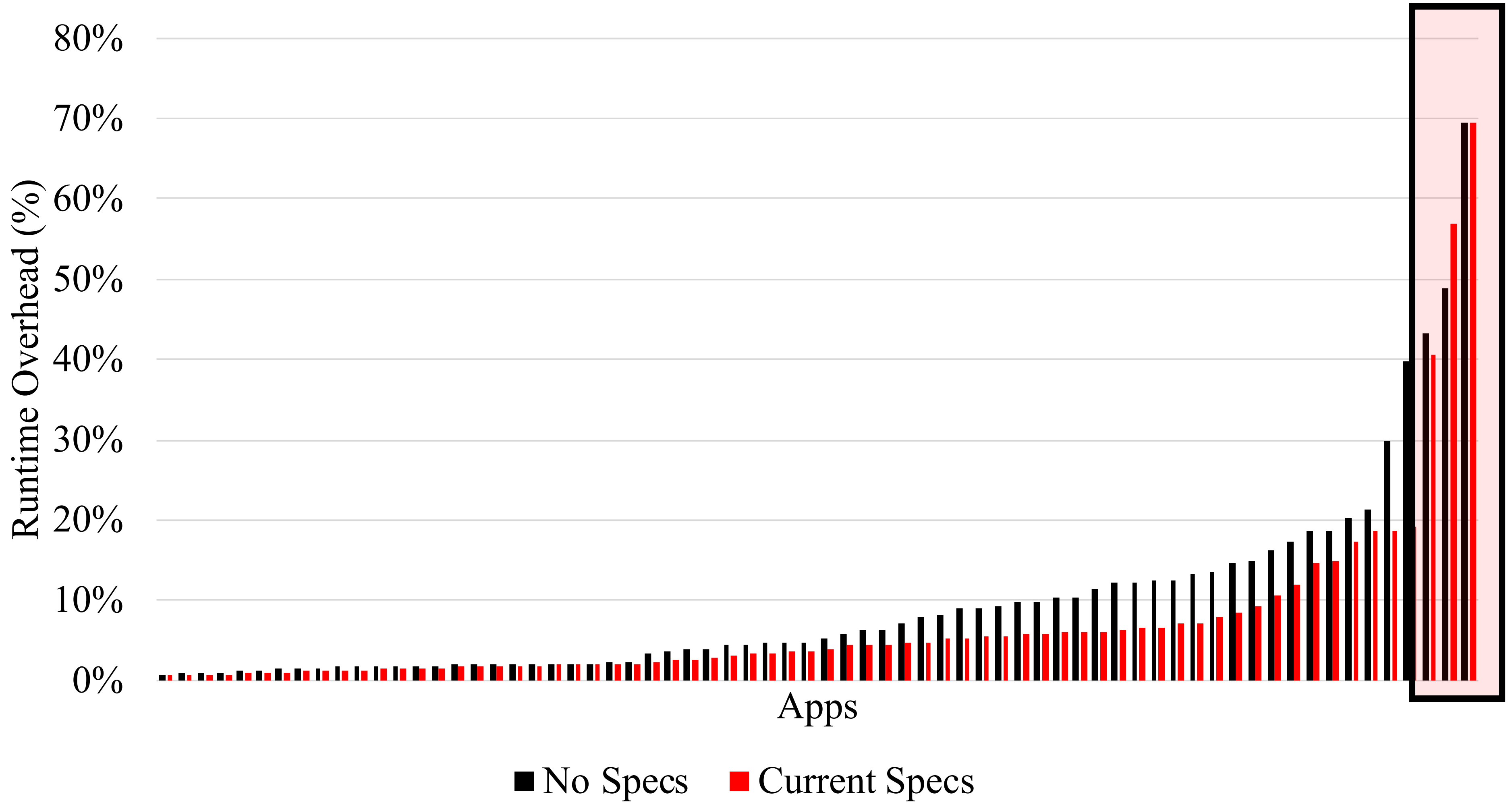
Overhead (Running Time)



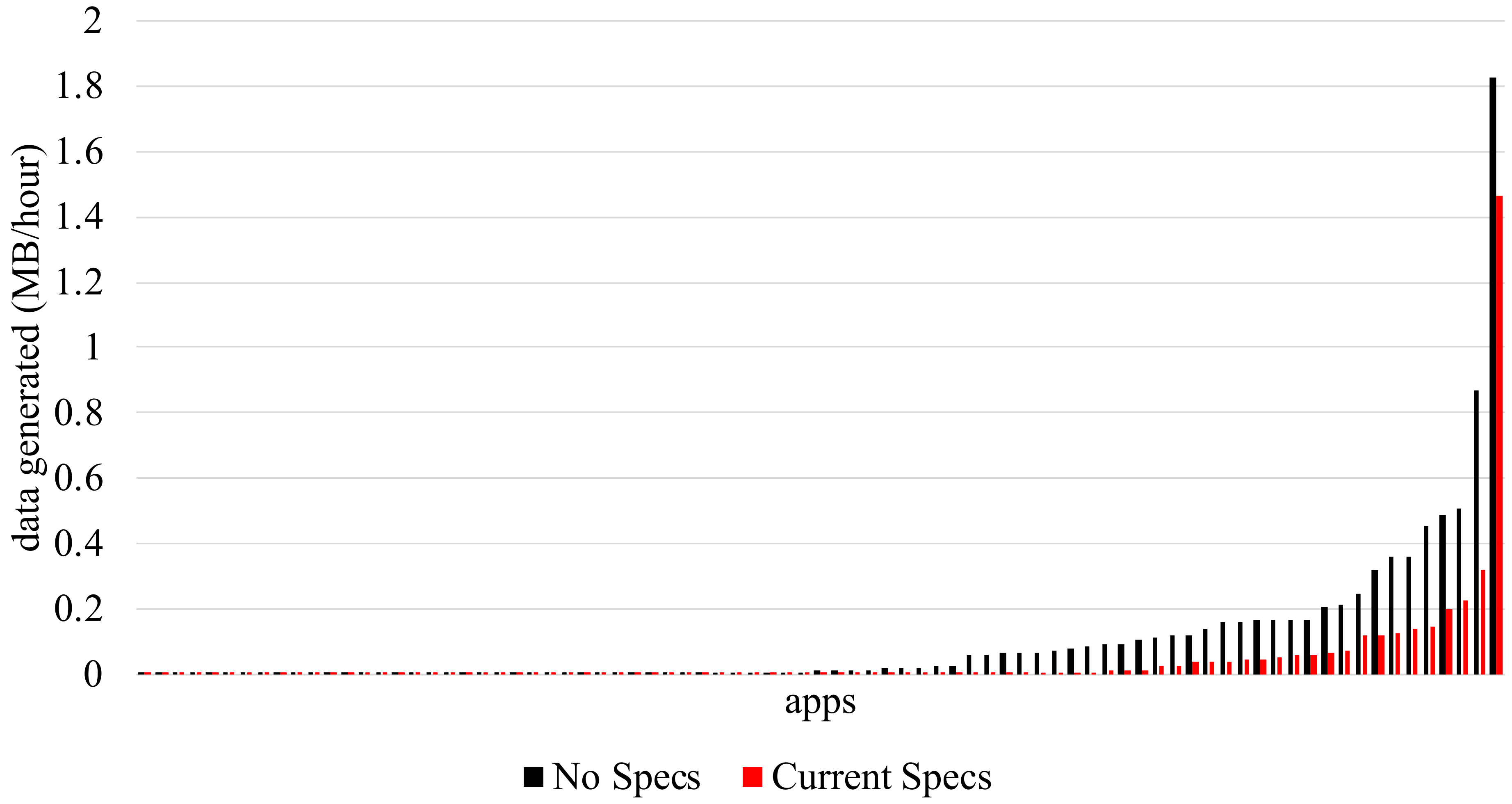
Overhead (Running Time)



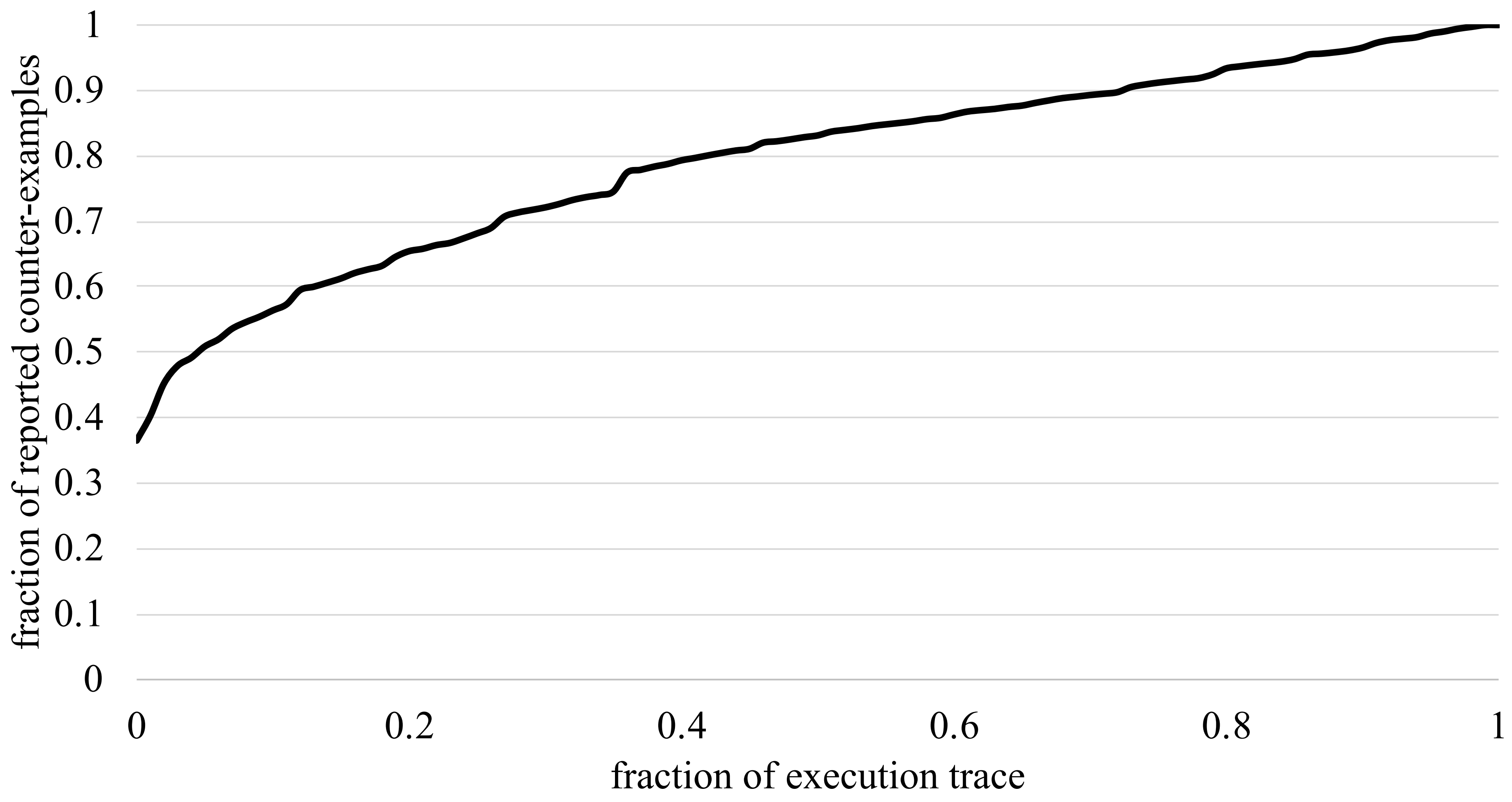
Overhead (Running Time)



Overhead (Data Usage)



Reported Counter-Examples



Conclusions

- **Hard-to-Analyze Code**
  - Ubiquitous
  - Use specifications

- **Our approach:** Eventual soundness

Questions?

Backup Slides

Related Work

- Static analysis for points-to analysis, finding Android malware

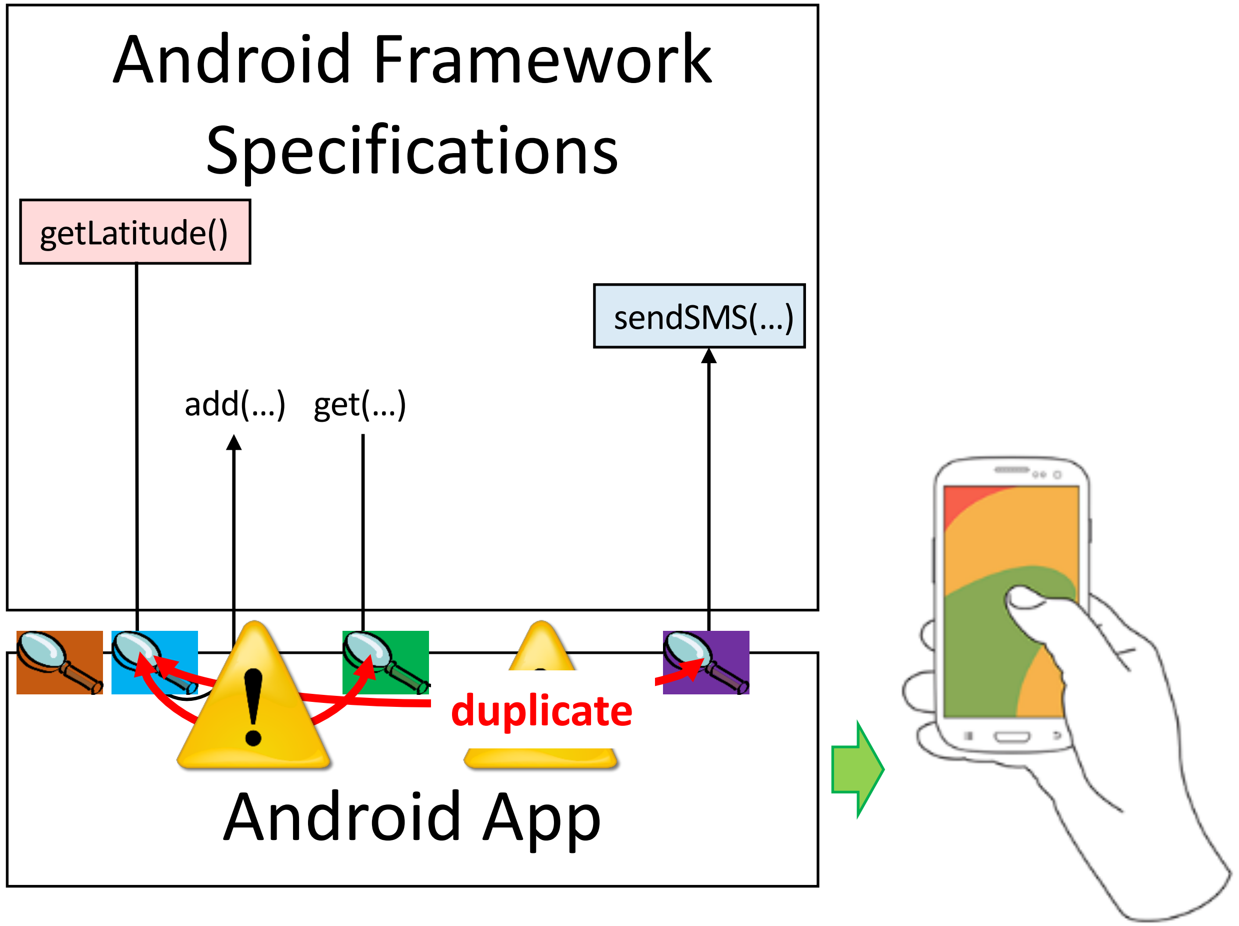
- Monitoring executions
  - Dynamic policy enforcement (Enck 2010)
  - Debugging (Liblit 2005, Jin 2012)

- Inferring specifications from executions
  - Inference of **taint** specifications (Clapp 2015)
  - Inference of Javascript library functions (Heule 2015)

Optimized Monitoring

```
0xAAAAA    Double latitude = getLatitude();
0x00000    List list = new List();
        3. list.add(latitude);
0xAAAAA    Double data = list.get(0);
0xAAAAA    Double dataDup = data;
        6. sendSMS(dataDup);
```

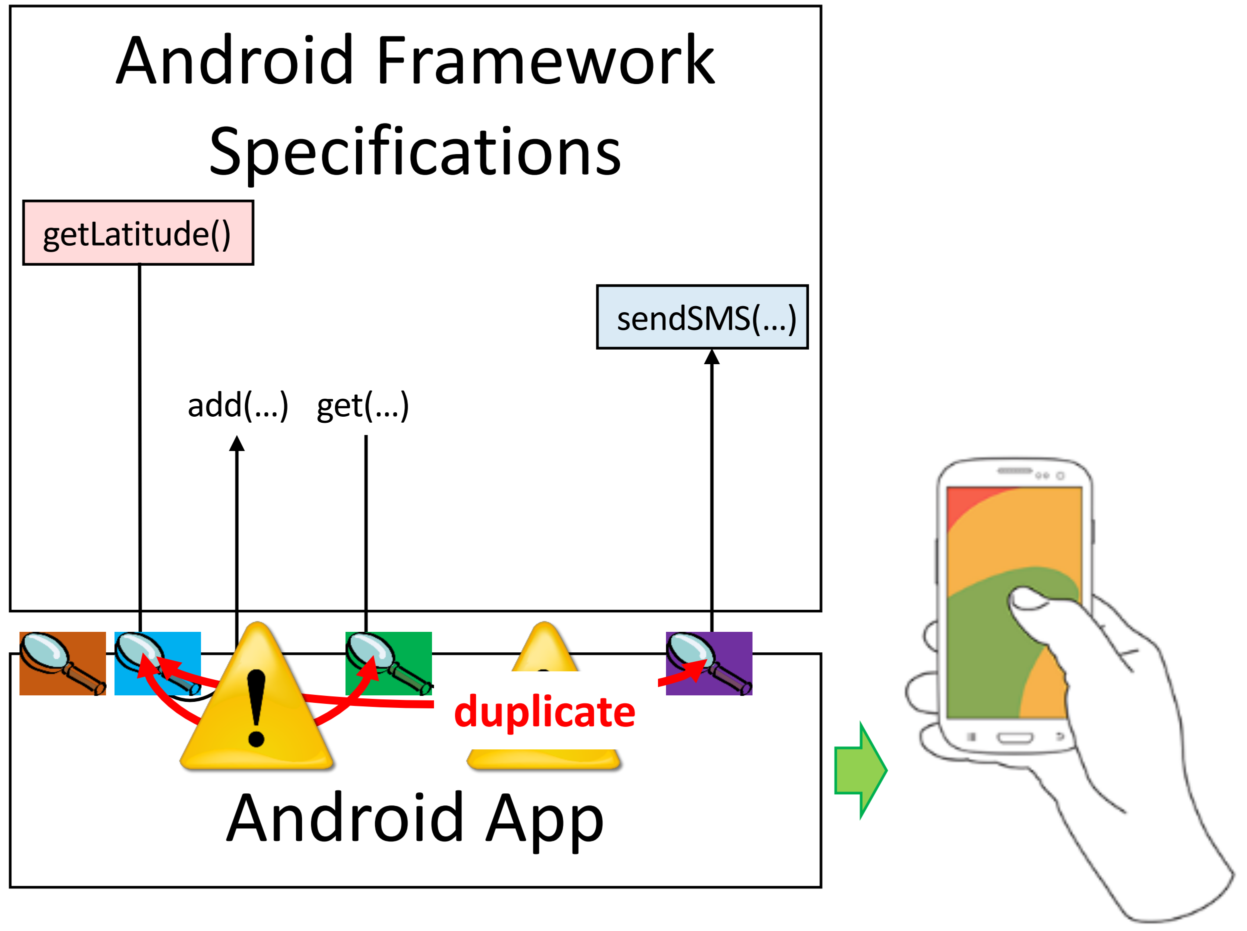
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```



Optimized Monitoring

```
0xAAAAA   Double latitude = getLatitude();
0x00000   List list = new List();
          list.add(latitude);
0xAAAAA   Double data = list.get(0);
0xAAAAA   Double dataDup = data;
          sendSMS(dataDup);
```

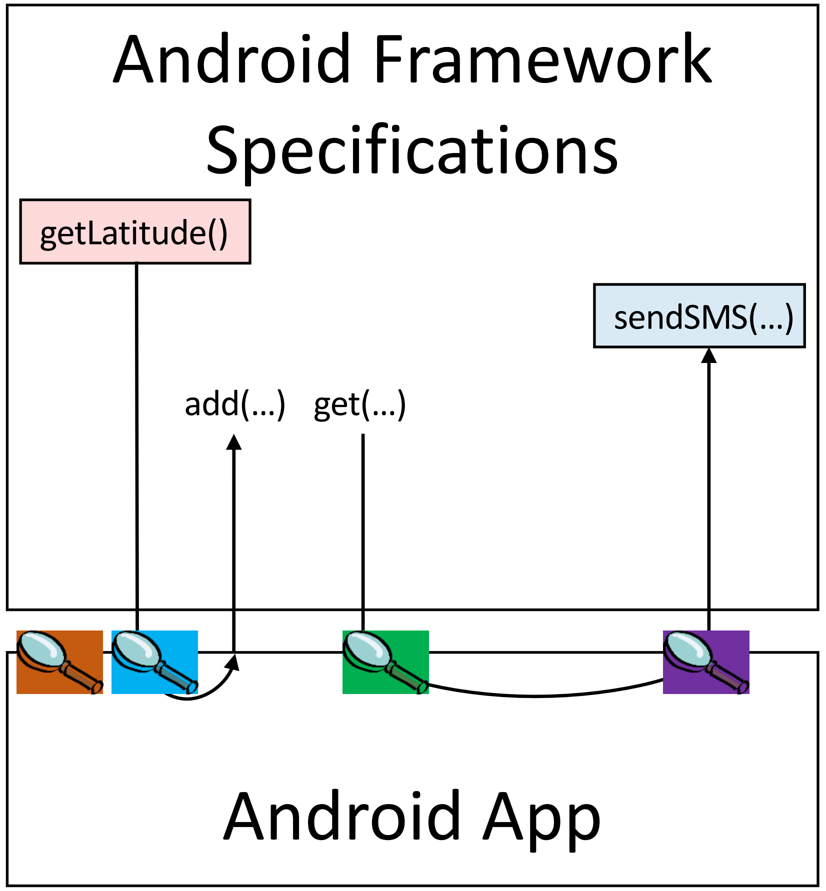
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Optimized Monitoring

```
method call 🔍 Double latitude = getLatitude();
 allocation 🔍 List list = new List();
           3.  list.add(latitude);
method call 🔍 Double data = list.get(0);
            🔍 Double dataDup = data;
           6.  sendSMS(dataDup);
```

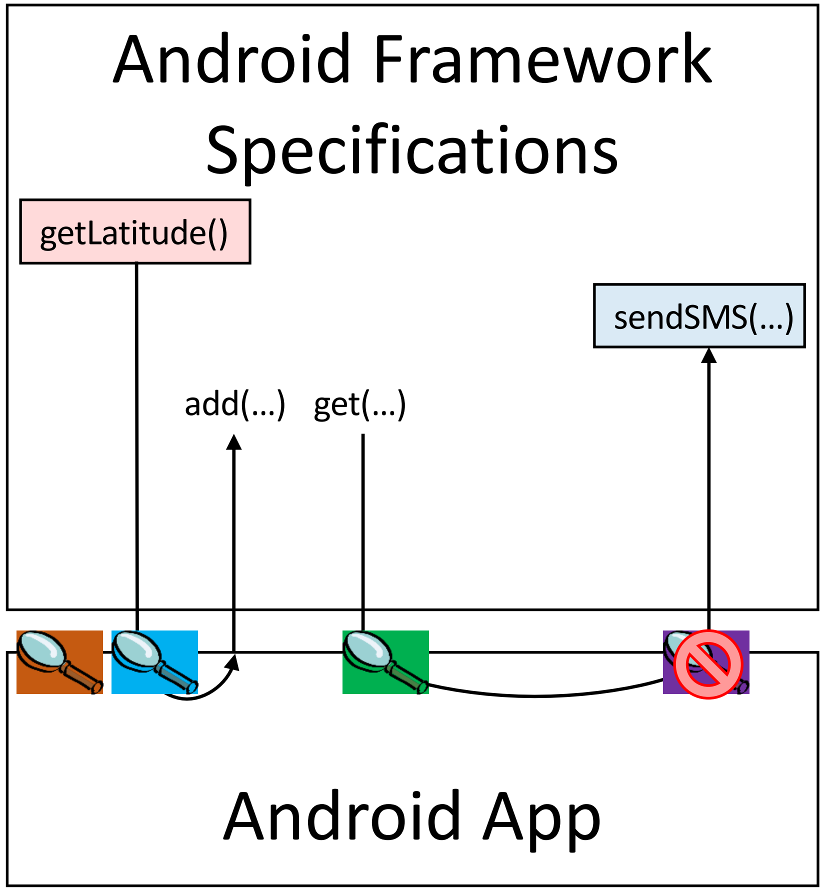
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```



Optimized Monitoring

```
method call     Double latitude = getLatitude();
 allocation     List list = new List();
            3.  list.add(latitude);
method call     Double data = list.get(0);
                Double dataDup = data;
            6.  sendSMS(dataDup);
```

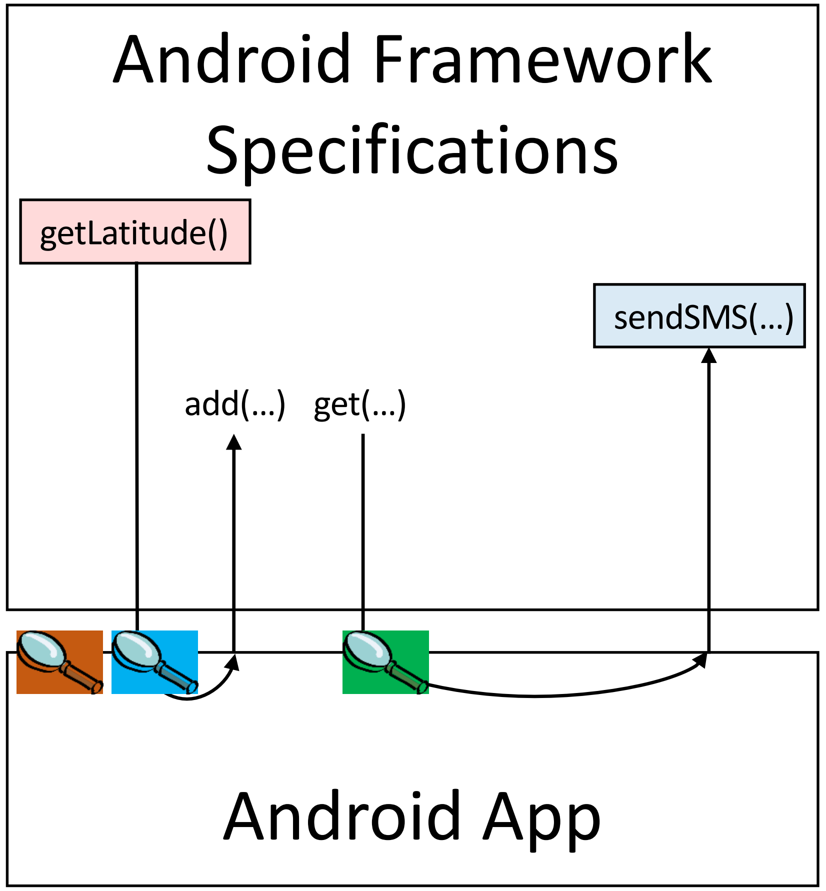
```
5.  class LocationManager:
6.      @Flow(LOC, return)
7.      static String getLatitude() {}
8.  class SMS:
9.      @Flow(text, SMS)
10.     static void sendSMS(String text) {}
```



Optimized Monitoring

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Framework Specification Inference

Framework Specification Inference

- CFL-reachability based specification inference algorithm (POPL 2015)

Framework Specification Inference

- CFL-reachability based specification inference algorithm (POPL 2015)
- **Step A:** Pessimistic static analysis

Framework Specification Inference

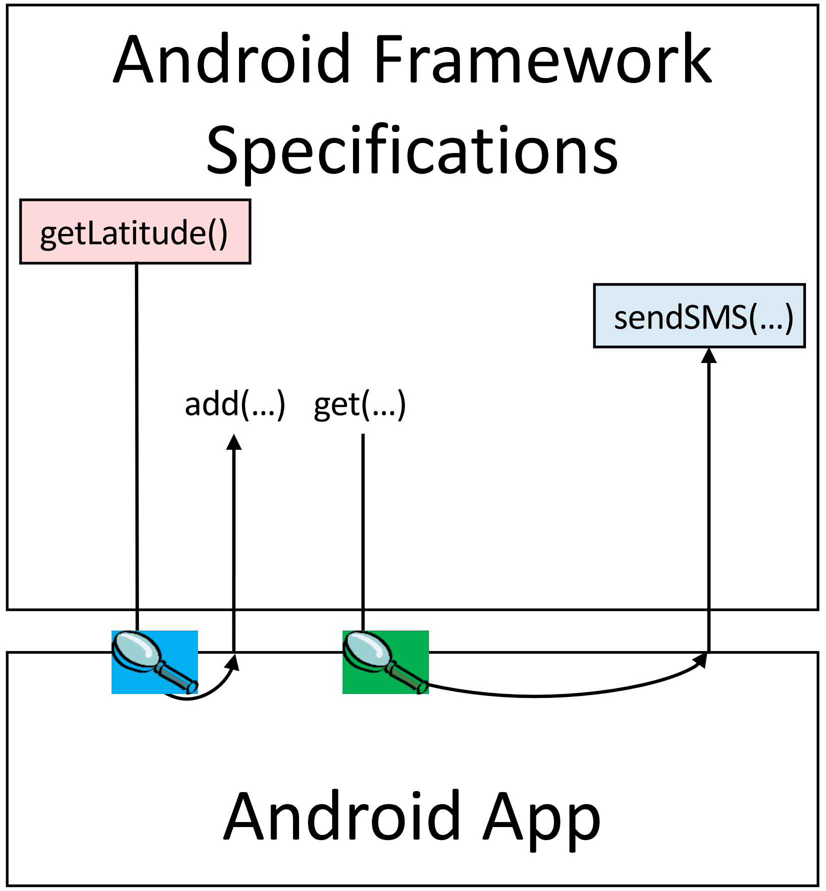
- CFL-reachability based specification inference algorithm (POPL 2015)

- **Step A:** Pessimistic static analysis
- **Step B:** Minimal assumption to derive missing aliasing
  - Fewer assumptions ⇒ more likely to be correct

Framework Specification Inference

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

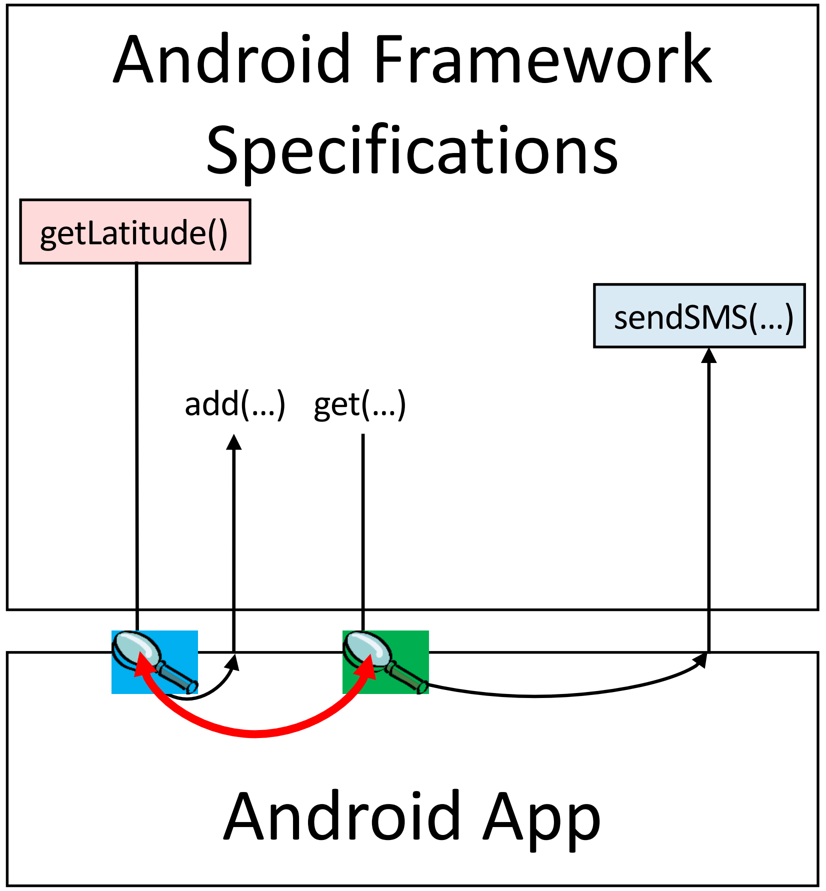
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Framework Specification Inference

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

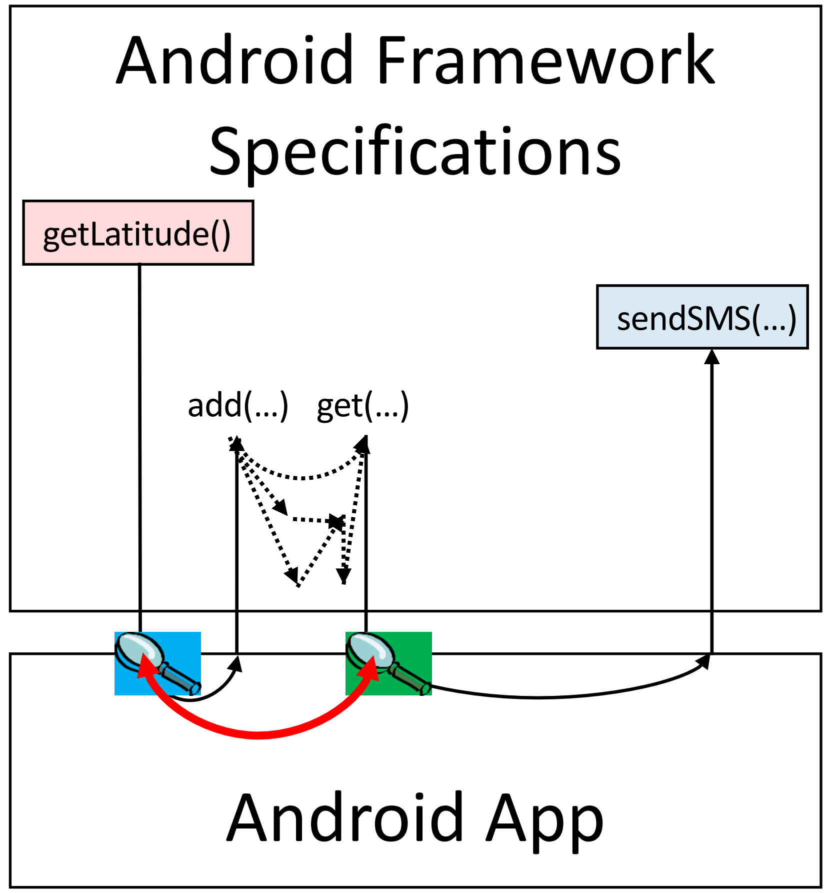
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step A: Pessimistic Assumptions

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

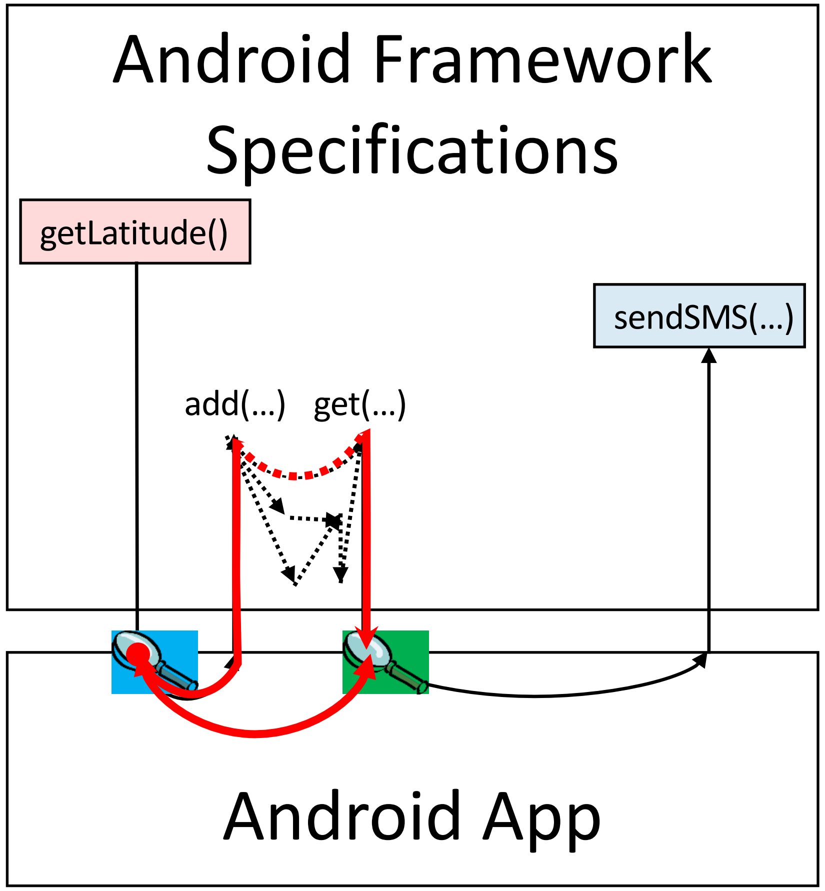
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Step B: Shortest Path

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

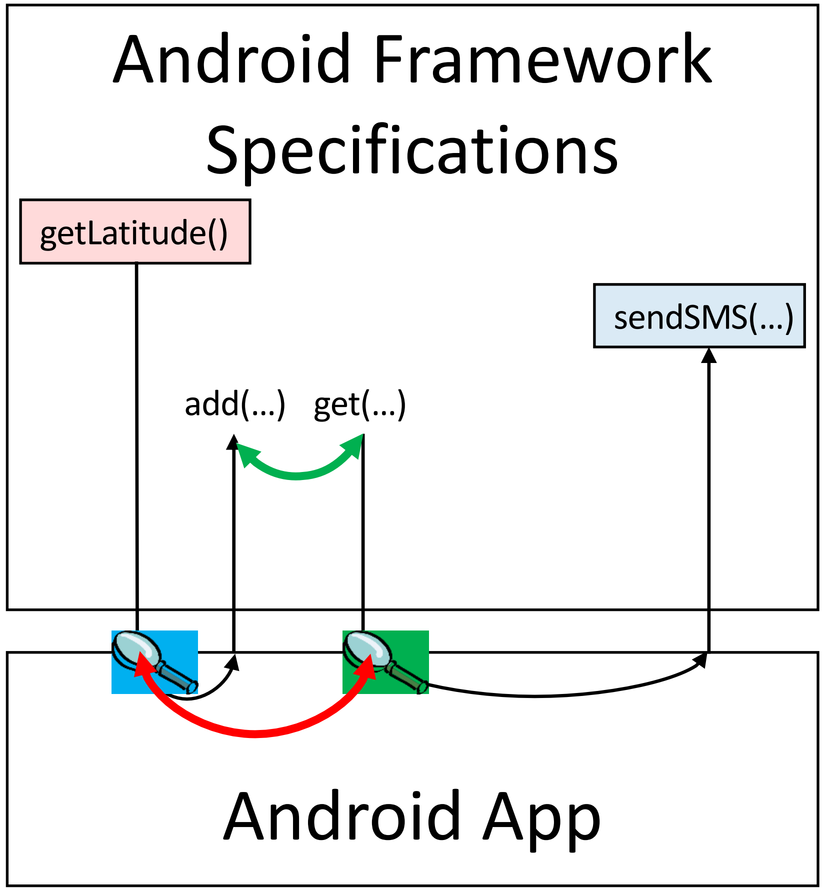
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



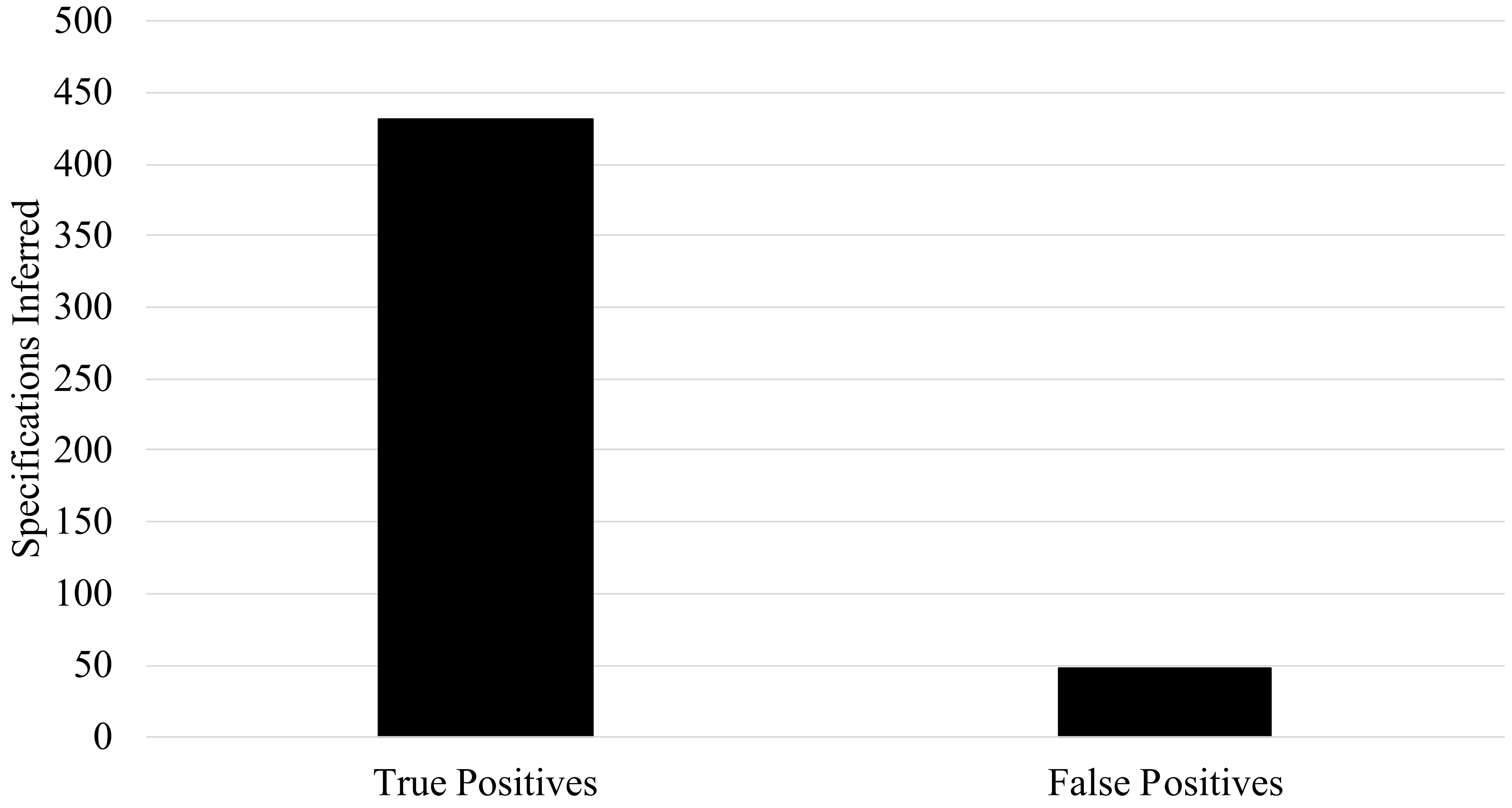
Step B: Shortest Path

```
Double latitude = getLatitude();
List list = new List();
list.add(latitude);
Double data = list.get(0);
Double dataDup = data;
sendSMS(dataDup);
```

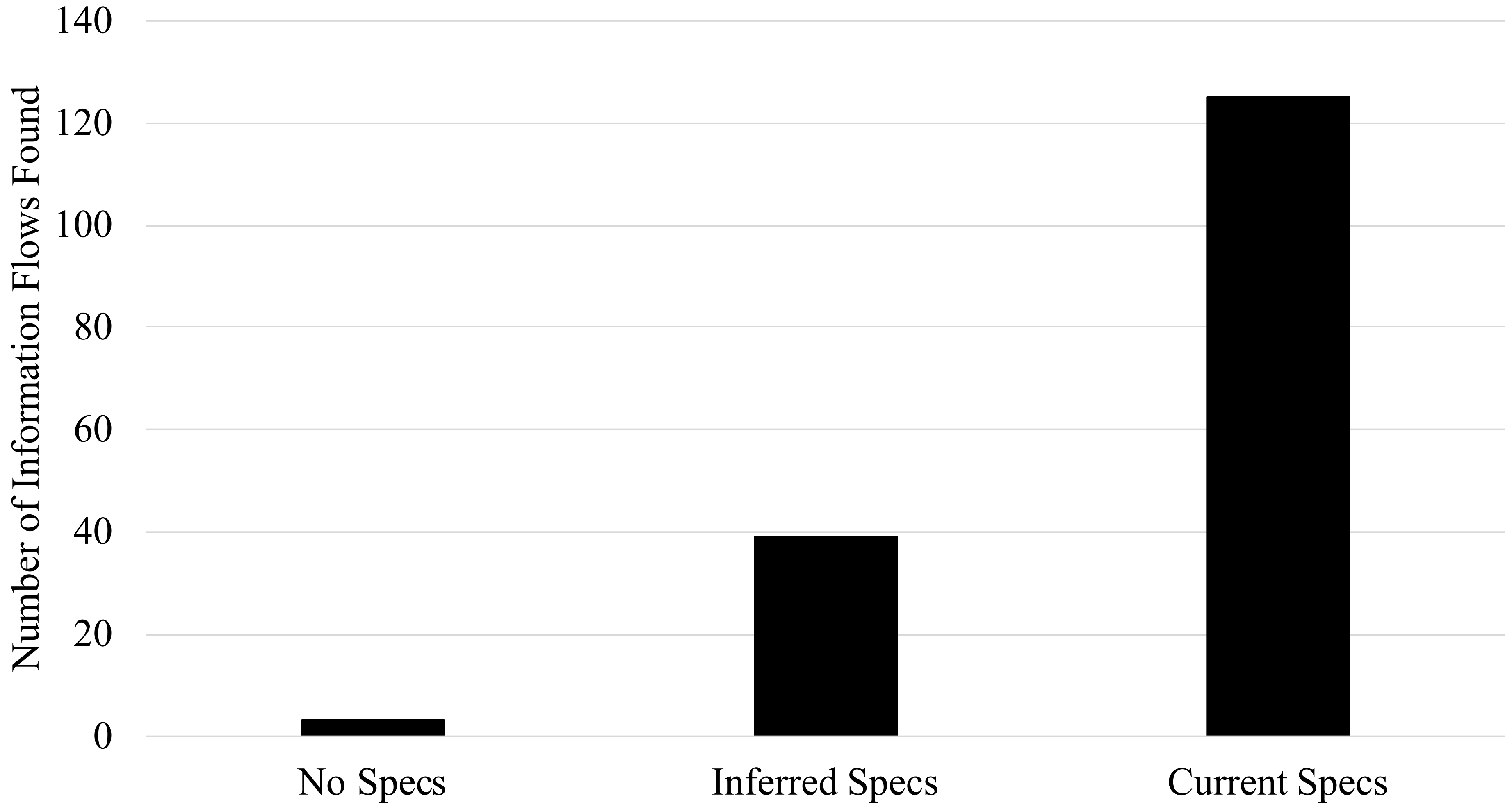
```
@Alias(add.arg, get.return)
class List:
    void add(Object arg) {}
    Object get(Integer index) {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



Specification Inference



Information Flows


Number of Information Flows Found
No Specs
Inferred Specs
Current Specs