# Specification Inference Using Context-Free Language Reachability

Osbert Bastani, Saswat Anand, and Alex Aiken

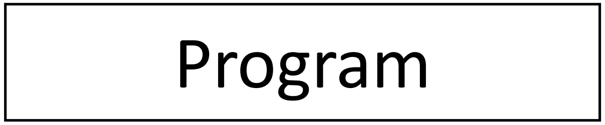
Stanford University

# Specification Inference Using Context-Free Language Reachability

# Specification Inference Using Context-Free Language Reachability

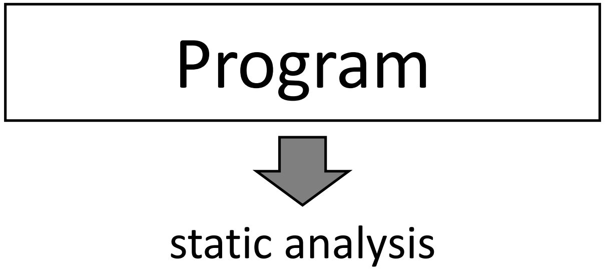
# Partial Programs

# Partial Programs



Program

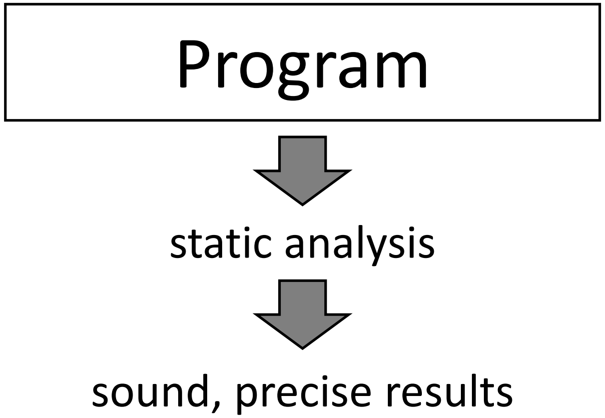
# Partial Programs



Program

static analysis

# Partial Programs

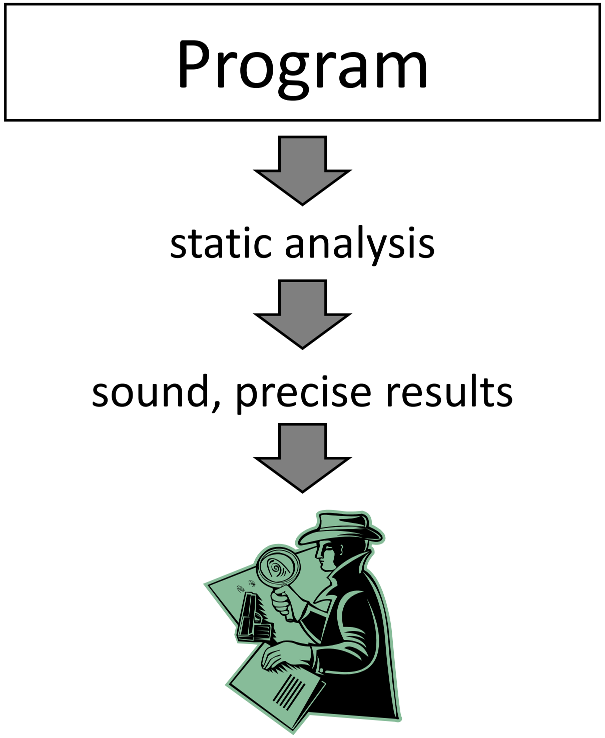


Program

static analysis

sound, precise results

# Partial Programs



Program

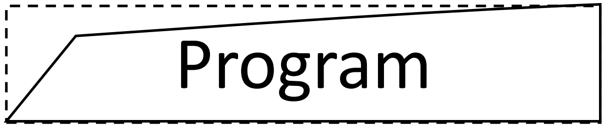
static analysis

sound, precise results

# Partial Programs

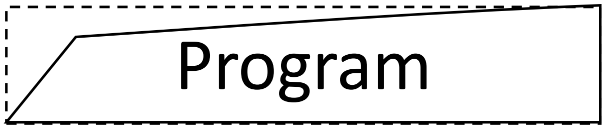
Program

# Partial Programs



Program

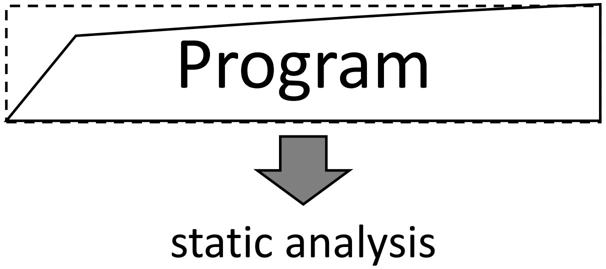
# Partial Programs



Program

**Approach 1:** treat as no-ops

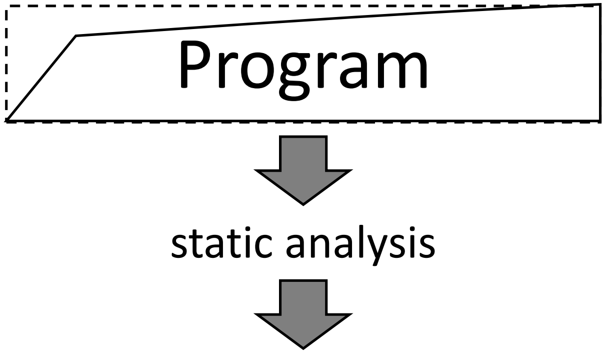
# Partial Programs



**Approach 1:** treat as no-ops

# Partial Programs

**Approach 1:** treat as no-ops



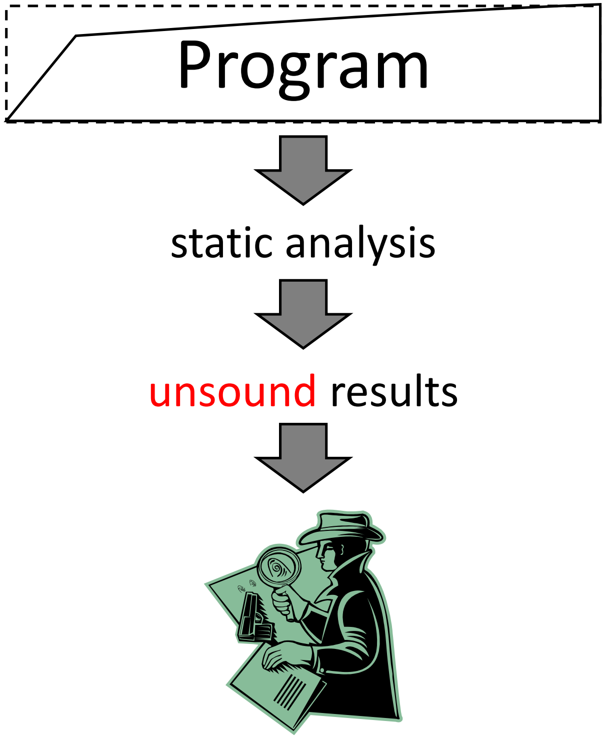
Program

static analysis

unsound results

# Partial Programs

**Approach 1:** treat as no-ops

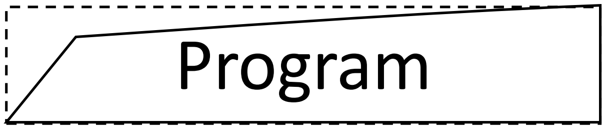


Program

static analysis

unsound results

# Partial Programs

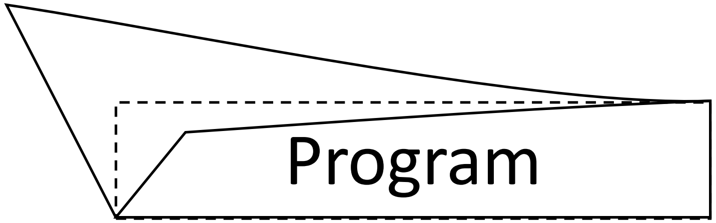


Program

**Approach 1:** treat as no-ops

**Approach 2:** worst-case

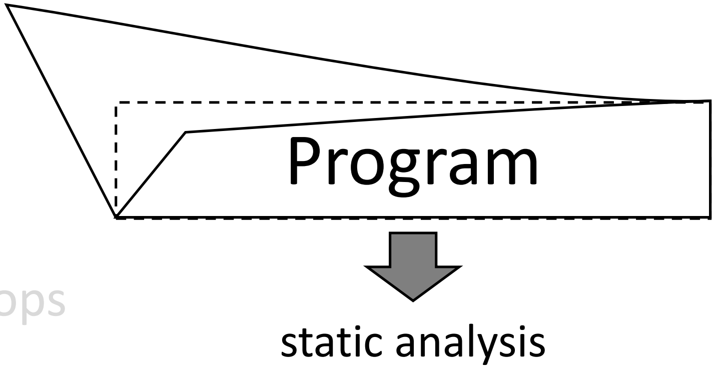
# Partial Programs



**Approach 1:** treat as no-ops

**Approach 2:** worst-case

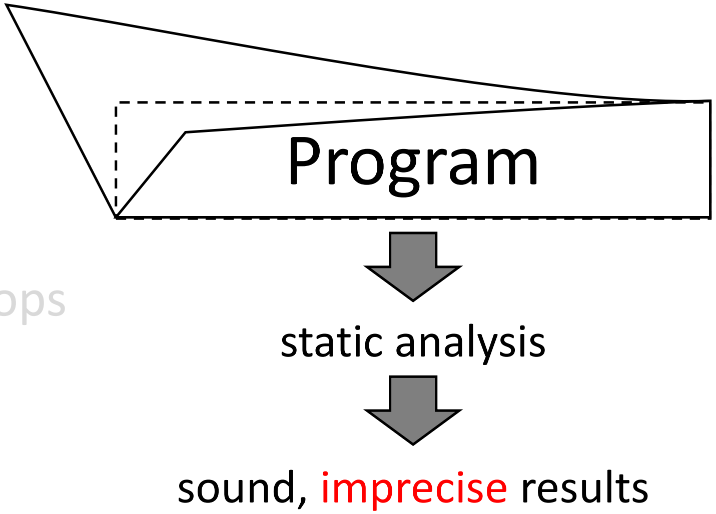
# Partial Programs



**Approach 1:** treat as no-ops

**Approach 2:** worst-case

# Partial Programs



static analysis

sound, imprecise results

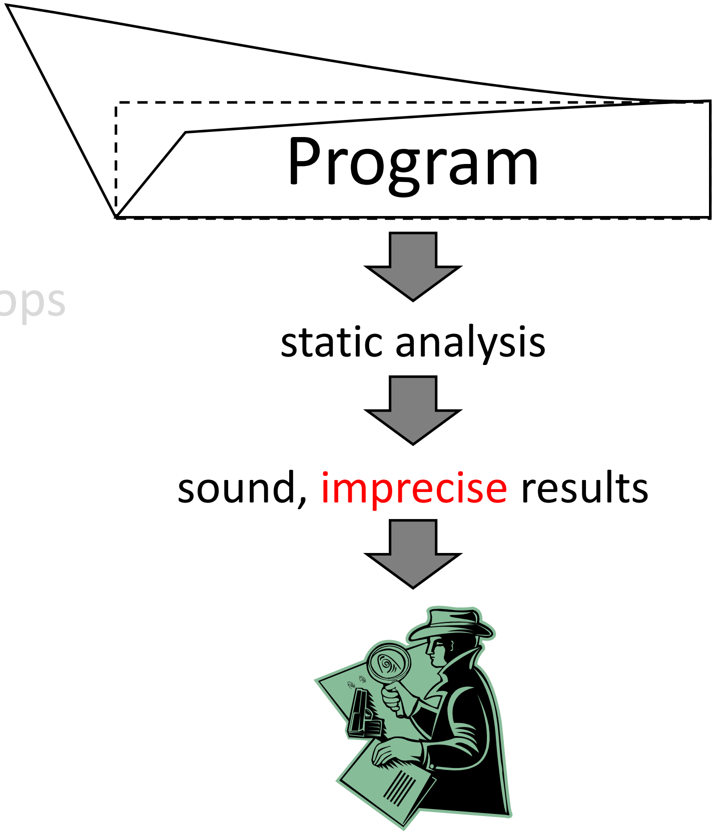
**Approach 1:** treat as no-ops

**Approach 2:** worst-case

# Partial Programs

Approach 1: treat as no-ops

**Approach 2:** worst-case

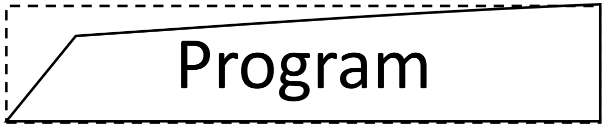


Program

static analysis

sound, imprecise results

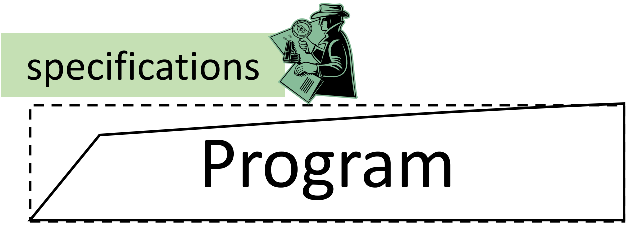
# Partial Programs



Program

**Approach 1:** treat as no-ops
**Approach 2:** worst-case
**Approach 3:** specifications

# Partial Programs

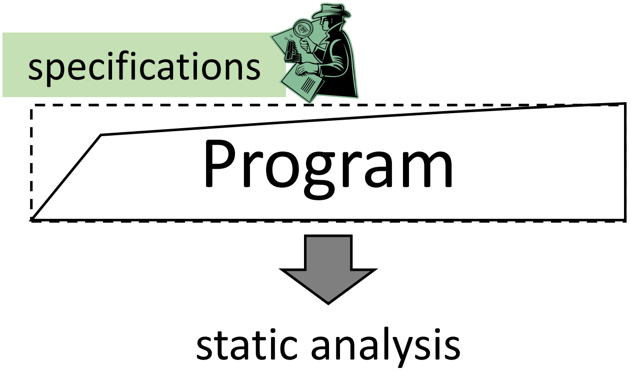


**Approach 1:** treat as no-ops

**Approach 2:** worst-case

**Approach 3:** specifications

# Partial Programs

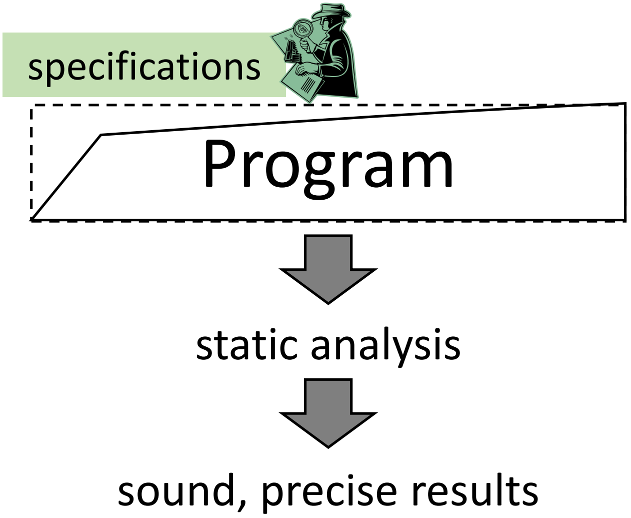


**Approach 1:** treat as no-ops

**Approach 2:** worst-case

**Approach 3:** specifications

# Partial Programs



**Approach 1:** treat as no-ops

**Approach 2:** worst-case

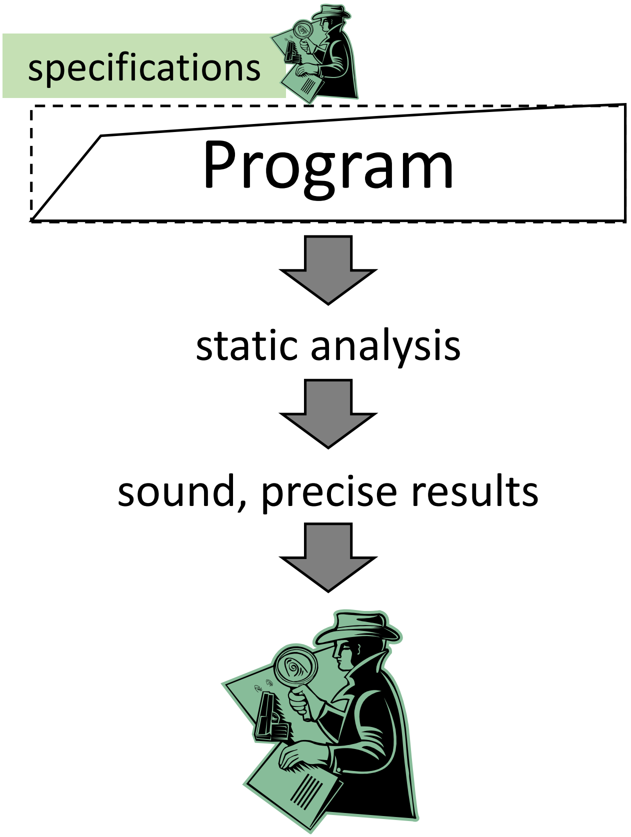
**Approach 3:** specifications

# Partial Programs

**Approach 1:** treat as no-ops

**Approach 2:** worst-case

**Approach 3:** specifications



specifications

Program

static analysis

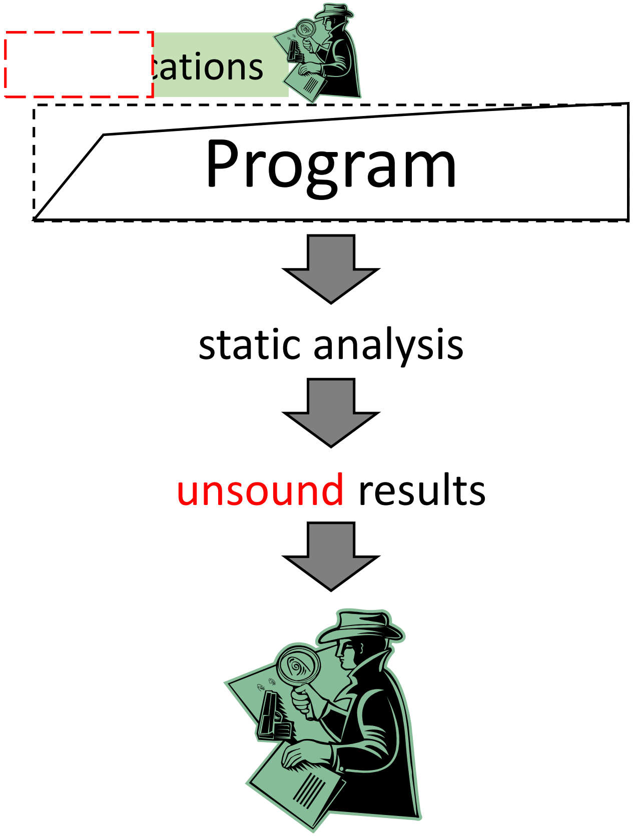
sound, precise results

# Partial Programs

**Approach 1:** treat as no-ops

**Approach 2:** worst-case

**Approach 3:** specifications



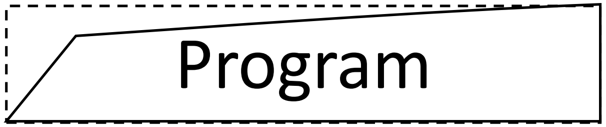
ations

Program

static analysis

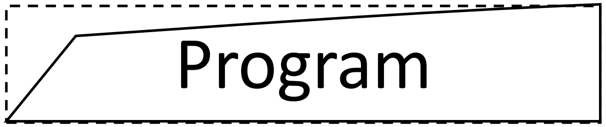
unsound results

# Specification Inference



Program

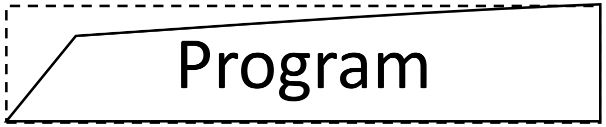
# Specification Inference



Program

**Our approach:**

# Specification Inference

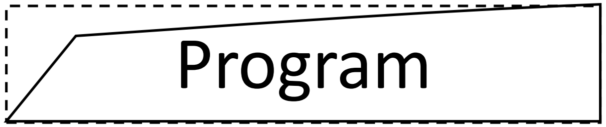


Program

**Our approach:**

**(builds on [Zhu, Dillig, Dillig 2013])**

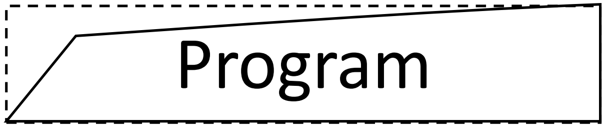
# Specification Inference



Program

**Our approach:**

# Specification Inference

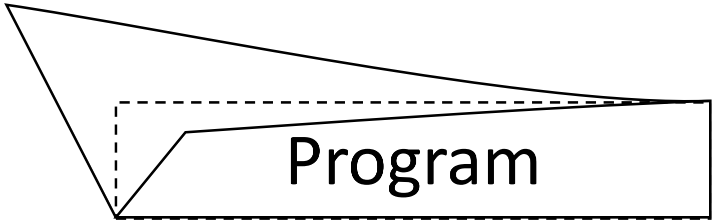


Program

**Our approach:**

1) Worst-case analysis

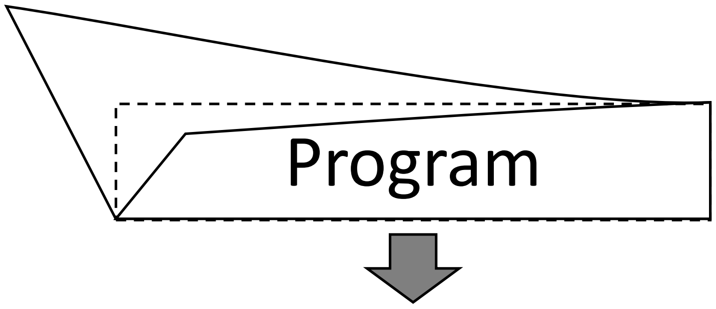
# Specification Inference



**Our approach:**

1) Worst-case analysis

# Specification Inference

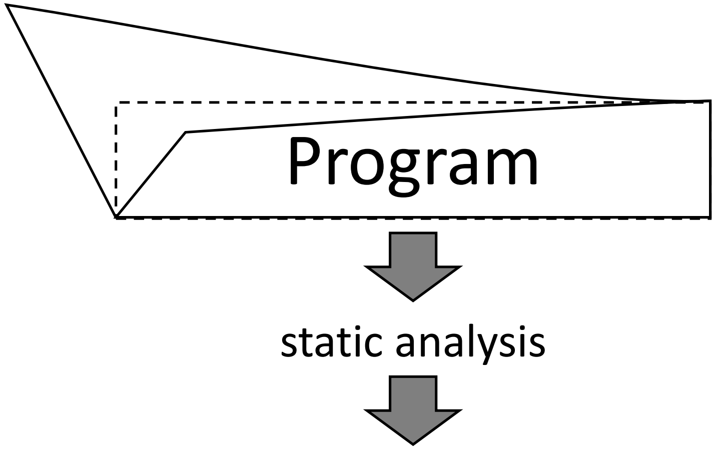


static analysis

**Our approach:**

1) Worst-case analysis

# Specification Inference



Program

static analysis

sound, imprecise results

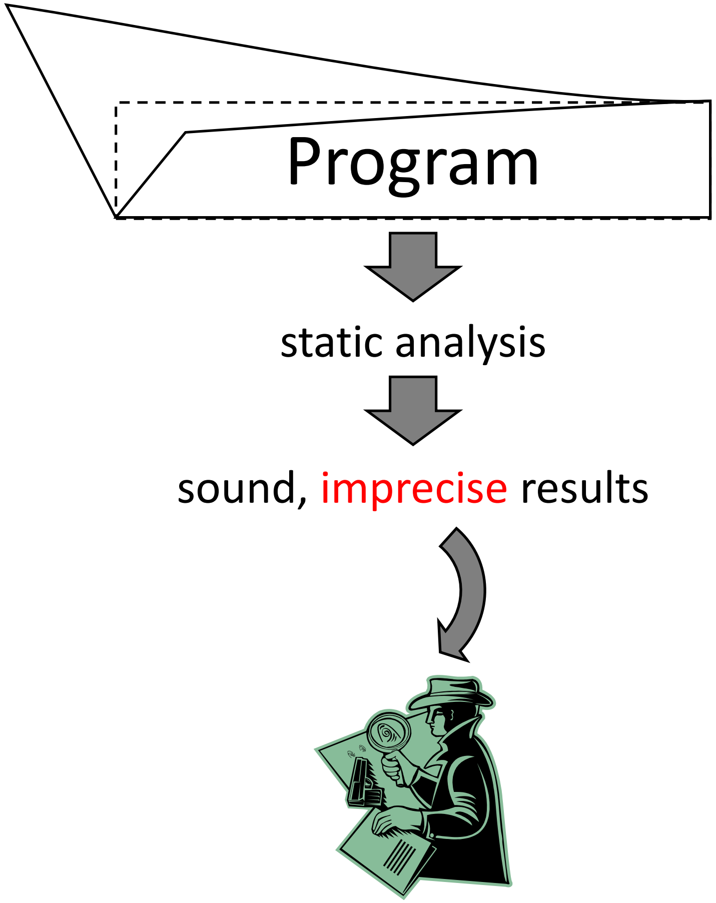
**Our approach:**

1) Worst-case analysis

# Specification Inference

**Our approach:**

1) Worst-case analysis



Program

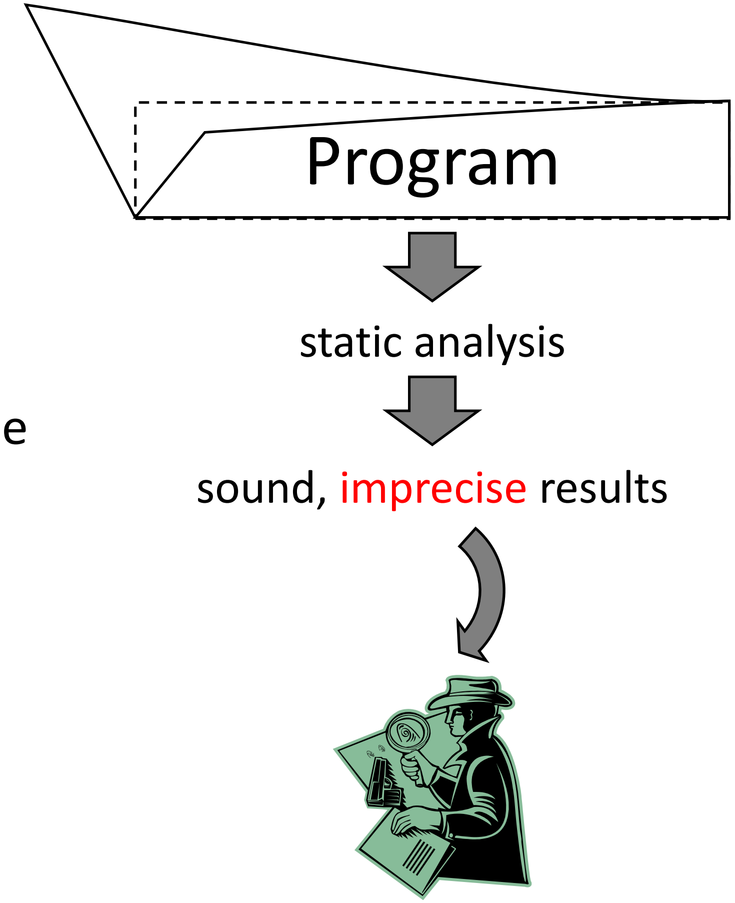
static analysis

sound, imprecise results

# Specification Inference

**Our approach:**

1) Worst-case analysis
2) Specification inference

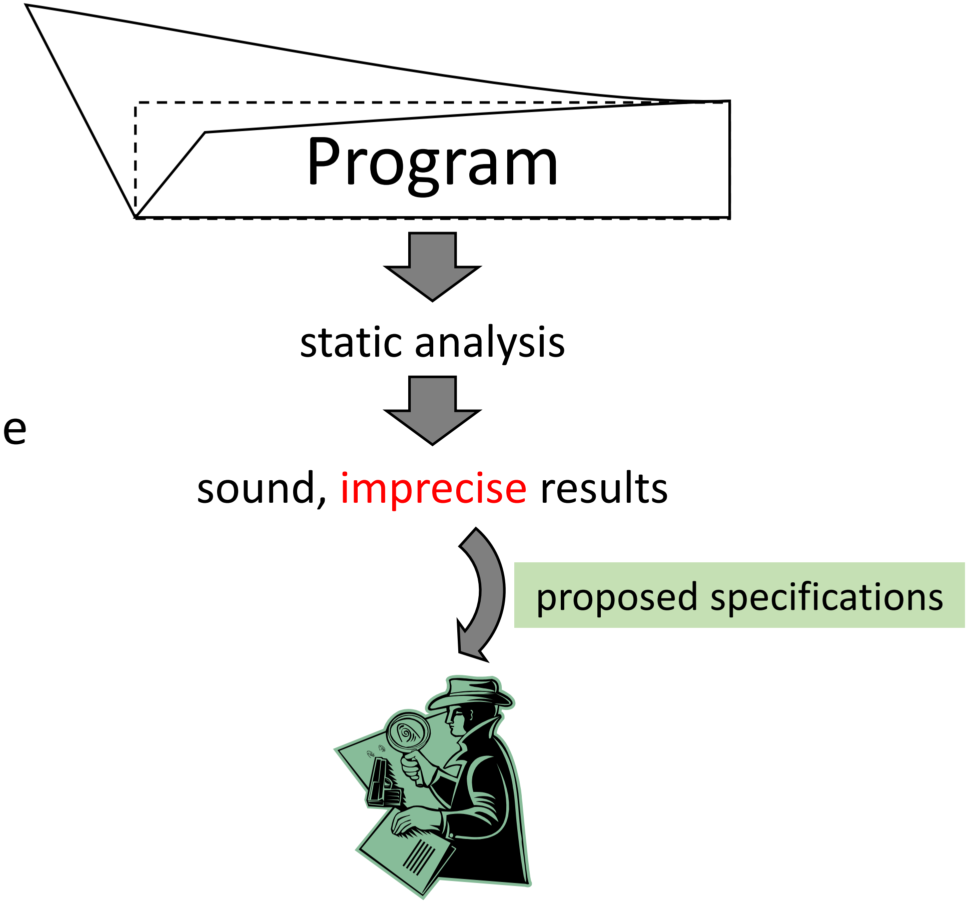


Program

static analysis

sound, imprecise results

# Specification Inference



Program

static analysis

sound, imprecise results

proposed specifications

**Our approach:**

1) Worst-case analysis

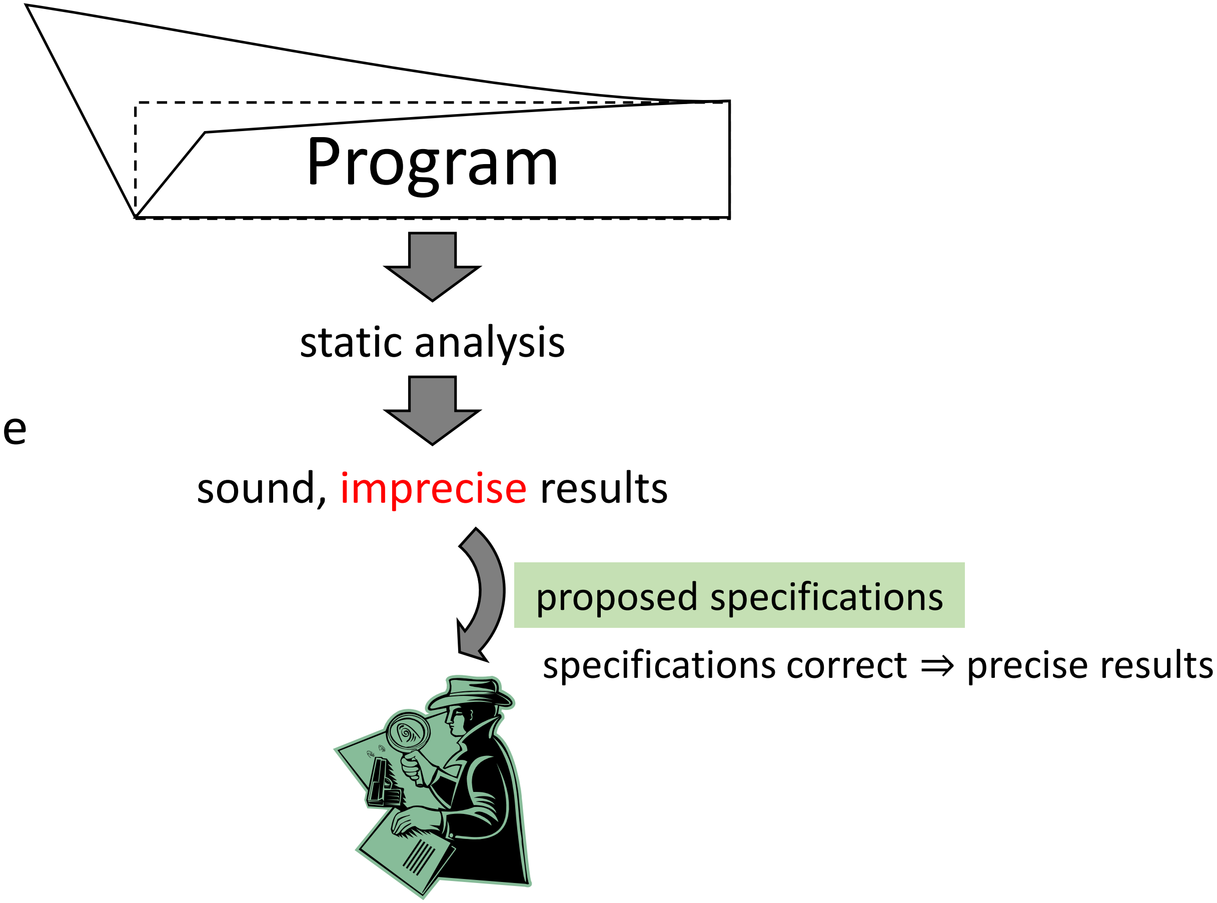
2) Specification inference

# Specification Inference

**Our approach:**

1) Worst-case analysis

2) Specification inference



Program

static analysis

sound, imprecise results

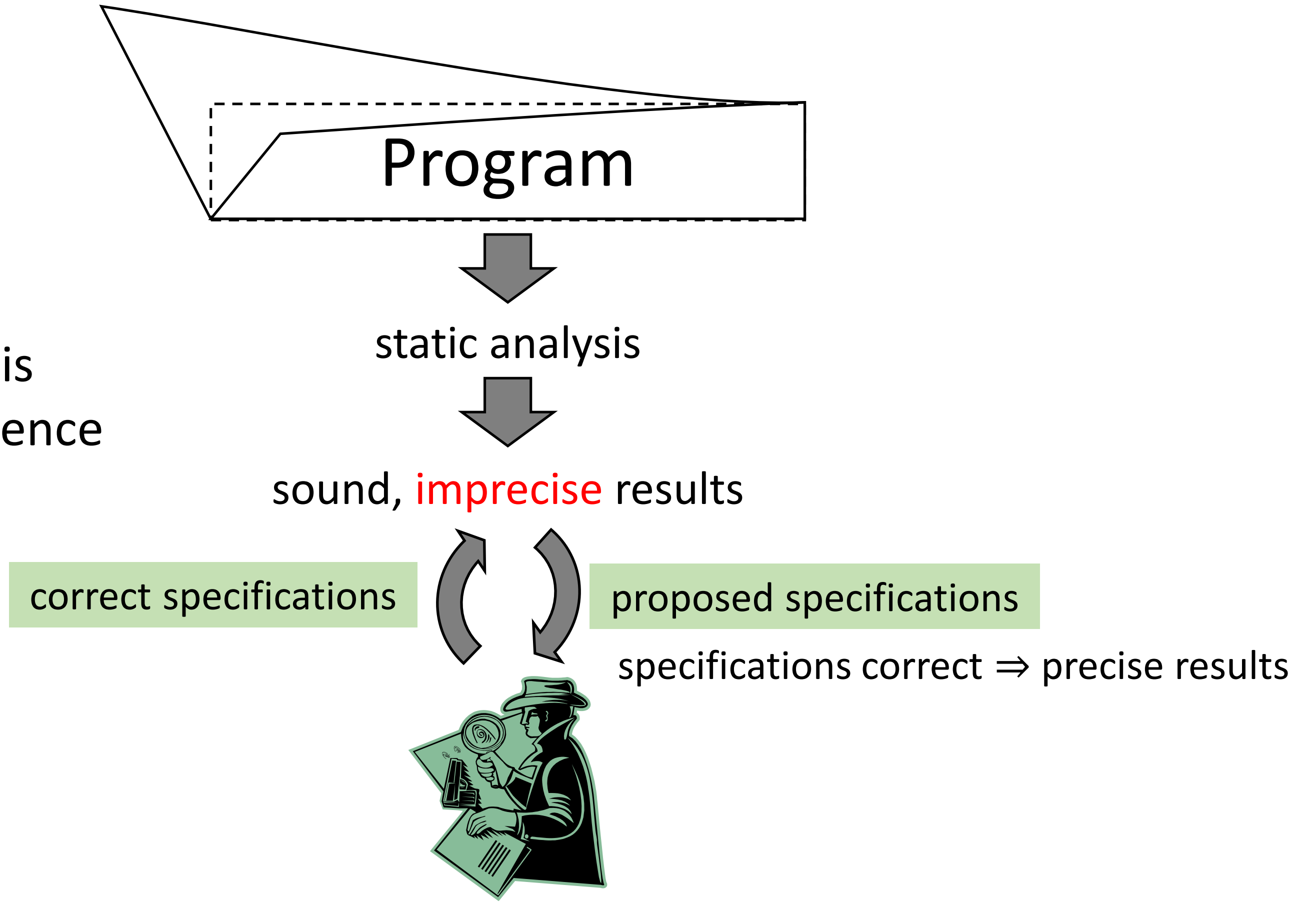
proposed specifications

specifications correct ⇒ precise results

# Specification Inference

**Our approach:**

1) Worst-case analysis
2) Specification inference



Program

static analysis

sound, imprecise results

correct specifications

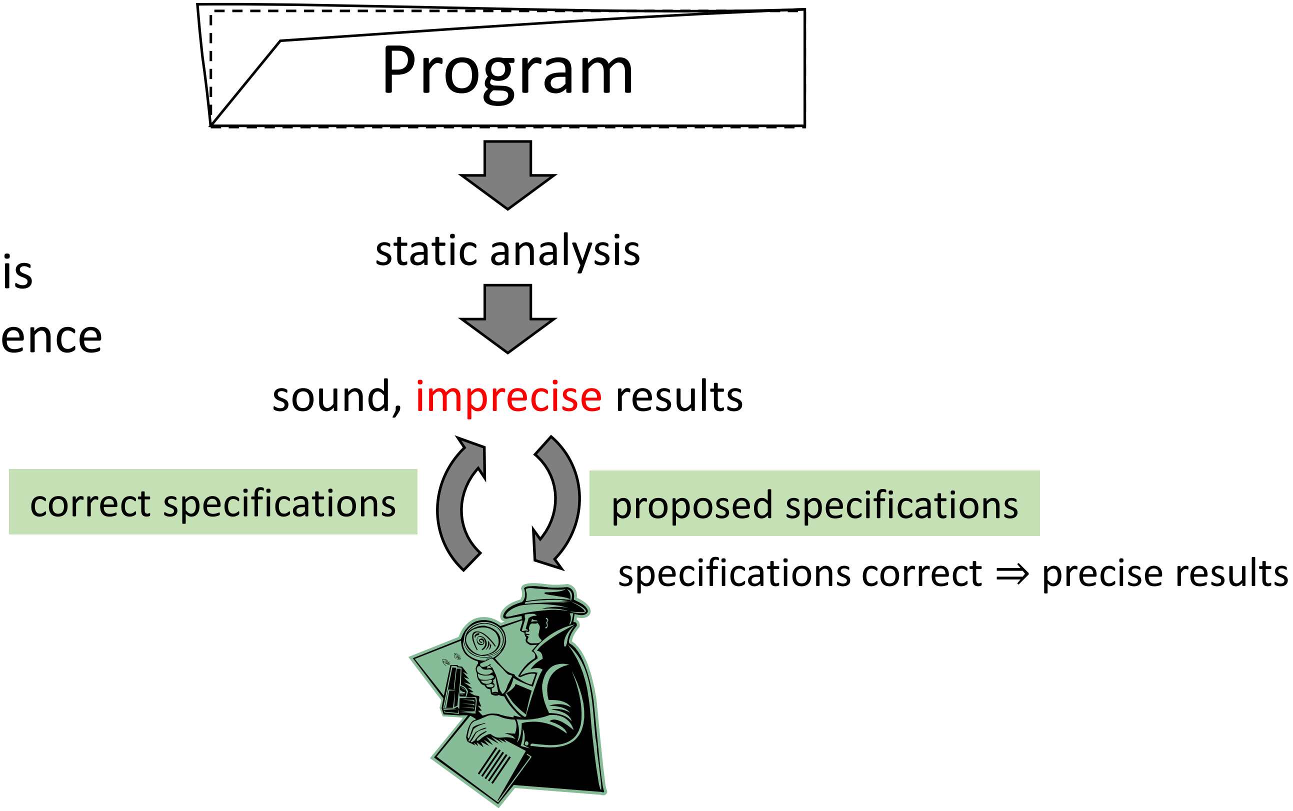
proposed specifications

specifications correct ⇒ precise results

# Specification Inference

**Our approach:**

1) Worst-case analysis
2) Specification inference



Program

static analysis

sound, imprecise results

correct specifications

proposed specifications

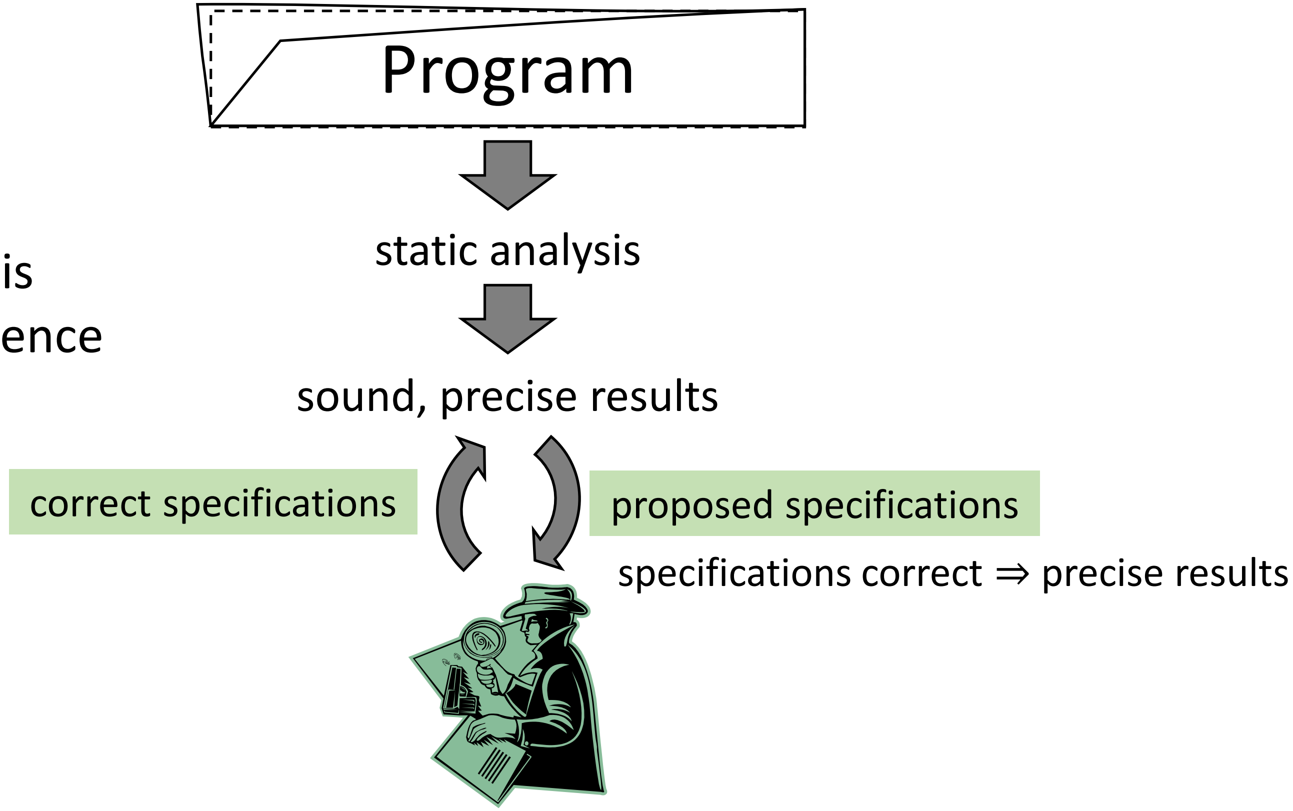
specifications correct ⇒ precise results

# Specification Inference

**Our approach:**

1) Worst-case analysis

2) Specification inference



Program

static analysis

sound, precise results

correct specifications

proposed specifications

specifications correct ⇒ precise results

# Specification Inference

- **Sound & Precise**
    - Using interaction
    - Finds the same results as if all specifications are written

# Information Flow for Android

- Finding Android malware using source to sink flows

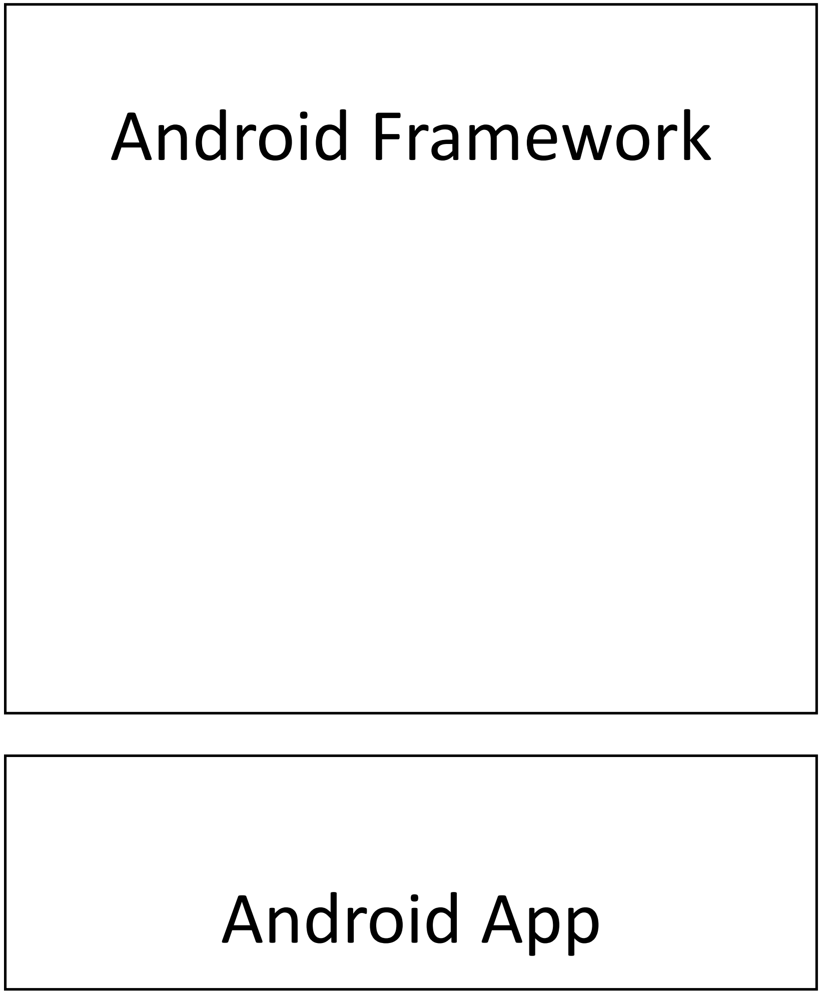
| | | | |
|---|---|---|---|
| **Tracking:** | location | leaks to | Internet |
| **Premium SMS:** | phone # | used in | SMS send |
| **Ransomware:** | network packets | encrypt | files |

# Information Flow for Android

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

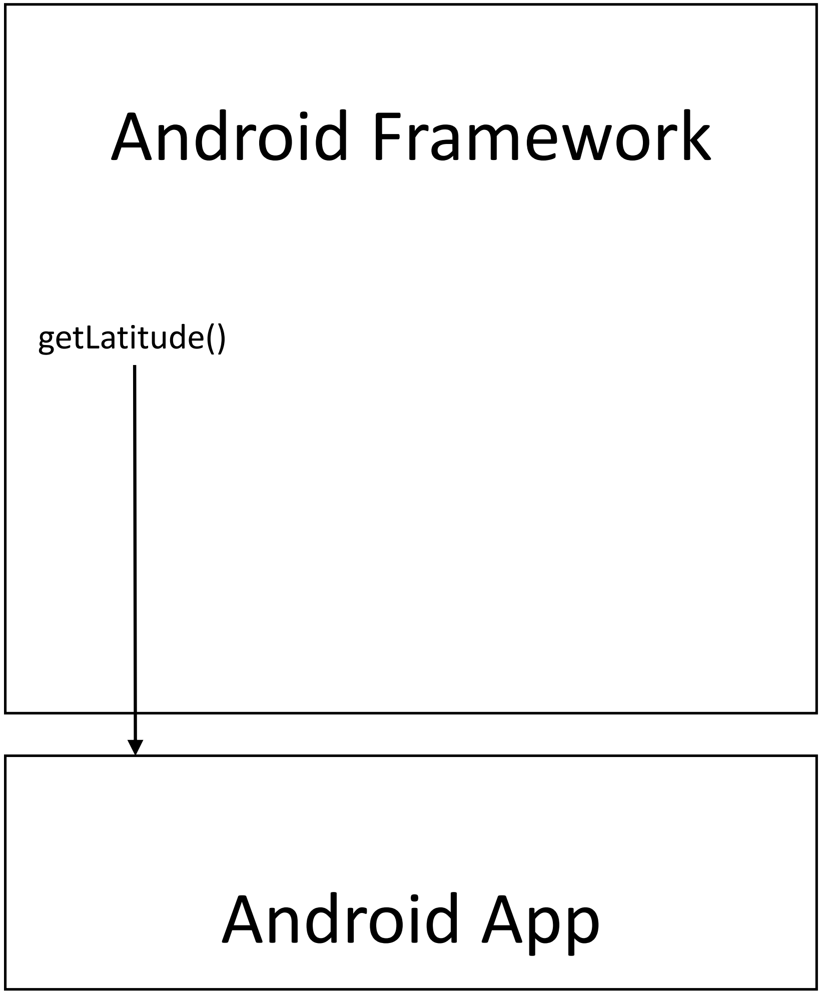
# Information Flow for Android

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```



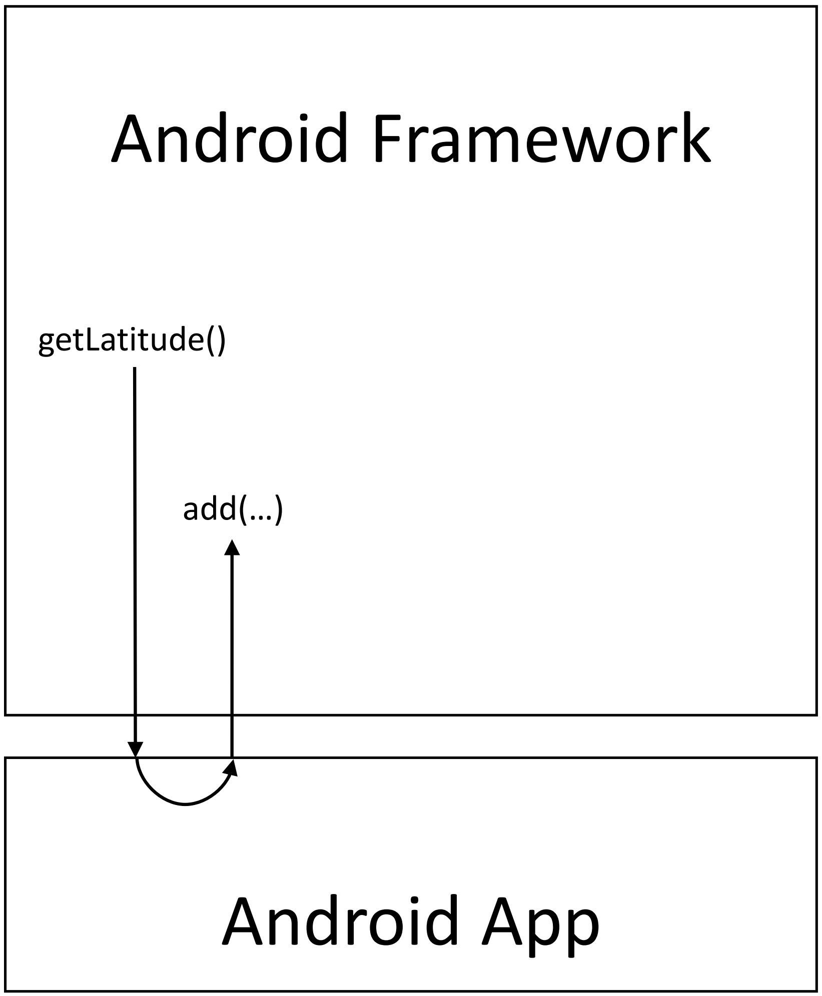
# Information Flow for Android

```
1. Double lat = getLatitude();
2. List list = new List();
3. list.add(lat);
4. Double latAlias = list.get(0);
5. String latStr = latAlias.toString();
6. sendSMS(latStr);
```



# Information Flow for Android

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```



Android Framework

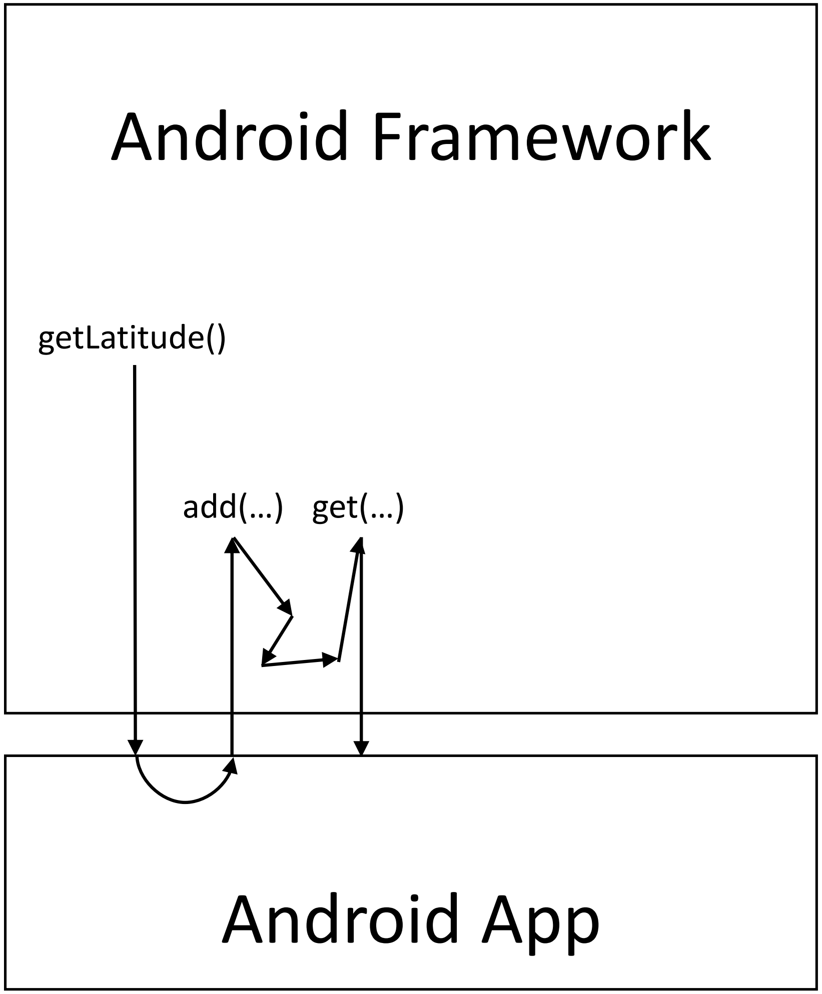
getLatitude()

add(…)

Android App

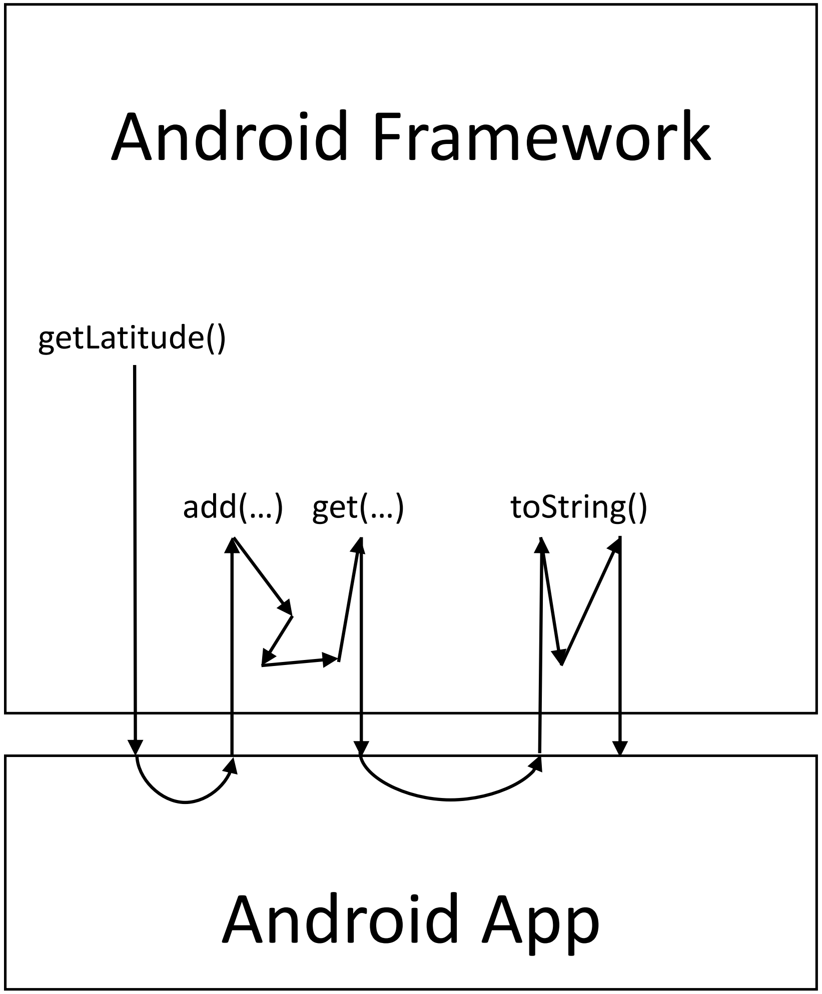
# Information Flow for Android

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```



# Information Flow for Android

```
1. Double lat = getLatitude();
2. List list = new List();
3. list.add(lat);
4. Double latAlias = list.get(0);
5. String latStr = latAlias.toString();
6. sendSMS(latStr);
```



Android Framework

getLatitude()

add(…)

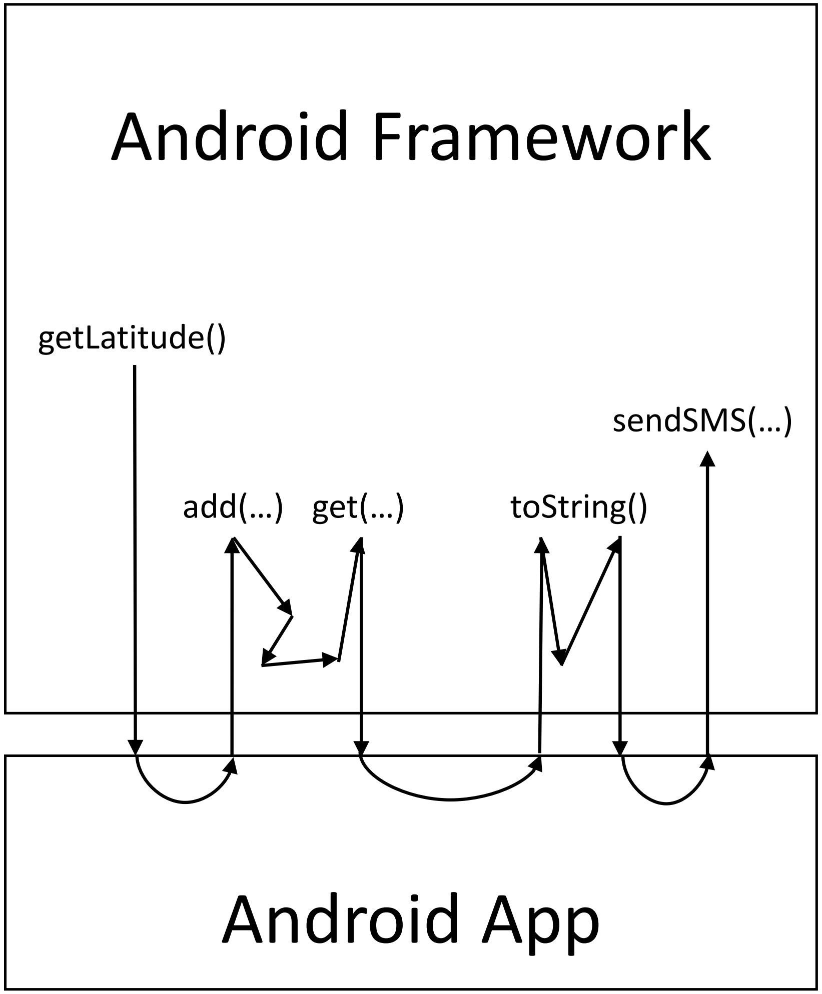
get(…)

toString()

Android App

# Information Flow for Android

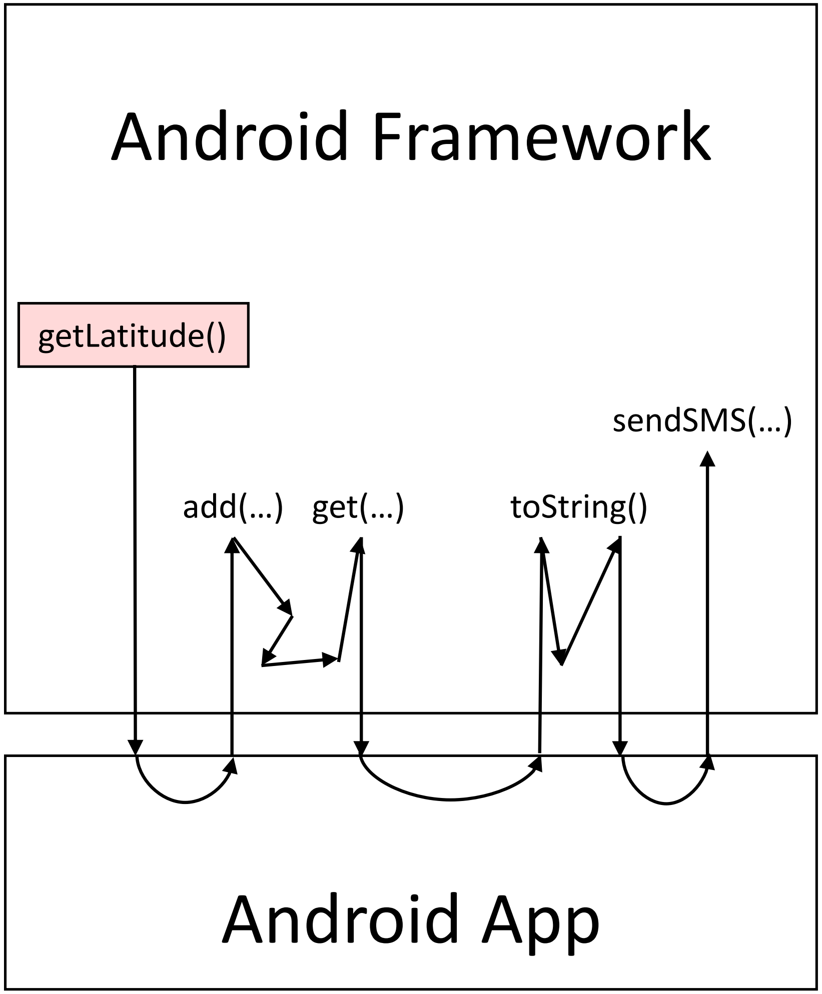
1. Double lat = getLatitude();
2. List list = new List();
3. list.add(lat);
4. Double latAlias = list.get(0);
5. String latStr = latAlias.toString();
6. sendSMS(latStr);



# Information Flow for Android

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

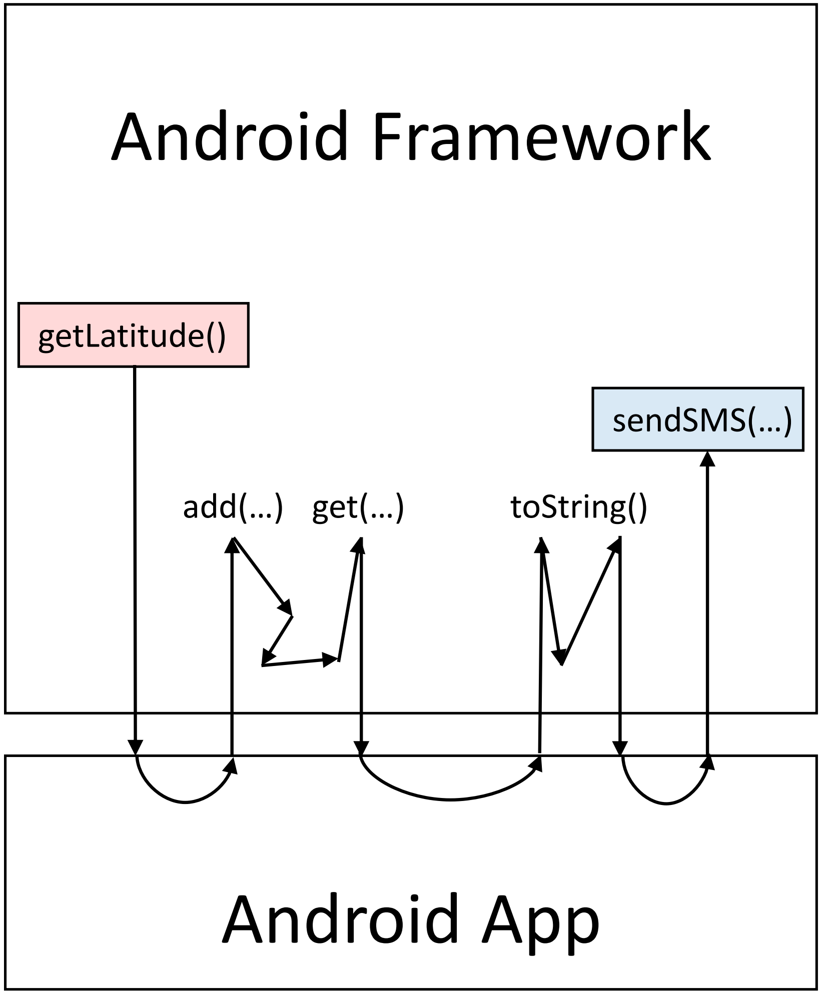
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() { … }
```



# Information Flow for Android

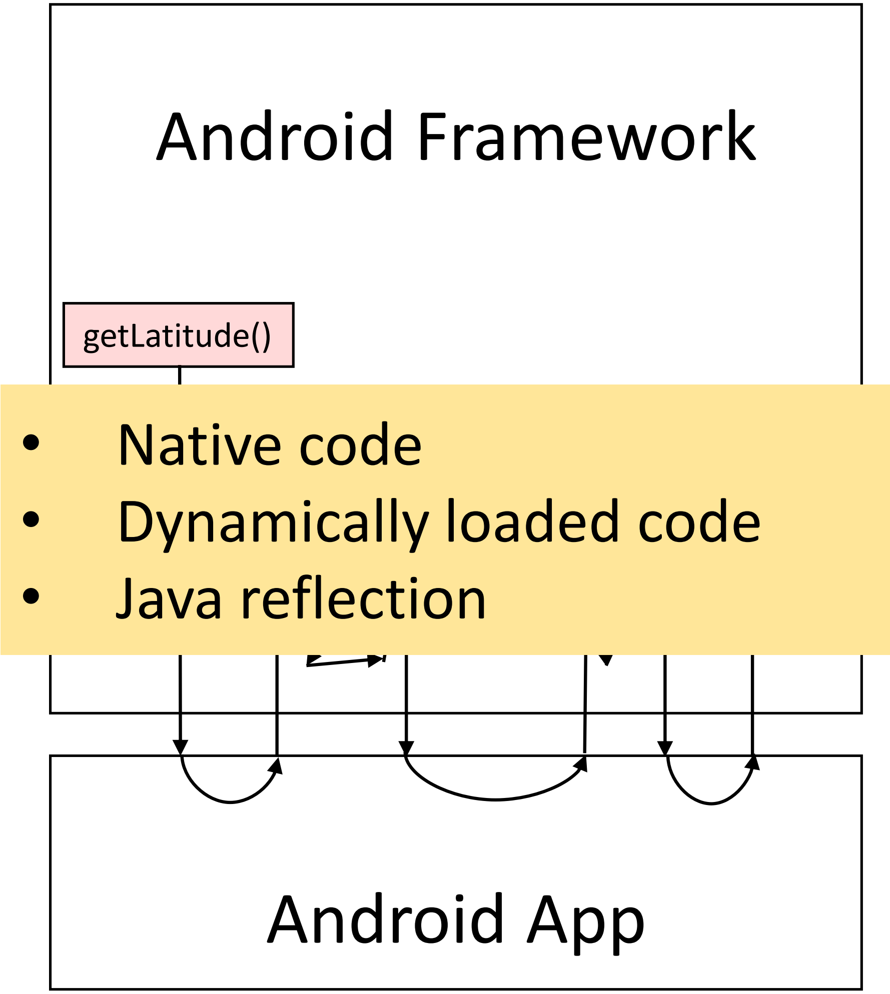
```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

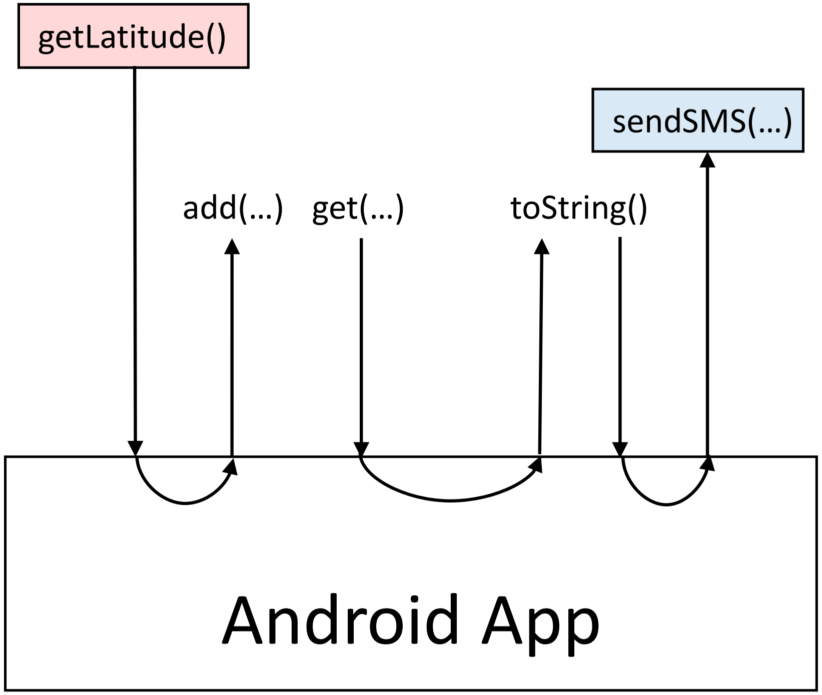


- Native code
- Dynamically loaded code
- Java reflection

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

# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

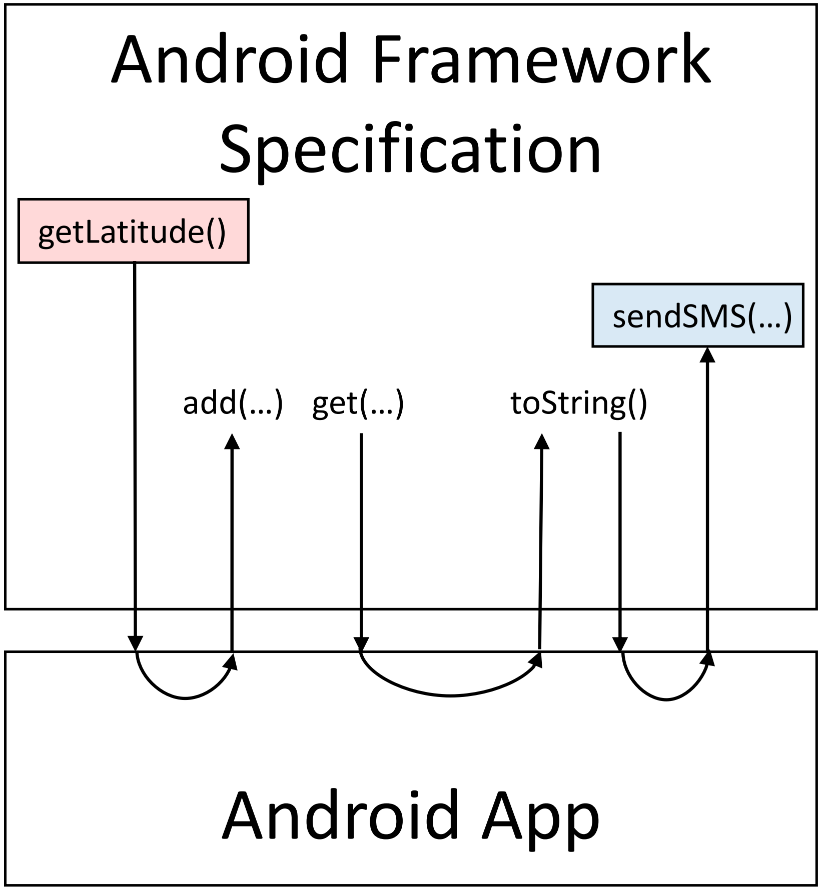


```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```

# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(…)

add(…)

get(…)

toString()

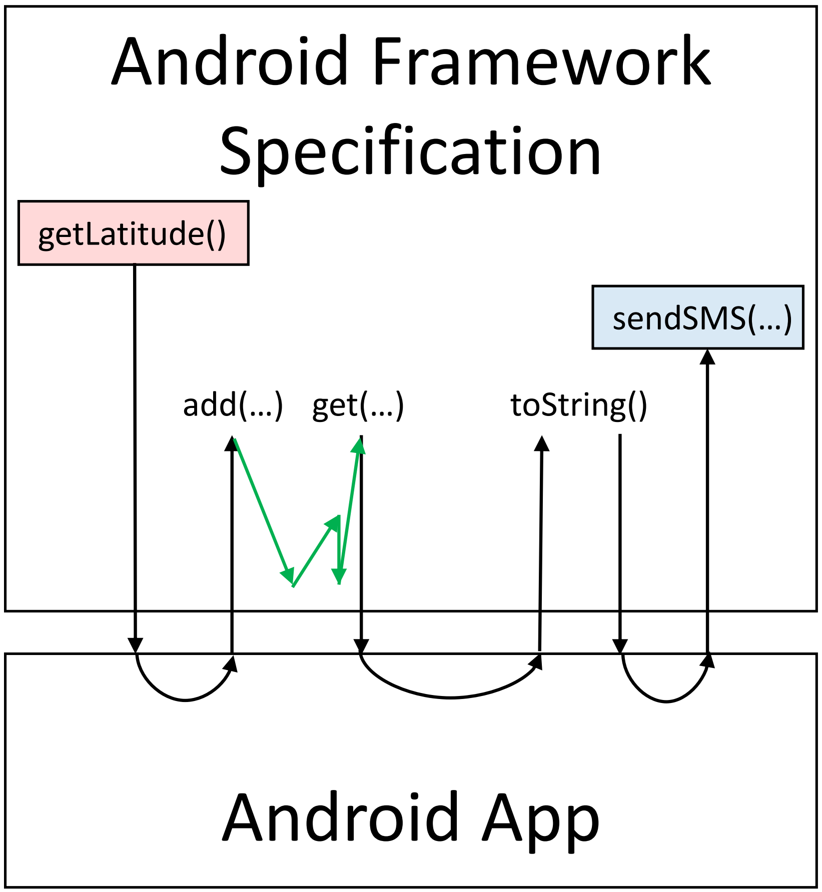
Android App

# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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

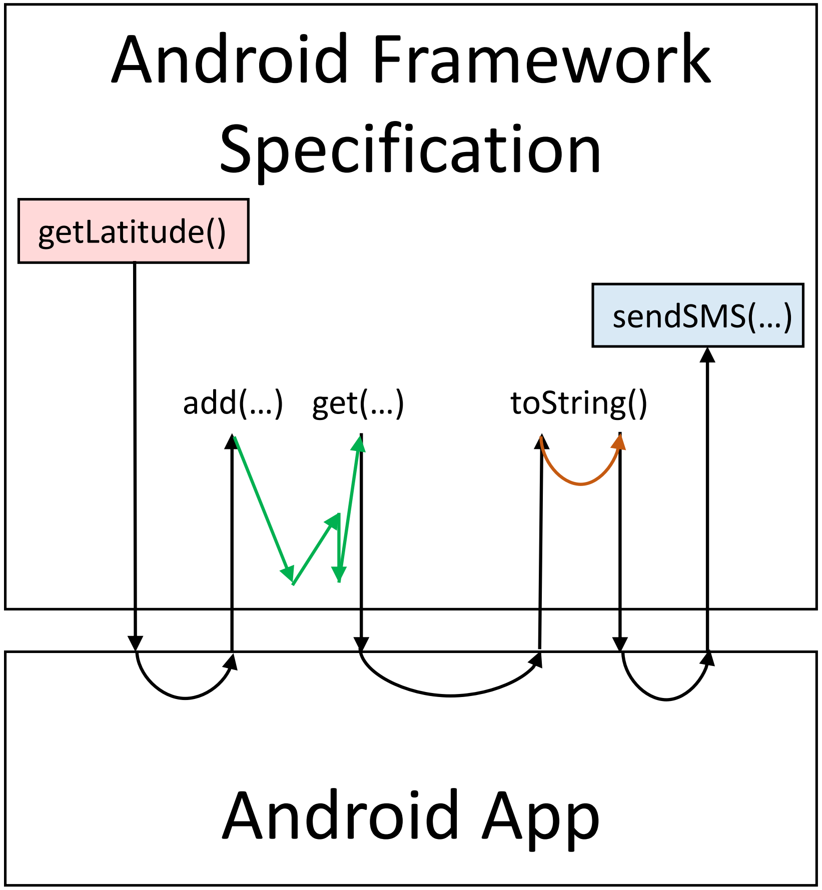
```
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text, SMS)
    static void sendSMS(String text) {}
```



# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

add(...)

get(...)

toString()

Android App

# Framework Specifications

- **Specification:** over-approximates behavior of framework methods
  - Provided by the user
  - More precise than automated approaches

# Framework Specifications

- **Alias Specifications:** describes aliasing
  - @Alias(x, y) means “x aliases y”

  - class List:

    ```
        @Alias(arg, this.val)
        void add(Object arg) {}
        @Alias(this.val, return)
        Object get(Integer index) {}
    ```

# Framework Specifications

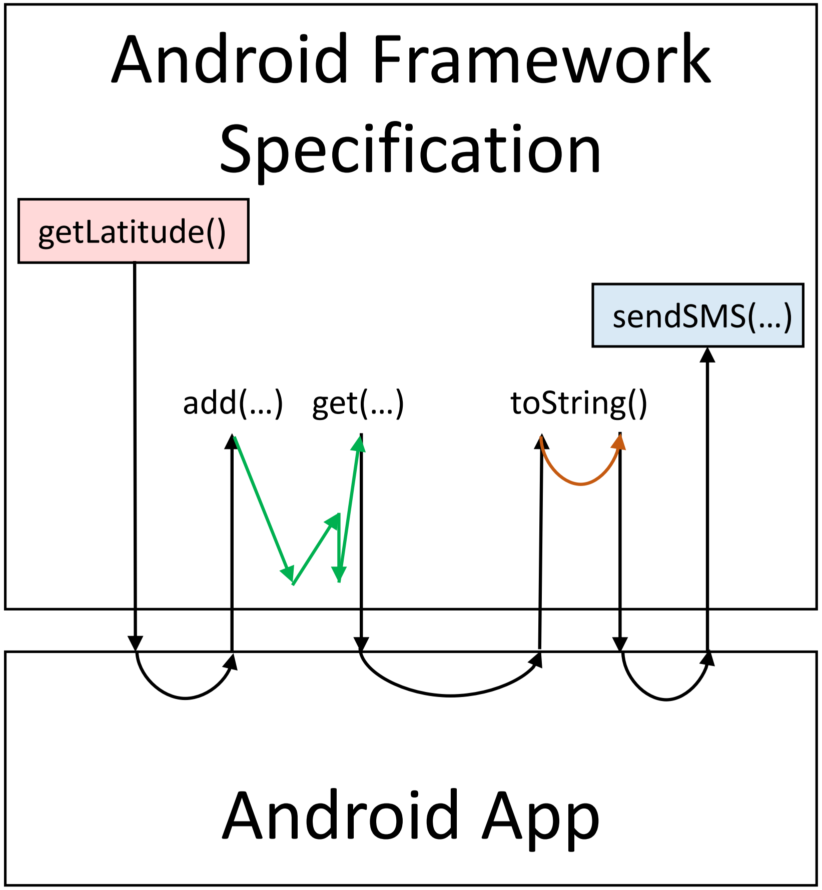
- **Flow Specifications:** describe information flows
    - @Flow(x, y) means “x tainted ⇒ y tainted”
    - class Double:

      ```
      @Flow(this, return)
      String toString() {}
      ```

# Framework Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

add(...) get(...) toString()

Android App

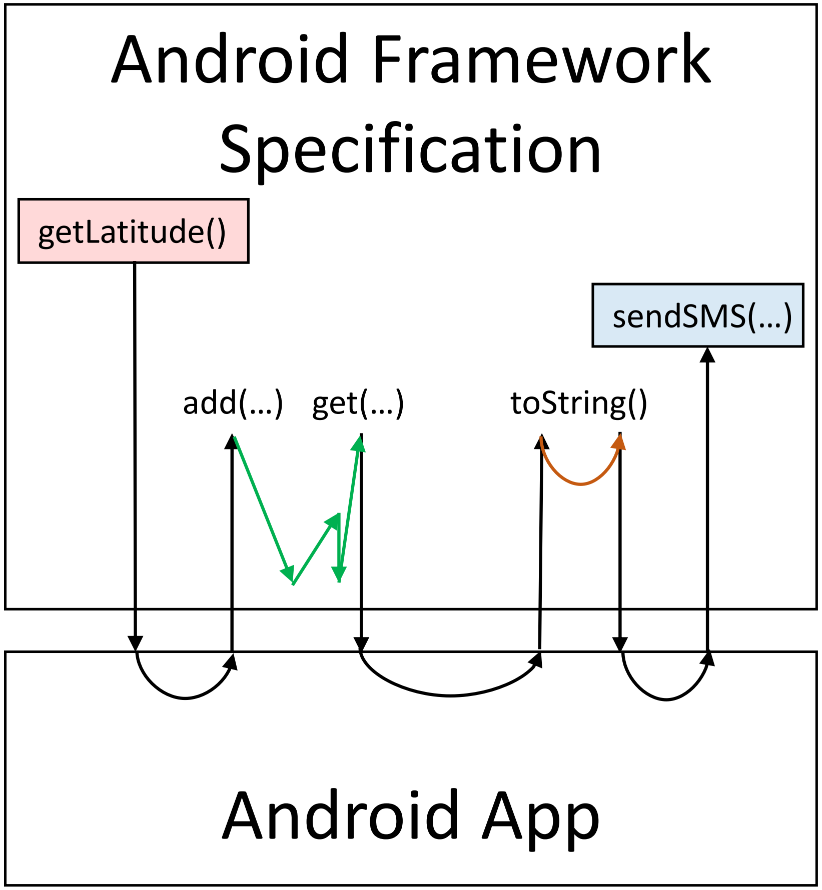
# Missing Specifications

- Specifications typically written as needed
  - $\approx 4{,}000$ framework classes
  - A given app may use hundreds of classes
  - For a given app, only a few classes are relevant for finding information flows
  - Our experience: specifications for $\approx 175$ classes over course of a year

# Missing Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

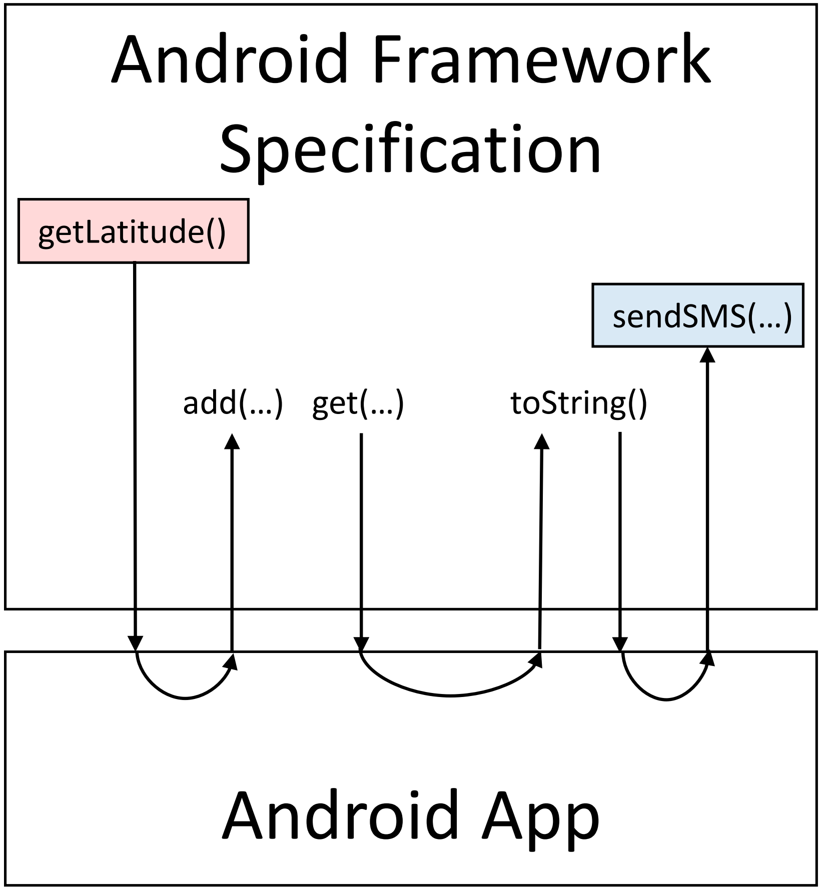
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Missing Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

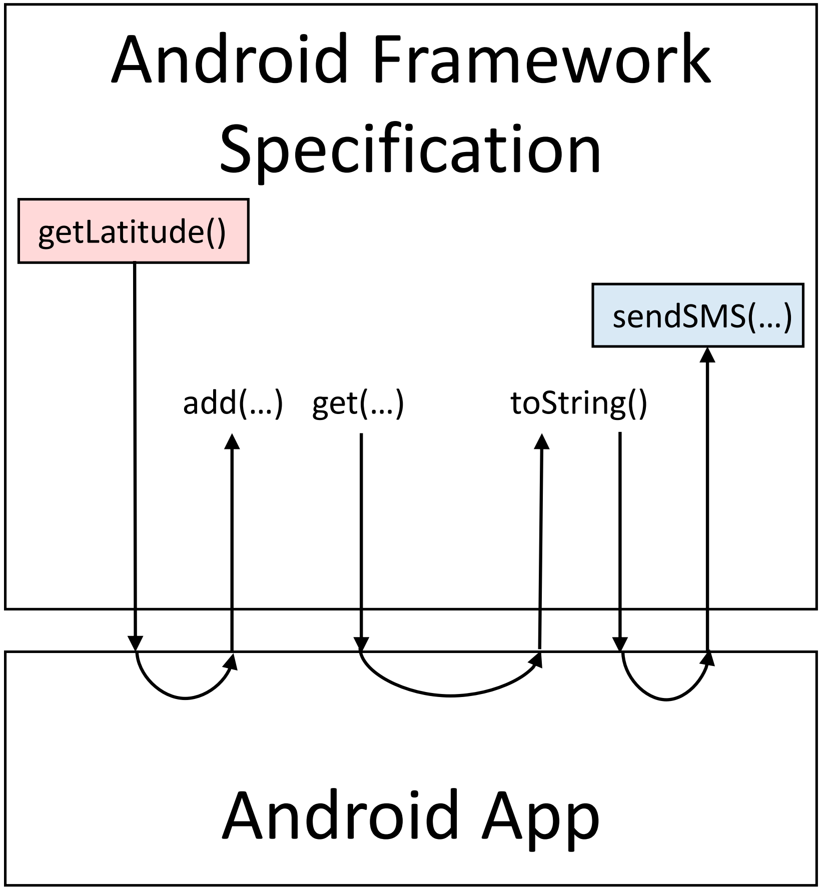
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Step 1: Worst-case Analysis

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

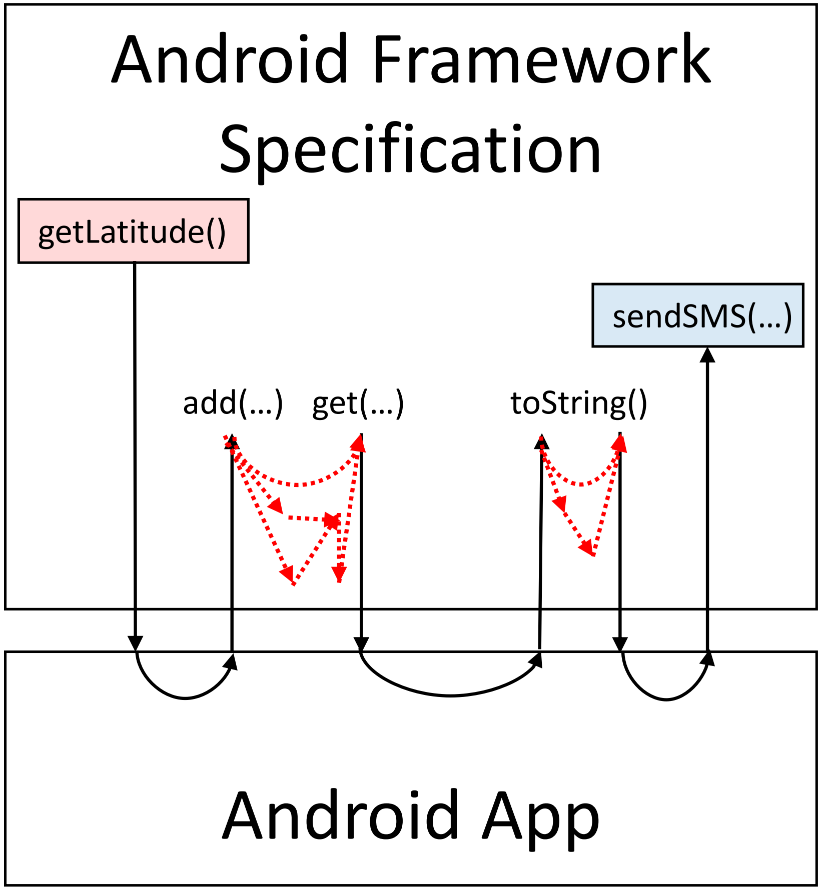
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Step 1: Worst-case Analysis

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

add(...)

get(...)

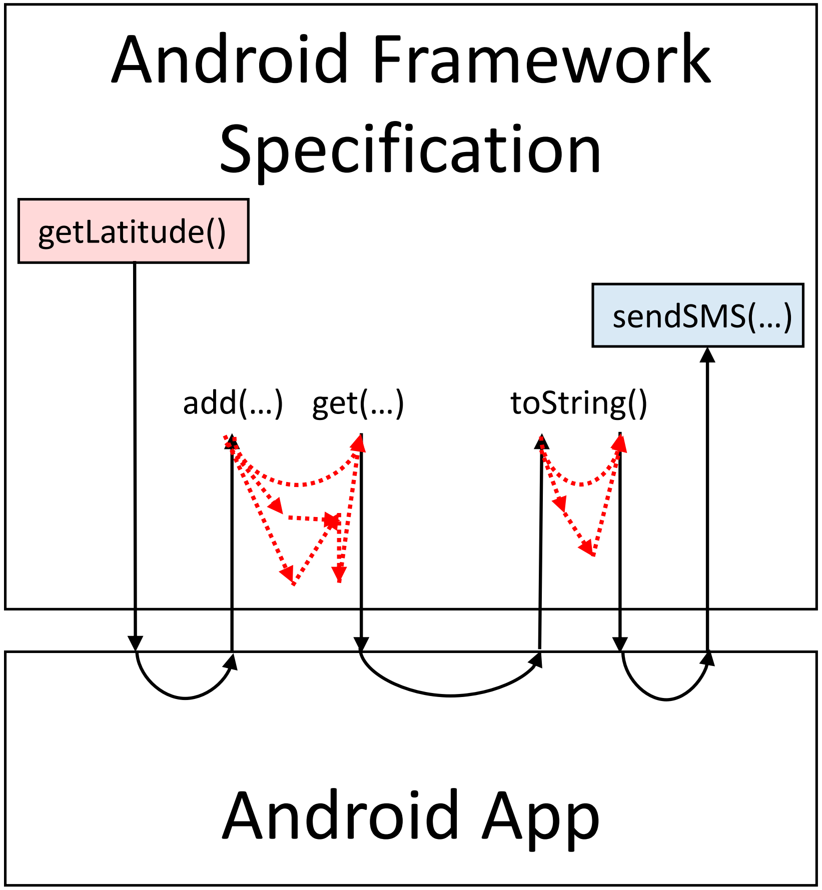
toString()

Android App

# Step 2: Specification Inference

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

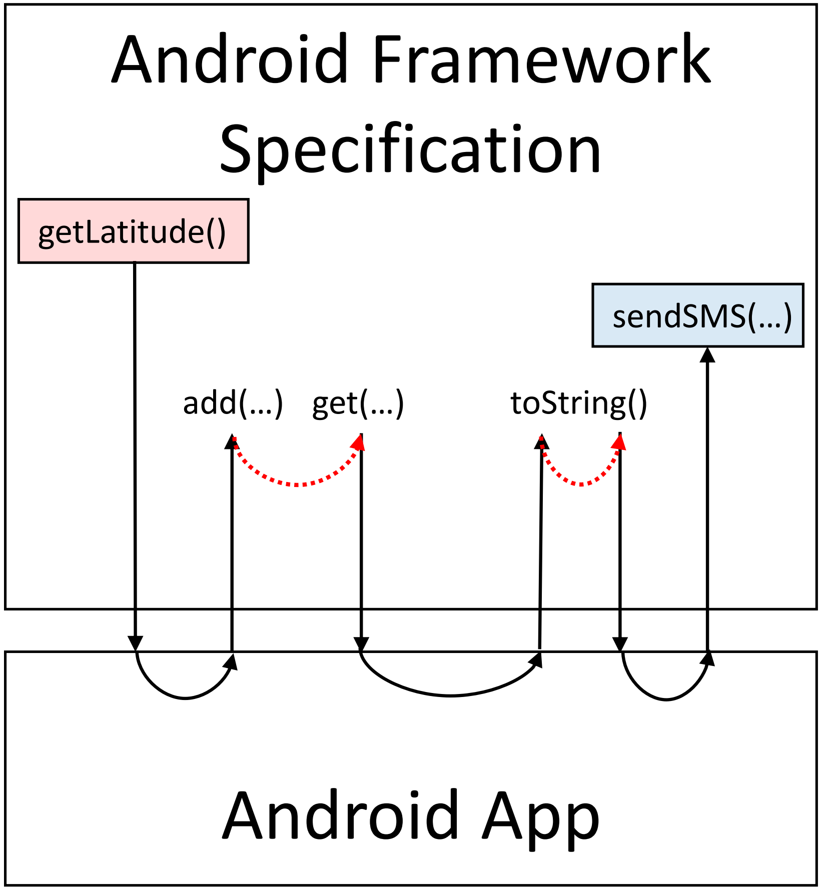
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Step 2: Specification Inference

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(…)

add(…)

get(…)

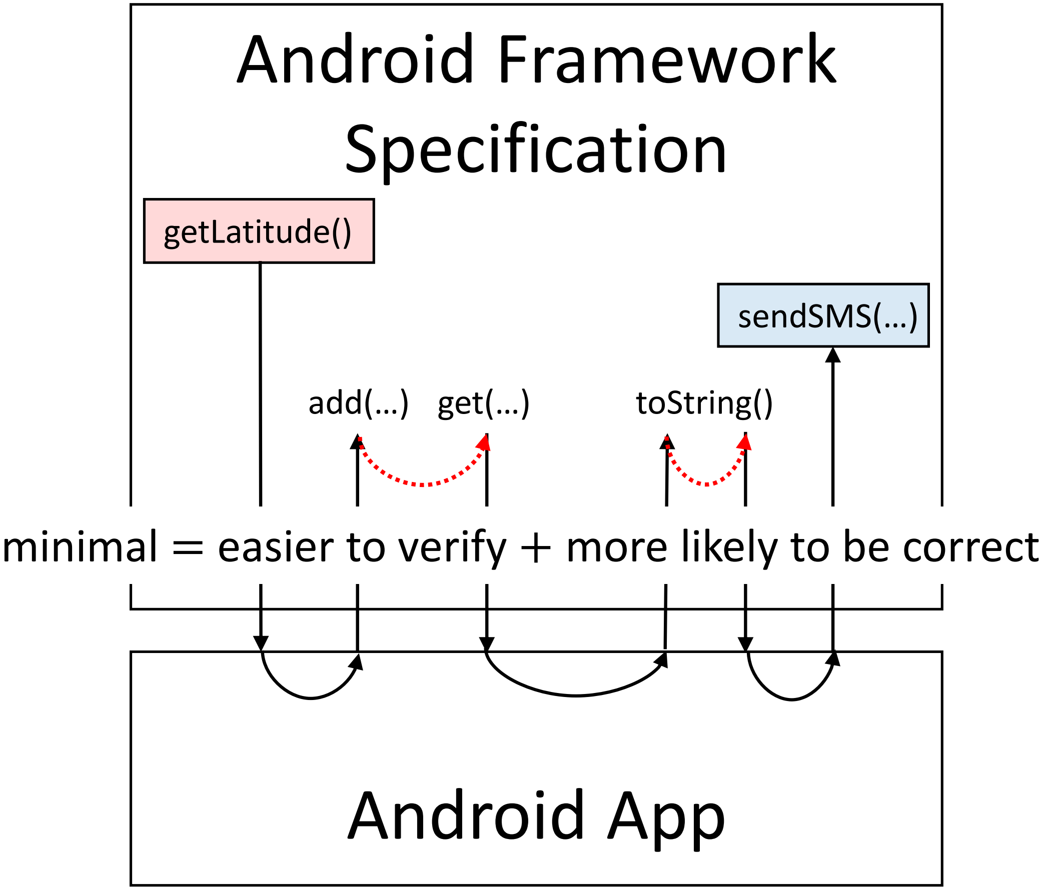
toString()

Android App

# Step 2: Specification Inference

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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

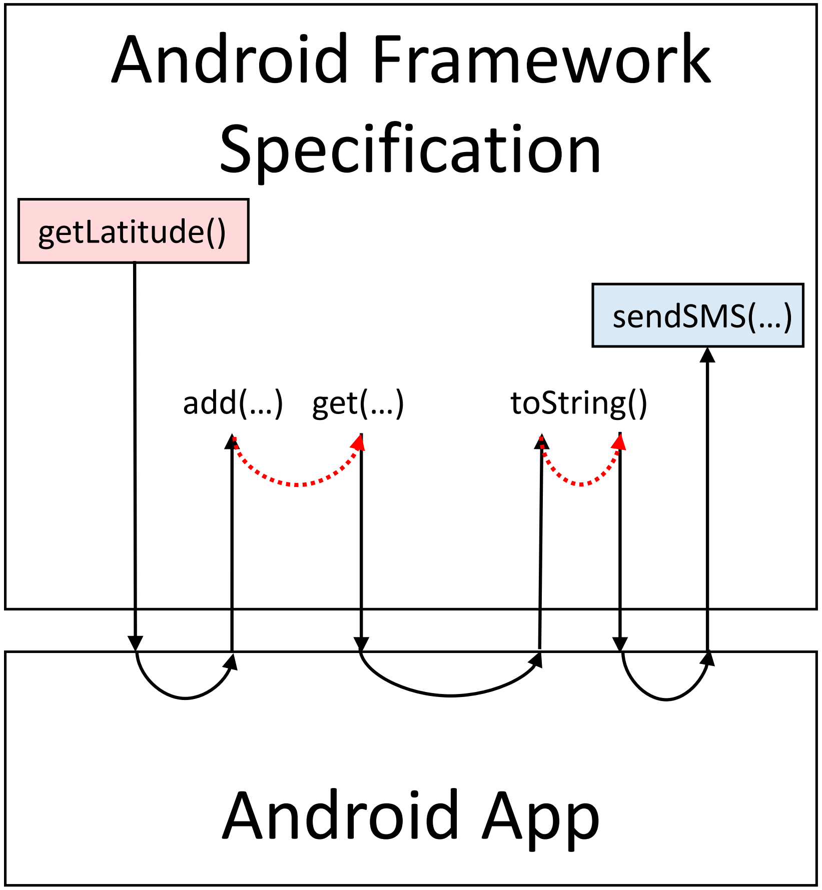


minimal = easier to verify + more likely to be correct

# Step 2: Specification Inference

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

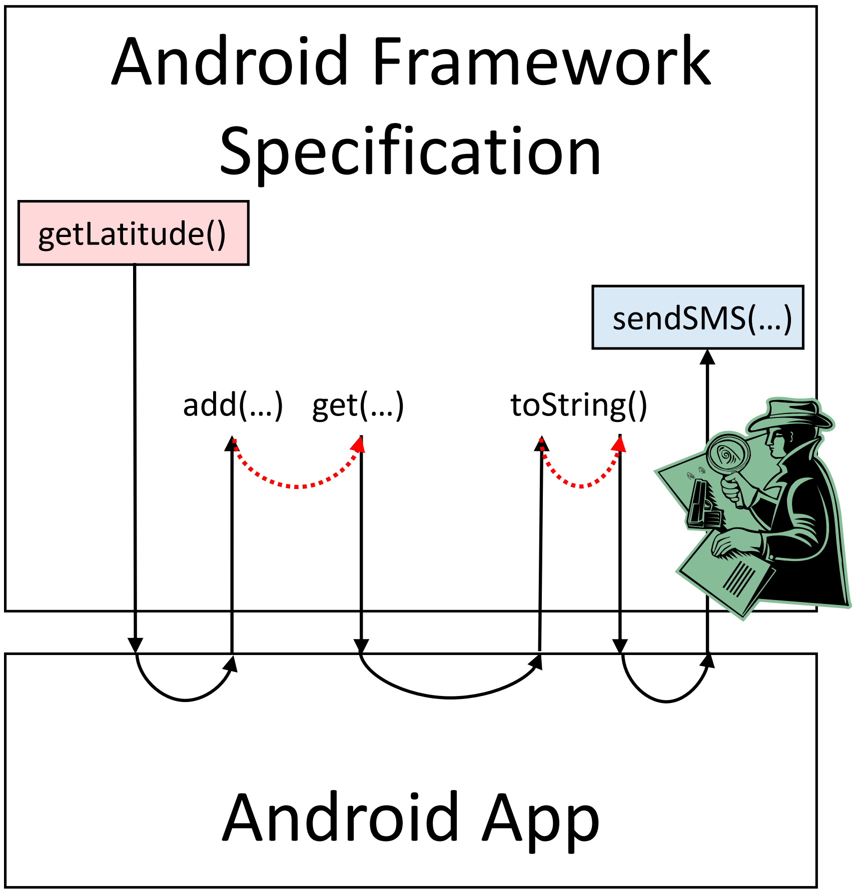
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

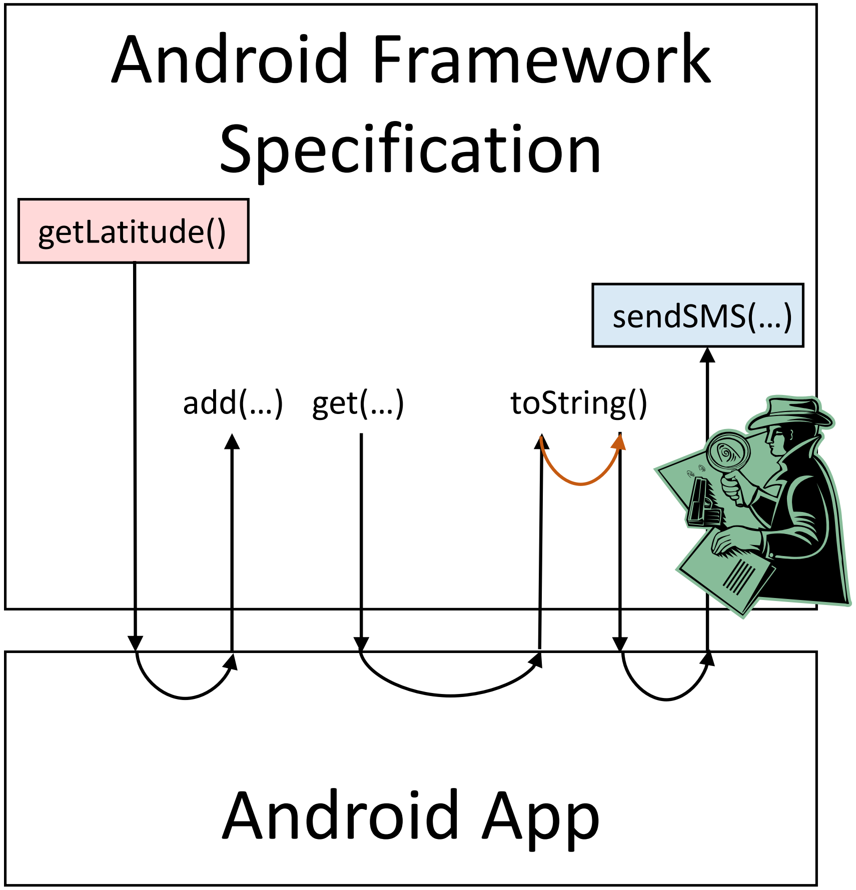
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

add(...)

get(...)

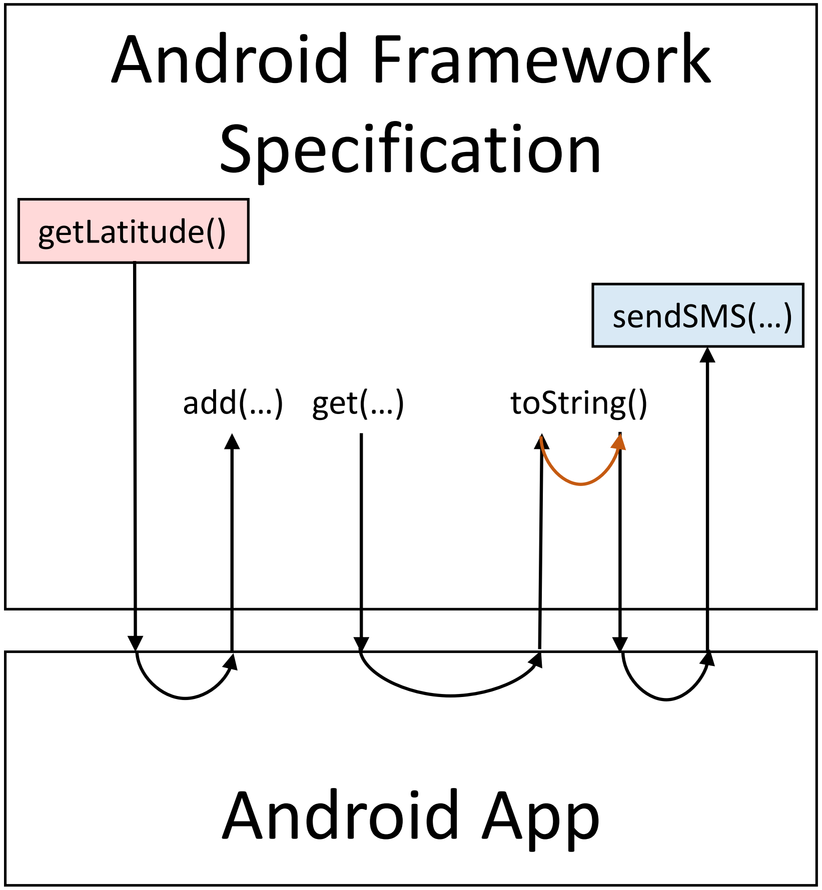
toString()

Android App

# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

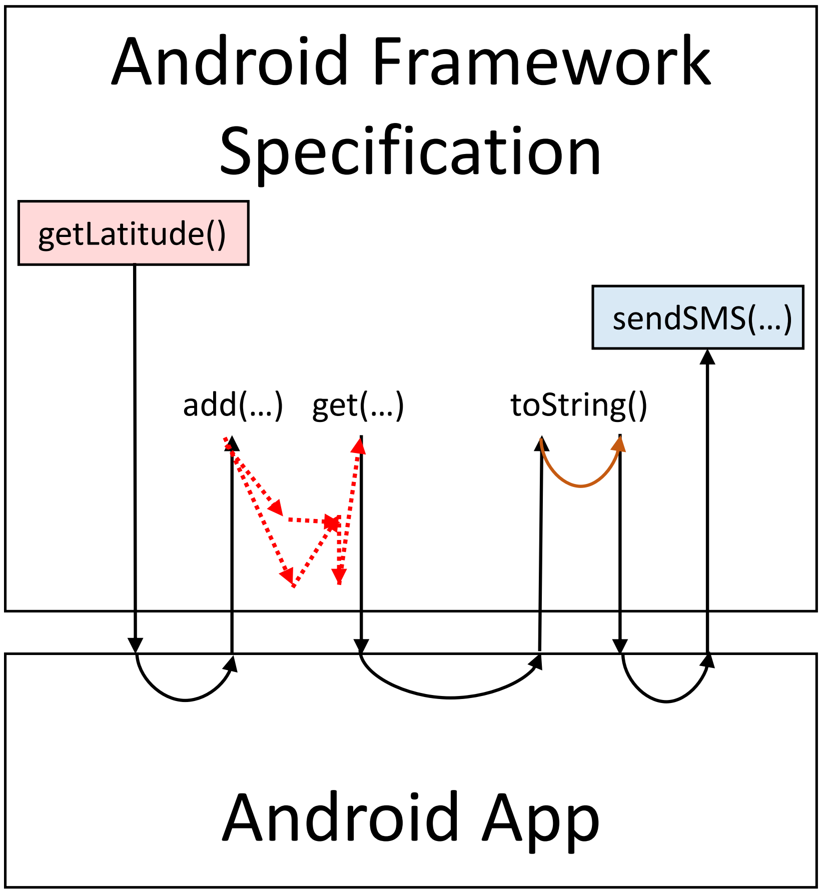
add(...) get(...) toString()

Android App

# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

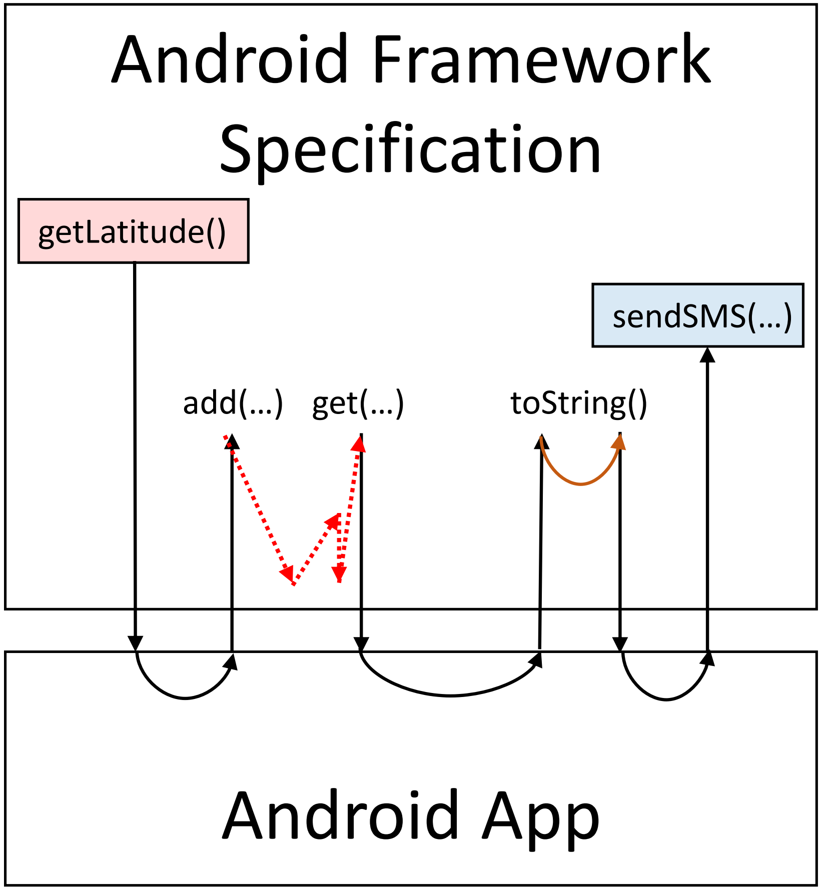
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(…)

add(…)

get(…)

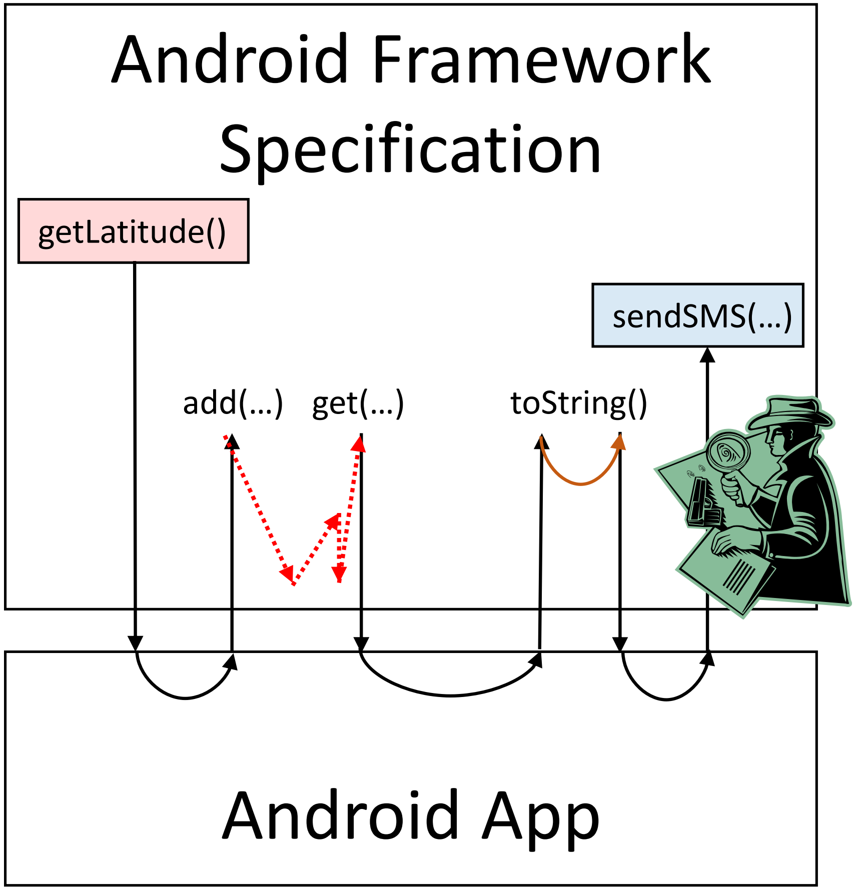
toString()

Android App

# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



Android Framework Specification

getLatitude()

sendSMS(...)

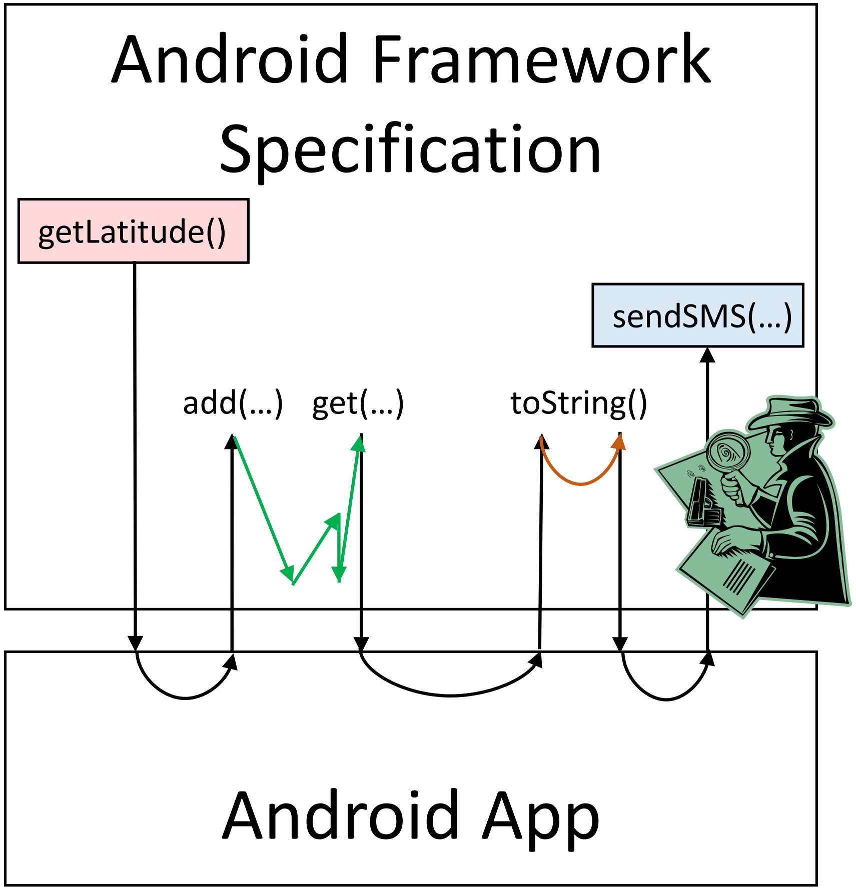
add(...) get(...) toString()

Android App

# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

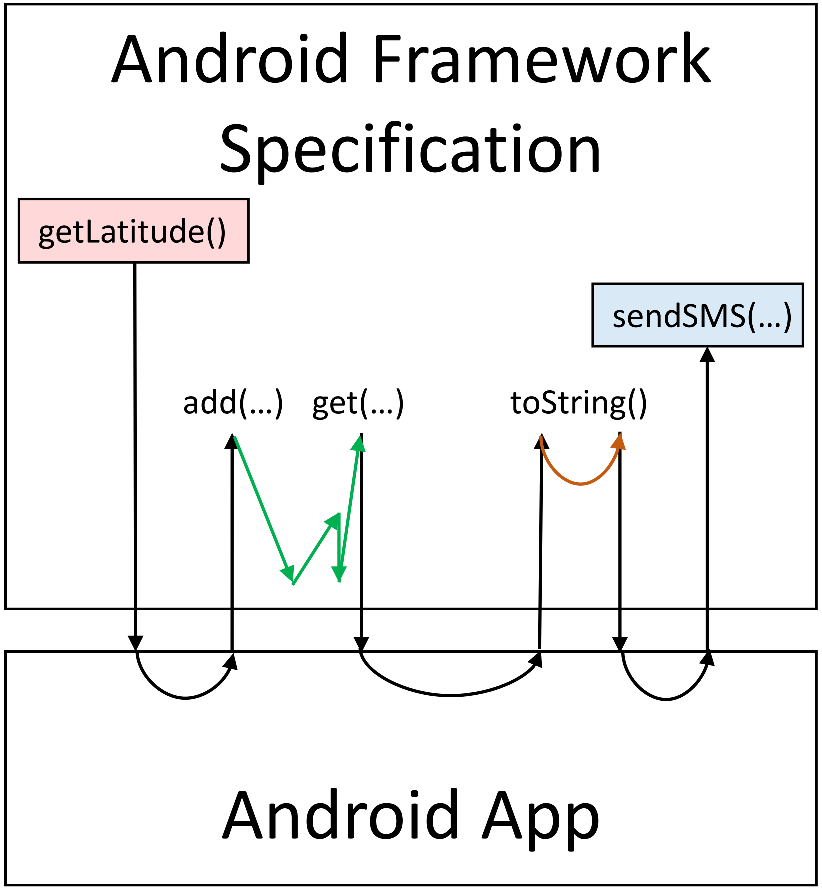
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



# Interactive Refinement

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

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



# Interactive Refinement

- Two problems to solve
  - Step 1: Worst-case analysis
  - Step 2: Specification inference

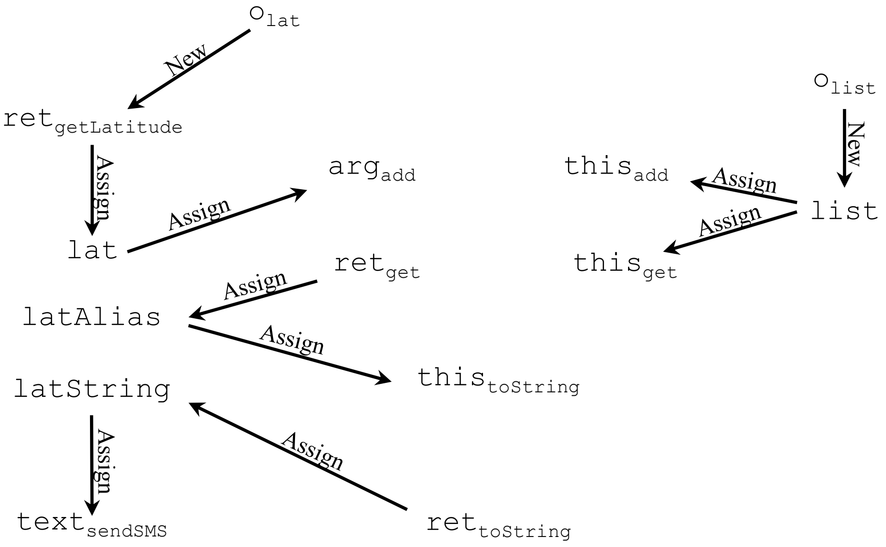
# CFL Reachability

# CFL Reachability

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

# CFL Reachability: Stage 1

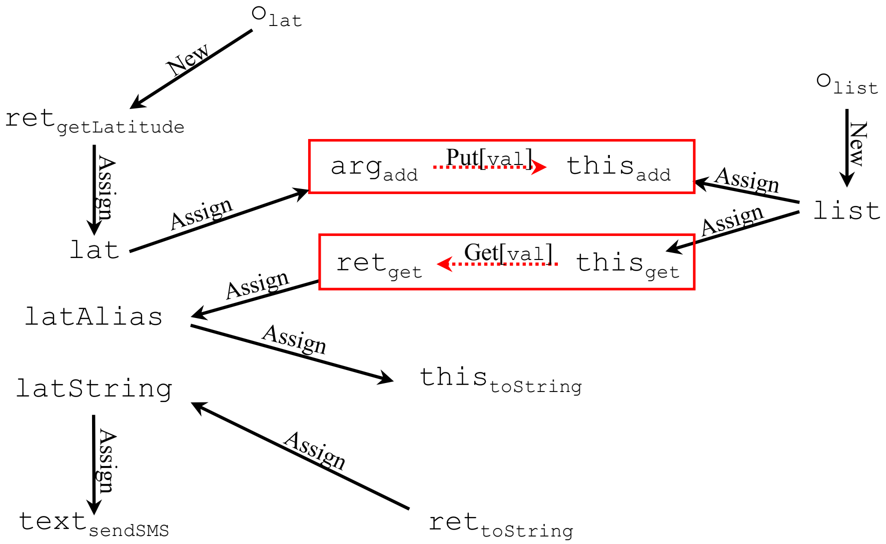
```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```



# CFL Reachability: Stage 1

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

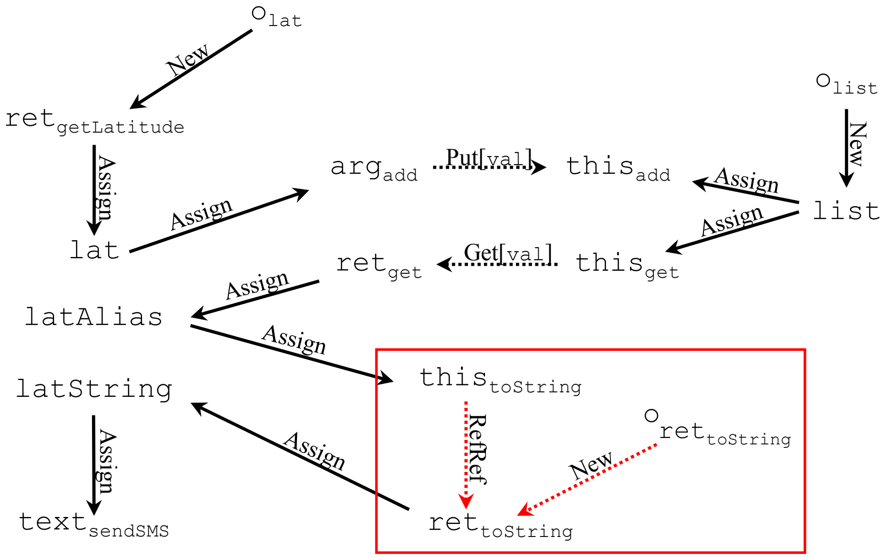
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
```



# CFL Reachability: Stage 1

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

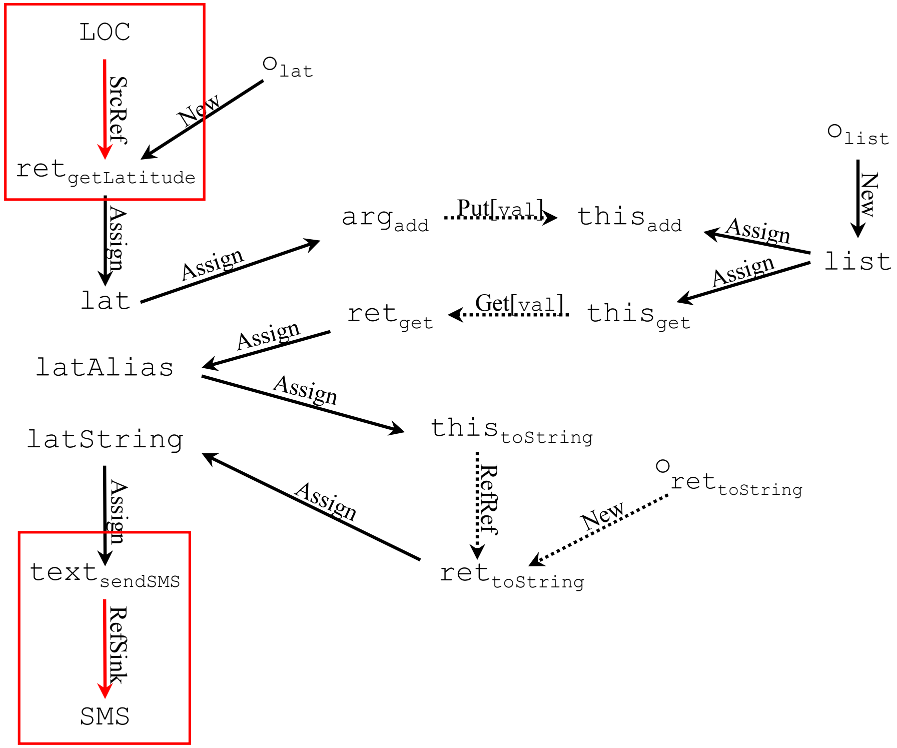
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
```



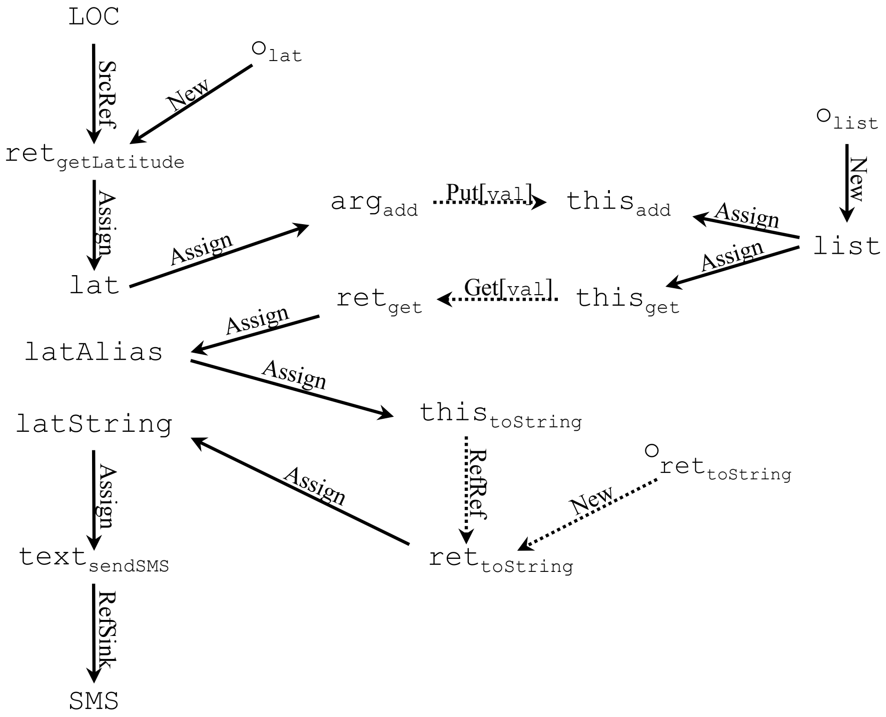
# CFL Reachability: Stage 1

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

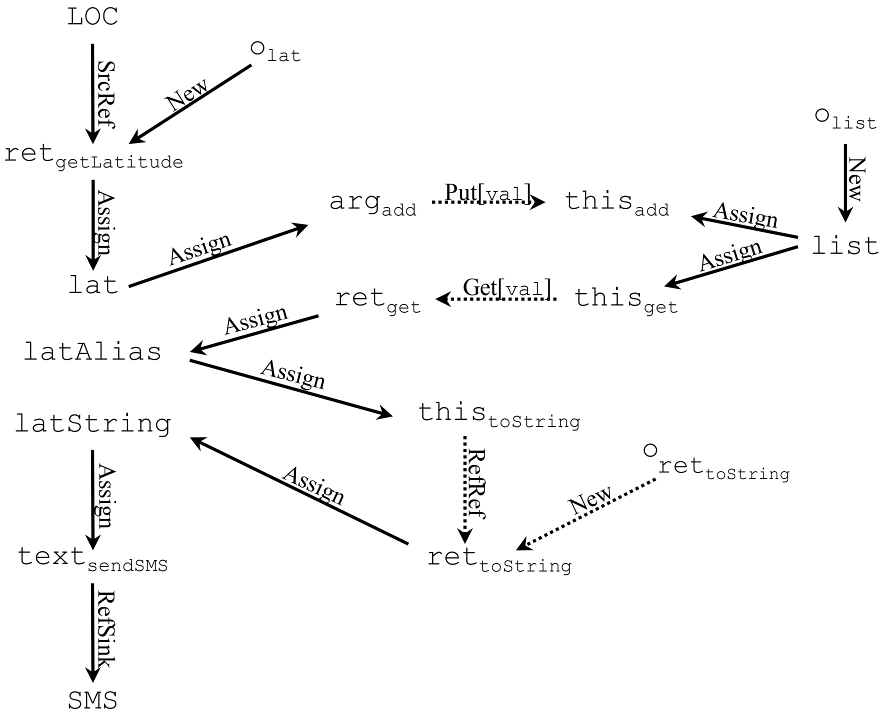
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



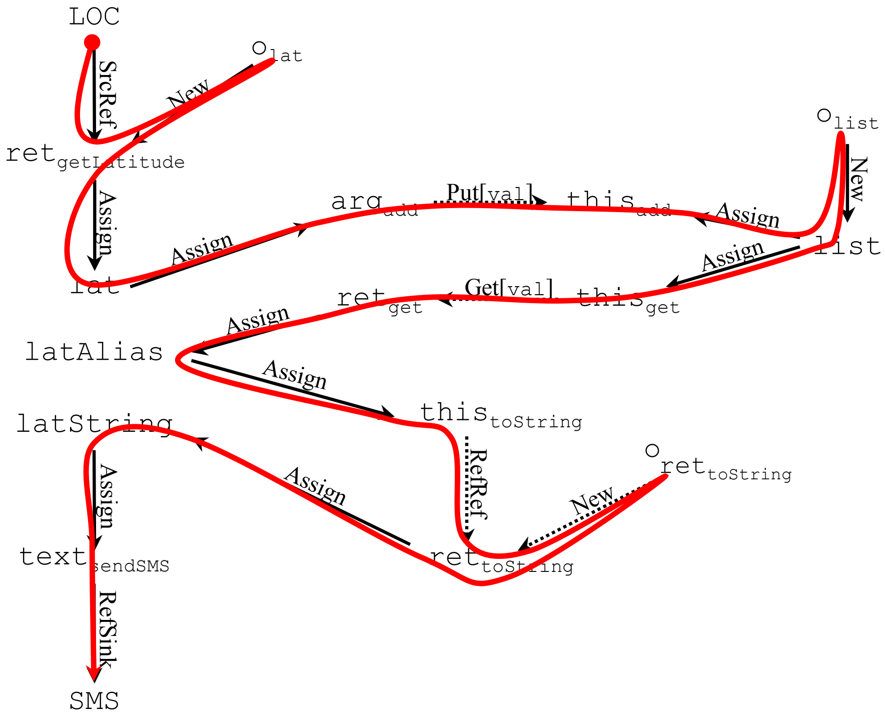
# CFL Reachability: Stage 2



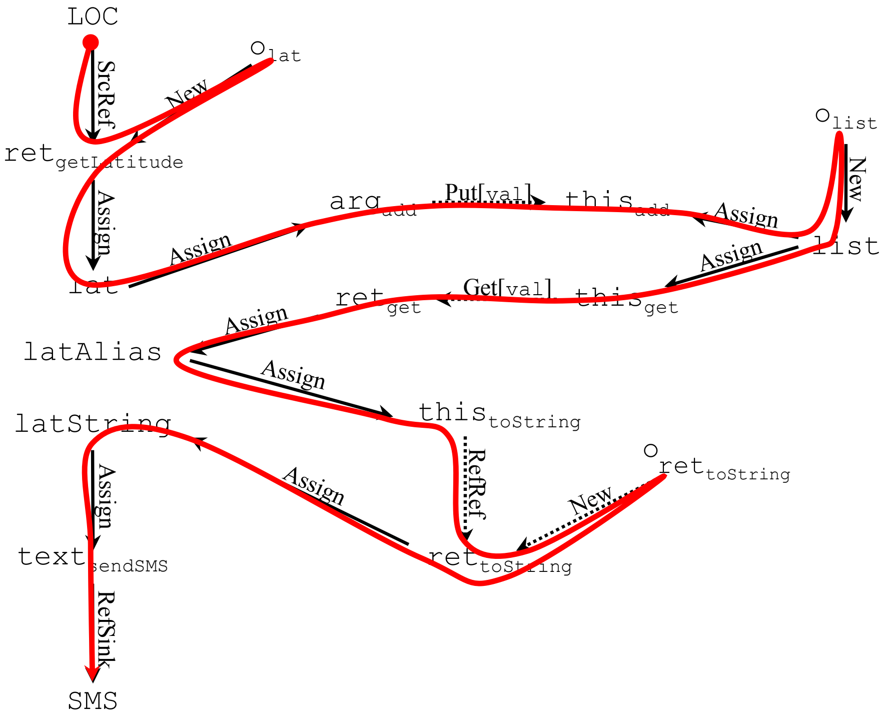
# CFL Reachability: Stage 2



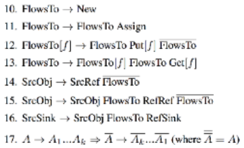
# CFL Reachability: Stage 2



# CFL Reachability: Stage 2



SrcRef $\overline{\text{New}}$ New Assign Assign Put[val] Assign $\overline{\text{New}}$ New Assign Get[Val] Assign Assign RefRef $\overline{\text{New}}$ New Assign Assign RefSink $\in L($



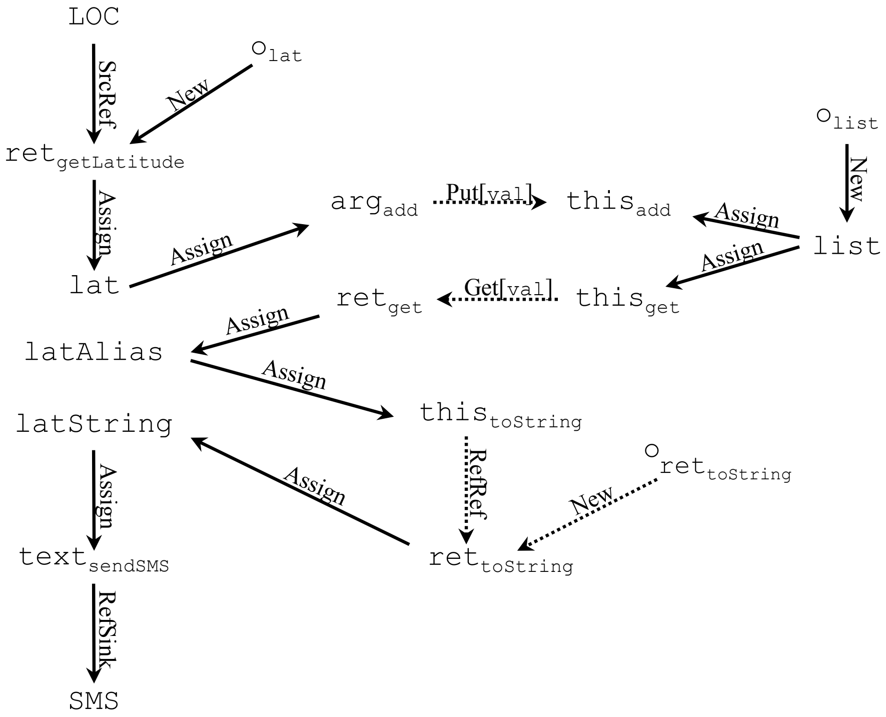
10. FlowsTo → New
11. FlowsTo → FlowsTo Assign
12. FlowsTo[$f$] → FlowsTo Put[$f$] $\overline{\text{FlowsTo}}$
13. FlowsTo → FlowsTo[$f$] FlowsTo Get[$f$]
14. SrcObj → SrcRef $\overline{\text{FlowsTo}}$
15. SrcObj → SrcObj FlowsTo RefRef $\overline{\text{FlowsTo}}$
16. SrcSink → SrcObj FlowsTo RefSink
17. $A \rightarrow A_1 \ldots A_k \Rightarrow \overline{A} \rightarrow \overline{A_k} \ldots \overline{A_1}$ (where $\overline{\overline{A}} = A$)

$)$

# Missing Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

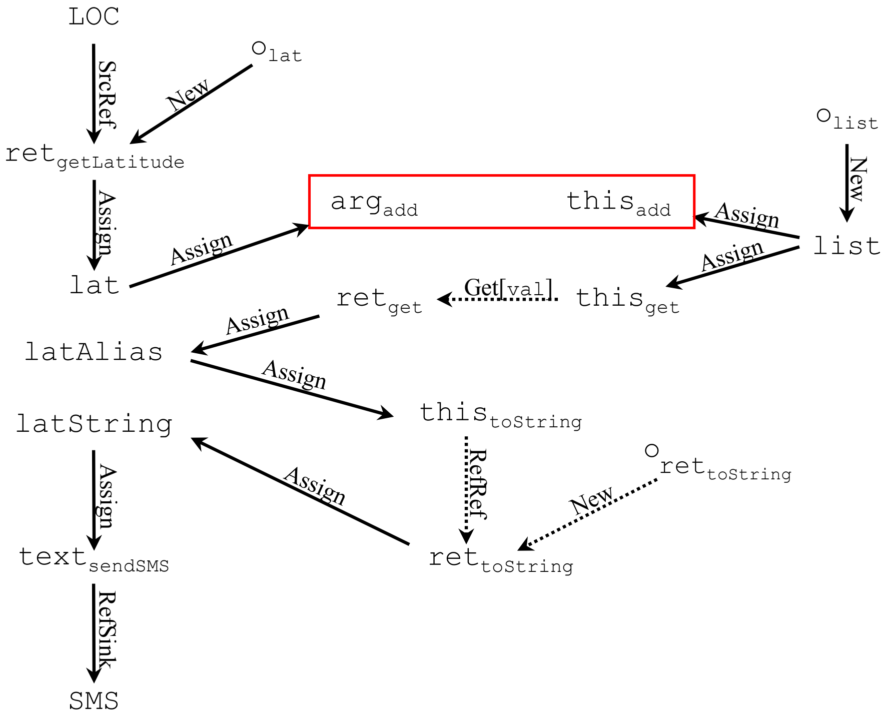
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



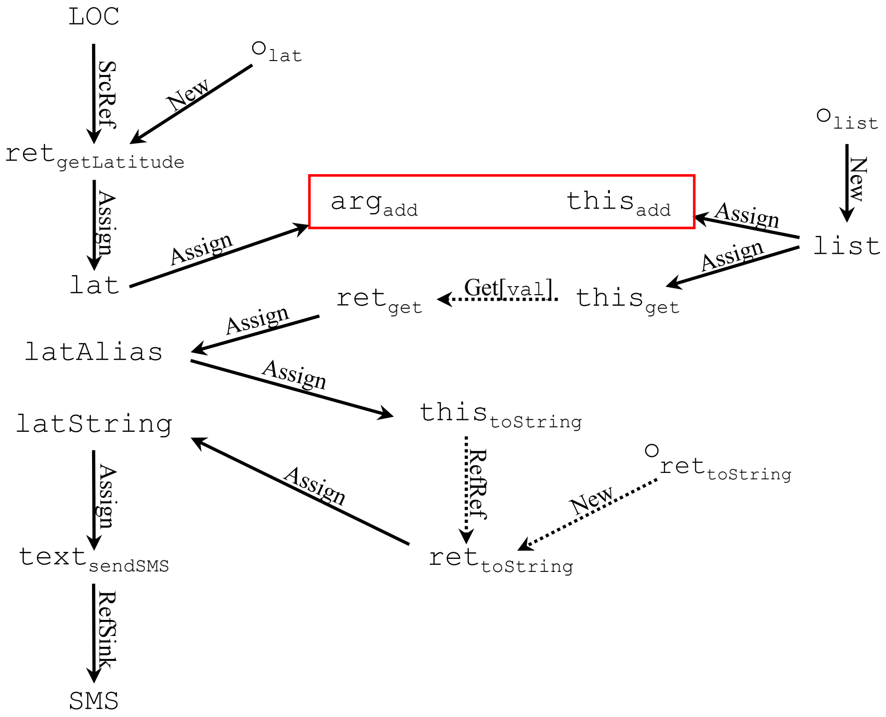
# Missing Specifications

```
Double lat = getLatitude();
List list = new List();
list.add(lat);
Double latAlias = list.get(0);
String latStr = latAlias.toString();
sendSMS(latStr);
```

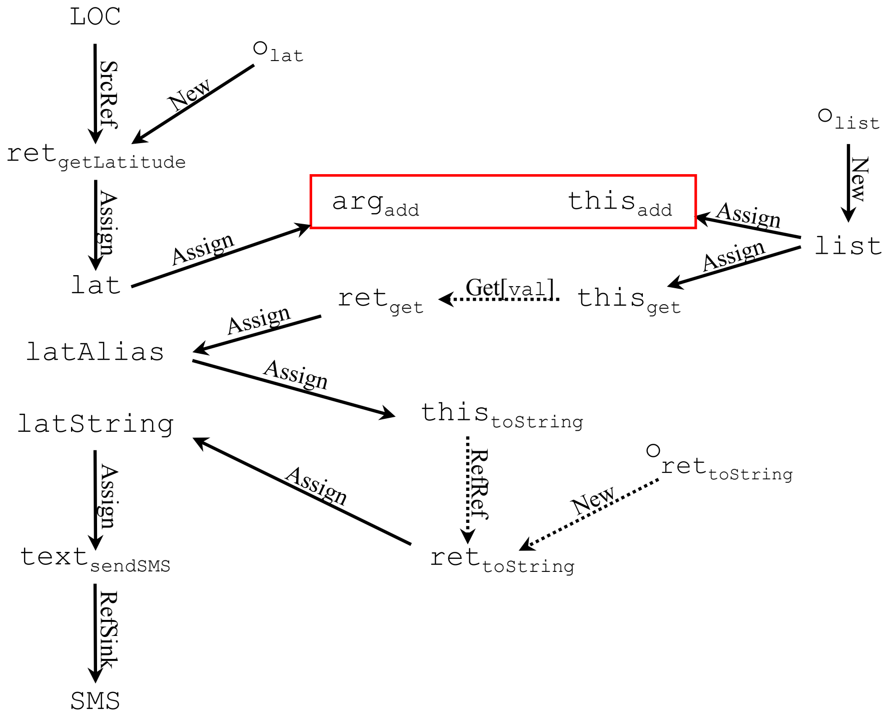
```
class List:
    @Alias(arg, this.val)
    void add(Object arg) {}
    @Alias(this.val, return)
    Object get(Integer index) {}
class Double:
    @Flow(this, return)
    String toString() {}
class LocationManager:
    @Flow(LOC, return)
    static String getLatitude() {}
class SMS:
    @Flow(text,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



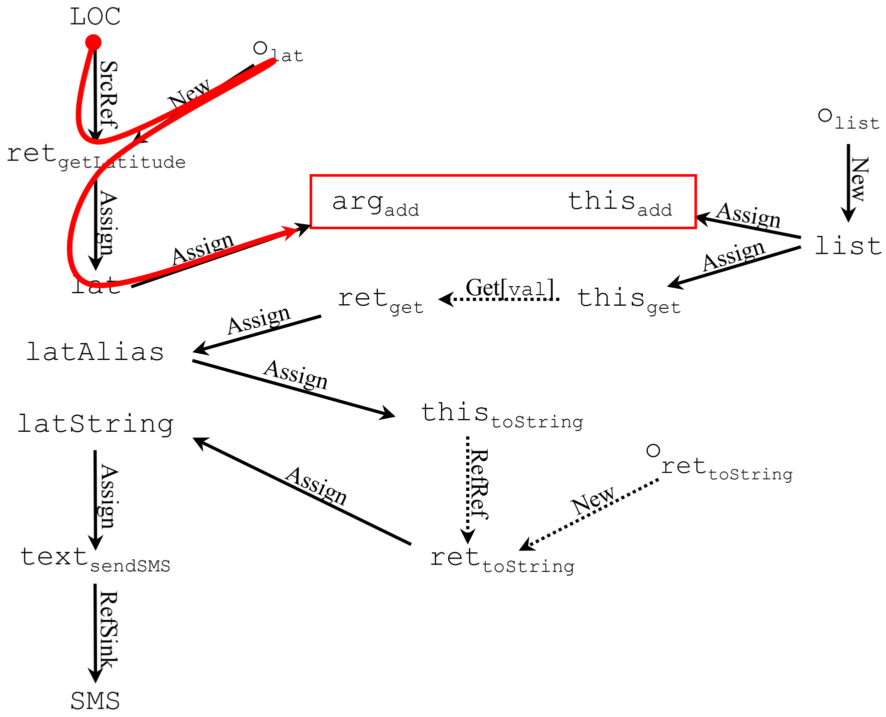
# Missing Specifications



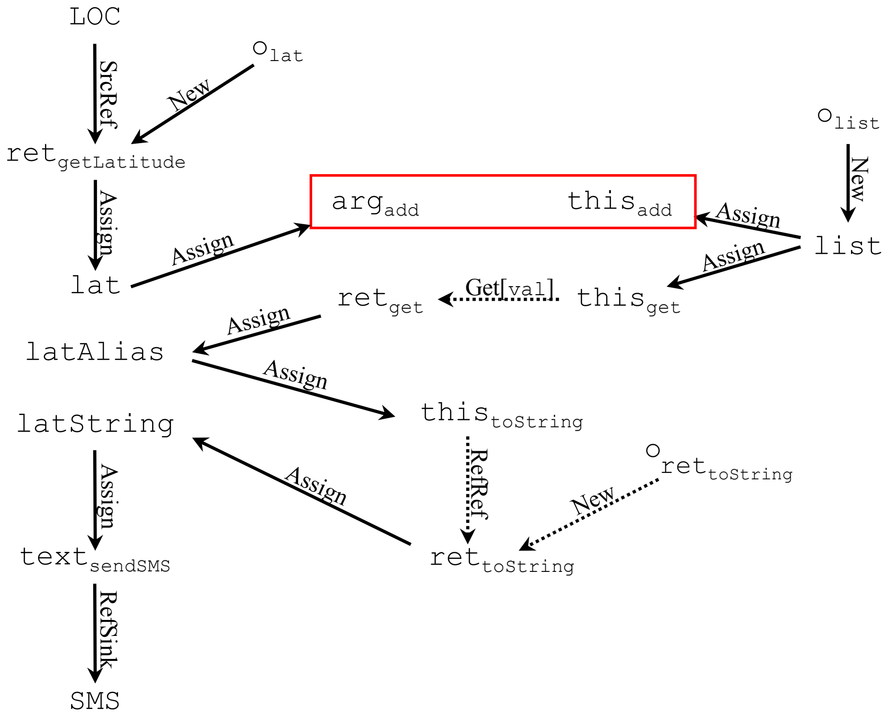
# Missing Specifications



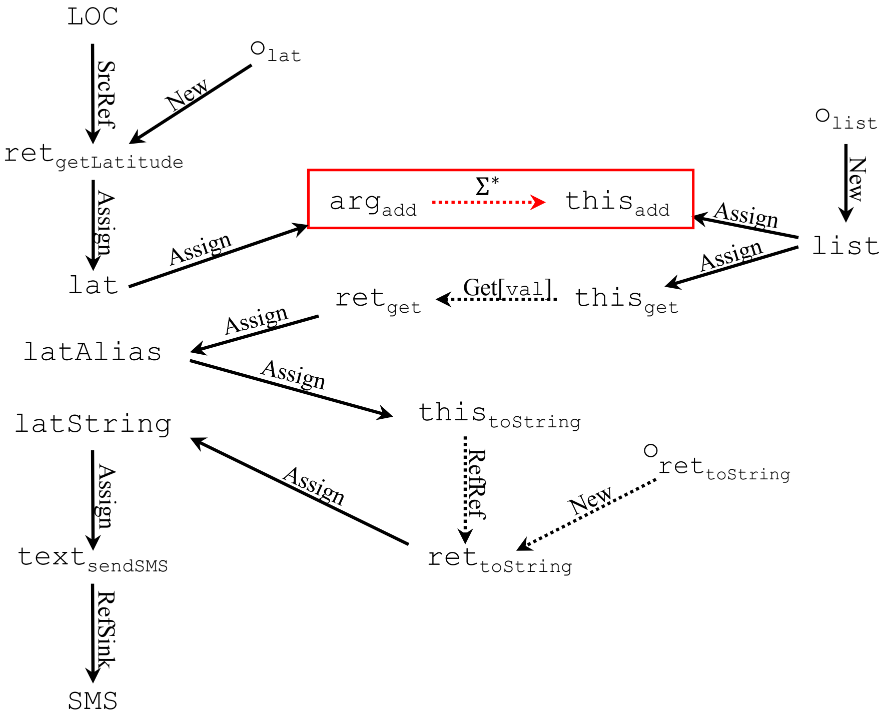
# Missing Specifications



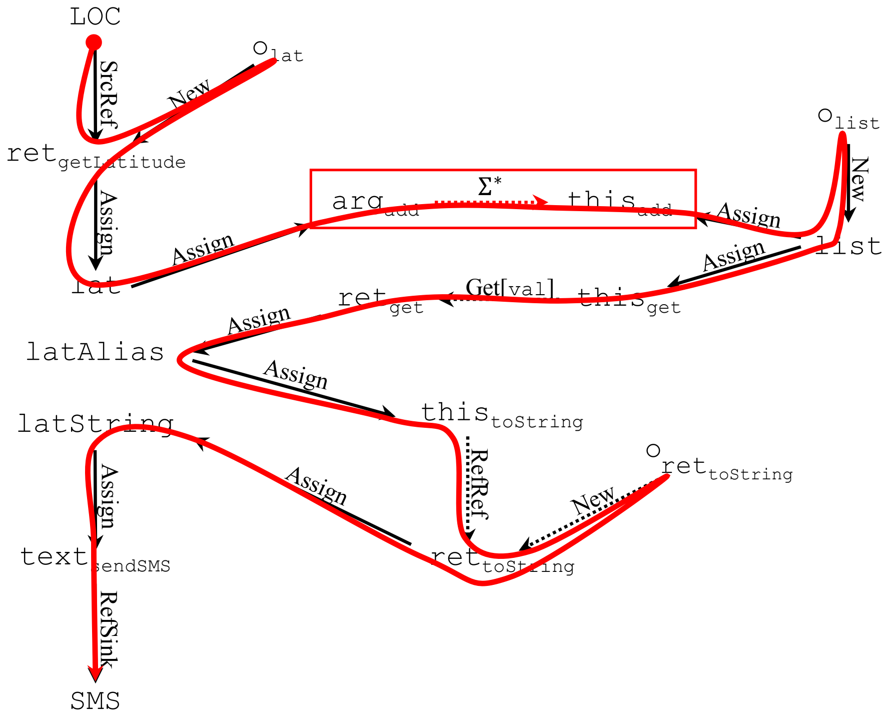
# Step 1: Worst-Case Analysis



# Step 1: Worst-Case Analysis



# Step 1: Worst-Case Analysis



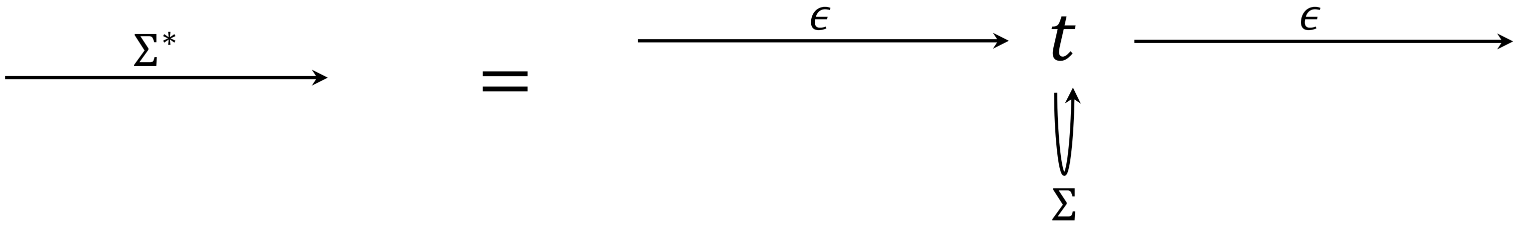
SrcRef $\overline{\text{New}}$ New Assign Assign $(\Sigma^* = \text{Put[val]})$ Assign $\overline{\text{New}}$ New Assign Get[Val] Assign Assign RefRef $\overline{\text{New}}$ New Assign Assign RefSink $\in L($

10. FlowsTo → New
11. FlowsTo → FlowsTo Assign
12. $\text{FlowsTo}[f] \rightarrow \text{FlowsTo Put}[f]\ \overline{\text{FlowsTo}}$
13. $\text{FlowsTo} \rightarrow \text{FlowsTo}[f]\ \text{FlowsTo Get}[f]$
14. $\text{SrcObj} \rightarrow \text{SrcRef}\ \overline{\text{FlowsTo}}$
15. $\text{SrcObj} \rightarrow \text{SrcObj FlowsTo RefRef}\ \overline{\text{FlowsTo}}$
16. SrcSink → SrcObj FlowsTo RefSink
17. $A \rightarrow A_1 \ldots A_k \Rightarrow \overline{A} \rightarrow \overline{A_k} \ldots \overline{A_1}$ (where $\overline{\overline{A}} = A$)

$)$

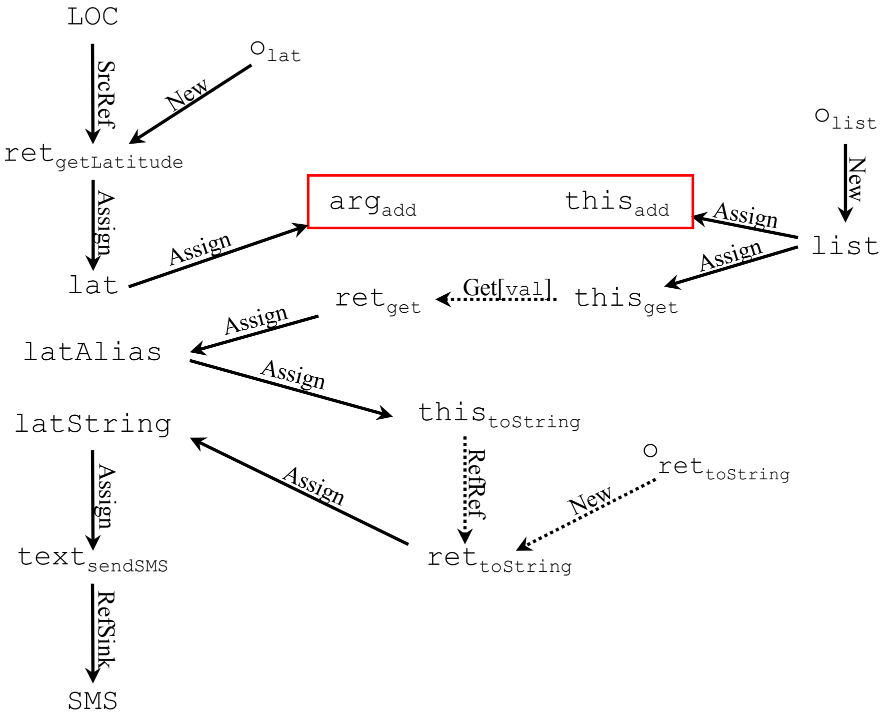
# Step 1: Worst-Case Analysis

- Use “do anything” subgraph:

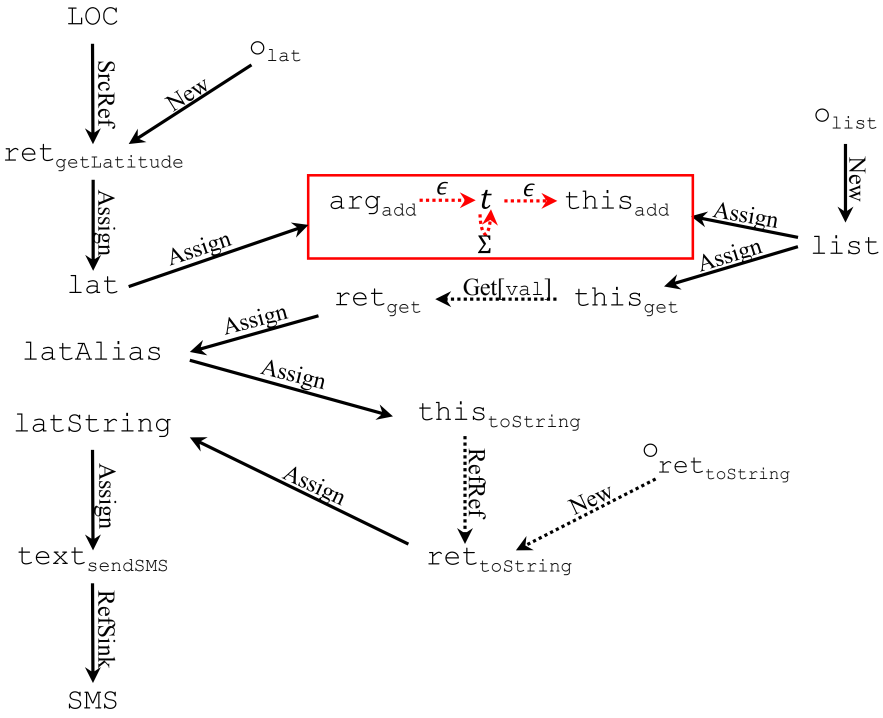


- Finite state automata that accepts $\Sigma^*$

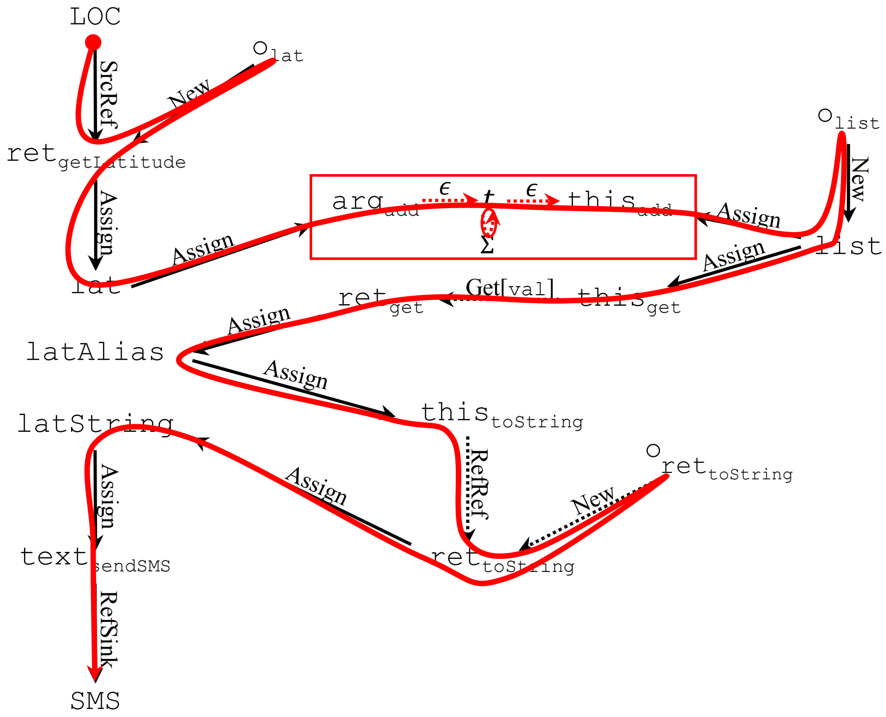
# Step 1: Worst-Case Analysis



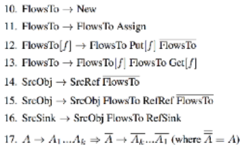
# Step 1: Worst-Case Analysis



# Step 1: Worst-Case Analysis



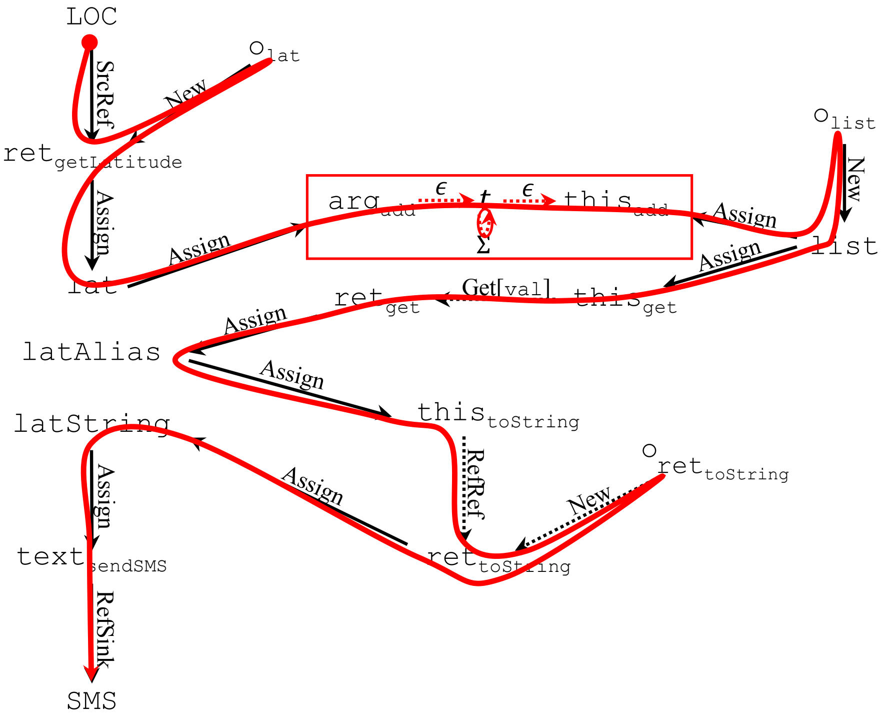
SrcRef $\overline{\text{New}}$ New Assign Assign $\epsilon$ Put[val] $\epsilon$ Assign $\overline{\text{New}}$ New Assign
Get[Val] Assign Assign RefRef $\overline{\text{New}}$ New Assign Assign RefSink $\in L($



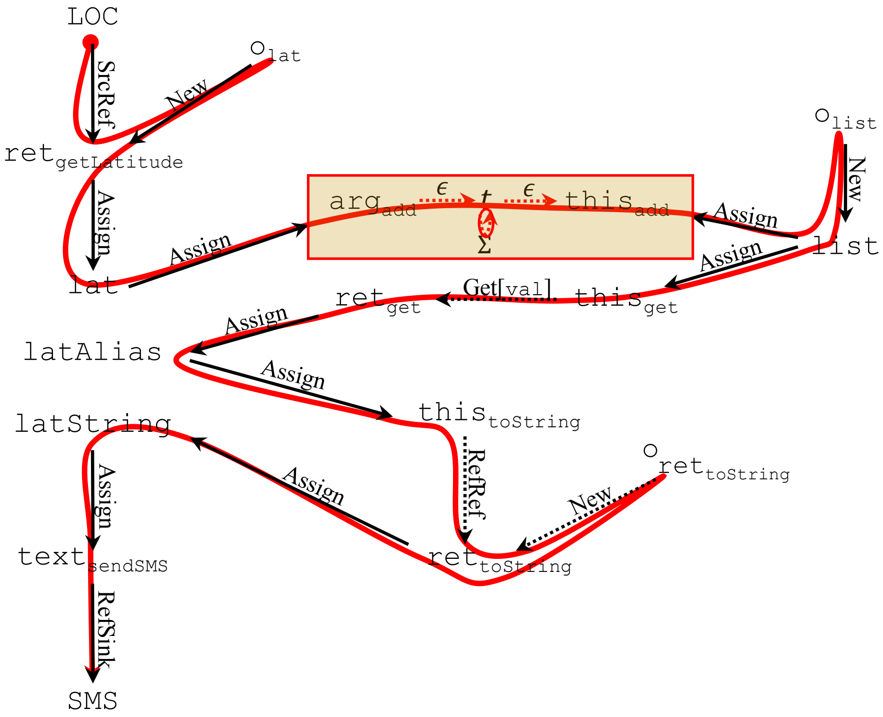
10. FlowsTo → New
11. FlowsTo → FlowsTo Assign
12. FlowsTo[$f$] → FlowsTo Put[$f$] $\overline{\text{FlowsTo}}$
13. FlowsTo → FlowsTo[$f$] FlowsTo Get[$f$]
14. SrcObj → SrcRef $\overline{\text{FlowsTo}}$
15. SrcObj → SrcObj FlowsTo RefRef $\overline{\text{FlowsTo}}$
16. SrcSink → SrcObj FlowsTo RefSink
17. $A \to A_1 \ldots A_k \Rightarrow \overline{A} \to \overline{A_k} \ldots \overline{A_1}$ (where $\overline{\overline{A}} = A$)

$)$

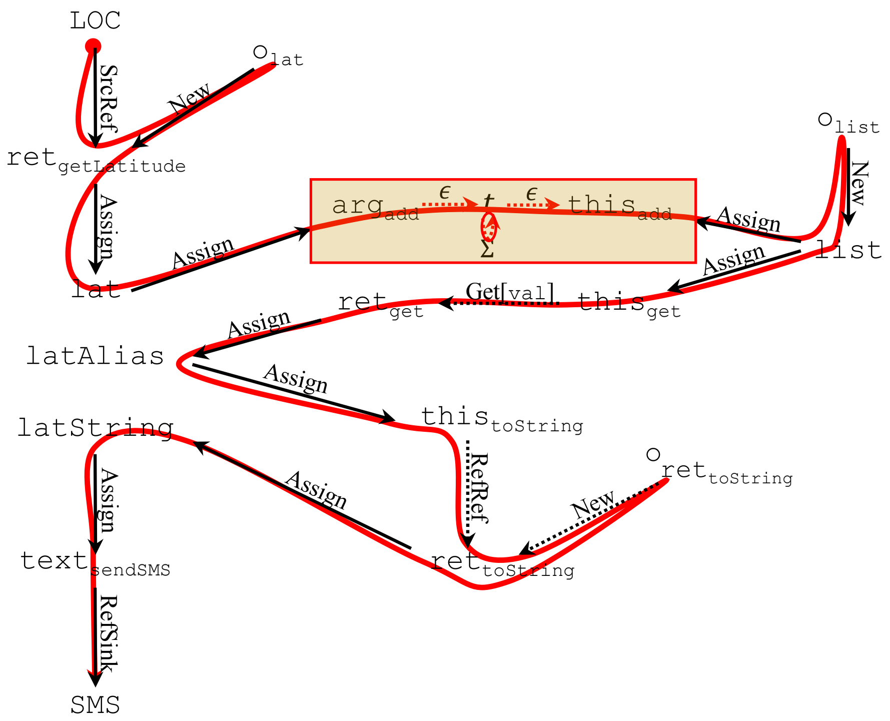
# Step 2: Specification Inference



# Step 2: Specification Inference



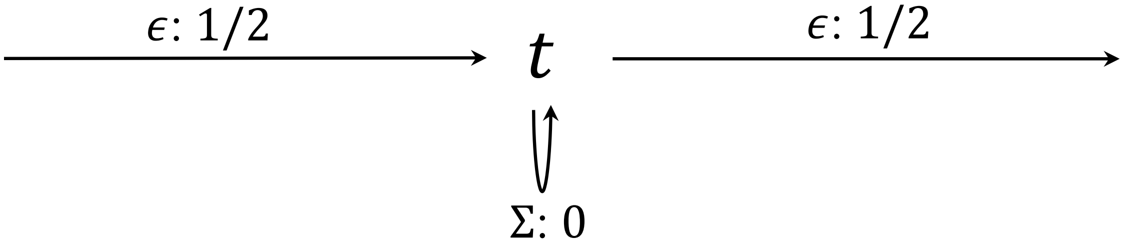
# Step 2: Specification Inference



**How do we ensure there are no paths passing through fewer missing specifications?**

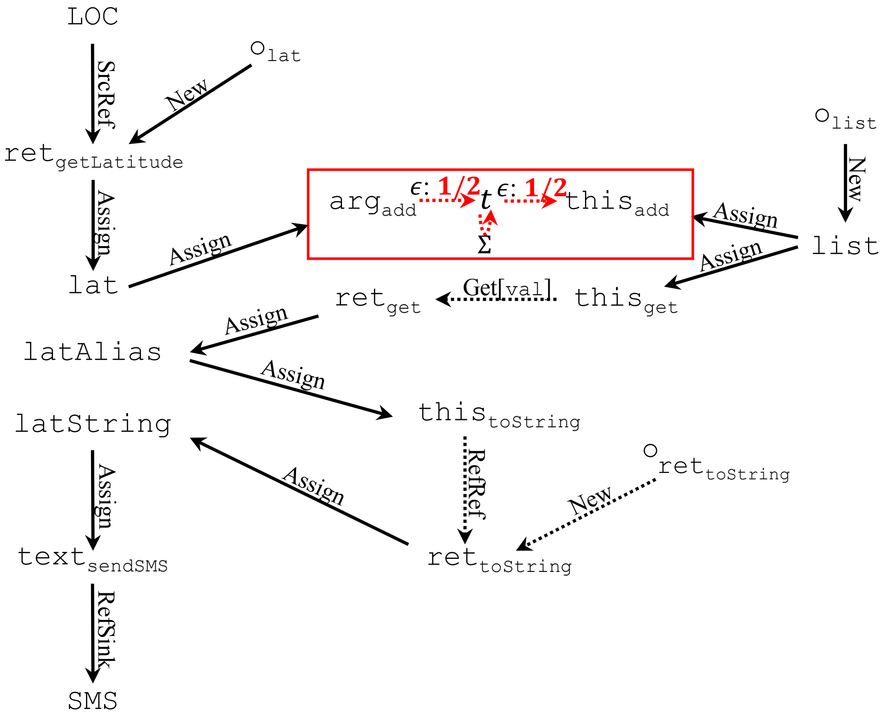
# Step 2: Specification Inference

- **Idea:** use ***shortest path*** CFL reachability

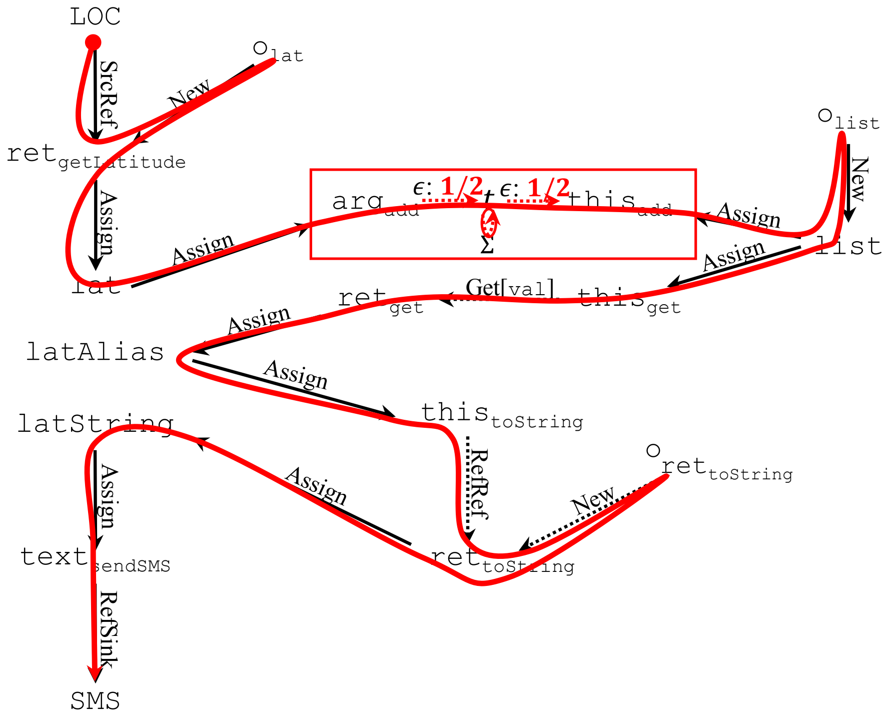


- Other edges have weight 0

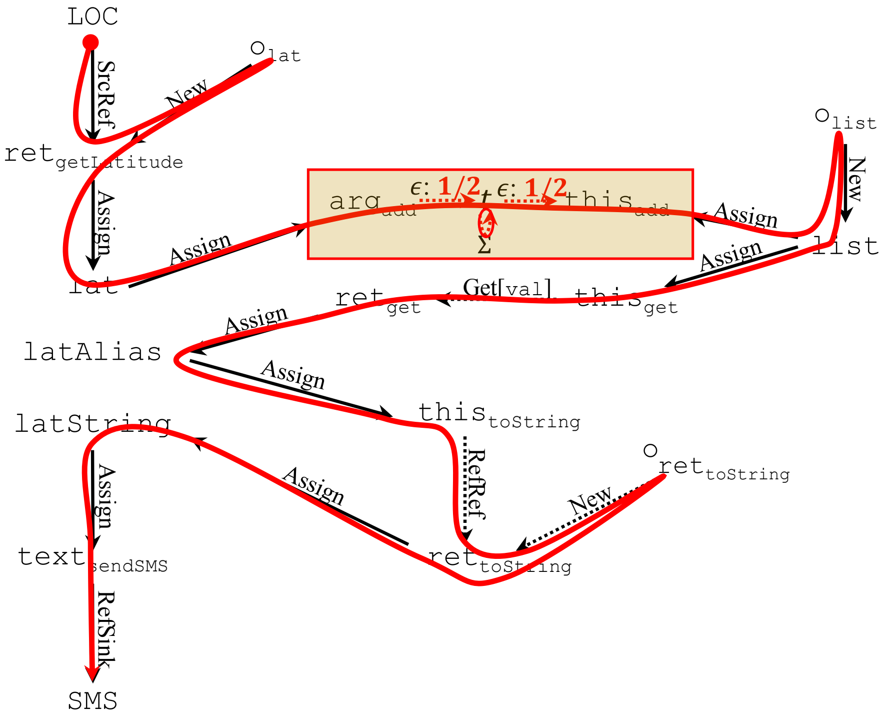
# Step 2: Specification Inference



# Step 2: Specification Inference



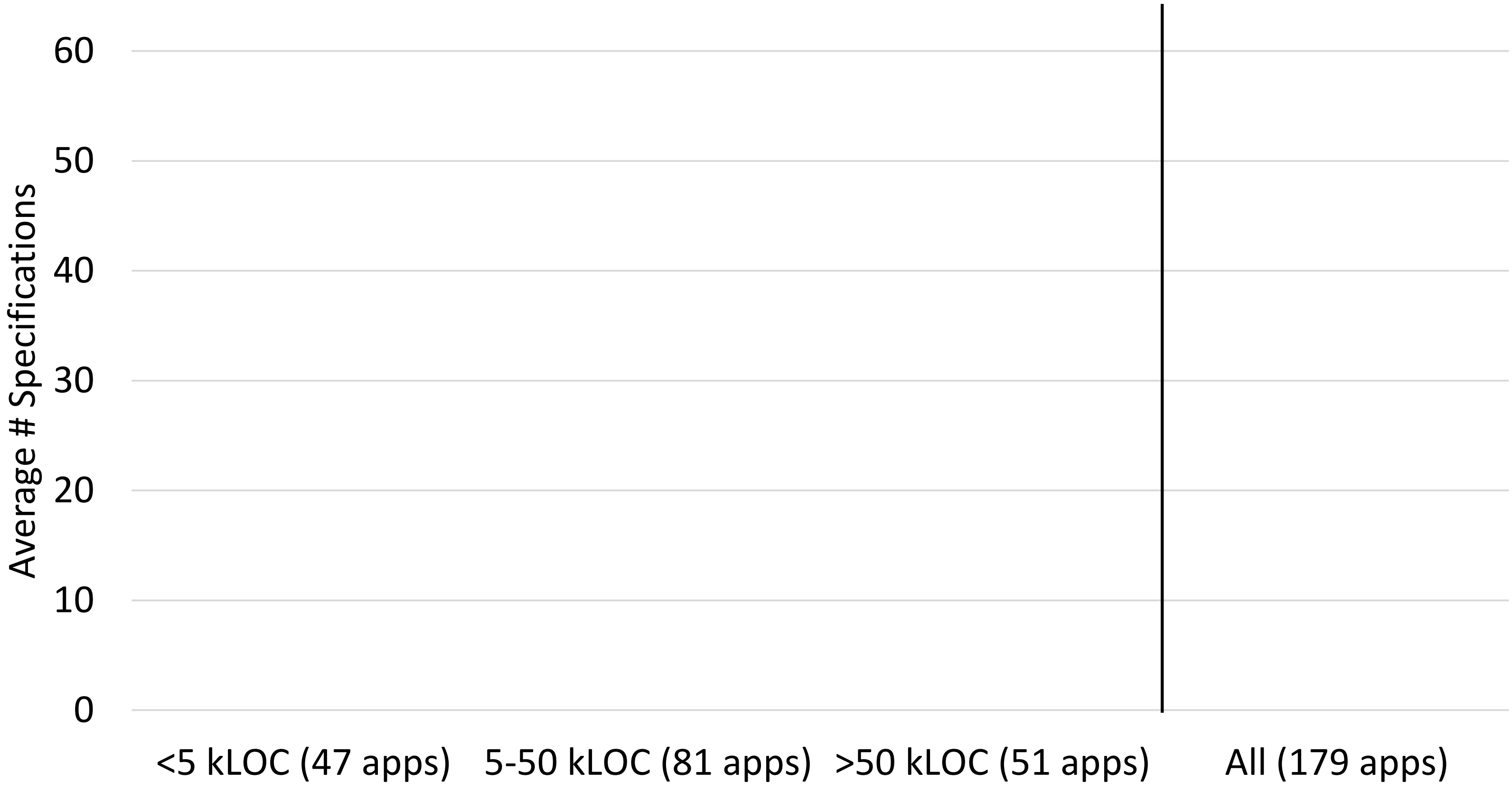
# Step 2: Specification Inference



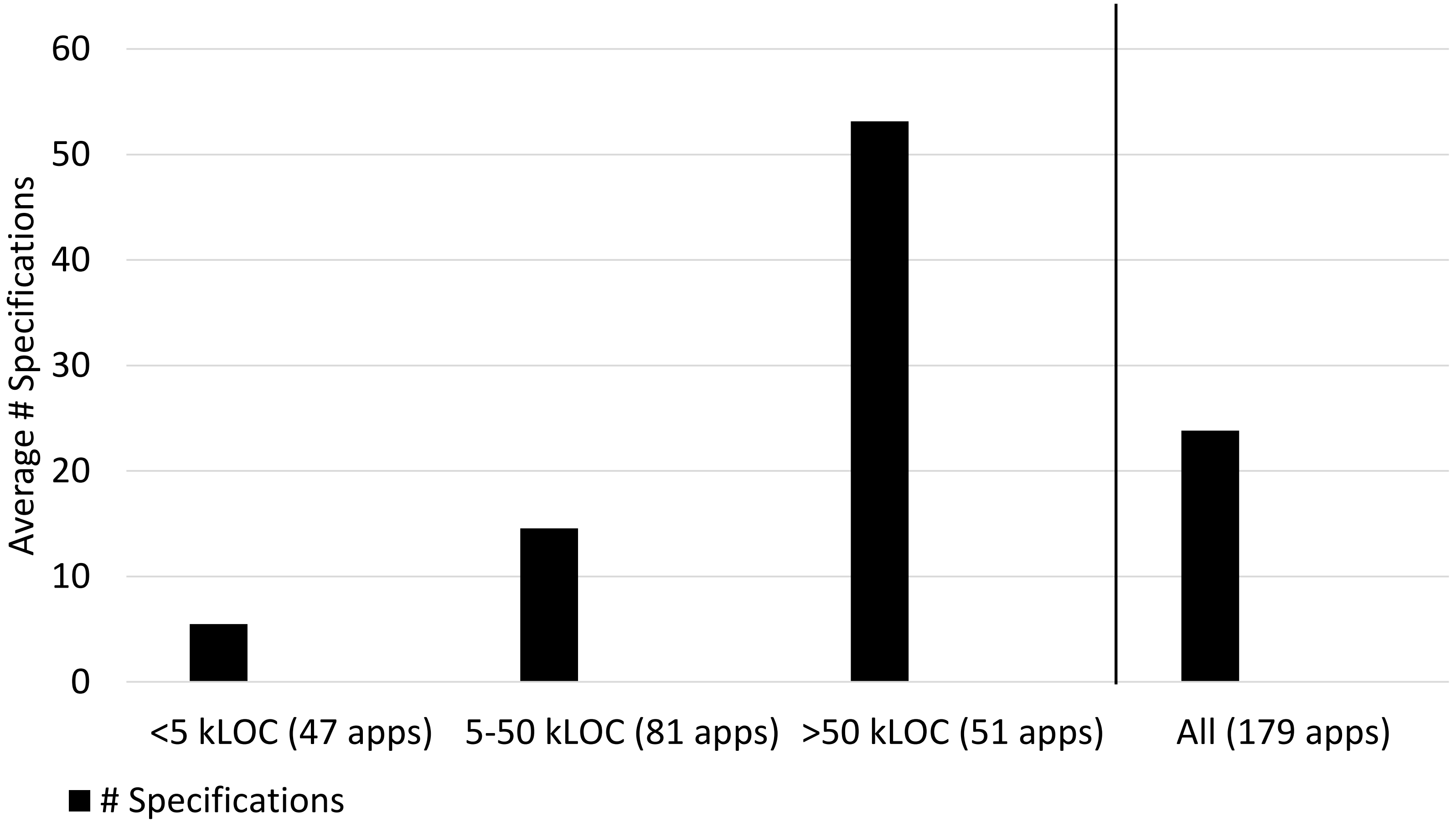
# Experiments

- 179 apps from Symantec, Google Play, and Darpa
- Flow specifications
  - Ran on all 179 apps
- Alias specifications
  - Type filters (points-to edges satisfy type constraints)
  - Ran on 156 apps

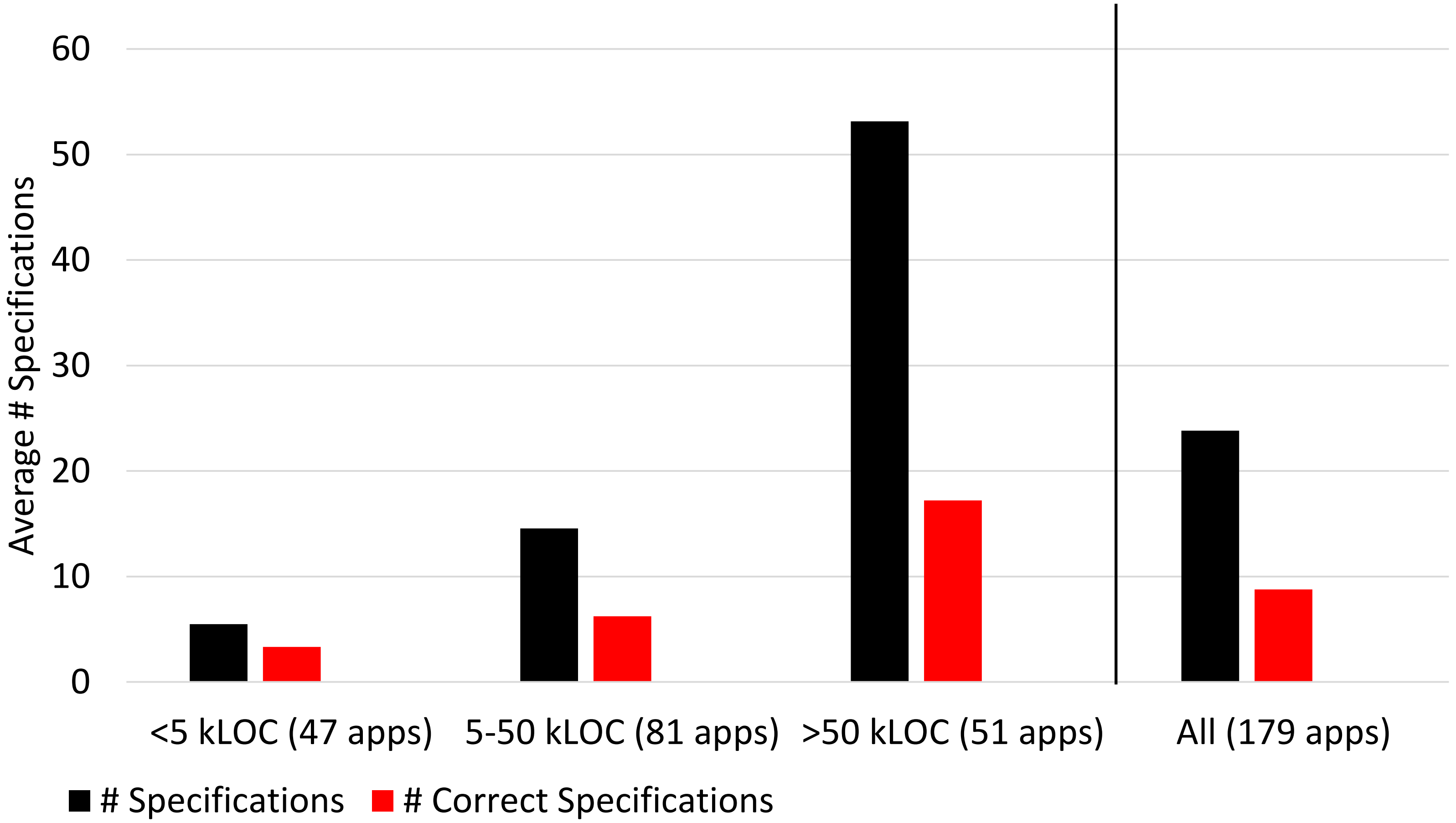
# Flow Specifications Inferred



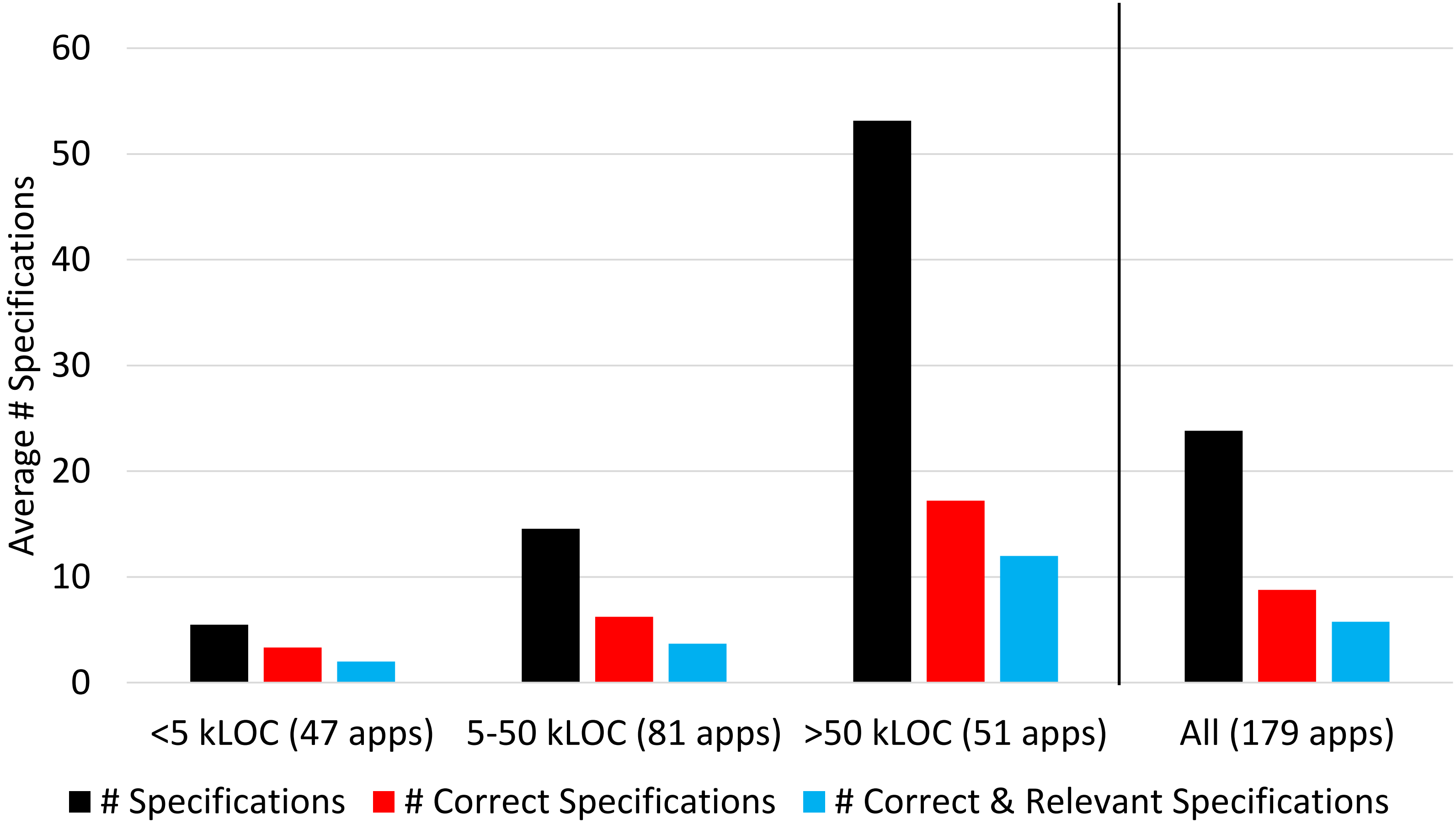
# Flow Specifications Inferred



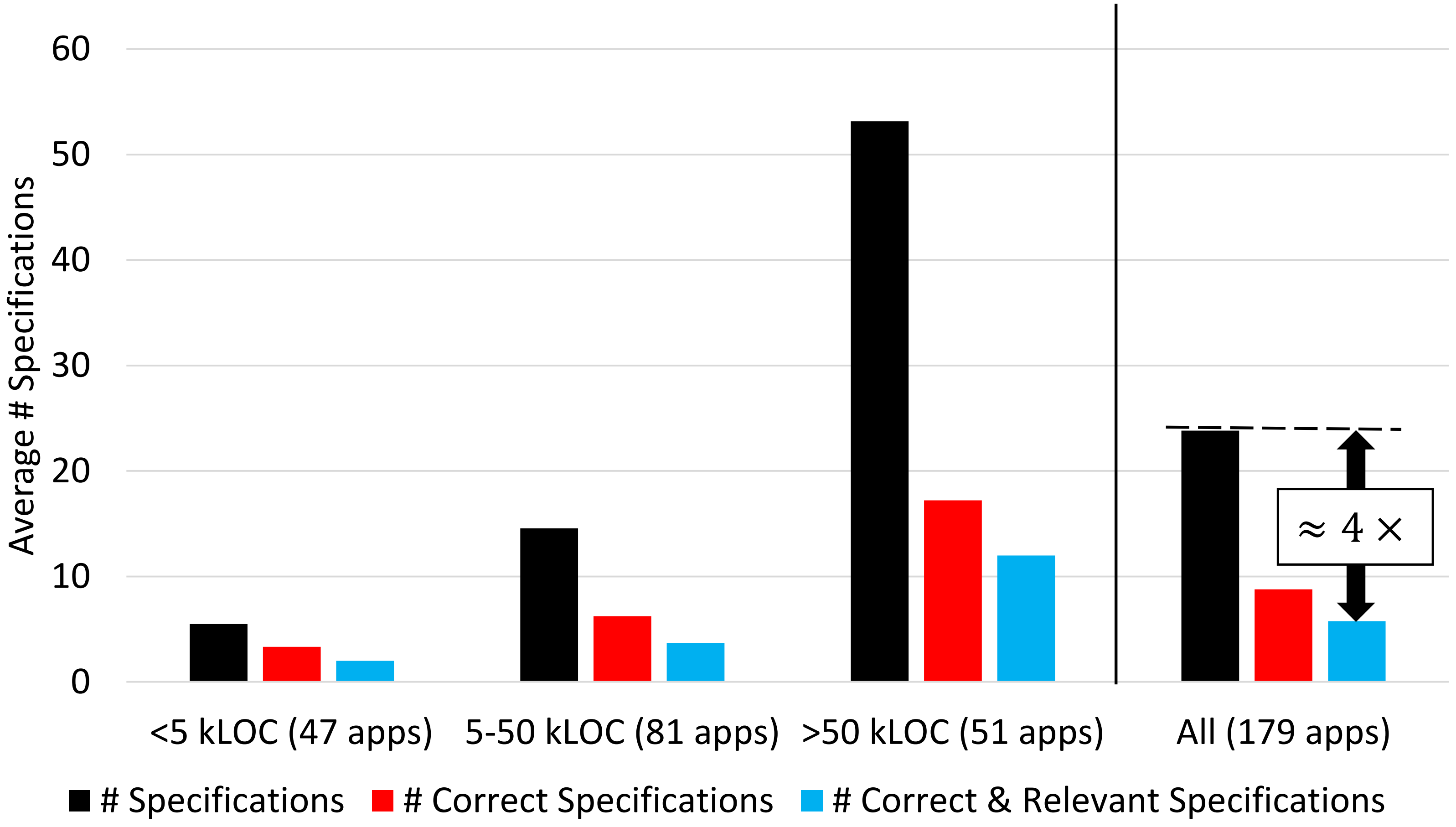
# Flow Specifications Inferred



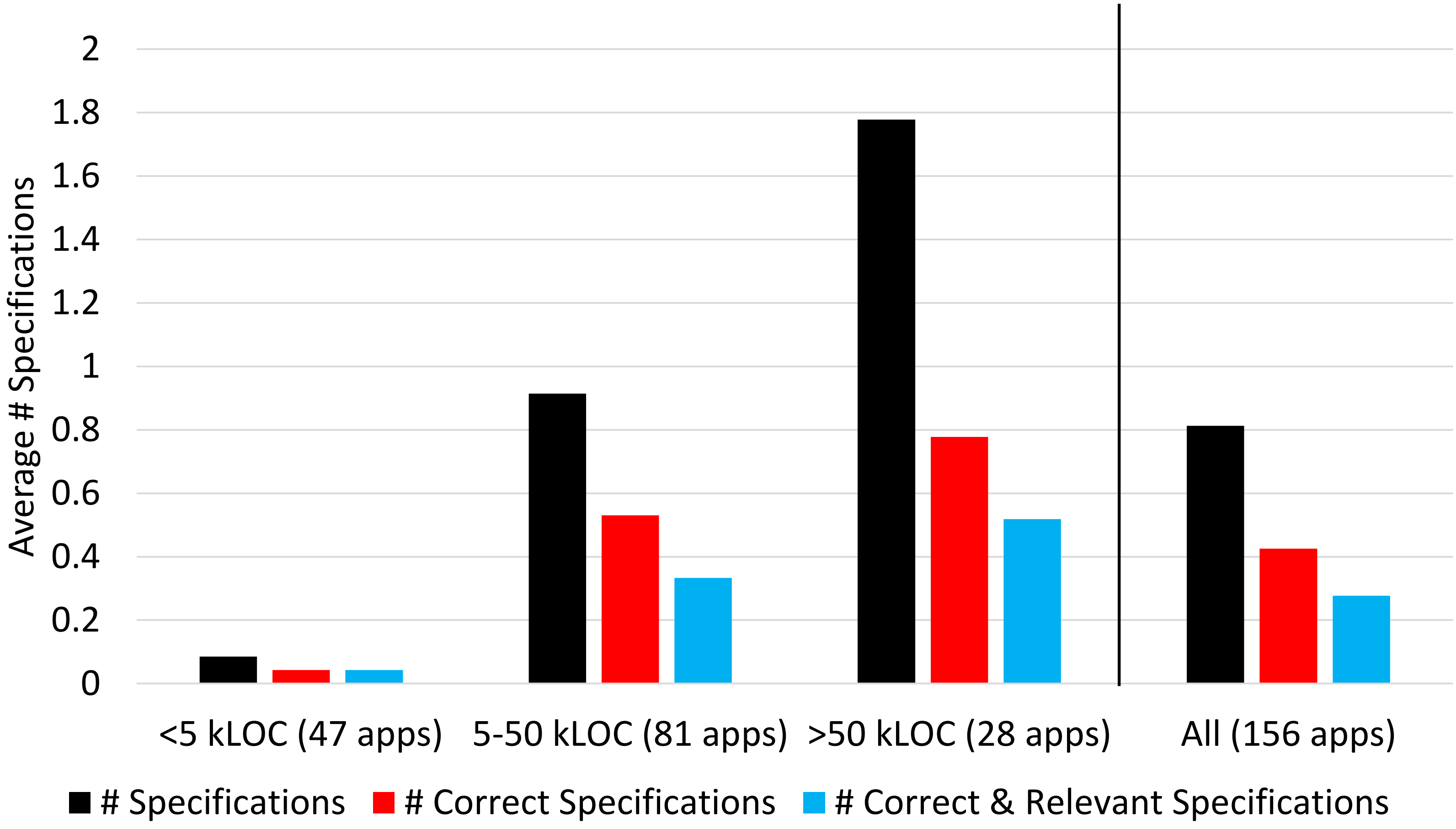
# Flow Specifications Inferred



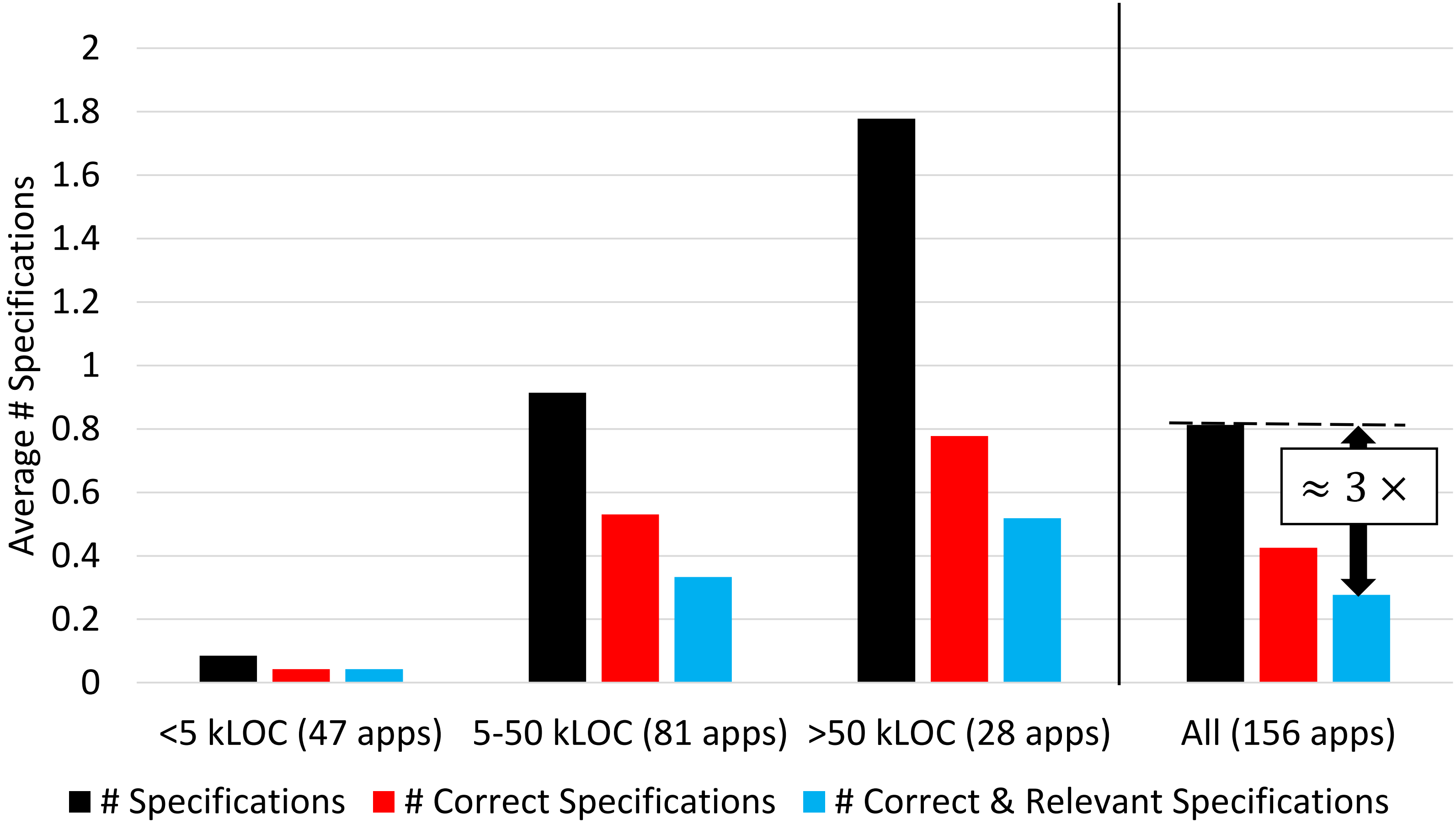
# Flow Specifications Inferred



# Alias Specifications Inferred



# Alias Specifications Inferred

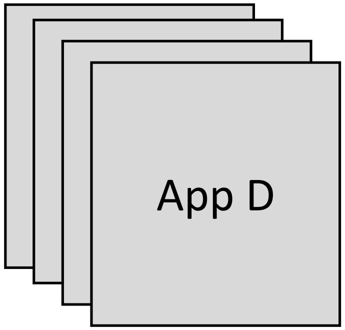


# Benefits of Aggregation

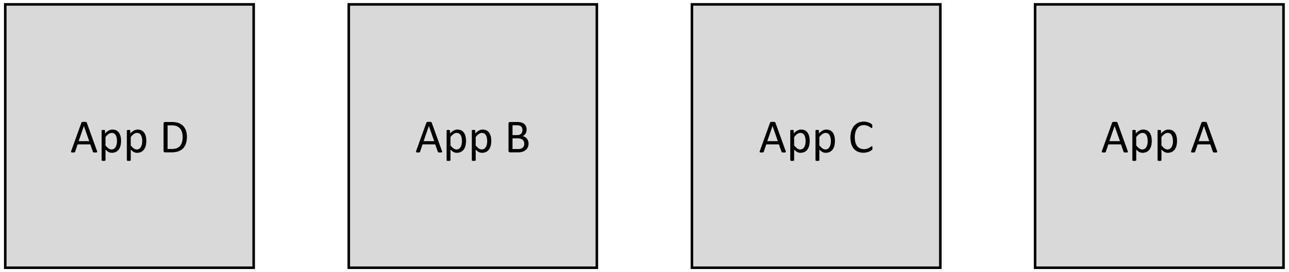
- **Hypothesis:** Specifications frequently reused
- **Idea:** Aggregate specifications across apps

# Benefits of Aggregation

# Benefits of Aggregation



# Benefits of Aggregation



App D

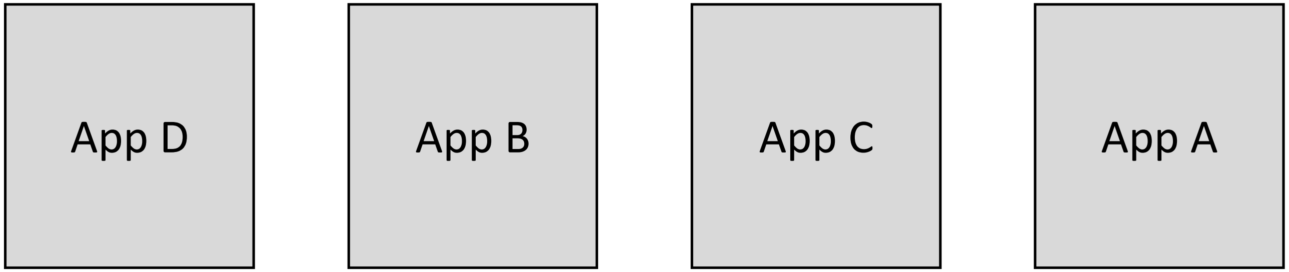
App B

App C

App A

# Benefits of Aggregation

Vanilla approach:



App D

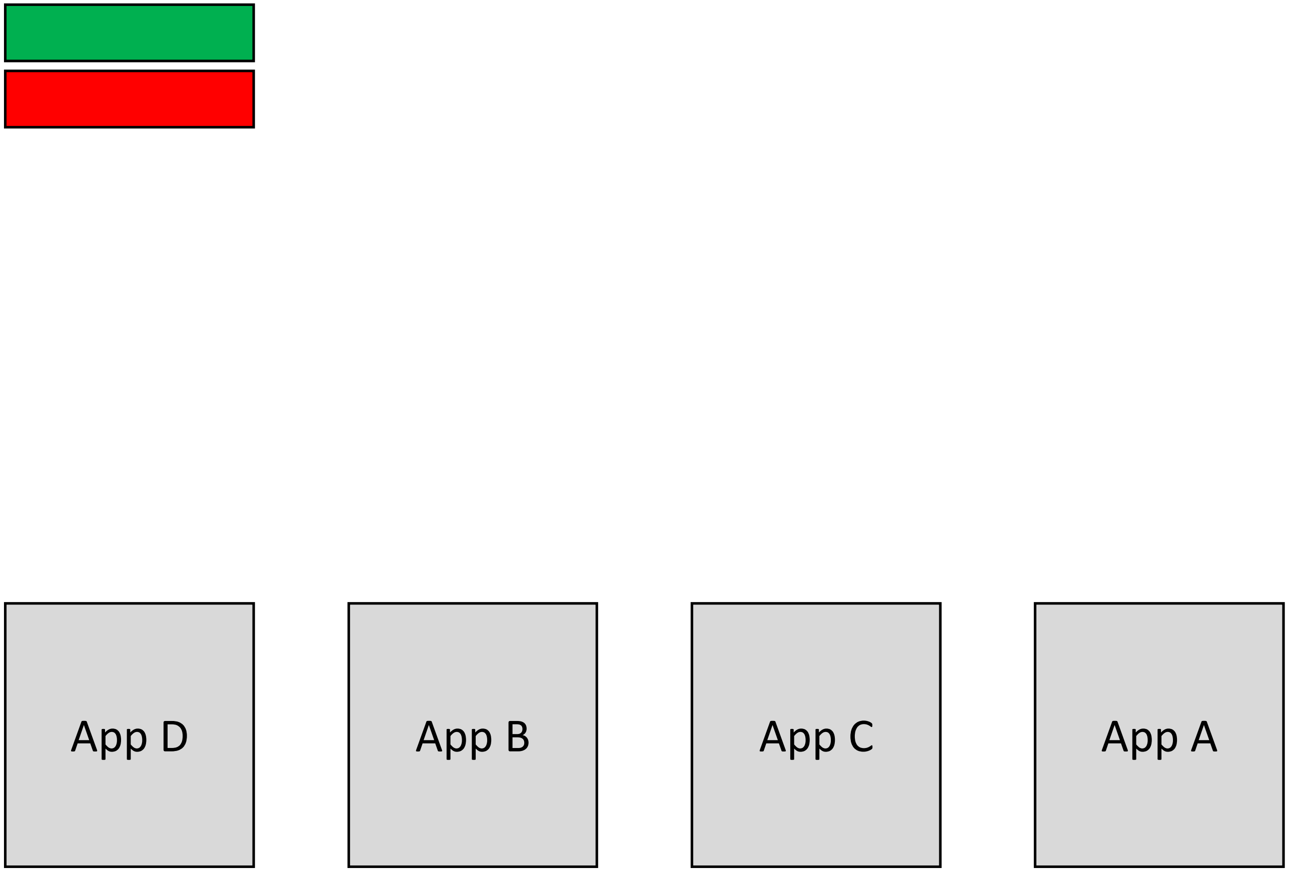
App B

App C

App A

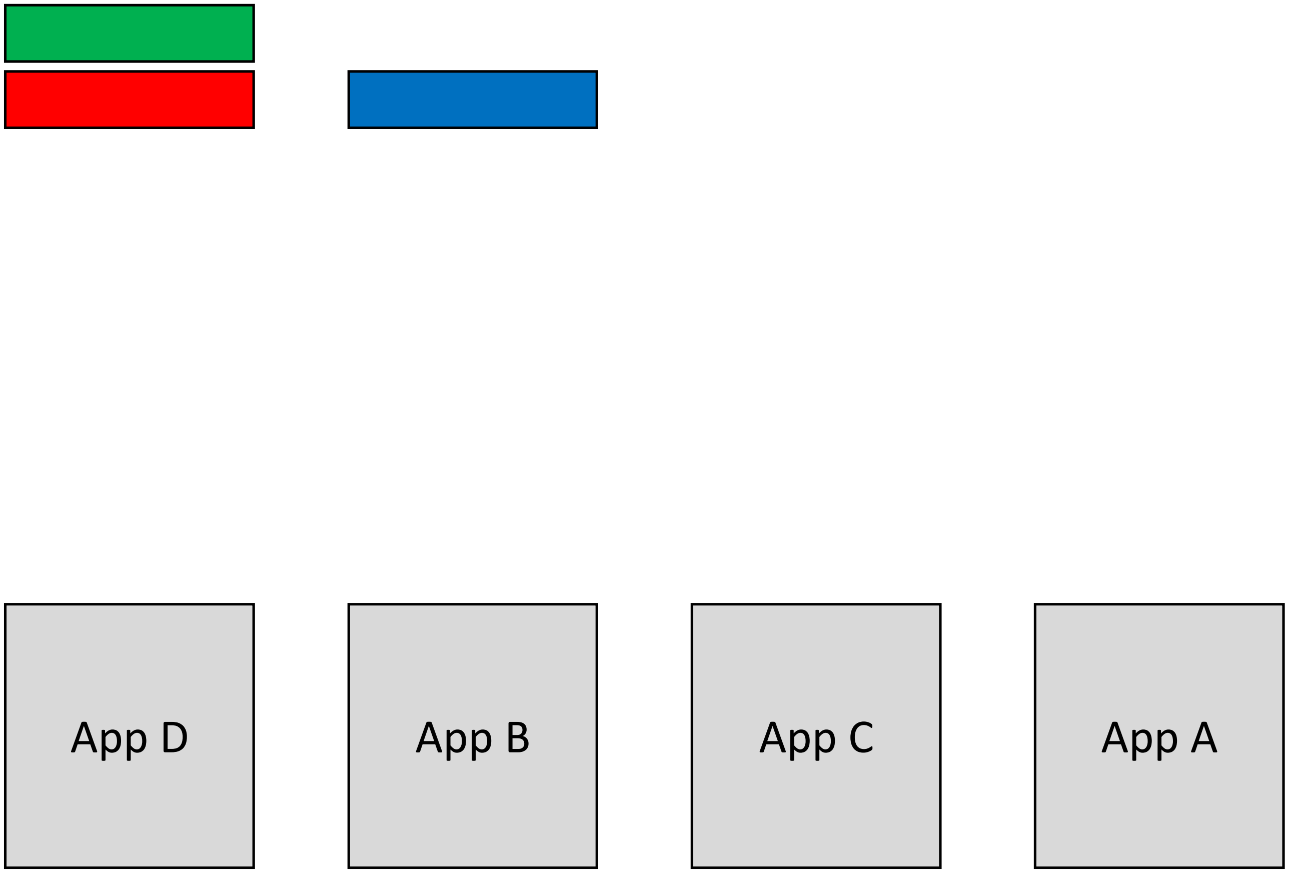
# Benefits of Aggregation

Vanilla approach:



# Benefits of Aggregation

Vanilla approach:



App D

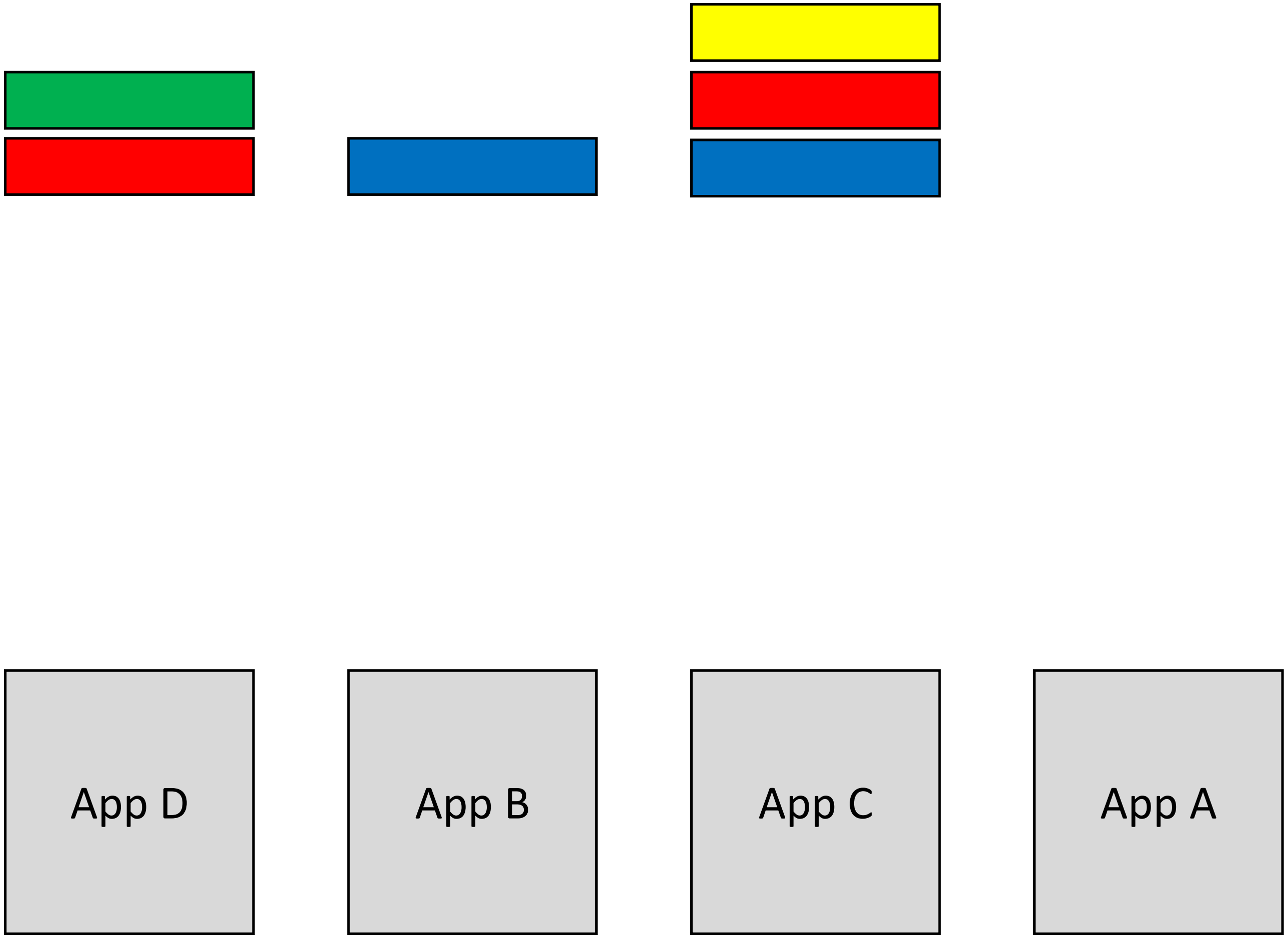
App B

App C

App A

# Benefits of Aggregation

Vanilla approach:



App D

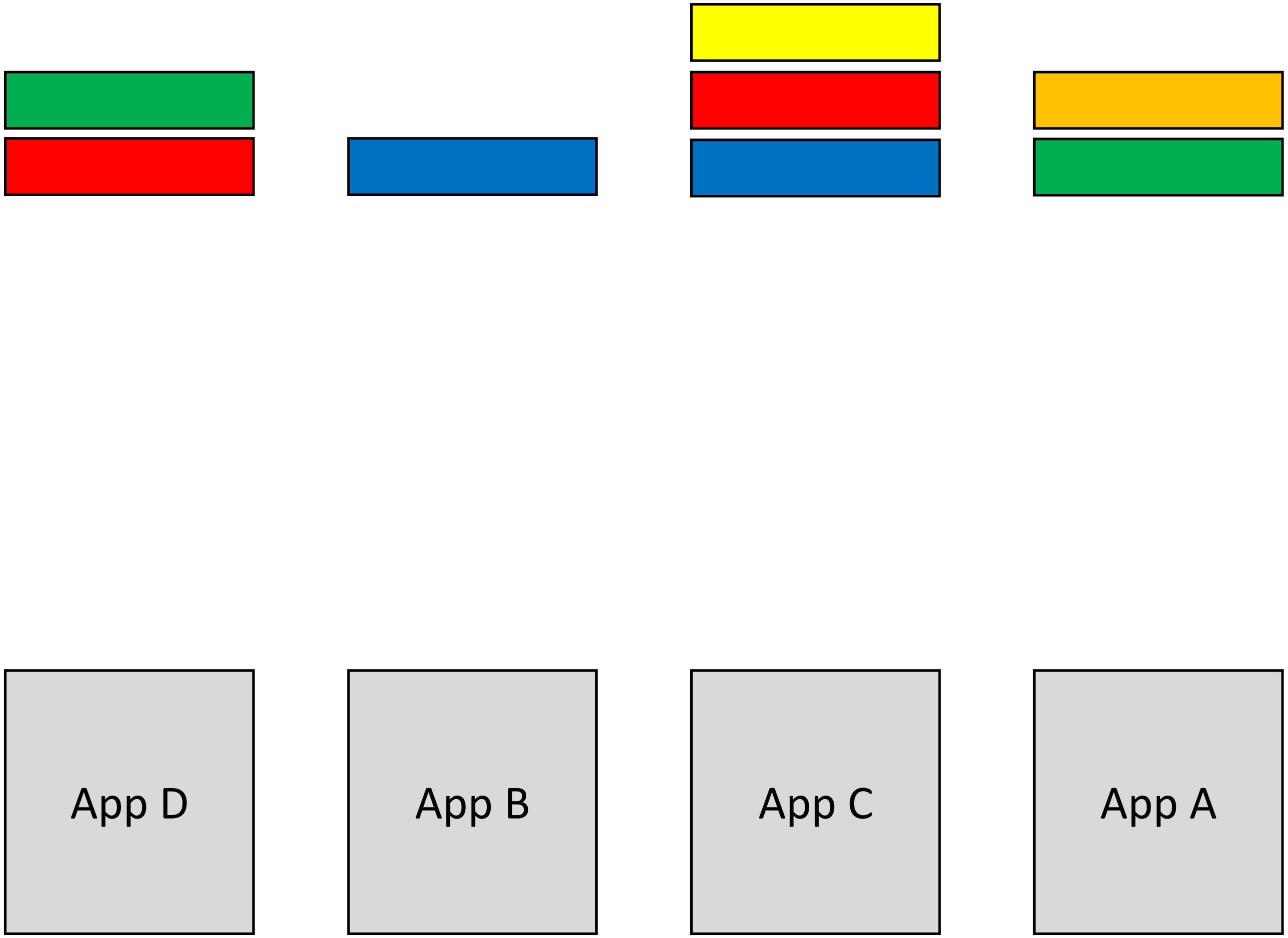
App B

App C

App A

# Benefits of Aggregation

Vanilla approach:



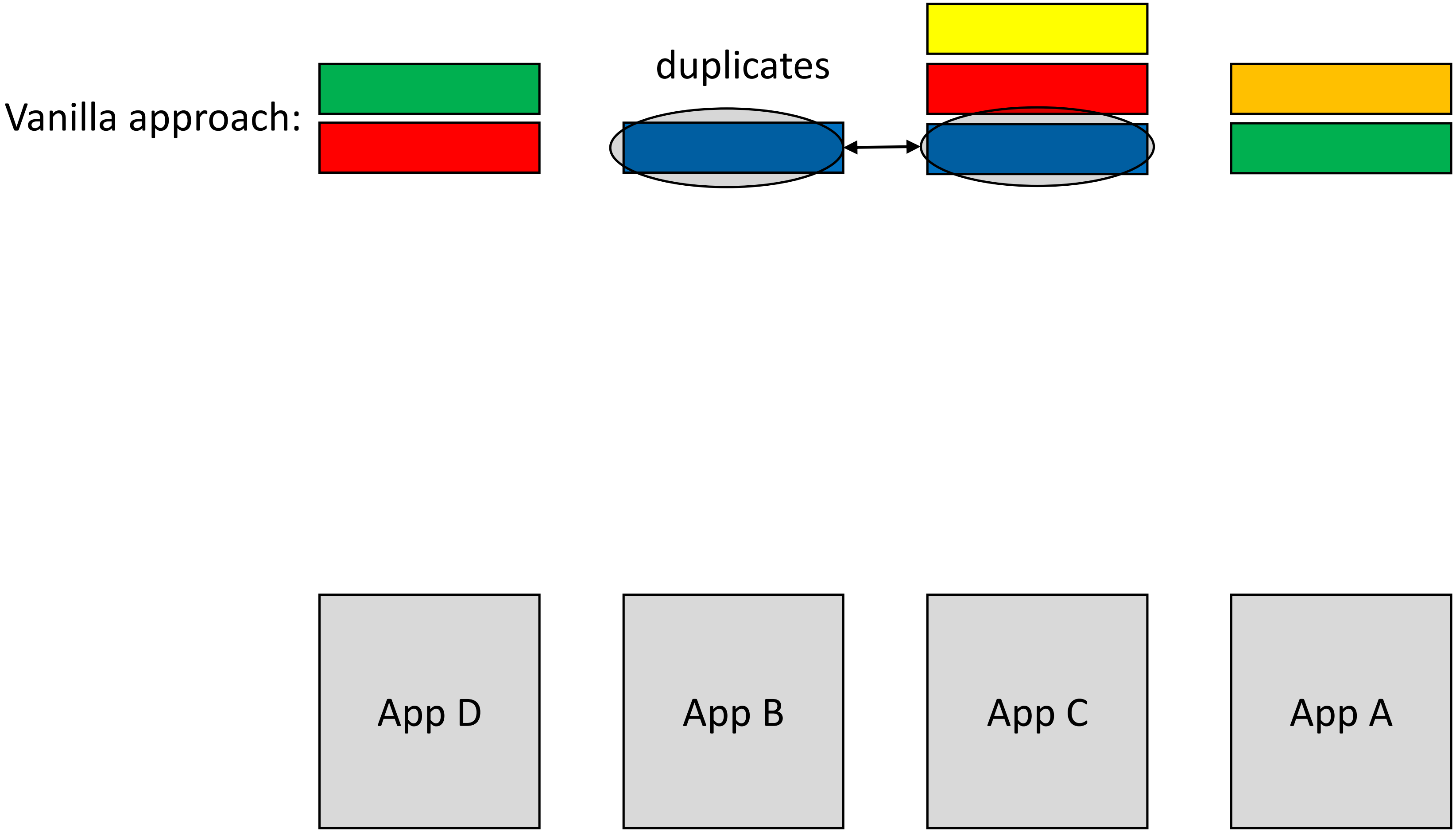
App D

App B

App C

App A

# Benefits of Aggregation



Vanilla approach:

duplicates

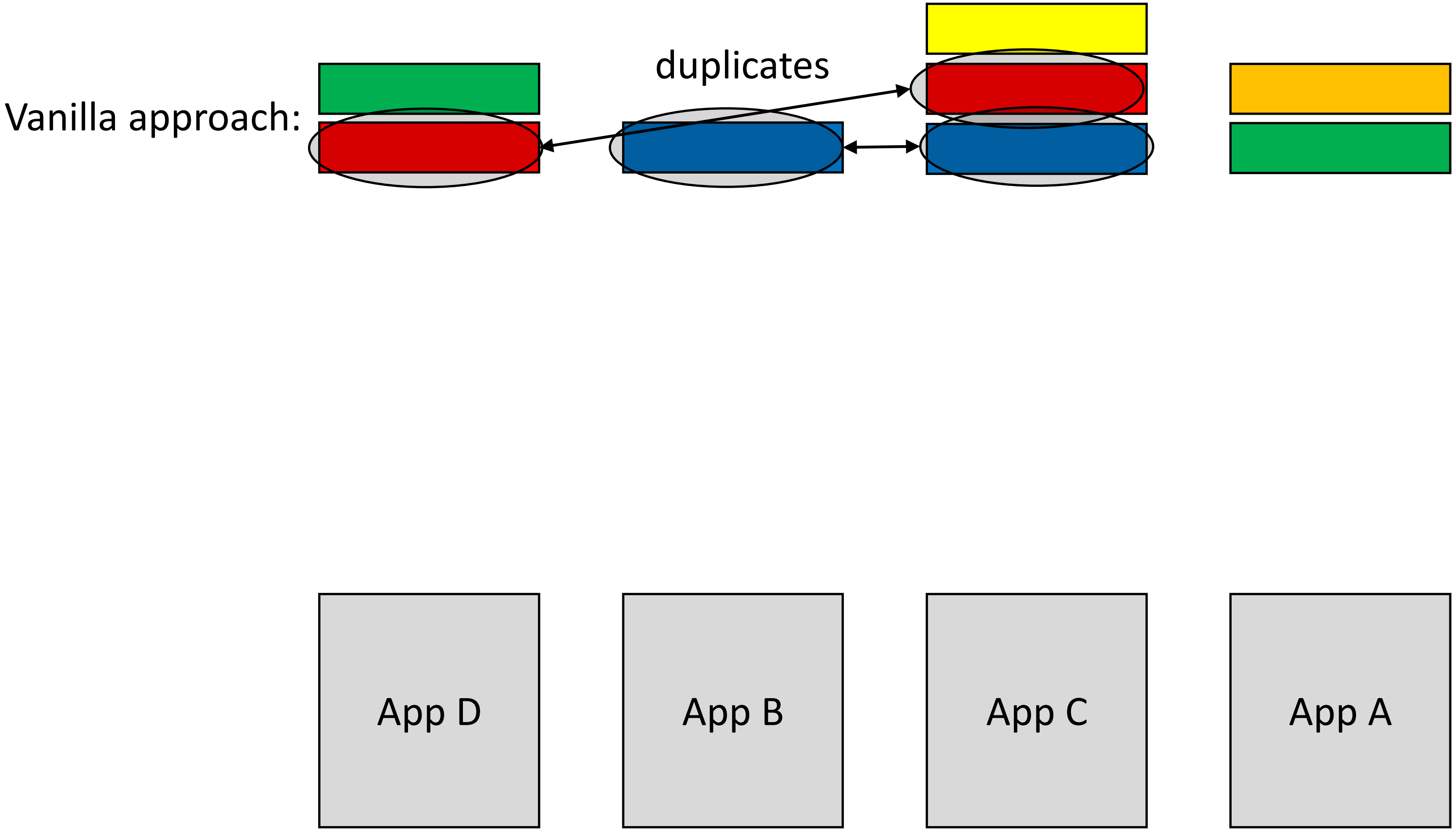
App D

App B

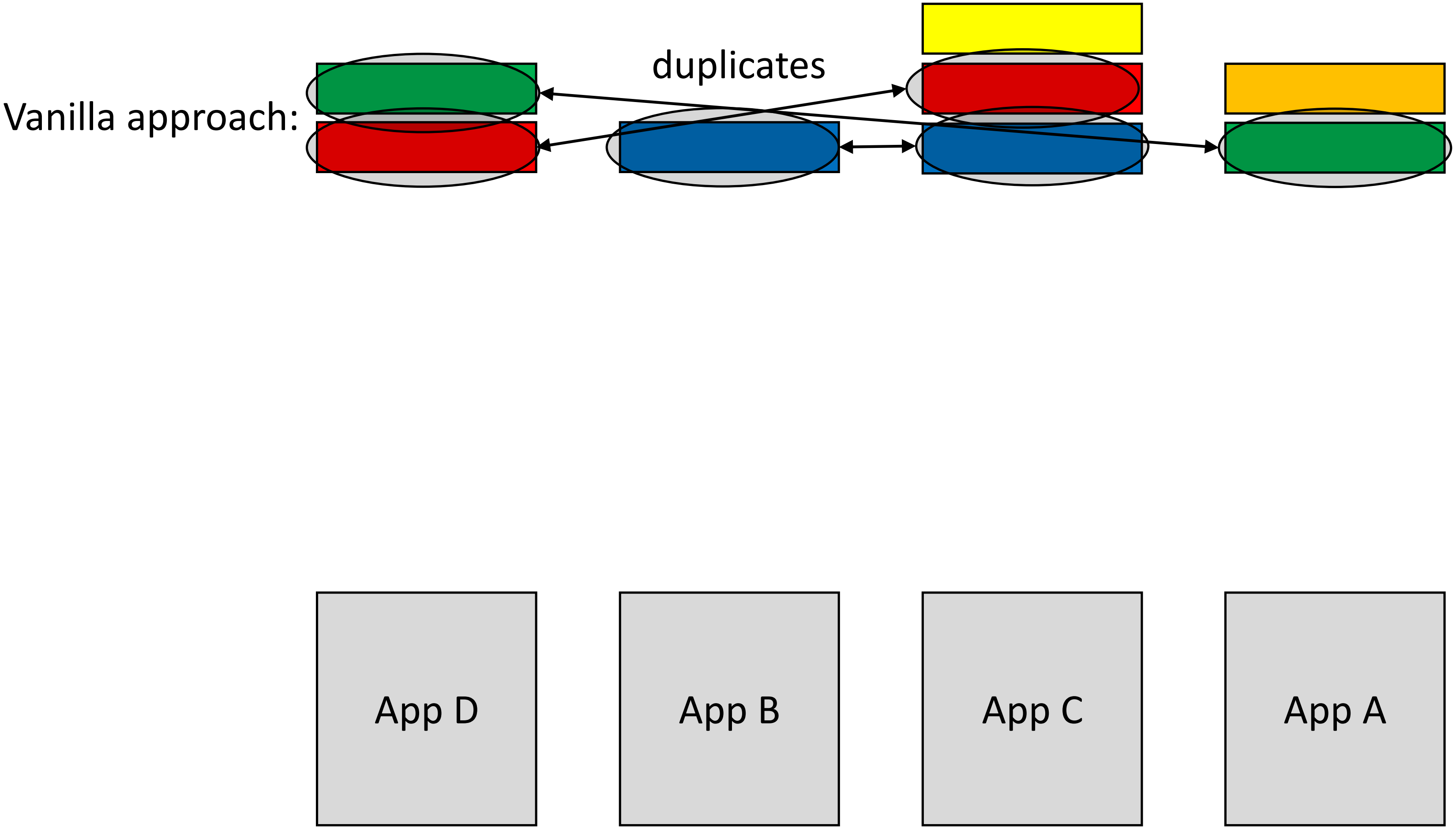
App C

App A

# Benefits of Aggregation



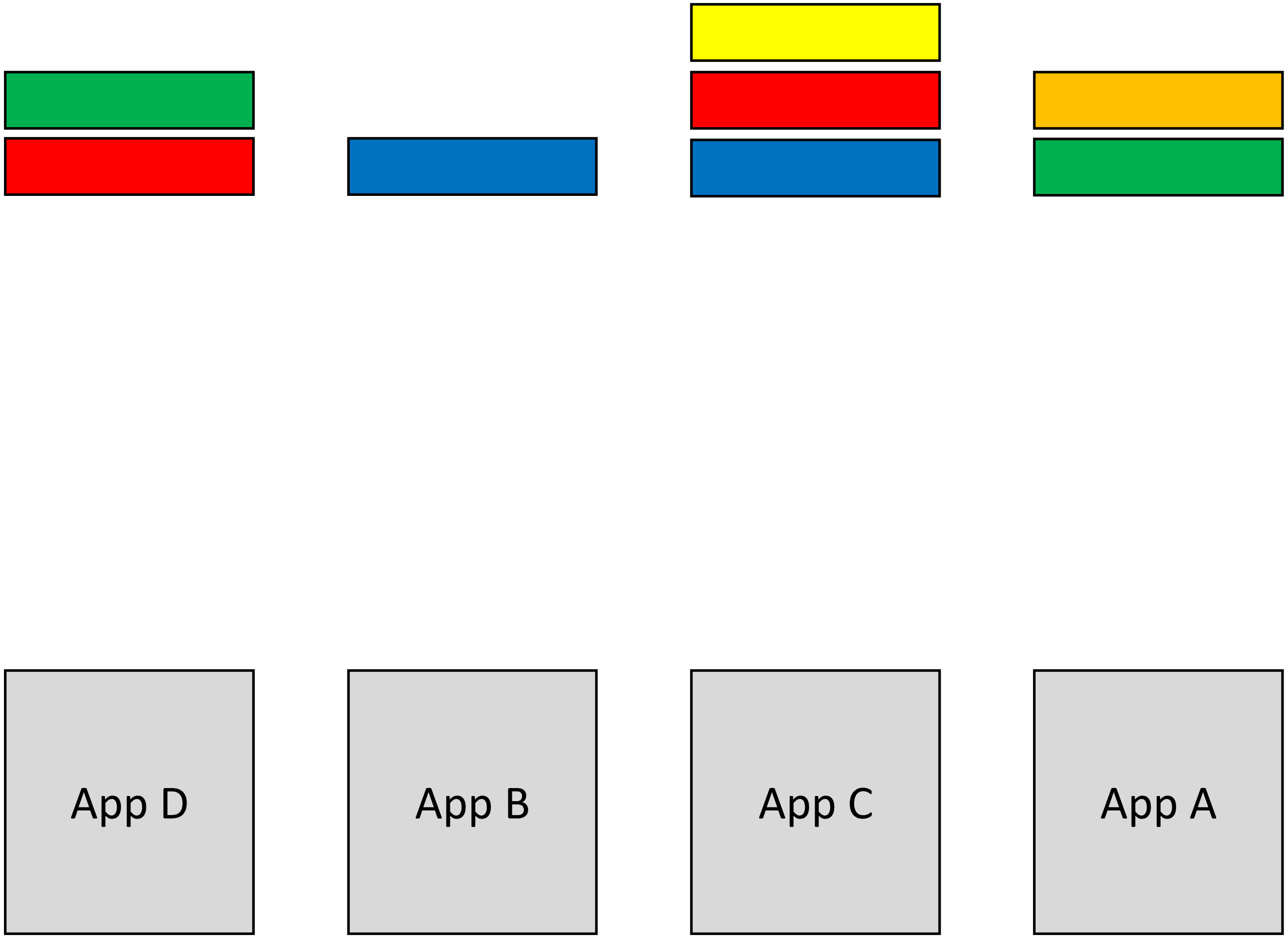
# Benefits of Aggregation



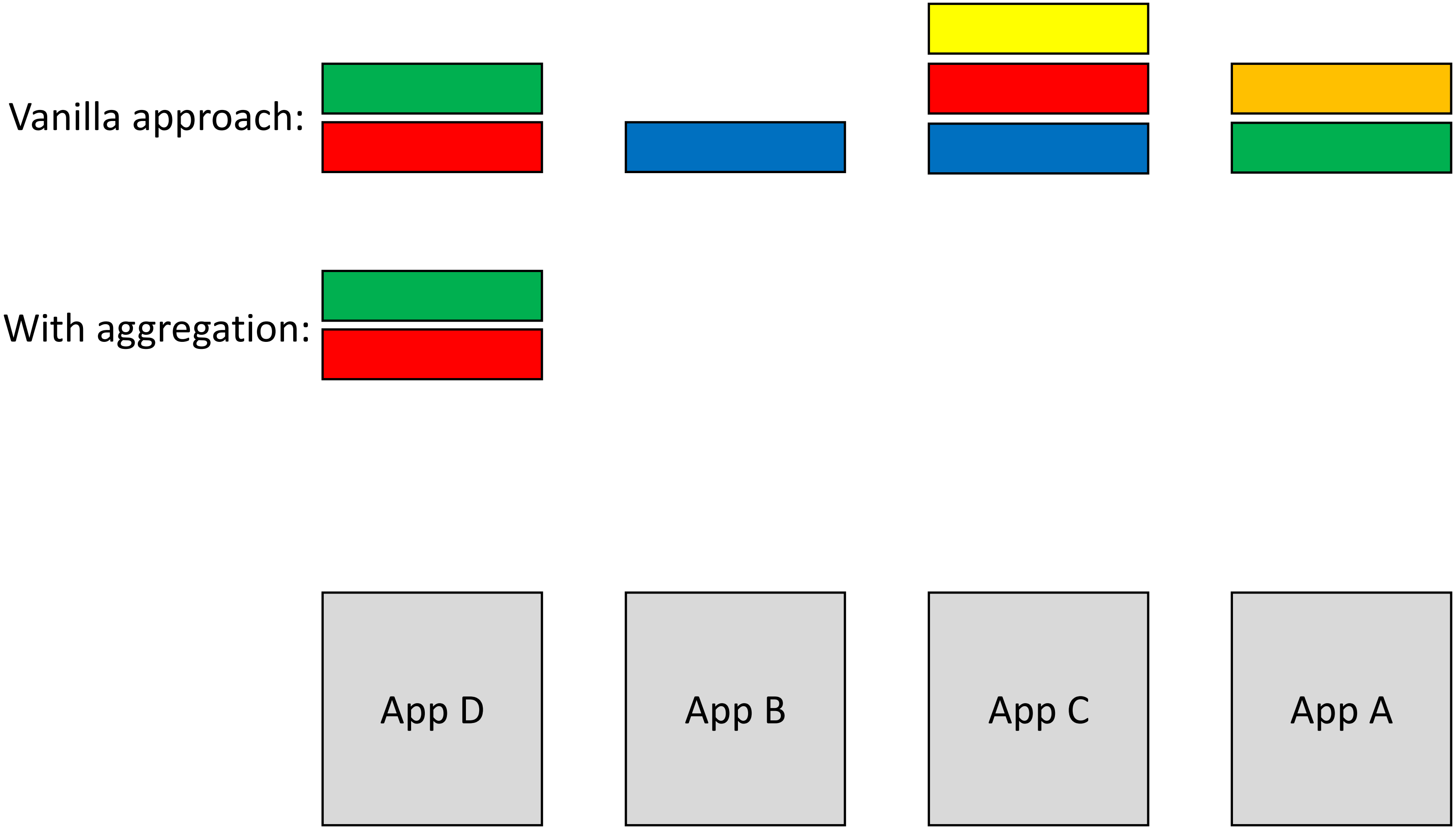
# Benefits of Aggregation

Vanilla approach:

With aggregation:



# Benefits of Aggregation



Vanilla approach:

With aggregation:

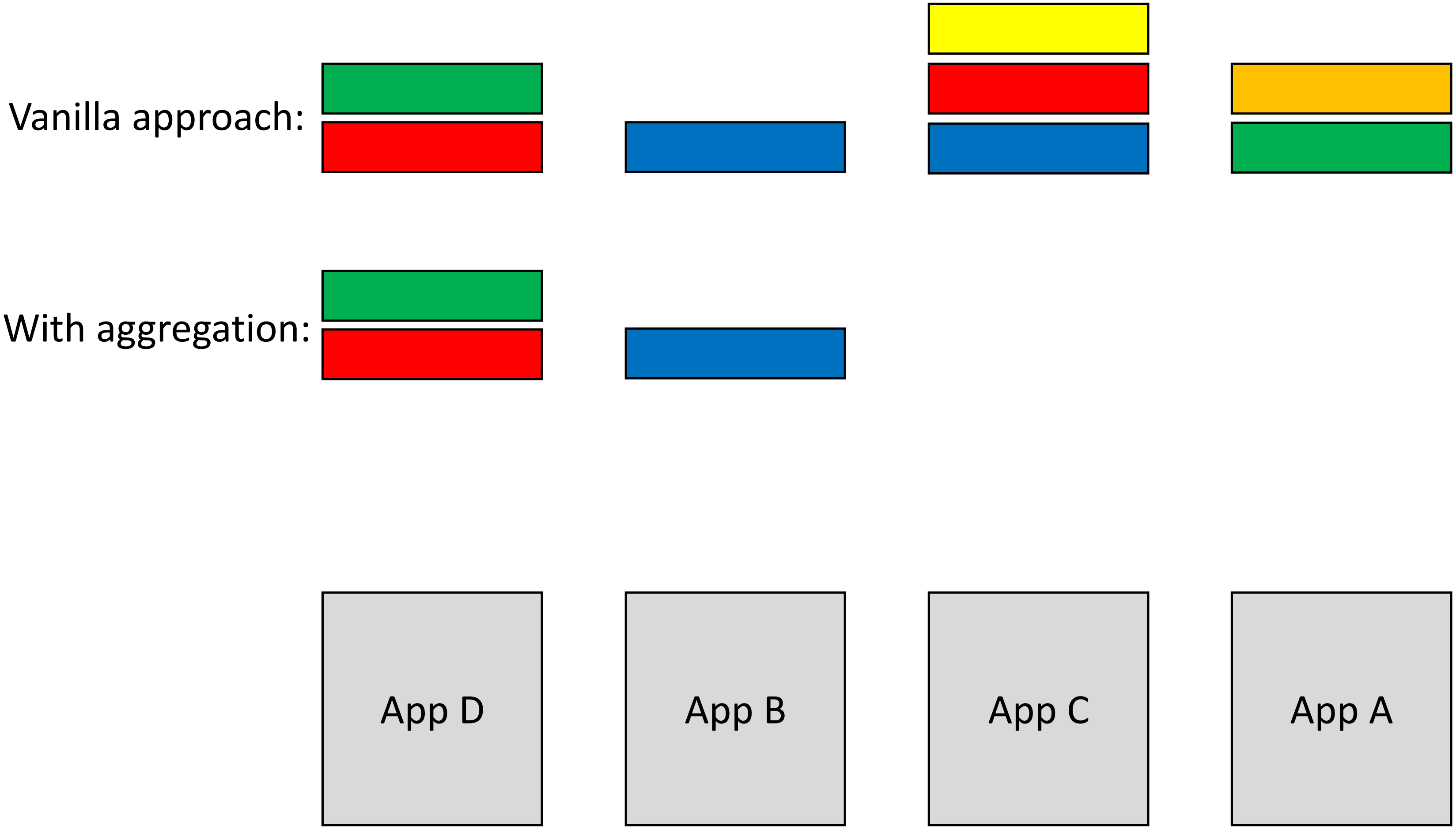
App D

App B

App C

App A

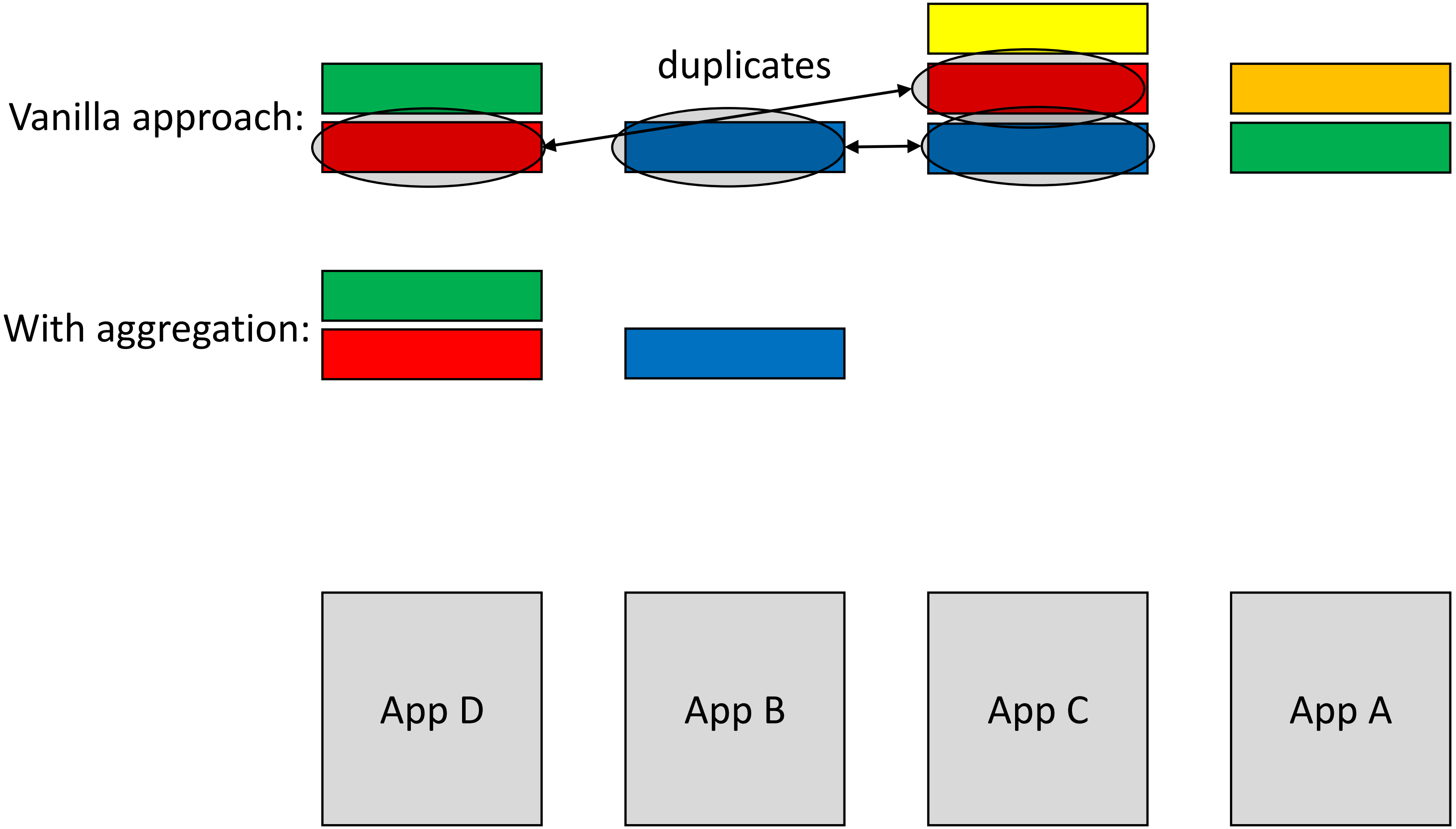
# Benefits of Aggregation



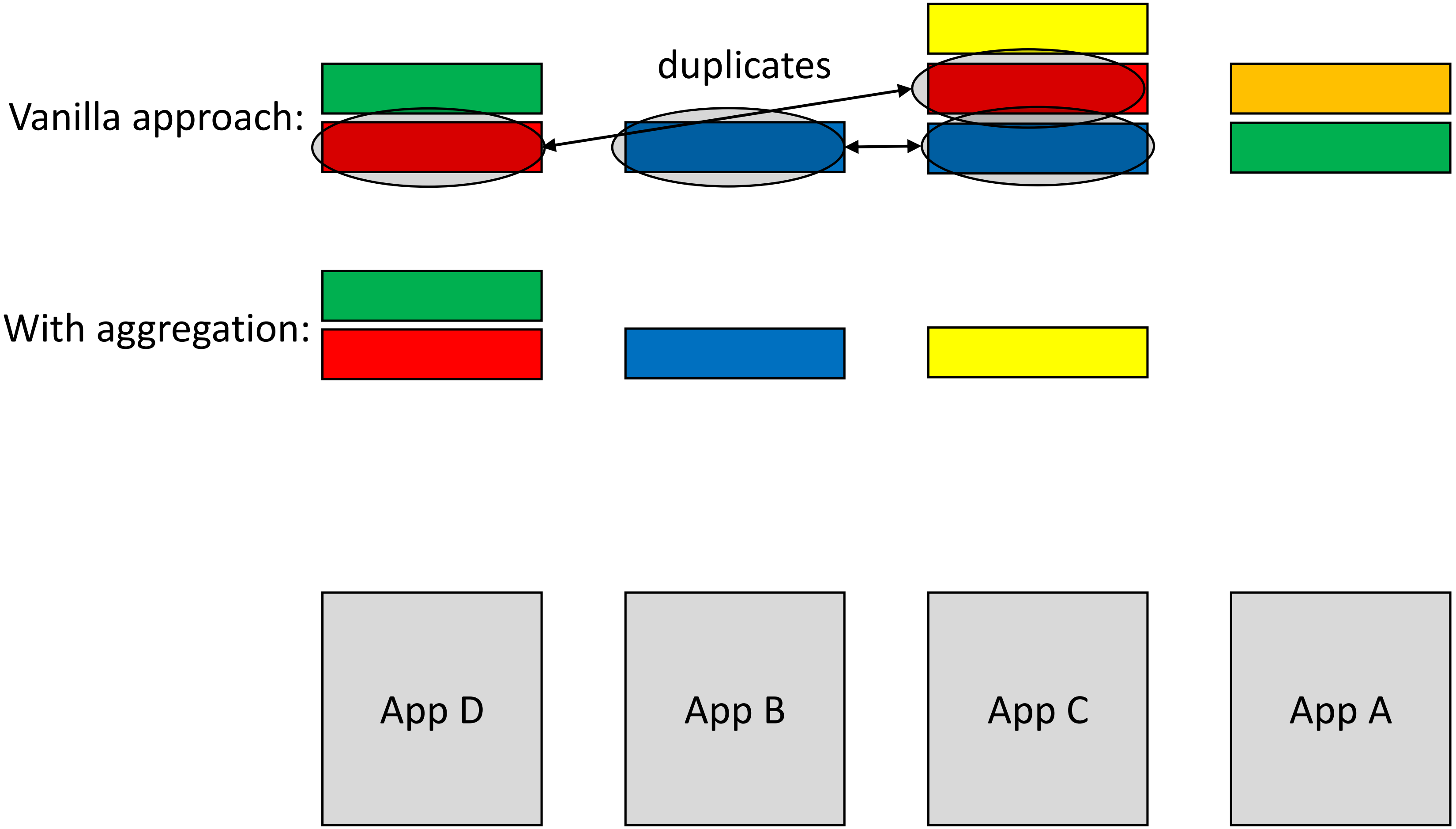
Vanilla approach:

With aggregation:

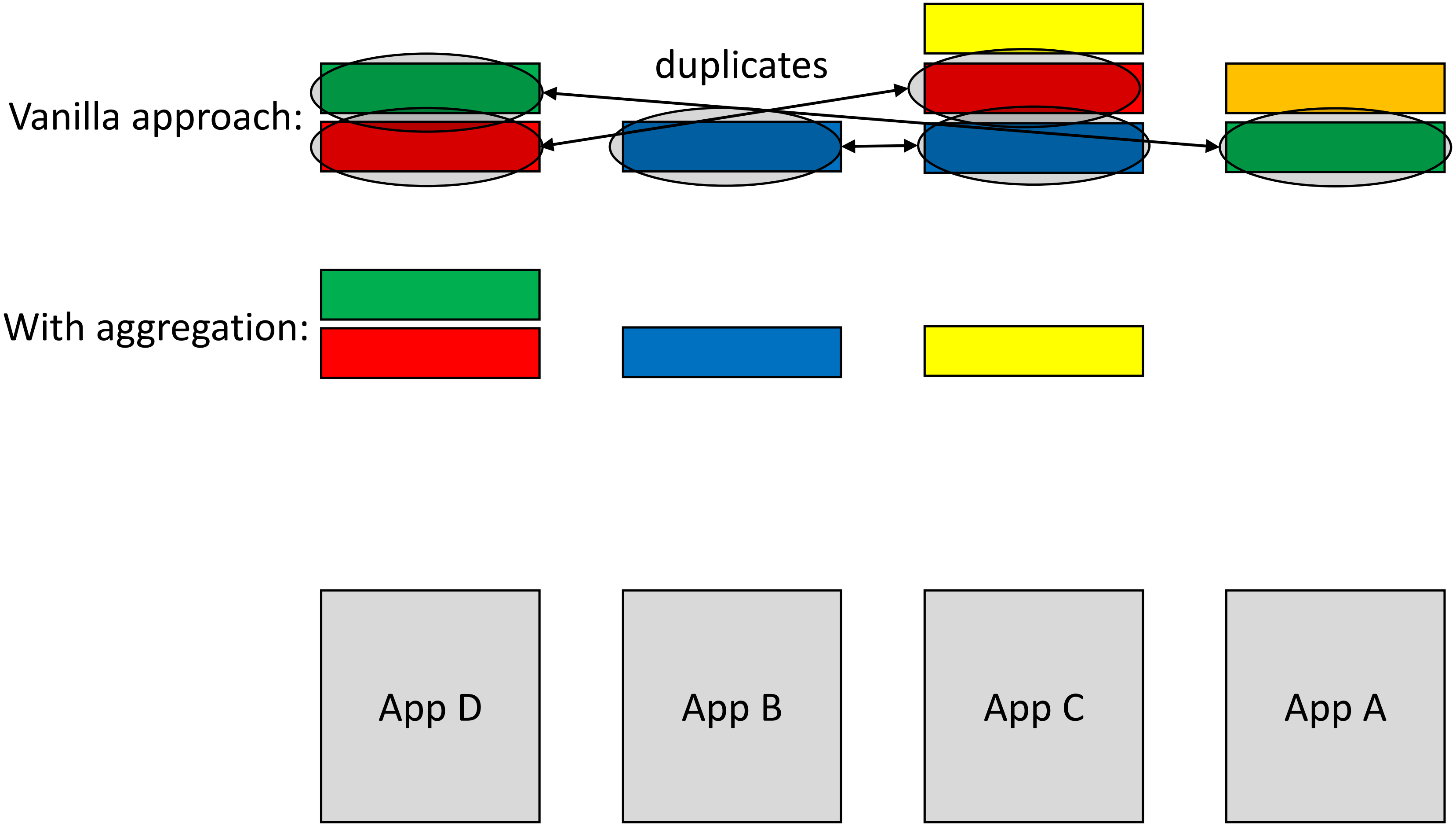
# Benefits of Aggregation



# Benefits of Aggregation



# Benefits of Aggregation



Vanilla approach:

duplicates

With aggregation:

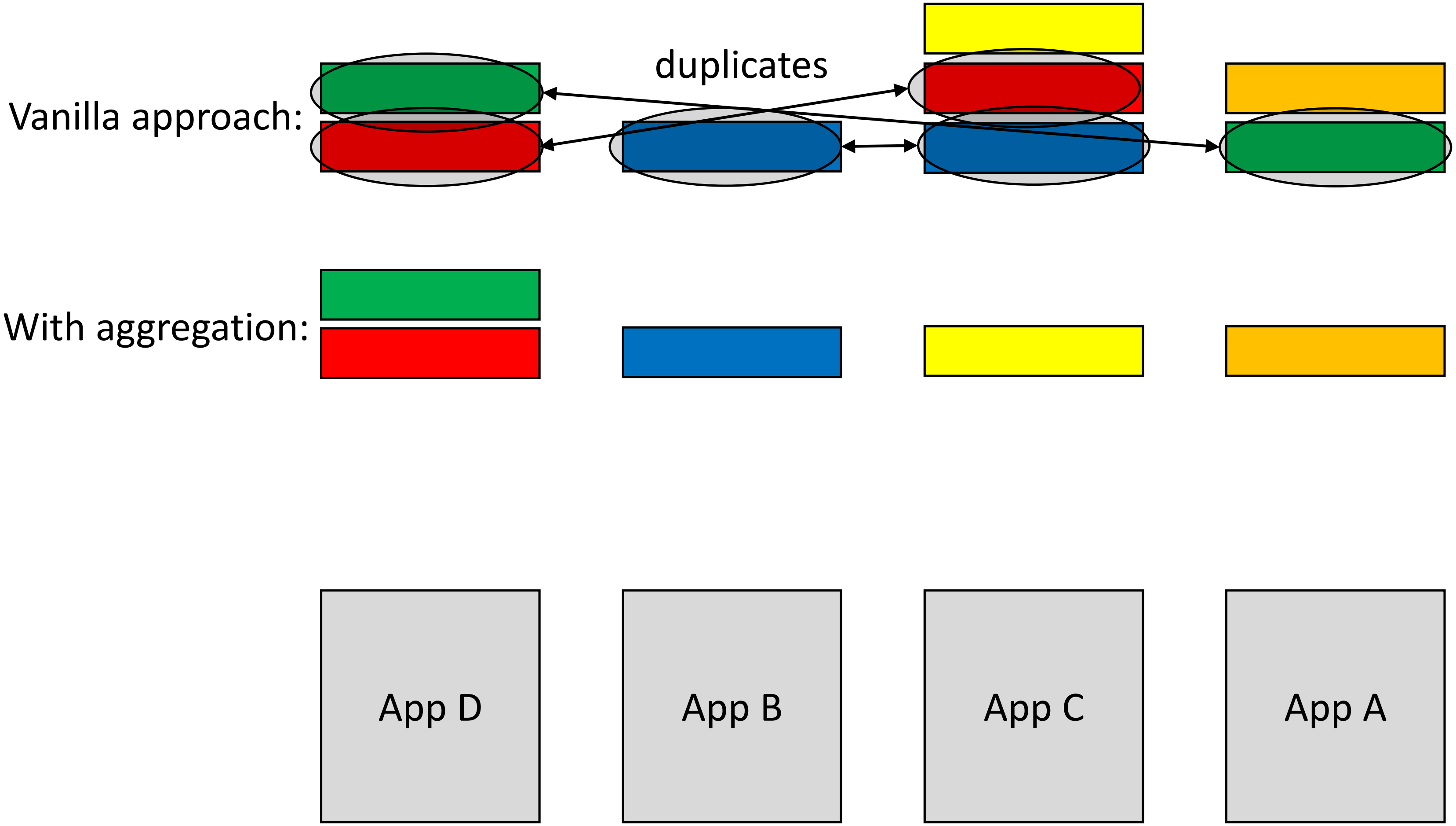
App D

App B

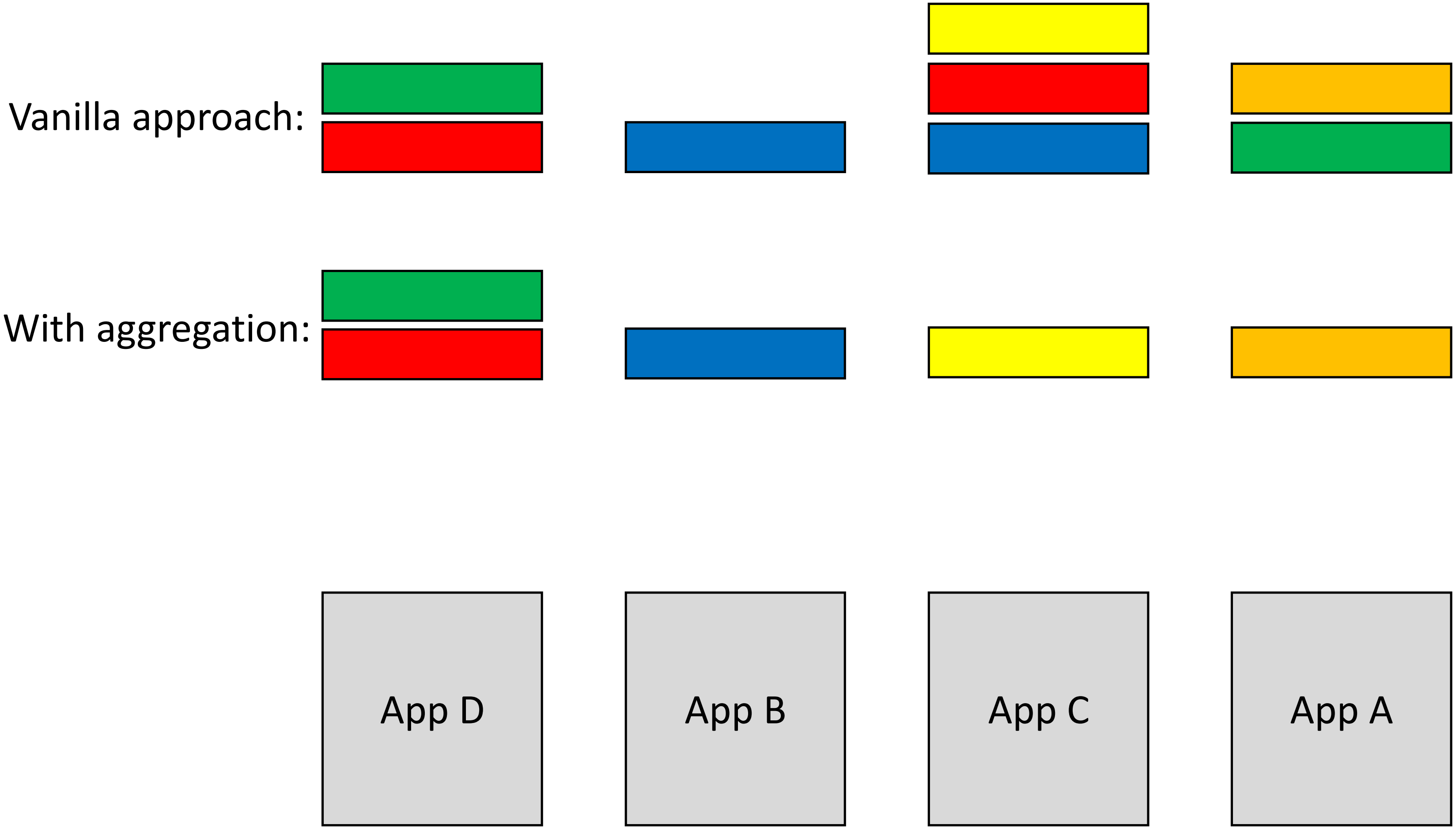
App C

App A

# Benefits of Aggregation



# Benefits of Aggregation



Vanilla approach:

With aggregation:

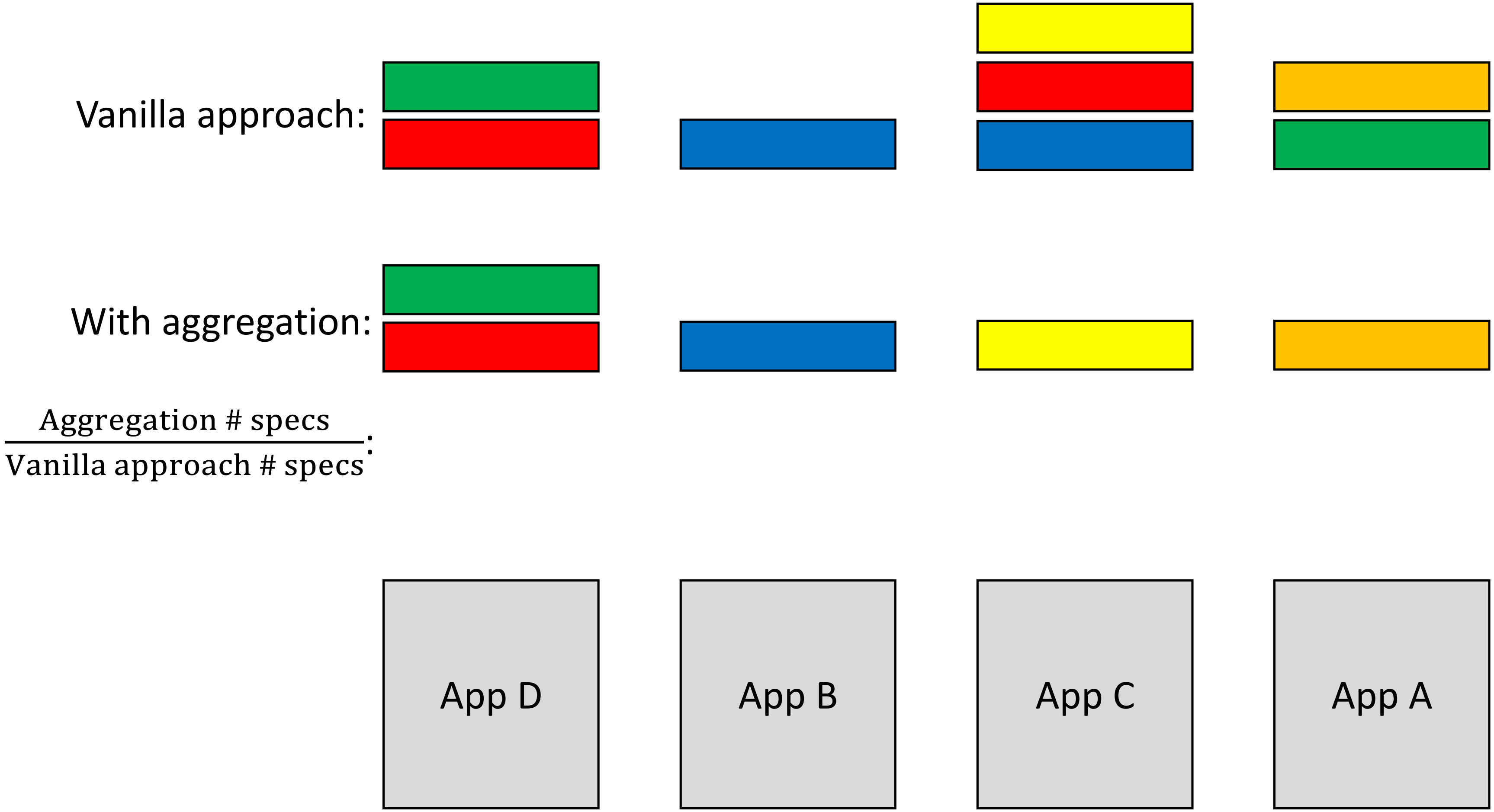
App D

App B

App C

App A

# Benefits of Aggregation

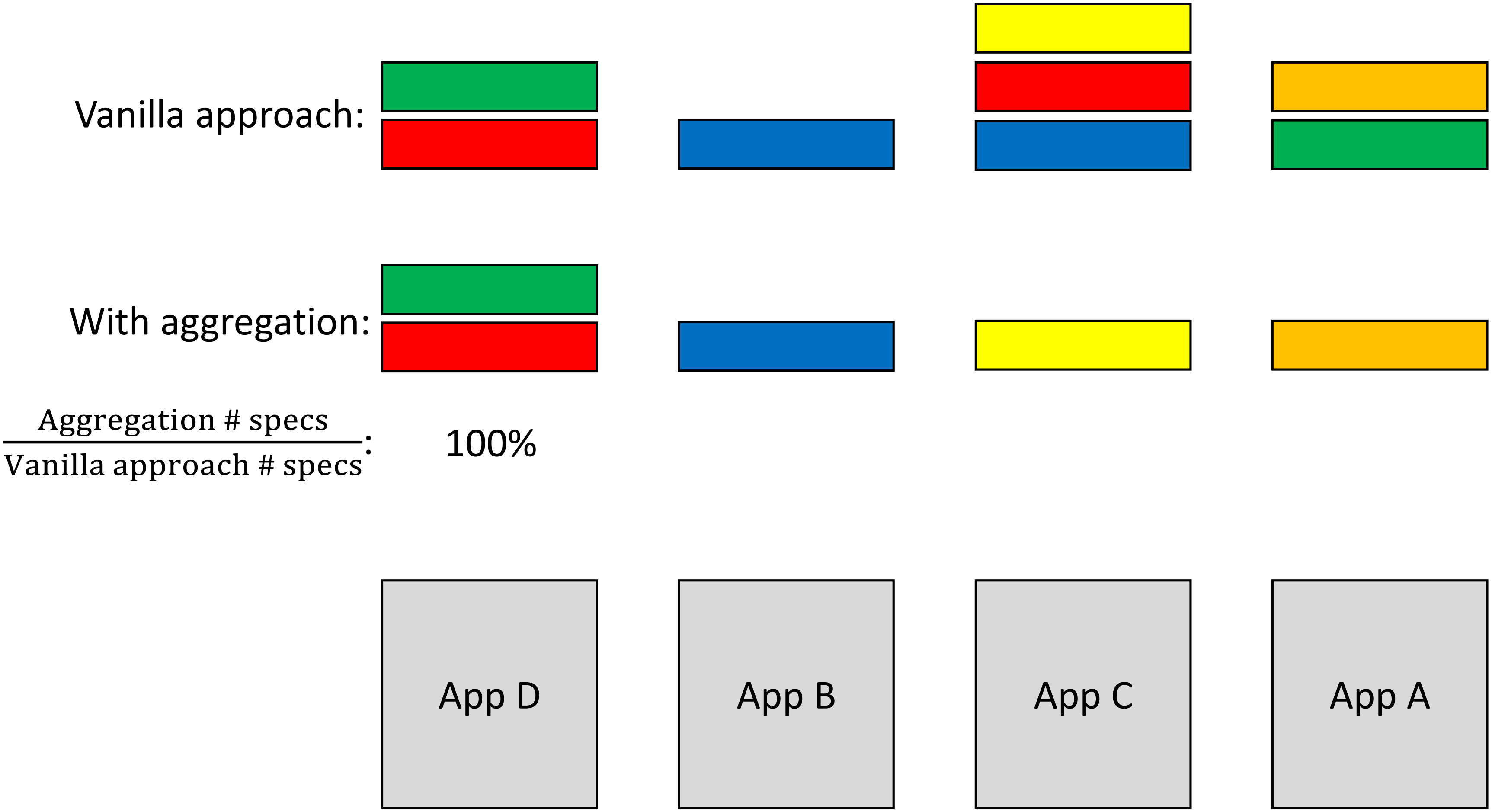


Vanilla approach:

With aggregation:

$$\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}:$$

# Benefits of Aggregation

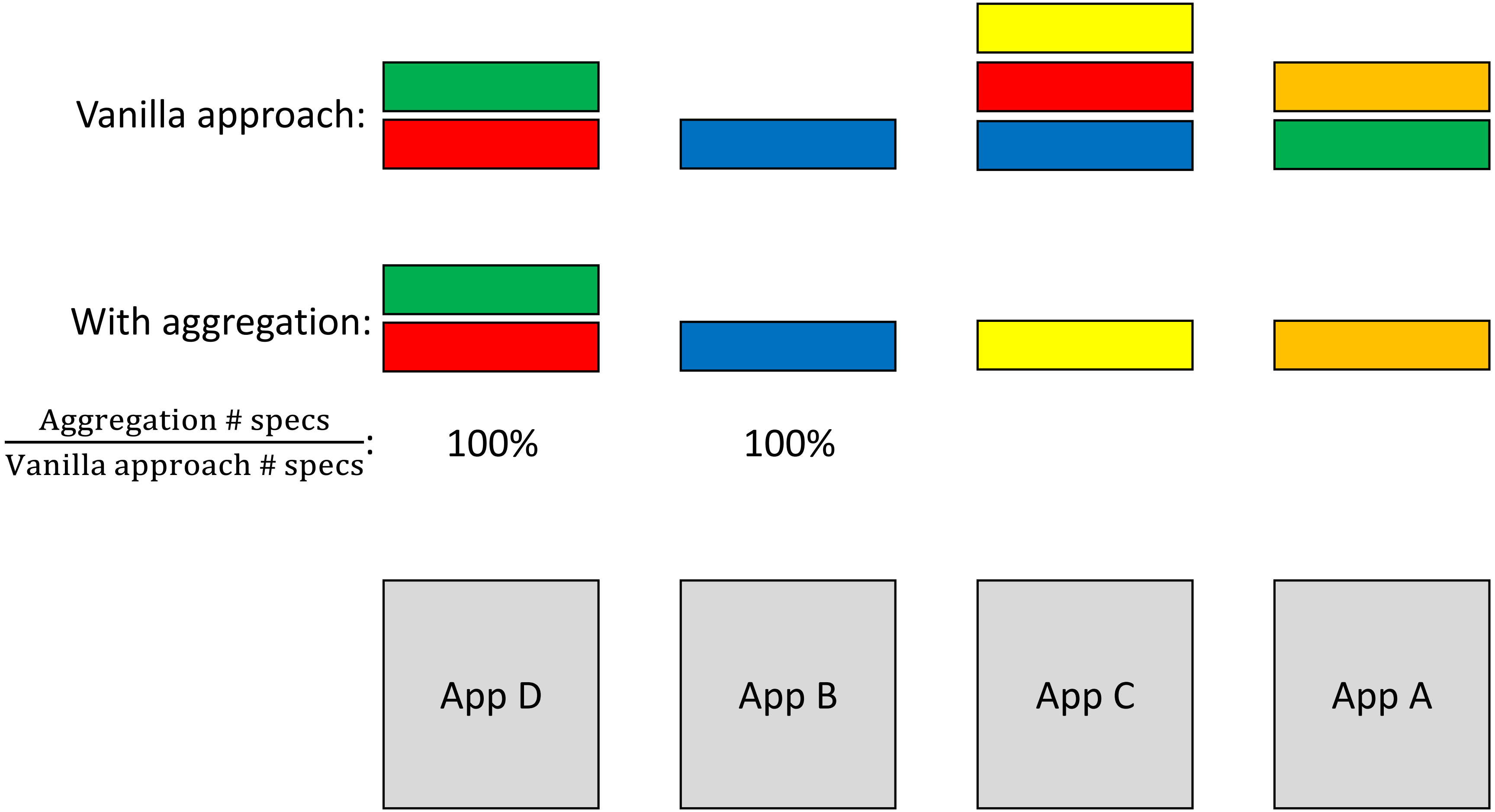


Vanilla approach:

With aggregation:

$\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$: 100%

# Benefits of Aggregation



Vanilla approach:

With aggregation:

$\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$: 100% 100%

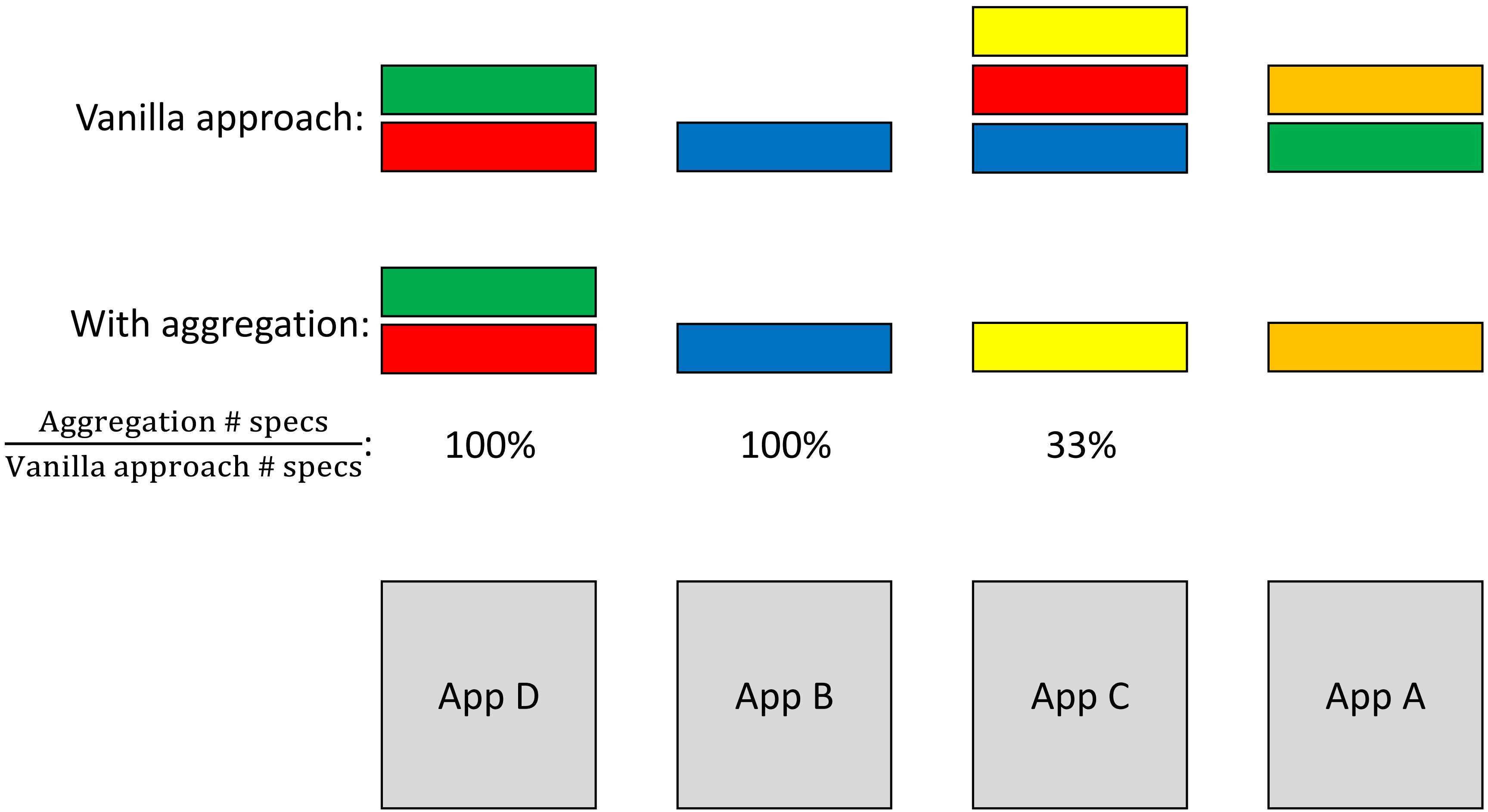
App D

App B

App C

App A

# Benefits of Aggregation



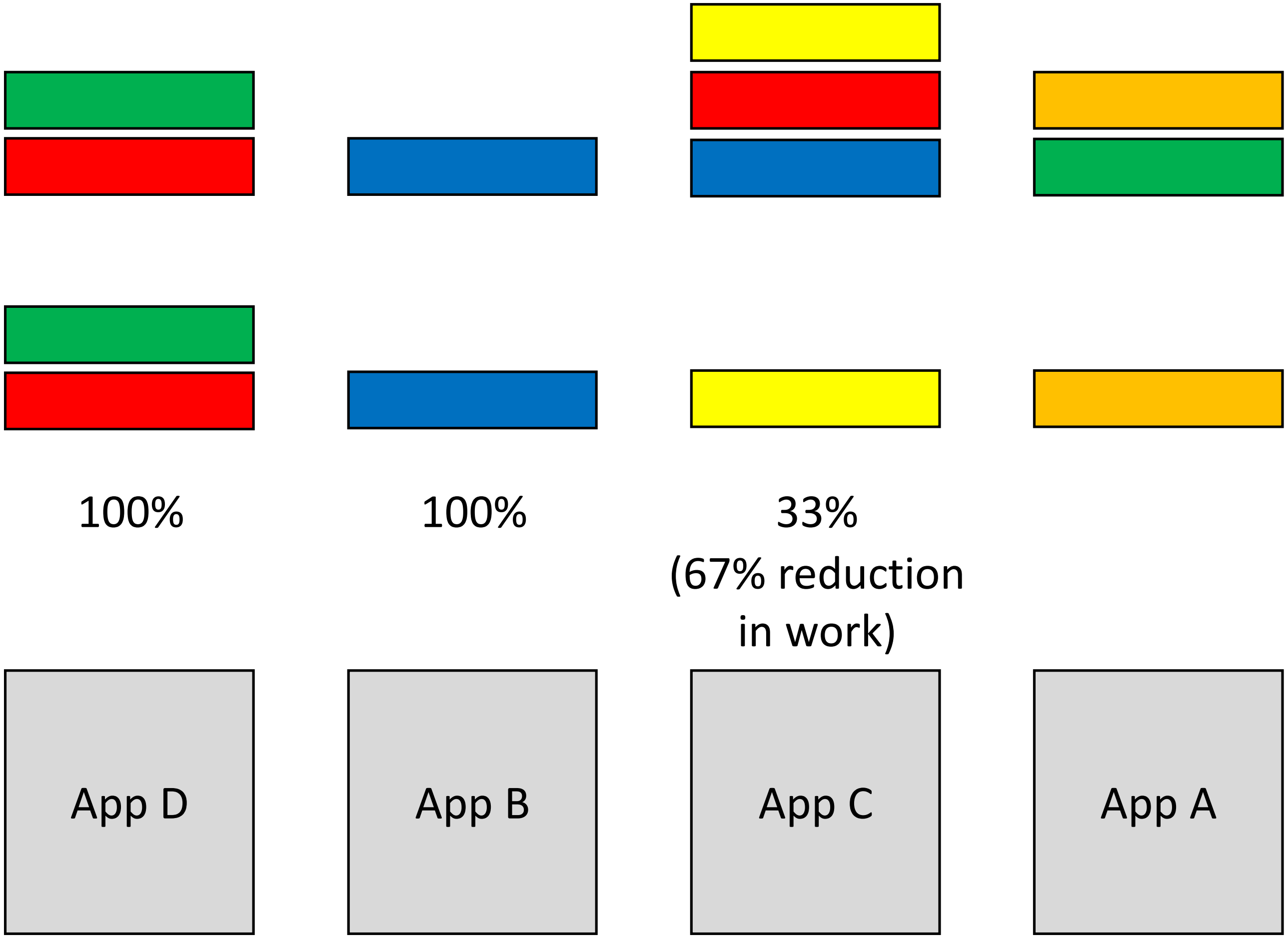
Vanilla approach:

With aggregation:

$\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$: 100% 100% 33%

App D App B App C App A

# Benefits of Aggregation



Vanilla approach:

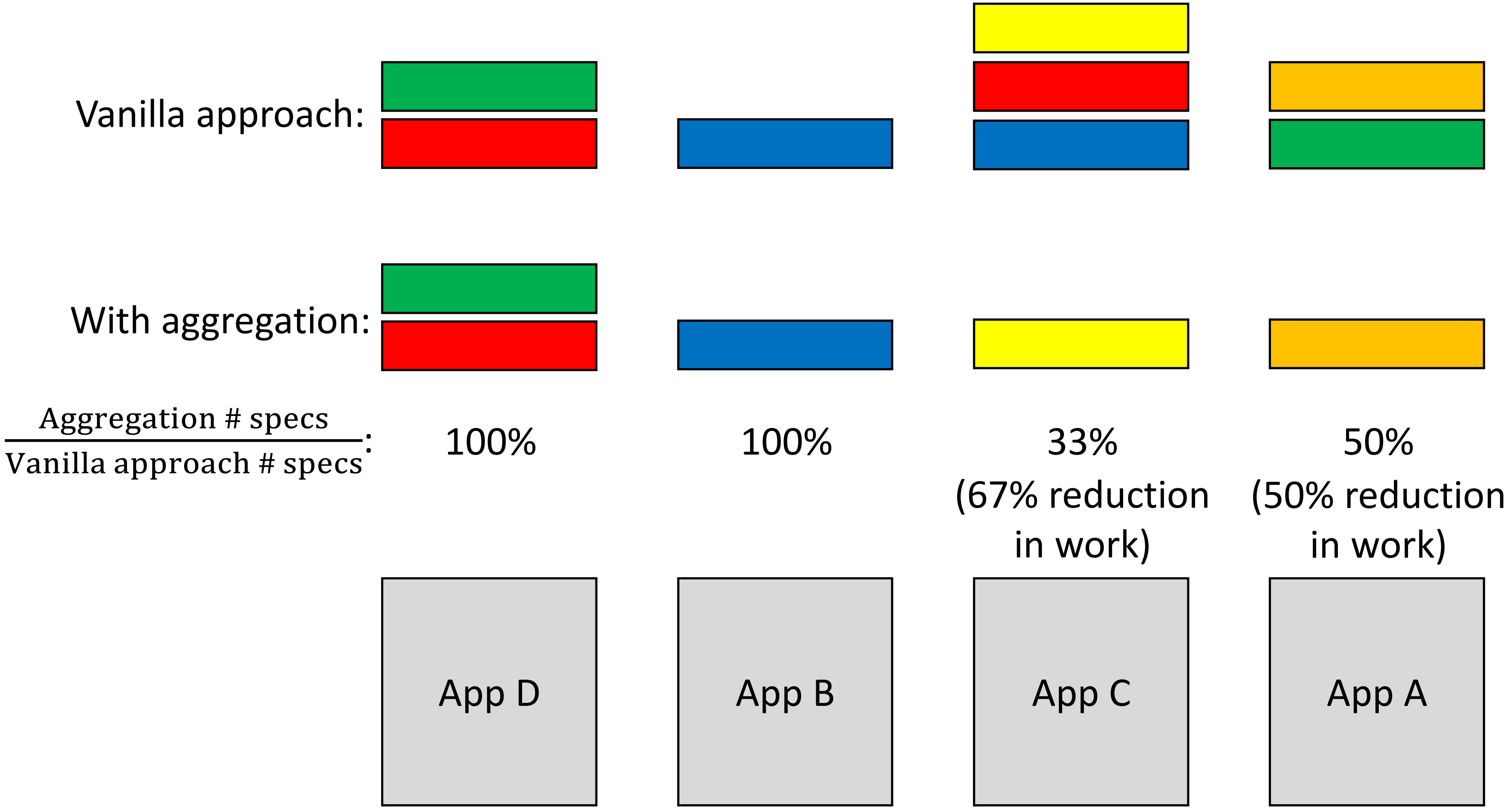
With aggregation:

| | App D | App B | App C | App A |
|---|---|---|---|---|
| $\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$: | 100% | 100% | 33%<br>(67% reduction in work) | |

# Benefits of Aggregation

| | | | | |
|---|---|---|---|---|
| Vanilla approach: | 🟩<br>🟥 | 🟦 | 🟨<br>🟥<br>🟦 | 🟧<br>🟩 |
| With aggregation: | 🟩<br>🟥 | 🟦 | 🟨 | 🟧 |
| $\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$: | 100% | 100% | 33%<br>(67% reduction in work) | 50% |
| | App D | App B | App C | App A |

# Benefits of Aggregation



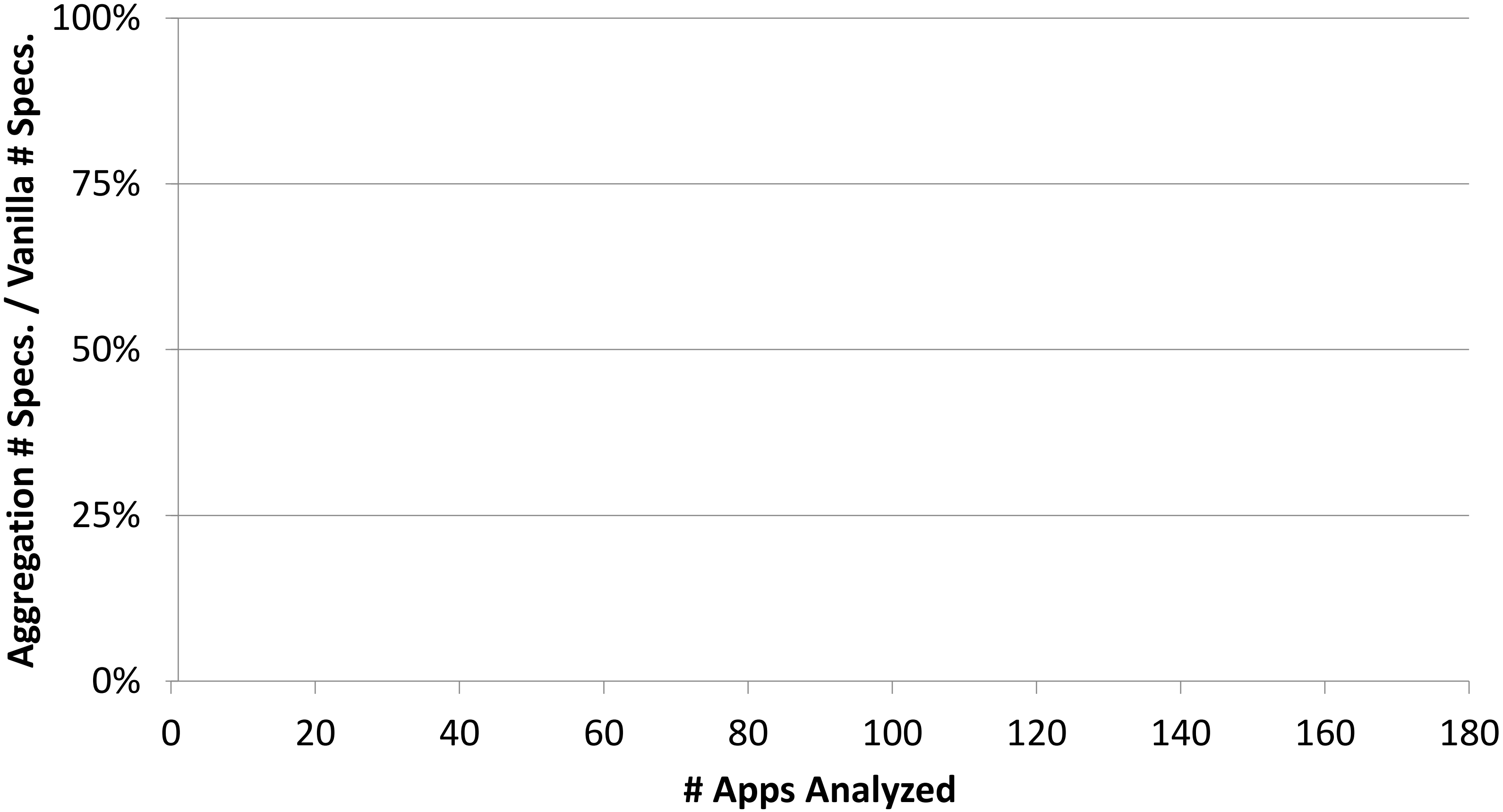
Vanilla approach:

With aggregation:

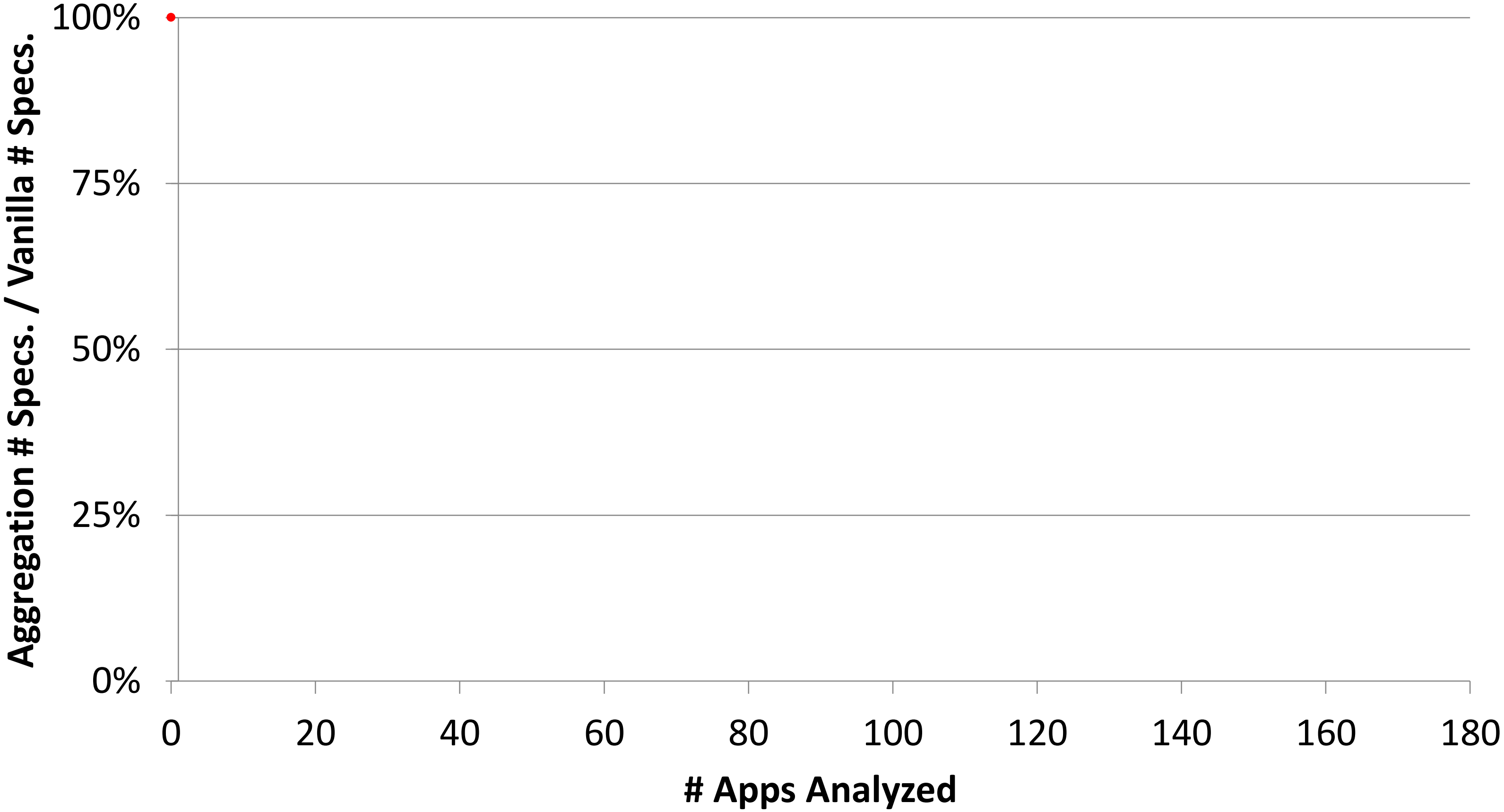
$\frac{\text{Aggregation \# specs}}{\text{Vanilla approach \# specs}}$:

| App D | App B | App C | App A |
|---|---|---|---|
| 100% | 100% | 33%<br>(67% reduction in work) | 50%<br>(50% reduction in work) |

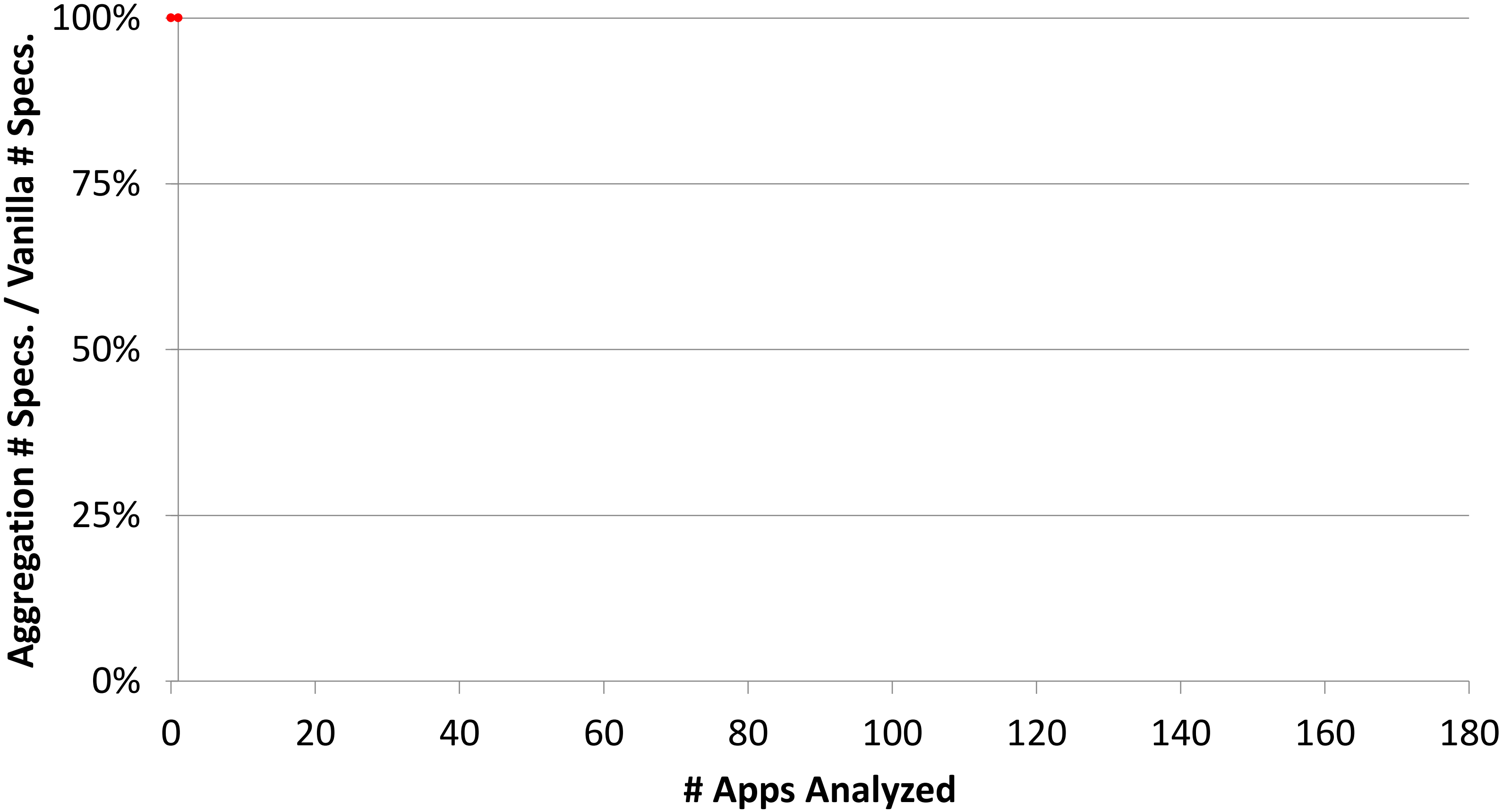
# Benefits of Aggregation



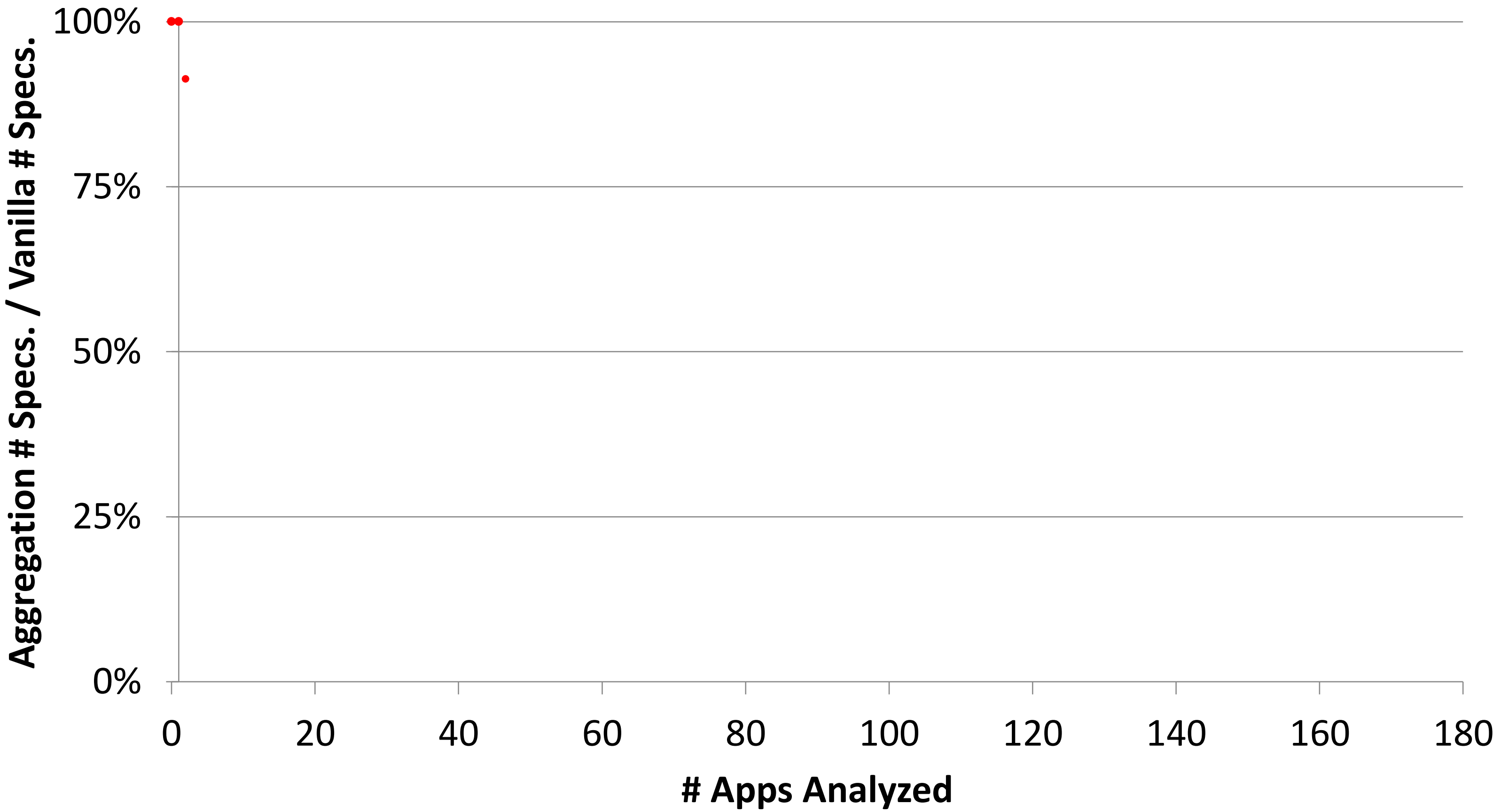
# Benefits of Aggregation



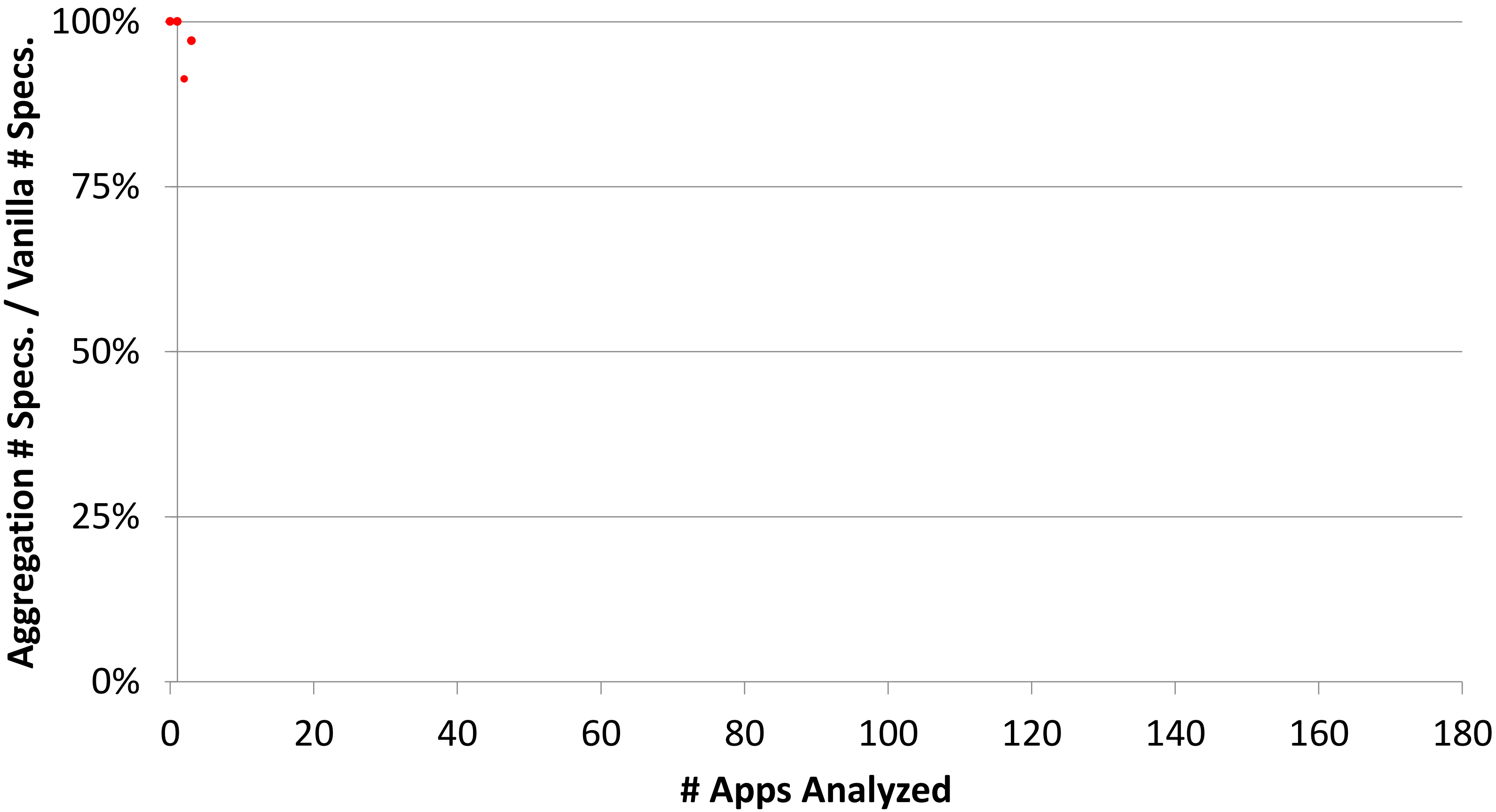
# Benefits of Aggregation



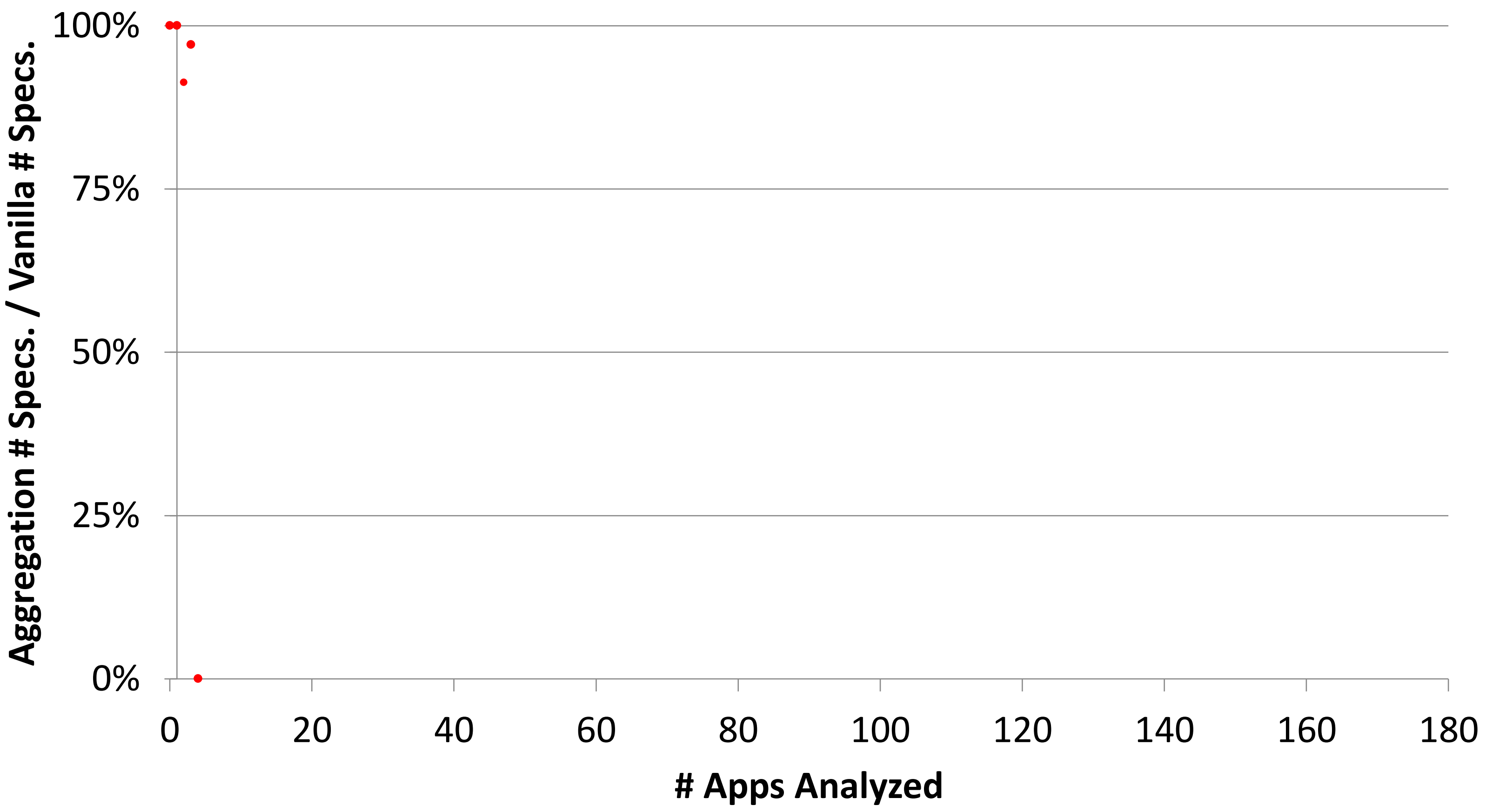
# Benefits of Aggregation



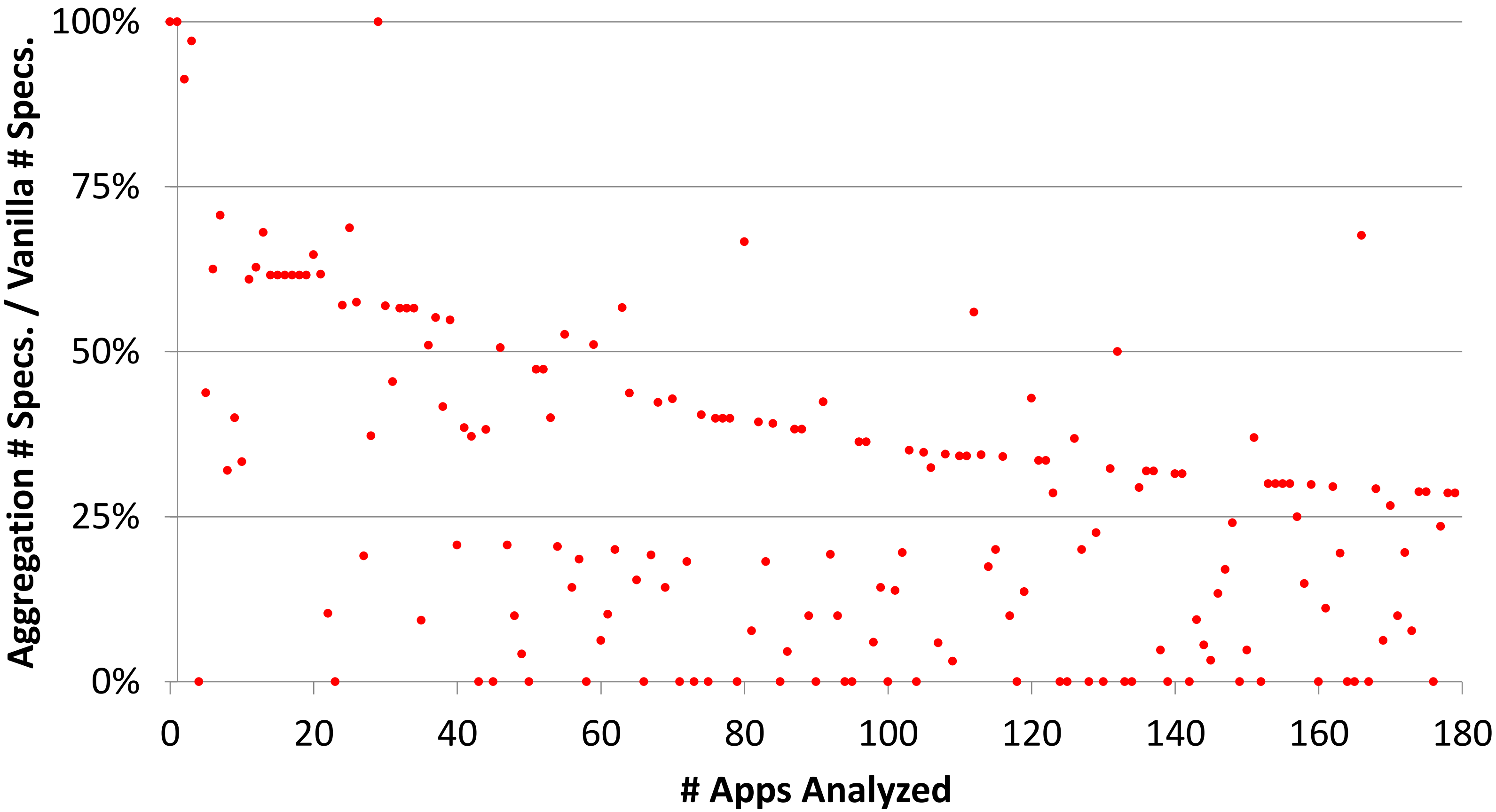
# Benefits of Aggregation



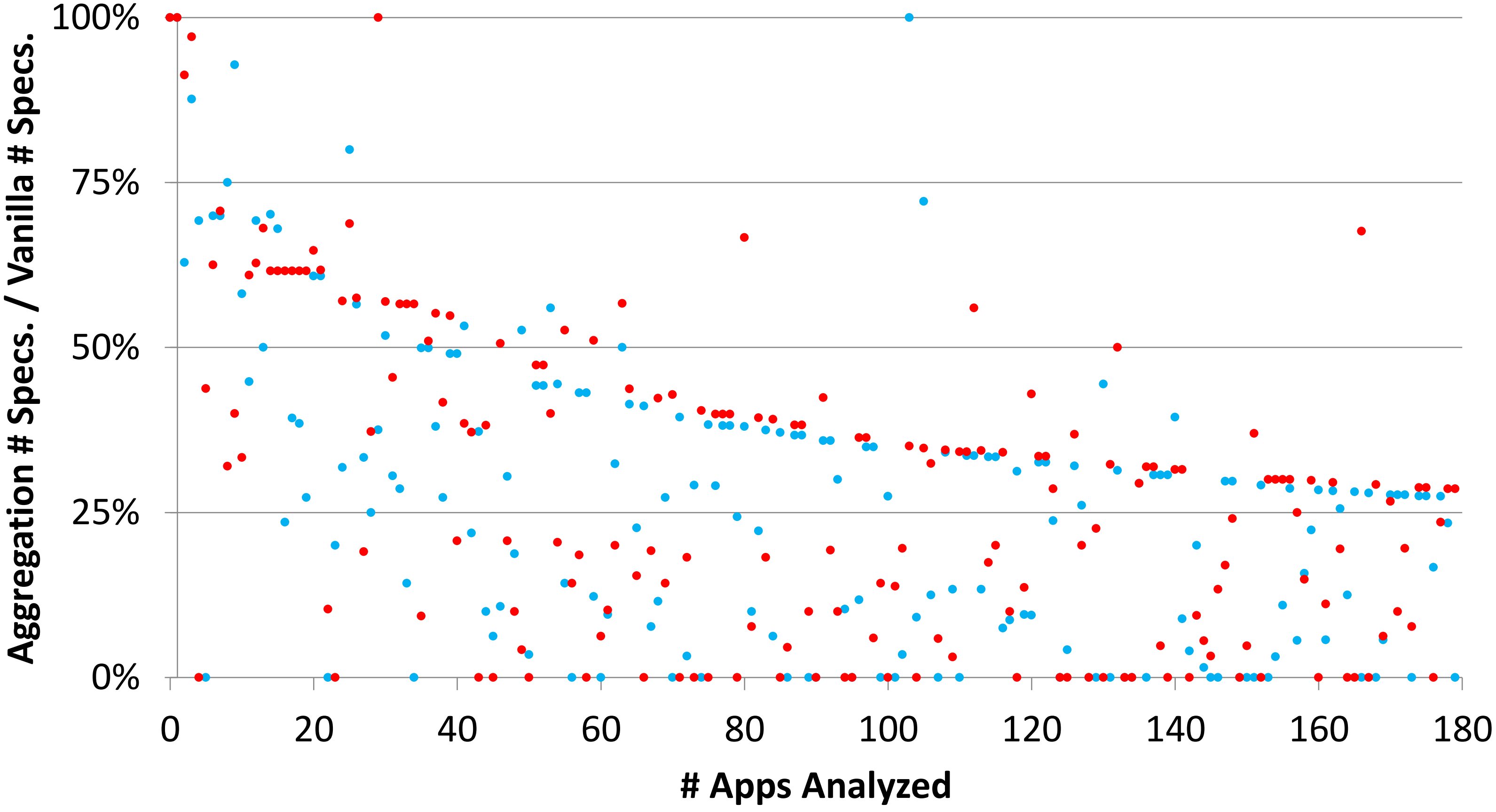
# Benefits of Aggregation



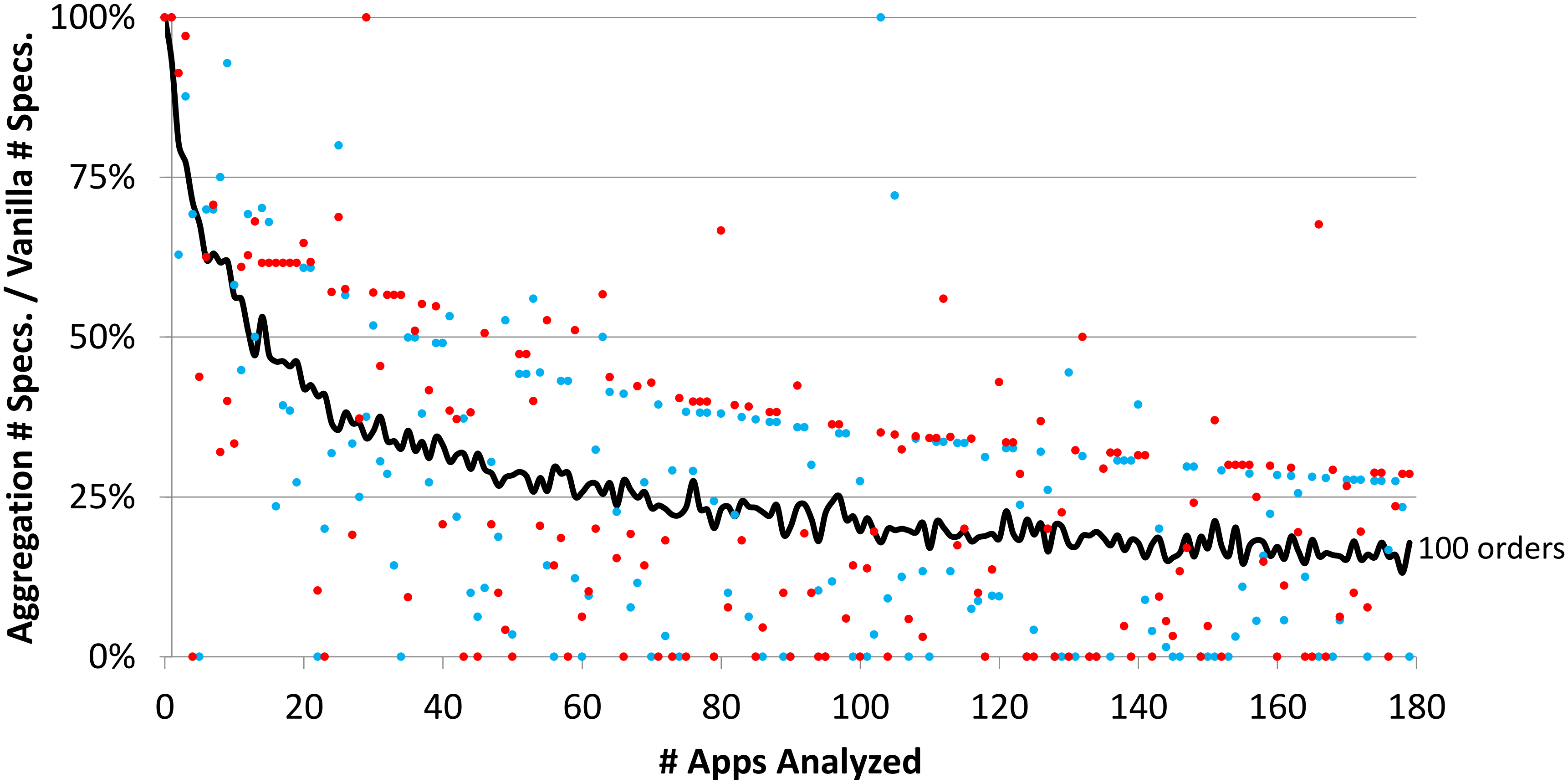
# Benefits of Aggregation



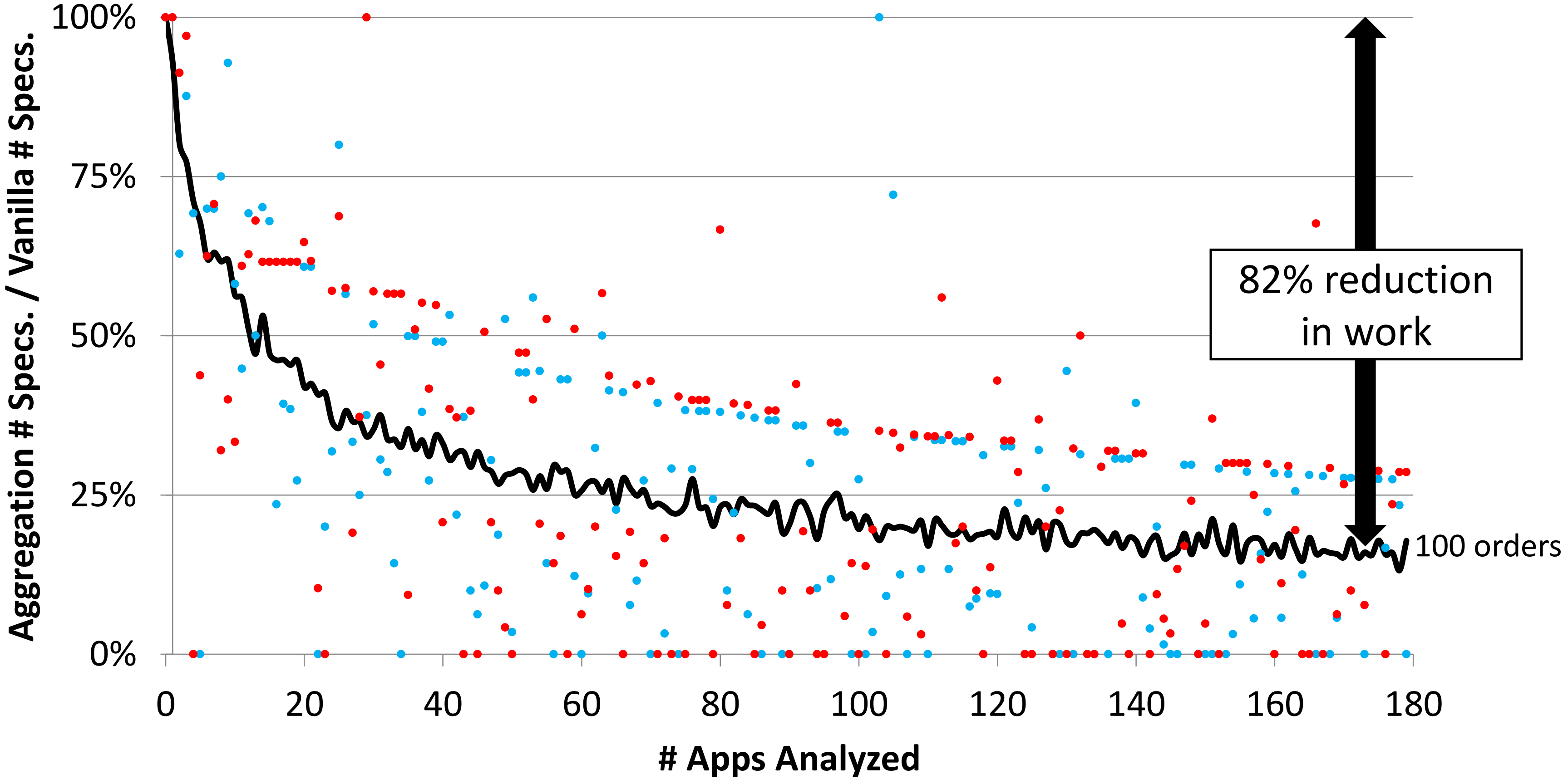
# Benefits of Aggregation



# Benefits of Aggregation



# Benefits of Aggregation



# Conclusions

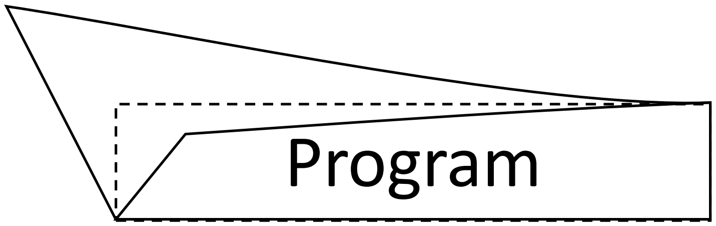
- Approach for analyzing partial programs
  - Step 1: Worst-case analysis (soundness)
  - Step 2: Specification inference
  - Interactive refinement (precision)
- Inferred Android framework specifications
  - $\approx 4\times$ workload compared to oracle
  - Further 82% reduction with aggregation

# References

- H. Zhu, T. Dillig, I. Dillig. Automated inference of library specifications for source-sink property verification. In APLAS, 2013.
- G. Ammons, R. Bodík, J. Larus. Mining specifications. In POPL, 2002.
- J. W. Nimmer, M. D. Ernst. Automatic generation of program specifications. In ISSTA, 2002.
- T. Kremenek, P. Twohey, G. Back, A. Ng, D. Engler. From uncertainty to belief: inferring the specification within. In OSDI, 2006.
- N. Beckman, A. Nori. Probabilistic, modular and scalable inference of typestate specifications. In PLDI, 2011.
- B. Livshits, A. V. Nori, S. K. Rajamani, A. Banerjee. Merlin: specification inference for explicit information flow problems. In PLDI, 2009.
- S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, P. McDaniel. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In PLDI, 2014.
- D. Knuth. A generalization of Dijkstra’s algorithm. In Information Processing Letters, 6(1):1-5, 1977.
- T. Reps. Program analysis via graph reachability. In ILPS, 1997.
- M. Sridharan, D. Gopan, L. Shan, R. Bodik. Demand-driven points-to analysis for Java. In OOPSLA, 2005.

# Questions?

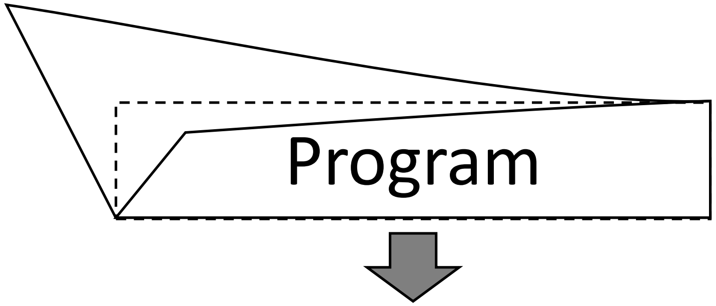
# Specification Inference



**[Zhu 2013] approach:**

1) Over-approximate
2) Specification inference

# Specification Inference



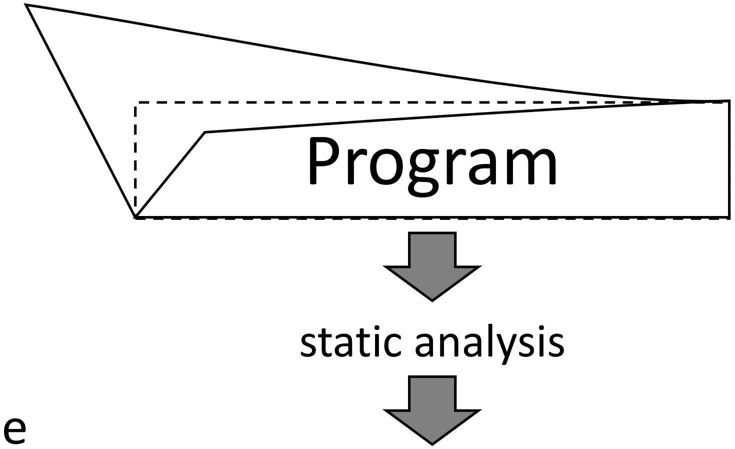
static analysis

**[Zhu 2013] approach:**

1) Over-approximate

2) Specification inference

# Specification Inference



Program

static analysis

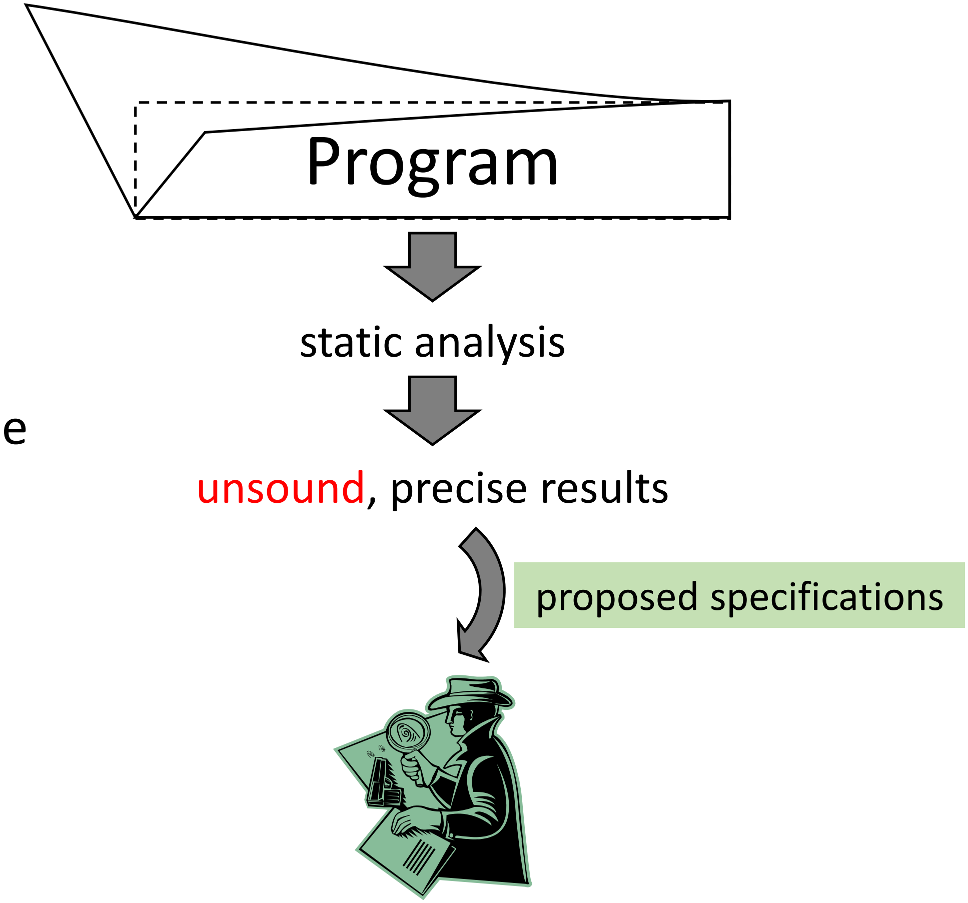
unsound, precise results

**[Zhu 2013] approach:**

1) Over-approximate

2) Specification inference

# Specification Inference



Program

static analysis

unsound, precise results

proposed specifications

**[Zhu 2013] approach:**

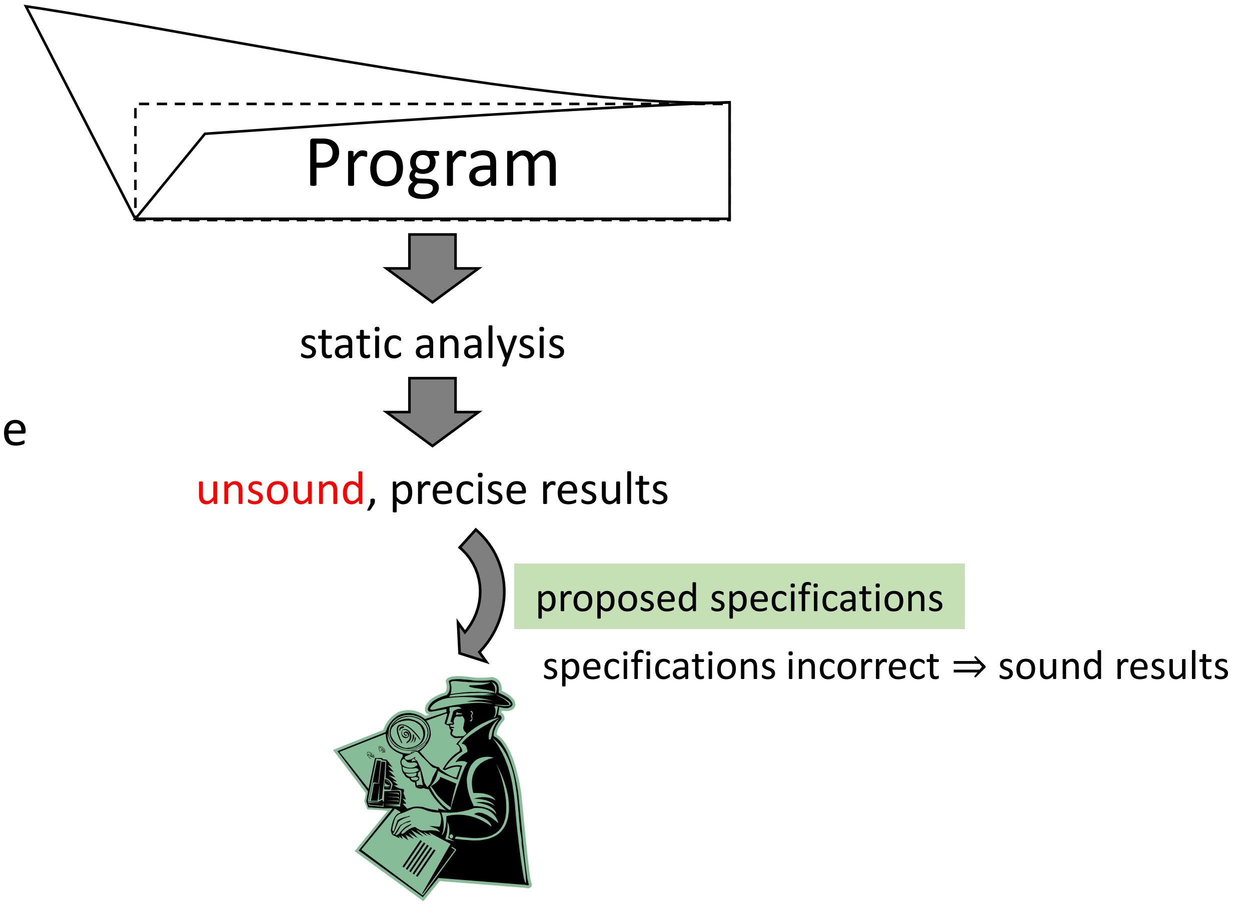
1) Over-approximate

2) Specification inference

# Specification Inference

**[Zhu 2013] approach:**

1) Over-approximate
2) Specification inference



Program

static analysis

unsound, precise results

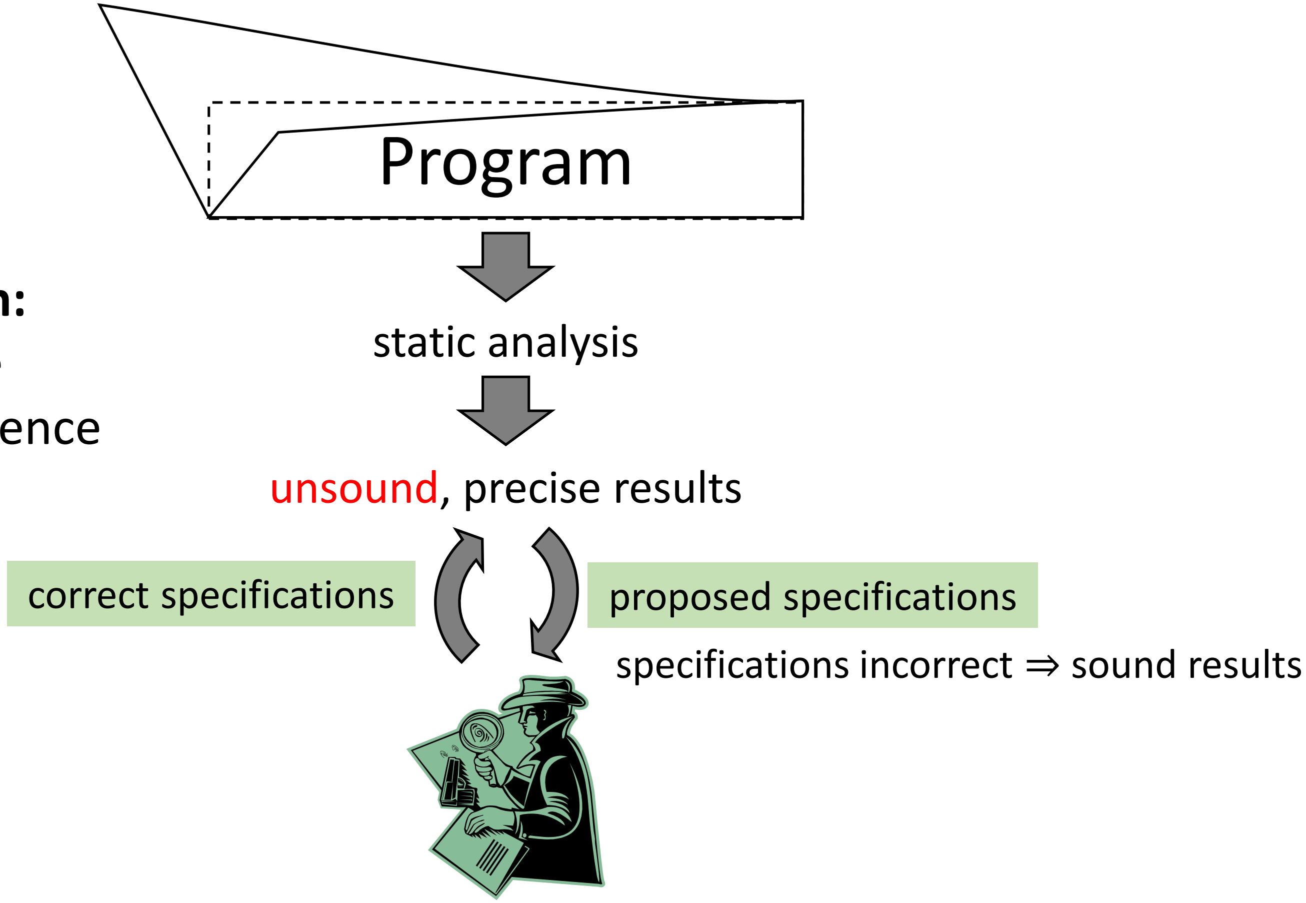
proposed specifications

specifications incorrect ⇒ sound results

# Specification Inference

**[Zhu 2013] approach:**

1) Over-approximate
2) Specification inference



Program

static analysis

unsound, precise results

correct specifications

proposed specifications

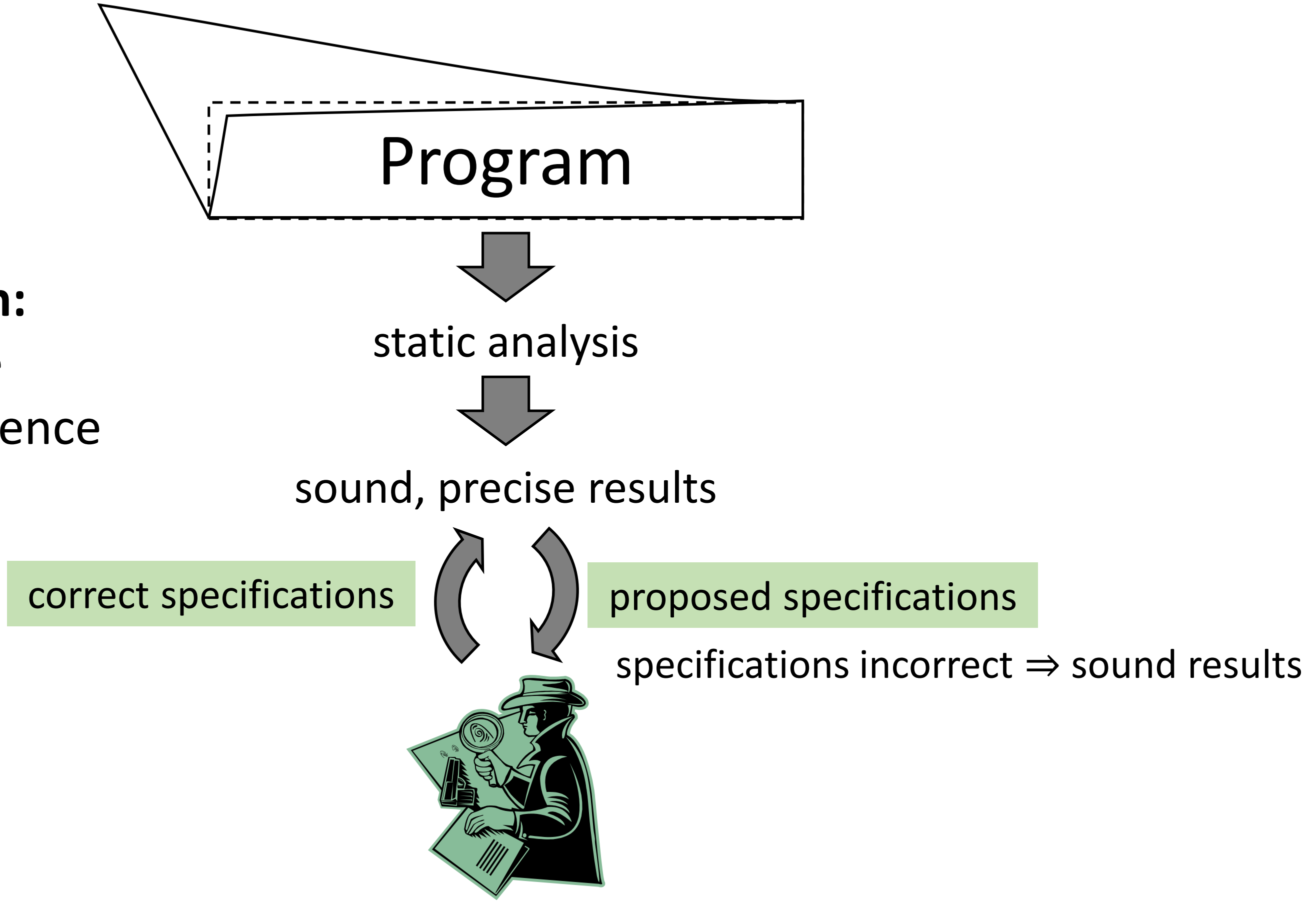
specifications incorrect ⇒ sound results

# Specification Inference

**[Zhu 2013] approach:**

1) Over-approximate

2) Specification inference



Program

static analysis

sound, precise results

correct specifications

proposed specifications

specifications incorrect ⇒ sound results