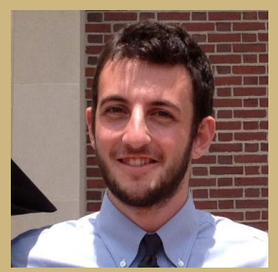
Safe Stream-Based Programming with Refinement Types



Benno Stein
University of Colorado
benno.stein@colorado.edu

Lazaro Clapp
Uber Technologies, Inc.
lazaro@uber.com

Manu Sridharan
Uber Technologies, Inc.
msridhar@uber.com

Bor-Yuh Evan Chang
University of Colorado
evan.chang@colorado.edu

UBER

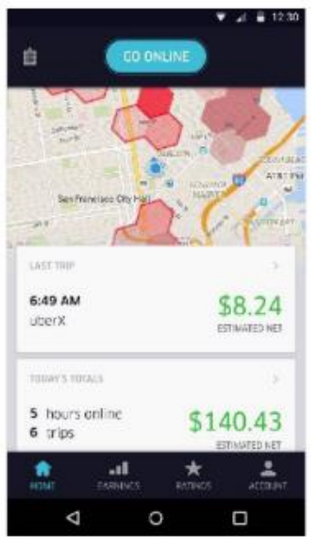
CUPLV

ASE '18
September 6, 2018

Mobile app reliability is *crucial*

UBER

Rider



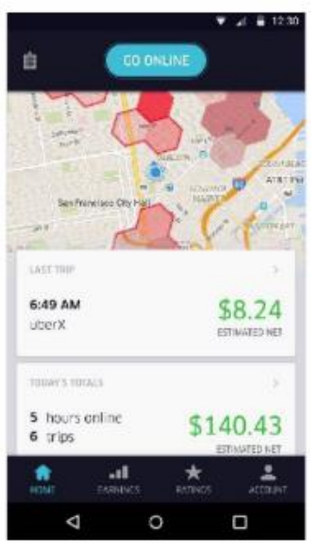
Driver

Eats

Mobile app reliability is *crucial*

UBER

Rider



Driver

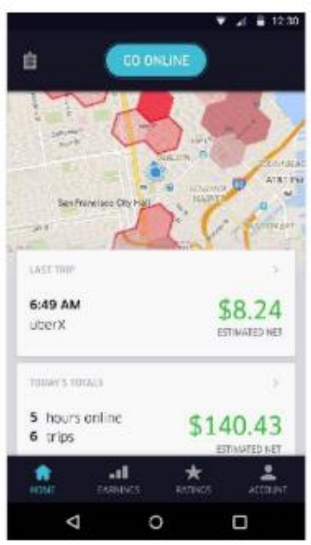
Eats

- Rider crash: can't get home
- Driver crash: can't earn

Mobile app reliability is *crucial*

UBER

Rider



Driver

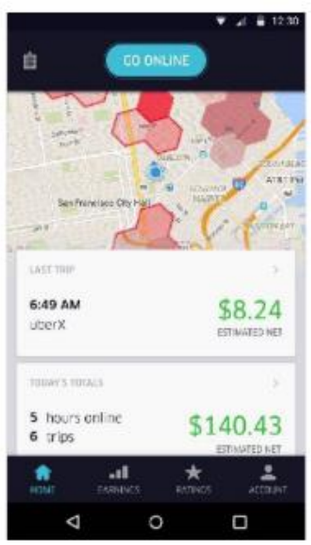
Eats

- Rider crash: can't get home
- Driver crash: can't earn
- Whole business depends on mobile apps

Mobile app reliability is *crucial*

UBER

Rider



Driver

Eats

- Rider crash: can't get home
- Driver crash: can't earn
- Whole business depends on mobile apps
- Patching through third-party app stores is *slow*

Apps are fast-moving, large, and complex

- Hundreds of developers working simultaneously
- Millions of lines of code
- Apps depend upon numerous general-purpose libraries

UI Thread Safety

Don't touch the UI from off the main thread. *Easy enough.*

UI Thread Safety

Don't touch the UI from off the main thread. *Easy enough.*

... not even transitively or through a library.

e.g. `innocuousLookingMethod` calls `foo` calls `bar` calls `uiMethod`

UI Thread Safety

Don't touch the UI from off the main thread. *Easy enough.*

... not even transitively or through a library.

e.g. `innocuousLookingMethod` calls `foo` calls `bar` calls `uiMethod`

... especially when using stream-based programming libraries with complex threading behavior

→ Stream-Based Programming

→ Effect & Thread Type Refinements

→ UI-Thread Safety

→ Evaluation

Reactive Extensions

ReactiveX

"An API for asynchronous programming with observable streams"

Reactive Extensions

ReactiveX

"An API for asynchronous programming with observable streams"

- Create or receive streams of events and data
- Use expressive operators to compose and transform streams
- Subscribe callbacks to streams to perform side effects

Reactive Extensions

ReactiveX

"An API for asynchronous programming with observable streams"

Used by:

Applied Duality · Microsoft · NETFLIX · GitHub · SOUNDCLOUD · Trello · treehouse · SeatGeek · oovoo · Couchbase · futurice · O.C.TANNER · UBER · airbnb

- Create or receive streams of events and data
- Use expressive operators to compose and transform streams
- Subscribe callbacks to streams to perform side effects

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

```
Observable<...> carLocationData = ... ;
```

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

```
Observable<...> carLocationData = ... ;

carLocationData

  .filter( car -> /* car has no passenger */ )
```

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
.observeOn(AndroidSchedulers.mainThread())
```

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
```

Stream-Based Programming

Reactive Extensions (ReactiveX) example:

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(
    car -> { /* display car on map */   },
    err -> { /* render error message */ });
```

→ Stream-Based Programming

→ Effect & Thread Type Refinements

→ UI-Thread Safety

→ Evaluation

Effects & Refinement Types

Function types typically only encode input and output:

$$\tau_{in} \rightarrow \tau_{out}$$

Effects & Refinement Types

Function types typically only encode input and output:

$$\tau_{in} \rightarrow \tau_{out}$$

Effect types refine function types by their side-effects:

$$\tau_{in} \rightarrow_{e} \tau_{out}$$

Effects & Refinement Types

Function types typically only encode input and output:

$$\tau_{in} \rightarrow \tau_{out}$$

Effect types refine function types by their side-effects:

e.g. UI access, network I/O, heavy computation

Effects

```
// java.lang.Math
int max(int x, int y) {...}

// android.widget.Button
void setText(String text) {...}

// com.example.MyApp
void foobar() {...}

// some obscure Android library
void poorlyDocumentedMethod() {...}
```

Effects

```
// java.lang.Math
@SafeEffect int max(int x, int y) {...}

// android.widget.Button
void setText(String text) {...}

// com.example.MyApp
void foobar() {...}

// some obscure Android library
void poorlyDocumentedMethod() {...}
```

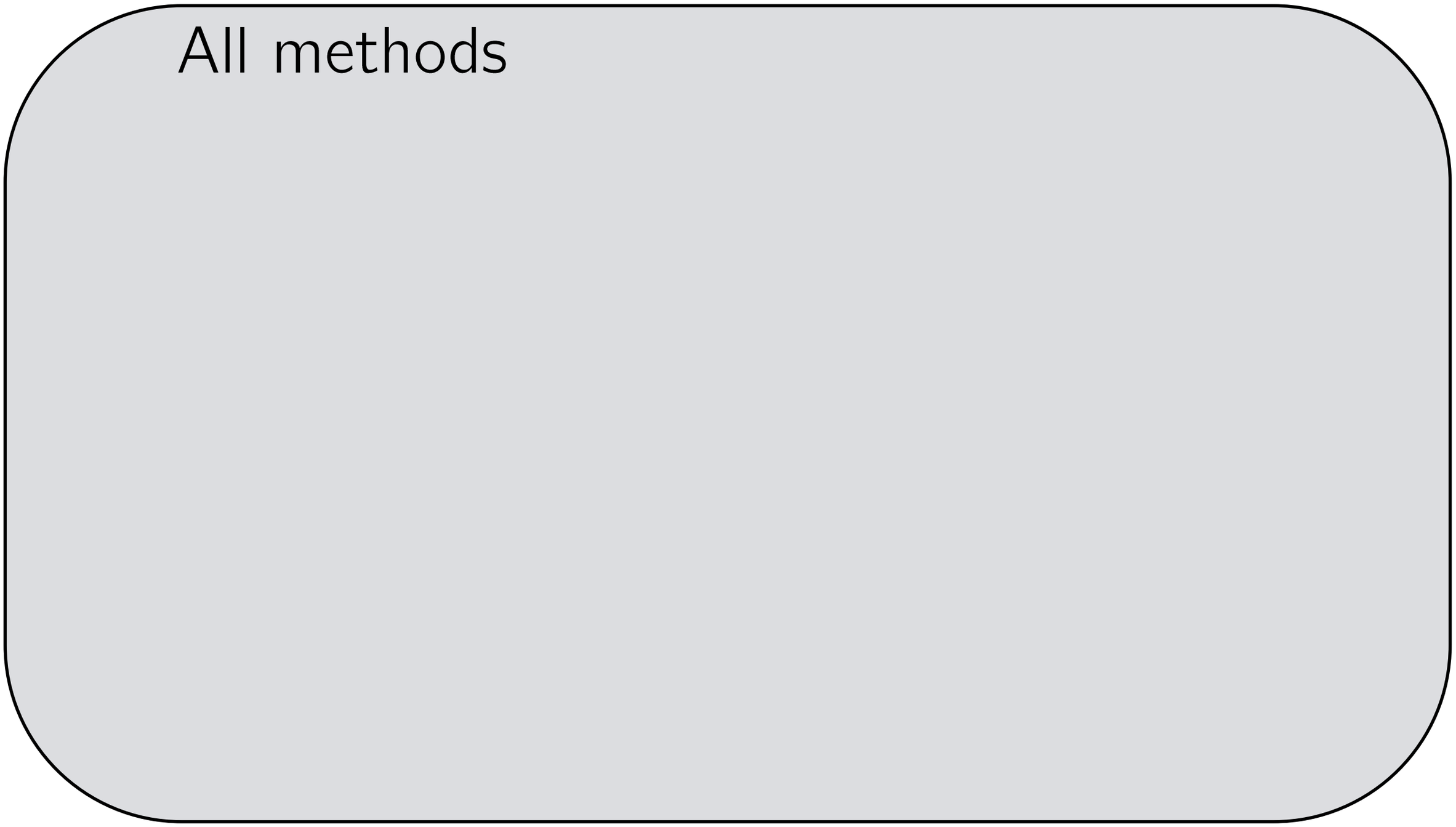
Effects

```
// java.lang.Math
@SafeEffect int max(int x, int y) {...}

// android.widget.Button
@UIEffect void setText(String text) {...}

// com.example.MyApp
void foobar() {...}

// some obscure Android library
void poorlyDocumentedMethod() {...}
```

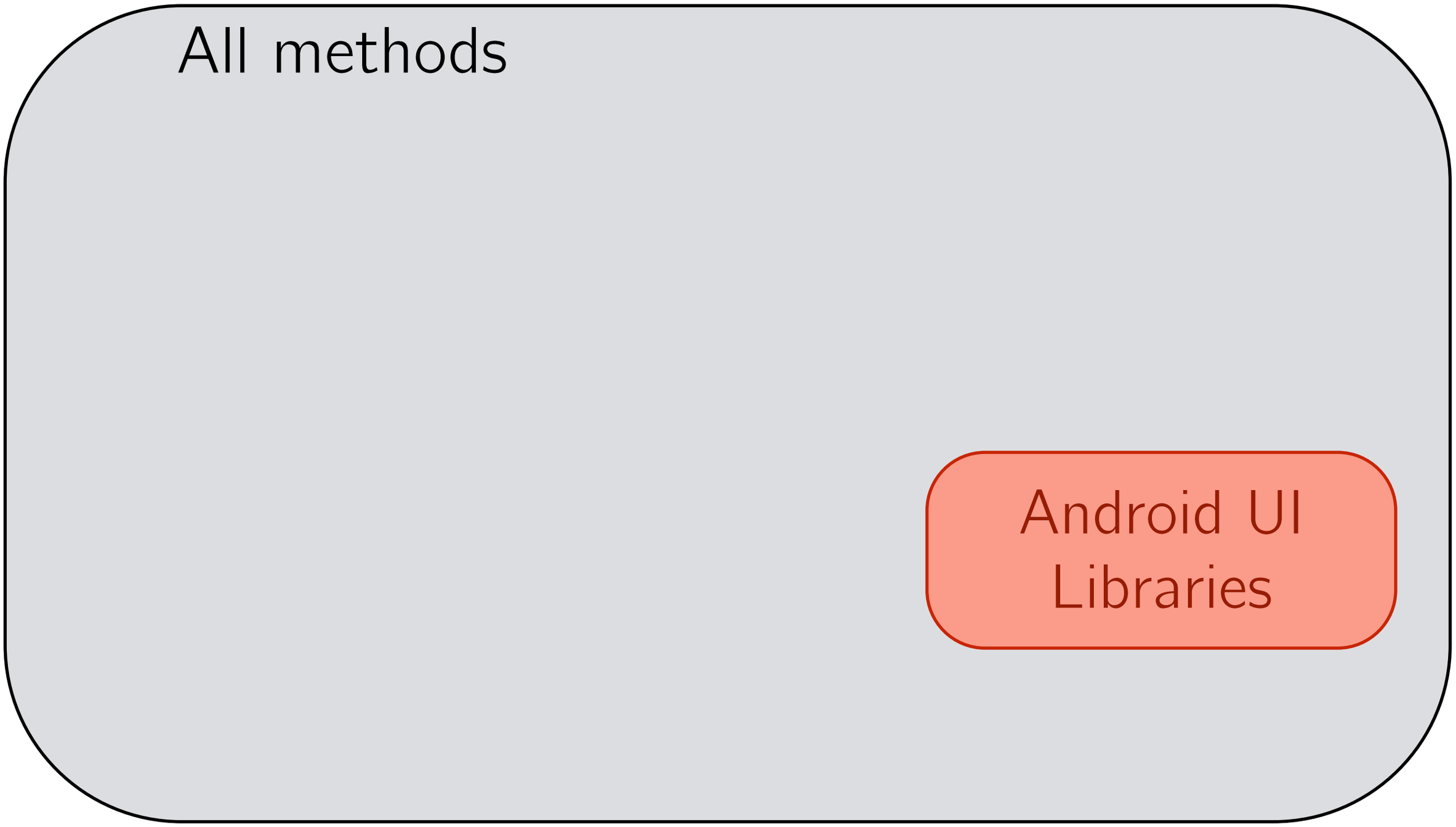
Effects

```
// java.lang.Math
@SafeEffect int max(int x, int y) {...}

// android.widget.Button
@UIEffect void setText(String text) {...}

// com.example.MyApp
????? void foobar() {...}

// some obscure Android library
????? void poorlyDocumentedMethod() {...}
```

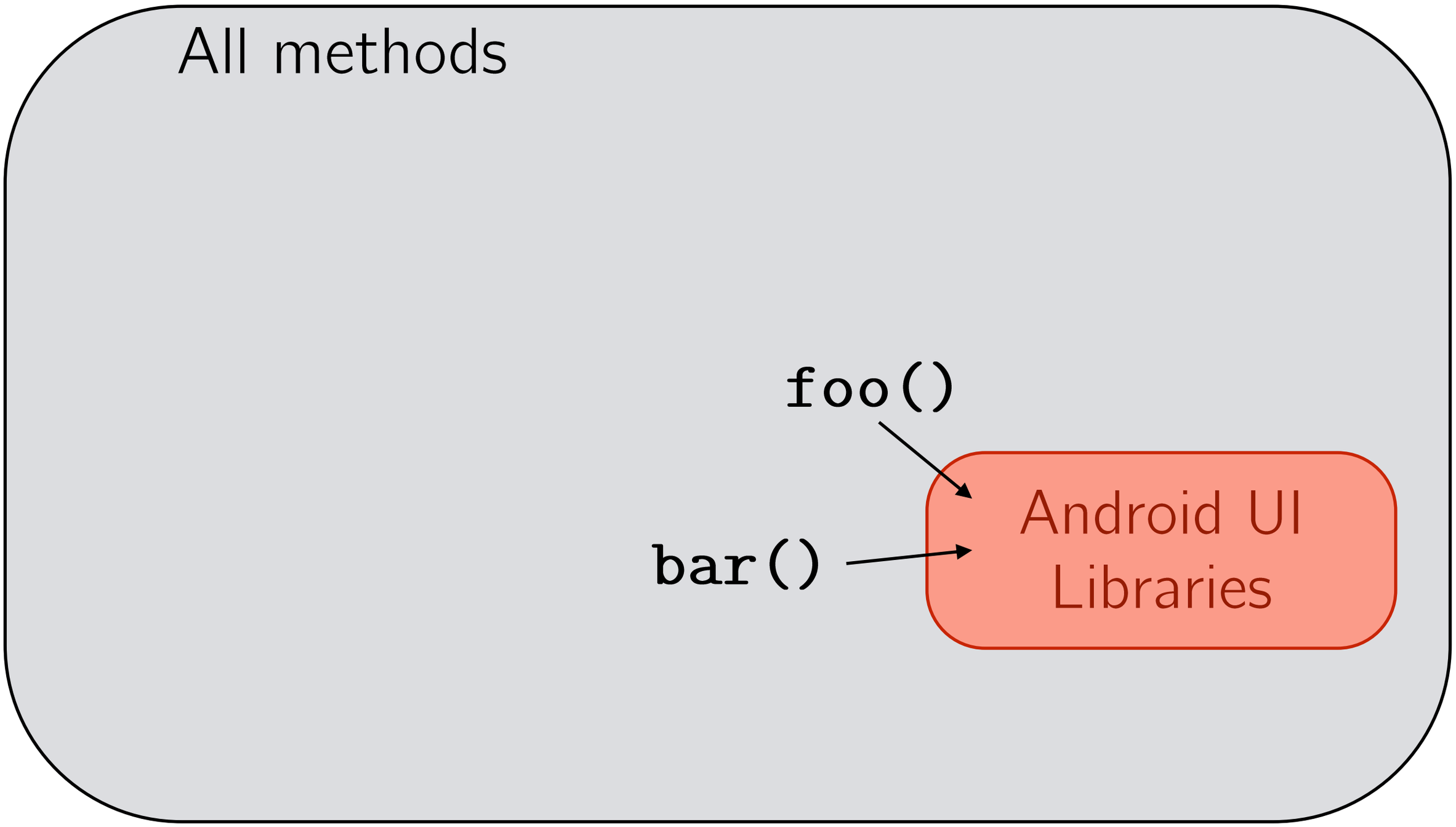
Effect Typing as Call-graph Reachability



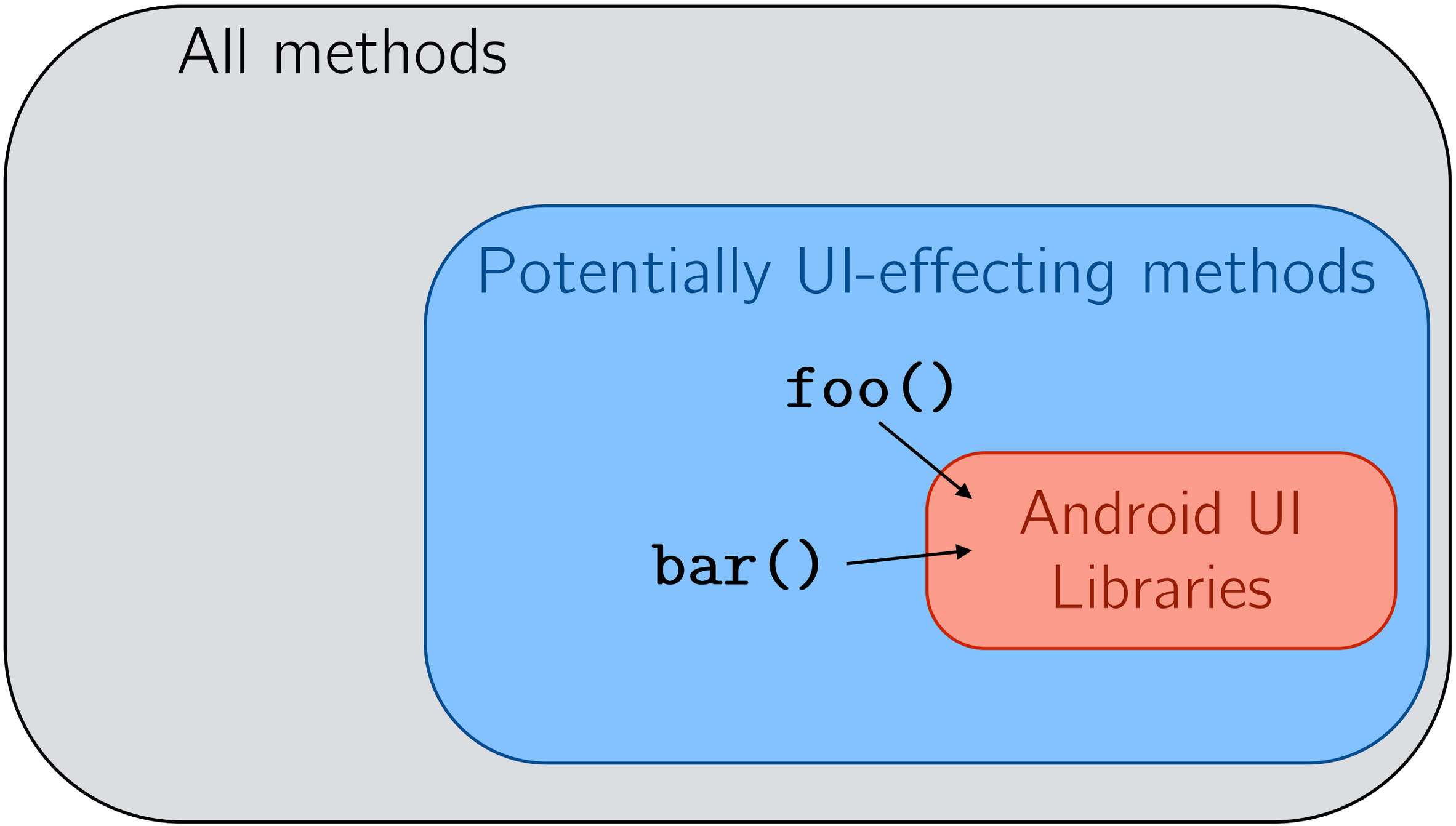
Effect Typing as Call-graph Reachability



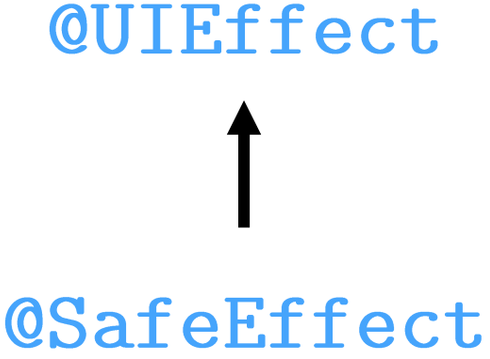
Effect Typing as Call-graph Reachability



Effect Typing as Call-graph Reachability

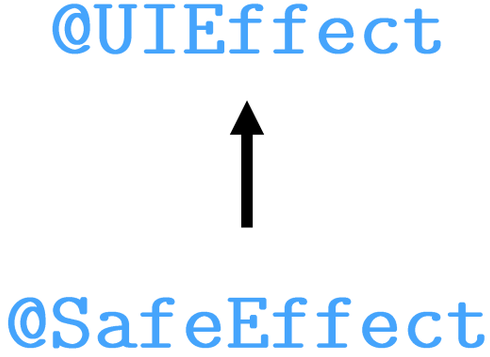


Effect Type Refinements



"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effect Type Refinements

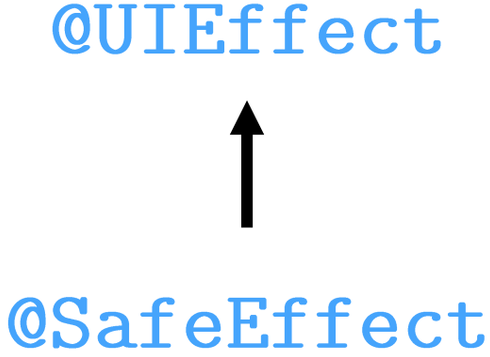


Transitivity:

A method with effect annotation e can **call** a method with effect annotation e' if and only if $e \preccurlyeq e'$

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effect Type Refinements



Transitivity:
A method with effect annotation e can **call** a method with effect annotation e' if and only if $e \preccurlyeq e'$

Inheritance:
A method with effect annotation e can **override** a method with effect annotation e' if and only if $e \preccurlyeq e'$

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effects alone are insufficient

Previous work with effect types handles UI library interfaces with *fixed* threading behavior, such as:

```
runOnUiThread : Runnable -> void
```

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effects alone are insufficient

Previous work with effect types handles UI library interfaces with *fixed* threading behavior, such as:

```
runOnUiThread : Runnable -> void
```

Definitely runs on the UI thread, can safely touch the UI

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effects alone are insufficient

Previous work with effect types handles UI library interfaces with *fixed* threading behavior, such as:

```
runOnUiThread : Runnable -> void
```

Definitely runs on the UI thread, can safely touch the UI

Stream-based interfaces have *dynamic* threading behavior, such as:

```
subscribe : Observable<T> -> Consumer<T> -> void
```

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

Effects alone are insufficient

Previous work with effect types handles UI library interfaces with *fixed* threading behavior, such as:

```
runOnUiThread : Runnable -> void
```

Definitely runs on the UI thread, can safely touch the UI

Stream-based interfaces have *dynamic* threading behavior, such as:

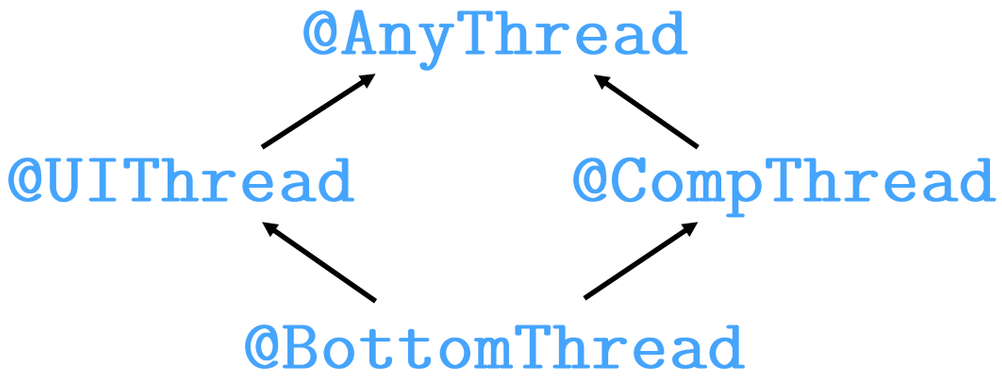
```
subscribe : Observable<T> -> Consumer<T> -> void
```

Runs on a thread determined dynamically by the scheduler of the receiver stream

"Effects for Controlling UI Object Access" Gordon et al., ECOOP '13

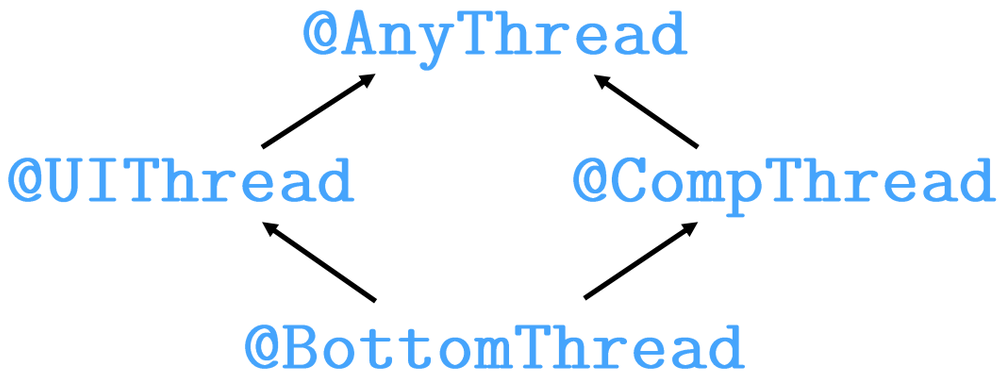
Thread Type Refinement

Type Lattice:



Thread Type Refinement

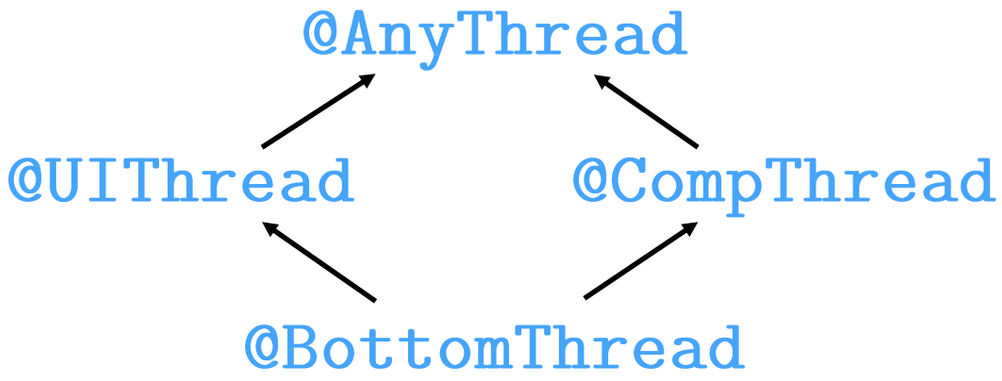
Type Lattice:



Example stream function types:

Thread Type Refinement

Type Lattice:

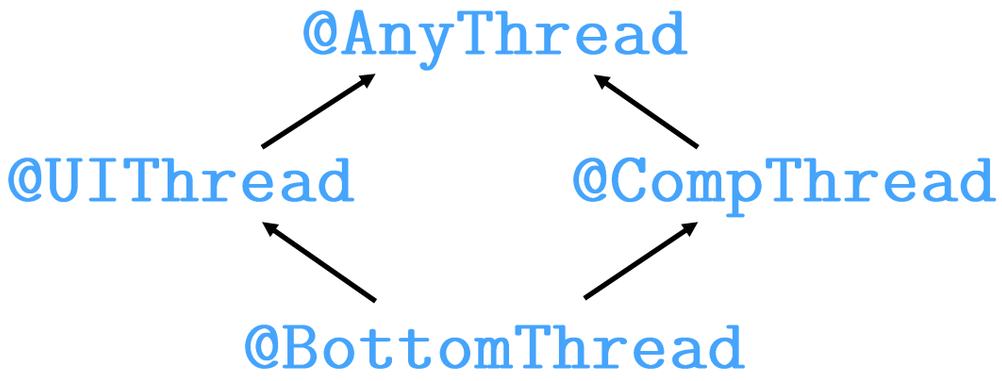


Example stream function types:

```
filter :
 @PolyThread Observable<T> -> Predicate<T> -> @PolyThread Observable<T>
```

Thread Type Refinement

Type Lattice:

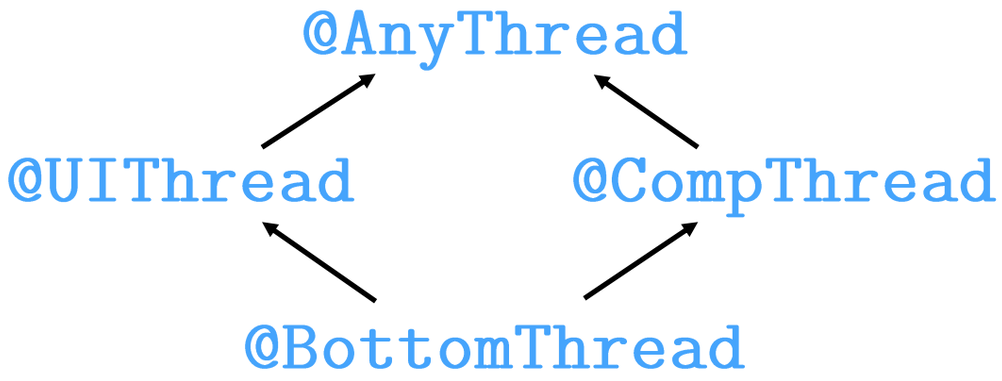


Example stream function types:

```
filter :
 @PolyThread Observable<T> -> Predicate<T> -> @PolyThread Observable<T>


delay :
 @AnyThread Observable<T> -> int -> TimeUnit -> @CompThread Observable<T>
```

Thread Type Refinement

Type Lattice:



Example stream function types:

```
filter :
 @PolyThread Observable<T> -> Predicate<T> -> @PolyThread Observable<T>


delay :
 @AnyThread Observable<T> -> int -> TimeUnit -> @CompThread Observable<T>


observeOn :
 @AnyThread Observable<T>
  -> @PolyThread Scheduler -> @PolyThread Observable<T>
```

→ Stream-Based Programming

→ Effect & Thread Type Refinements

→ UI Thread Safety

→ Evaluation

UI Thread Safety

A stream-based program is *guaranteed* never to access the UI from a non-UI thread if `@UIEffect` callbacks are only subscribed to `@UIThread` streams.

Example Revisited

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(
    car -> { /* display car on map   */ },
    err -> { /* render error message */ });
```

Example Revisited

@AnyThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Example Revisited

@AnyThread

@AnyThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Example Revisited

@AnyThread

@AnyThread

@UIThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Example Revisited

```
delay : @AnyThread Observable<T>
            -> int -> TimeUnit
                -> @CompThread Observable<T>
```

@AnyThread

@AnyThread

@UIThread

@SafeEffect

@UIEffect

@UIEffect

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

Example Revisited

```
delay : @AnyThread Observable<T>
            -> int -> TimeUnit
                -> @CompThread Observable<T>
```

@AnyThread

@AnyThread

@UIThread

@CompThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Example Revisited

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(100, TimeUnit.MILLISECONDS)
  .subscribe(
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@CompThread

@UIEffect

@UIEffect

Example Revisited

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .observeOn(AndroidSchedulers.mainThread())
  .delay(                                t.MILLISECONDS)
  .subscribe(
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

ERROR!

@CompThread

@UIEffect

@UIEffect

Fixed Example

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(
    car -> { /* display car on map   */ },
    err -> { /* render error message */ });
```

Fixed Example

@AnyThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Fixed Example

@AnyThread

@AnyThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Fixed Example

@AnyThread

@AnyThread

@CompThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Fixed Example

@AnyThread

@AnyThread

@CompThread

@UIThread

```
Observable<...> carLocationData = ... ;

carLocationData                                    @SafeEffect
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(                          @UIEffect
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

@UIEffect

Fixed Example

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(
    car -> { /* display car on map   */ },
    err -> { /* render error message */ });
```

@UIThread

@UIEffect

@UIEffect

Fixed Example

```
Observable<...> carLocationData = ... ;

carLocationData
  .filter( car -> /* car has no passenger */ )
  .delay(100, TimeUnit.MILLISECONDS)
  .observe                                dulers.mainThread())
  .subscribe(
    car -> { /* display car on map    */ },
    err -> { /* render error message */ });
```

NO ERROR!

@UIThread

@UIEffect

@UIEffect

→ Stream-Based Programming

→ Effect & Thread Type Refinements

→ UI Thread Safety

→ Evaluation

Experiments

RQ1: *Is the typechecker practical and easy-to-use?*

- Manual annotation burden is small
- Compile-time performance cost is low
- Error messages and warnings are understandable

RQ2: *Does the typechecker find real bugs and help fix them?*

- Stream-based threading bugs exist in practice
- Typechecker identifies them successfully
- Checked programs are reliably bug-free

Test Corpora

Open Source Android apps:

Java applications on GitHub

... that import ReactiveX `AndroidSchedulers`,

... have at least 15 “stars”

... and had been indexed recently.

Uber Case Study:

- Deployed in production for **Driver** and **Eats** apps.
  - Over 500k LoC in total

Usability

RQ1: *Is the typechecker practical and easy-to-use?*

- Manual annotation burden is small
- Compile-time performance cost is low
- Error messages and warnings are understandable by real developers

App	KLoC	Annotations	Reported Errors	Compile Time (sec.)
ForPDA	33.0	197	4	27
chat-sdk-android	34.6	102	6	21
trust-wallet-android	8.8	27	2	17
arch-components-date	0.7	2	0	8
MVPArms	6.3	59	1	9
rxbus	3.3	12	0	3
SmartReceiptsLibrary	39.9	217	16	30
OpenFoodFacts	14.9	146	4	41
Averages	17.7	95	4.1	19.5

Usability

RQ1: *Is the typechecker practical and easy-to-use?*

- Manual annotation burden is small ✔
- Compile-time performance cost is low
- Error messages and warnings are understandable by real developers

App	KLoC	Annotations	Reported Errors	Compile Time (sec.)
ForPDA	33.0	197	4	27
chat-sdk-android	34.6	102	6	21
trust-wallet-android	8.8	27	2	17
arch-components-date	0.7	2	0	8
MVPArms	6.3	59	1	9
rxbus				3
SmartReceiptsLib				30
OpenFoodFacts	14.9	146	4	41
Averages	17.7	95	4.1	19.5

One annotation per 186 LoC

Usability

RQ1: *Is the typechecker practical and easy-to-use?*

- Manual annotation burden is small ✔
- Compile-time performance cost is low ✔
- Error messages and warnings are understandable by real developers

App	KLoC	Annotations	Reported Errors	Compile Time (sec.)
ForPDA	33.0	197	4	27
chat-sdk-android	34.6	102	6	21
trust-wallet-android	8.8	27	2	17
arch-components-date	0.7	2	0	8
MVPArms	6.3	59	1	9
rxbus	3.3	12	0	3
SmartReceiptsLibrary	39.9	217	16	30
OpenFoodFacts	14.9	146	4	41
Averages	17.7	95	4.1	19.5

Usability

RQ1: *Is the typechecker practical and easy-to-use?*

- Manual annotation burden is small ✔
- Compile-time performance cost is low ✔
- Error messages and warnings are understandable by real developers ✔

Uber Case Study:

- Over 4000 commits by 176 Uber developers
- One annotation per 178 LoC by Uber developers

Effectiveness

RQ2: *Does the typechecker find real bugs and help fix them?*

- Stream-based threading bugs exist in practice
- Typechecker identifies them successfully
- Checked programs are reliably bug-free

Effectiveness

RQ2: *Does the typechecker find real bugs and help fix them?*

- Stream-based threading bugs exist in practice ✔ ?
- Typechecker identifies them successfully ✔
- Checked programs are reliably bug-free

App	KLoC	Annotations	Reported Errors	Compile Time (sec.)
ForPDA	33.0	197	4	27
chat-sdk-android	34.6	102	6	21
trust-wallet-android	8.8	27	2	17
arch-components-date	0.7	2	0	8
MVPArms	6.3	59	1	9
rxbus	3.3	12	0	3
SmartReceiptsLibrary	39.9	217	16	30
OpenFoodFacts	14.9	146	4	41
Averages	17.7	95	4.1	19.5

Effectiveness

RQ2: *Does the typechecker find real bugs and help fix them?*

- Stream-based threading bugs exist in practice ✔ ?
- Typechecker identifies them successfully ✔
- Checked programs are reliably bug-free

Uber Case Study:

- 41 changes to threading behavior of stream-processing code during initial setup
- 135 additions of **`observeOn(mainThread)`** by developers in response to alarms after initial setup

Effectiveness

RQ2: *Does the typechecker find real bugs and help fix them?*

- Stream-based threading bugs exist in practice ✔
- Typechecker identifies them successfully ✔
- Checked programs are reliably bug-free ✔

Uber Case Study:

- Zero `CalledFromWrongThreadException` crashes in production in checked code!
  - monitoring period of one month
  - non-zero crash rates in unchecked apps

Contributions:

- Refinement type system for stream threads

- Typechecker implementation for Android

- Evaluation on open-source and industrial apps

UBER

CUPLV

Benno Stein
University of Colorado
benno.stein@colorado.edu

ASE ‘18
September 6, 2018