Thread-safety analysis with Infer

Sam Blackshear
Facebook

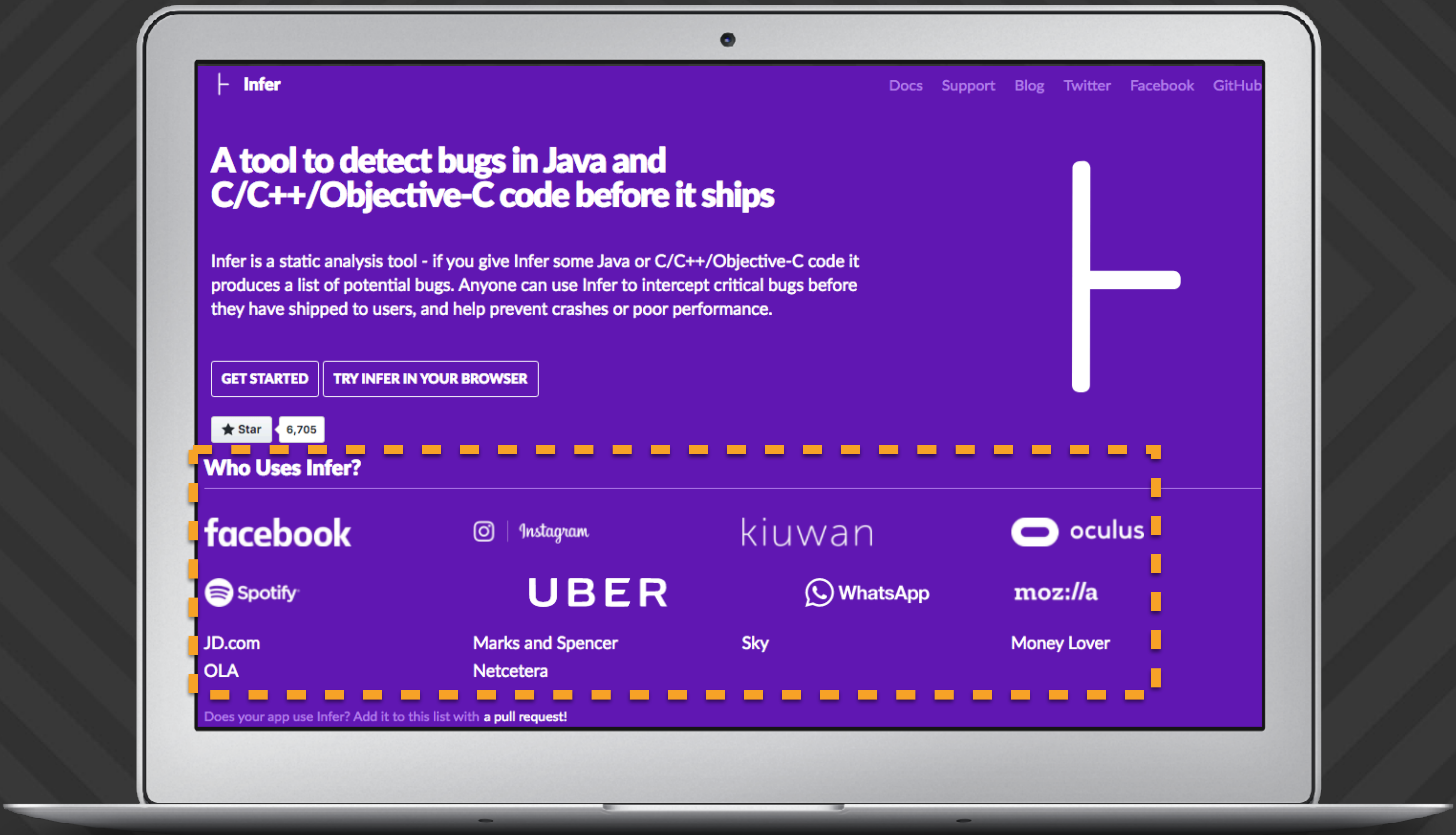
Roadmap

1 | Infer Overview

- Android bug types
- Tool architecture
- Usage at Facebook

2 | Deep dive: thread-safety analysis

- What it does/how it works
- By example: bugs, fixes, annotations
- Current status and future plans



A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

Who Uses Infer?

facebook
Instagram
kiuwan
oculus
Spotify
UBER
WhatsApp
moz://a
JD.com
Marks and Spencer
Sky
Money Lover
OLA
Netcetera

Does your app use Infer? Add it to this list with a pull request!

How do I get Infer?

```
brew install infer
```

github.com/facebook/infer

http://fbinfer.com/docs/getting-started.html

What does Infer find?

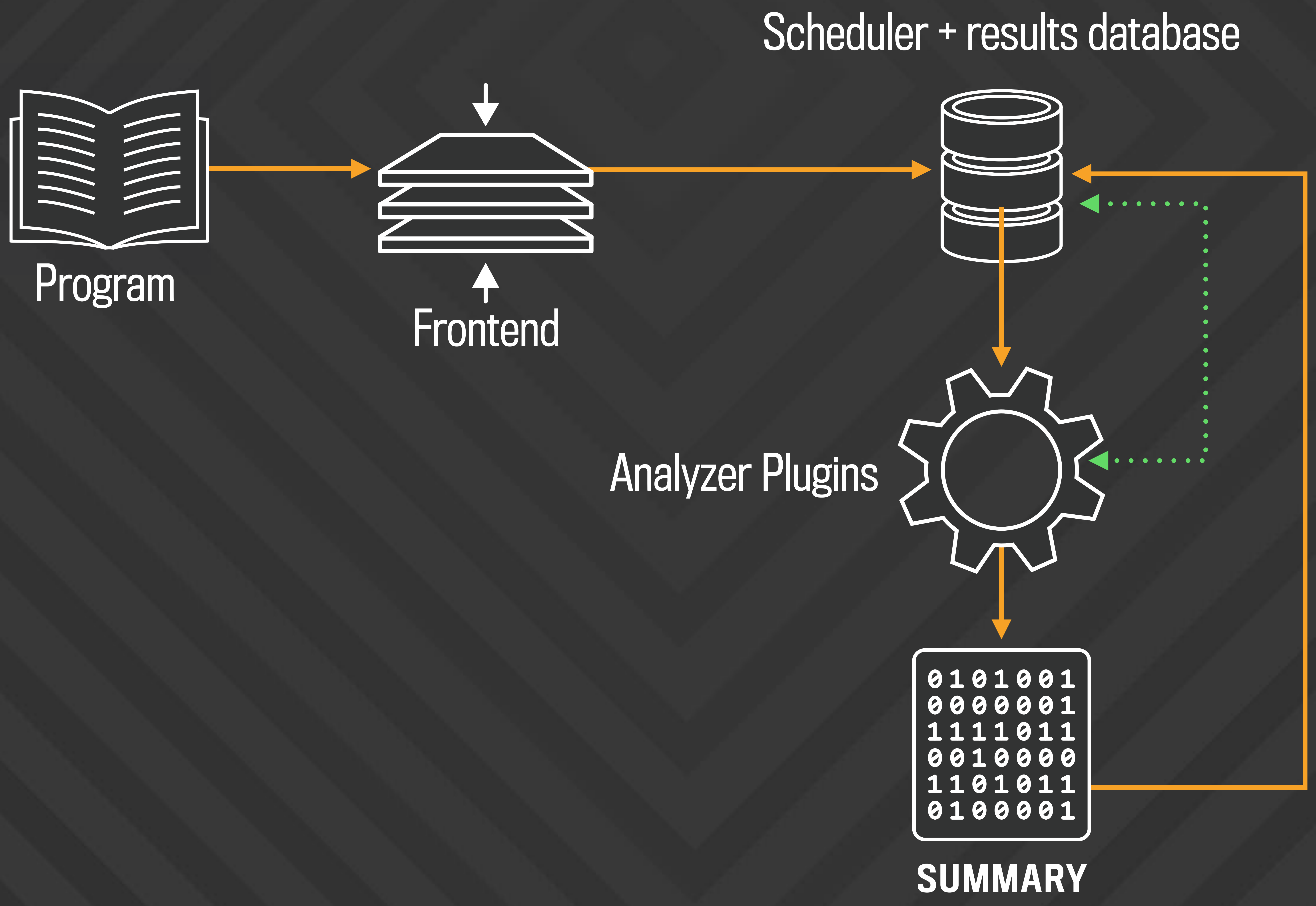
- Old goodness: null dereference, resource leak, context leak, missing lock guard

```
infer -- <your_build_command>
infer -a eradicate -- <your_build_command>
```

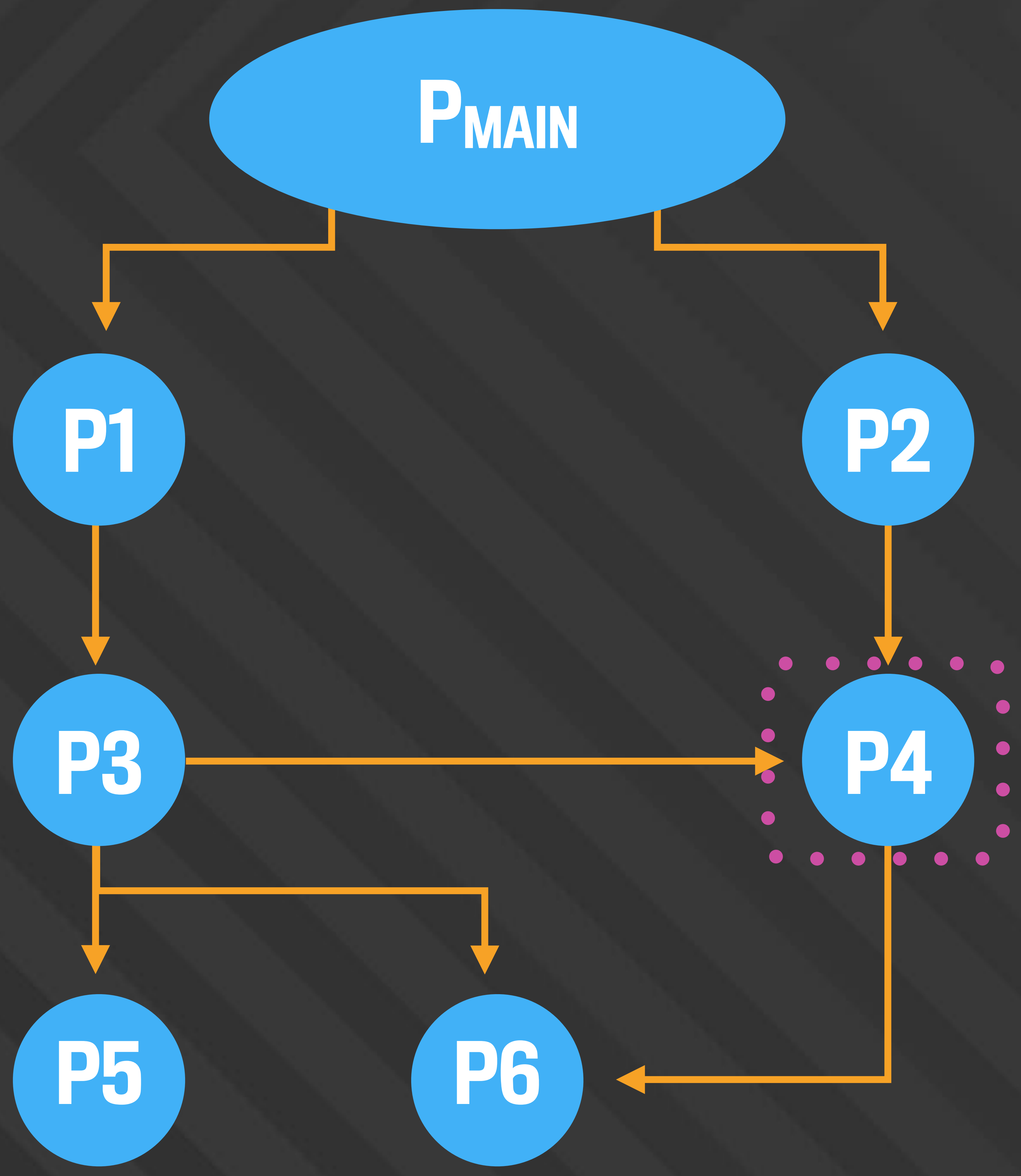
- New hotness: thread-safety, Quandary taint analysis (security bugs), annotation reachability

```
infer -a checkers -- <your_build_command>
```

Recipe for an scalable/extensible analyzer



Recipe for a scalable analyzer: modular/compositional bottom-up analysis

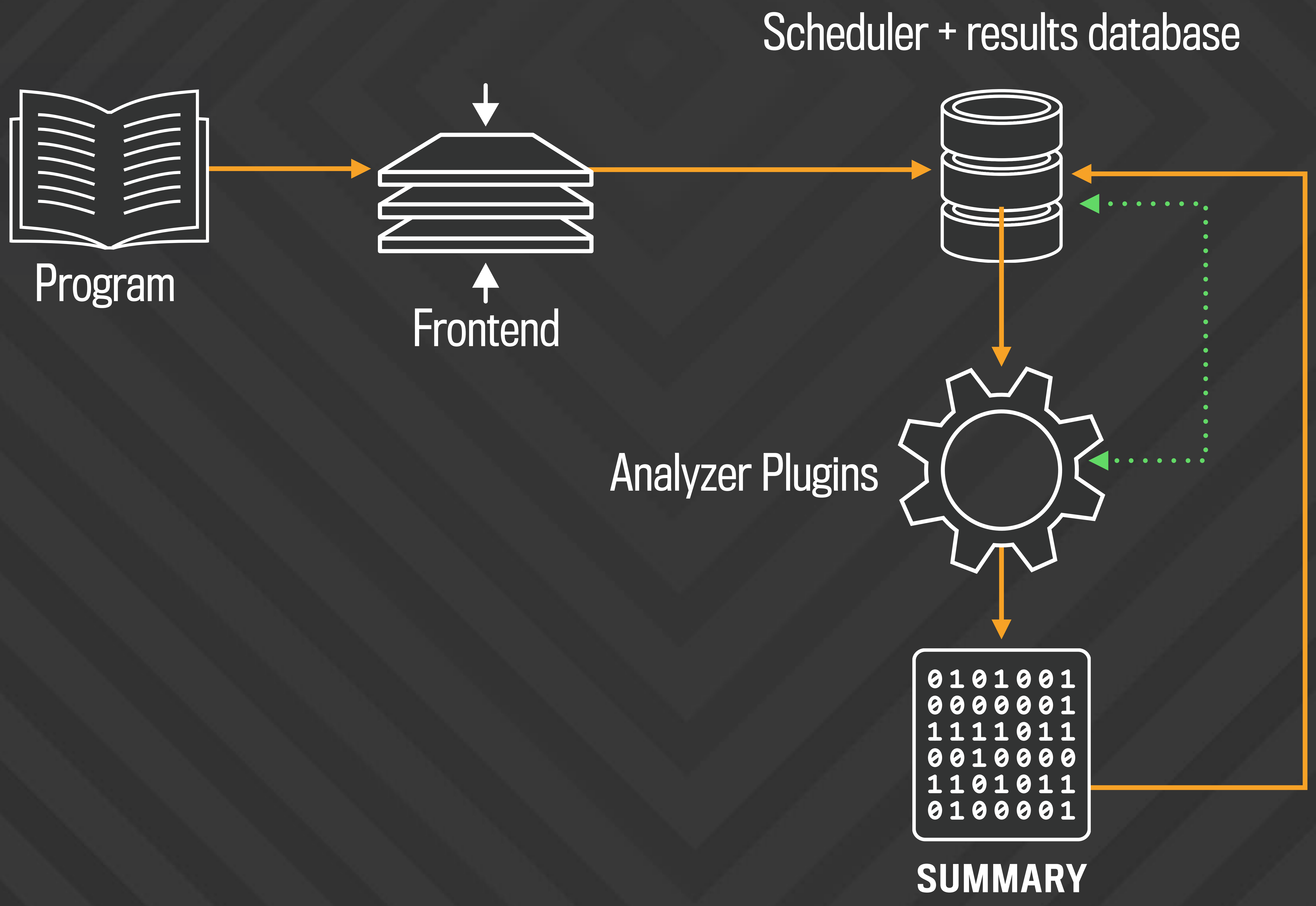


- Will have **summary** for callee P6
- But don't know anything about callers P2, P3
- Analyze P4, compute summary usable in **any** calling context

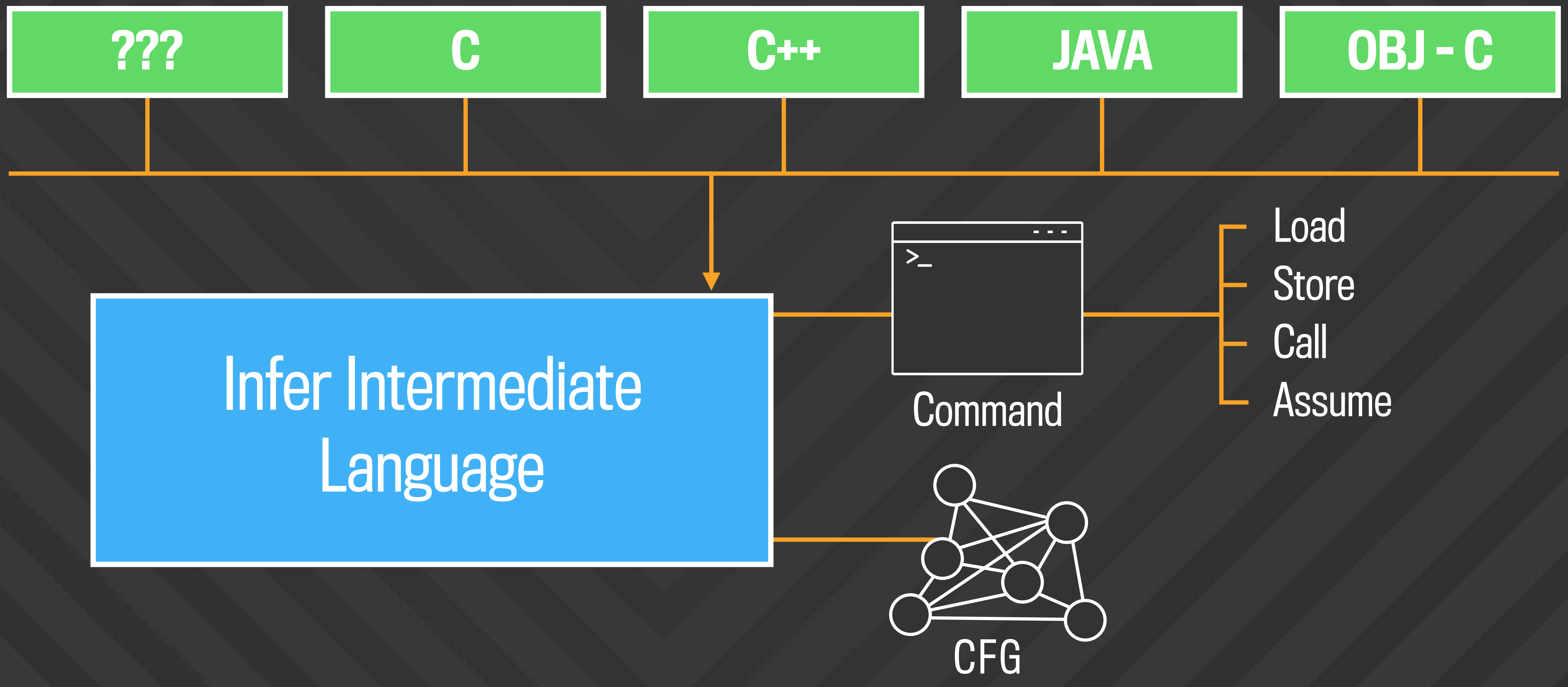
Why modular + compositional matters

- Scalable: linear in the number of procedures
- Incremental: easy to transition from-scratch analysis -> fast diff analysis for analyzing changes only
- Extensible: for new analysis, just need new summary type + intraprocedural analysis to compute it

Recipe for an extensible analyzer



Adding new languages



Roadmap

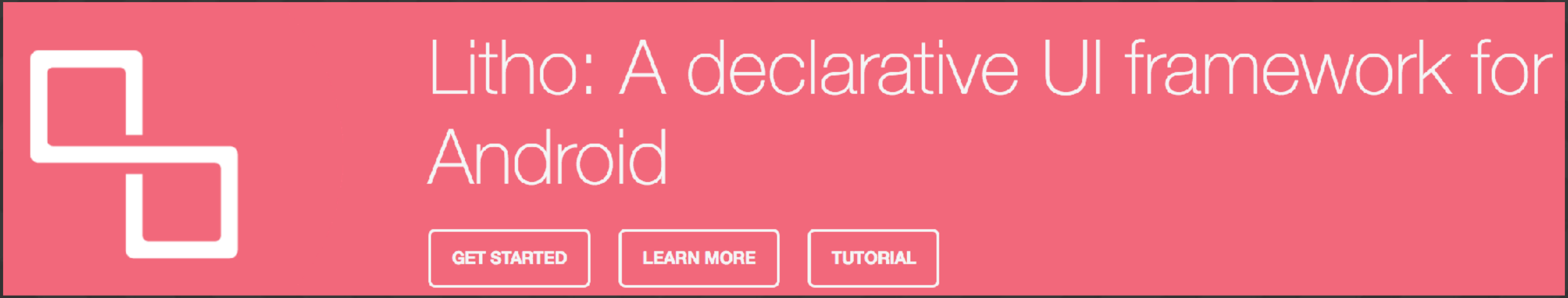
1 | Infer Overview

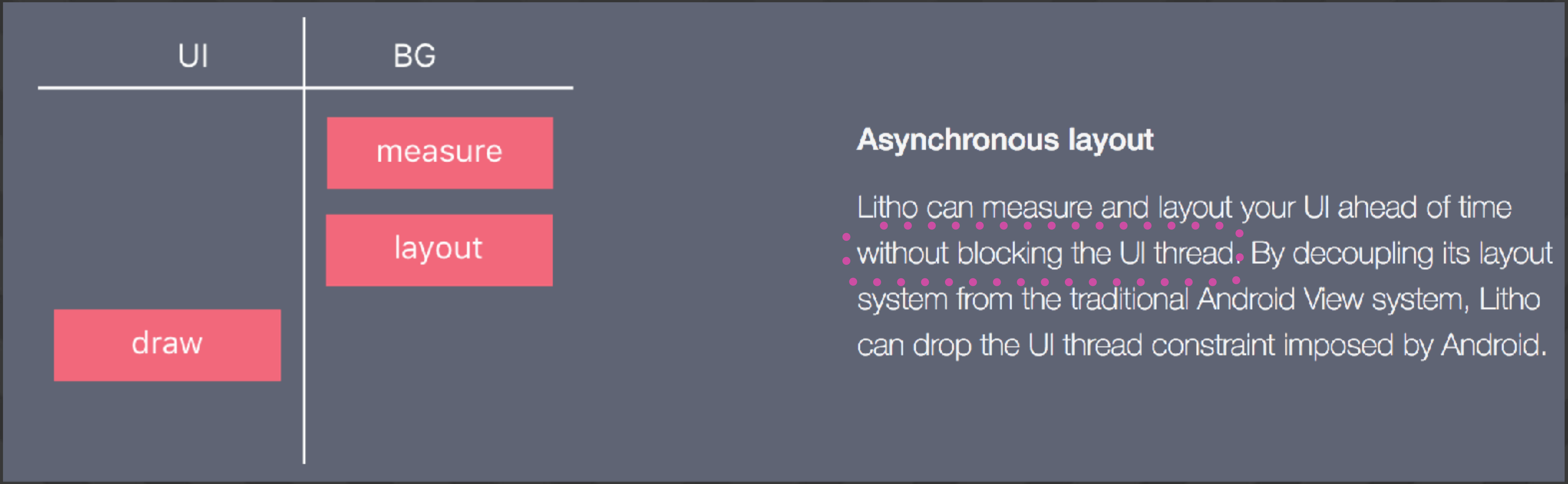
- Bug types
- Architecture
- Usage at Facebook

2 | Deep dive: thread-safety analysis

- What it does/how it works
- By example: bugs, fixes, annotations
- Current status and future plans

Who wants concurrency analysis?

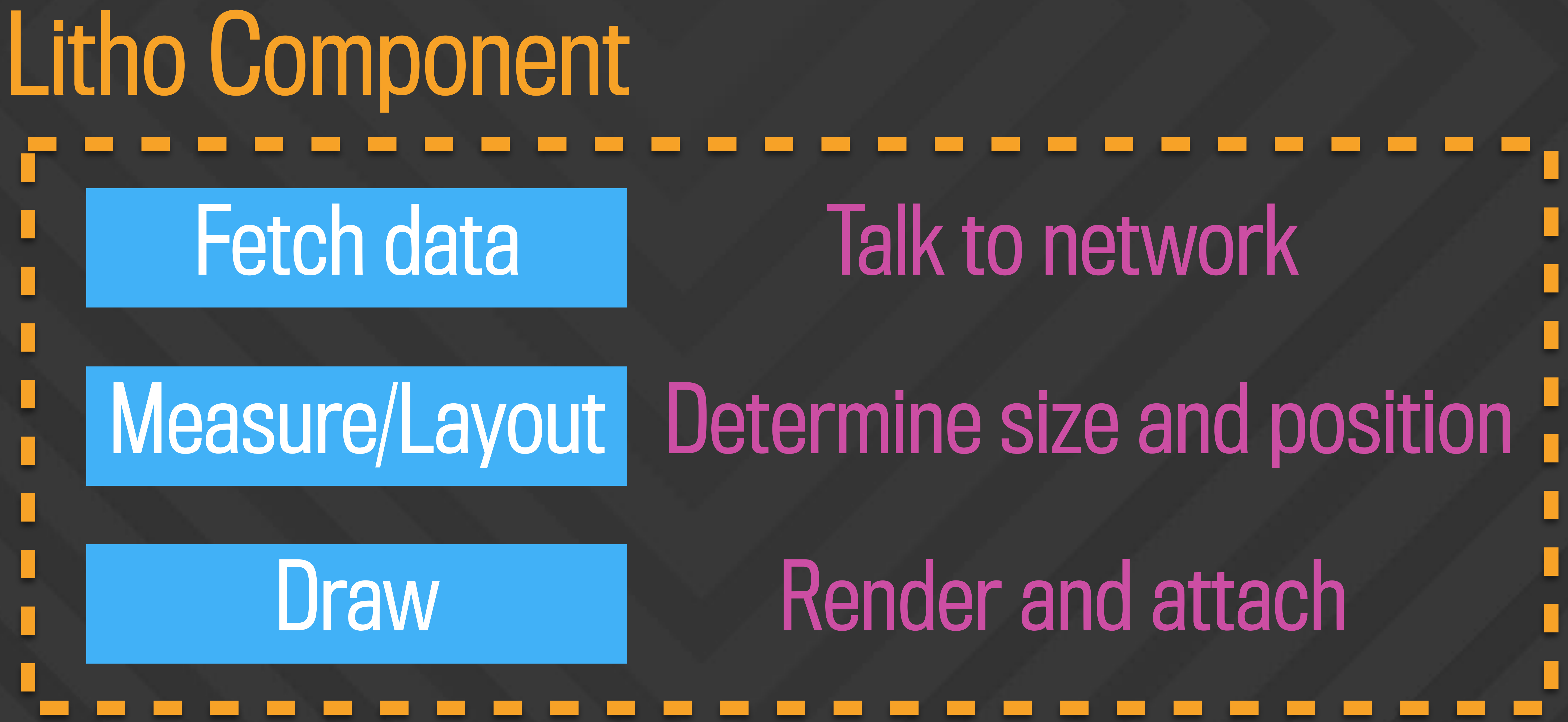

Litho: A declarative UI framework for Android



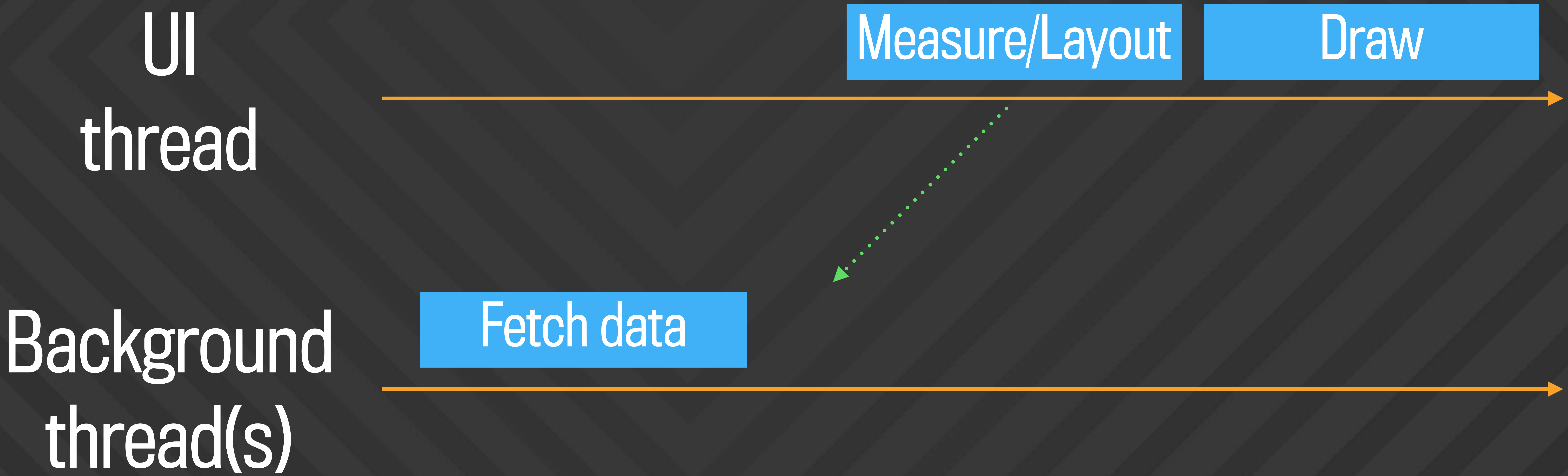
Asynchronous layout

Litho can measure and layout your UI ahead of time without blocking the UI thread. By decoupling its layout system from the traditional Android View system, Litho can drop the UI thread constraint imposed by Android.

Litho: declarative framework for building Android UI

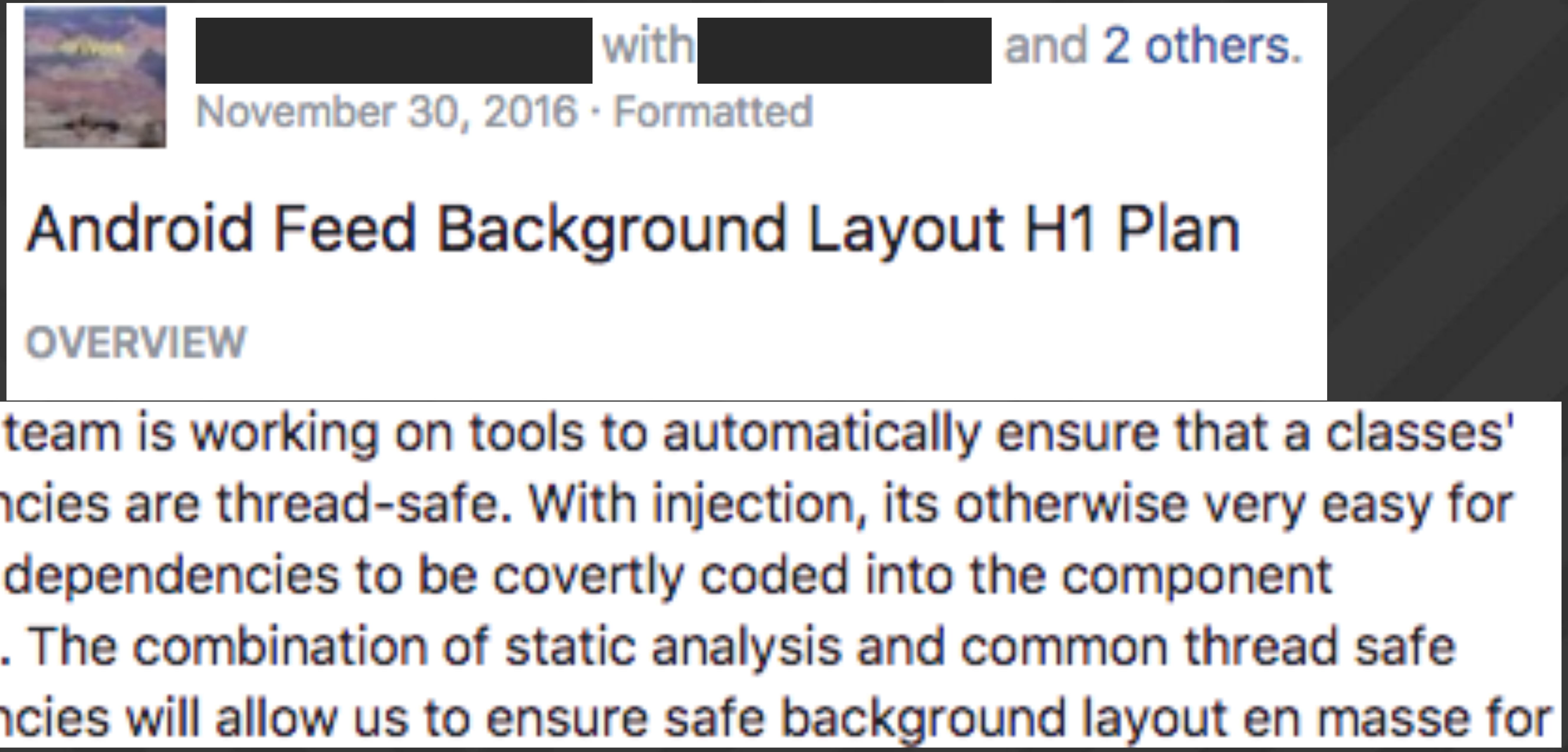


Improve performance by moving layout to background



Measure/Layout step needs to be thread-safe

Motivation for thread-safety analysis: help devs safely speed up News Feed



[redacted] with [redacted] and 2 others.
November 30, 2016 · Formatted

Android Feed Background Layout H1 Plan

OVERVIEW

The Infer team is working on tools to automatically ensure that a classes' dependencies are thread-safe. With injection, its otherwise very easy for non-safe dependencies to be covertly coded into the component hierarchy. The combination of static analysis and common thread safe dependencies will allow us to ensure safe background layout en masse for

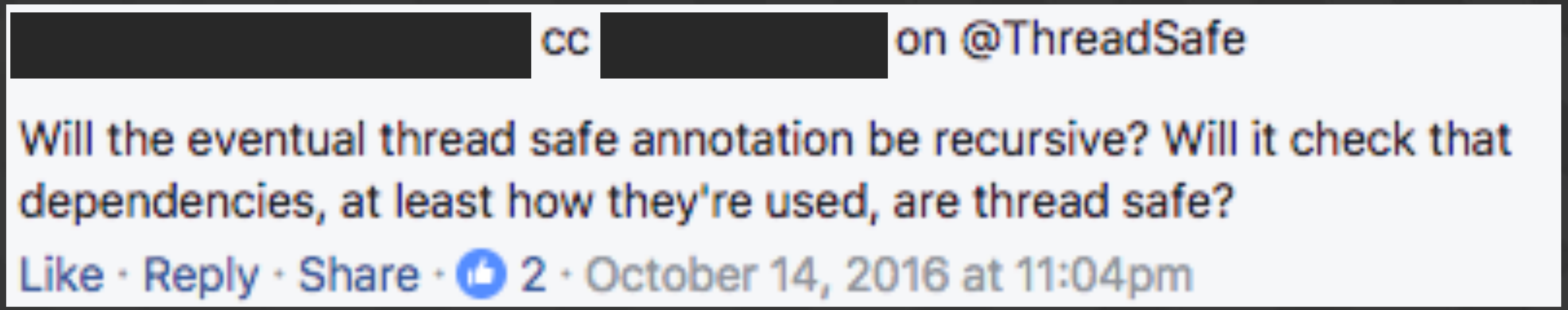
Requirements for thread-safety analysis

Interprocedural

Low annotation burden

Modular

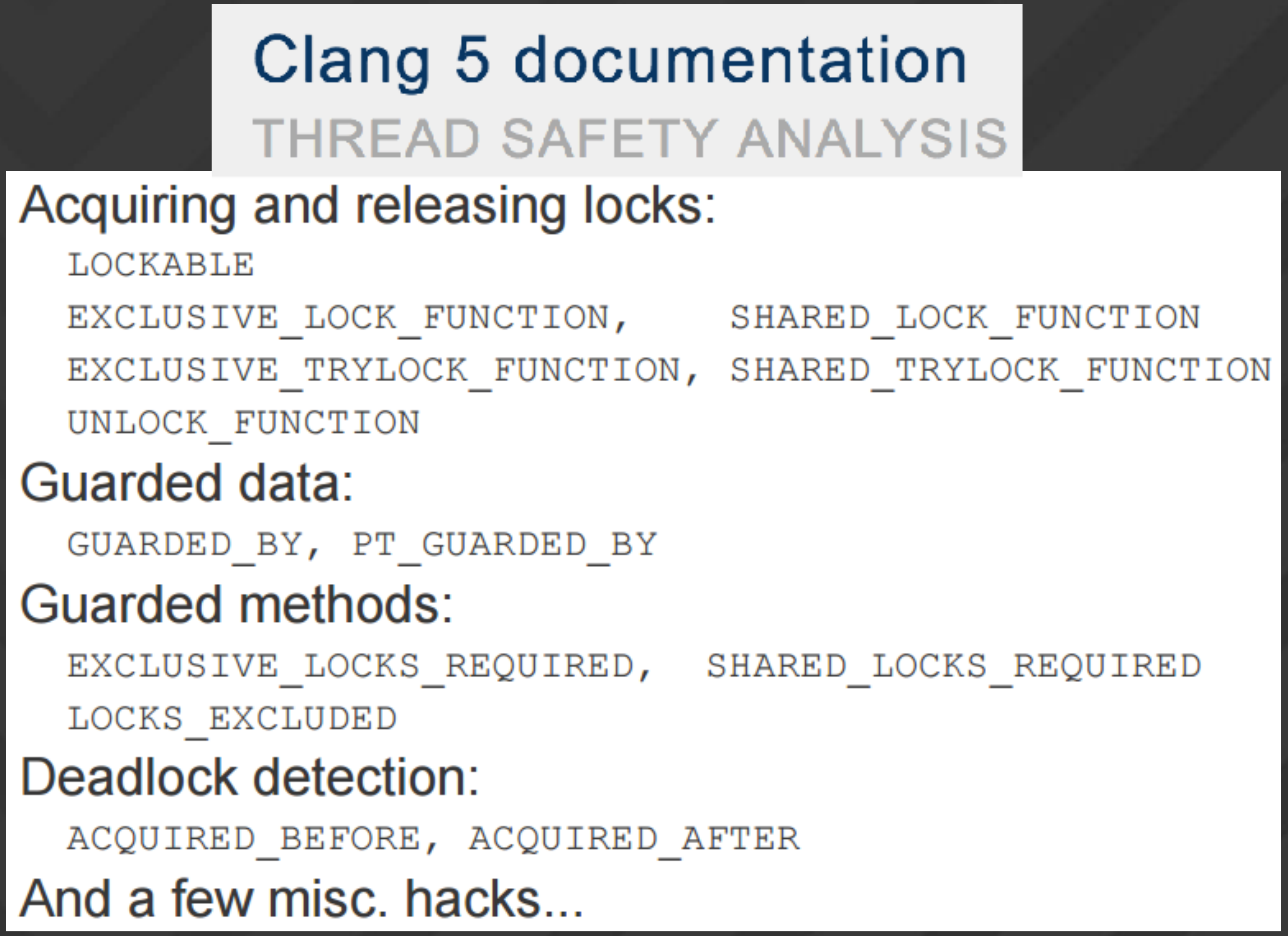
Compositional



[redacted] cc [redacted] on @ThreadSafe

Will the eventual thread safe annotation be recursive? Will it check that dependencies, at least how they're used, are thread safe?

Like · Reply · Share · 2 · October 14, 2016 at 11:04pm



Clang 5 documentation

THREAD SAFETY ANALYSIS

Acquiring and releasing locks:

```
LOCKABLE
EXCLUSIVE_LOCK_FUNCTION,    SHARED_LOCK_FUNCTION
EXCLUSIVE_TRYLOCK_FUNCTION, SHARED_TRYLOCK_FUNCTION
UNLOCK_FUNCTION
```

Guarded data:

```
GUARDED_BY, PT_GUARDED_BY
```

Guarded methods:

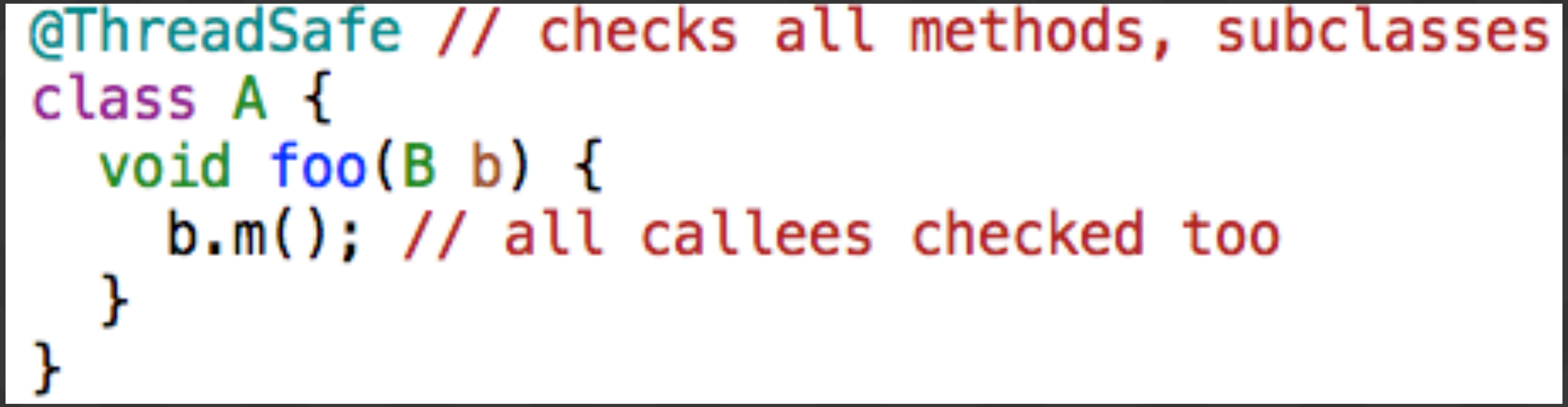
```
EXCLUSIVE_LOCKS_REQUIRED,  SHARED_LOCKS_REQUIRED
LOCKS_EXCLUDED
```

Deadlock detection:

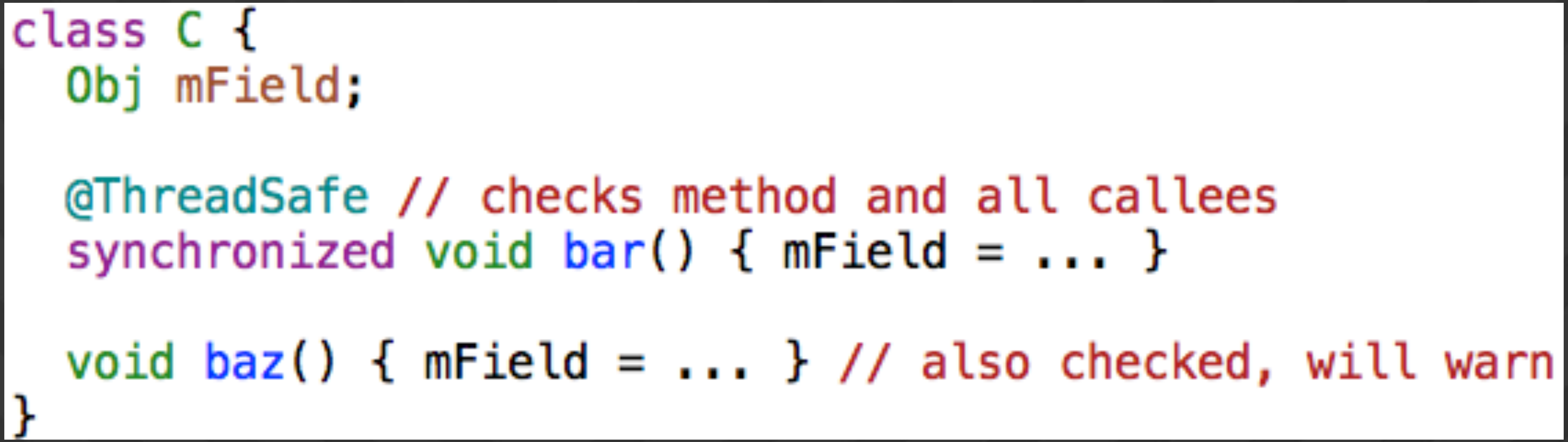
```
ACQUIRED_BEFORE, ACQUIRED_AFTER
```

And a few misc. hacks...

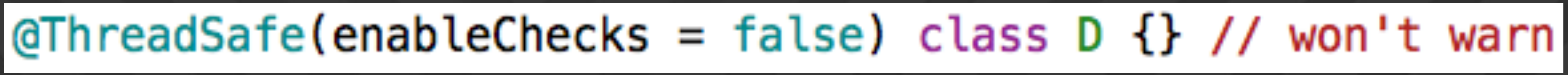
How to trigger analysis: just add @ThreadSafe



```
@ThreadSafe // checks all methods, subclasses
class A {
  void foo(B b) {
    b.m(); // all callees checked too
  }
}
```



```
class C {
  Obj mField;

  @ThreadSafe // checks method and all callees
  synchronized void bar() { mField = ... }

  void baz() { mField = ... } // also checked, will warn
}
```



```
@ThreadSafe(enableChecks = false) class D {} // won't warn
```

How to trigger analysis: add @ThreadSafe aliases to .inferconfig

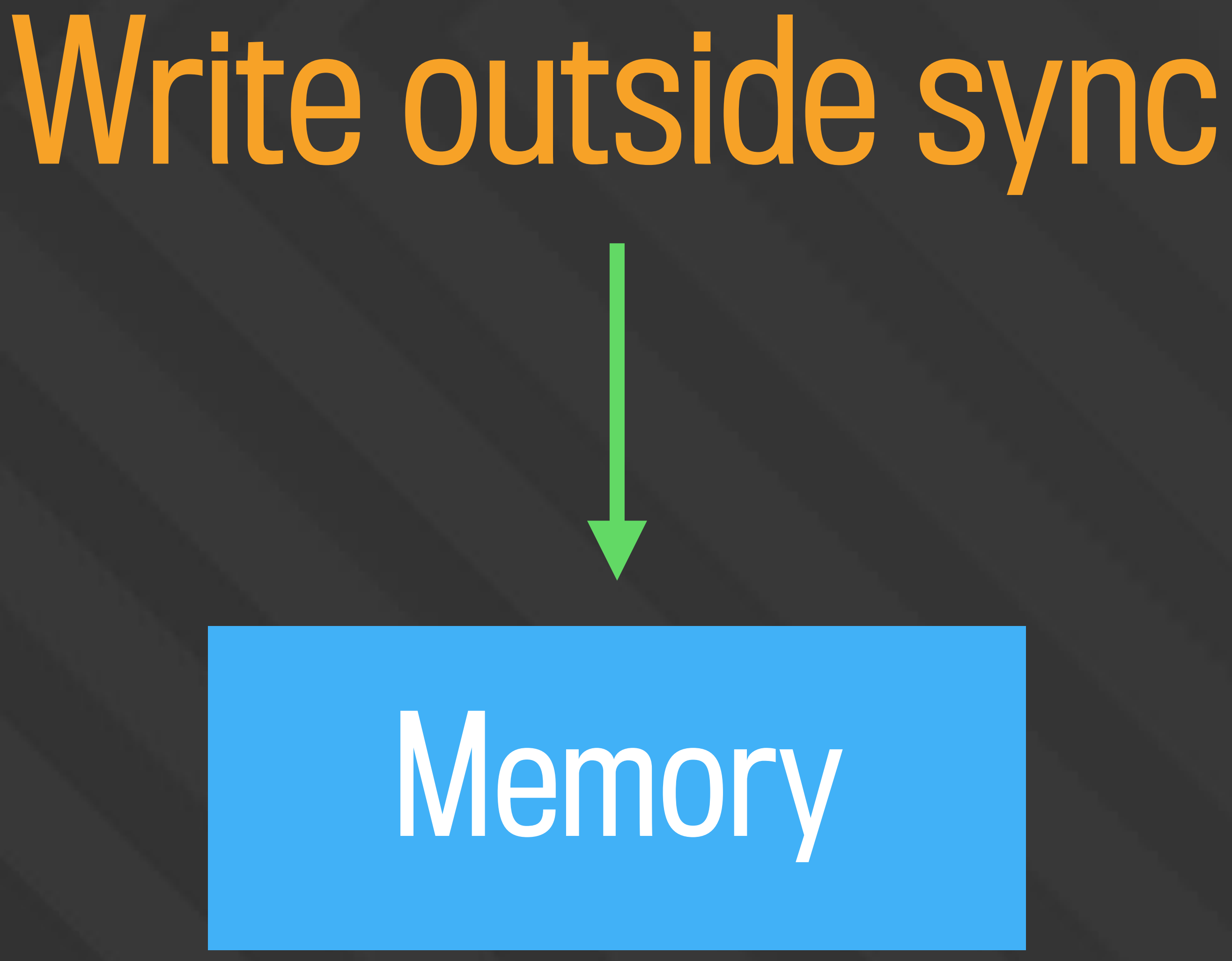
```
"threadsafe-aliases": [
  "com.facebook.litho.annotations.LayoutSpec",
  "com.facebook.litho.annotations.MountSpec"
]
```

This checks all Litho components for thread-safety

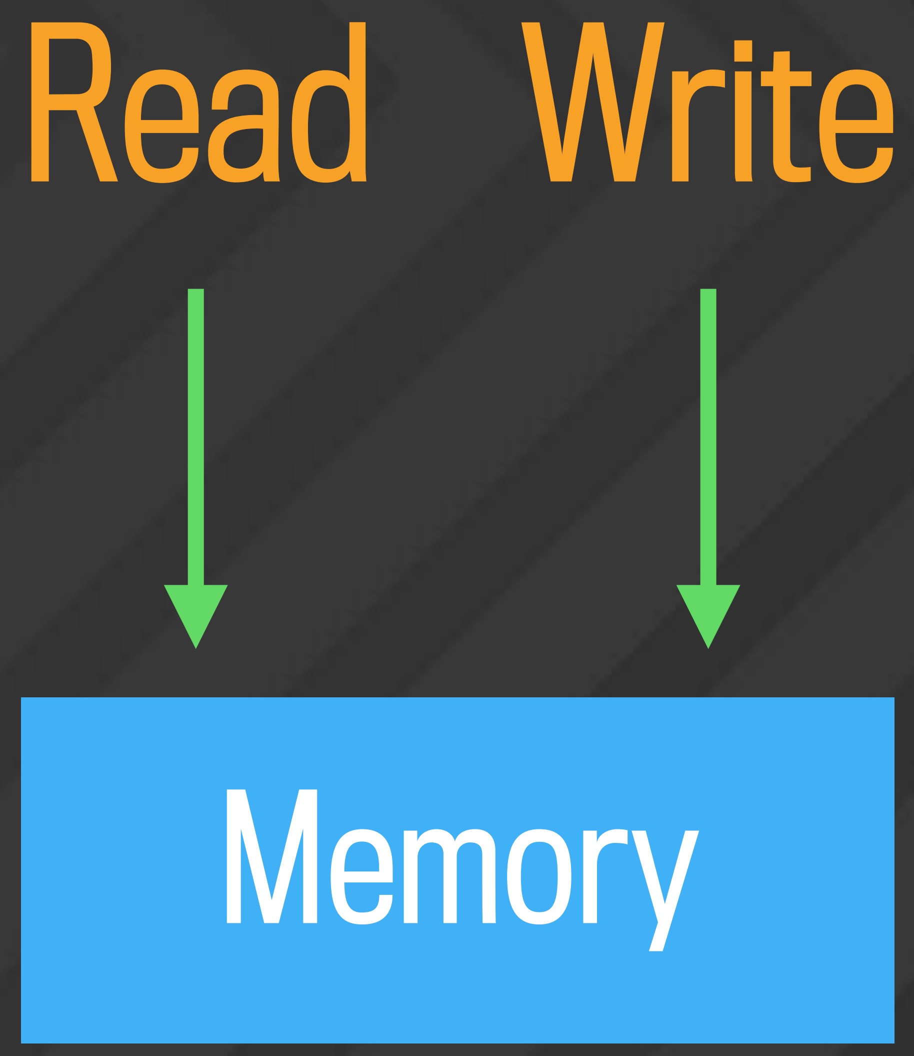
Infer thread-safety analysis: what does it do?

Find data races:
two simultaneous accesses to the same memory location
where at least one is a write.

Report data races with two warning types



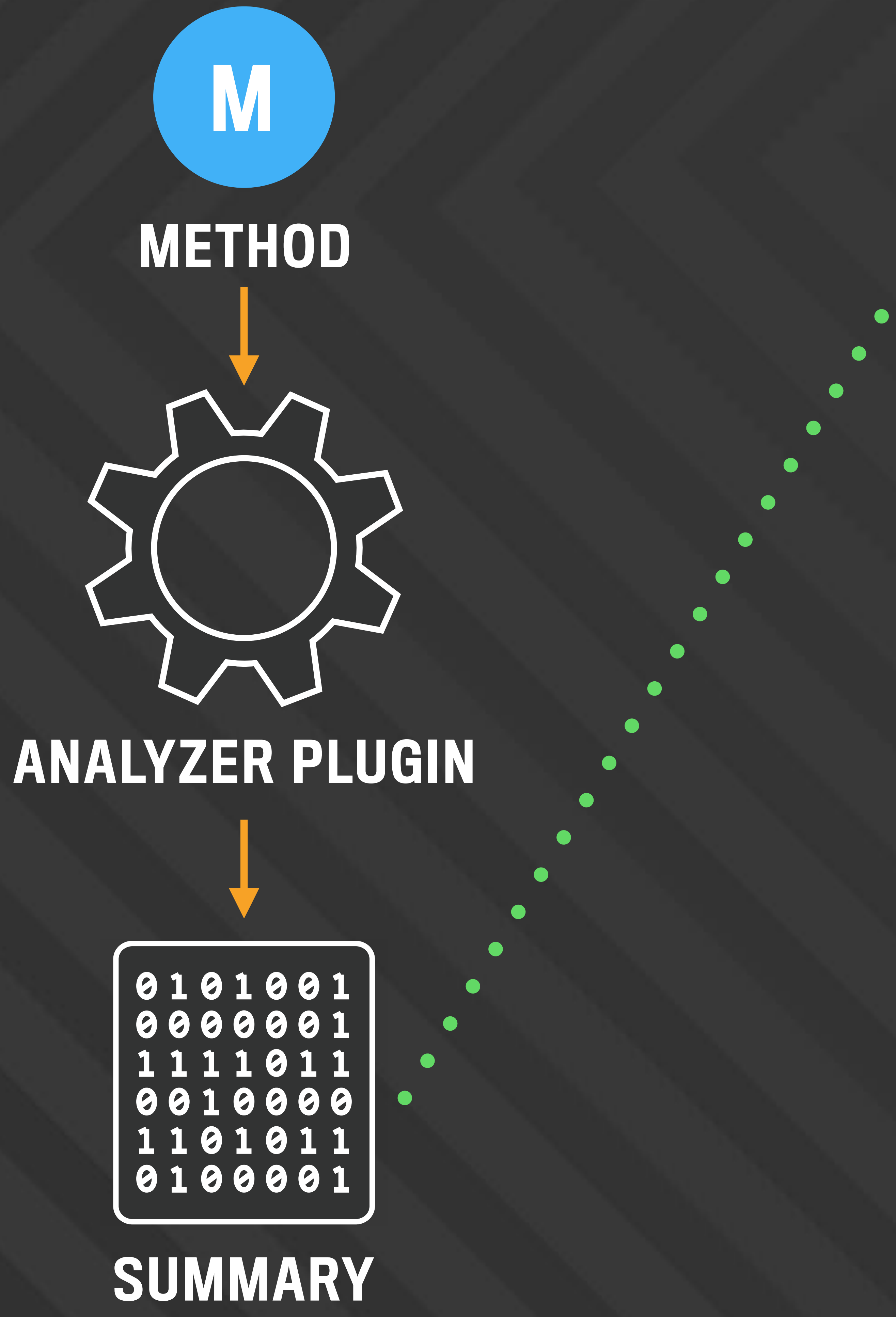
Unprotected write warning (self-race)



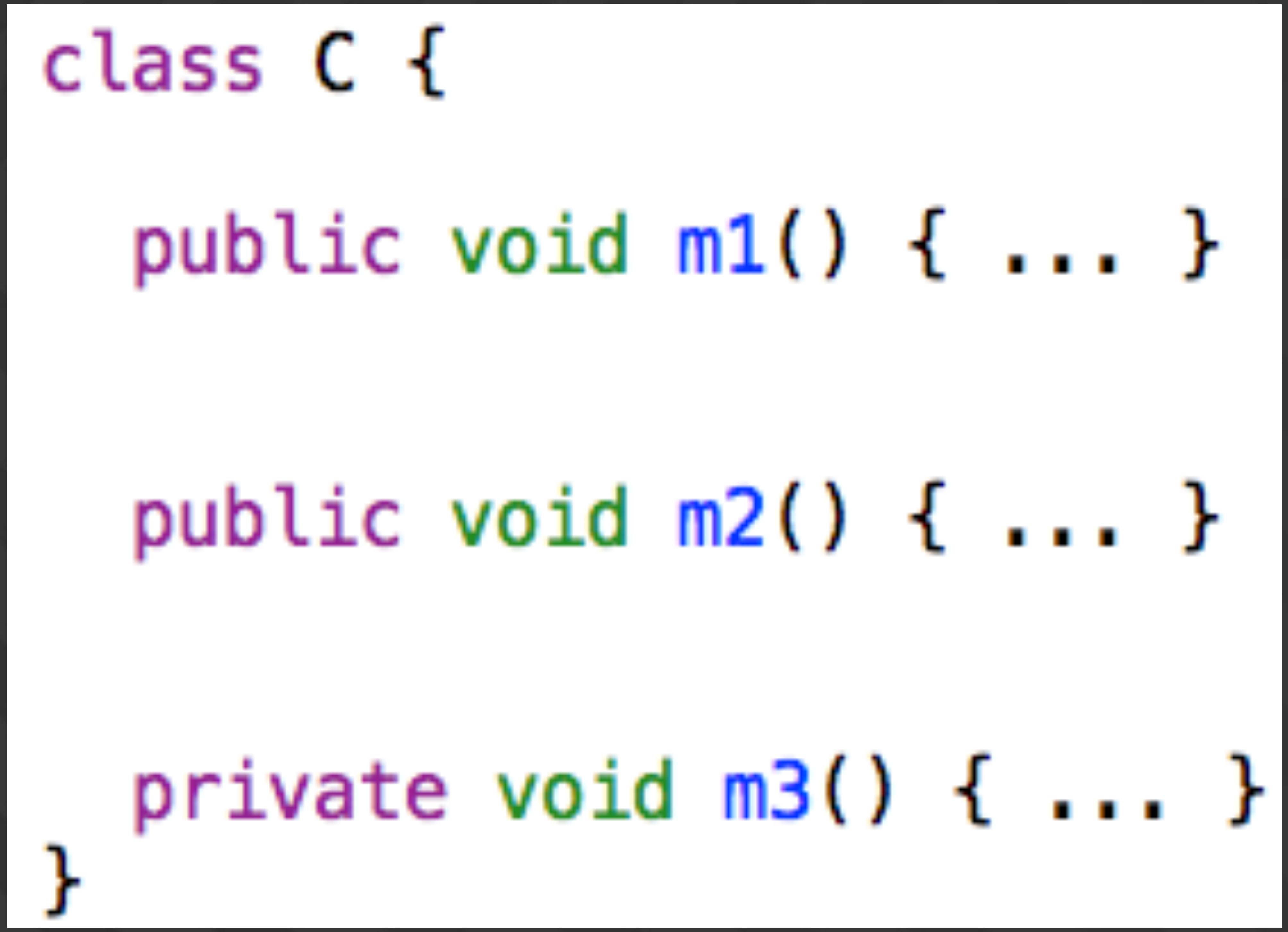
Read/write race warning

How does it work?

(1) Stack trace to access
(2) Lock held?
(3) On UI thread?
(4) Ownership info



Aggregate summaries for class and report



```
class C {

  public void m1() { ... }


  public void m2() { ... }


  private void m3() { ... }
}
```

M1 SUMMARY

M2 SUMMARY

M3 SUMMARY

Report when:

- reachable from non-private method
- can find conflicting access(es)

Summaries track last call that leads to access OOS

```
private void setF(Obj o) {
  o.f = ... // line 1
}
summ: { (o.f, 1, _) }
```

```
private void callSetF(Obj o) {
  setF(o); // line 2
}
summ: { (o.f, 2, setF) }
```

```
public void publicMethod(Obj o) {
  callSetF(o); // line 3
}
summ: { (o.f, 3, callSetF) }
```

Synchronization lets us forget accesses

```
void callSetFSync(Obj o) {
  synchronized(o) {
    lockHeld
    setF(o); summ: { (o.f, 1, _) }
  }
}
summ: { }
```

```
void lockWithBranch(Obj o) {
  if (needsLock) {
    Lock.lock();
    lockHeld
  }
  setF(o); // will warn
}
summ: { (o.f, 4, setF) }
```

Example error trace

Refinement: ownership via allocation

```
Obj local = new Obj();
owned(local)
local.f = ... // safe write
global.g = ... // unsafe write
owned(local), { (g, 3, _) }
```

Local ownership

```
Obj objFactory() {
  return new Obj();
}
summ: owned(ret)

Obj local = objFactory();
owned(local)
local.f = ... // safe write
```

Returning ownership

Refinement: conditional ownership

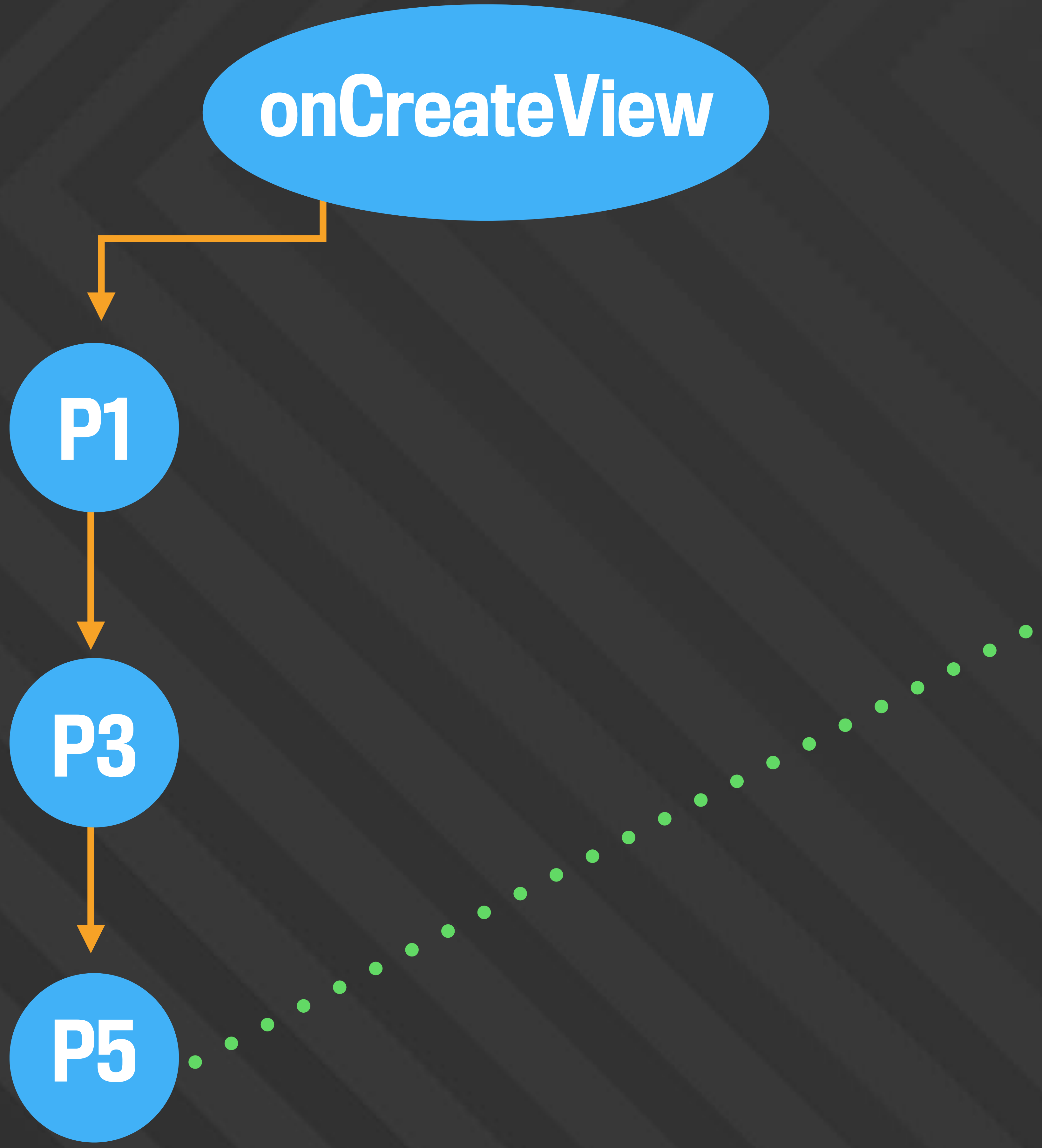
```
private void writeF(Obj a) {
  a.f = ...
}
summ: { (a.f, 1) if ¬owned(a) }

Obj o = new Obj();
owned(o)
writeF(o); // safe
```

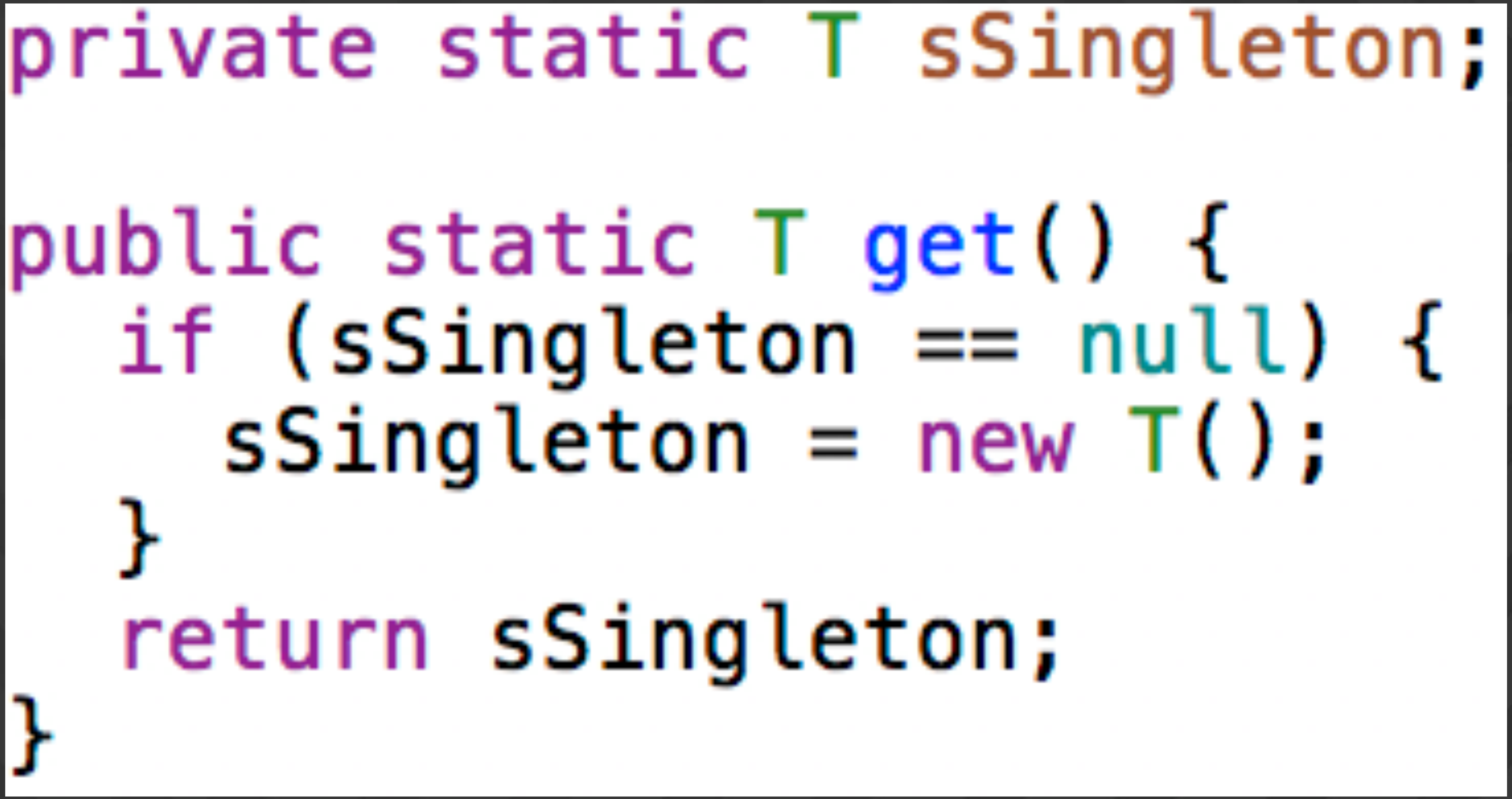
Safe if formal is owned by caller

```
Builder setX(X x) {
  this.x = x;
  return this;
}
summ: { (this.x if ¬owned(this) },
        owned(ret) if owned(this)
// owned(a)
Builder b = a.setX(x); // safe
// owned(a) ^ owned(b)
b.setY(y); // safe
```

Returns ownership if x is owned by caller

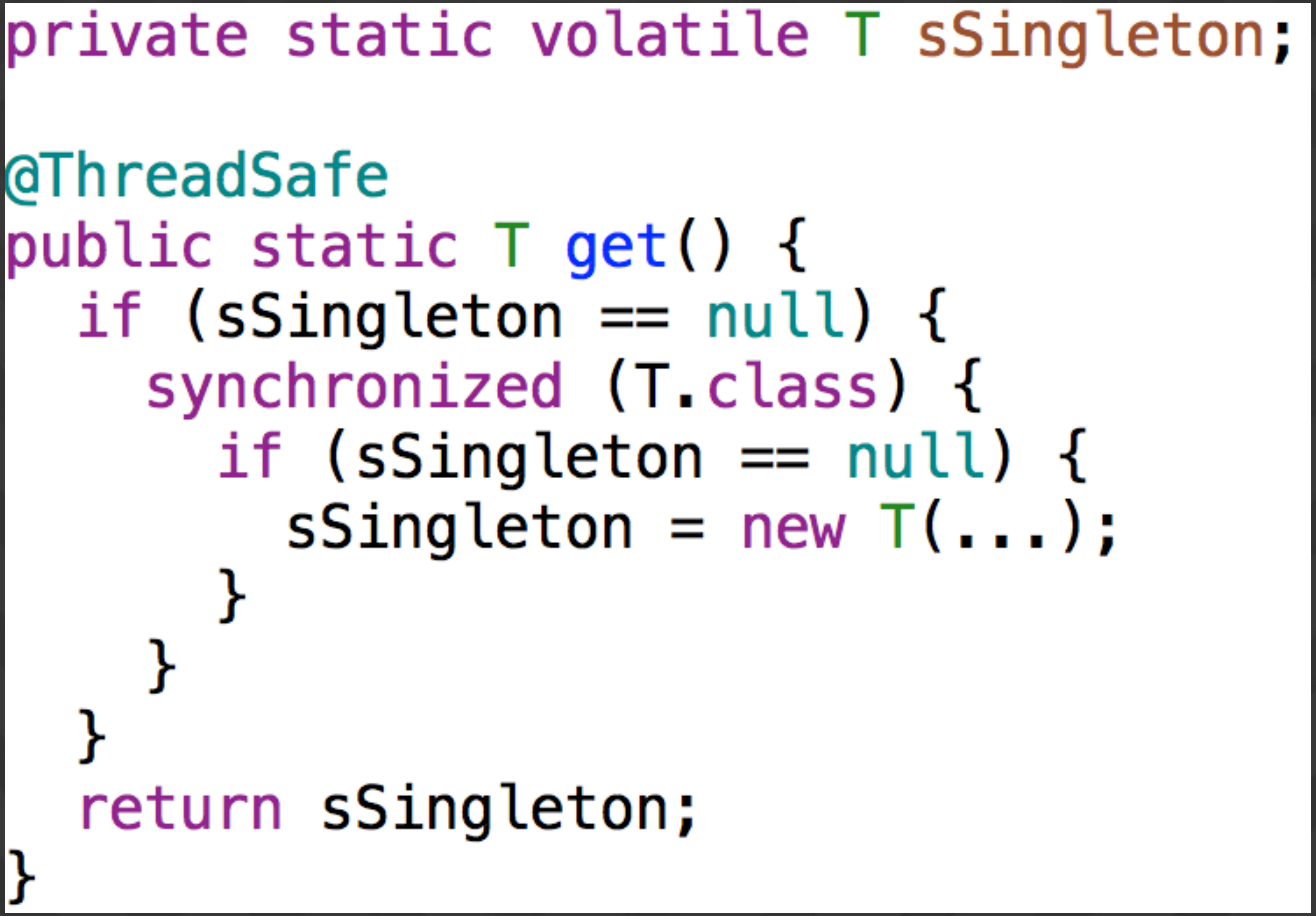
Example bug: unsafe singleton called from ThreadSafe code





```
private static T sSingleton;

public static T get() {
  if (sSingleton == null) {
    sSingleton = new T();
  }
  return sSingleton;
}
```

Fix: use double-checked locking



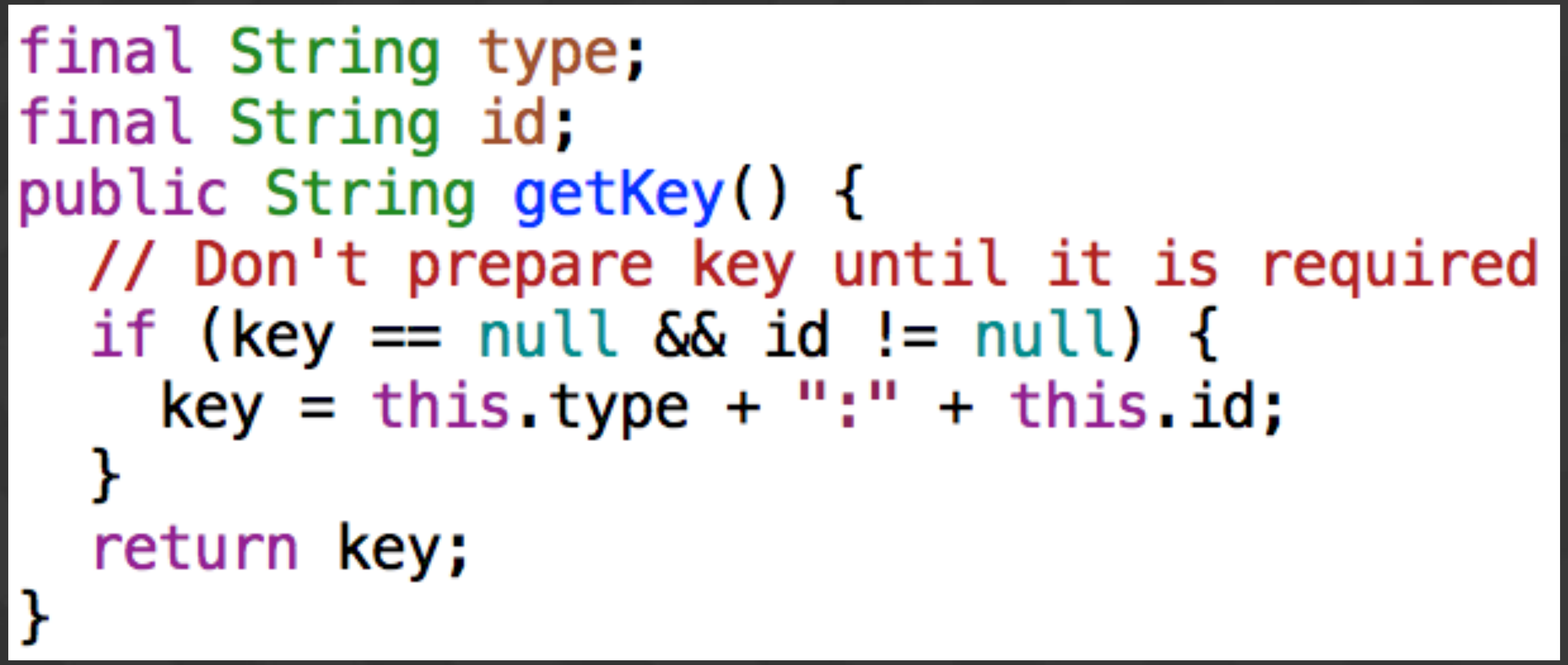
```
private static volatile T sSingleton;

@ThreadSafe
public static T get() {
  if (sSingleton == null) {
    synchronized (T.class) {
      if (sSingleton == null) {
        sSingleton = new T(...);
      }
    }
  }
  return sSingleton;
}
```

Annotations and programming methodology

Philosophy: create annotations that will be trusted now, checked later

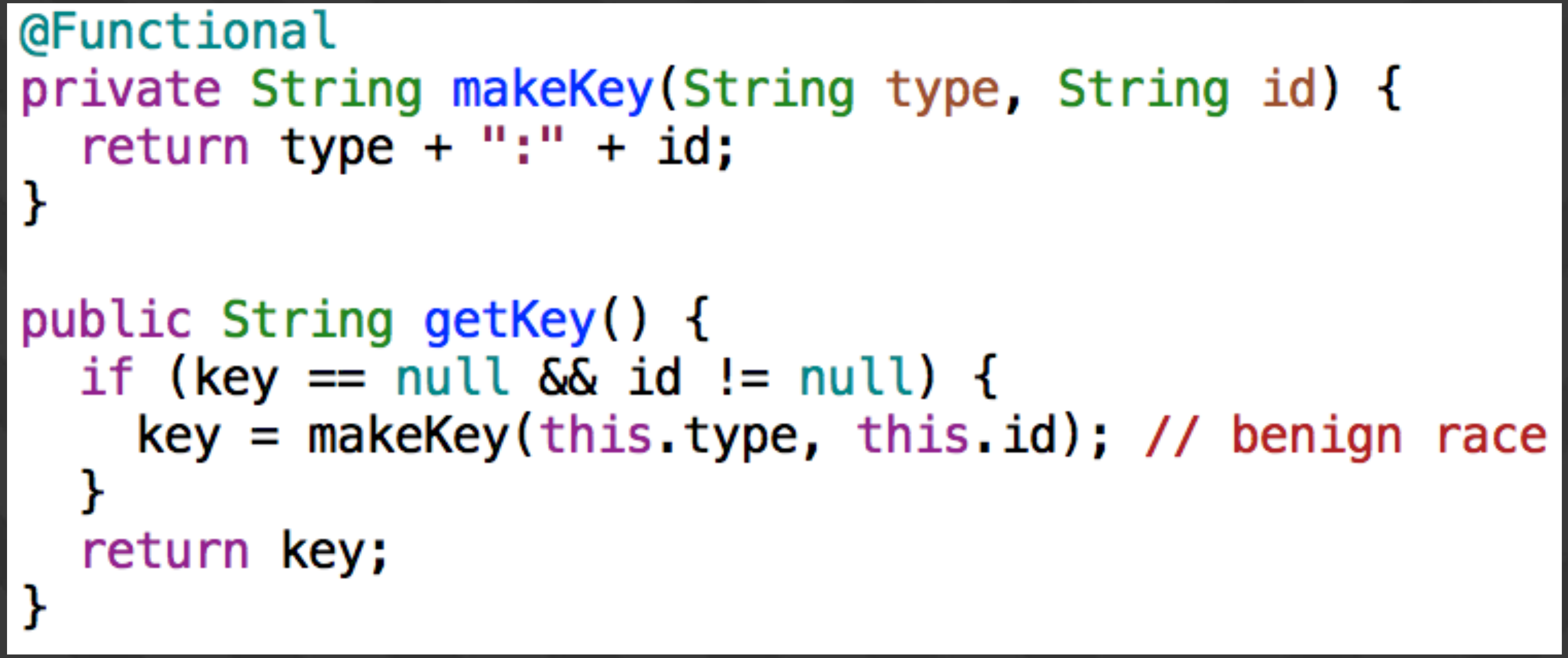
Note: all annotations available in Maven Central **infer-annotation** package

Allow benign races with @Functional



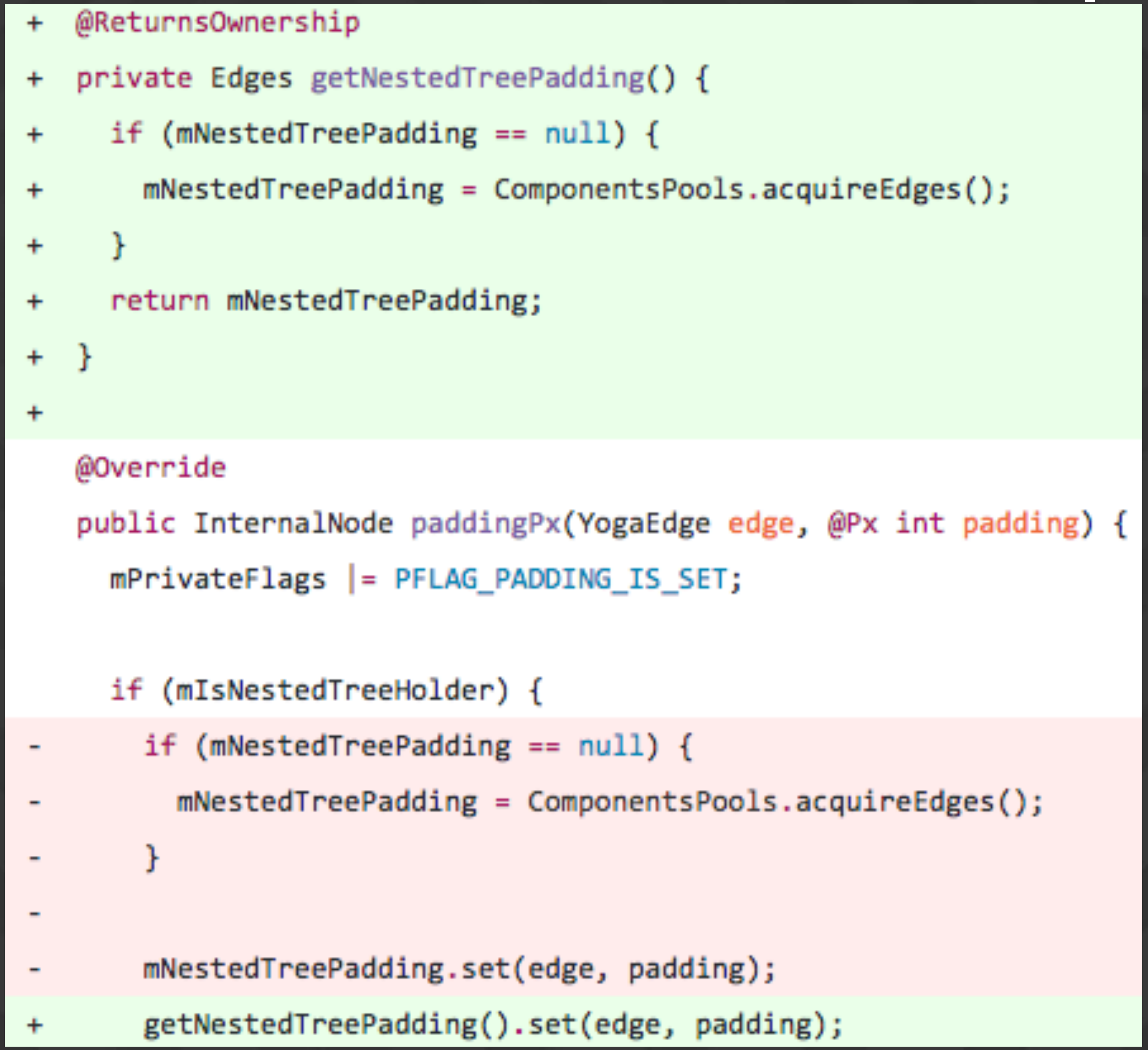
```java
final String type;
final String id;
public String getKey() {
  // Don't prepare key until it is required
  if (key == null && id != null) {
    key = this.type + ":" + this.id;
  }
  return key;
}
```

Make sure:
- makeKey() pure, args immutable
- key is not written/read elsewhere



```java
@Functional
private String makeKey(String type, String id) {
  return type + ":" + id;
}

public String getKey() {
  if (key == null && id != null) {
    key = makeKey(this.type, this.id); // benign race
  }
  return key;
}
```

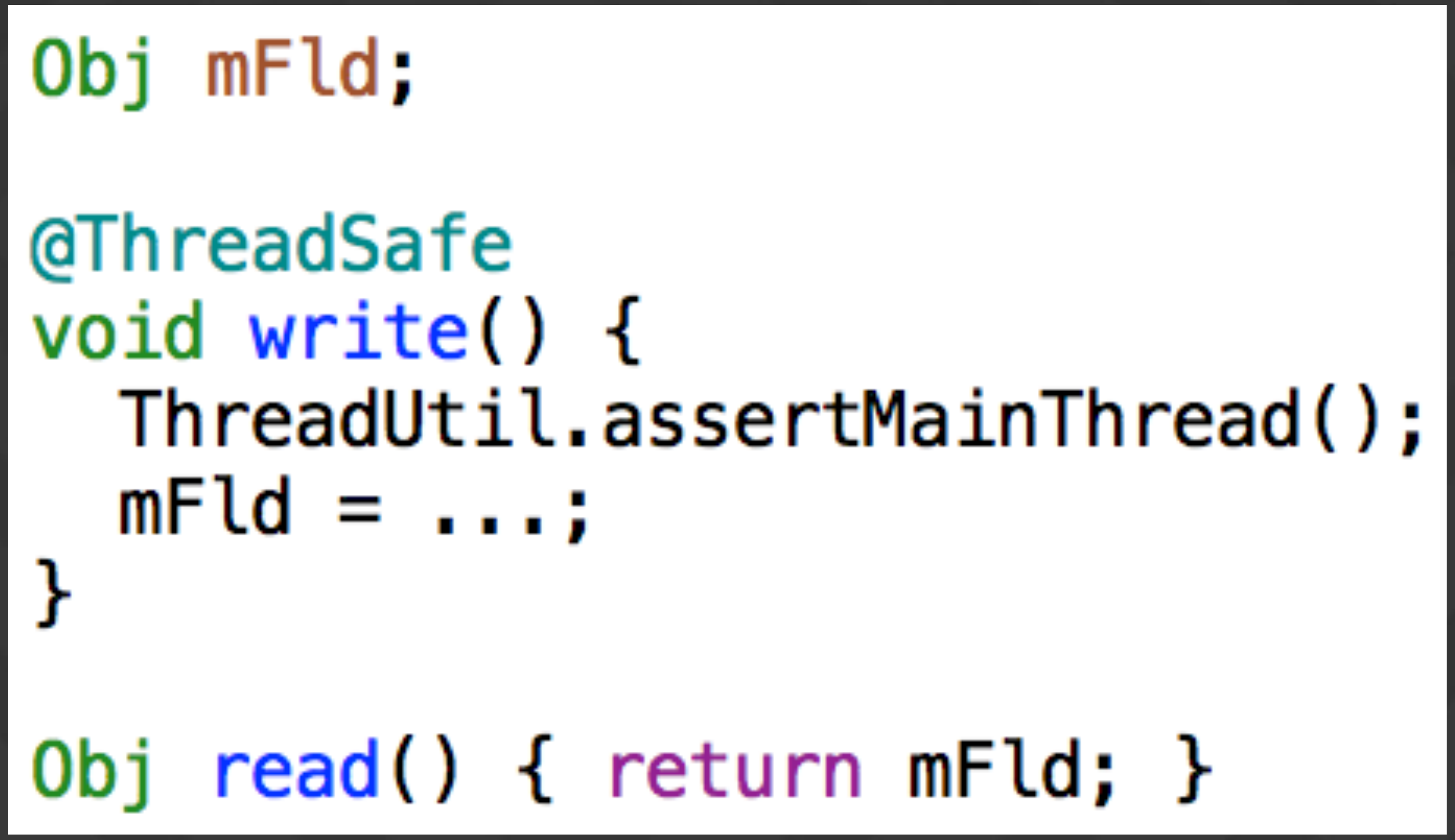
Lazy initialization with @ReturnsOwnership



```
+  @ReturnsOwnership
+  private Edges getNestedTreePadding() {
+    if (mNestedTreePadding == null) {
+      mNestedTreePadding = ComponentsPools.acquireEdges();
+    }
+    return mNestedTreePadding;
+  }
+
   @Override
   public InternalNode paddingPx(YogaEdge edge, @Px int padding) {
     mPrivateFlags |= PFLAG_PADDING_IS_SET;

     if (mIsNestedTreeHolder) {
-      if (mNestedTreePadding == null) {
-        mNestedTreePadding = ComponentsPools.acquireEdges();
-      }
-
-      mNestedTreePadding.set(edge, padding);
+      getNestedTreePadding().set(edge, padding);
```

Make sure:
- This is the only write to mNestedTreePadding
- mNestedTreePadding never leaks

Documenting thread structure with @ThreadConfined



```
Obj mFld;

@ThreadSafe
void write() {
  ThreadUtil.assertMainThread();
  mFld = ...;
}

Obj read() { return mFld; }
```

- Add @ThreadConfined(UI) to read()
- Call ThreadUtil.assertMainThread() in read()
- Annotate mFld with @ThreadConfined(UI)

Disclaimer: bug-finder, not prover

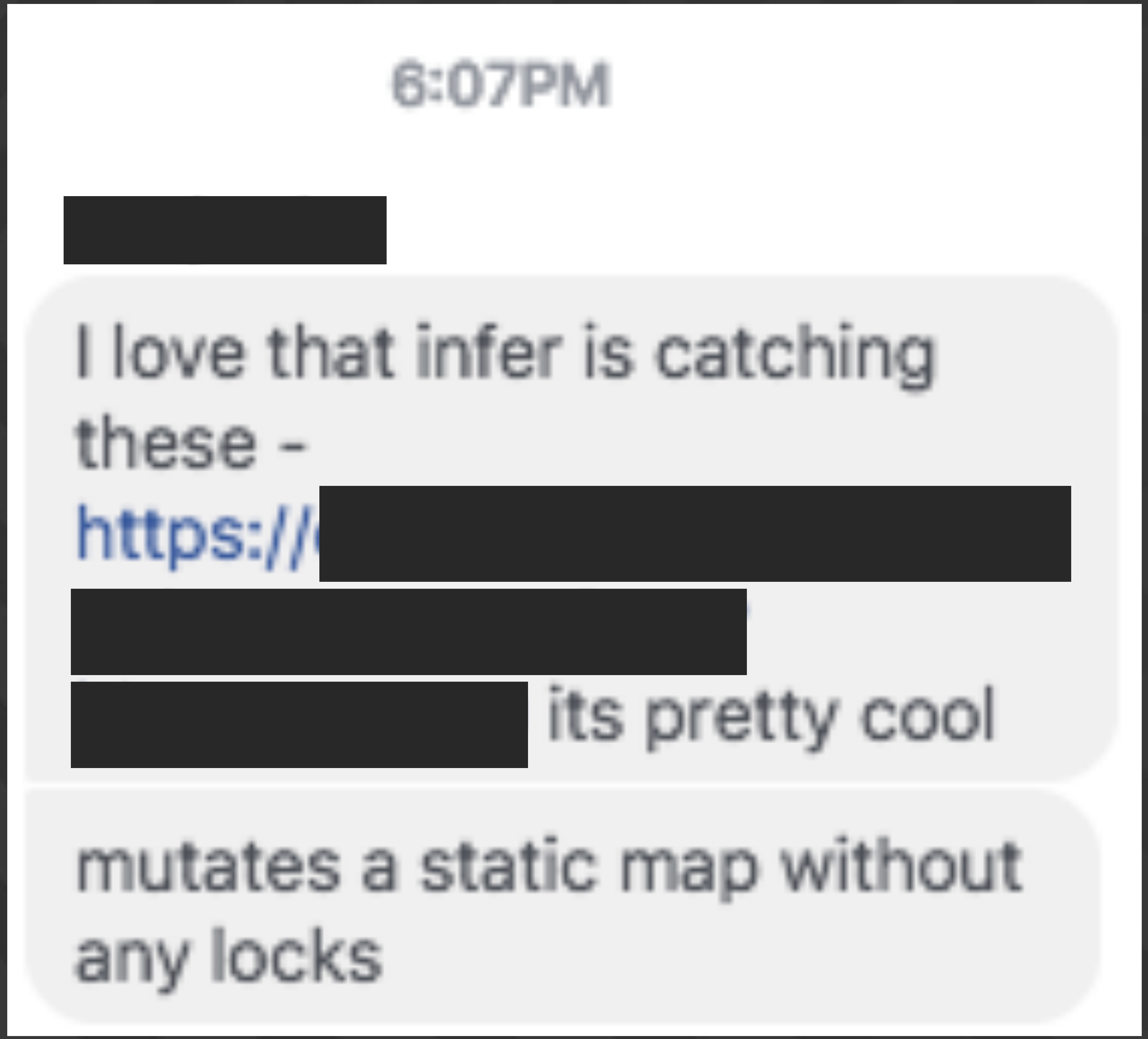
- Make sure you're holding the right lock
- @ReturnsOwnership, @Functional, @ThreadConfined trusted
- Accesses to **volatile** fields assumed safe
- Need @ThreadSafe annotations for checking

... but we're working on addressing all of these

Thread-safety analysis makes conversion faster/safer

- 100+ Litho components moved to background layout
- Only three major crashes (one caught by Infer, but ignored!)
- Analysis enabled for all Litho diffs
- 300+ thread-safety regressions caught/ fixed on diffs

Conclusion: try Infer's new analyses


6:07PM

[redacted]

I love that infer is catching these - https://[redacted] its pretty cool

mutates a static map without any locks

- Static analysis helps developers move faster and with more confidence
- Modular + compositional powerful: make thread-safety analysis possible
- Find bugs with Infer!

fbinfer.com/docs/threadsafety.html