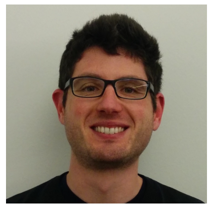
Dynamic Lifestate Verification of Android Applications

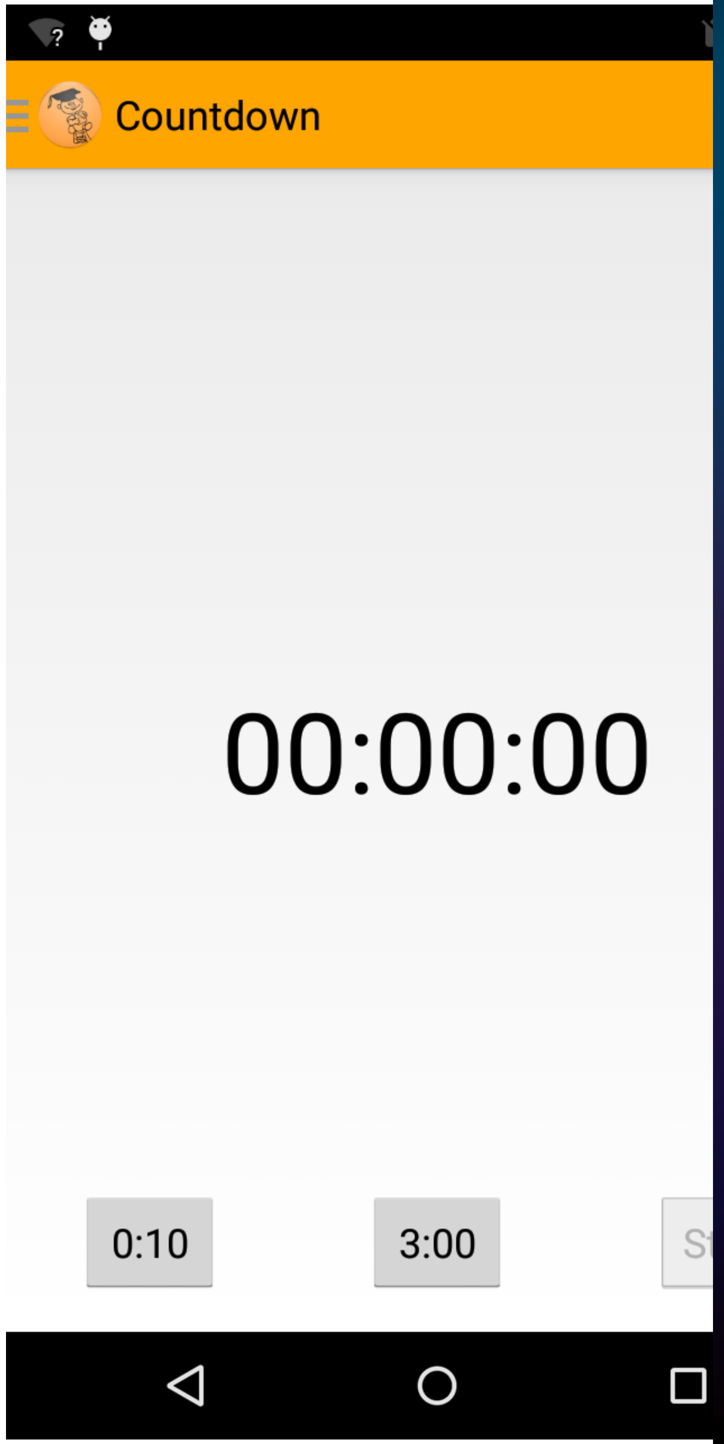
Shawn Meier


Sergio Mover

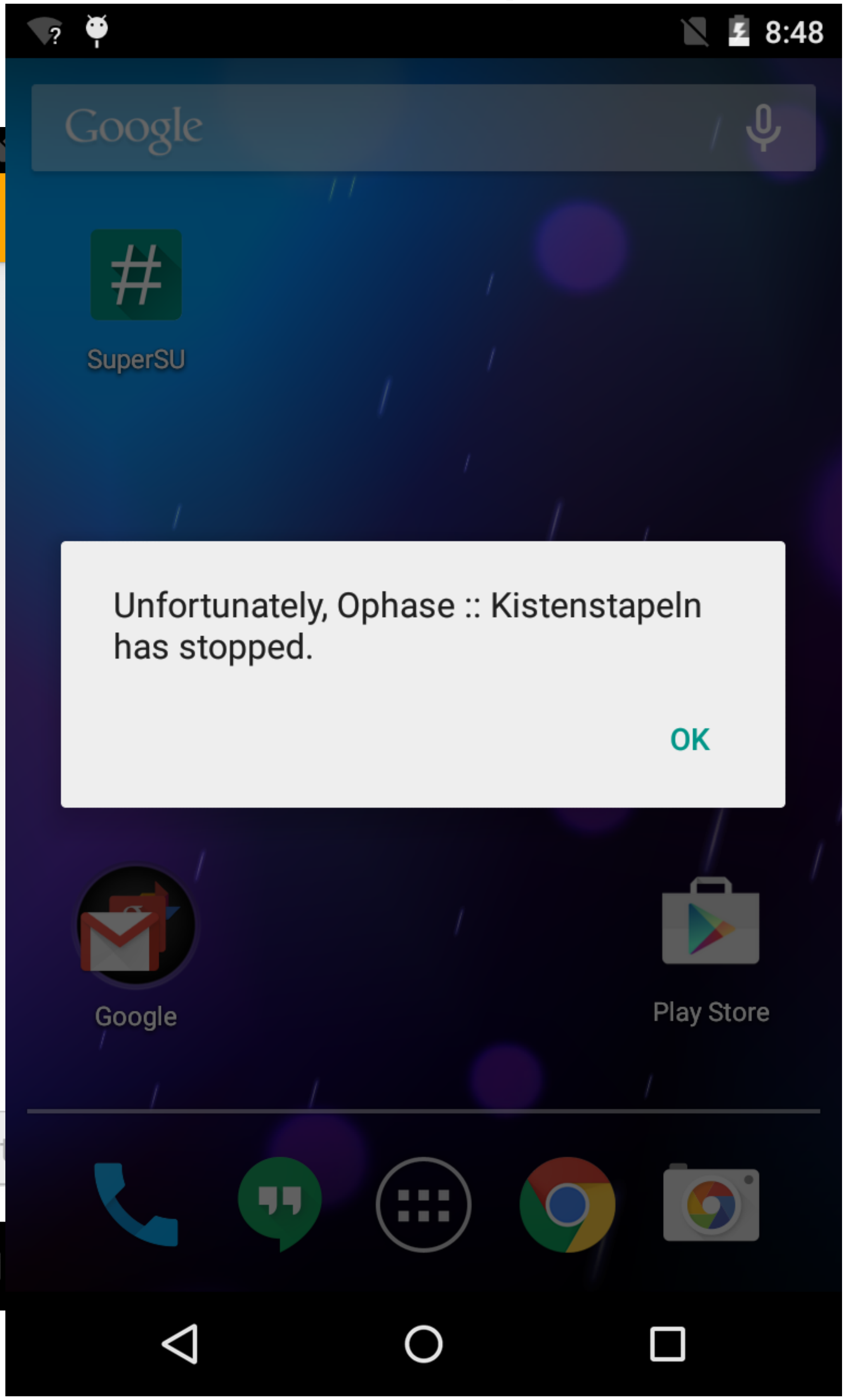
Bor-Yuh Evan Chang

CUPLV

Kistenstapeln

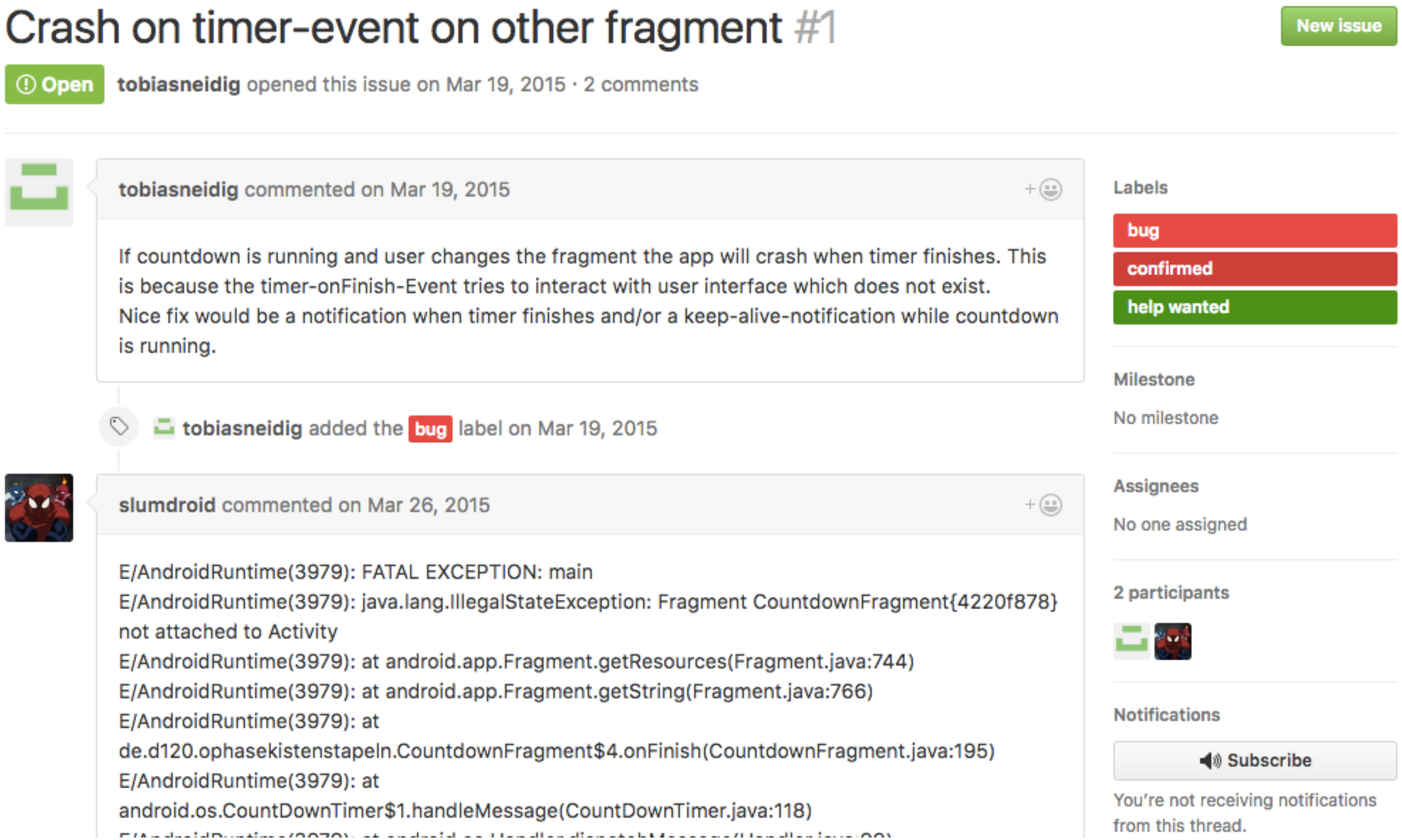


00:00:00



Unfortunately, Ophase :: Kistenstapeln has stopped.

Kistenstapeln



Crash on timer-event on other fragment #1

New issue

Open tobiasneidig opened this issue on Mar 19, 2015 · 2 comments

tobiasneidig commented on Mar 19, 2015

If countdown is running and user changes the fragment the app will crash when timer finishes. This is because the timer-onFinish-Event tries to interact with user interface which does not exist.
Nice fix would be a notification when timer finishes and/or a keep-alive-notification while countdown is running.

tobiasneidig added the bug label on Mar 19, 2015

slumdroid commented on Mar 26, 2015

```
E/AndroidRuntime(3979): FATAL EXCEPTION: main
E/AndroidRuntime(3979): java.lang.IllegalStateException: Fragment CountdownFragment{4220f878}
not attached to Activity
E/AndroidRuntime(3979): at android.app.Fragment.getResources(Fragment.java:744)
E/AndroidRuntime(3979): at android.app.Fragment.getString(Fragment.java:766)
E/AndroidRuntime(3979): at
de.d120.ophasekistenstapeln.CountdownFragment$4.onFinish(CountdownFragment.java:195)
E/AndroidRuntime(3979): at
android.os.CountDownTimer$1.handleMessage(CountDownTimer.java:118)
```

Labels: bug, confirmed, help wanted

Milestone: No milestone

Assignees: No one assigned

2 participants

Notifications

Subscribe

You're not receiving notifications from this thread.

What causes this defect?

```
class CountdownFragment extends Fragment{
    void onActivityCreated(Activity a){
        Button b = (Button)findViewById(R.id.button);
        b.setOnClickListener( new OnClickListener{
            void onClick(Button b){
                startTimer();
            }
        }
    }

    void startTimer(){
        new CountDownTimer(10000, 10){
            void onFinish(){
                textboxCountdown.setText(getString(R.string.done));
            }
        }.start();
    }
```

Callback invoked when the **fragment is viewable**

Callback invoked when the **button is pressed**

getString can be invoked after the fragment is no longer viewable

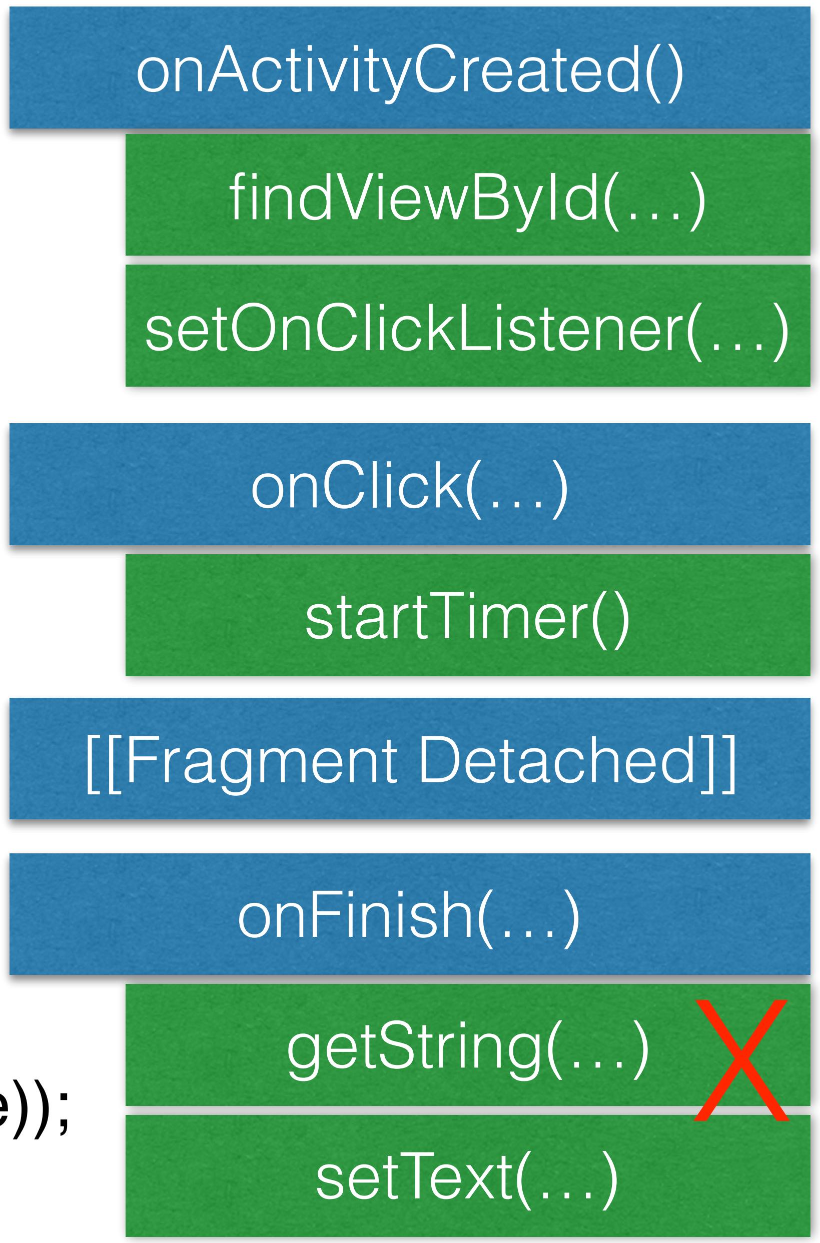
Callback invoked when the **timer finishes**

Calling a method in the wrong state.

Tracing the Application

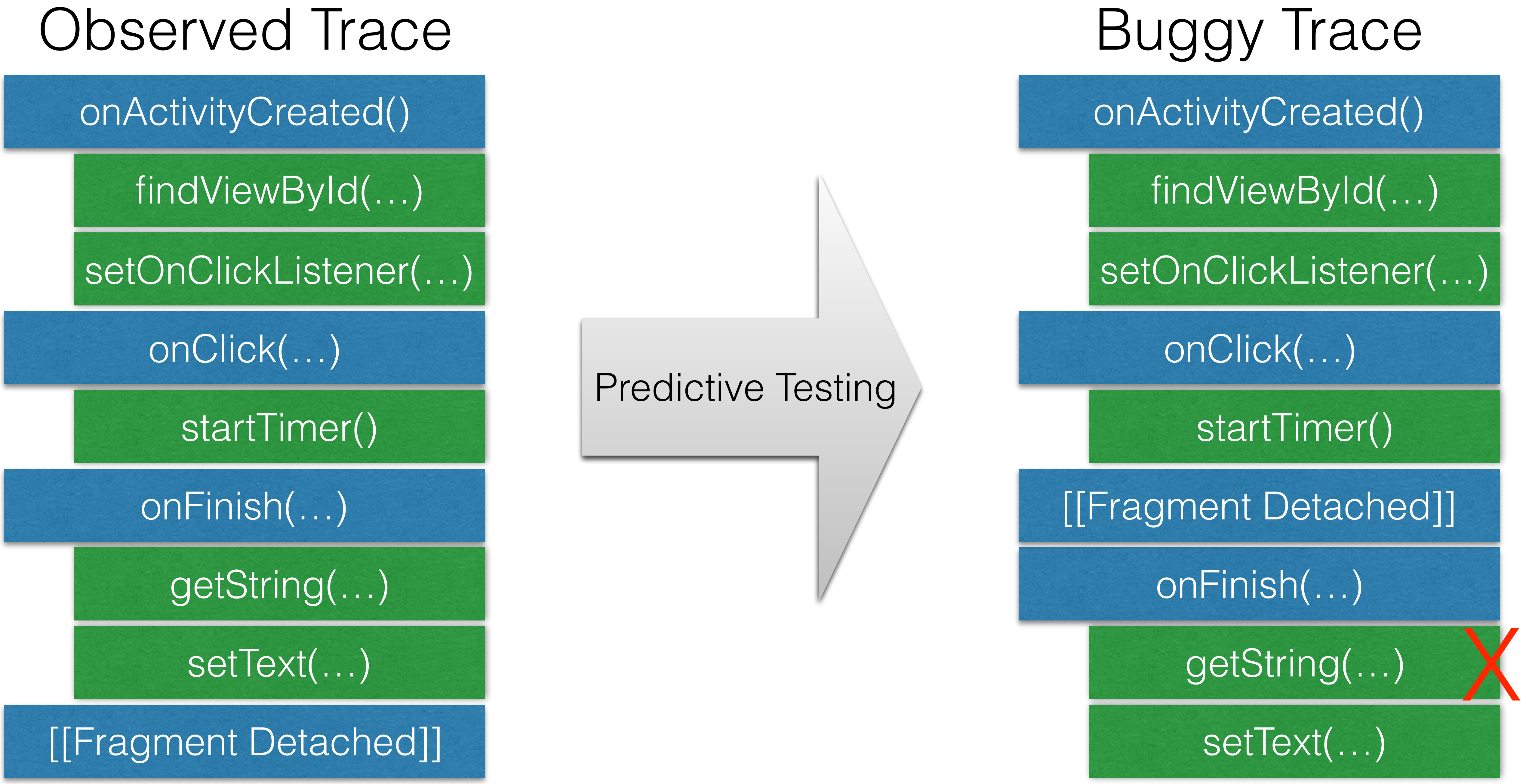
```
class CountdownFragment extends Fragment{
    void onActivityCreated(Activity a){
        Button b = (Button)findViewById(R.id.button);
        b.setOnClickListener( new OnClickListener{
            void onClick(Button b){
                startTimer();
            }
        }
    }

    void startTimer(){
        new CountDownTimer(10000, 10){
            void onFinish(){
                textboxCountdown.setText(getString(R.string.done));
            }
        }.start();
    }
}
```



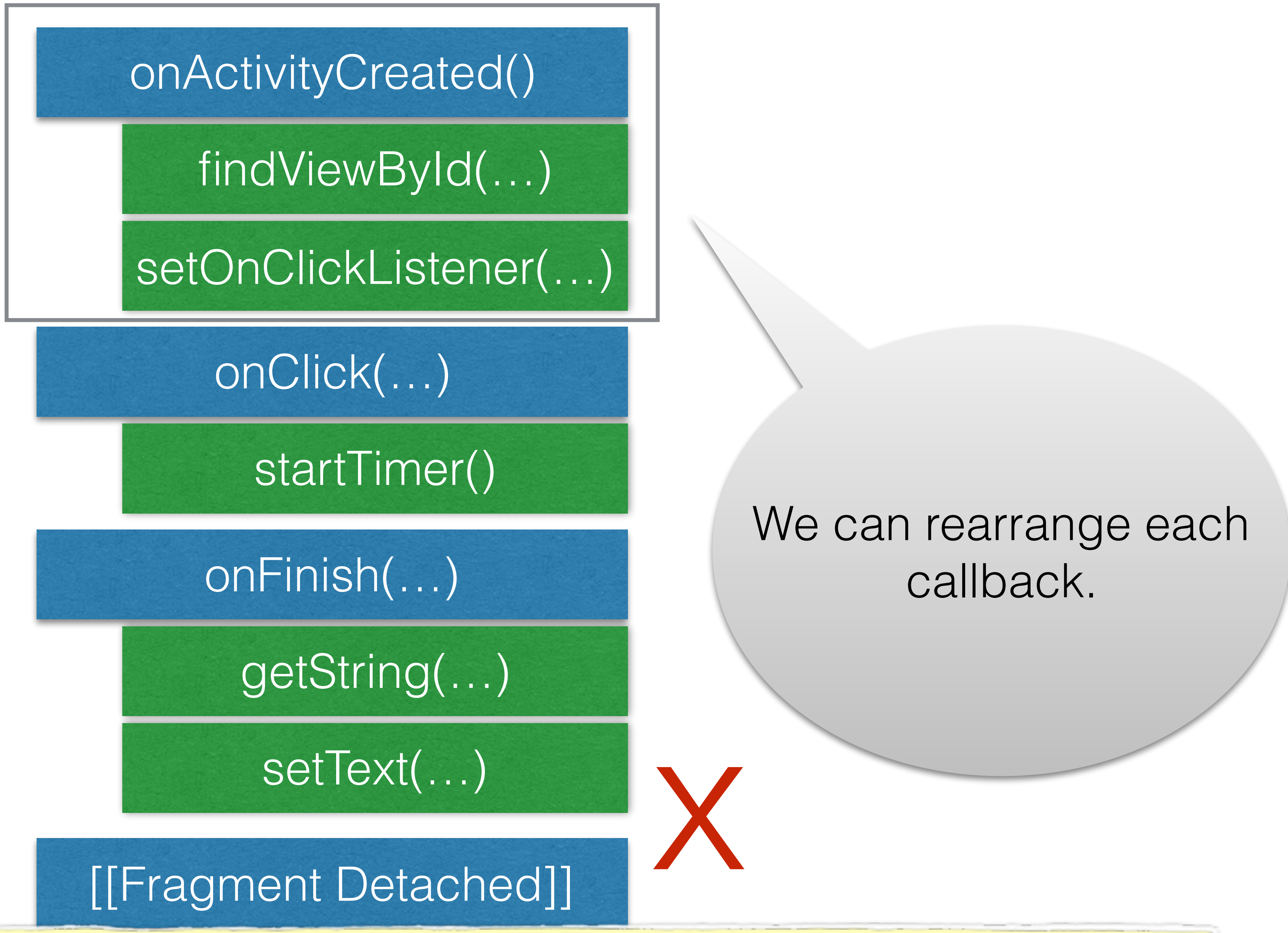
We can observe the order in which callbacks are invoked to cause the problem.

What if we only see the correct trace



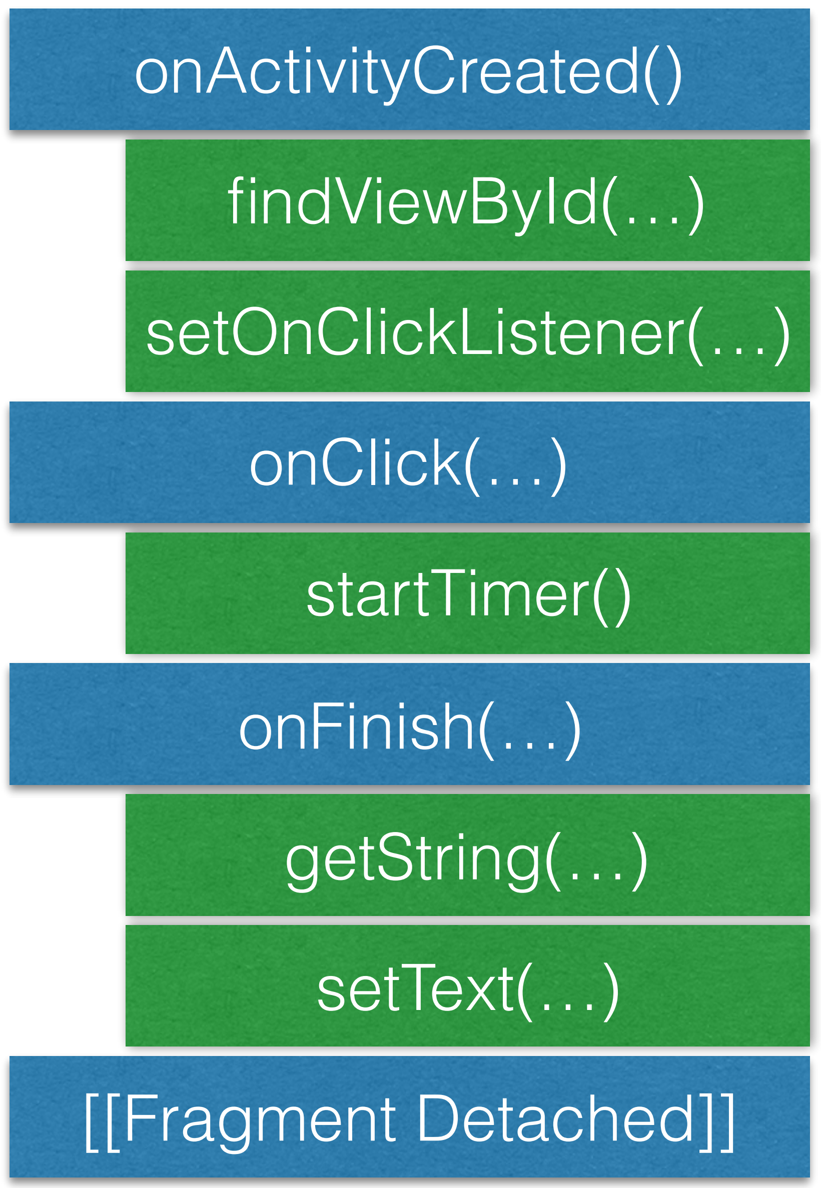
We would like to apply dynamic verification to expose this defect.

Reordering



It is possible to observe a bad ordering exposing the defect without actually observing the defect.

Some of The Reorderings of Transitions are infeasible



What prevents reordering in arbitrary ways that cannot be realized?

Two rules

```
<init> ↛(cb) onFinish()

startTimer() → (cb) onFinish()
```

We write rules to constrain the system from reordering the trace in bogus ways.

Lifestate Specification: Rules to Restrict the Possible Reorderings

enable

`message → (cb) message`

allow

`message →(ci) message`

Lifestate Rules

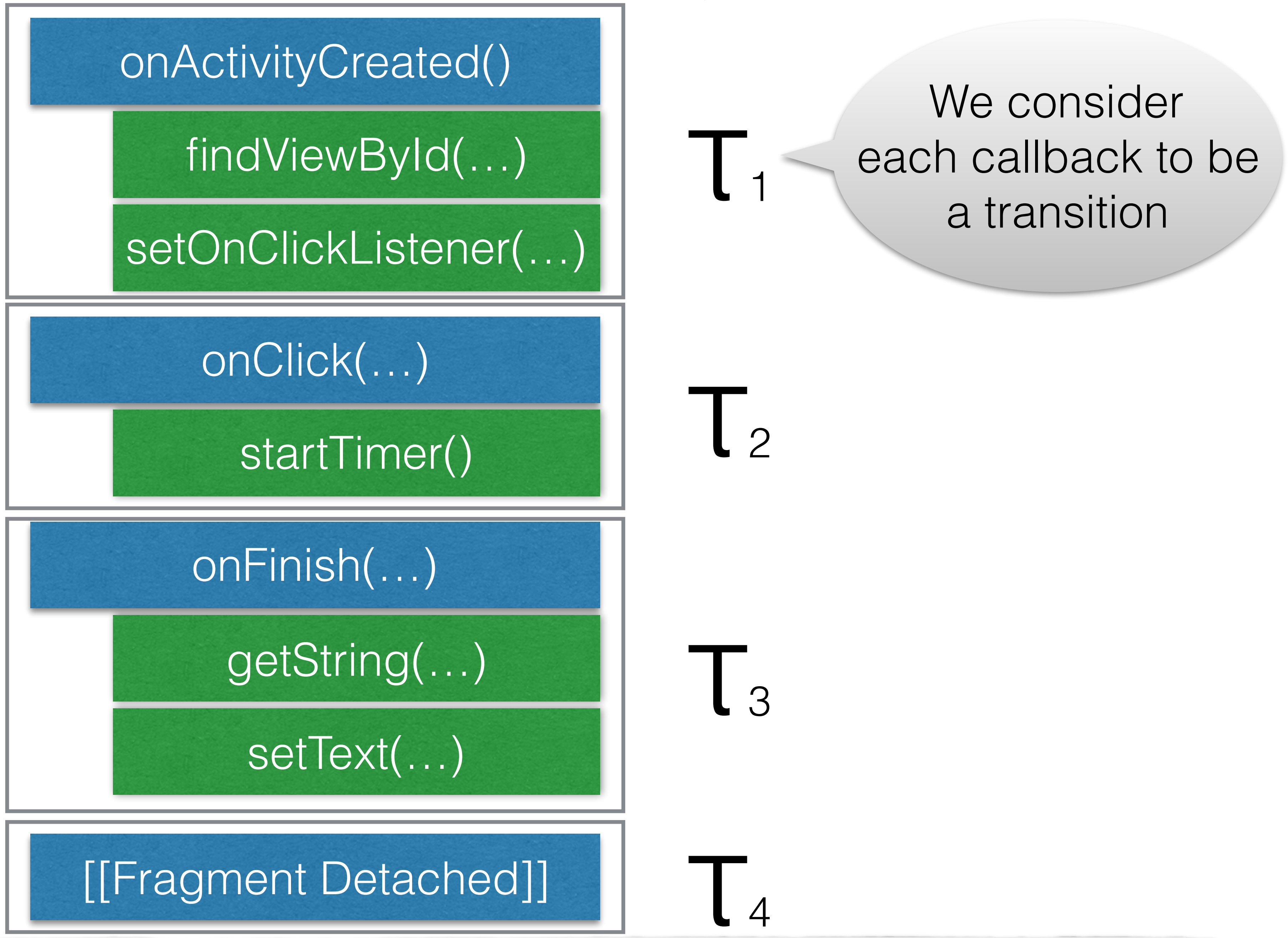
disable

`message ↛ (cb) message`

disallow

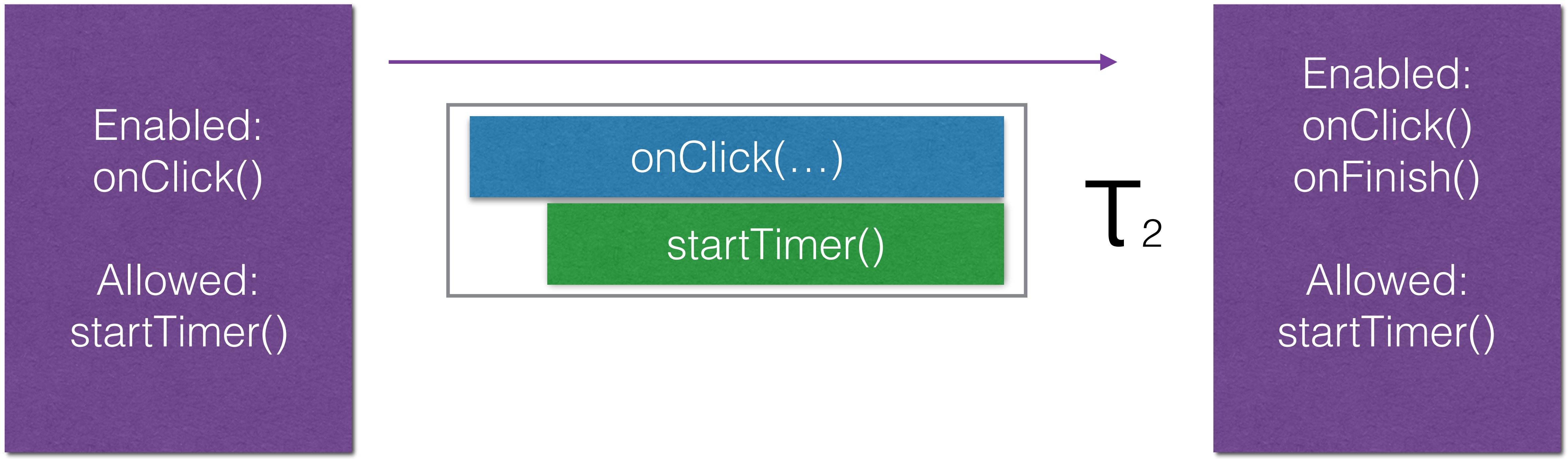
`message ↛(ci) message`

Reduced to Transition System



We create a transition system and can apply techniques such as Bounded Model Checking.

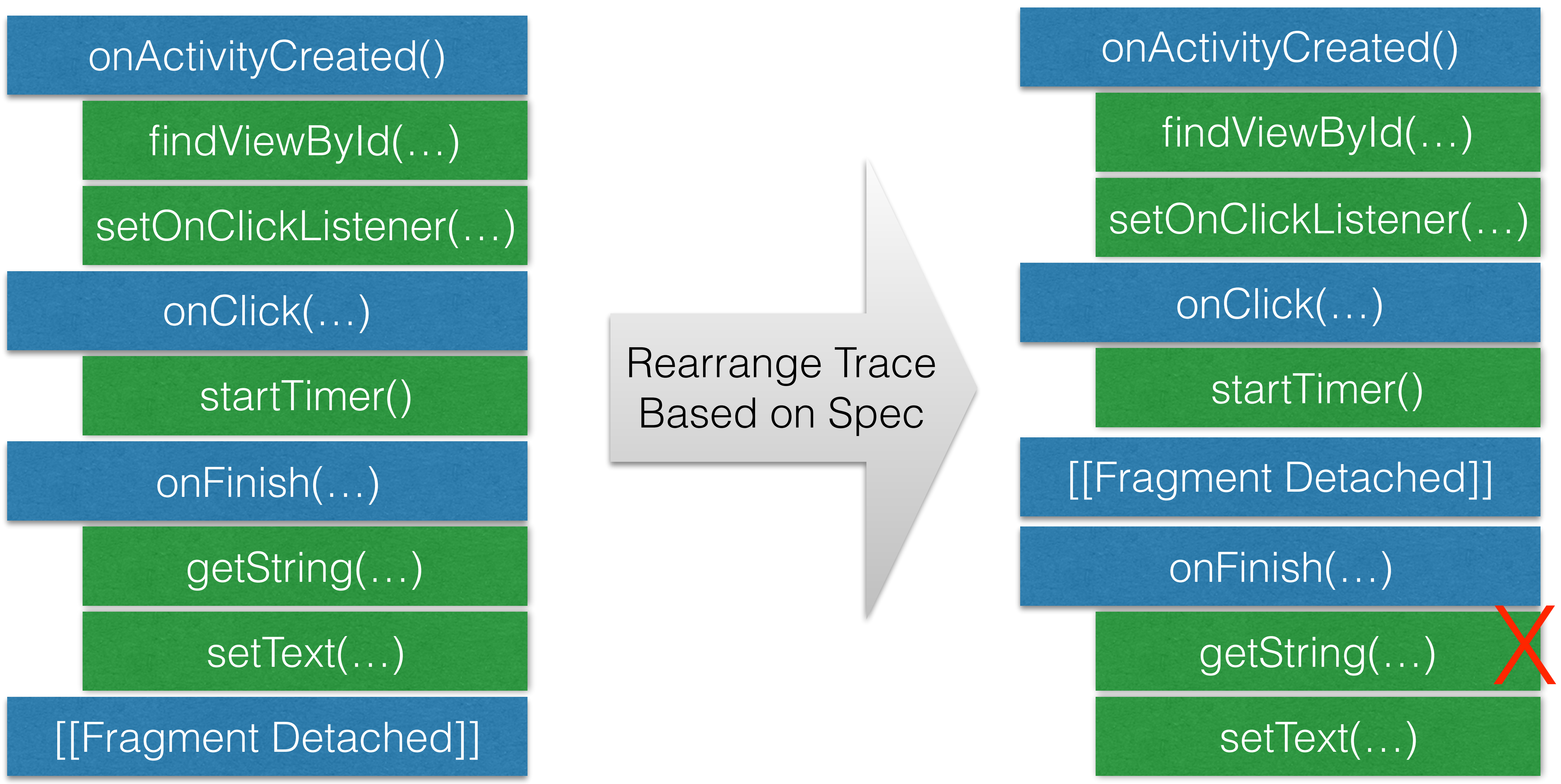
Transition System



`startTimer() → (cb) onFinish()`

Combining a callback with its relevant rules gives us a transition and a new state explaining what can happen next.

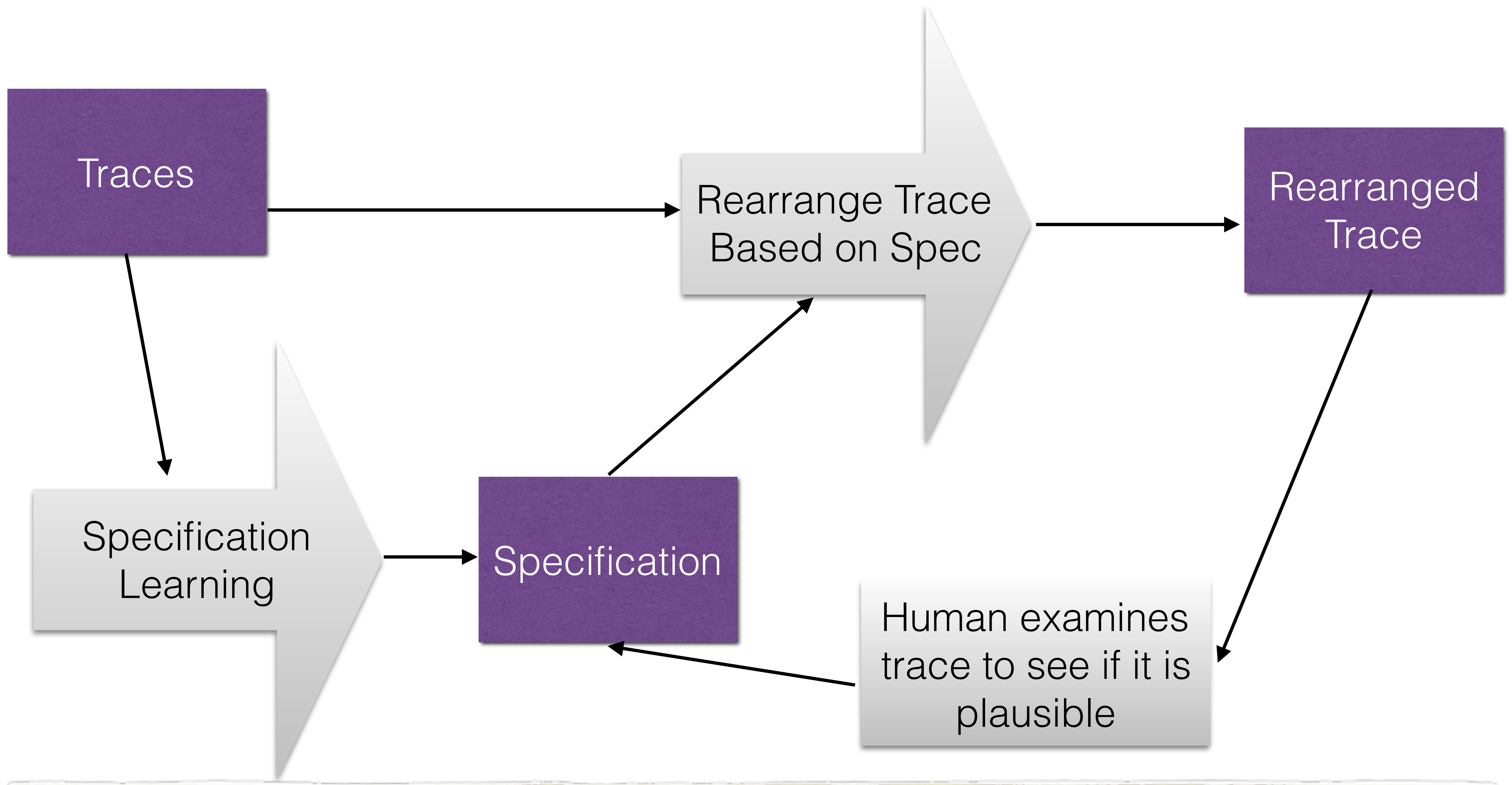
Reduced to Model Checking



We create a transition system and can apply techniques such as Bounded Model Checking.

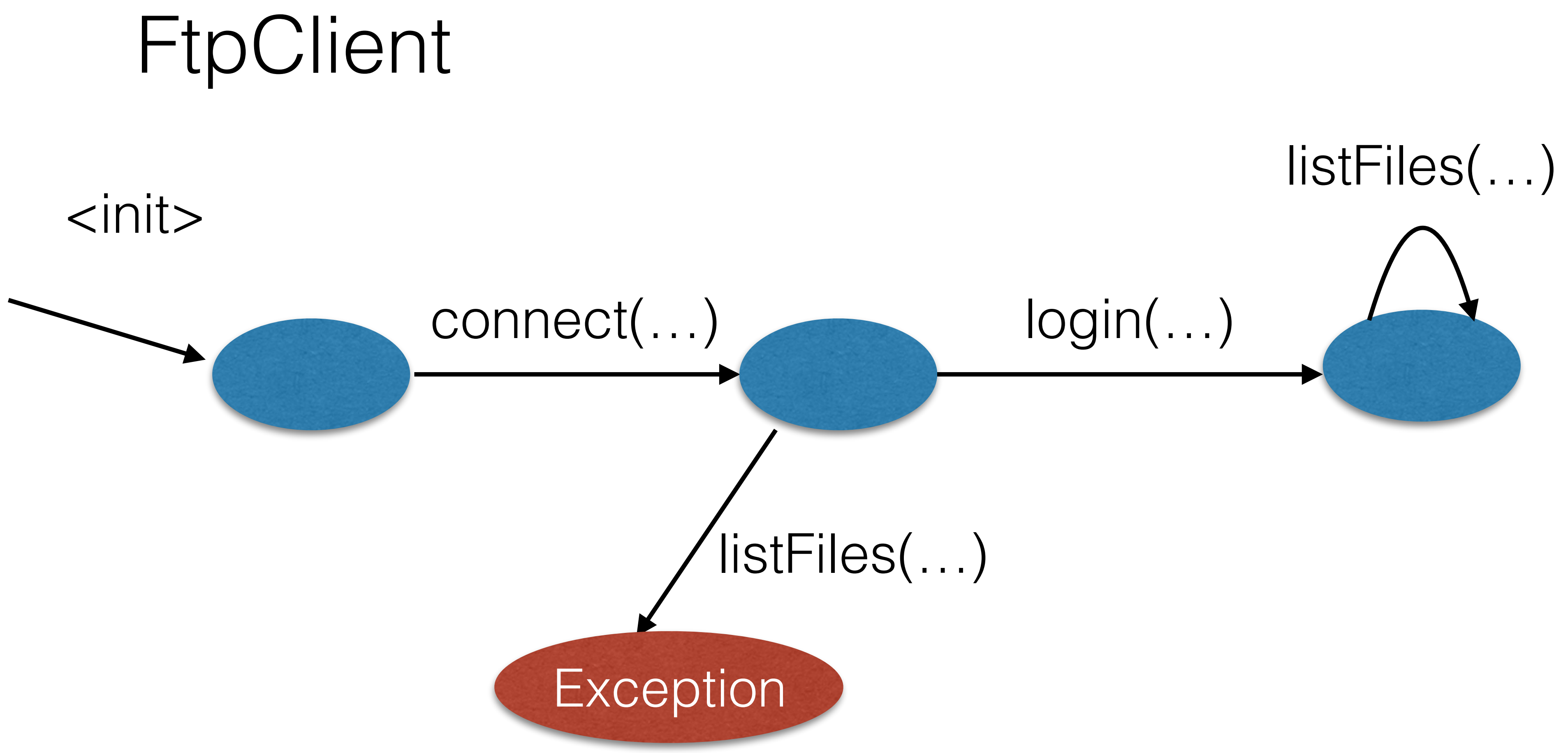
FEUERWEHR HANNOVER
FEUERWEHR HANNOVER

Mining and Refining Specifications


Traces
Rearrange Trace Based on Spec
Rearranged Trace
Specification Learning
Specification
Human examines trace to see if it is plausible

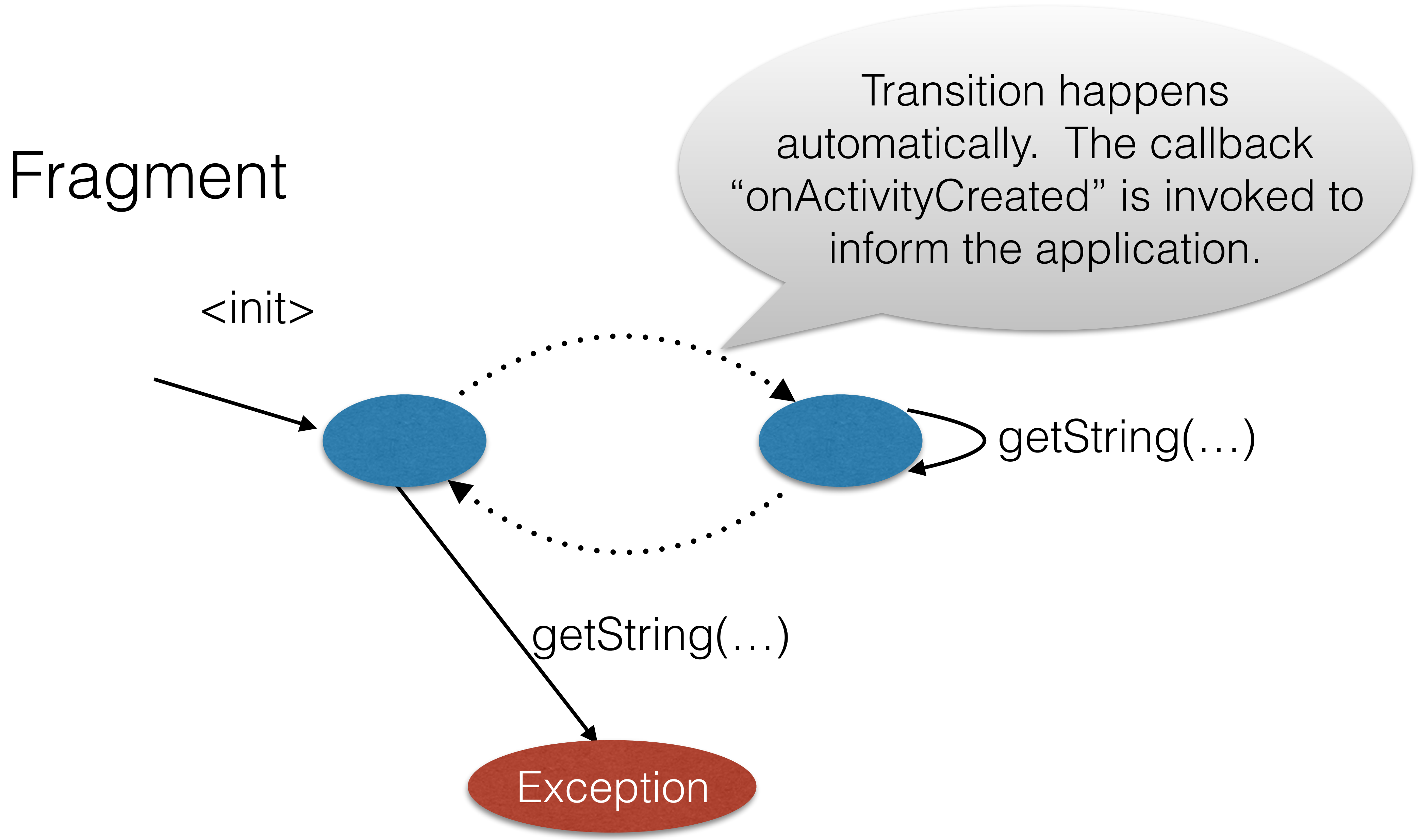
We combine mining and verification to get the correct set of specifications.

Can We Solve This With Typestate?



There is a problem however…

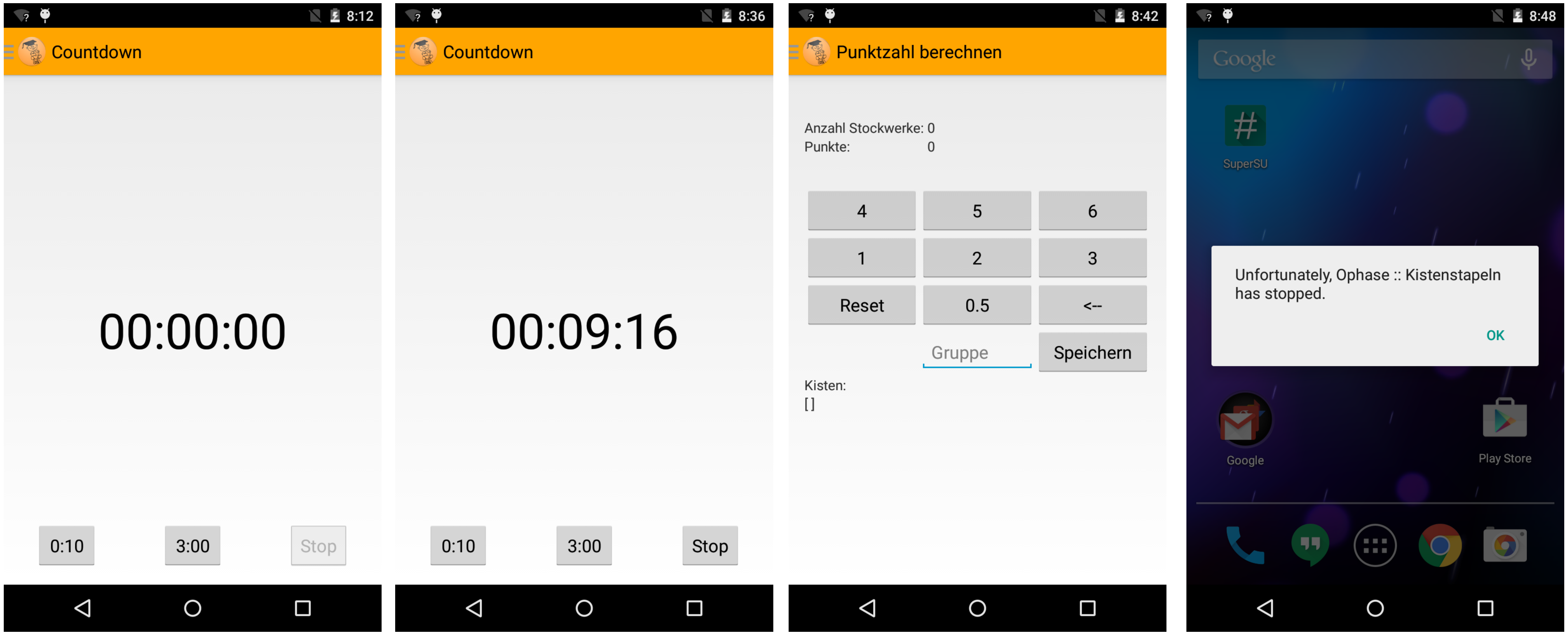
Typestate an Incomplete solution


Transition happens automatically. The callback "onActivityCreated" is invoked to inform the application.
Fragment
<init>
getString(…)
getString(…)
Exception

An application must react to callbacks to react to such transitions, this is not handled by typestate.

Template

Detection is difficult.



Anzahl Stockwerke: 0
Punkte: 0

Kisten:
[]

Unfortunately, Ophase :: Kistenstapeln has stopped.

Pressing buttons at the right time is required to cause the crash, so a better method is needed.