Towards Verifying Android Apps for the Absence of No-Sleep Energy Bugs

Panagiotis Vekris

Ranjit Jhala, Sorin Lerner, Yuvraj Agarwal

University of California, San Diego



UCSDCSE

Computer Science and Engineering

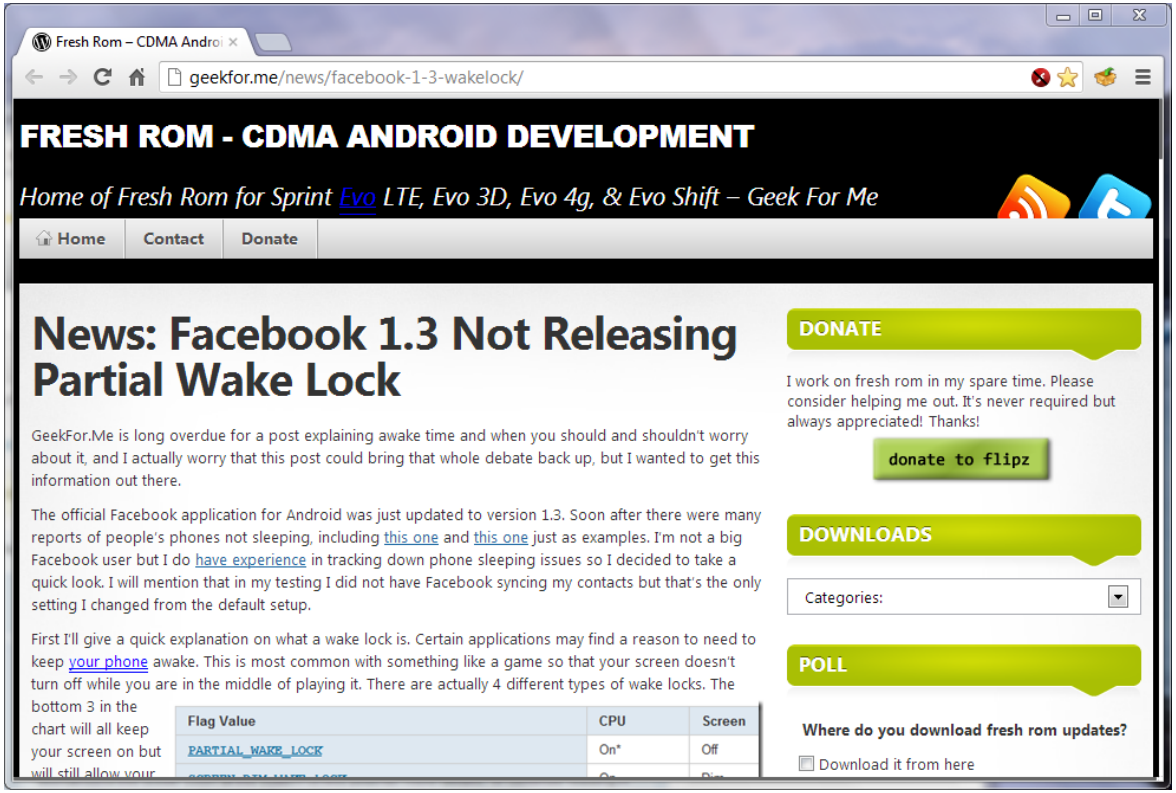

14:59:45
15:00:14
Battery information
Health: Good
Voltage: 3925mV
Technology: Li-ion
Temp: 34.3°C
Level: 2%
0%
100%
OK
Phone
Please connect charger
The battery is getting low:
less than 1% remaining.
OK
129
METERS
0.0KMH
1.8KM
00:02
17:16
GPS

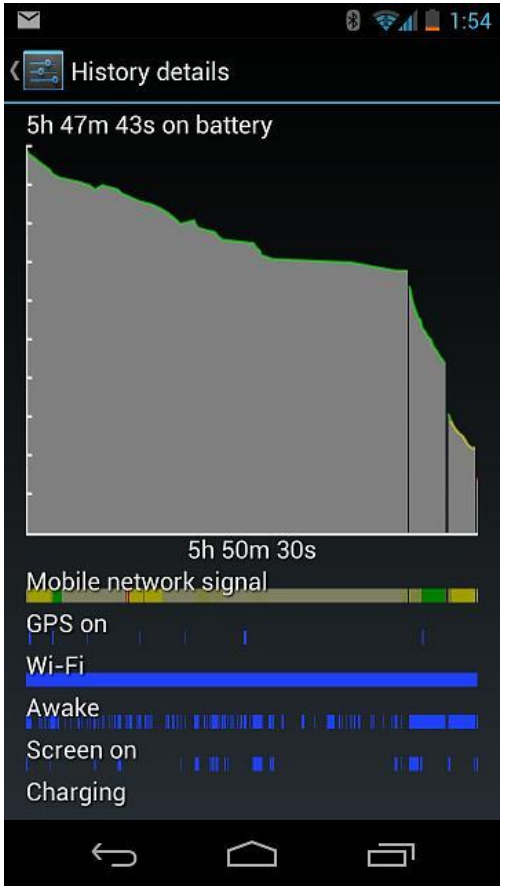
Software Energy Bugs





Software Energy Bugs





Power Manager



Concept

Opportunistic suspend

WakeLock API

App developer directives

Keep a device component awake

WakeLock Use

Lock Based Mechanism

```
PowerManager pm = getSystemService(POWER);
WakeLock l = pm.newWakeLock( SCREEN );
l.acquire();
//... screen stays on ...
l.release();
```

WakeLock Use

Lock Based Mechanism

```
PowerManager pm = getSystemService(POWER);
WakeLock l = pm.newWakeLock( SCREEN );
l.acquire();
//... screen stays on ...
```

Screen on indefinitely

"No-sleep" bugs

Misuse of WakeLock API

Screen Awake => 25% battery drain per hour [Pathak'12]

Static Analysis

Track No-Sleep Bugs

Pathak [MobiSys'12]

- *Reaching definitions dataflow analysis*
- *Detect known instances & find new ones*

Our Work

- *Define policies => Verify absence of bug*
- *Track asynchronous calls*
- *Precise dataflow analysis*

Activity

A screen to interact with

Application Component

Building block

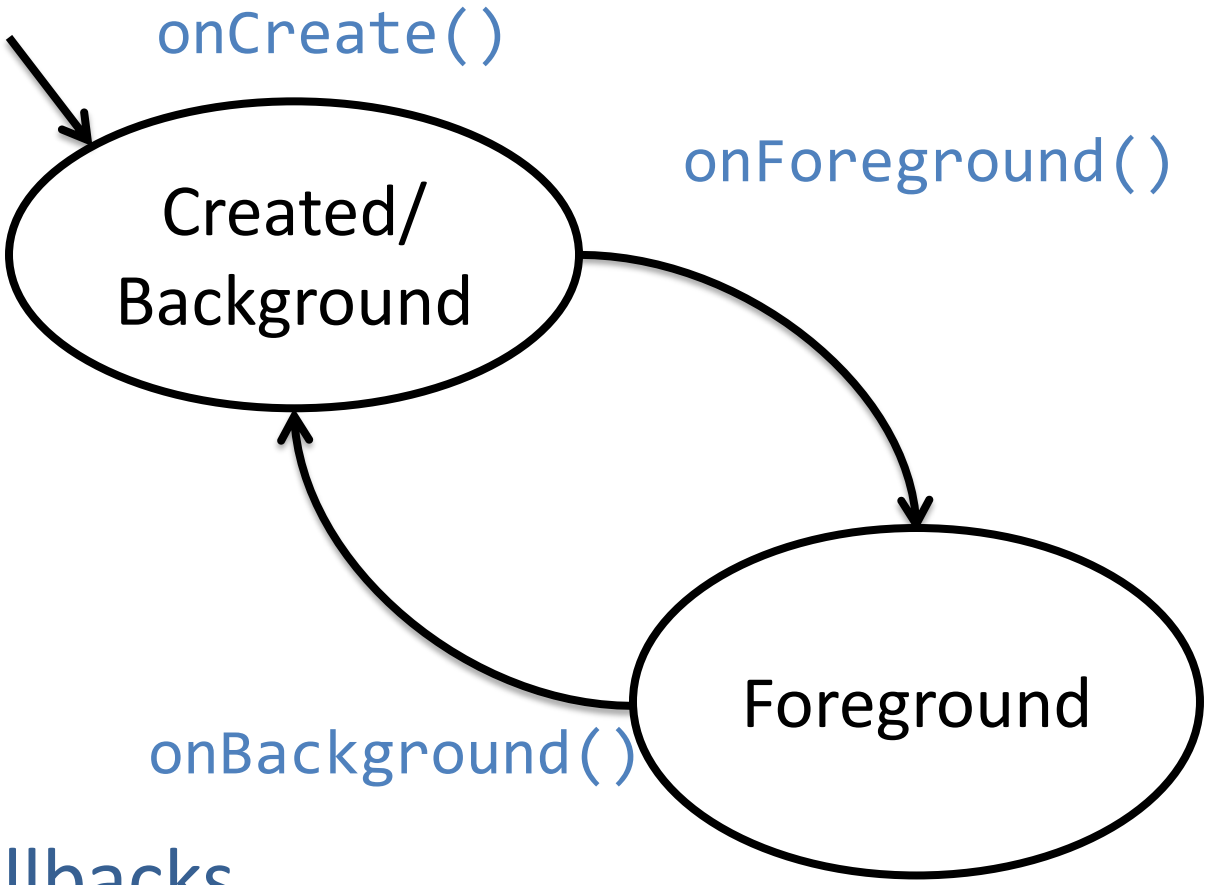
Entry point

Stack Model

Top of stack = visible activity



Activity States







Callbacks

Called by system on state transitions

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock( SCREEN );
}

onForeground()



onBackground()
```

- **Create a WakeLock**
- **Associate with screen**

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock( SCREEN );
}

onForeground() {
 // ...
 l.acquire();
}

onBackground()
```



```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock( SCREEN );
}

onForeground() {
 // ...
 l.acquire();
}

onBackground()
```



On Foreground with Screen WakeLock held

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock( SCREEN );
}

onForeground() {
 // ...
 l.acquire();
}

onBackground() {
 // ...
 l.release();
}
```

Release WakeLock l

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock( SCREEN );
}

onForeground() {
 // ...
 l.acquire();
}

onBackground() {
 // ...
 l.release();
}
```



Correct Use

At Background with Screen WakeLock released

iZen Lite V2.3

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock(CPU);
   l.acquire();
}

//unimplemented
onForeground();

//unimplenented
onBackground();
```

Acquire CPU WakeLock

Application in background

iZen Lite V2.3

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock(CPU);
   l.acquire();
}

//unimplemented
onForeground();

//unimplenented
onBackground();
```

Acquire CPU WakeLock

Application in background holds CPU WakeLock

iZen Lite V2.3

```
onCreate() {
   super.onCreate();
   //  setting up
   l = newWakeLock(CPU);
   l.acquire();
}

//unimplemented
onForeground();

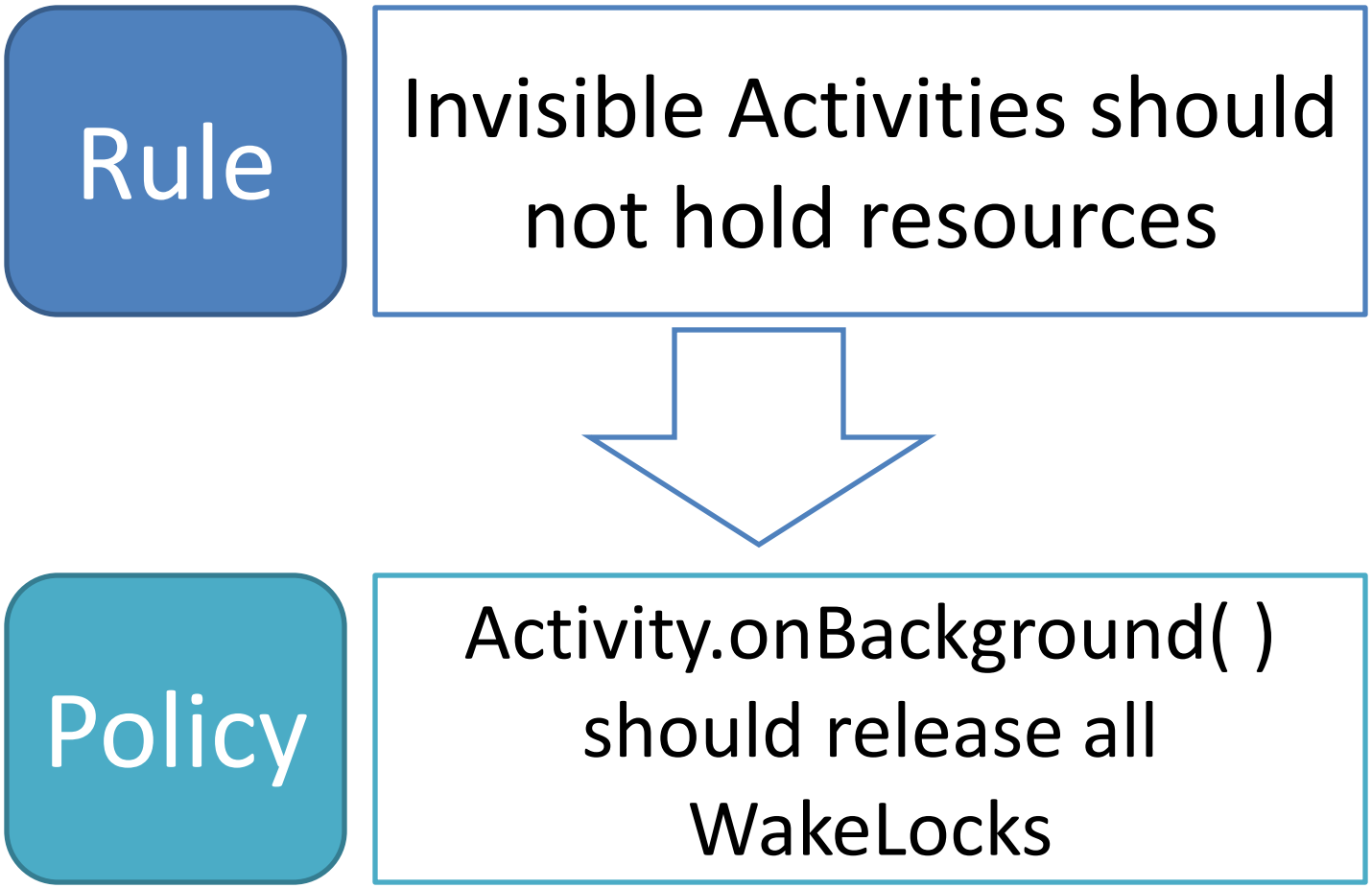
//unimplenented
onBackground();
```

Acquire CPU WakeLock

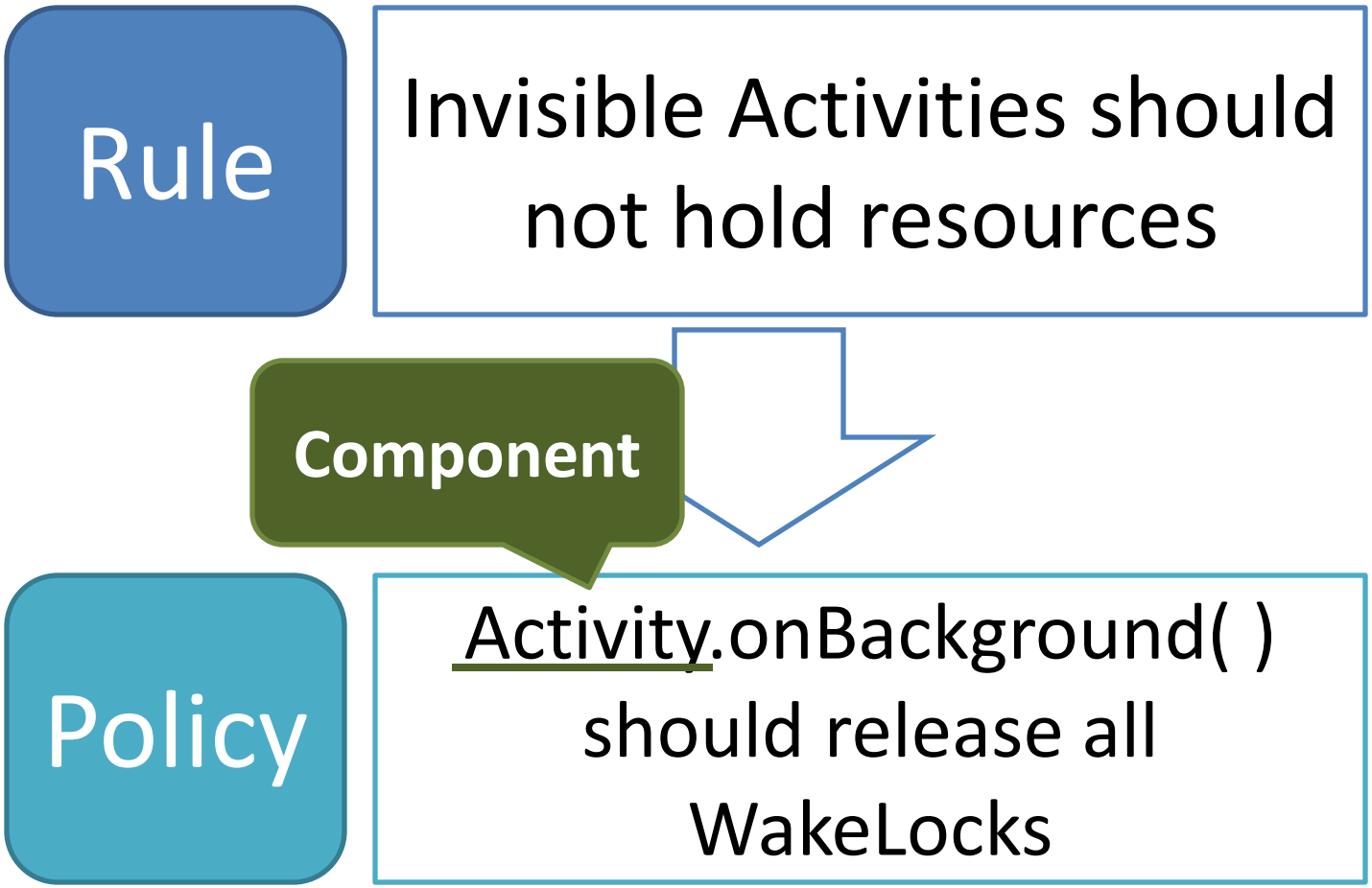
Application in background holds CPU WakeLock without doing useful work

Incorrect Use

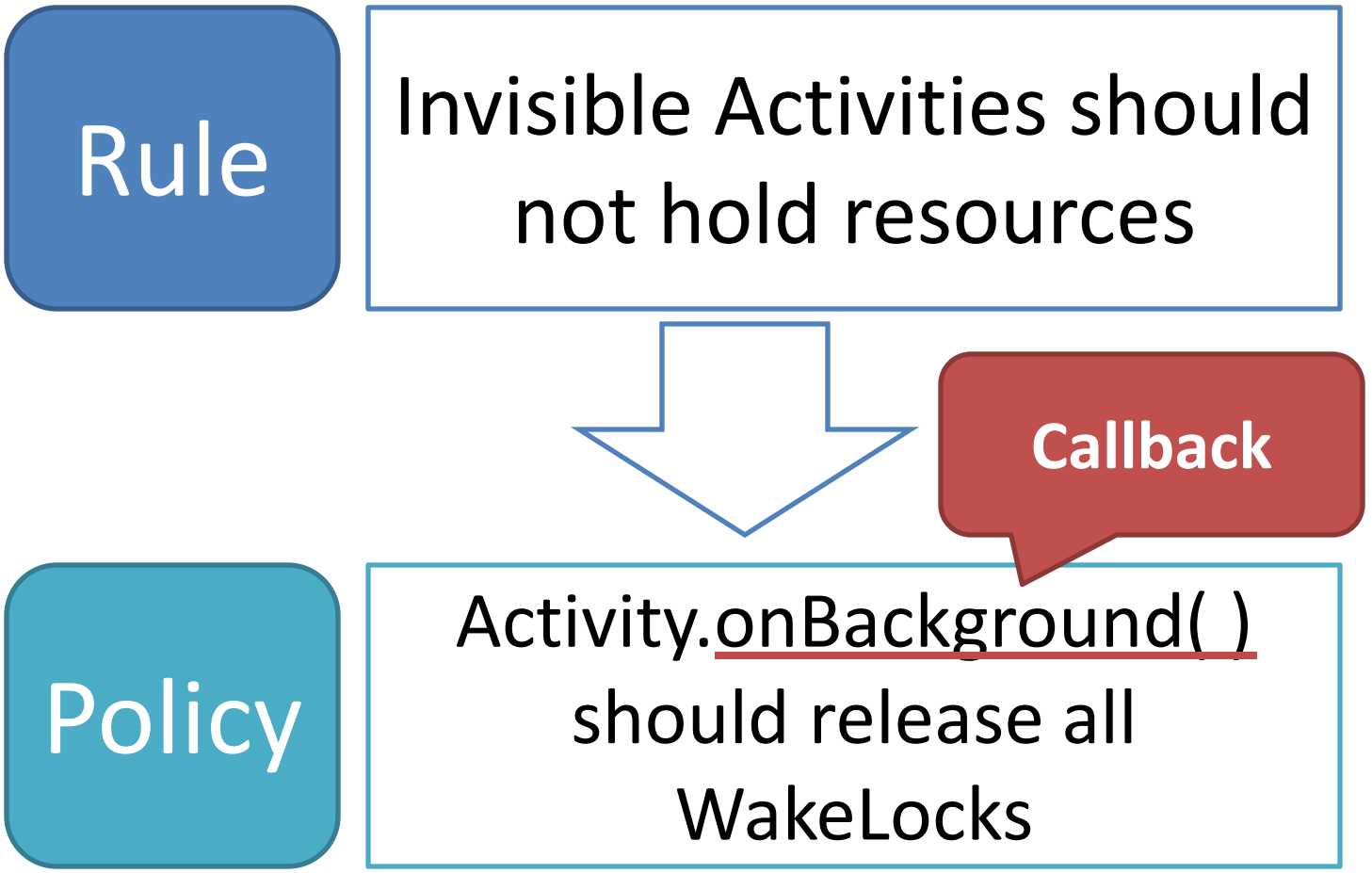
Policies


Rule
Invisible Activities should not hold resources
Policy
Activity.onBackground( ) should release all WakeLocks

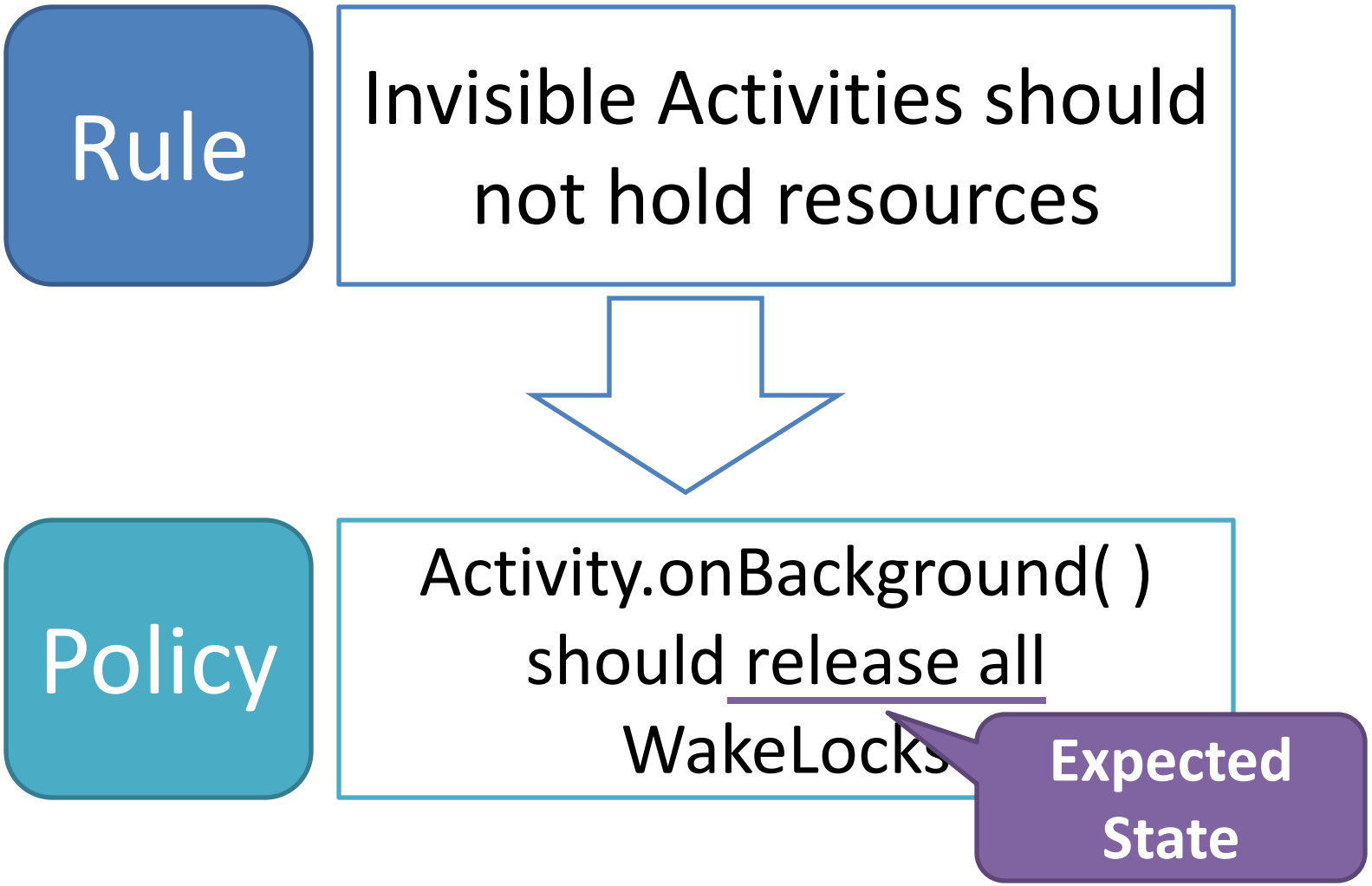
Policies


Rule
Invisible Activities should not hold resources
Component
Policy
Activity.onBackground( ) should release all WakeLocks

Policies


Rule
Invisible Activities should not hold resources
Callback
Policy
Activity.onBackground( )
should release all WakeLocks

Policies


Rule
Invisible Activities should not hold resources
Policy
Activity.onBackground( ) should release all WakeLocks
Expected State

Policies

WakeLocks

Used in any component - Not only Activities

Their effects cross component boundaries

General policies



We defined policies for basic components

Conservative for the rest

Static Analysis to enforce them

Tool Overview

Uses a precise Dataflow algorithm [RHS' 95]

State: Set of WakeLocks that

may *be held*

at given program point

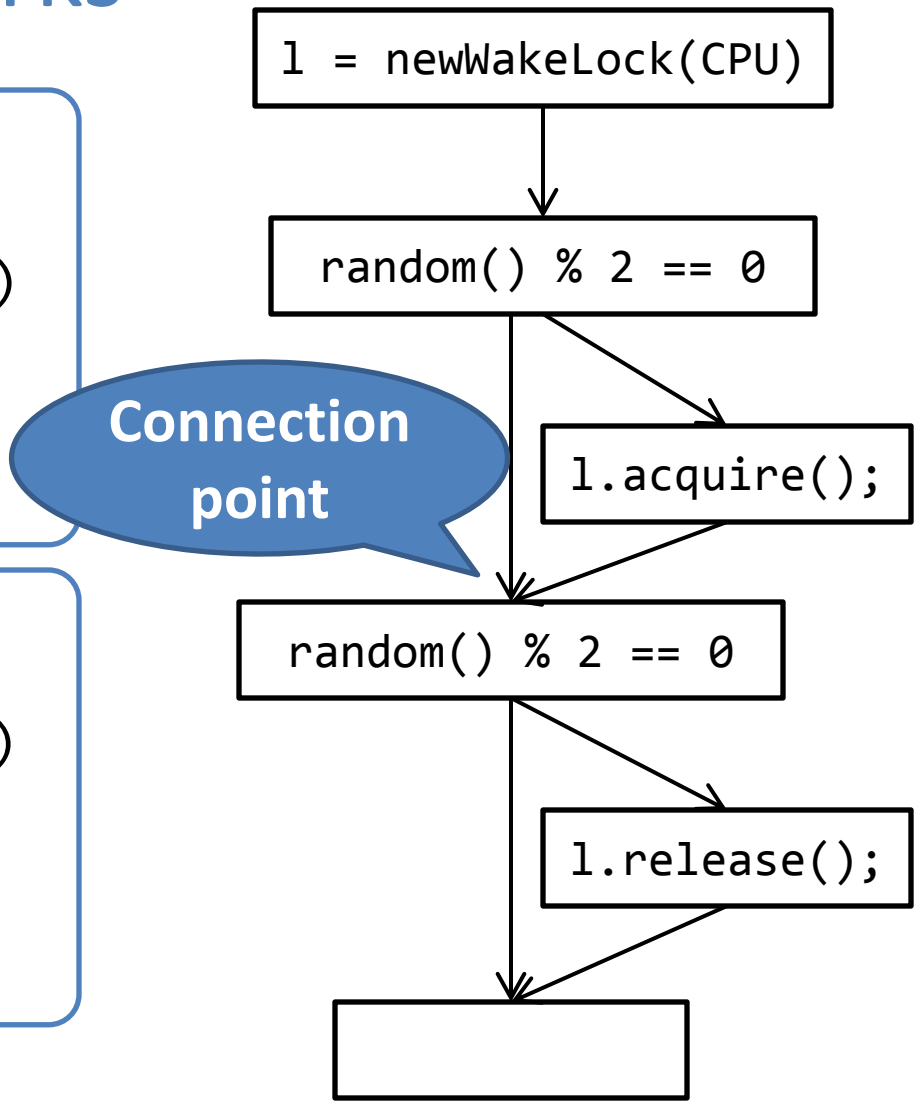
Validate

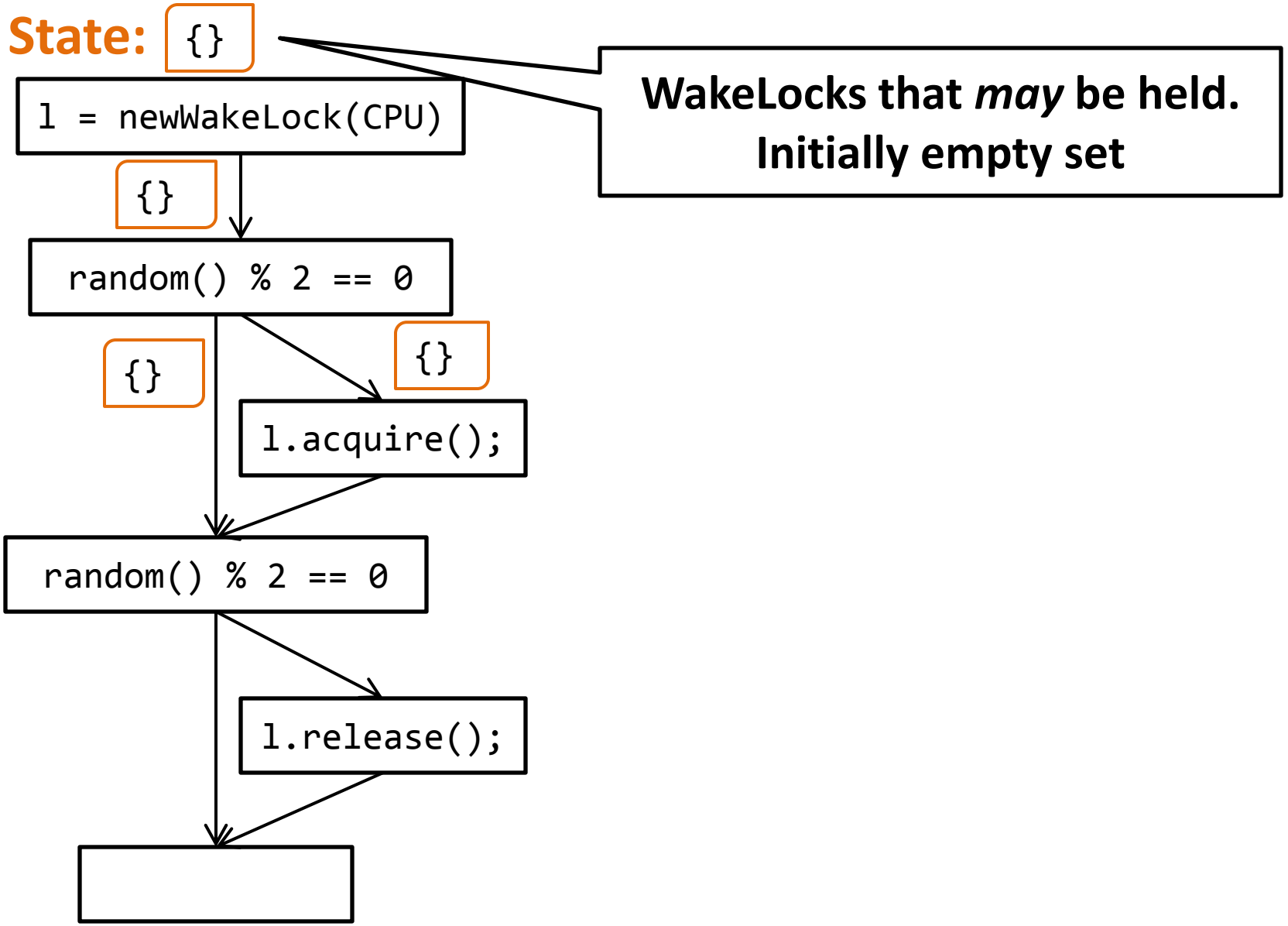
Check if state complies to policies

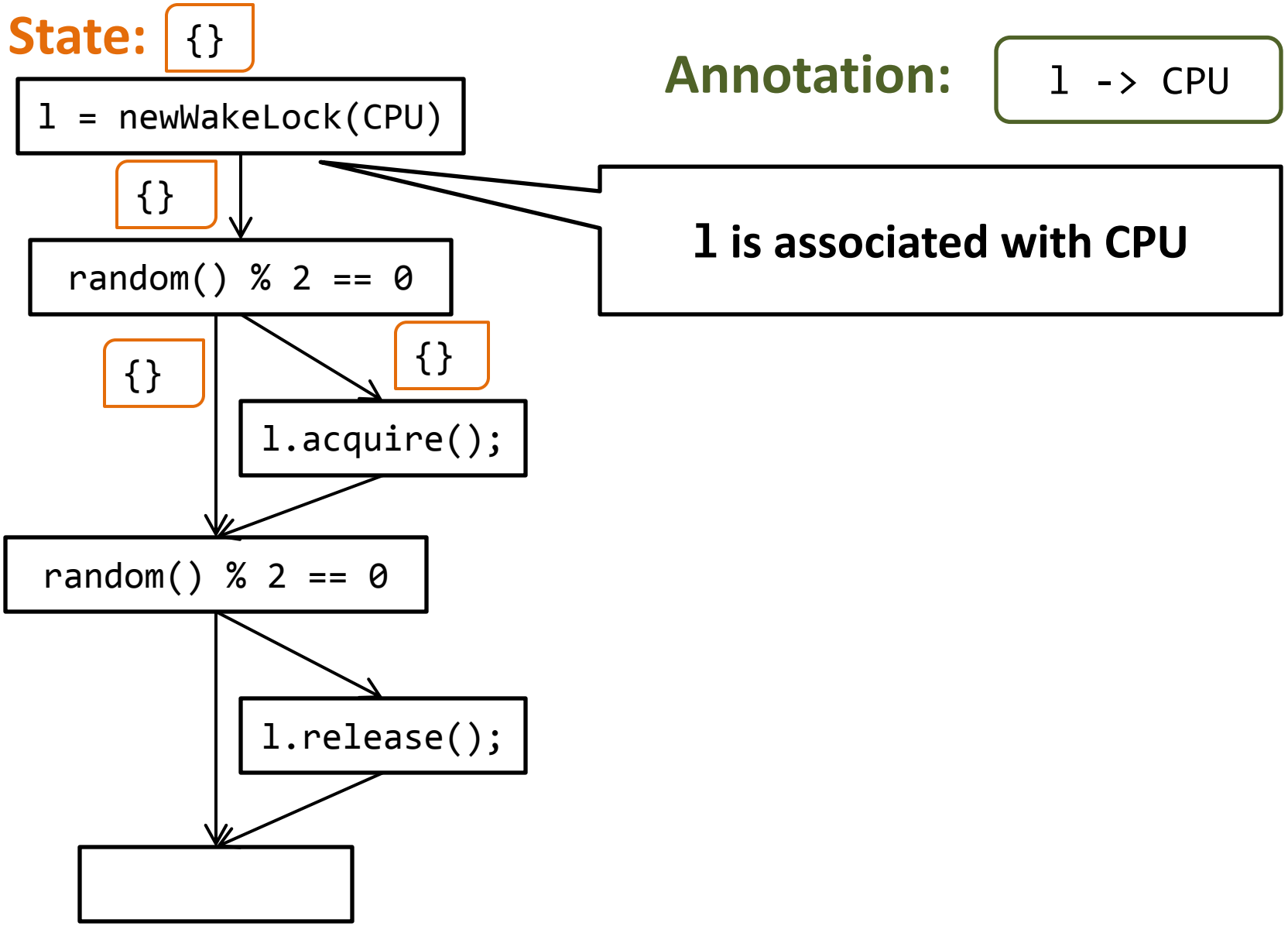
How our tool works

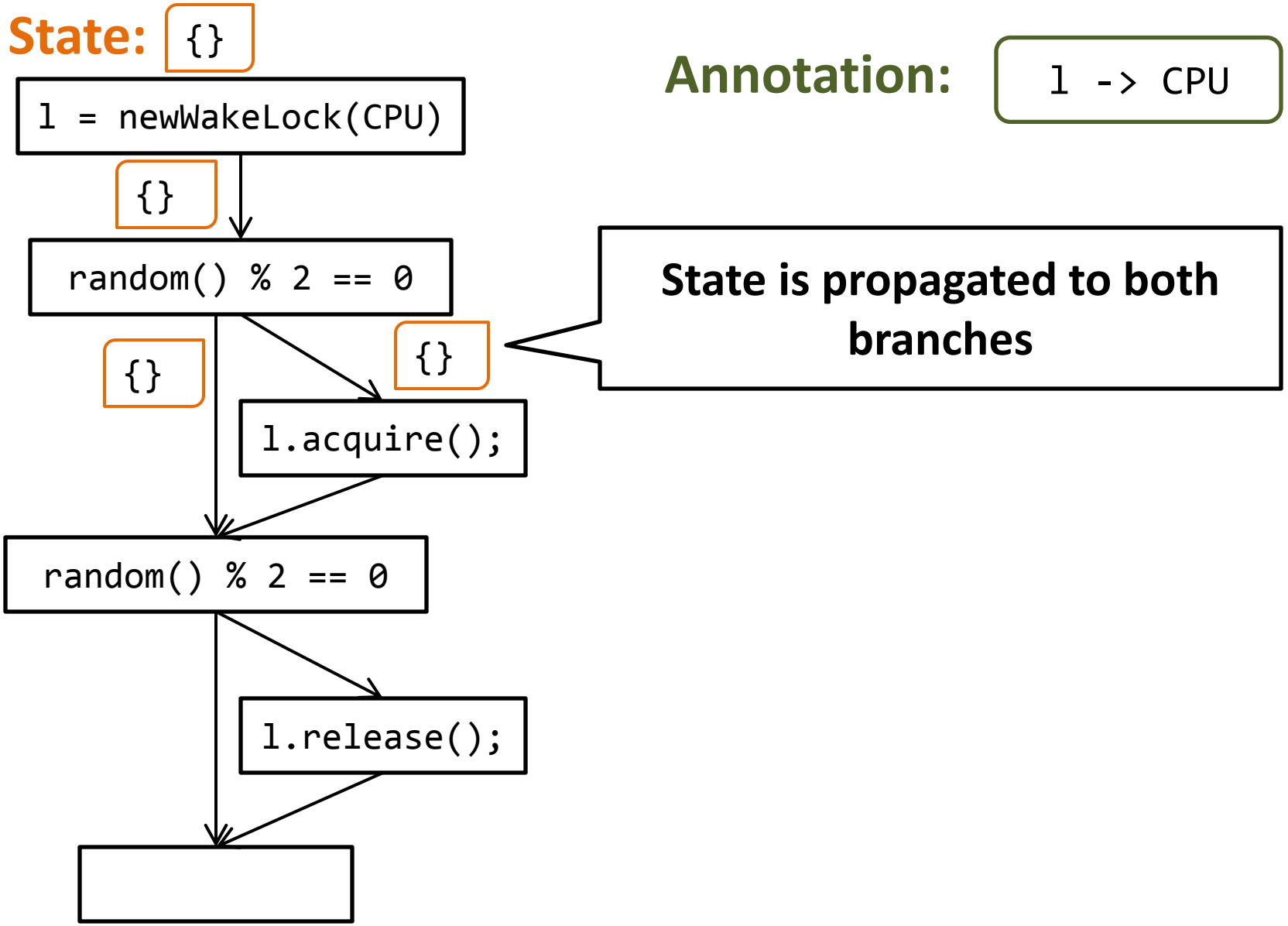
```
onForeground() {
  l = newWakeLock(CPU);
  if (random() % 2 == 0)
    l.acquire();
}
```

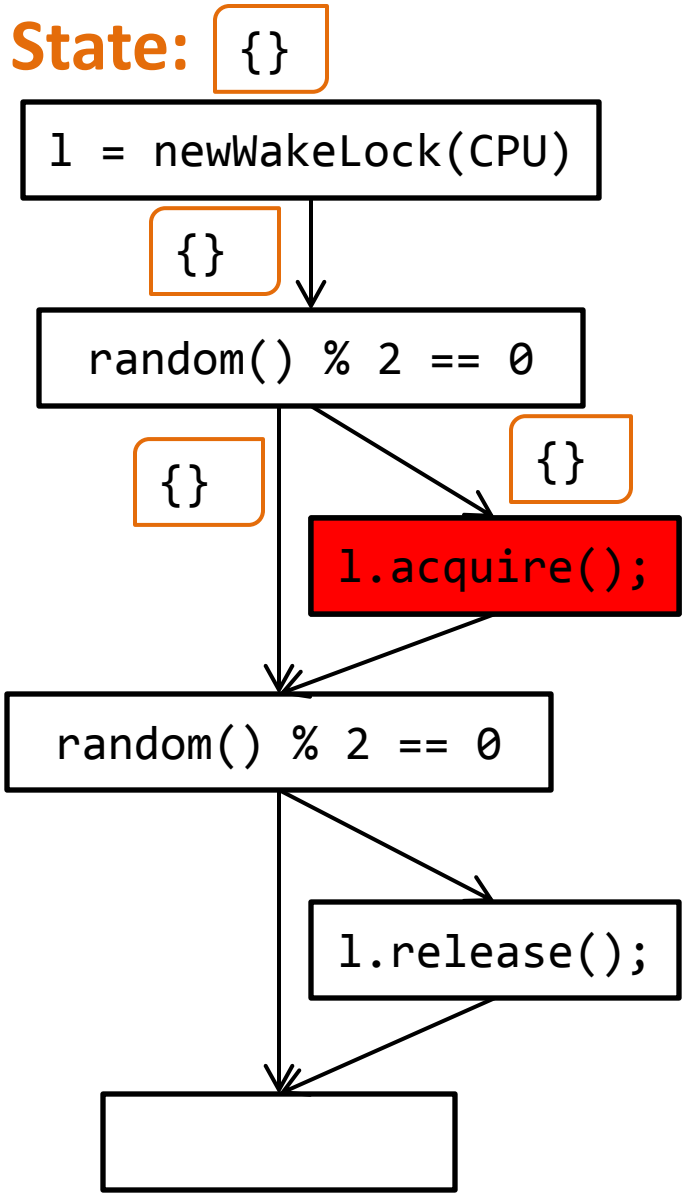
```
onBackground() {
  if (random() % 2 == 0)
    l.release();
}
```



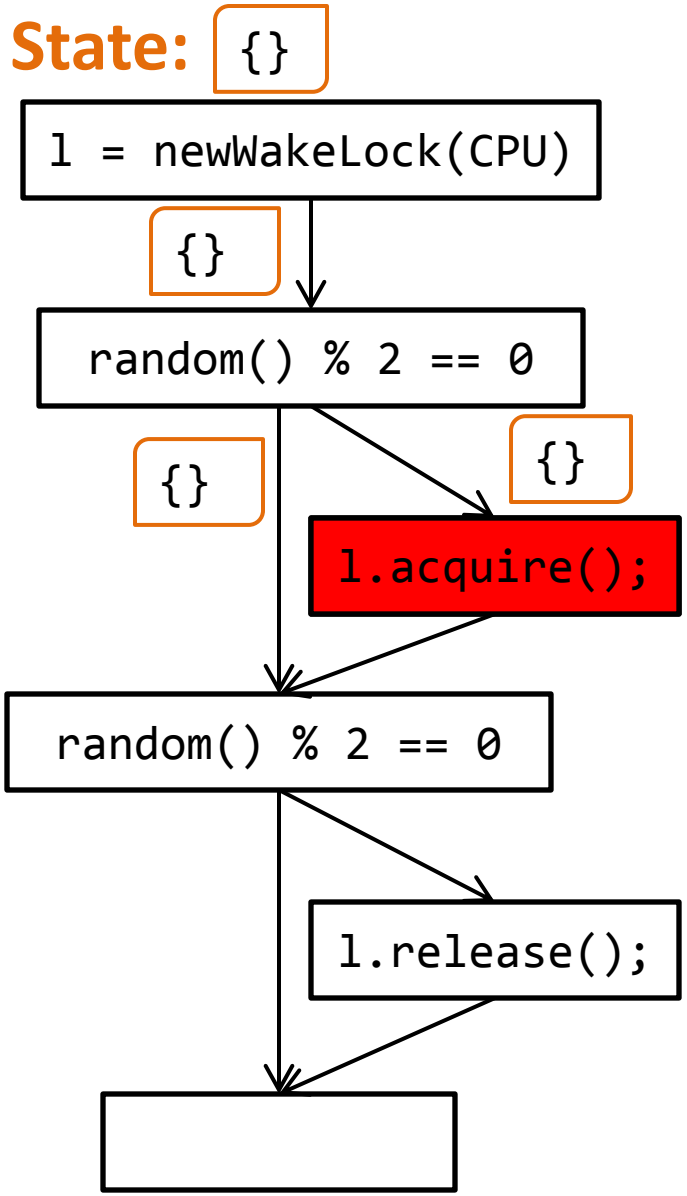

State:
{}
WakeLocks that *may* be held.
Initially empty set
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
random() % 2 == 0
l.release();

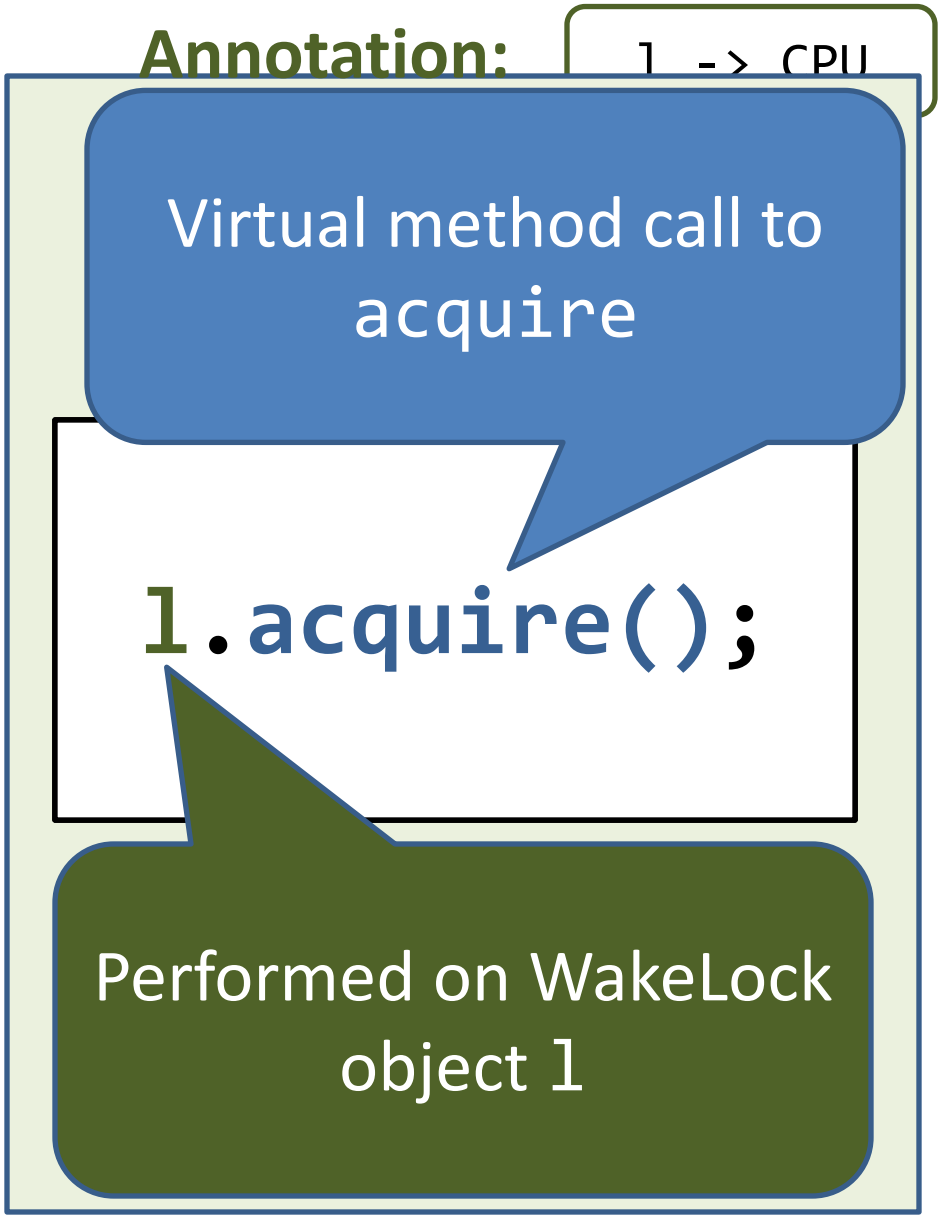

State:
{}
Annotation:
l -> CPU
l = newWakeLock(CPU)
{}
l is associated with CPU
random() % 2 == 0
{}
{}
l.acquire();
random() % 2 == 0
l.release();

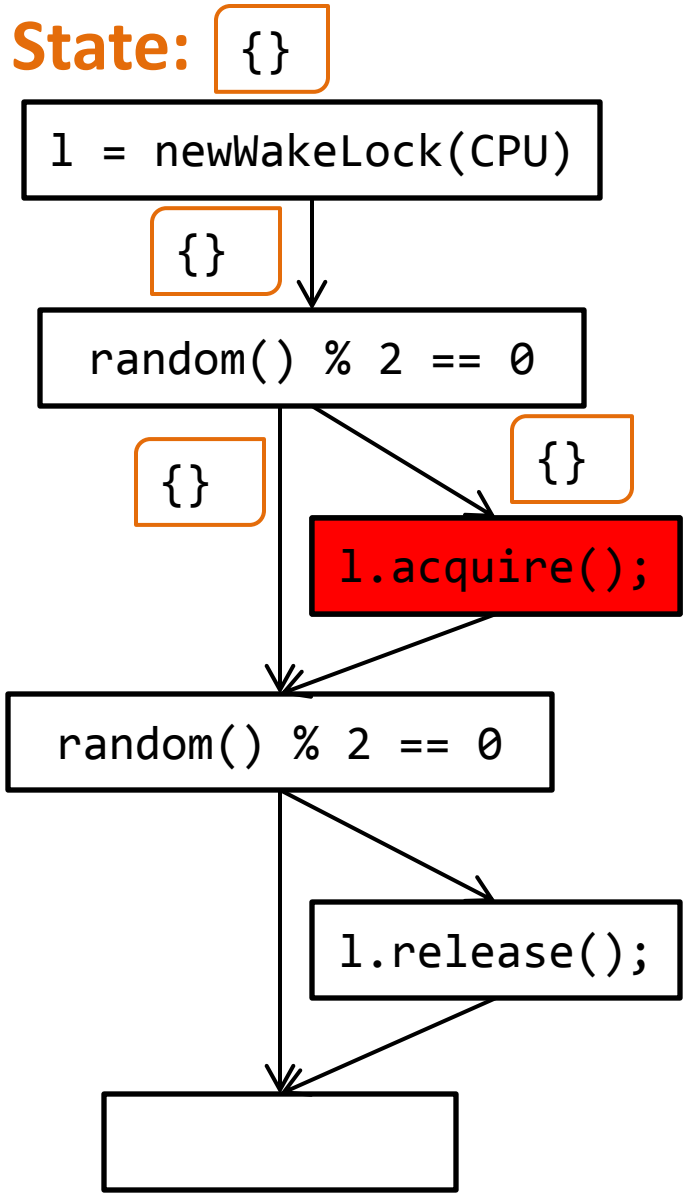

State:
{}
Annotation:
l -> CPU
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
State is propagated to both branches
l.acquire();
random() % 2 == 0
l.release();

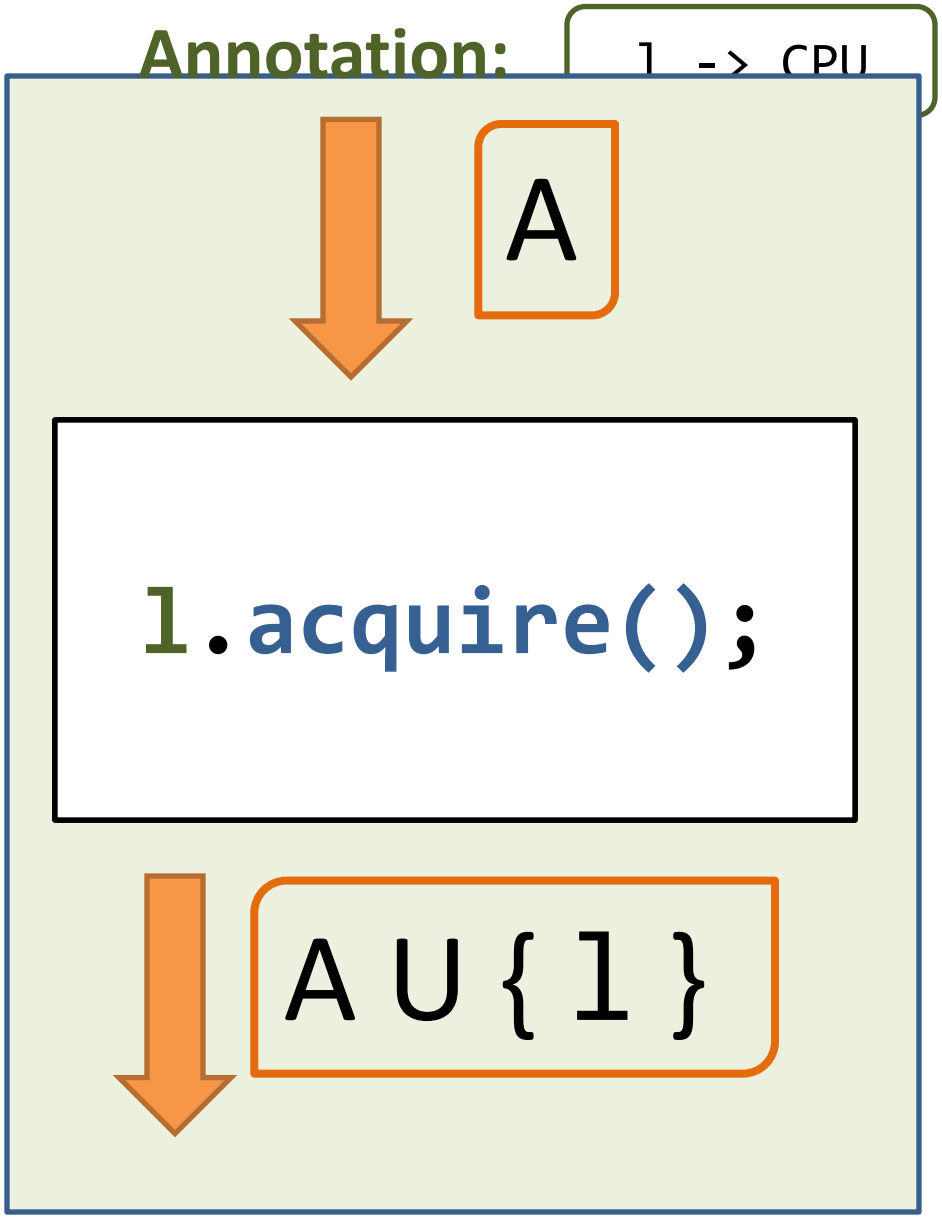


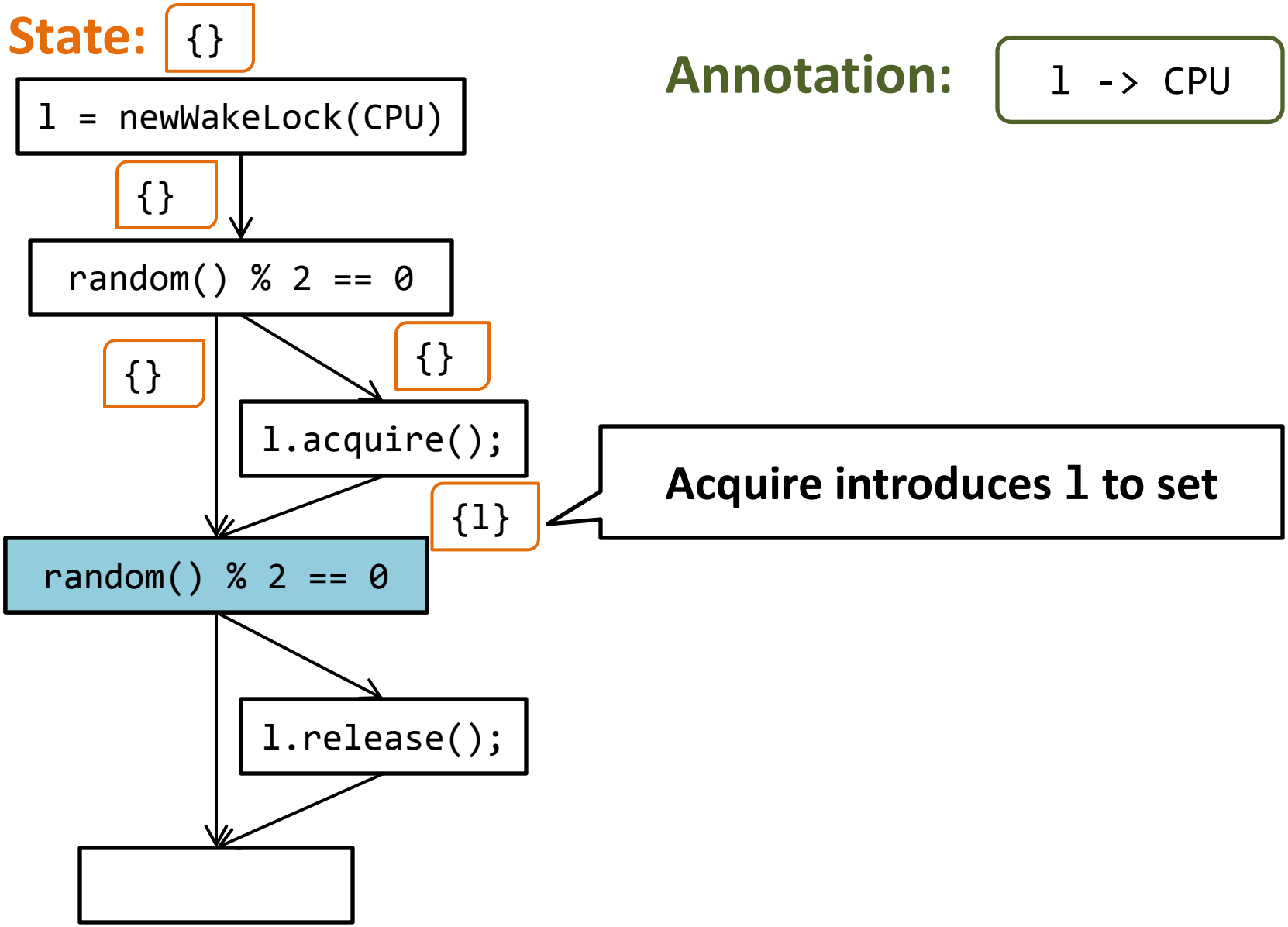
Annotation: `l -> CPU`

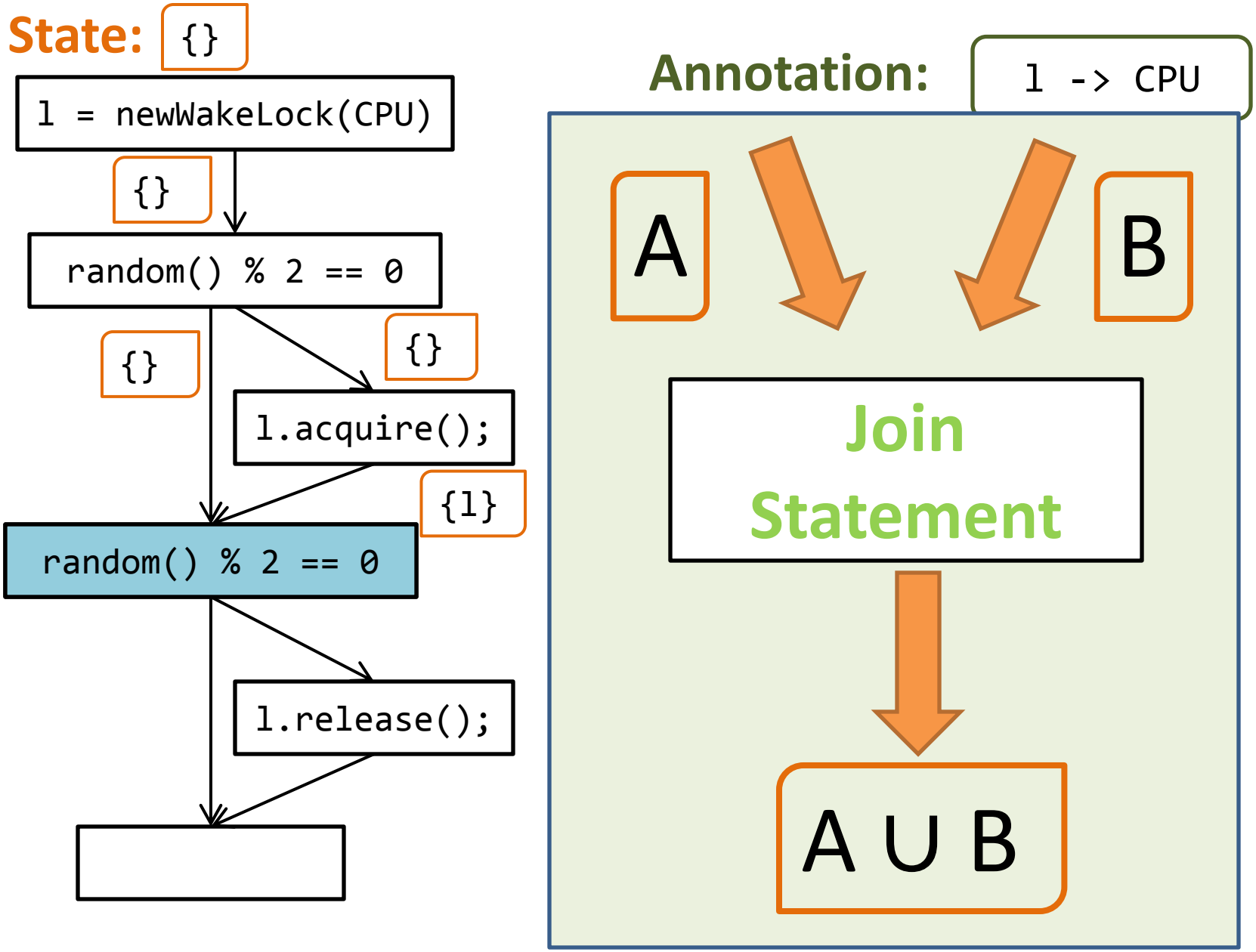

State:
{}
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
random() % 2 == 0
l.release();

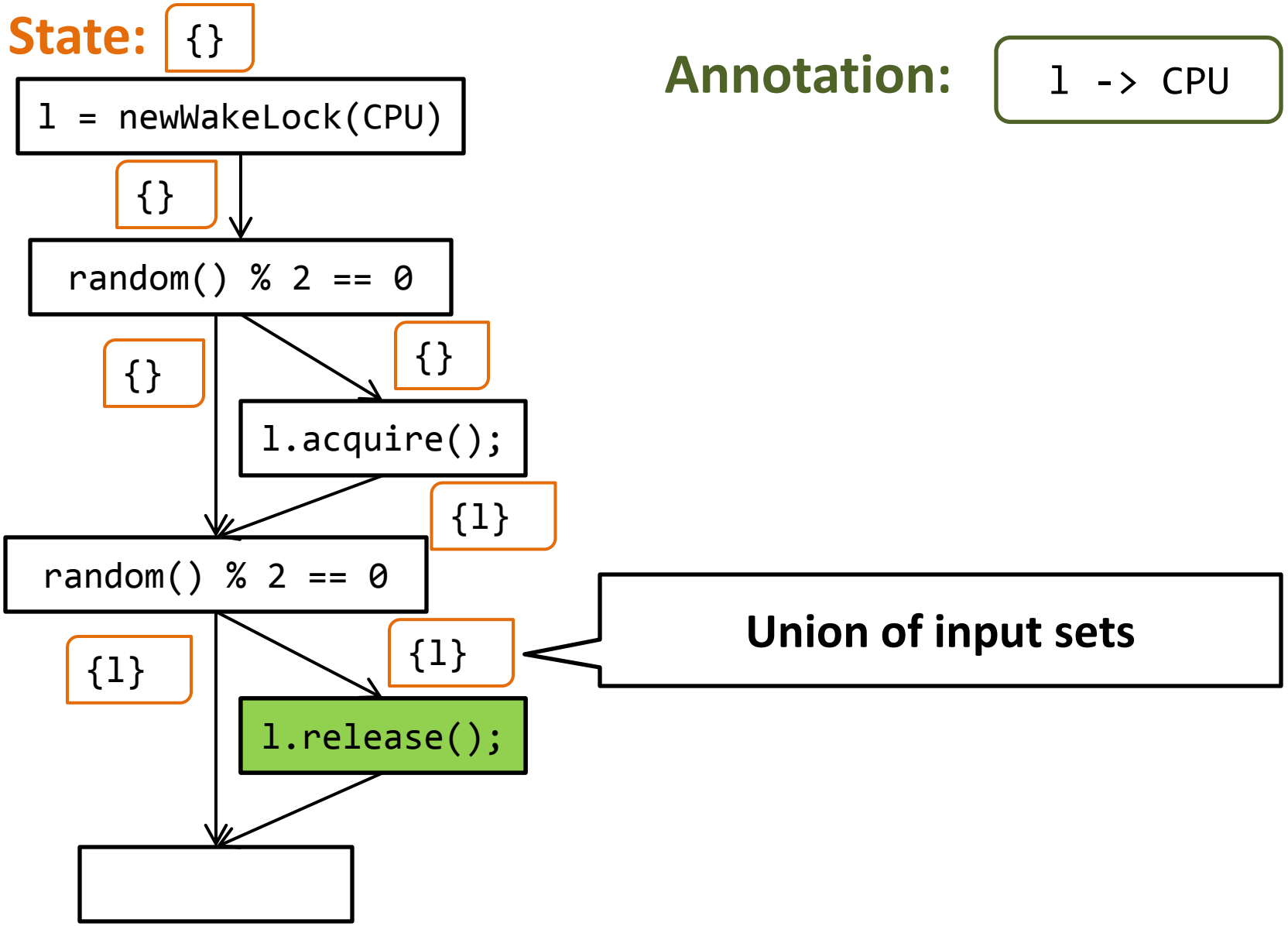

Annotation:
l -> CPU
Virtual method call to acquire
l.acquire();
Performed on WakeLock object l

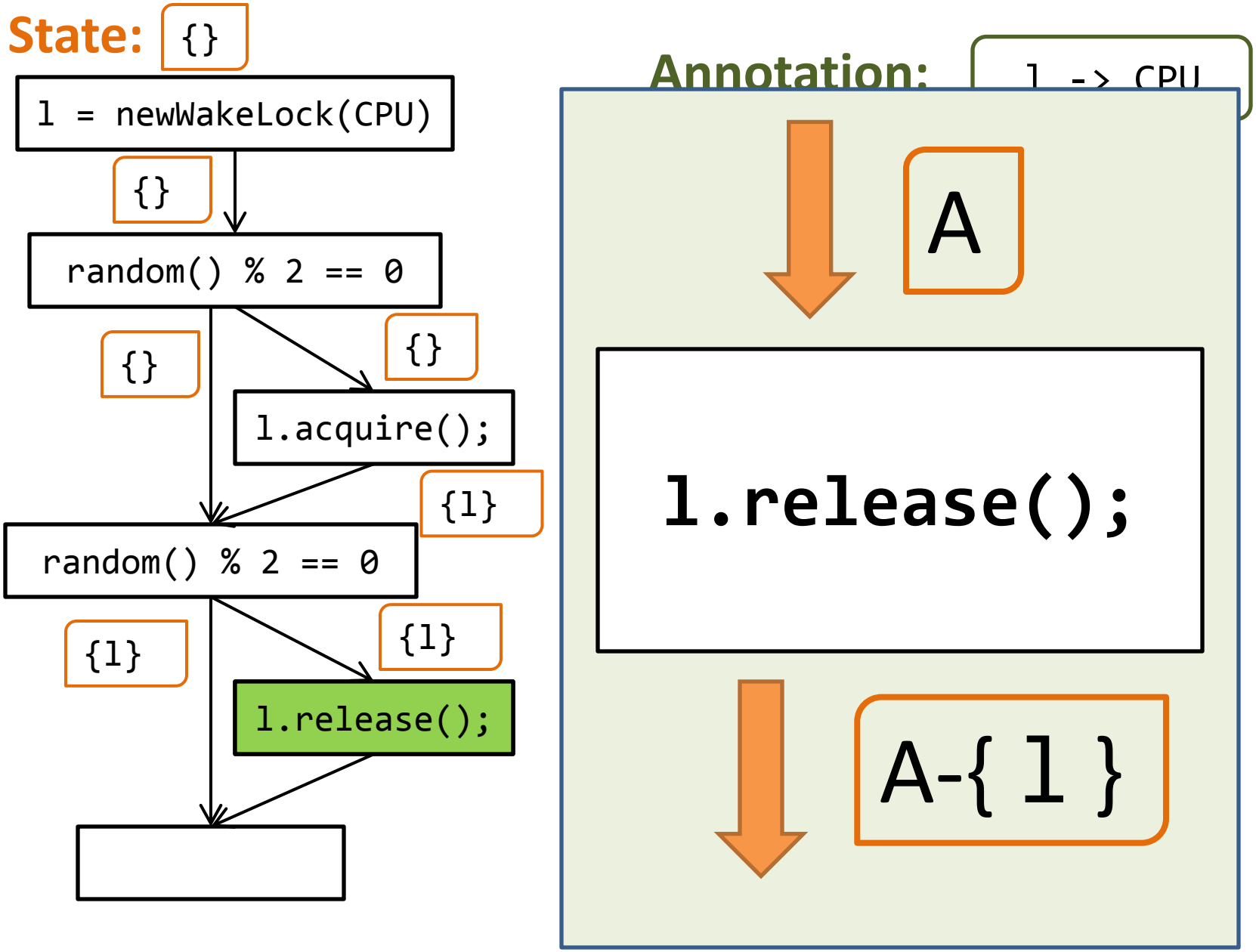

State:
{}
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
random() % 2 == 0
l.release();

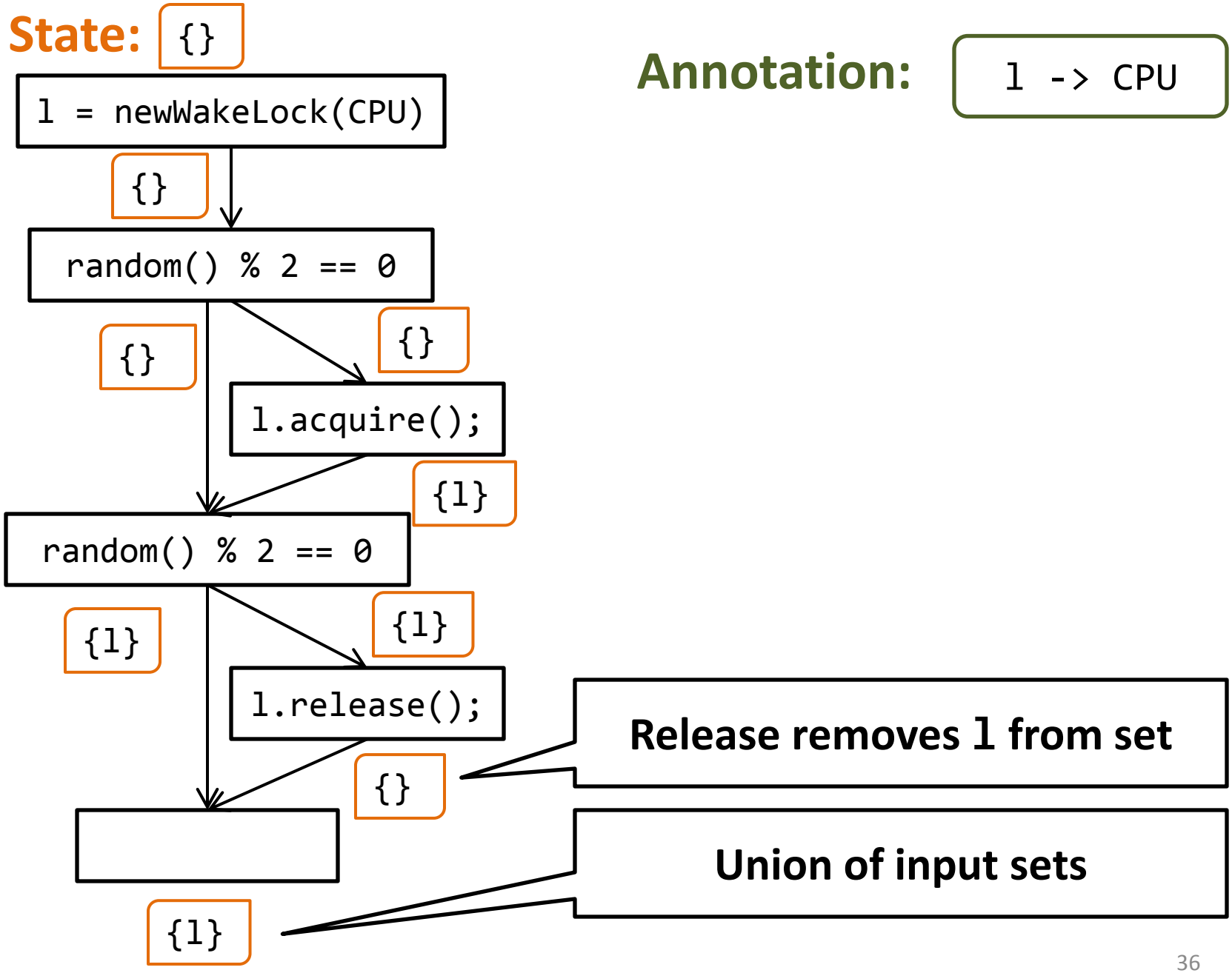

Annotation:
l -> CPU
A
l.acquire();
A U {l}

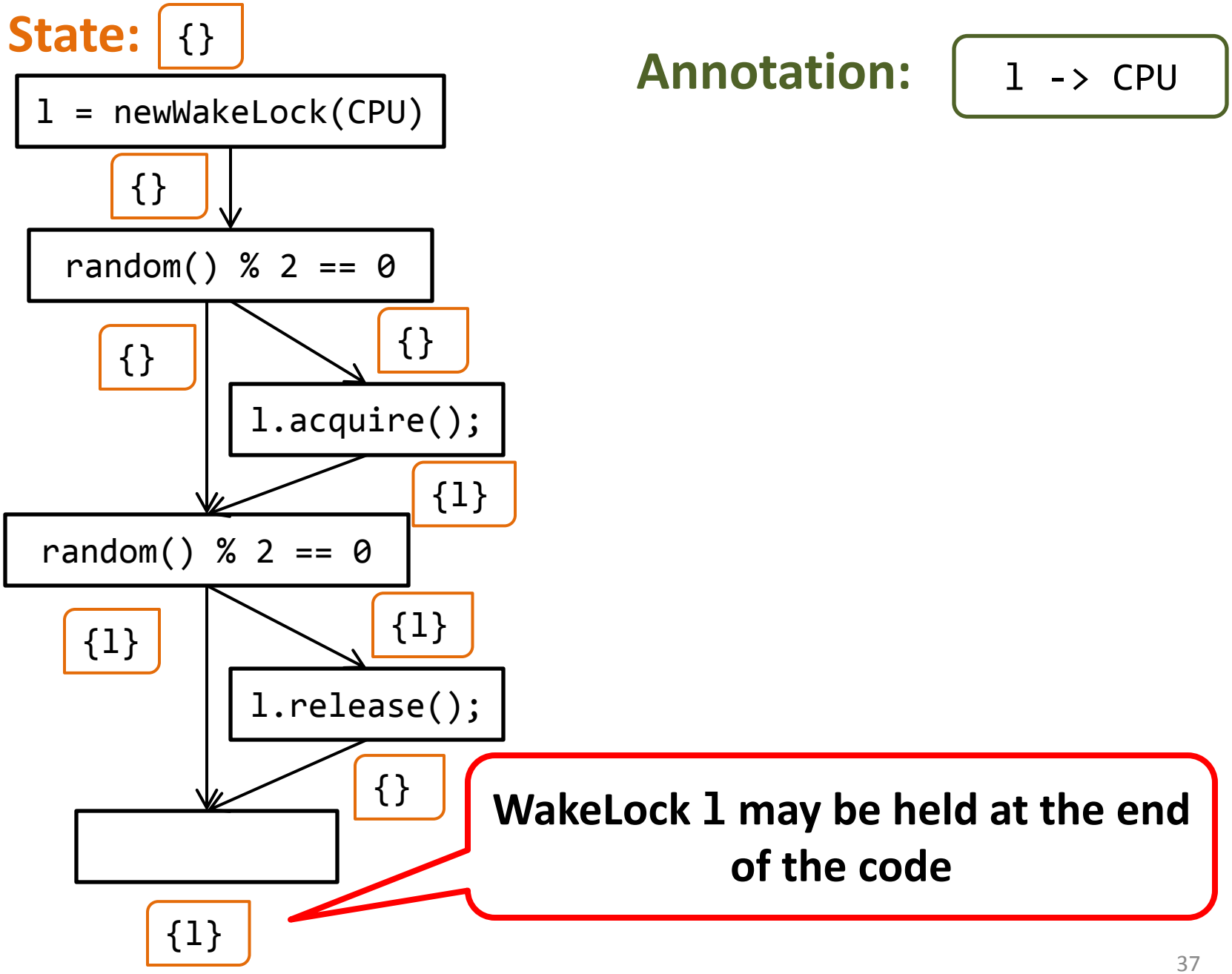

State:
{}
Annotation:
l -> CPU
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
Acquire introduces l to set
random() % 2 == 0
l.release();


State:
{}
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
random() % 2 == 0
l.release();
Annotation:
l -> CPU
A
B
Join
Statement
A U B


State:
{}
Annotation:
l -> CPU
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
random() % 2 == 0
{l}
{l}
Union of input sets
l.release();


State:
{}
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
random() % 2 == 0
{l}
{l}
l.release();
Annotation:
l -> CPU
A
l.release();
A-{ l }


State:
{}
Annotation:
l -> CPU
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
random() % 2 == 0
{l}
{l}
l.release();
{}
Release removes l from set
Union of input sets
{l}


State:
{}
l = newWakeLock(CPU)
{}
random() % 2 == 0
{}
{}
l.acquire();
{l}
random() % 2 == 0
{l}
{l}
l.release();
{}
{l}
Annotation:
l -> CPU
WakeLock l may be held at the end of the code

Path Sensitivity

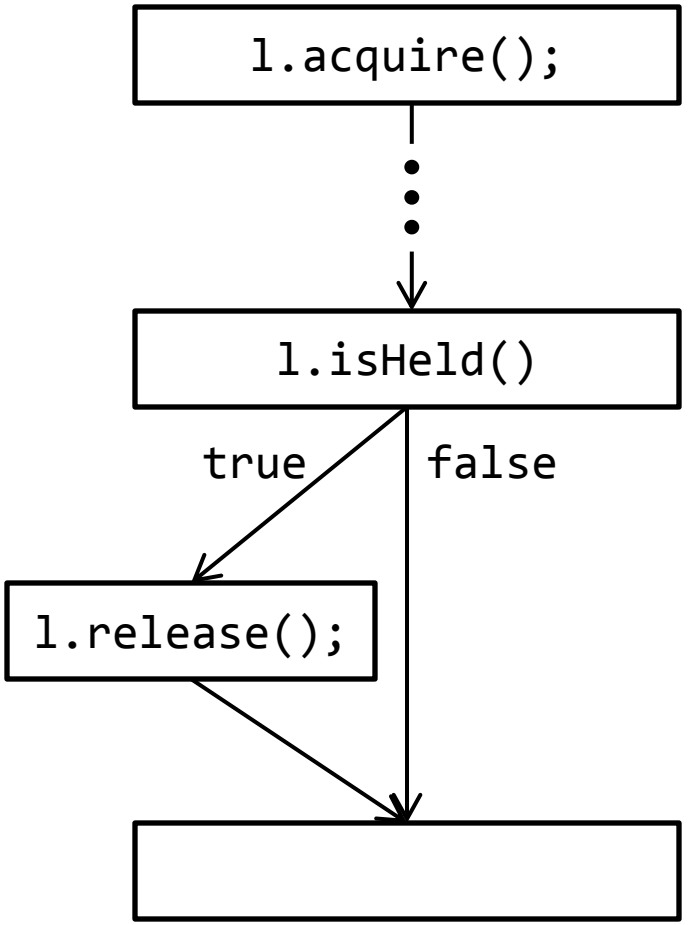
```
l.acquire();

// ... critical operation

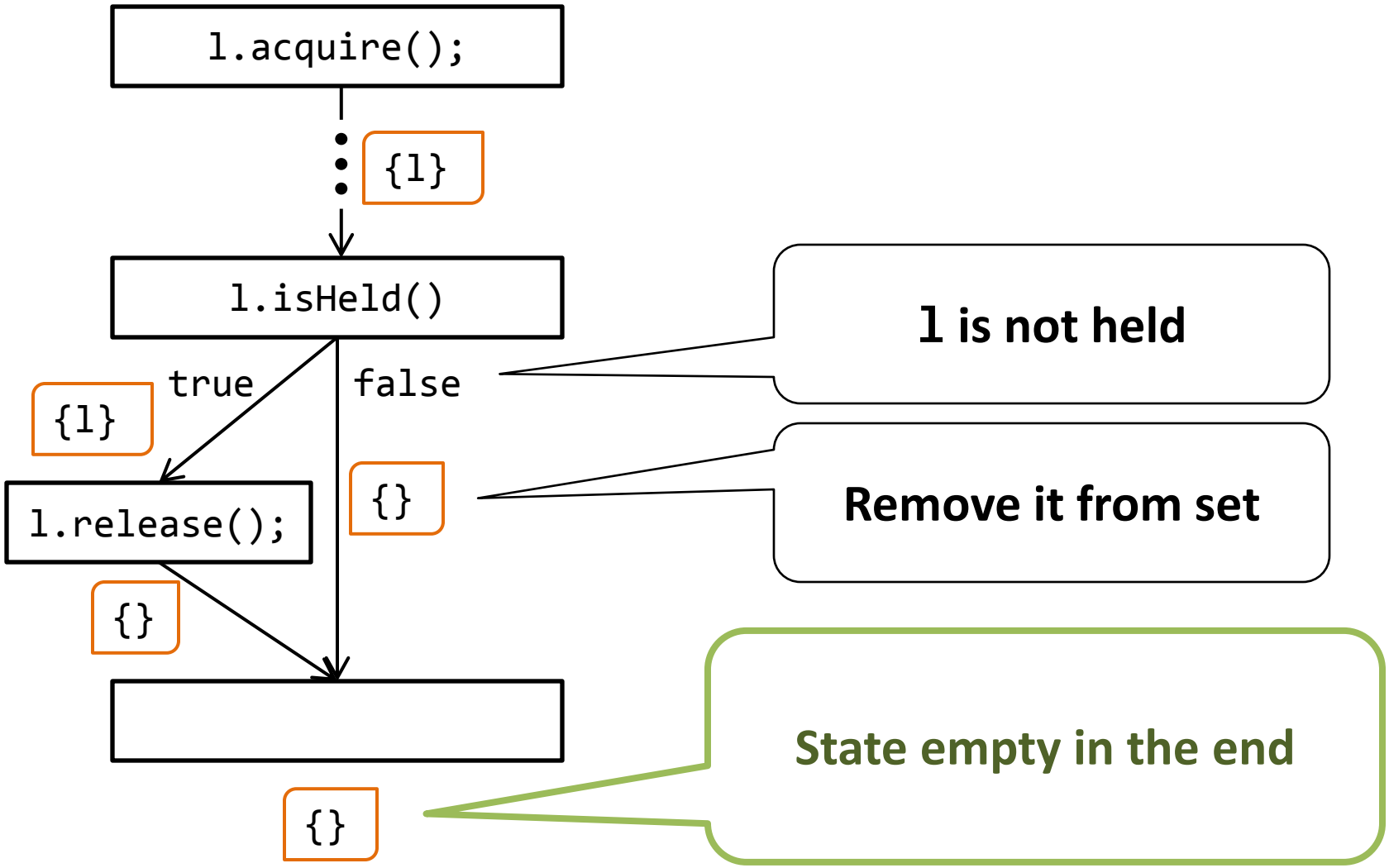
if (l.isHeld())

  l.release();

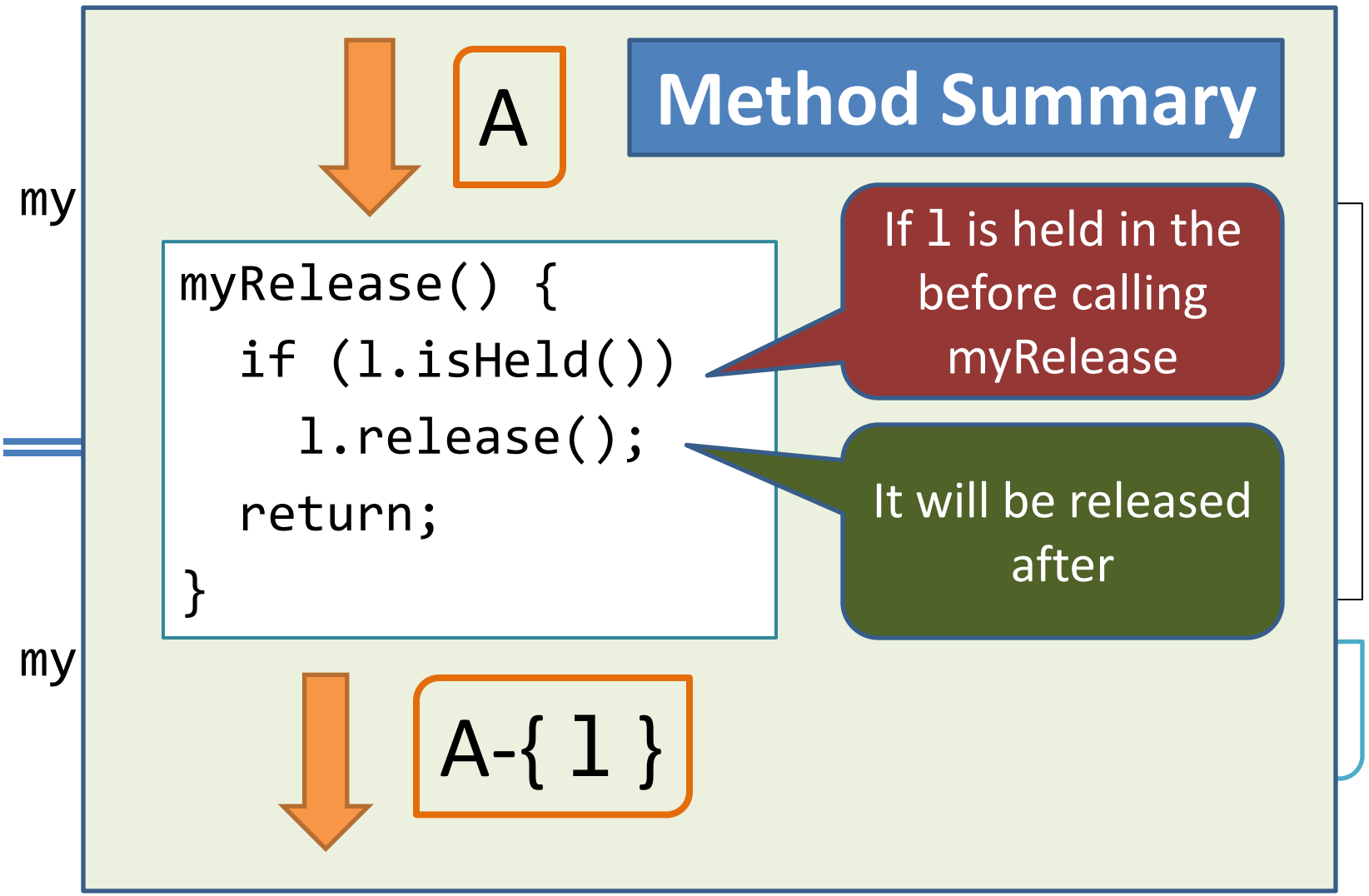
// ... Wakelock released
```



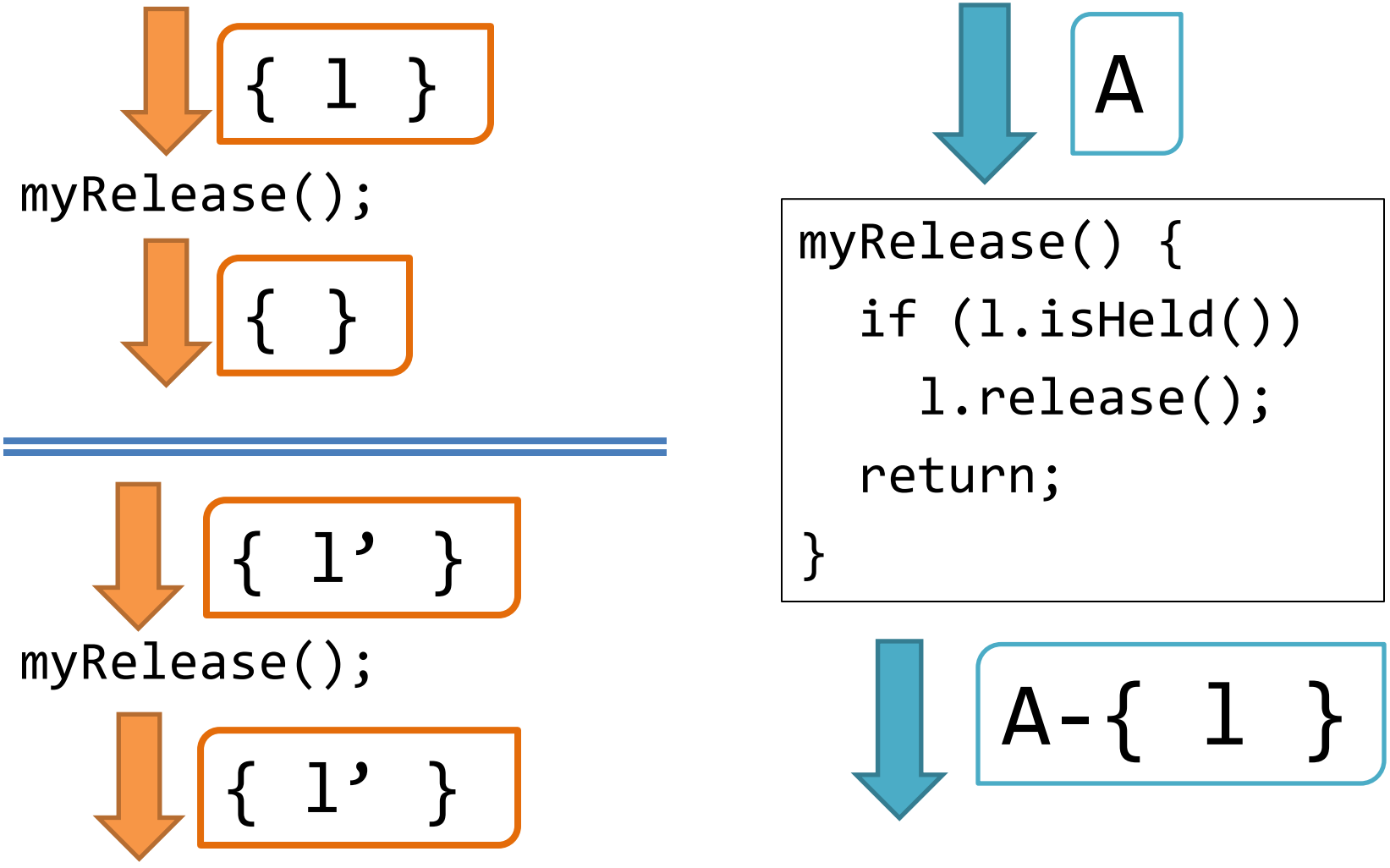
Path Sensitivity



Context Sensitive Method Calls



Context Sensitive Method Calls



Handling Asynchrony

Intent calls

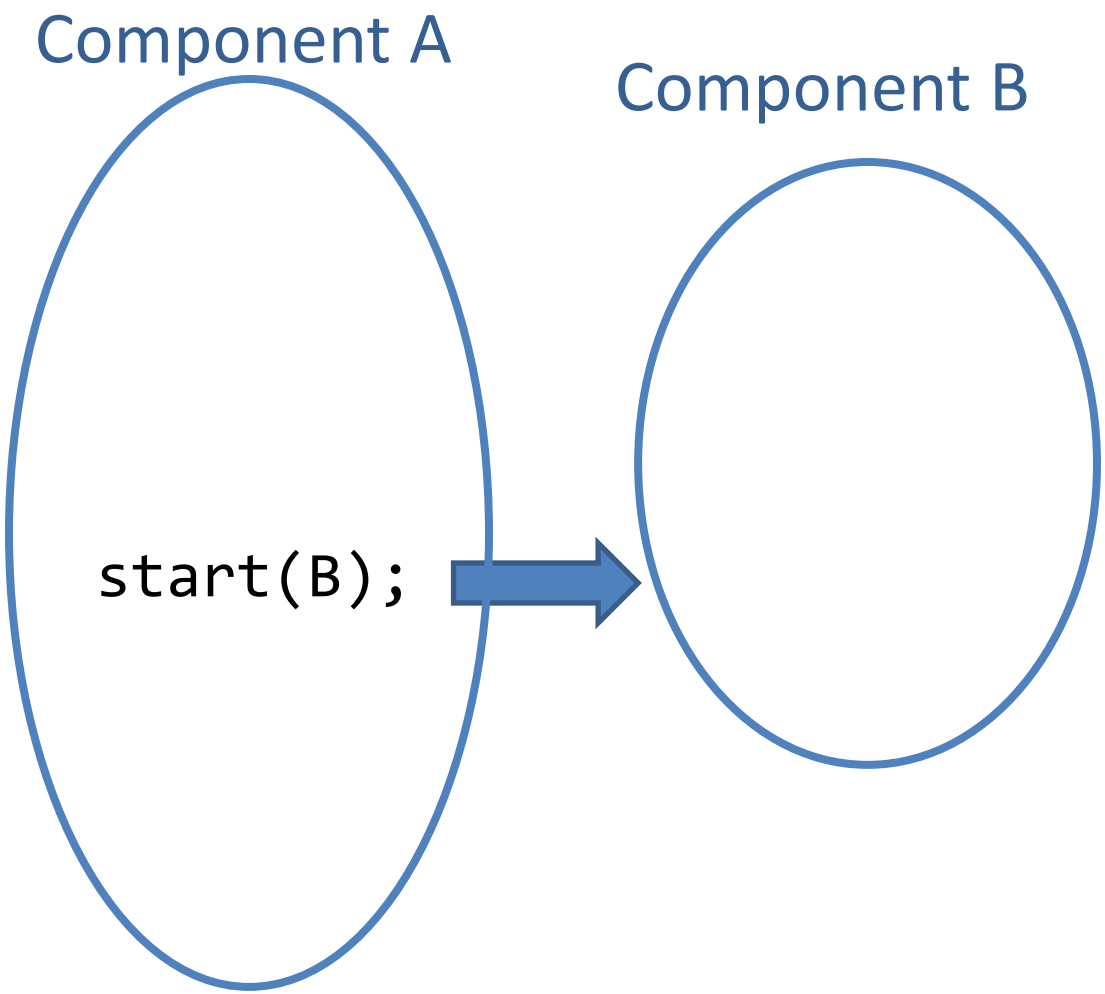
Asynchronous messages among components

Threaded execution

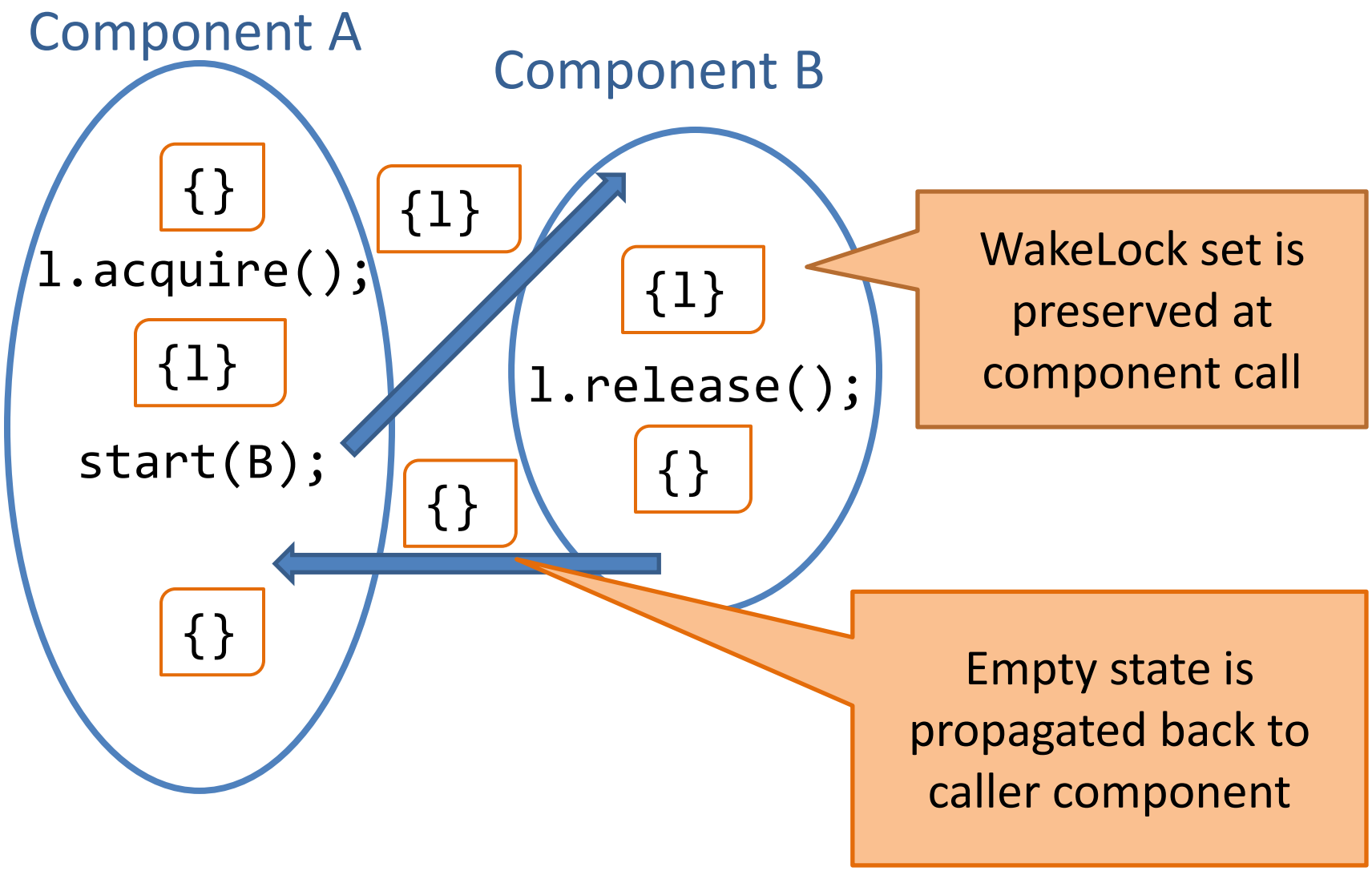
Java Threads

Posted on Message Queue (Android)

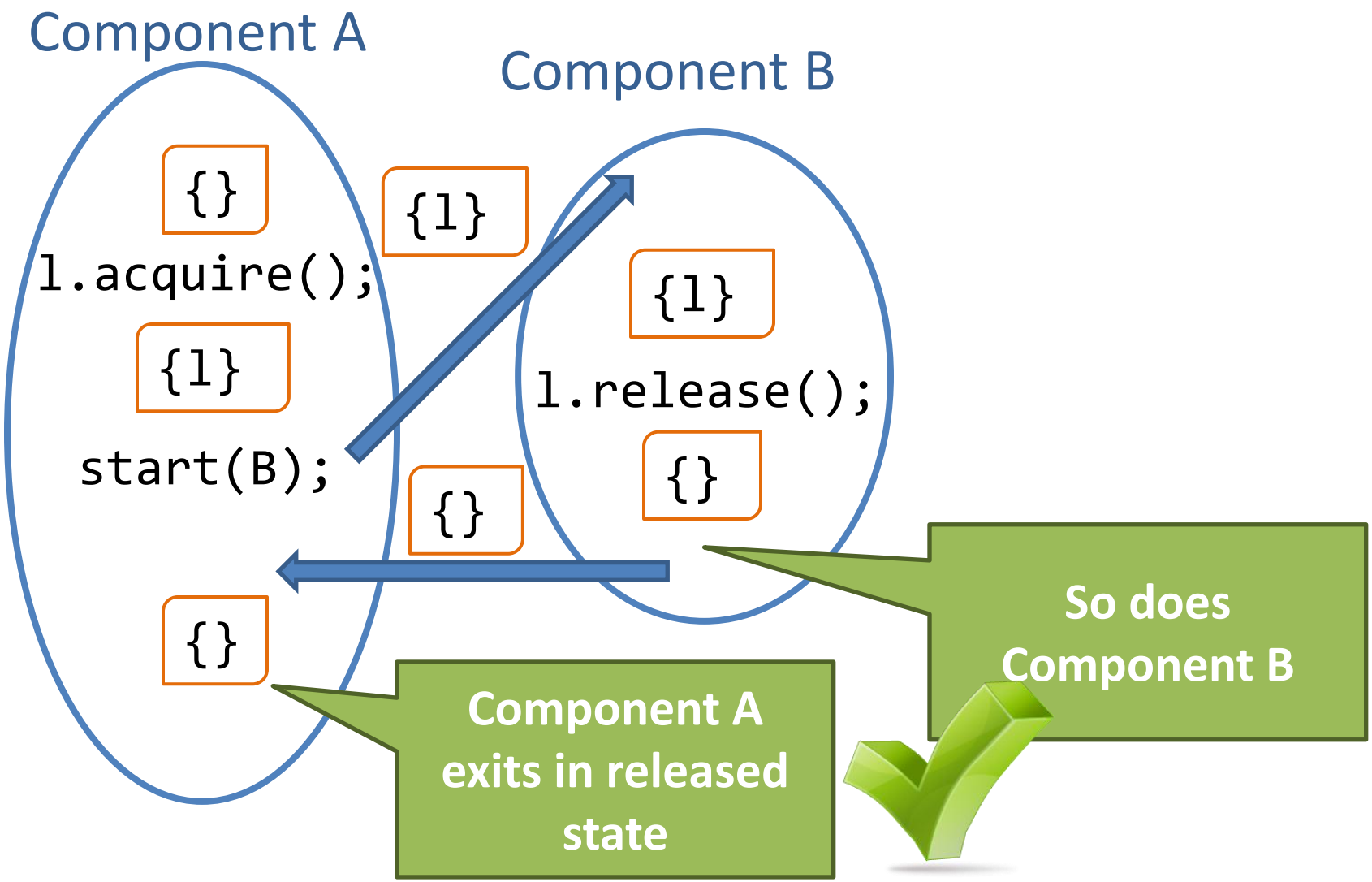
Handling Asynchrony



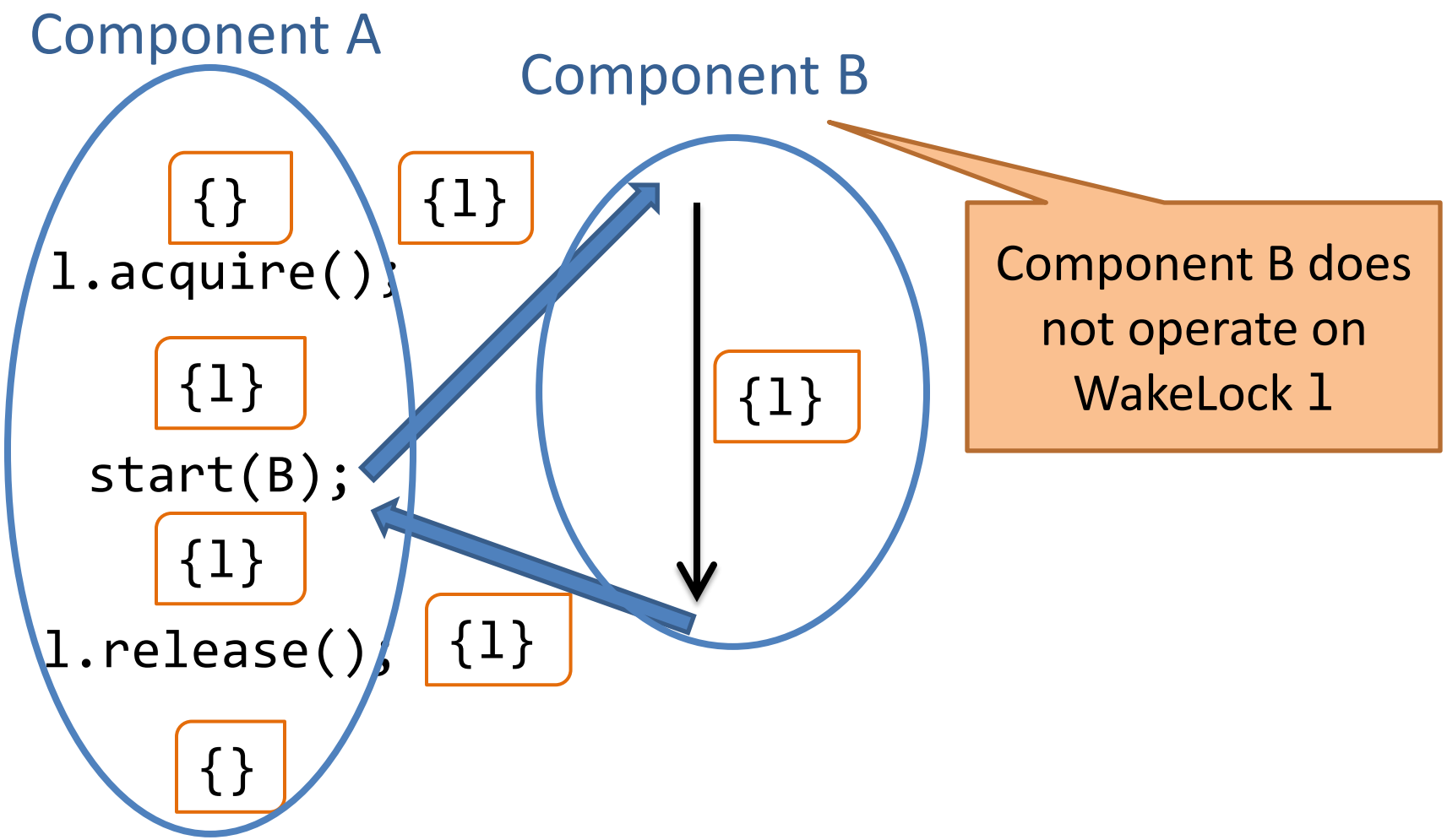
Handling Asynchrony



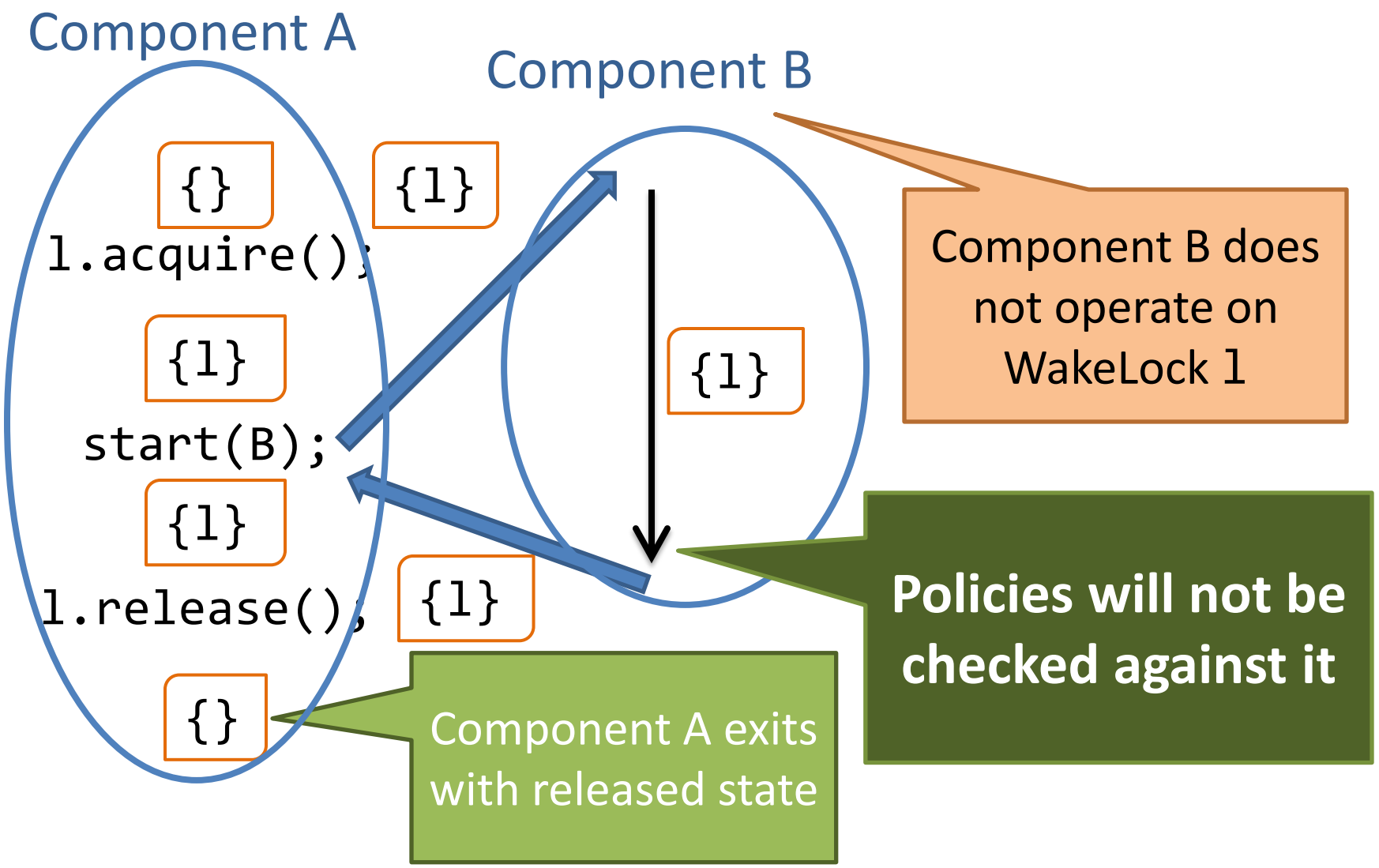
Handling Asynchrony



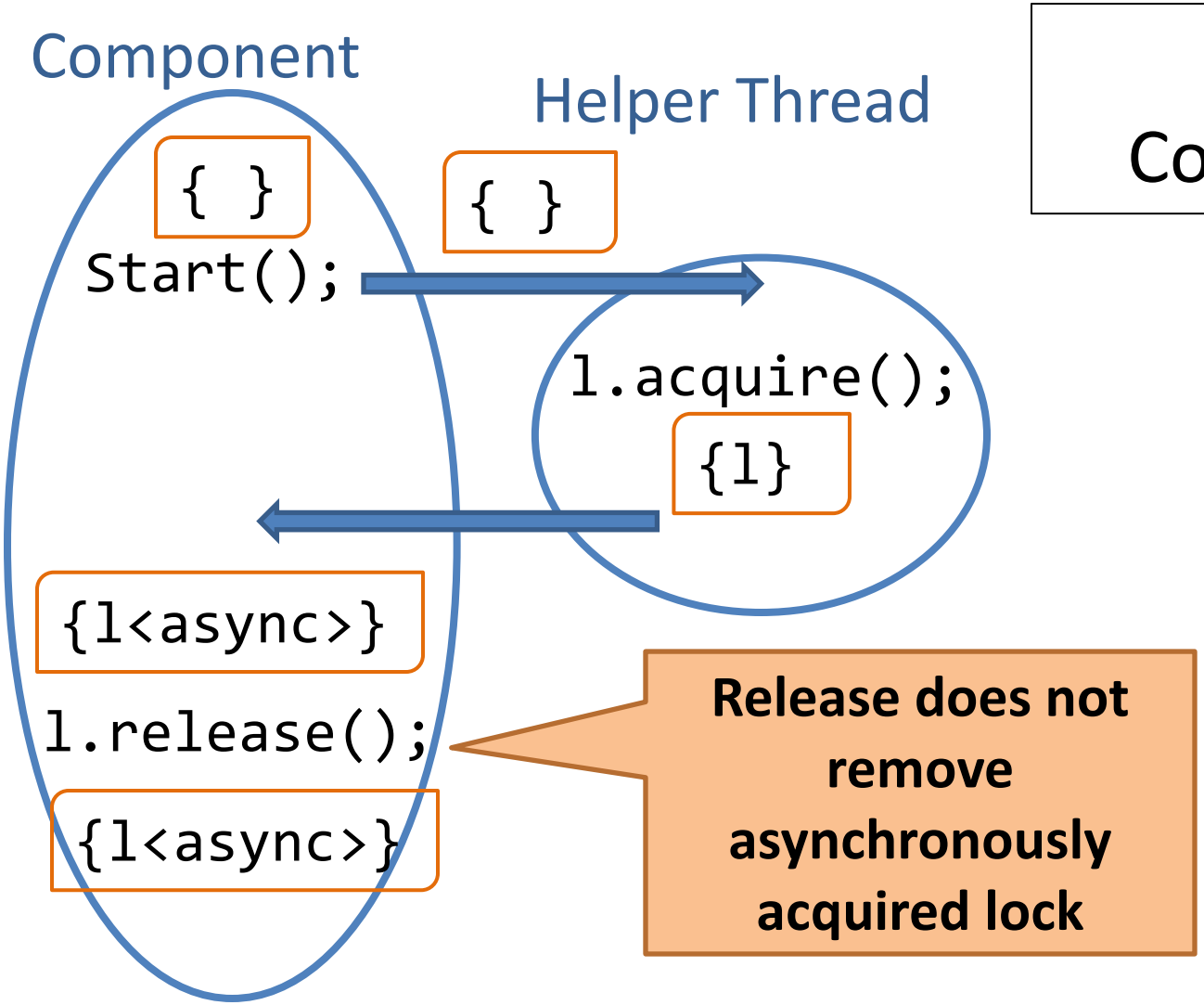
Handling Asynchrony



Handling Asynchrony

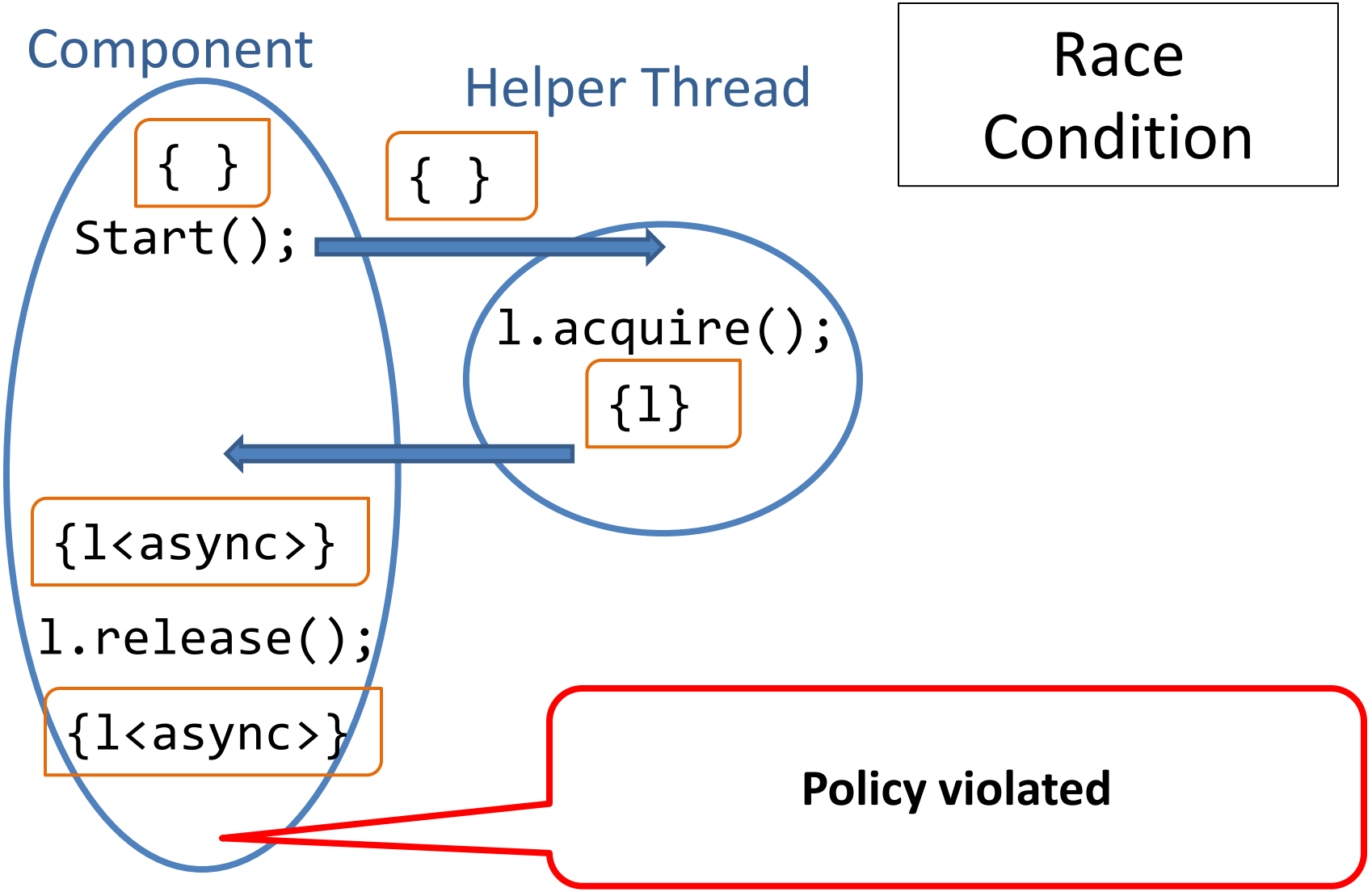


Handling Asynchrony

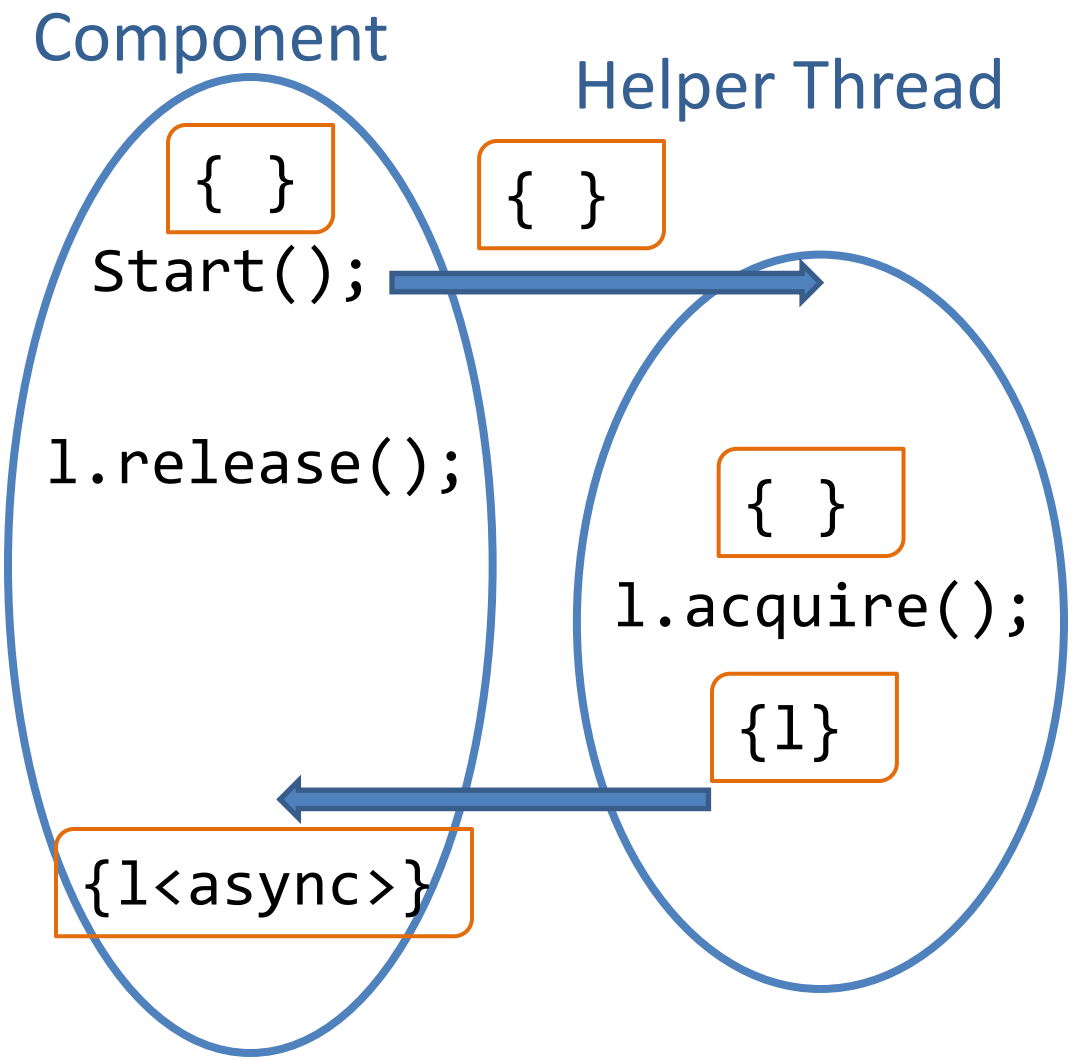


Race Condition

Handling Asynchrony

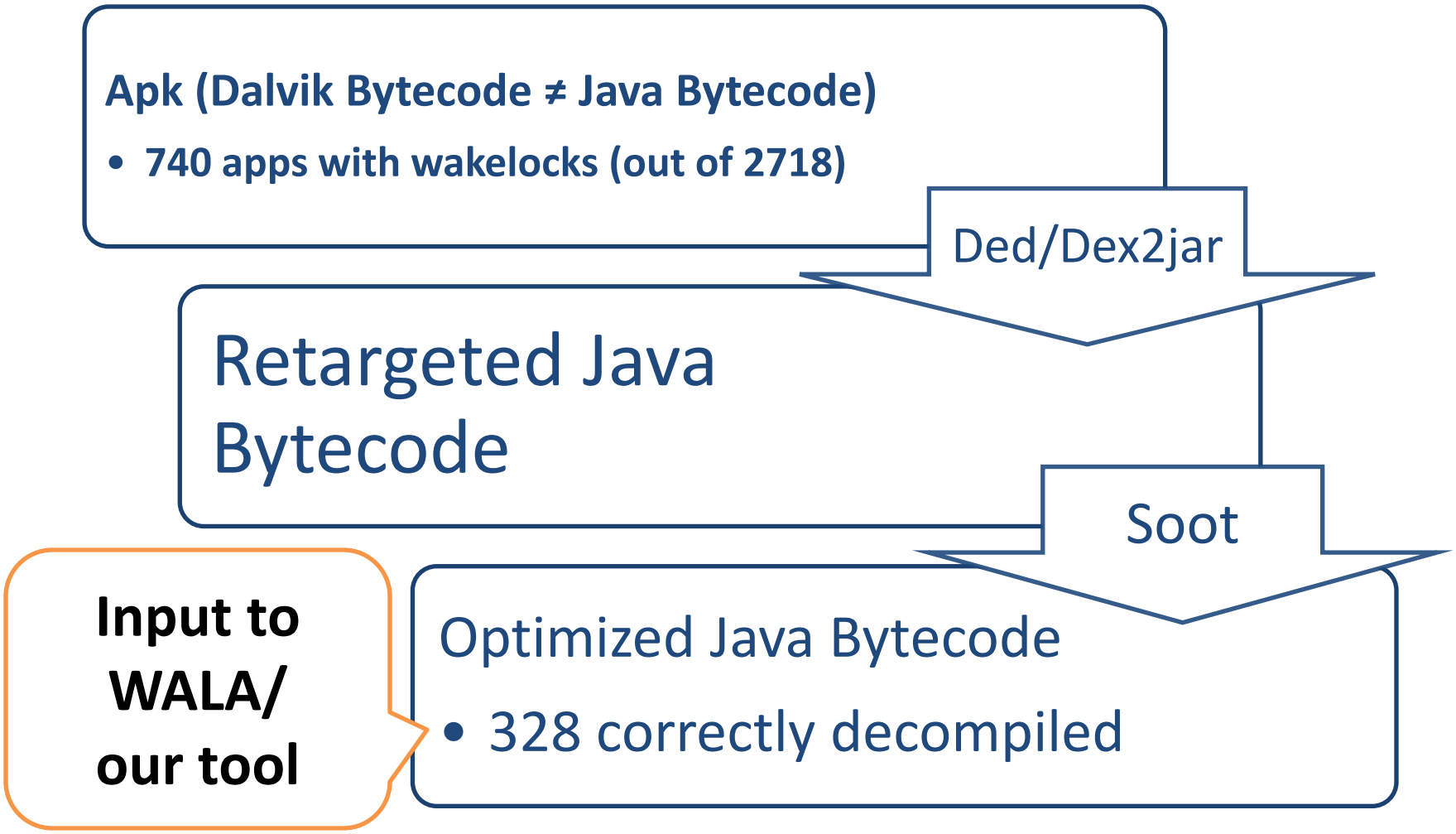


Handling Asynchrony

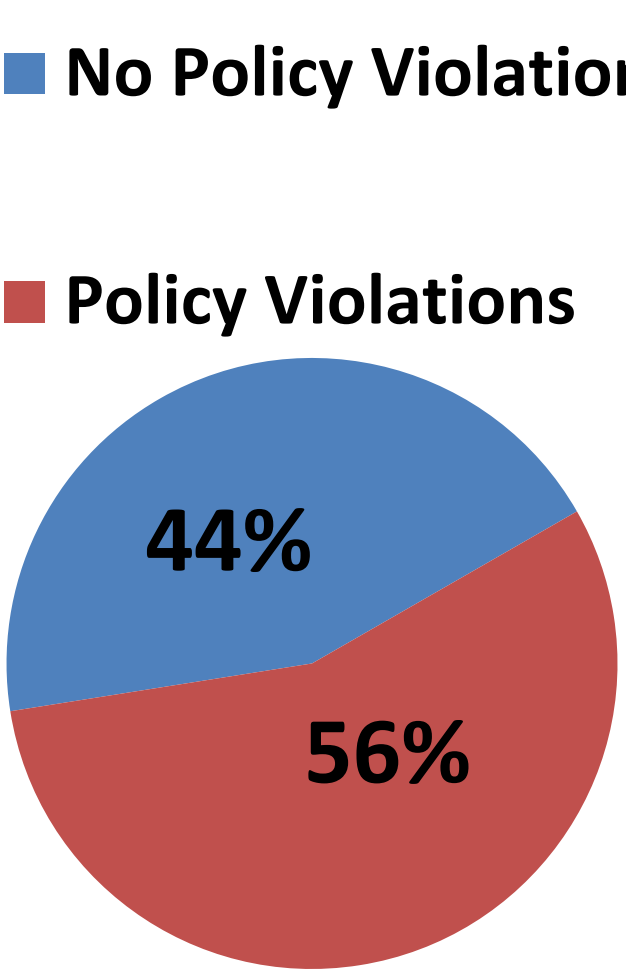


Race Condition

Tool Workflow



Analysis Results for 328 apps



Many bugs in real apps

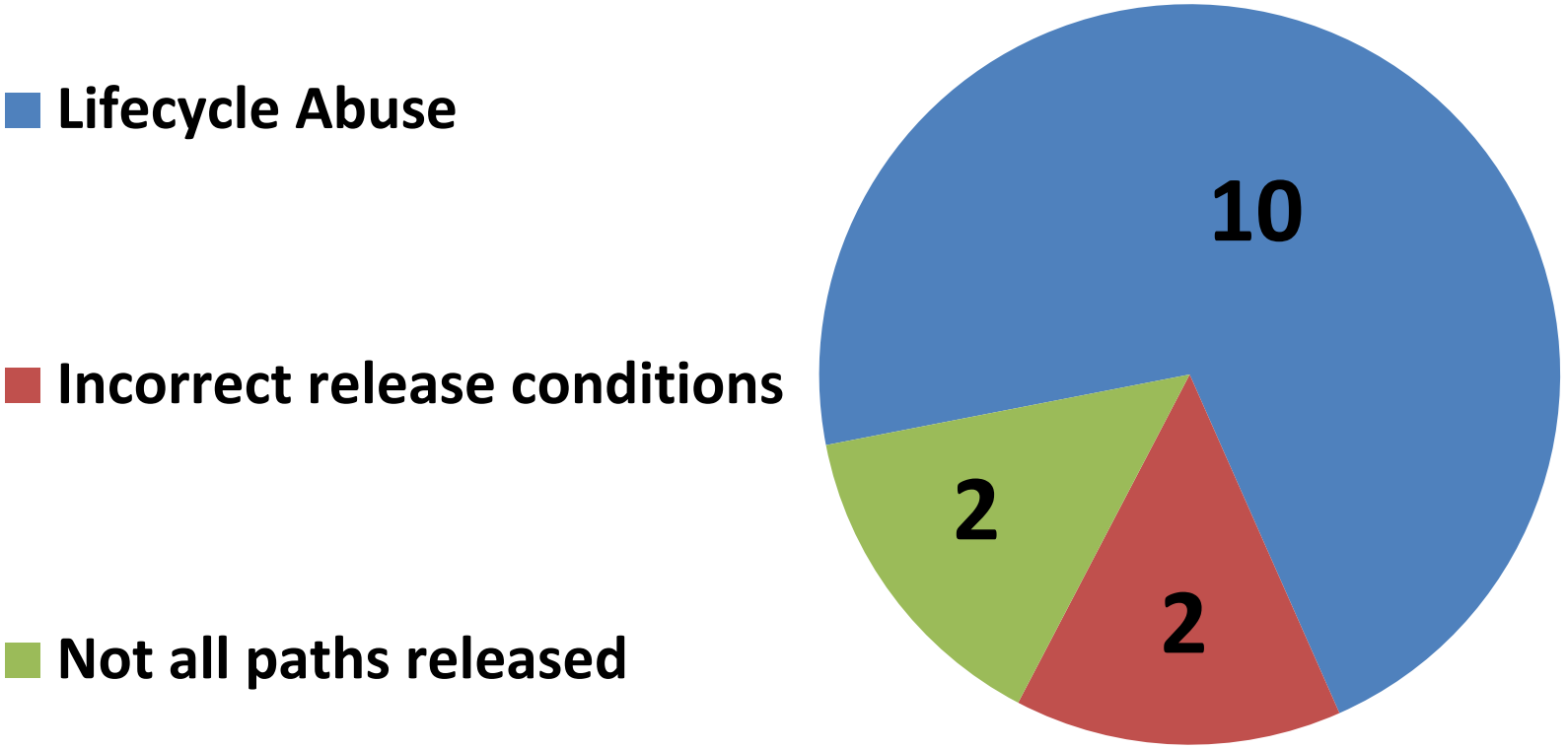
or

Too imprecise analysis

Manual inspection of 31 violating apps

True Positives

Apps confirmed to contain a bug: 14 of 31



False Positives

Correct apps flagged as buggy: 17 of 31

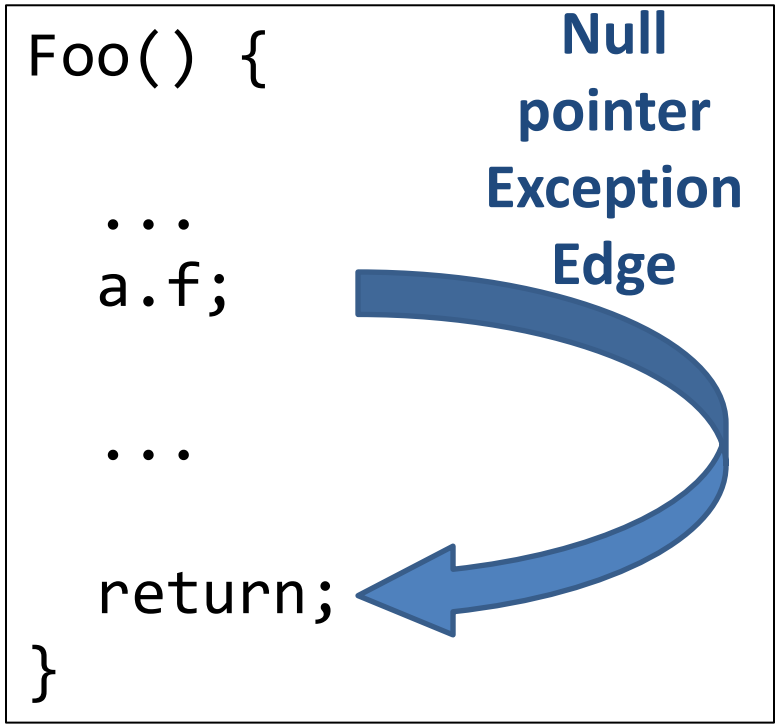
- **Complex release conditions**
- **Too strict policies**
- **Unresolved Intent calls**
- **Wrapper with parameter**
- **Unrecognized callback**



Soundness

WALA

Conservative at handling exception edges



Keep analysis precise:
Exception edges bypass release operations

Kill state over exception edges

Future work

Soundness

- *Exception edges*

Precision

- *Add path sensitivity*
- *Asynchronous calls -> precise points-to analysis*
- *Refine policies*

Retarget Dalvik effectively

Conclusion

No-sleep bugs are real

We presented a tool that:

- *Uses static analysis*
- *Guarantees for absence of no-sleep bugs*
- *Tracks asynchronous calls*