

android Bootcamp 2019

Windowing in Q

March 14, 2019



Team Goals for Q

- *Wale Ogunwale*

Team Goals for Q

- Improve codebase architecture
- Deliver high-impact features
- Improve tools and automated test coverage

Improving Codebase - Multi-Year Journey

O

- Window hierarchy
- Layer hierarchy - SurfaceView only
- Improve Keyguard handling
- Task snapshots
- Increased test coverage



P

- Window configuration
- SurfaceControl hierarchy
- Synchronize app transitions
- Activity Life Cyclor - client only
- WinScope tool
- Even more tests!



Q

- Unified window hierarchy
- SurfaceFlinger-based input
- Optimized VirtualDisplay
- Public SurfaceControl
- WM flags migration and Inset API rewrite - finish in R
- Even more tests!!



~_(\ツ)_/~

Unified Window Hierarchy

- *Wale Ogunwale*

Background

- Initial design of framework 10+ years ago had good separation of concerns for the use case then
 - **WindowManagerService (WM)**: Responsible for window policies
 - **ActivityManagerService (AM)**: Responsible for policies for core components (**activities**, services, broadcast, process management, ...)
- Over the years, the number of uses cases that activities need to support has grown significantly
 - Split-screen, Pip, freeform multi-window, multi-display, keyguard management, ...
- Changes (often duplicates) were usually made to both the WM and AM packages since the use cases affected both activities and windows
- The state of hierarchy objects in both packages needed to be kept in sync even though they were under different locks

Background

AM	Link Controllers	WM
ActivityDisplay	DisplayWindowController	DisplayContent
ActivityStack	StackWindowController	TaskStack
TaskRecord	TaskWindowController	Task
ActivityRecord	AppWindowController	AppWindowToken

Objects representing the same entity are present in both AM and WM packages and kept in sync with a third controller object.

Problems with the Model

- Obvious code duplication across the WM and AM packages
- Difficult to make atomic changes to the hierarchy since it is split between the two packages under different locks
- Difficult having a consistent snapshot of other packages in the system
- Lock contention in AM package since it has lots of other responsibilities
- Increased code complexity to manage the interaction between both packages

Problems with the Model

- Awesome bugs like these

Fix issue with where display is removed while creating it in AM and WM

It is possible for a display to be removed while we are in the ctor of ActivityDisplay in AM, but before we can get the Display object in the ctor of DisplayWindowController in WM. This causes us to throw an exception because the caller is trying to add a display we can't find in display manager. Unfortunately there isn't a good way to handle this race. To work around it we will now pass the Display object from AM to WM to use and depend on the fact that AM will remove the display shortly after.

Change-Id: Ie3f9d86bad67f5a023e3e7dfce5219b98c796864

Fixes: 72893961

Test: [go/wm-smoke](#)

Use correct windowingMode when computing override config

The StackWindowController was calculating task bounds with the WM-side windowingMode instead of the AM-side. In this case, we are resizing during a windowingMode change (fullscreen -> freeform). Since the windowingMode isn't sent to WM-side yet, the smallestScreenWidthDp was set using the old windowingMode resulting in the wrong resources being used.

This change makes windowingMode one of the parameters (like bounds/density) used to adjust the configuration.

Bug: [71028905](#)

Don't call into AM finish-recents-animation while holding onto the WM lock

- By default, only call back to AM to finish the animation from callers outside of the system, when we are not holding the WM lock

Bug: [78258614](#)

Solution

Have a single window hierarchy under the WM package and lock it by moving the activity management from the AM package to WM. This simplifies the interaction model and also solves all of the problems mentioned above.

Not so Easy...Tons of CLs and Code Cruft...

- 45+ CLs and counting
- Work still ongoing for 8+ months
- 90% done, but work will continue into the R release

Insets and Window Flag Migration

- *Jorim Jaggi*

Confusing Flags

`View.SYSTEM_UI_FLAG_FULLSCREEN`

→ Hides status bar, but doesn't exit split screen

`View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN`

→ Draws behind status bar, but doesn't hide it

`WindowManager.LayoutParams.FLAG_FULLSCREEN`

→ Like `SYSTEM_UI_FLAG_FULLSCREEN`, but different behavior regarding stable frames and immersive mode

Confusing Flags

`WindowManager.LayoutParams.FLAG_NOT_FULLSCREEN`

→ Became public API “by accident”

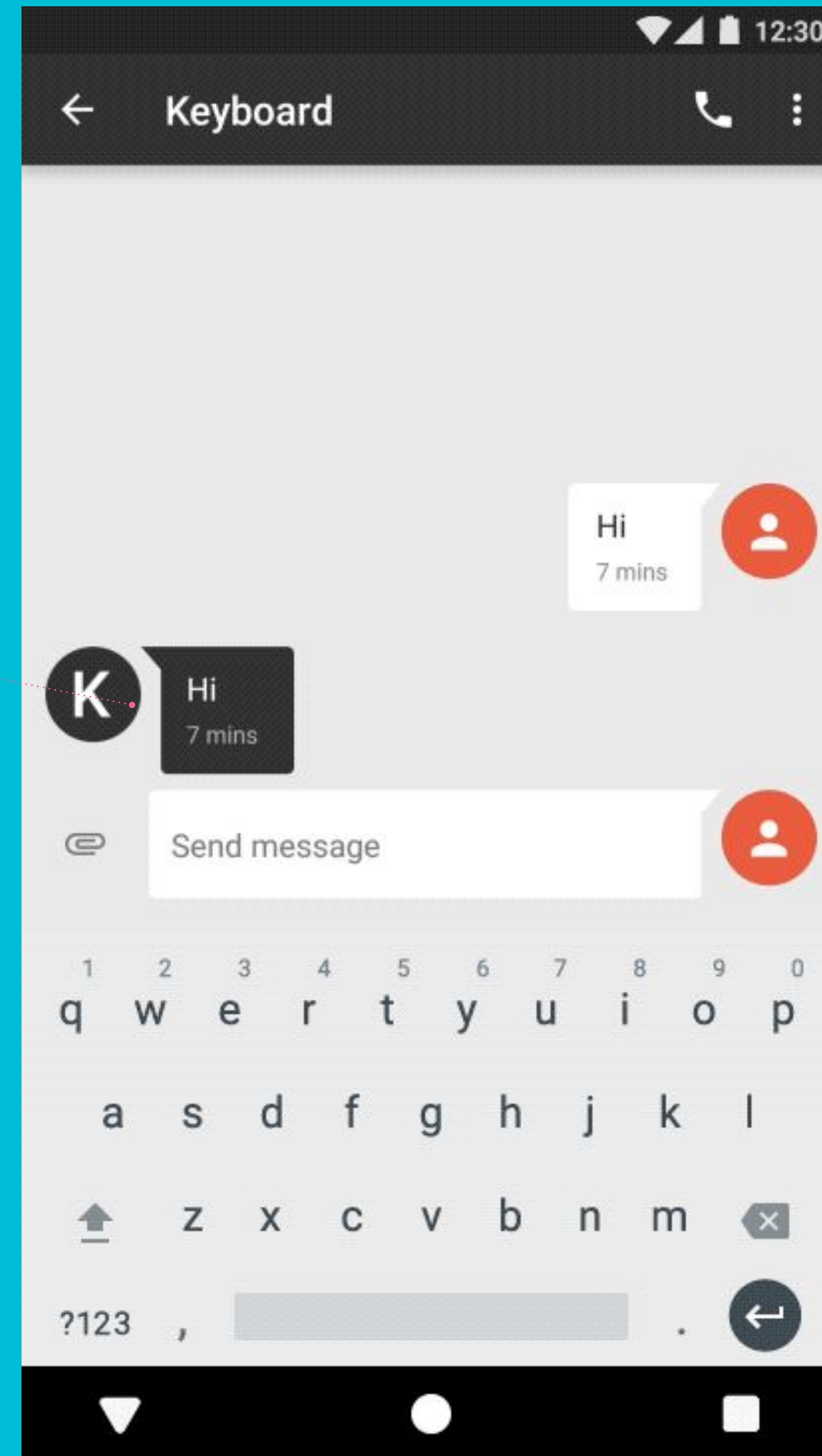
Is IME Visible?

Answer on [Stack Overflow](#)

```
getVTO().addOnGlobalLayoutListener(new OnGlobalLayoutListener() {  
    @Override public void onGlobalLayout() {  
        int heightDiff = getRootView().getHeight() - topView.getHeight();  
        if (heightDiff > dpToPx(this, 200)) {  
            // if more than 200 dp, it's probably a keyboard...  
        }  
    }  
});
```

Synchronized IME Transitions

- App content is fully synchronized with IME appear animation
- IME can be hidden/shown by scrolling up/down



Google Proprietary + Confidential

API Surface

Insets by type

`getInsets(@Type int type)`

`getMaxInsets(@Type int type)`

Animation listener & control

`setWindowInsetsAnimationListener`

`controlWindowInsetsAnimation`

Deprecate System UI flags

1. Window always extends below status and navigation bar
2. Use `Window.setFitsWindowInsets` to avoid placing content in these areas

Insets by Type

```
@Type int Type.systemBars();  
@Type int Type.topBar();  
@Type int Type.ime();  
@Type int Type.sideBars();  
@Type int Type.windowDecor();  
@Type int Type.all();
```

Insets by Type

```
Insets WindowInsets.getInsets(@Type int type);
```

→ Retrieves the insets of a given type

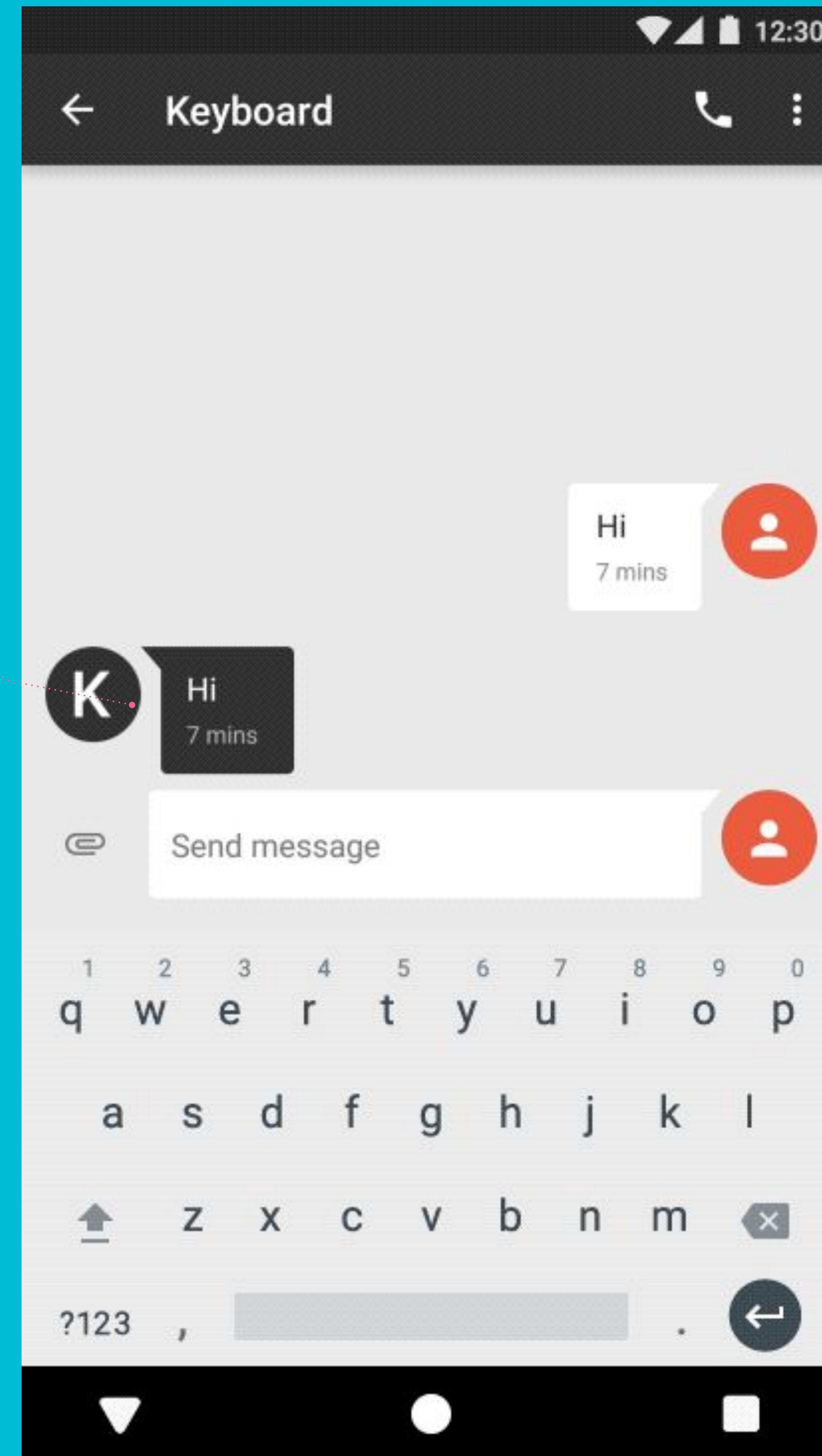
```
boolean WindowInsets.isVisible(@Type int type);
```

```
if insets.getInsets(ime()) not empty:  
    // IME is showing and occluding my content
```

```
if insets.isVisible(ime()):  
    // IME is showing, regardless of whether it's currently occluding my  
    // content
```

Synchronized IME Transitions

- **App content is fully synchronized with IME appear animation**
- IME can be hidden/shown by scrolling up/down



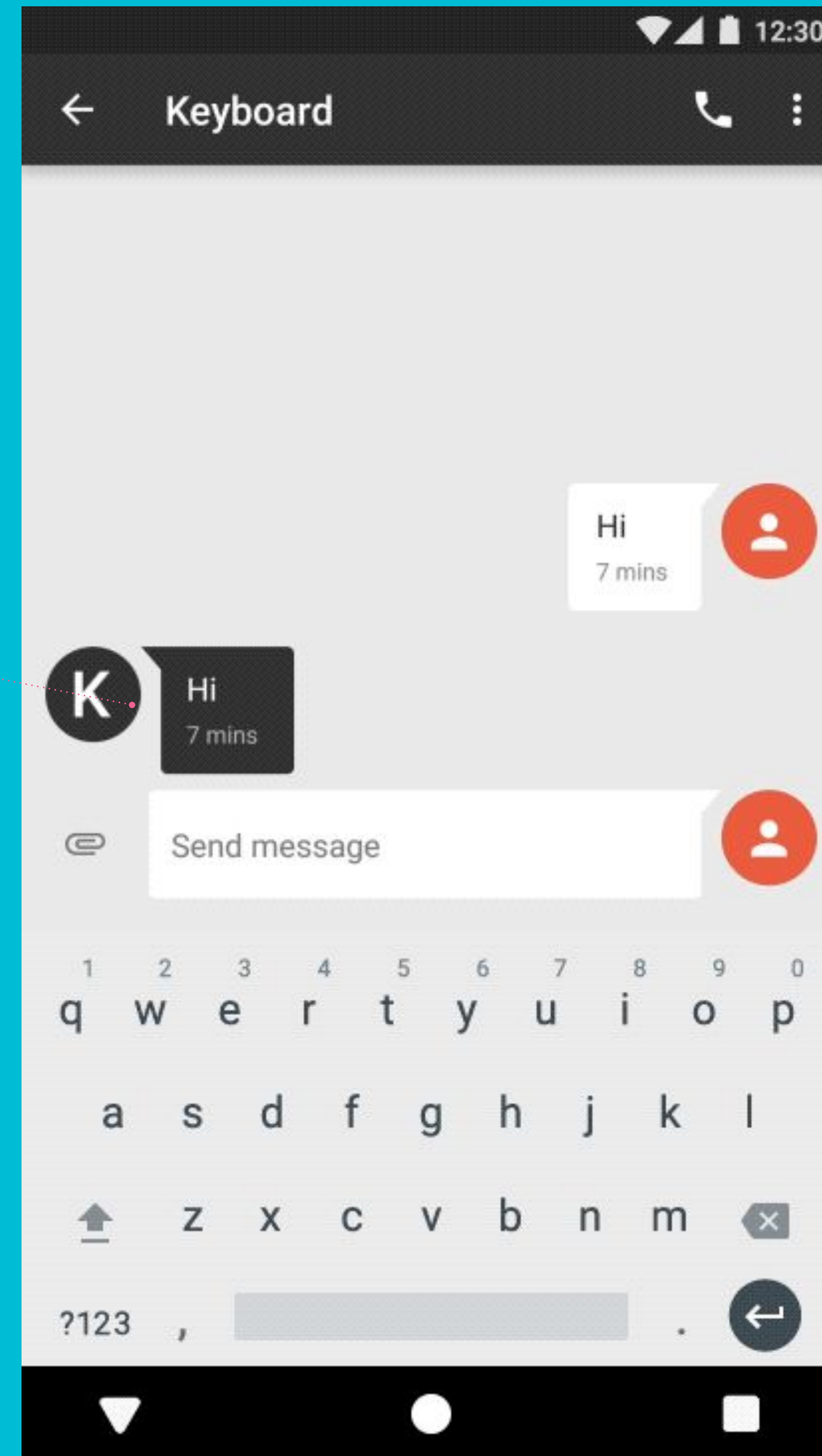
Animation Listeners

```
WindowInsetsAnimationCallback {  
    AnimationBounds onStart(InsetsAnimation anim, AnimationBounds bounds);  
    WindowInsets onProgress(WindowInsets insets);  
    void onFinish(InsetsAnimation anim);  
}
```

```
View.setWindowInsetsAnimationCallback(Callback callback);
```

Synchronized IME Transitions

- App content is fully synchronized with IME appear animation
- **IME can be hidden/shown by scrolling up/down**



Animation Control

```
View.getWindowInsetsController().controlWindowInsetsAnimation(@Type int types,  
    Listener listener);
```

```
Listener {  
    onReady(AnimationController controller);  
    onCancelled();  
}
```

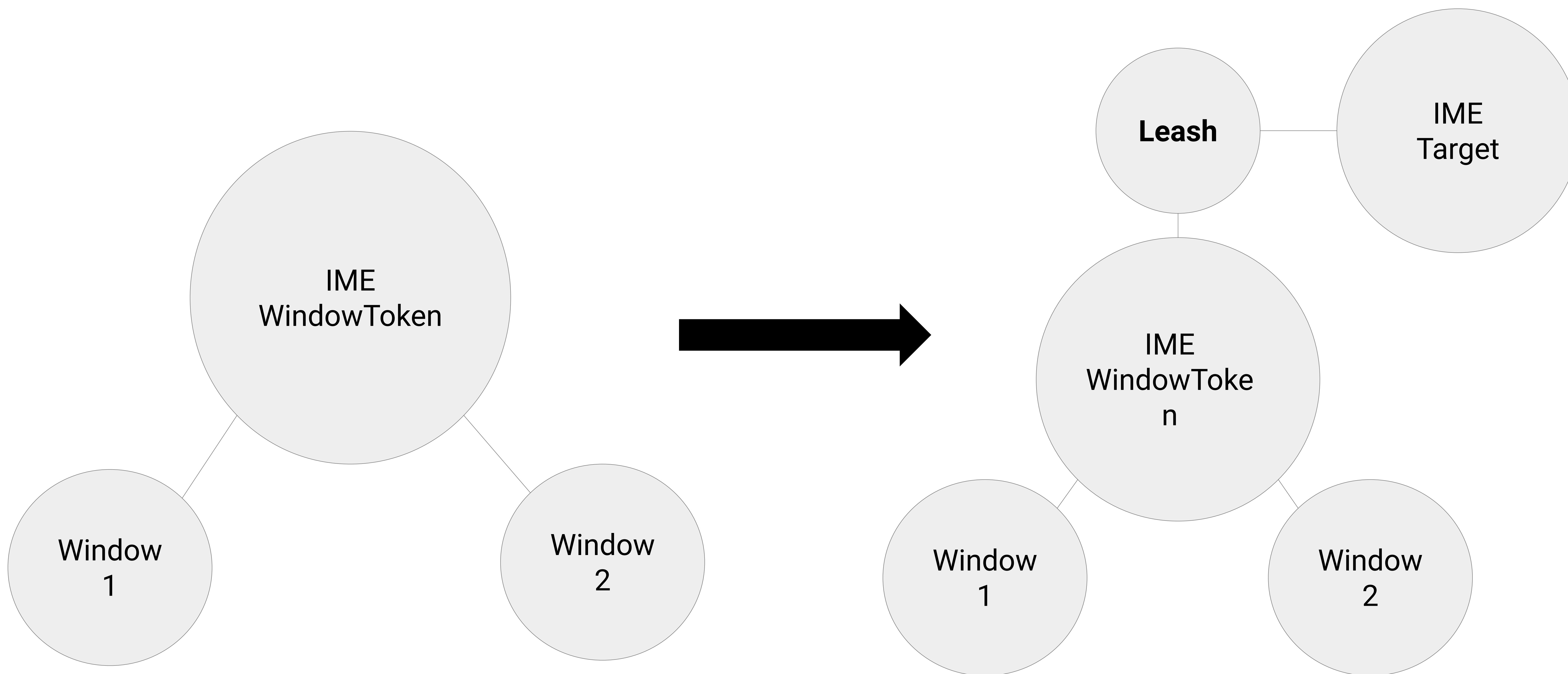
Animation Control

```
AnimationController {  
    Insets getShownStateInsets();  
    Insets getHiddenStateInsets();  
    void changeInsets(Insets insets);  
    void finish();  
}
```


Animation Control

```
// When scroll starts to reveal ime:  
v.getWController().controlWAnimation(ime(), new Listener {  
    onReady(AnimationController controller) {  
        // Depending on current scroll position, calculate how far IME  
        // extends into app content  
        Insets insets = calculateImeInsets(...);  
        controller.changeInsets(insets);  
    }  
});
```

IME Animations: Leash



Deprecating System UI Flags

~~SYSTEM_UI_FLAG_FULLSCREEN~~

~~SYSTEM_UI_FLAG_HIDE_NAVIGATION~~

→ `WindowInsetsController.hide(topBar() / sideBars())`;

~~SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN~~

~~SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION~~

→ `Window.setFitsSystemWindows(~topBar() / ~sideBars())`

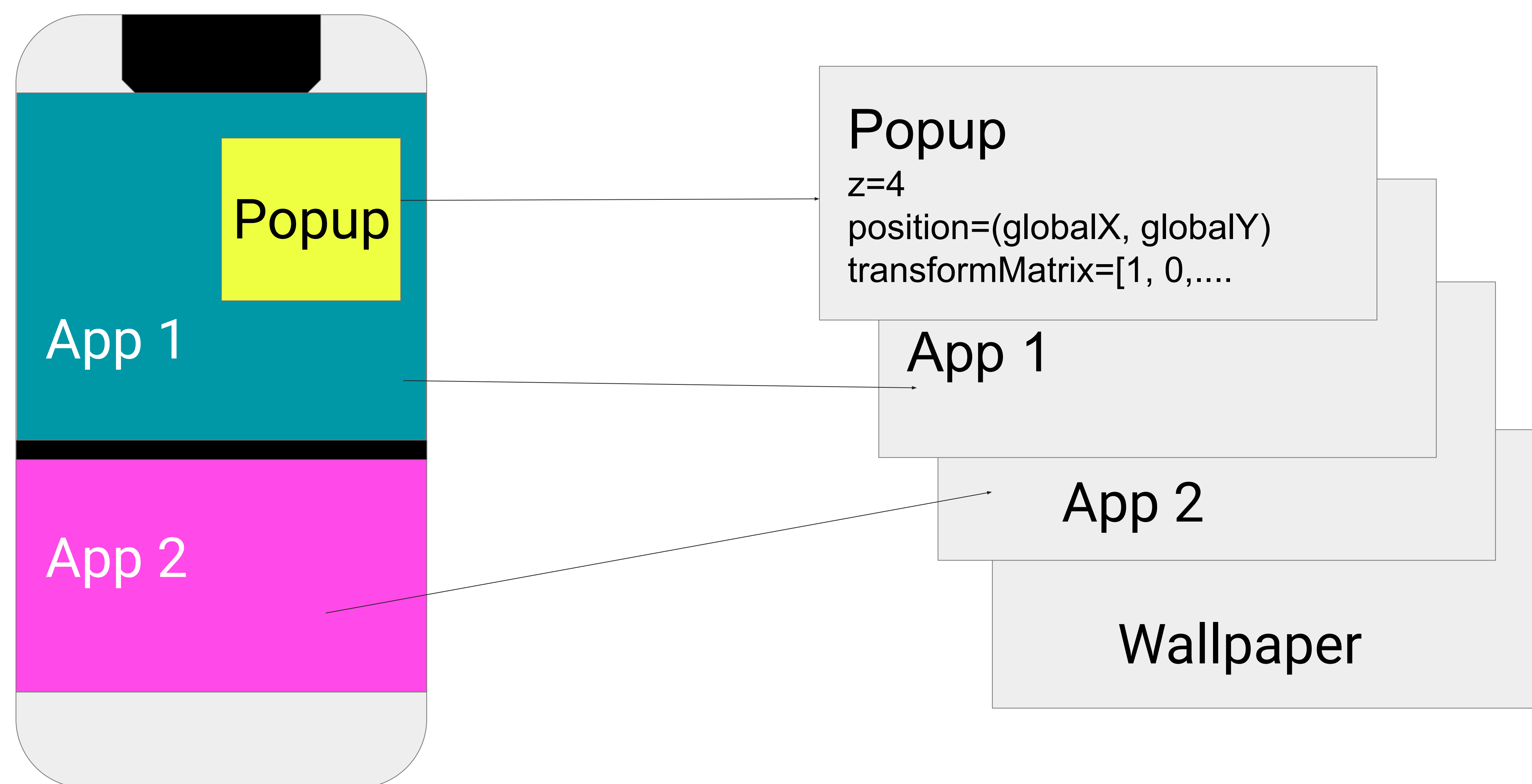
Similar approach for other flags

SurfaceFlinger Hierarchy - Part III?

- *Rob Carr*

SurfaceFlinger Hierarchy

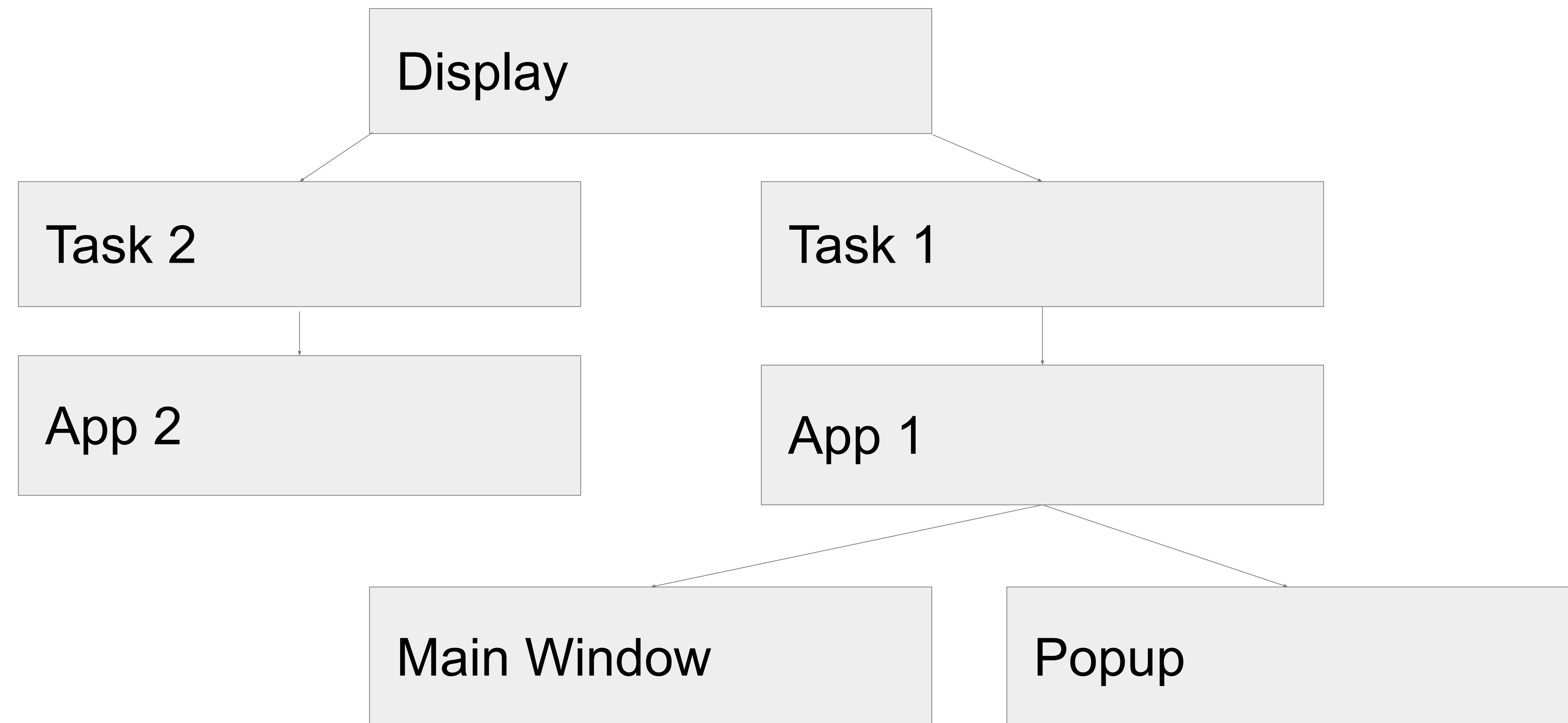
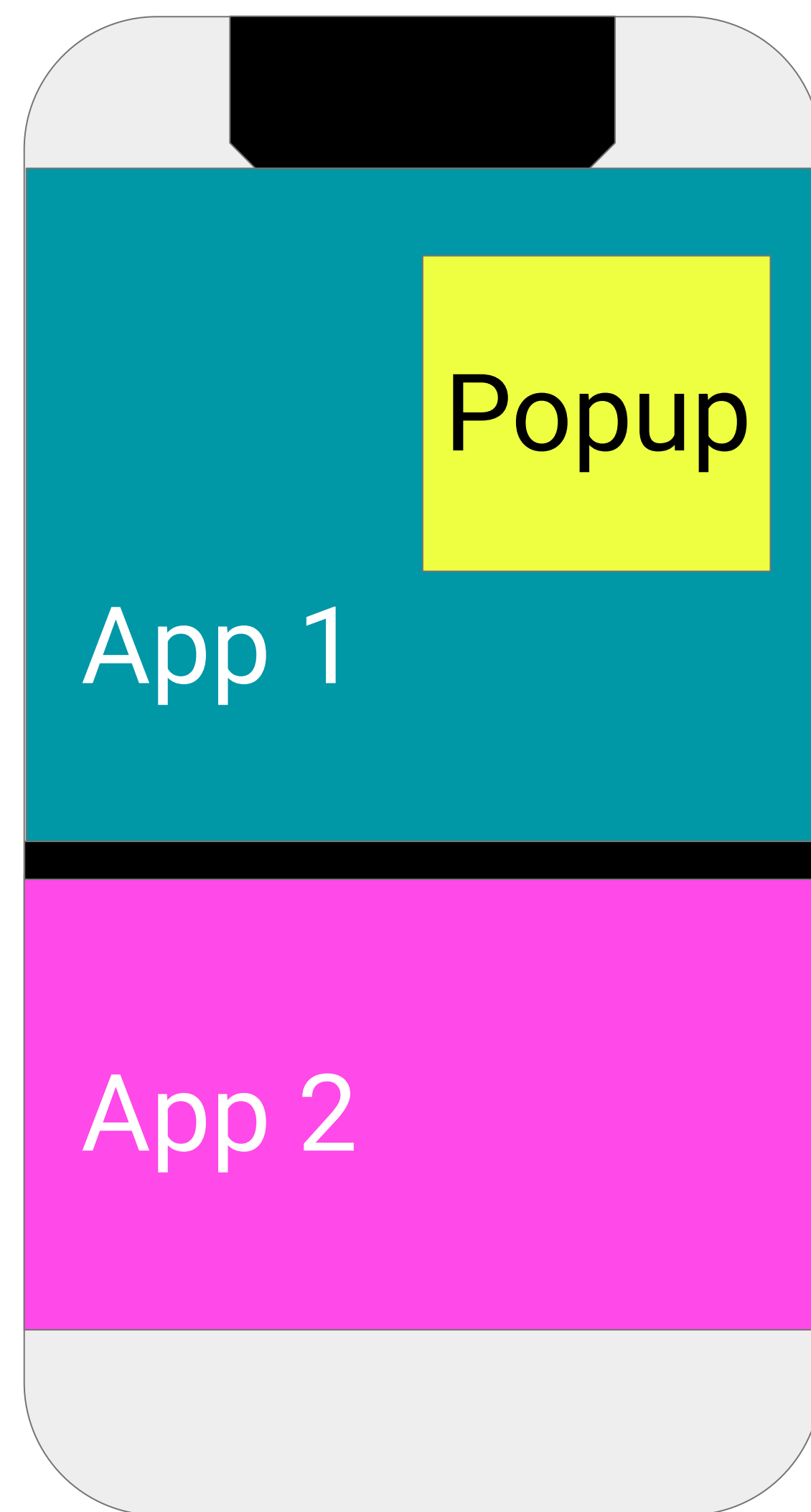
Old Model (SurfaceControl 'Retro')



SurfaceFlinger Hierarchy

New Model (SurfaceControl '*Nuevo*')

Google Proprietary + Confidential



Android O: Implementation, SurfaceView
Android P: WM hierarchy port, app controlled transitions, lock-free animations

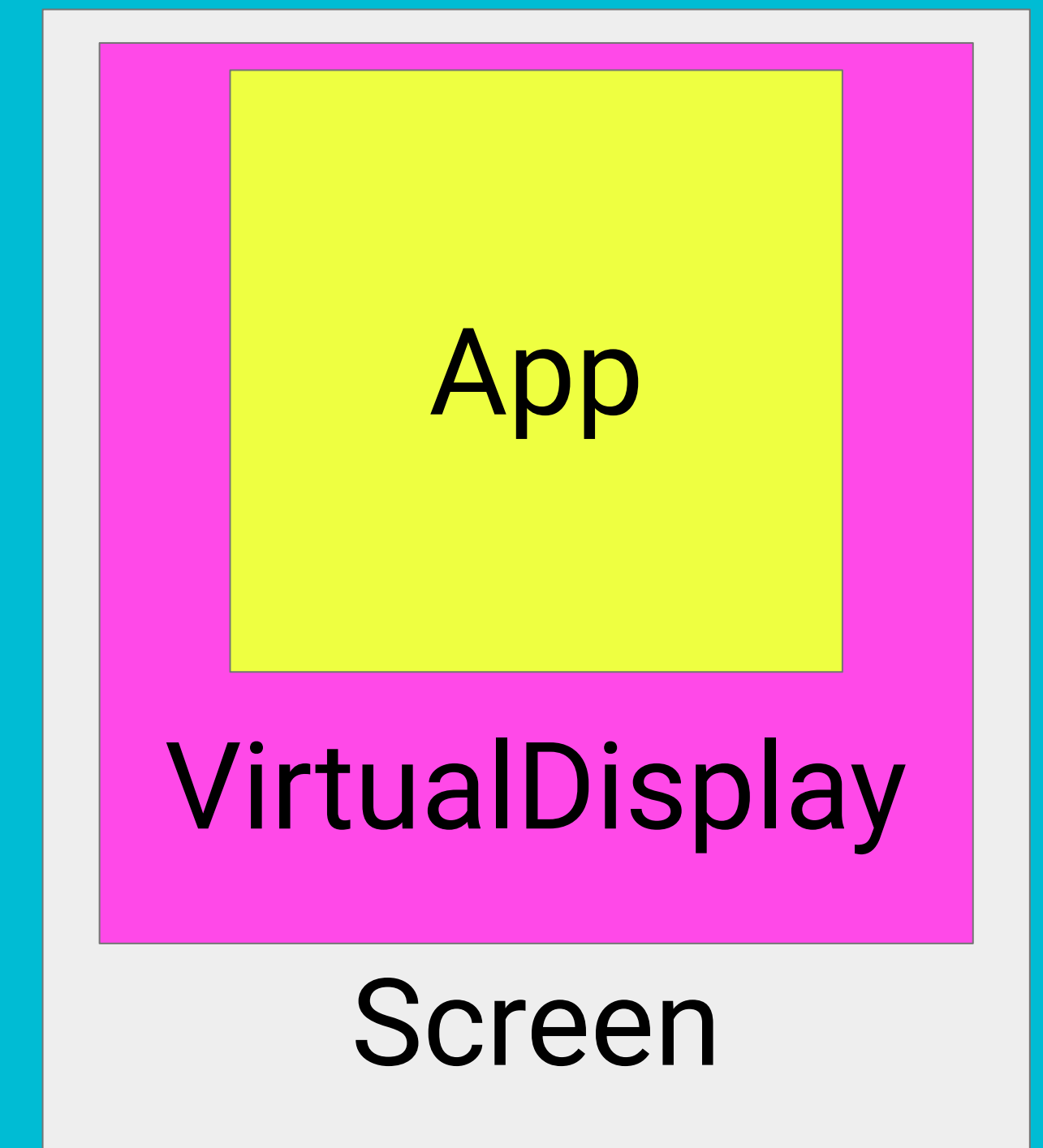
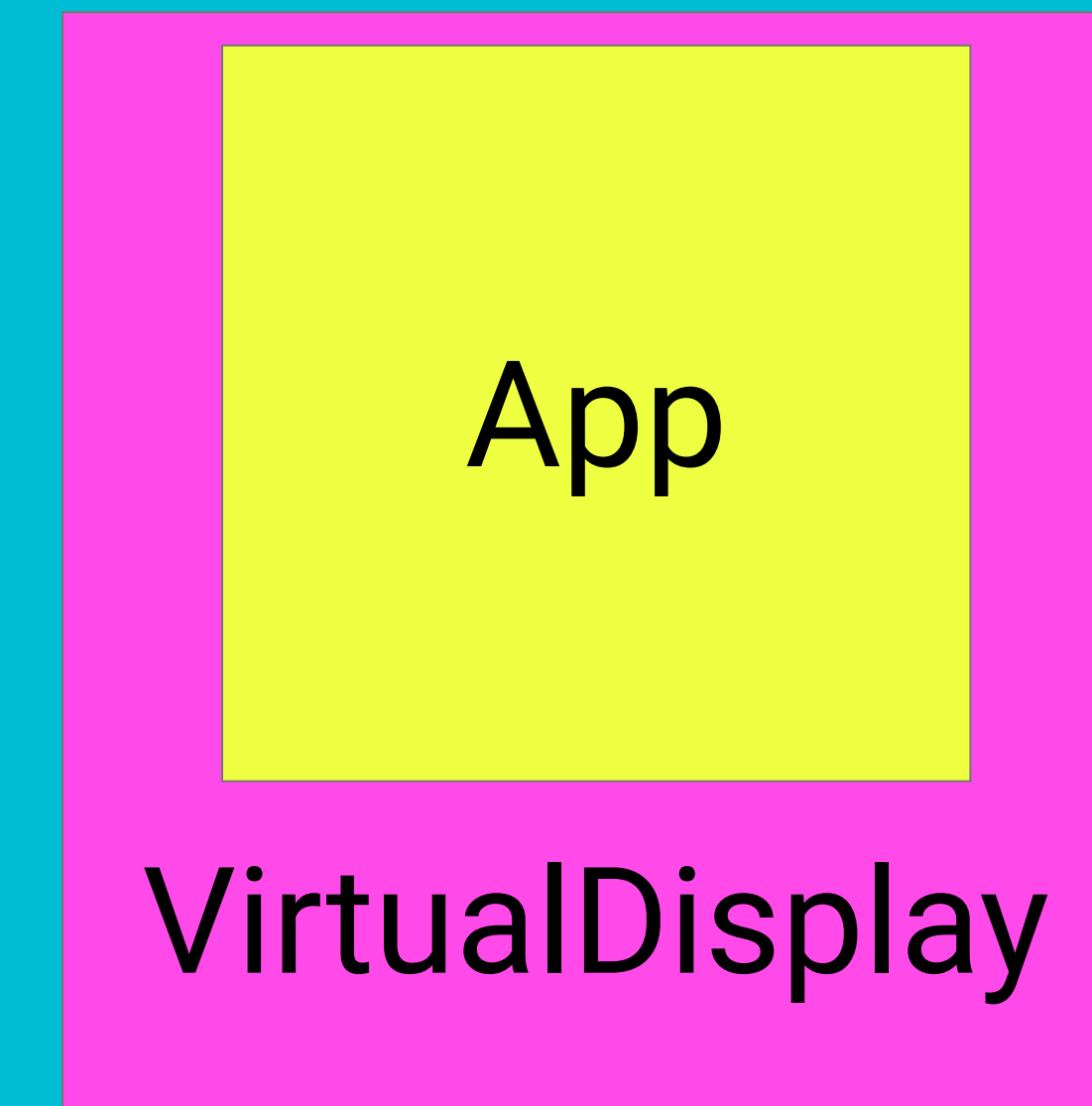
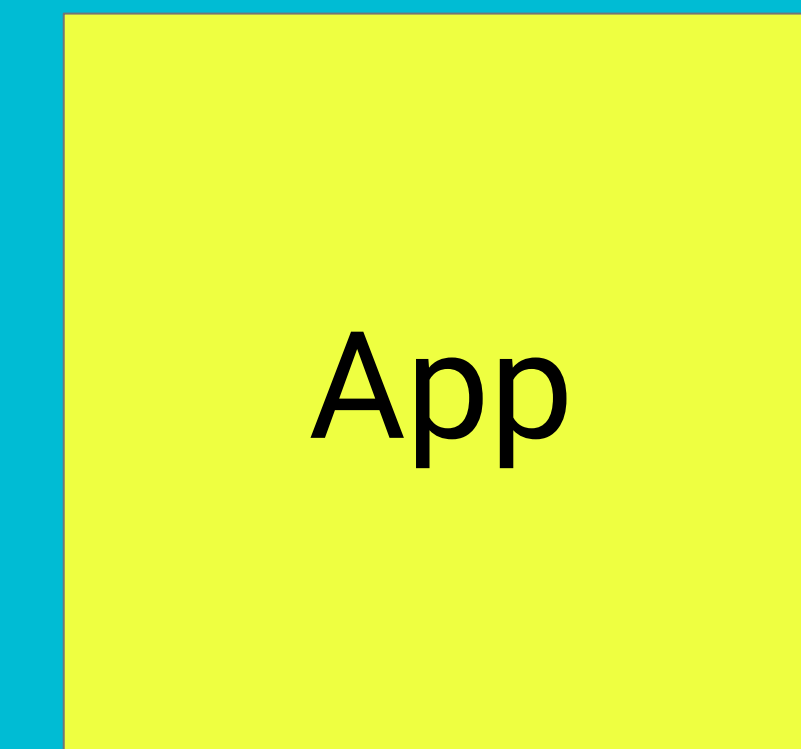
Android Q: Virtual Display 2.0, SurfaceFlinger input, public SurfaceControl, BLAST, inset animations

VirtualDisplay and ActivityView 2.0

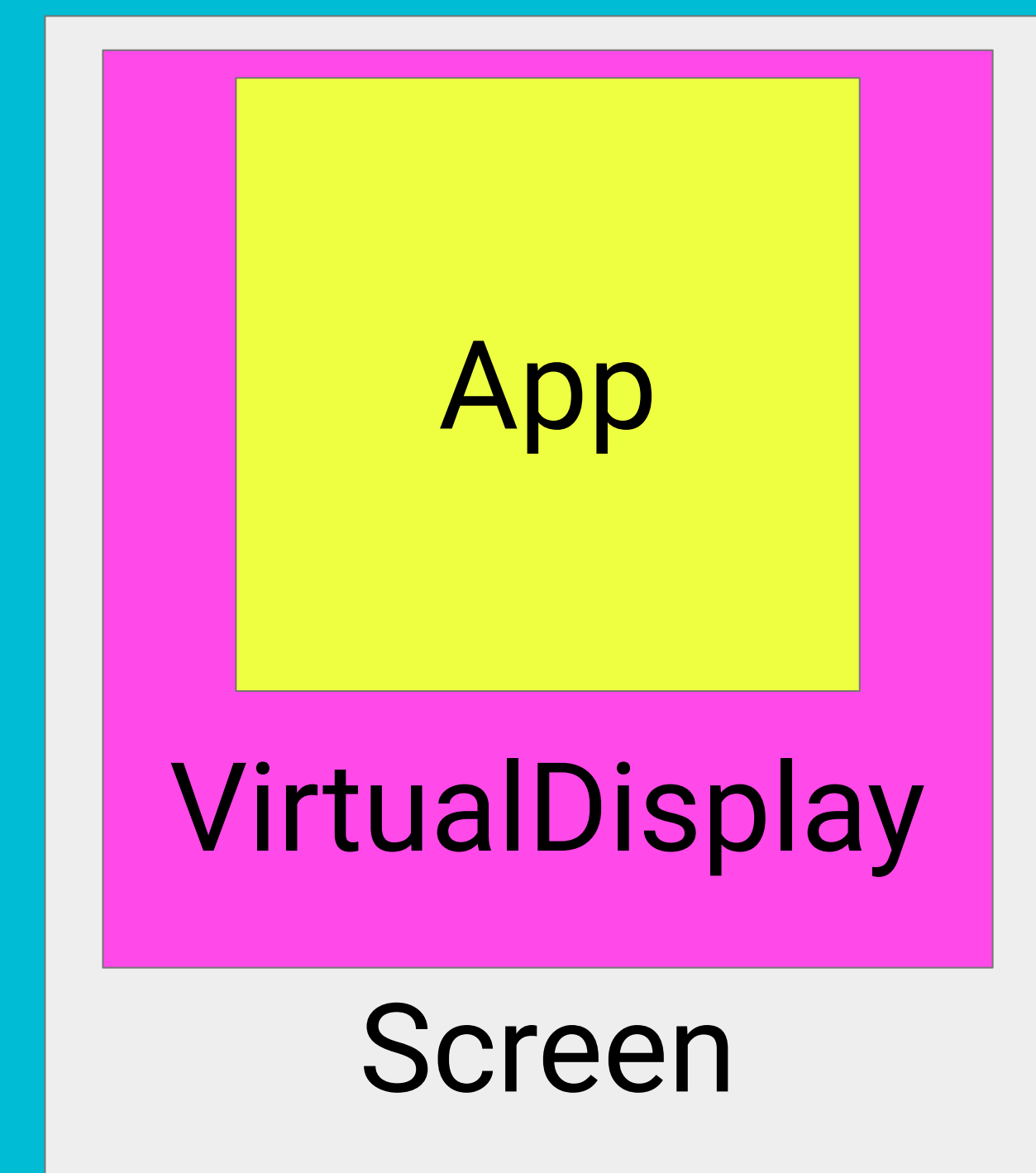
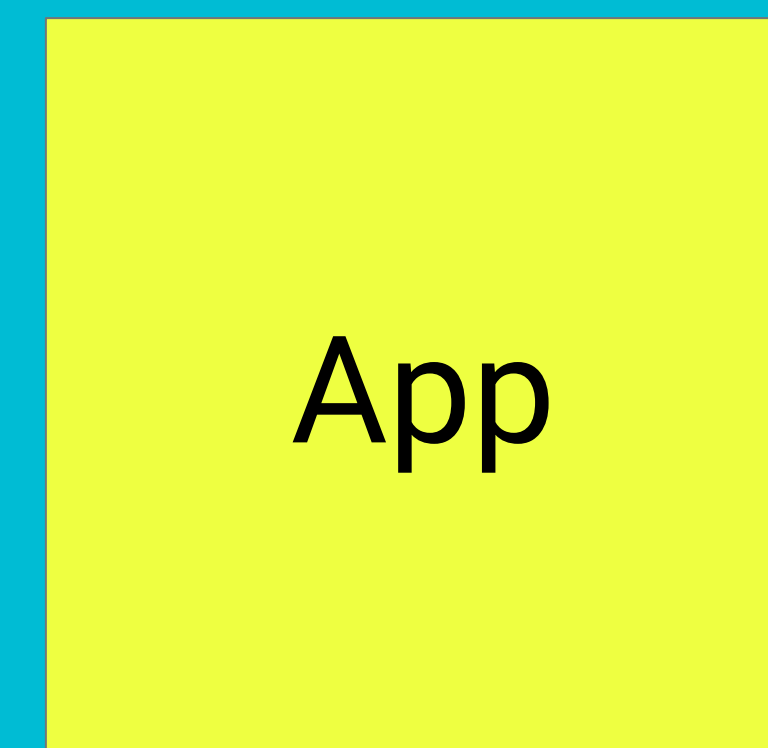
*Zero-overhead embedding of activities within
view hierarchy.*

Based on new implementation of
VirtualDisplay which returns a SurfaceControl
leash rather than rendering to a surface.

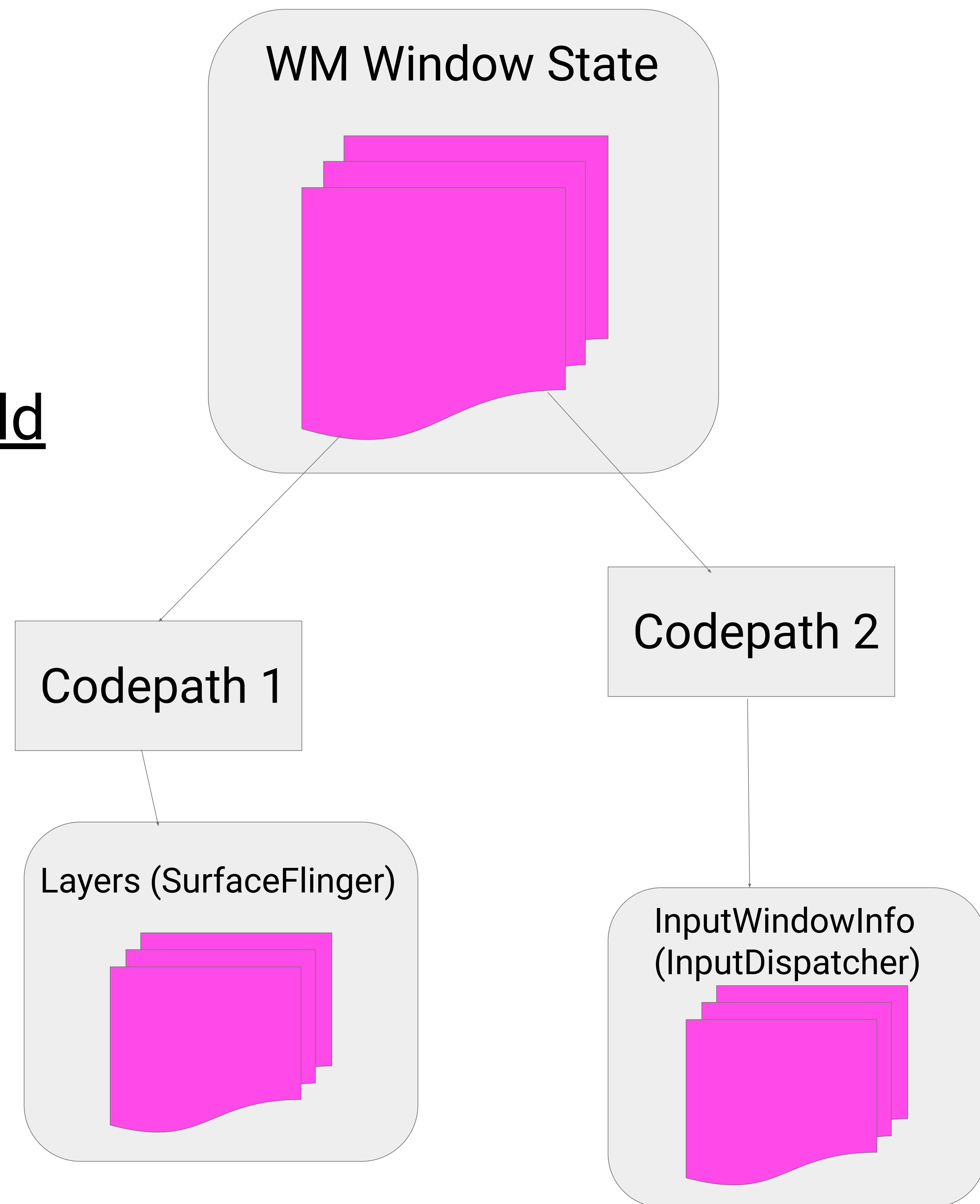
Old



New

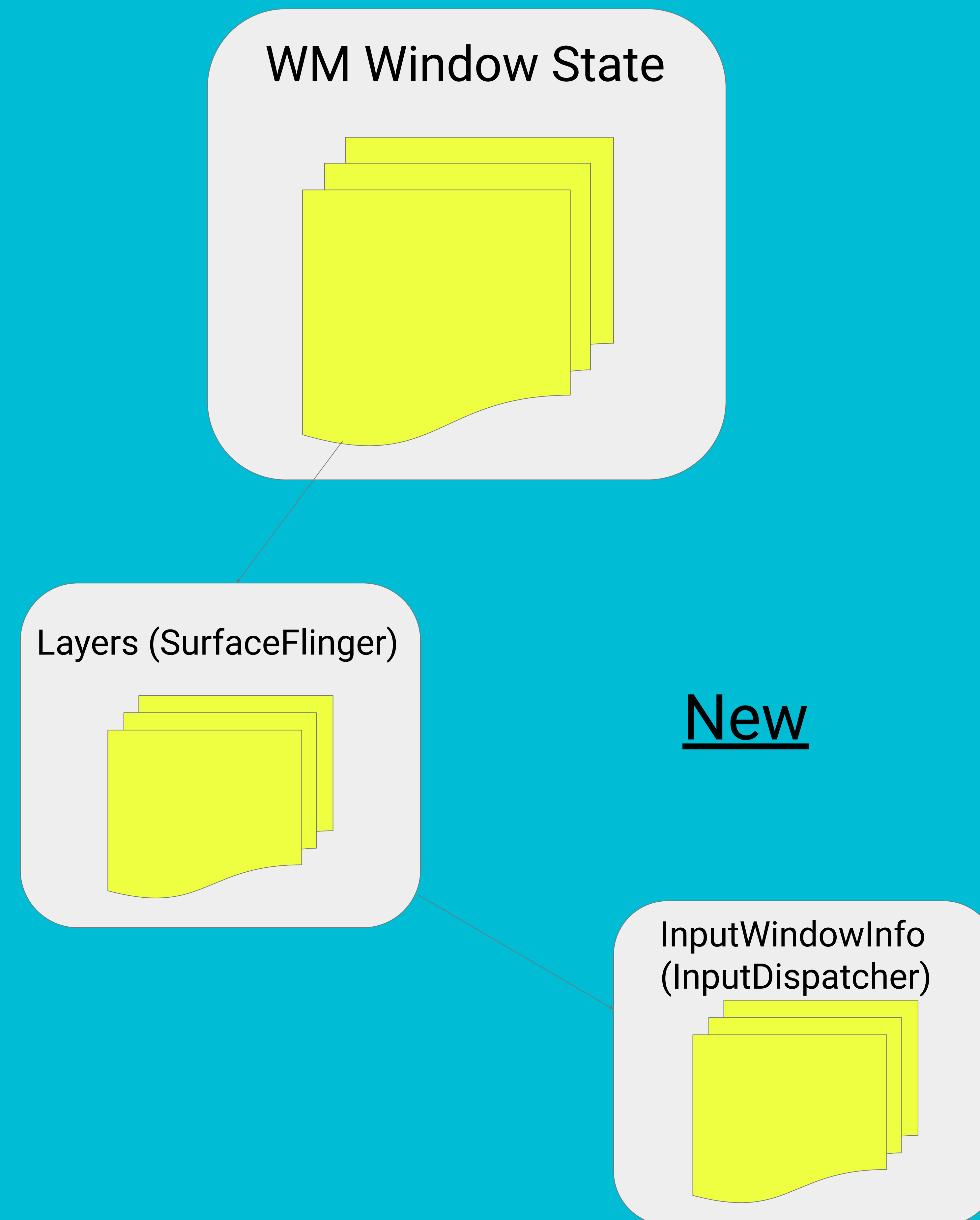


Old



SurfaceFlinger Input

Google Proprietary + Confidential



New

Public SurfaceControl API

- Surface control available in SDK and NDK
- Interesting for apps that do a lot of compositing, or have custom lifecycle requirements
- New Buffering API *BLAST* in NDK - lower level than BufferQueue

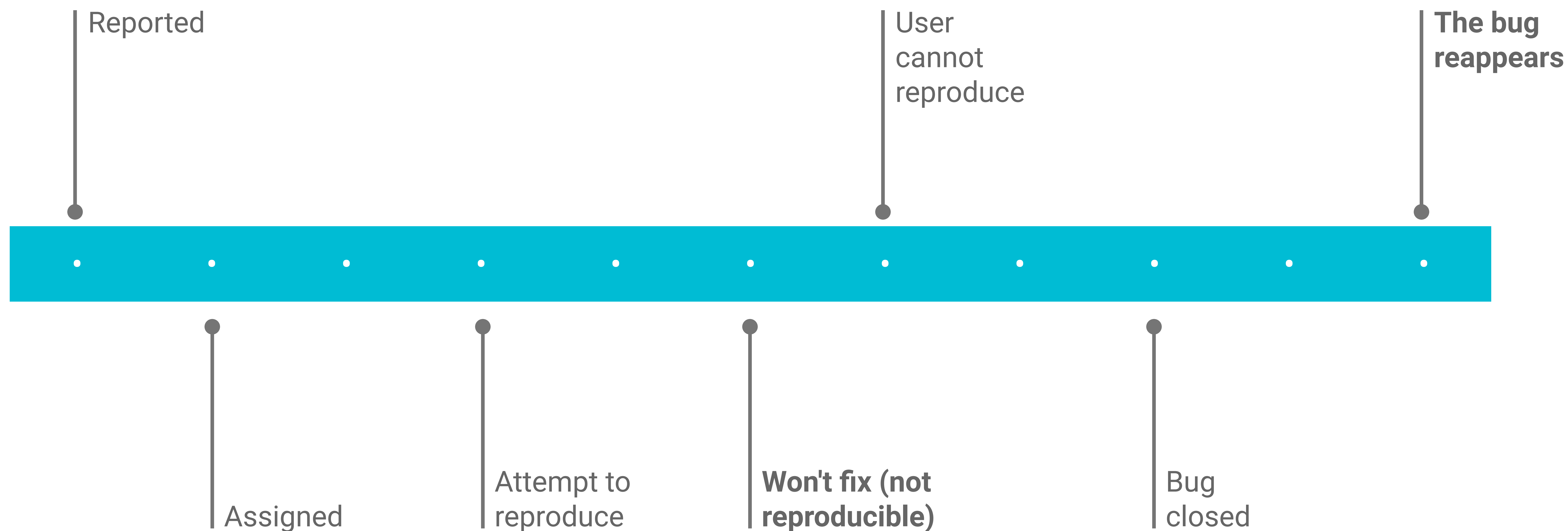
Improving Debugging

- *Nataniel Borges*

Continuous Logging

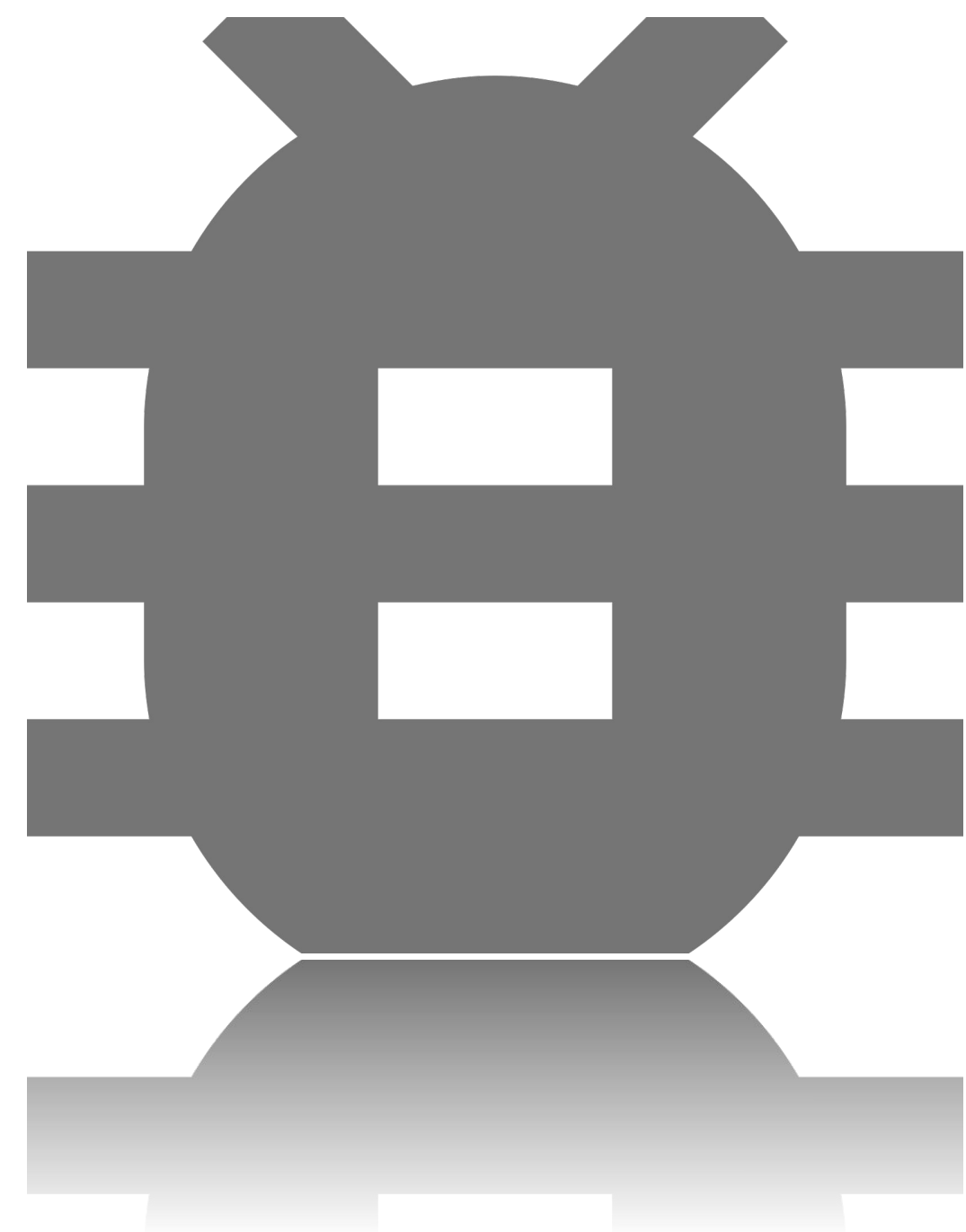
- *Nataniel Borges*

Life of a Bug

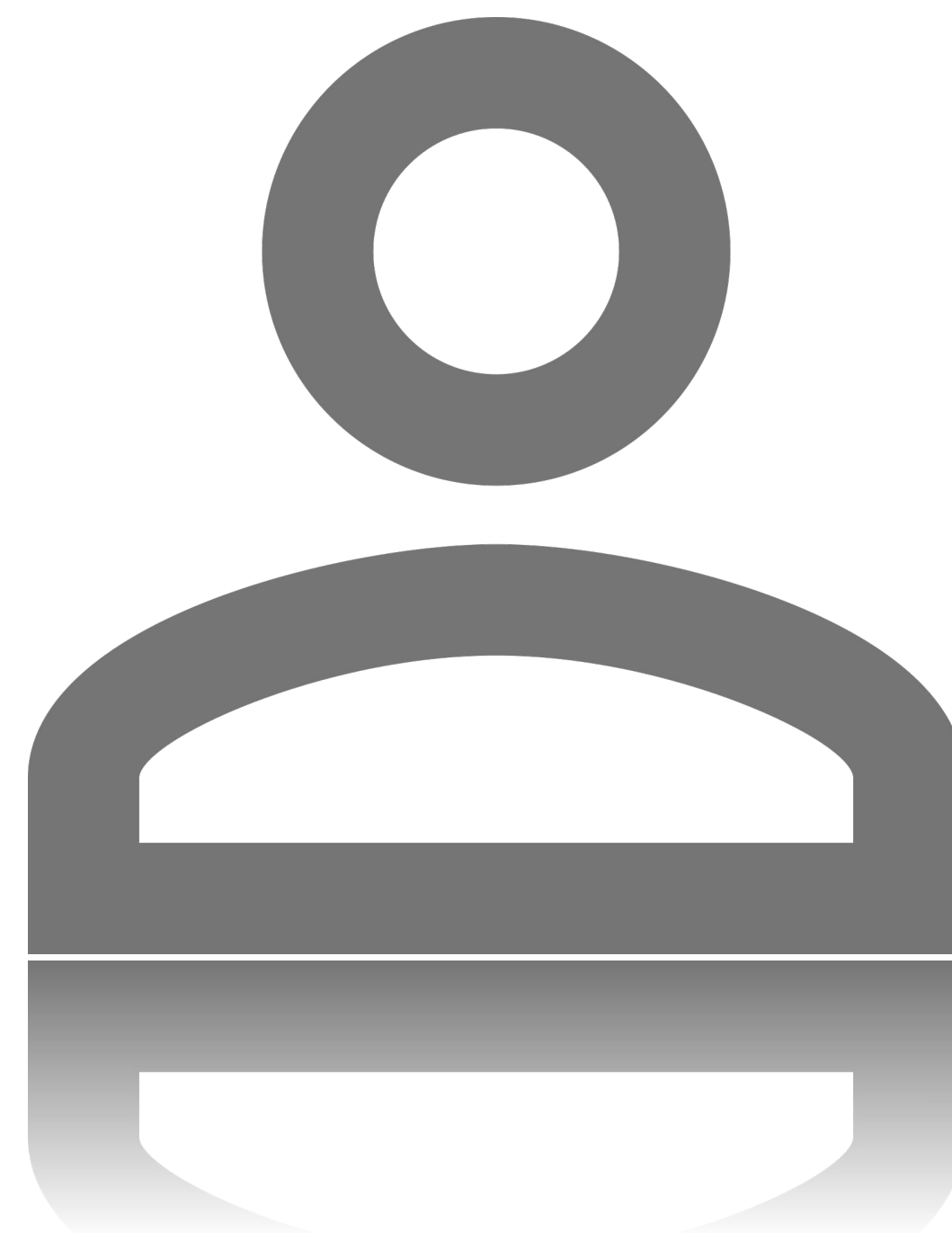


Goal

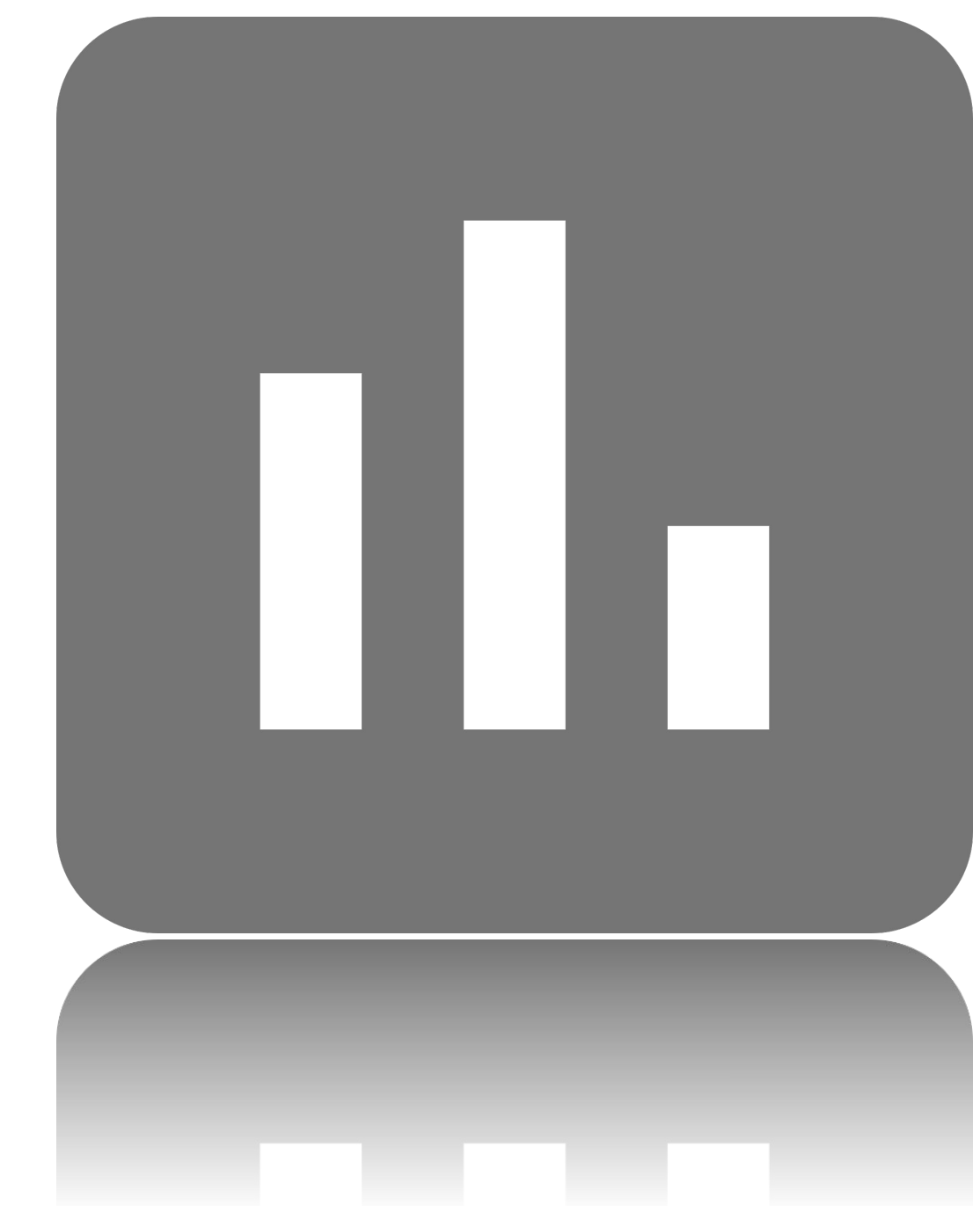
Google Proprietary + Confidential



Traces in all bug reports



No user effort



Minimal overhead

Continuous Logging

Log last WindowManager and SurfaceFlinger states to bug reports

- ~30 seconds
- Minimal overhead
- Only on non-user builds

WindowManager

- WM hierarchy (displays, tasks, apps, windows)
- Animators
- Insets
- Only visible elements

SurfaceFlinger

- SF hierarchy (layers)
- Bounds
- Buffer properties (transform, pixel format)
- All layers

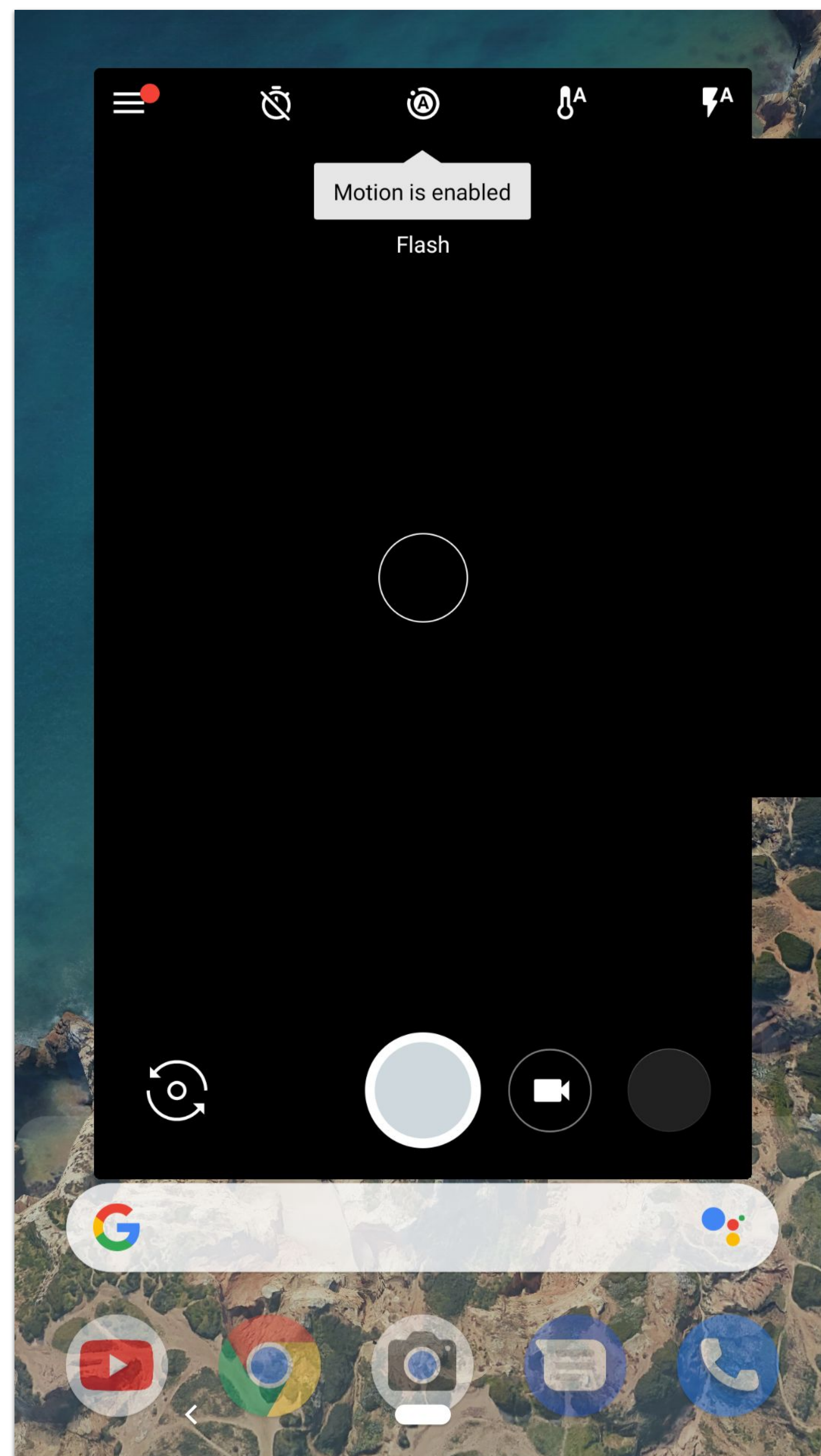
Impact

	WindowManager	SurfaceFlinger
Memory	Up to 2MB	Up to 10MB
Performance	~0.4ms Outside critical path	~0.5ms Outside critical path
Battery	0.15% (Average case) 1.15% (Theoretical worst case)	

WinScope

- *Nataniel Borges*

WinScope



Screen

Hierarchy

☐ Only visible
 ☐ Flat

```

layers - layers
  layer - Display Root#0
    layer - mImeWindowsContainers#0 RelZ
      layer - WindowToken{f5a61a0
        android.os.Binder@f5e0aa3}#0
        layer - ecd9813 InputMethod#0
      layer - mBelowAppWindowsContainers#0
        layer - WallpaperWindowToken{e098510
          token=android.os.Binder@beed5d3}#0
          layer - 7512c4b
            com.android.systemui.ImageWallpaper#0
            layer -
              com.android.systemui.ImageWallpaper#0
              V
        layer -
          com.android.server.wm.DisplayContent$TaskStackCor
          layer - splitScreenDividerAnchor#0
          RelZParent
          layer - Stack=0#0
            layer - animation background stackId=0#0
          layer - homeAnimationLayer#0
            layer -
              Surface(name=Task=15649)/@0x9c6122f -
              animation-leash#0
              layer - Task=15649#0
                layer - AppWindowToken{97b2b13
                  token=Token{5551602
                    ActivityRecord{5af2b4d u0
                      com.google.android.apps.nexuslaunch
                        t15649}}#0
                      layer - 7f10fa8
                        com.google.android.apps.nexuslau

```

Properties

Filter

```

SurfaceView -
com.google.android.GoogleCamera/com
id: 5540
name: SurfaceView -
com.google.android.GoogleCamera/
children
  0: 5541
type: BufferLayer
transparentRegion
  rect: 0
visibleRegion
  rect: 0
    left: 142
    top: 209
    right: 938
    bottom: 1269
damageRegion
  rect: 0
    right: -1
    bottom: -1
z: -2
position
  x: 142.45718383789062
  y: 208.68212890625
requestedPosition
  y: 121
size
  w: 1280
  h: 960
crop
  right: 1280
  bottom: 960

```


WinScope

View WindowManager and
SurfaceFlinger logs

Easily identify errors

Display only changed properties for each
element

The WinScope interface is divided into three main sections:

- Screen:** A visual representation of the screen with various layers. A red rectangle highlights a specific area, and a pink dot indicates a selected element.
- Hierarchy:** A tree view showing the hierarchy of the UI elements. The selected element is highlighted in orange. The hierarchy includes:
 - layers - layers
 - layer - Display Root#0
 - layer - mImeWindowsContainers#0 **RelZ**
 - layer - WindowToken{f5a61a0 android.os.Binder@f5e0aa3}#0
 - layer - ecd9813 InputMethod#0
 - layer - mBelowAppWindowsContainers#0
 - layer - WallpaperWindowToken{e098510 token=android.os.Binder@beed5d3}#0
 - layer - 7512c4b com.android.systemui.ImageWallpaper#0
 - layer - com.android.systemui.ImageWallpaper#0 **V**
 - layer - com.android.server.wm.DisplayContent\$TaskStackCor
 - layer - splitScreenDividerAnchor#0
 - RelZParent**
 - layer - Stack=0#0
 - layer - animation background stackId=0#0
 - layer - homeAnimationLayer#0
 - layer - Surface(name=Task=15649)/@0x9c6122f - animation-leash#0
 - layer - Task=15649#0
 - layer - AppWindowToken{97b2b13 token=Token{5551602 ActivityRecord{5af2b4d u0 com.google.android.apps.nexuslauncher:15649}}#0
 - layer - 7f10fa8 com.google.android.apps.nexuslauncher
- Properties:** A panel showing the properties of the selected element. The selected element is a SurfaceView with the following properties:
 - id: 5540
 - name: SurfaceView - com.google.android.GoogleCamera/com.google.android.GoogleCamera/children
 - type: BufferLayer
 - transparentRegion rect: 0
 - visibleRegion rect: 0
 - left: 142
 - top: 209
 - right: 938
 - bottom: 1269
 - damageRegion rect: 0
 - right: -1
 - bottom: -1
 - z: -2
 - position x: 142.45718383789062 y: 208.68212890625
 - requestedPosition y: 121
 - size w: 1280 h: 960
 - crop right: 1280 bottom: 960

Intent Playground

- *Nataniel Borges*

Intent Playground

10:10 LTE 100%

RegularActivity ⓘ ↗ ⋮

Clear Task

☐ FLAG_ACTIVITY_CLEAR_TASK

☒ FLAG_ACTIVITY_CLEAR_TOP

☐ FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET

New Task

☒ FLAG_ACTIVITY_MULTIPLE_TASK

☐ FLAG_ACTIVITY_NEW_DOCUMENT

☒ FLAG_ACTIVITY_NEW_TASK

☐ FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

Non-user

☐ FLAG_ACTIVITY_BROUGHT_TO_FRONT

☐ FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY

Recents and UI

☐ FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS

☐ FLAG_ACTIVITY_LAUNCH_ADJACENT

☐ FLAG_ACTIVITY_NO_ANIMATION

< ▢



10:10 LTE 100%

TaskAffinity1Activity ⓘ ⋮

Current Tasks

Task #2904
.TaskAffinity1Activity

Task #2901
.RegularActivity

Current Intent

Action

Data URI

Type

Package

Categories

Flags

FLAG_ACTIVITY_CLEAR_TOP
FLAG_ACTIVITY_MULTIPLE_TASK
FLAG_ACTIVITY_NEW_TASK

↗

< ▢

Q & A

THANK YOU

Up Next...

[go/md-foldables-abc-2019](#)