Deep Inside Android Hacks

Keishin Yokomaku (KeithYokoma) @ Drivemode, Inc.

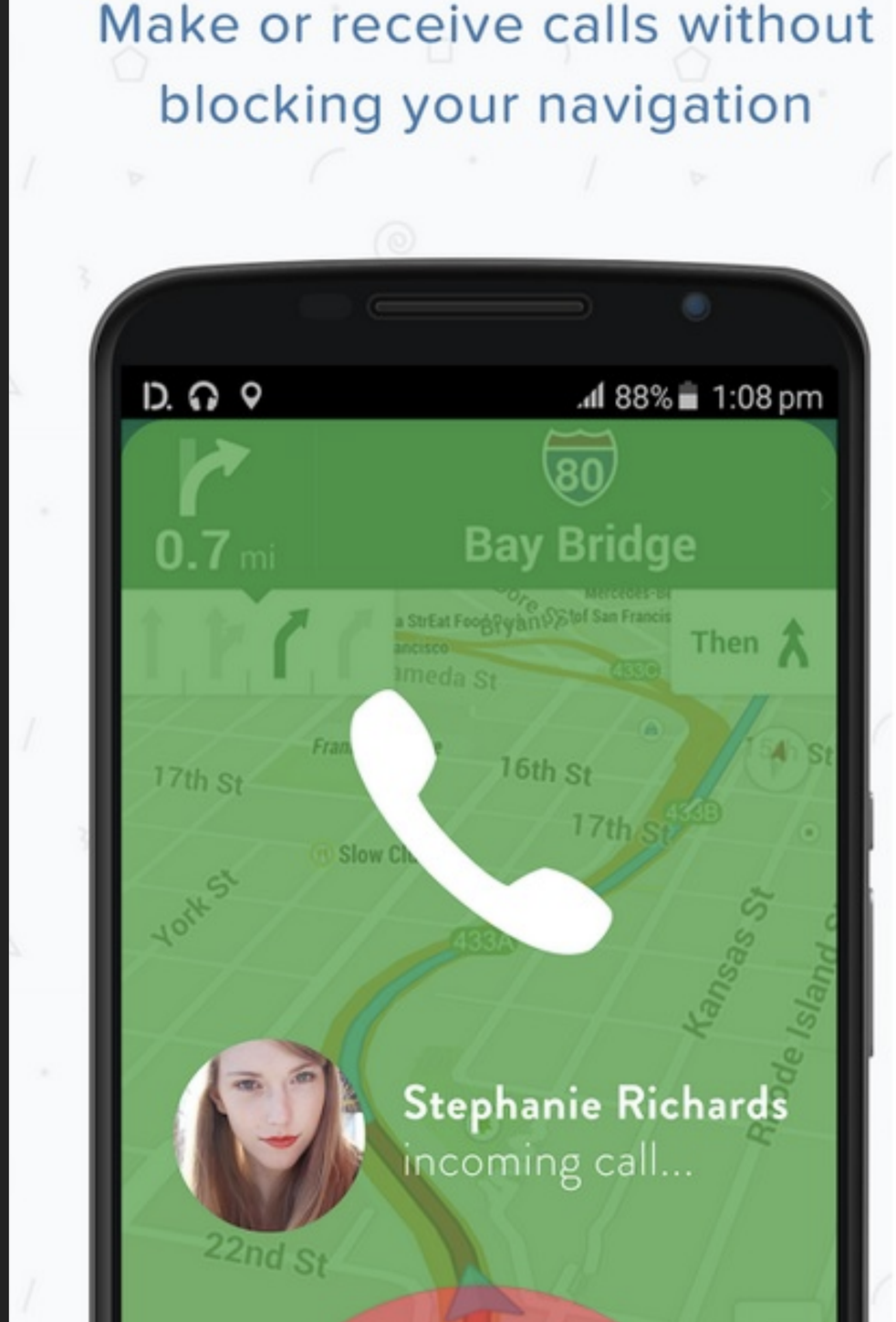
Android All Stars #dotsandroid

Profile

- Keishin Yokomaku at Drivemode, Inc.
- Social: @KeithYokoma
- Publication: Android Training
- Product:
- Like: Bicycle, Photography, Tumblr
- Nickname: Qiita Meister

Drivemode

- Now available on Play Store!
- http://bit.ly/1LYdxAg



E-book

- AndroidTraining / iOS Training
- http://amzn.to/1mZNydv

How to Hack Android

How to Hack Android

- Using public APIs
- Reflection to access hidden APIs
- AIDL to communicate with System services

Using public APIs

Using public APIs

- It’s official!
- Less likely to crash coming from customizations

android.media.session

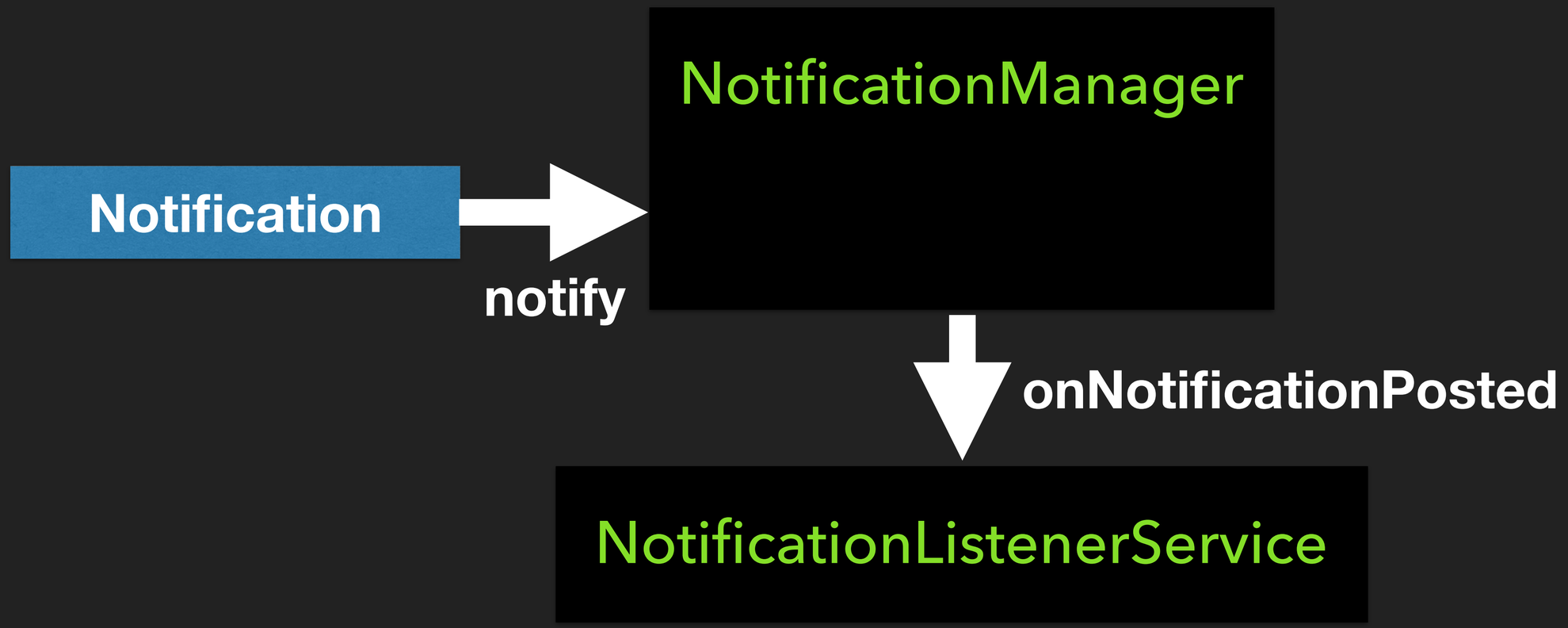
- Lollipop API
- Components
  - MediaSession and MediaSession.Token
  - MediaController
  - Notification.MediaStyle

MediaSession and MediaController

- Permit transport control by session token

NotificationListenerService

- Background service listening to status bar notification events



Notification.MediaStyle

- Pass MediaSession.Token to system notification
- When you listen to notification event…
  - MediaSession.Token is in `extras`
  - Key for `extras` is defined on Notification
  - MediaSession.Token is valid even if the receiver is not intended to

“This is more like session hijacking”

-Someone

Take control of music playback

```
public class MusicNotificationWatcher extends NotificationListenerService {
  private Bus eventBus;
  @Override
  public void onNotificationPosted(StatusBarNotification sbn) {
    MediaSession.Token token =
        sbn.getNotification().extras.getParcelable(Notification.EXTRA_MEDIA_SESSION);
    eventBus.post(new NewMusicNotification(token));
  }
}

public class MyActivity extends Activity {
  private MediaController controller;
  @Subscribe
  public void onNewMusicNotificationAdded(NewMusicNotification event) {
    controller = new MediaController(this, event.getToken())
  }
}
```

Take control of music playback

```
public class MusicNotificationWatcher extends NotificationListenerService {
  private Bus eventBus;
  @Override
  public void onNotificationPosted(StatusBarNotification sbn) {
    MediaSession.Token token =
        sbn.getNotification().extras.getParcelable(Notification.EXTRA_MEDIA_SESSION);
    eventBus.post(new NewMusicNotification(token));
  }
}

public class MyActivity extends Activity {
  private MediaController controller;
  @Subscribe
  public void onNewMusicNotificationAdded(NewMusicNotification event) {
    controller = new MediaController(this, event.getToken())
  }
}
```

Take control of music playback

```
public class MusicNotificationWatcher extends NotificationListenerService {
  private Bus eventBus;
  @Override
  public void onNotificationPosted(StatusBarNotification sbn) {
    MediaSession.Token token =
        sbn.getNotification().extras.getParcelable(Notification.EXTRA_MEDIA_SESSION);
    eventBus.post(new NewMusicNotification(token));
  }
}

public class MyActivity extends Activity {
  private MediaController controller;
  @Subscribe
  public void onNewMusicNotificationAdded(NewMusicNotification event) {
    controller = new MediaController(this, event.getToken())
  }
}
```

“Is it practical?”

Is it practical?

- It depends
  - You cannot fully cover overall music experience
  - For keyguard apps, this is good enough

“Is it affordable?”

Is it affordable?

- No
  - Because it works only on Lollipop and with Google Play Music

Reflection to access hidden APIs

Reflection to access hidden APIs

- It’s unofficial, dirty, unpaved…
- No IDE support for you
- Be careful about ProGuard settings
- Breaking changes might be happening under the hood
- Performance issue

ProGuard

- Do not obfuscate statements called via reflection

Reflection basics

- Class
  - Object#getClass(), Class.forName(), Class literal
- Method
  - Class#getDeclaredMethods(), Class#getDeclaredMethod()
- Field
  - Class#getDeclaredFields(), Class#getDeclaredField()

Example

```
package com.sample;

public class Something {
  private void foo(String str) {}
}

public class ReflectionSample {
  public void reflect(Something something) throws Exception {
    Method foo = something.getDeclaredMethod(“foo”, String.class);
    foo.setAccessible(true);
    foo.invoke(something, “bar”);
  }
}
```

Example

```
package com.sample;

public class Something {
  private void foo(String str) {}
}

public class ReflectionSample {
  public void reflect(Something something) throws Exception {
    Method foo = something.getDeclaredMethod(“foo”, String.class);
    foo.setAccessible(true);
    foo.invoke(something, “bar”);
  }
}
```

Example

```
package com.sample;

public class Something {
  private void foo(String str) {}
}

public class ReflectionSample {
  public void reflect(Something something) throws Exception {
    Method foo = something.getDeclaredMethod("foo", String.class);
    foo.setAccessible(true);
    foo.invoke(something, "bar");
  }
}
```

Example

```
package com.sample;

public class Something {
  private void foo(String str) {}
}

public class ReflectionSample {
  public void reflect(Something something) throws Exception {
    Method foo = something.getDeclaredMethod("foo", String.class);
    foo.setAccessible(true);
    foo.invoke(something, "bar");
  }
}
```

Accessing hidden APIs

- Methods / Fields
  - getDeclared** to get private (or hidden) one
  - setAccessible(true) to make it visible to us

“Why are you using wrecking reflection?”
“Because it is practical”

Practicality of reflection

- Aggressive usage
  - to get informations that is generally prohibited for normal apps
  - to use system functions
- Reluctant usage
  - to avoid/patch bugs in framework

Aggressive reflection

- You can do almost everything
  - Read `RemoteViews` operations
  - Read actual `Intent` on `PendingIntent`
  - Call hidden method to register object onto system service

Karma of aggressive reflection

- Spoil encapsulation
  - Need to pay attention to the object state
- It may not work
  - No guarantee that every phone has the same method or field
- Google is watching you

Reluctant reflection

- Avoid bugs in framework
  - Ex. http://bit.ly/1HkJvR4
    - Fix wrong path for preferences files on G***** S
  - Ex. http://bit.ly/1E5kB7L
    - Backward compatibility

Benefits of reluctant reflection

- Reduce a lot of pains on users who are using broken phone
- Keep new API available on old phones

AIDL to communicate with System services

AIDL to communicate with System services

- Yet another dirty work
- Be careful about ProGuard settings

ProGuard

- Do not obfuscate auto generated codes and implementation

AIDL basics

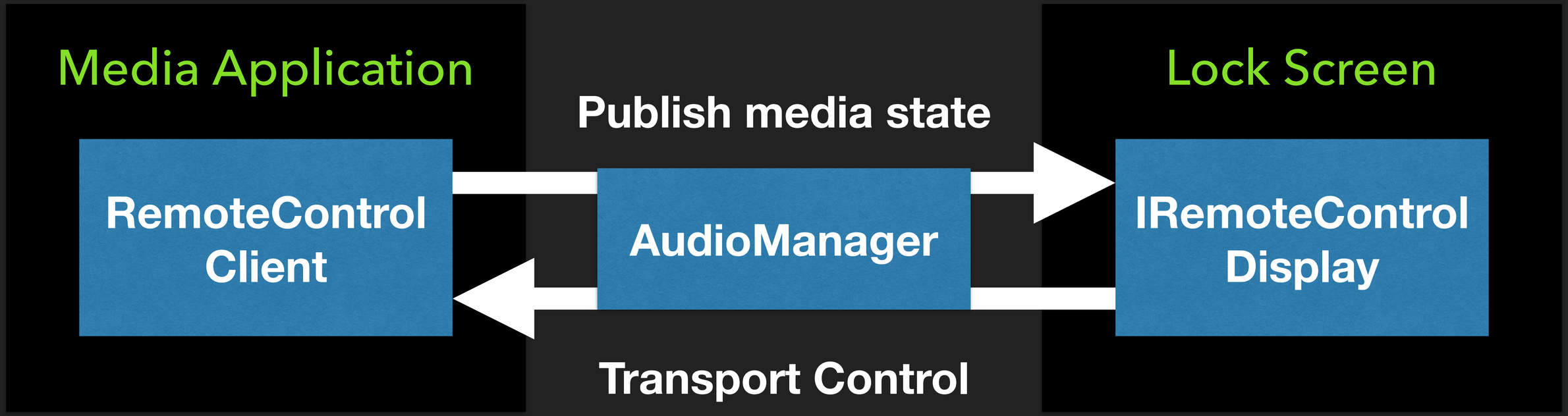
- Implementing AIDL stub
  - Put .aidl in `src/main/aidl`
  - Compile it
  - Implement stub methods in Java code

A lot of AIDLs in framework

- Intent, Bundle, Uri, Bitmap, Rect, Account…
  - Data container objects passed to each processes
- MediaSession, ICameraService, ITelephonyService…
  - Abstraction of some operations

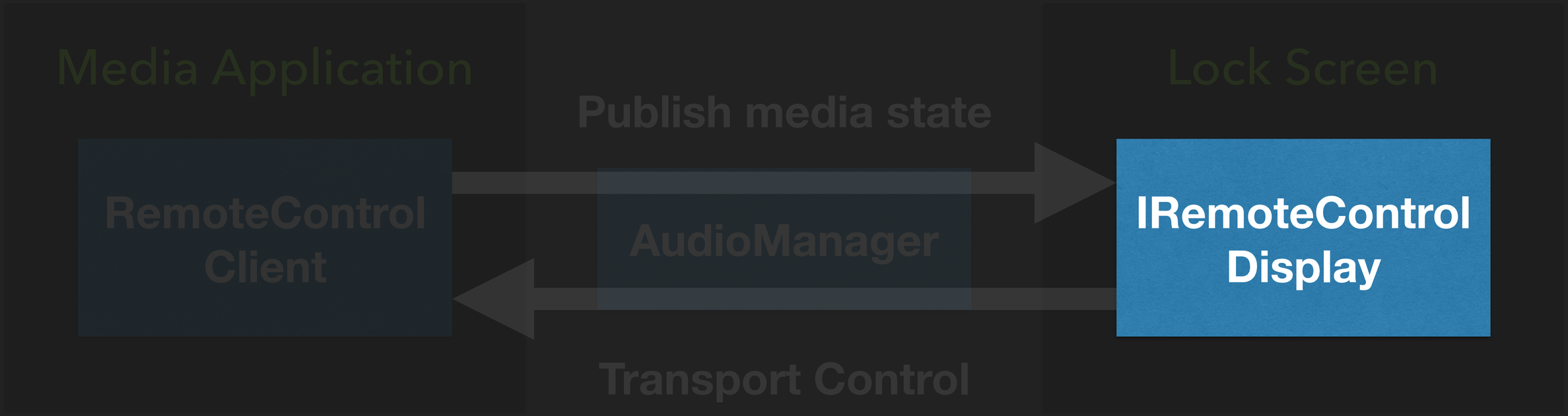
Media Controller on the Lock Screen

- Android 4.0 and above (up to Android 4.2)
- RemoteControlClient and IRemoteControlDisplay



Media Controller on the Lock Screen

- Android 4.0 and above (up to Android 4.2)
- RemoteControlClient and IRemoteControlDisplay



IRemoteControlDisplay

```
// in IRemoteControlDisplay.aidl
package android.media;

oneway interface IRemoteControlDisplay {
  // various methods declared…
}

// in Java code
public class RemoteControlDisplay extends IRemoteControlDisplay.Stub {
  // various implementations here…
}
```

AudioManager

```
private final AudioManager mAudioManager;
private final IRemoteControlDisplay mDisplay;

public void setUp() throws Exception {
  Method register = mAudioManager.getClass().getDeclaredMethod(
      “registerRemoteControlDisplay”, IRemoteControlDisplay.class);
  register.invoke(mAudioManager, mDisplay);
}

public void tearDown() throws Exception {
  Method unregister = mAudioManager.getClass().getDeclaredMethod(
      “unregisterRemoteControlDisplay”, IRemoteControlDisplay.class);
  unregister(mAudioManager, mDisplay);
}
```

AIDL Versioning

- IRemoteControlDisplay
  - IRemoteControlDisplay is available from ICS
  - New method is added on JB
- But…
  - The method name is the same (overloaded)
  - AIDL does not support overloading

Workaround for method overload

- AIDL definition
  - Keep the latest
- Java implementation
  - Declare every version of overloaded methods

IRemoteControlDisplay

```
// ICS version
oneway interface IRemoteControlDisplay {
  setPlaybackState(int id, int state, long stateChangeTimeMs);
}

// JB version
oneway interface IRemoteControlDisplay {
  setPlaybackState(int id, int state, long stateChangeTimeMs,
      long currentPosMs, float speed);
}
```

IRemoteControlDisplay

```
public class RemoteControlDisplay extends IRemoteControlDisplay.Stub {
  public void setPlaybackState(int id, int state, long stateChangeTimeMs) {

  }

  @Override
  public void setPlaybackState(int id, int state, long stateChangeTimeMs,
      long currentPosMs, float speed) {

  }
}
```


I'VE MENDED IT!
I'VE MENDED SOMETHING!

Internal AIDL usage

- Be aware of method declaration
  - If no method found, the app will crash
- Keep compatible
  - Define the same signature method on implementation

“Don’t be afraid.
Keep calm and happy hacking!”

Deep Inside Android Hacks

Keishin Yokomaku (KeithYokoma) @ Drivemode, Inc.
Android All Stars