# Android Services & Local IPC: The Broker Pattern (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

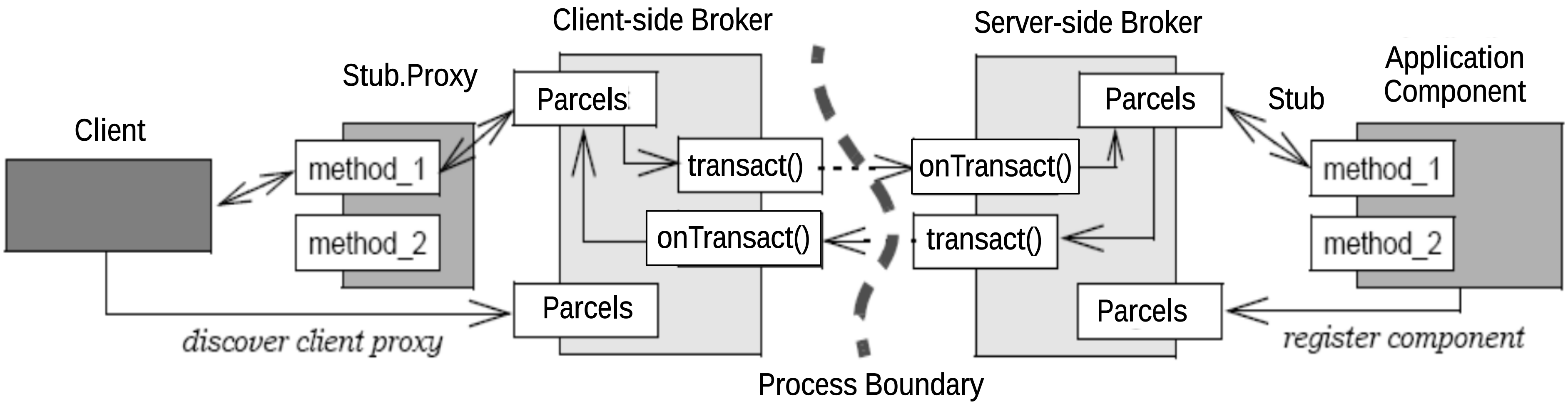
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Module

- Understand how the *Broker* pattern is applied in Android



# Broker POSA1 Architectural Pattern

**Implementation**

- Define an invocation interface
  - Requestor's invocation interface allows clients to construct & send requests

```
public class Binder
       implements IBinder {
  ...
  public final boolean
    transact(int code,
             Parcel data,
             Parcel reply,
             int flags) ... {
    if (data != null)
      data.setDataPosition(0);
    boolean r = onTransact(code,
                           data,
                           reply,
                           flags);
    if (reply != null)
      reply.setDataPosition(0);
    return r;
  }
```

frameworks/base/core/java/android/os/Binder.java has the source code

# Broker POSA1 Architectural Pattern

## Implementation

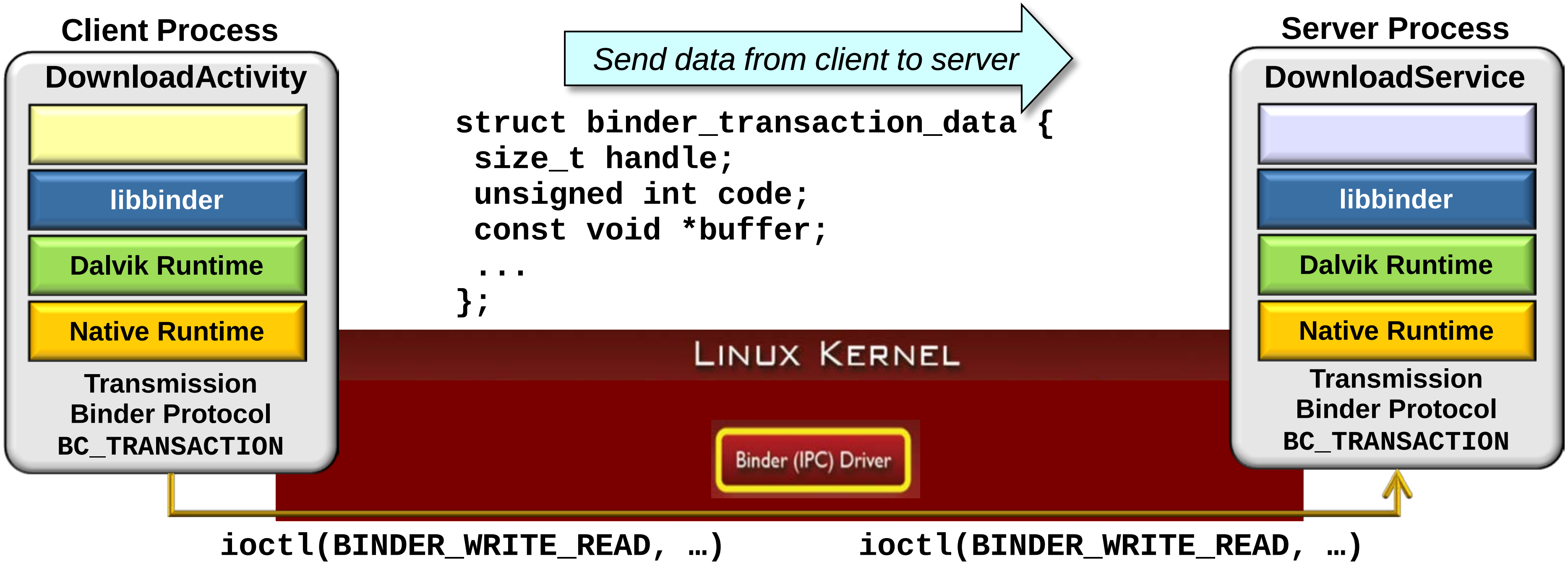
- Define an invocation interface
- Select & implement the marshaler
  - See the *Proxy* discussion for details

```
private static class Proxy
        implements IDownload {
  public String downloadImage(
    String uri) ... {
  android.os.Parcel _data =
    android.os.Parcel.obtain();
  android.os.Parcel _reply =
     android.os.Parcel.obtain();
  _data.writeString(uri);
  mRemote.transact
    (Stub.TRANSACTION_downloadImage,
     _data, _reply, 0);
  _reply.readException();
  java.lang.String _result =
    _reply.readString();
  ...
  return _result;
  ...
```

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
    - e.g., connection-oriented vs. connectionless



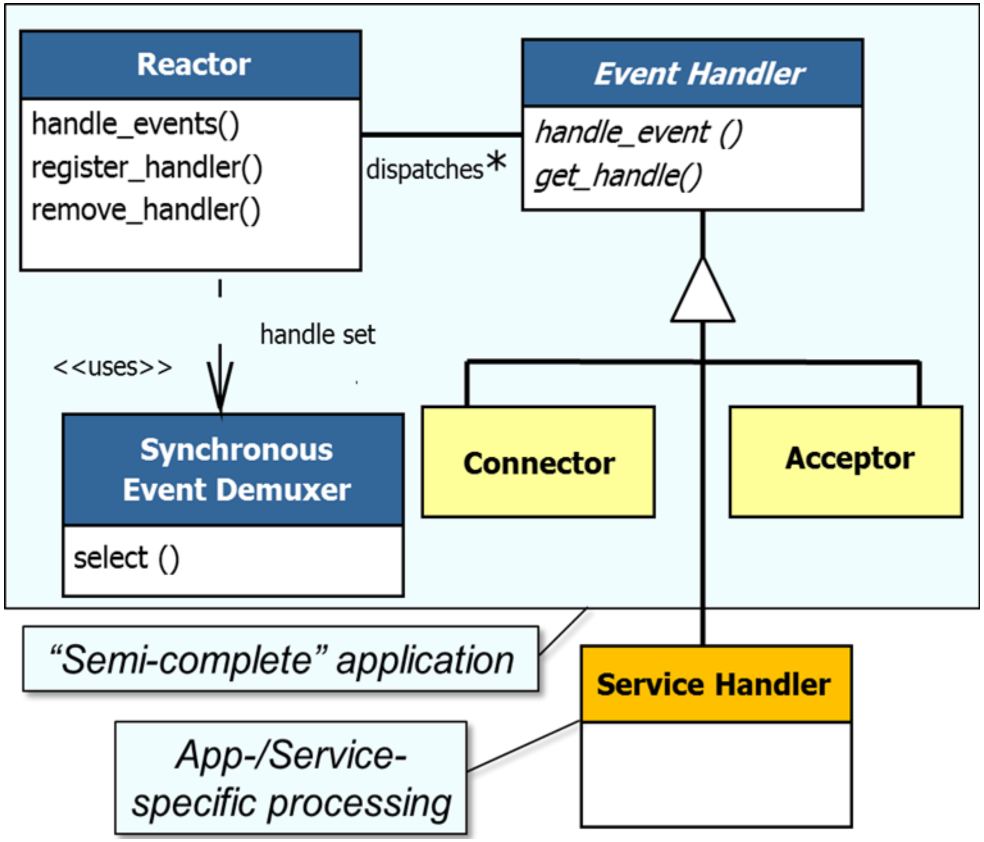
```
struct binder_transaction_data {
 size_t handle;
 unsigned int code;
 const void *buffer;
 ...
};
```

rts.lab.asu.edu/web_438/project_final/Talk%208%20AndroidArc_Binder.pdf

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
  - e.g. use the *Acceptor/Connector* pattern to establish connections between requestor & dispatcher & *Reactor* for demxuing incoming requests & responses

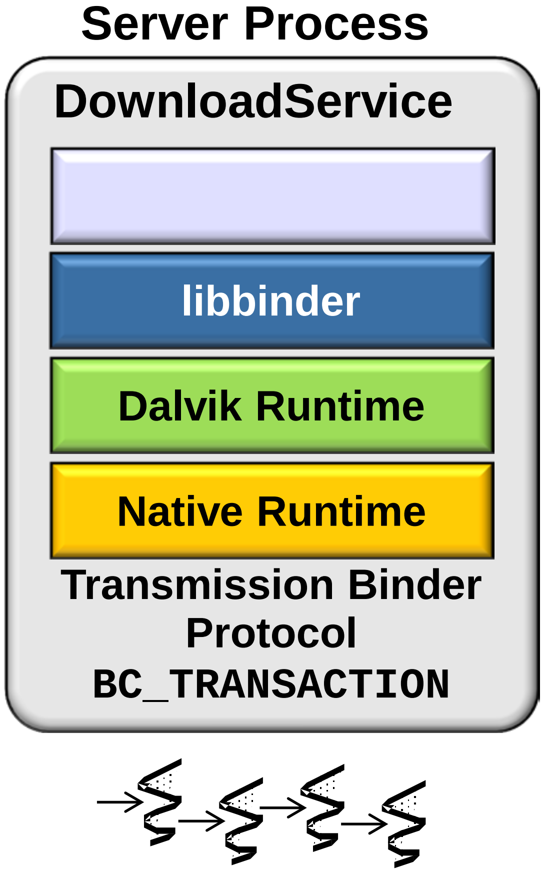


See www.dre.vanderbilt.edu/~schmidt/PDF/TAPOS.pdf for more info

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
  - Connections between requestors & dispatchers can be reused & shared using the Caching & Pooling pattern, respectively



kircher-schwanninger.de/michael/publications/{Caching,Pooling}.pdf

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
  - Provided by the dispatcher for the registration & unregistration of servants

```
public class Binder
       implements IBinder {
  ...
  public void attachInterface
               (IInterface owner,
                String descriptor)
    {
      mOwner = owner;
      mDescriptor = descriptor;
    }
  ...
```

frameworks/base/core/java/android/os/Binder.java has the source code

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
- Provide a mechanism to reference servants
  - To perform requests on remote objects, represented by servants, the clients have to obtain references to those remote objects

```
public class Service extends
                 ... {
  ...
   public abstract IBinder
      onBind(Intent intent);
    ...
}
```

*Factory method that returns a reference to a Binder object*

# Broker POSA1 Architectural Pattern

**Implementation**

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
- Provide a mechanism to reference servants
  - To perform requests on remote objects, represented by servants, the clients have to obtain references to those remote objects

```
public class Service extends
                 ... {
  ...
   public abstract IBinder
      onBind(Intent intent);
   ...
}
```

```
interface ServiceConnection {
  public void
    onServiceConnected
          (ComponentName name,
           IBinder service);
  ...
}
```

*Hook method to pass Binder reference back to client*

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
- Provide a mechanism to reference servants
- Implement the mechanism to transform request messages into upcalls on servants

```
public static abstract class Stub
       extends android.os.Binder
       implements IDownload {
  public boolean onTransact
          (int code,
           android.os.Parcel data,
           android.os.Parcel reply,
           int flags) ... {
    switch (code) {
    case TRANSACTION_downloadImage:
      ...
      java.lang.String _arg0 =
        data.readString();
      java.lang.String _result =
        this.downloadImage(_arg0);
      ...
```

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
- Provide a mechanism to reference servants
- Implement the mechanism to transform request messages into upcalls on servants
- Decide if/how to support asynchrony

```
interface IDownload {
  oneway void setCallback
    (in IDownloadCallback
        callback);
}

interface IDownloadCallback {
  oneway void sendPath
    (in String path);
}
```

# Broker POSA1 Architectural Pattern

## Implementation

- Define an invocation interface
- Select & implement the marshaler
- Select communication protocol
- Implement network communication
- Implement resource management
- Define an registration interface
- Provide a mechanism to reference servants
- Implement the mechanism to transform request messages into upcalls on servants
- Decide if/how to support asynchrony
- Optimize local invocations

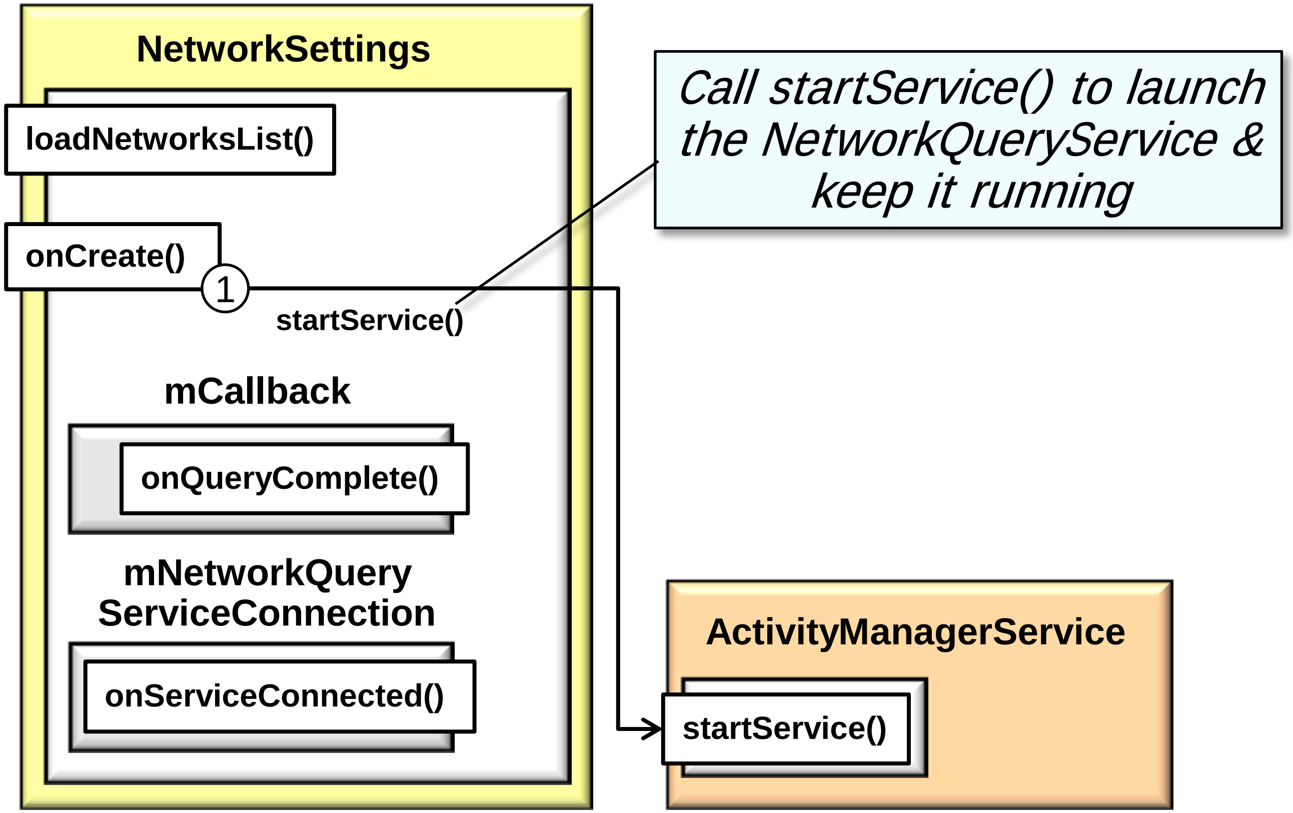
```
public static abstract class Stub
       extends android.os.Binder
       implements IDownload {
  ...
  public static IDownload
    asInterface
     (android.os.IBinder obj) {
    if ((obj==null)) return null;
    android.os.IInterface iin =
     (android.os.IInterface)
    obj.queryLocalInterface
                    (DESCRIPTOR);
    if(((iin != null) &&
      (iin instanceof IDownload)))
      return ((IDownload)iin);
    return new IDownload.Stub.
              Proxy(obj);
  }
```

www.dre.vanderbilt.edu/~schmidt/PDF/COOTS-99.pdf has more info

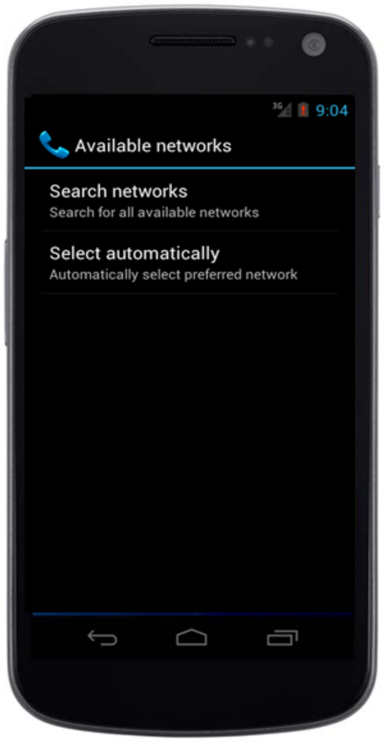
# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability



*Call startService() to launch the NetworkQueryService & keep it running*



9:04

Available networks

Search networks
Search for all available networks

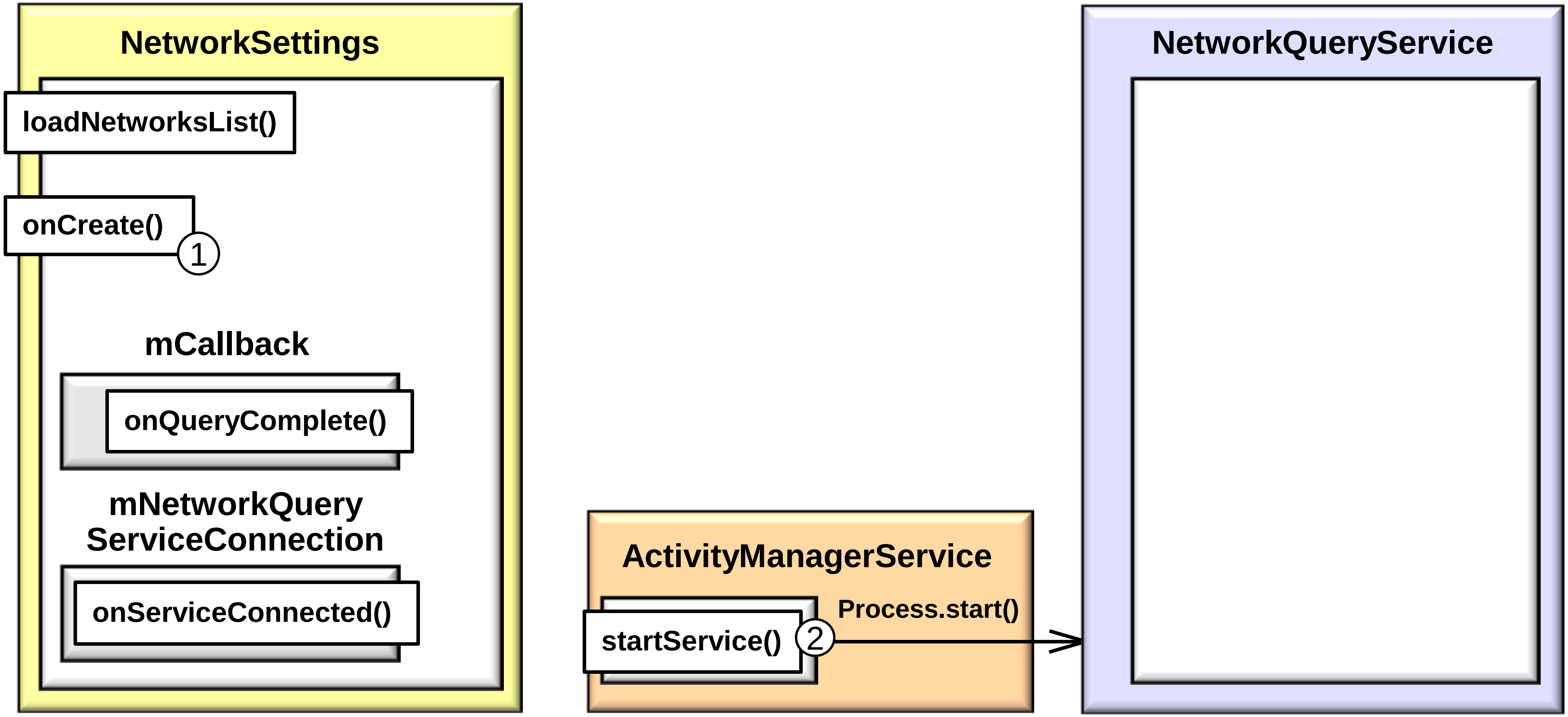
Select automatically
Automatically select preferred network

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

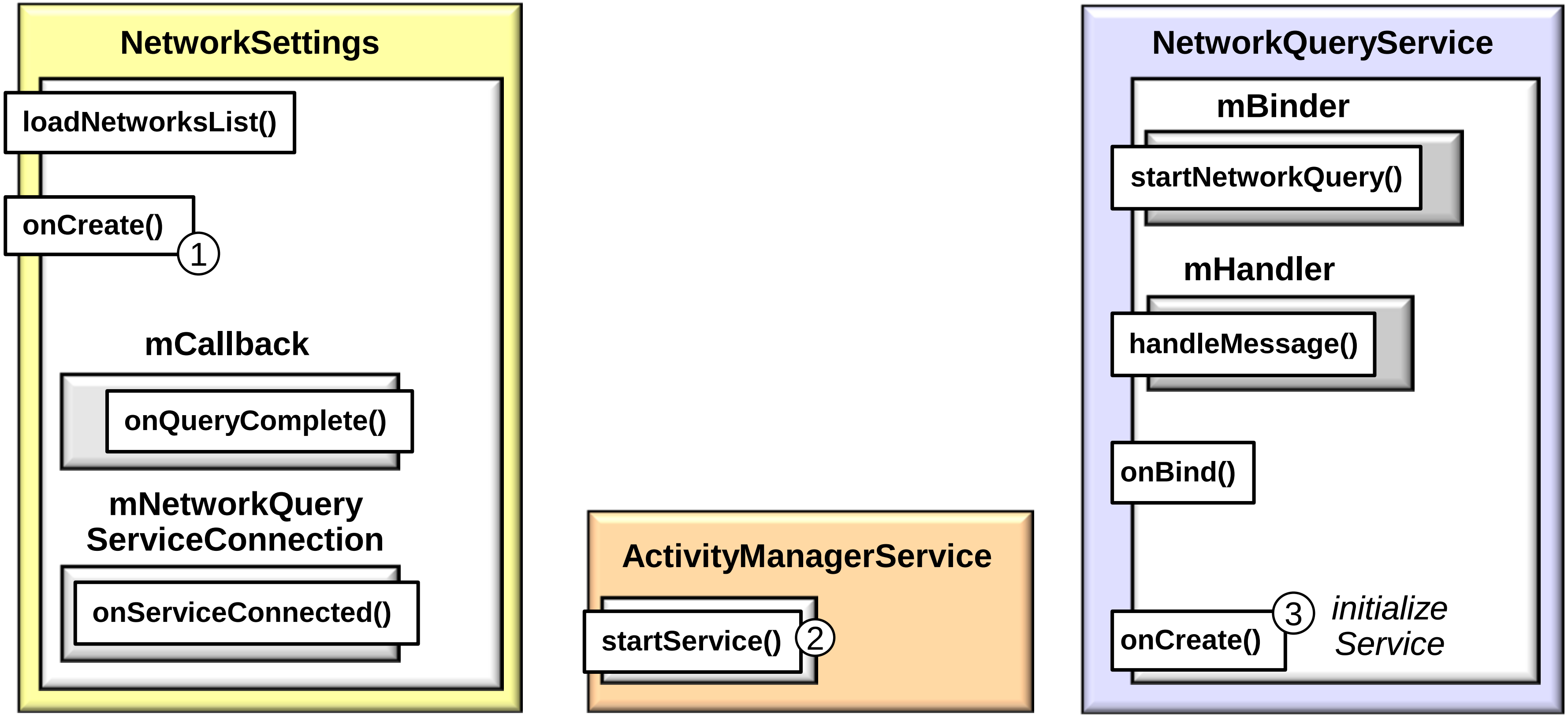


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

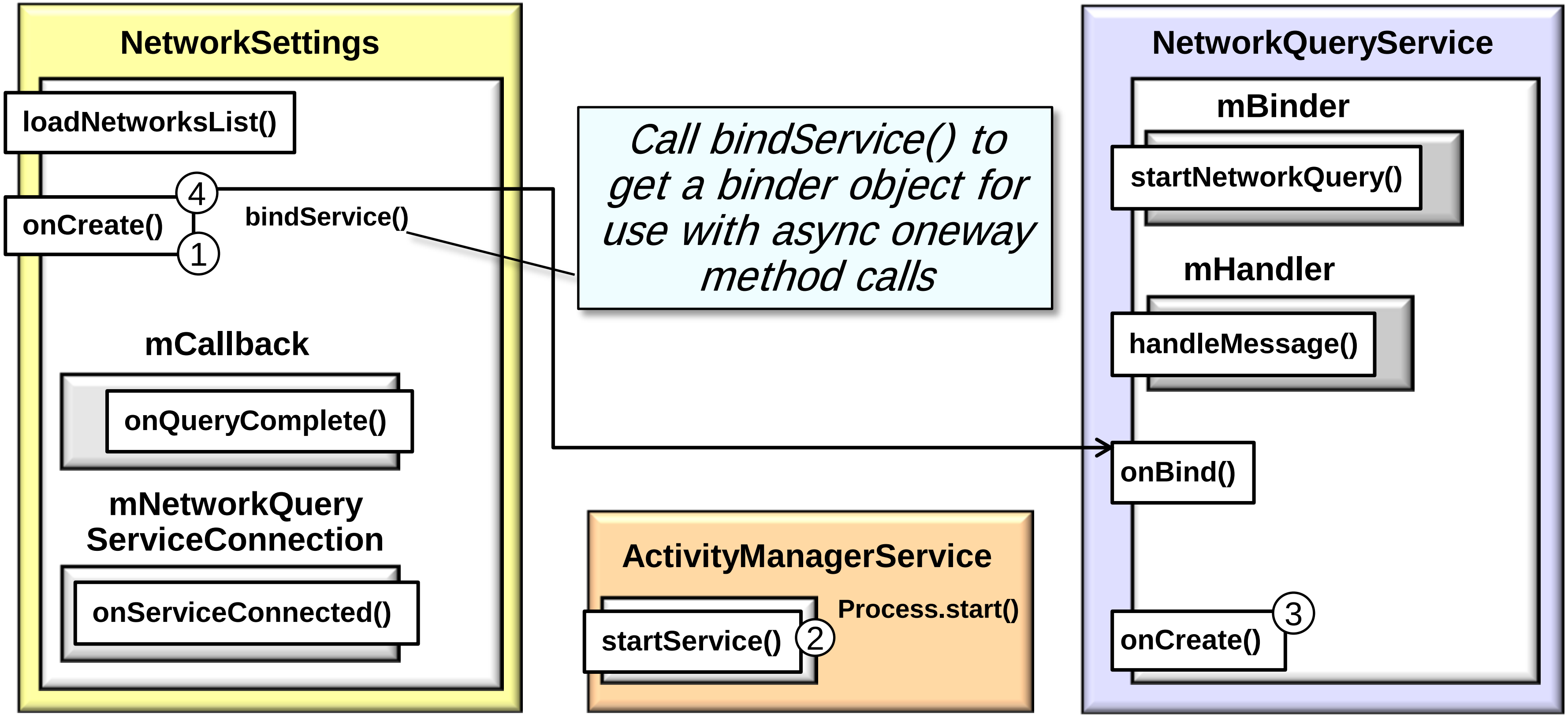


packages/apps/Phone/src/com/android/phone/NetworkQueryService.java has source

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability



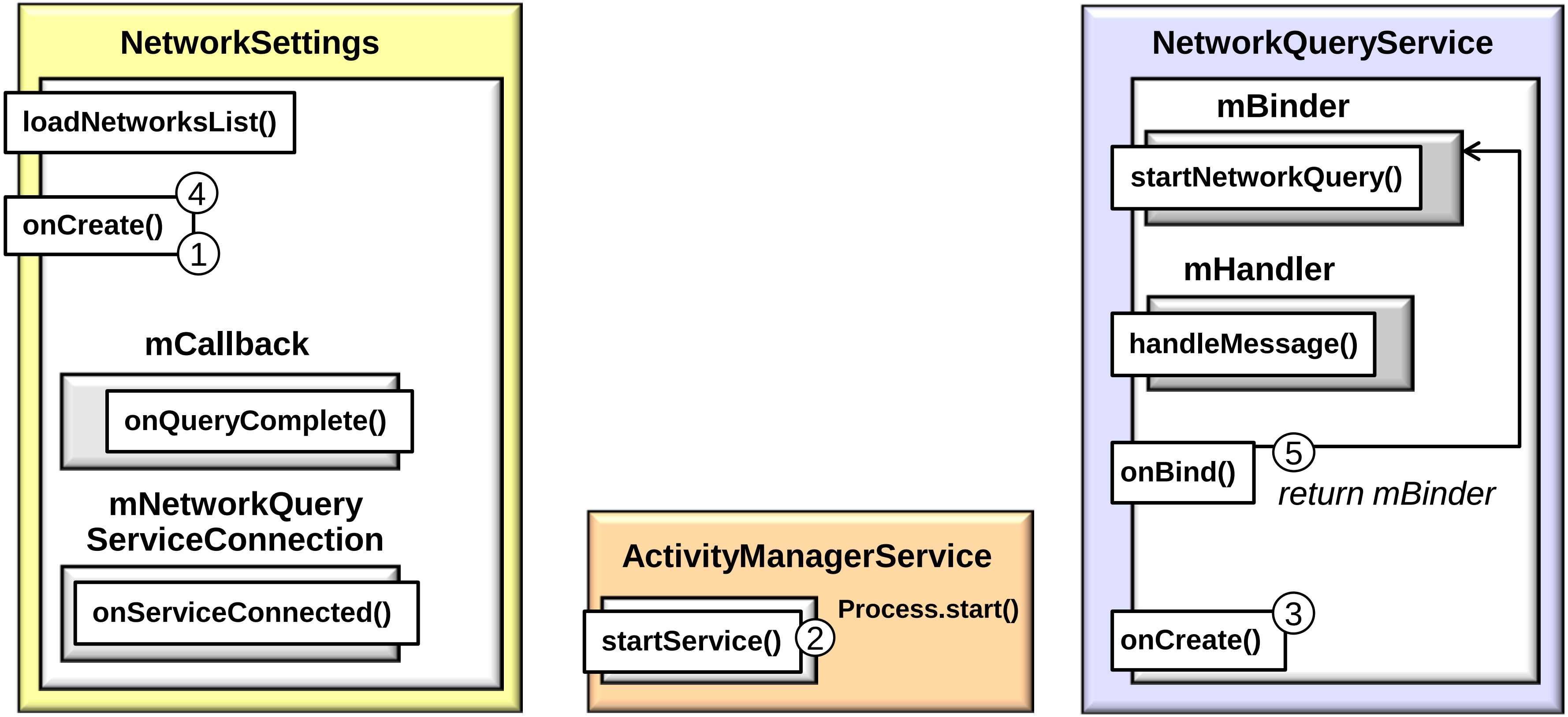
*Call bindService() to get a binder object for use with async oneway method calls*

packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

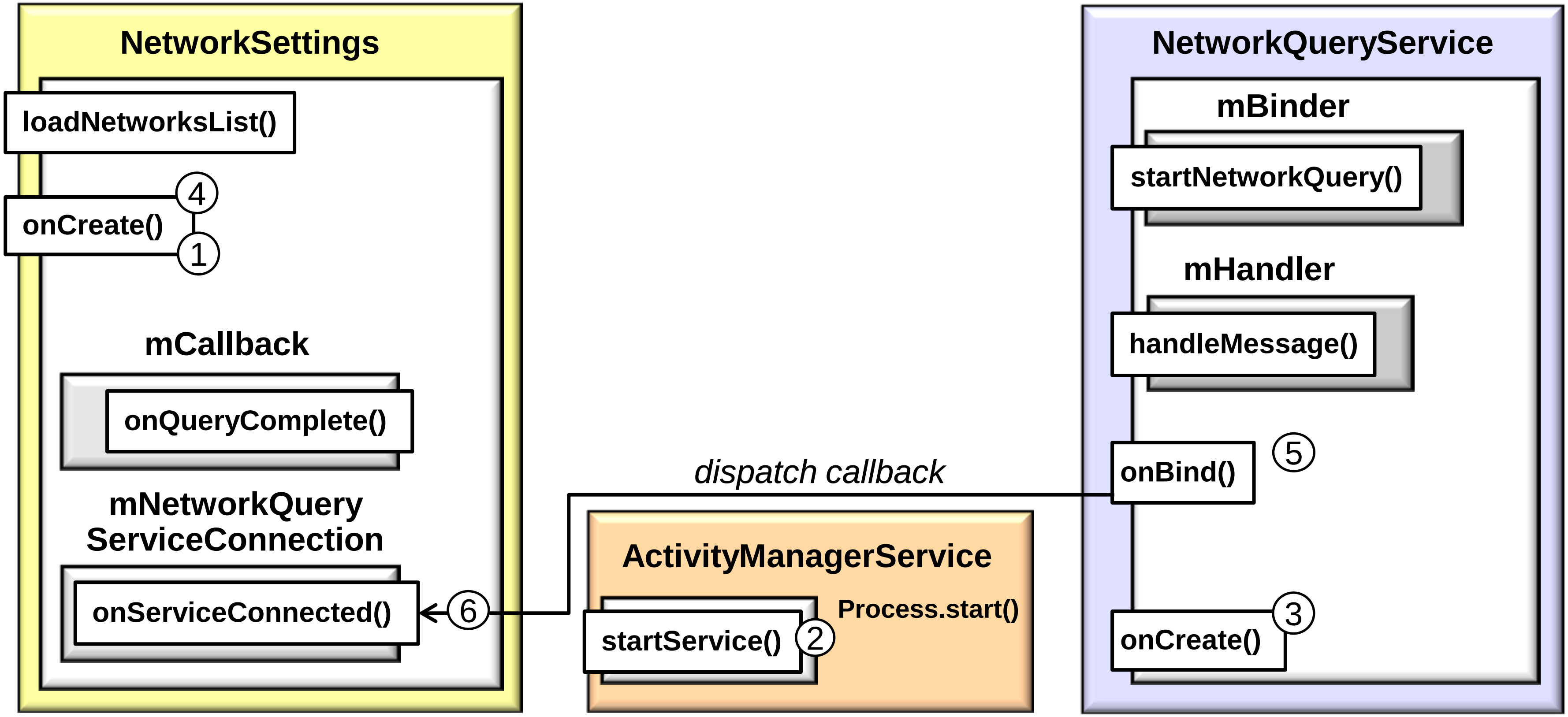


packages/apps/Phone/src/com/android/phone/NetworkQueryService.java has source

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

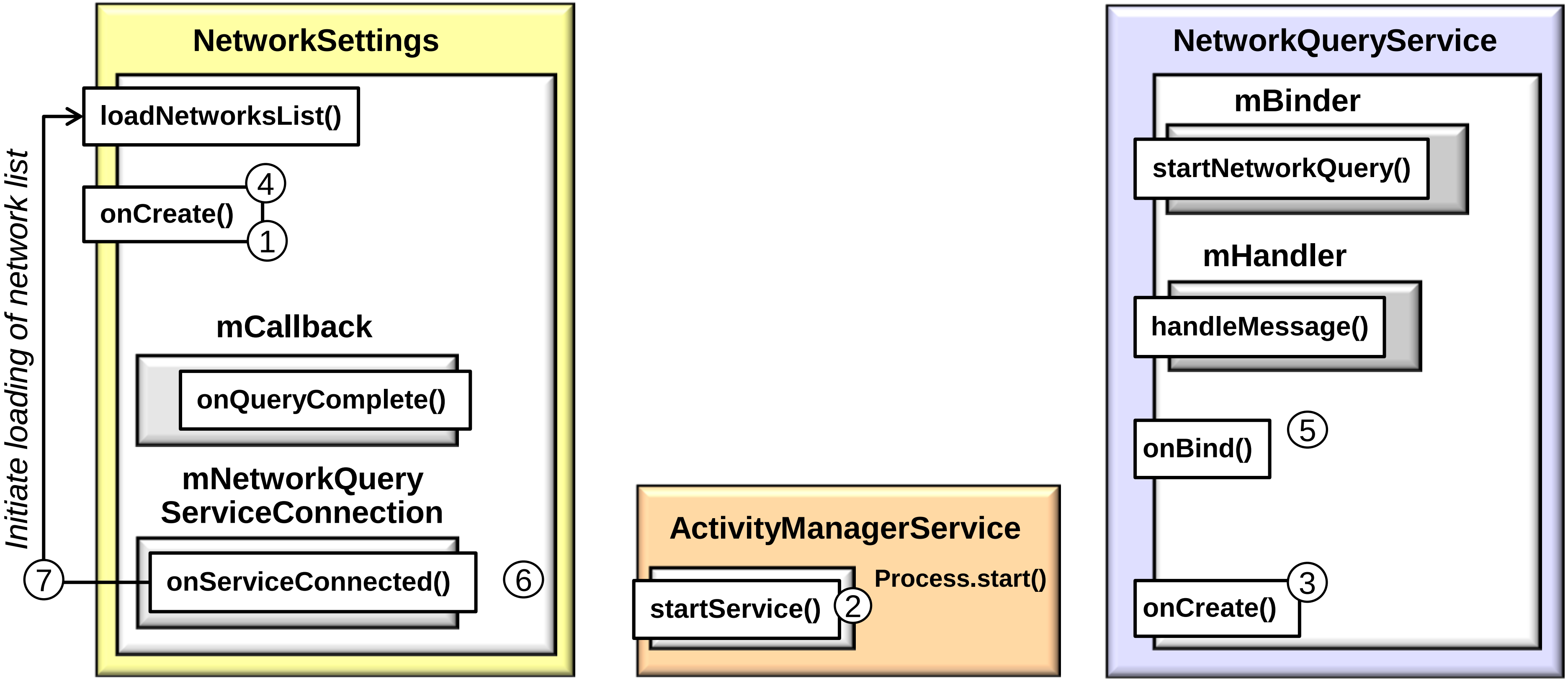


packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

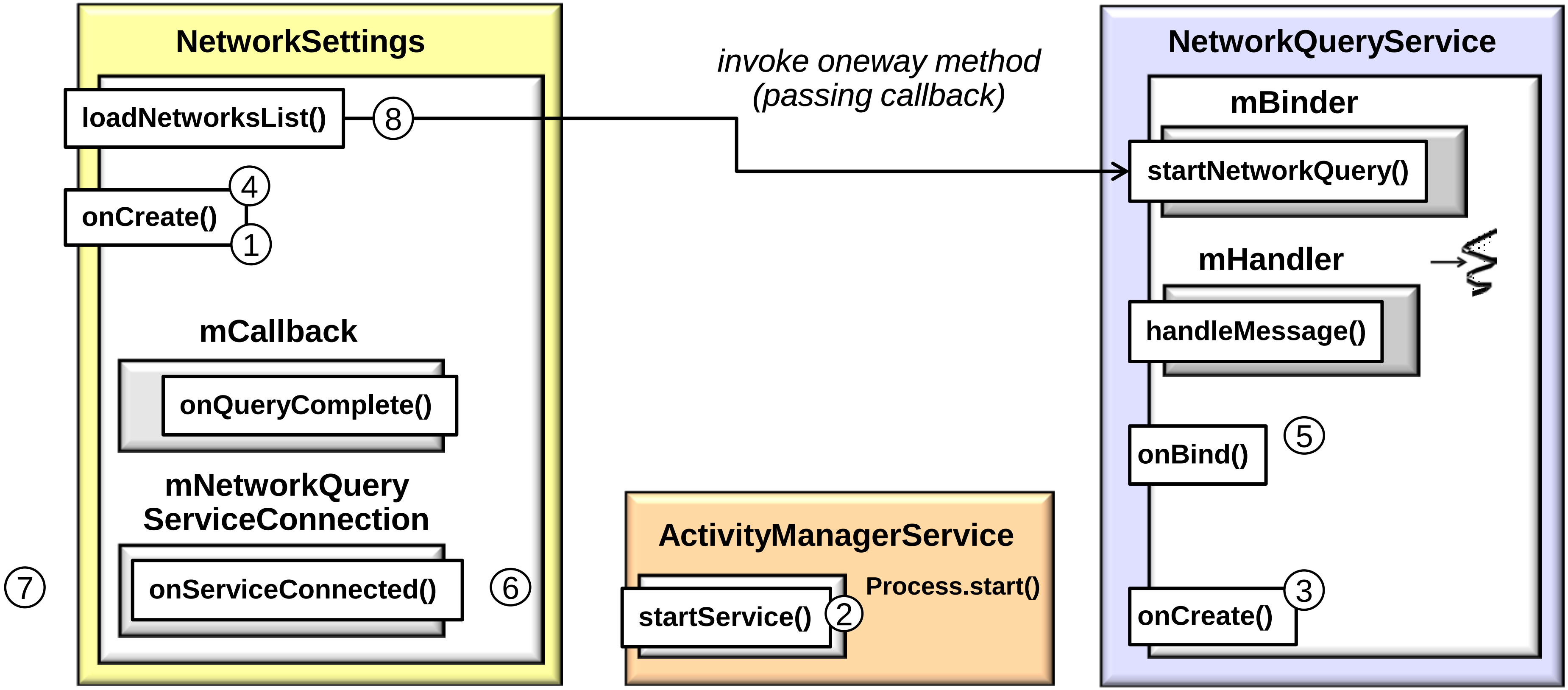


packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

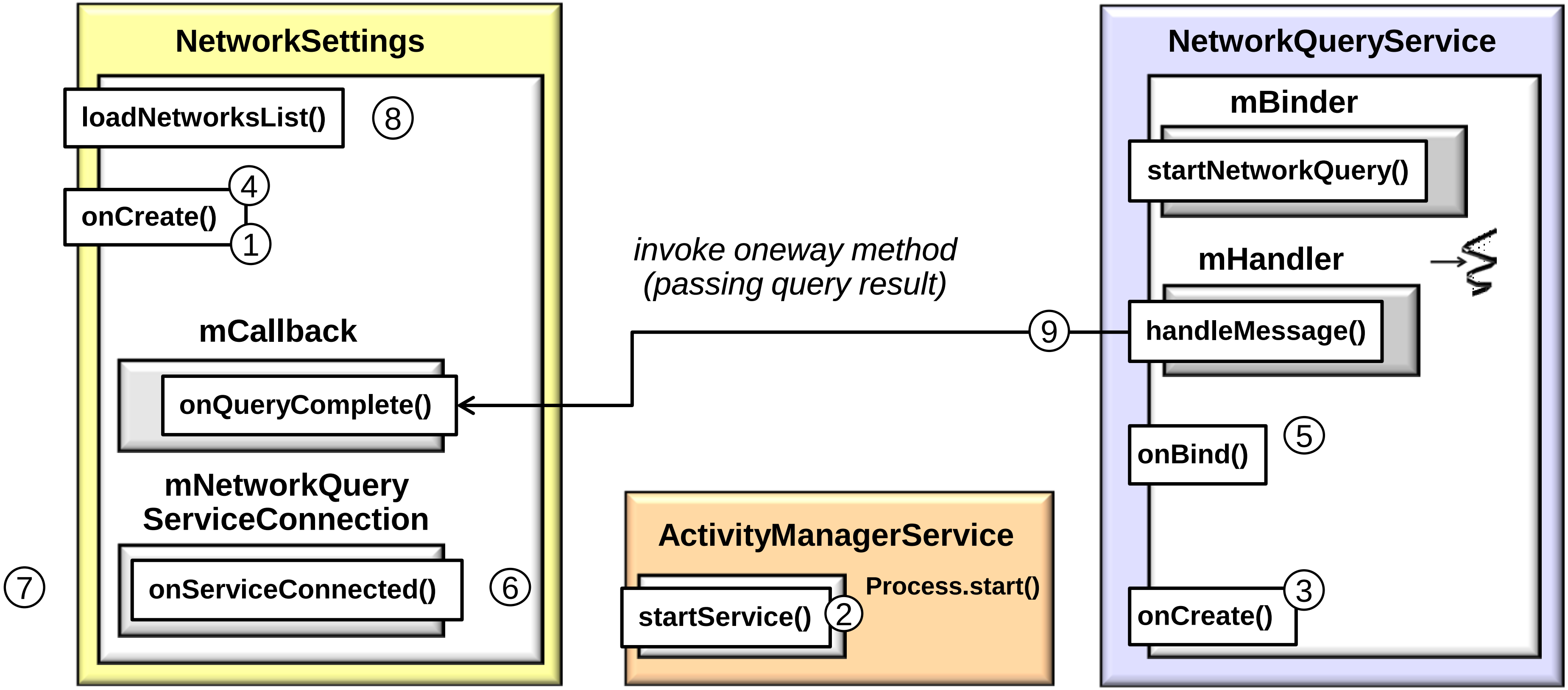


packages/apps/Phone/src/com/android/phone/NetworkQueryService.java has source

# Broker POSA1 Architectural Pattern

## Applying the Broker pattern in Android

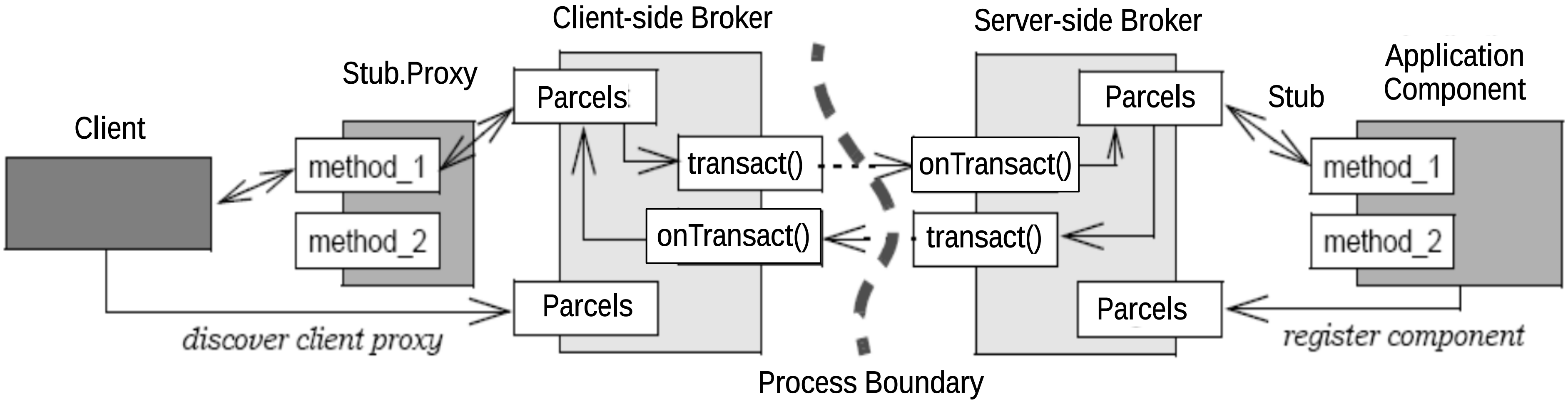
- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability



packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

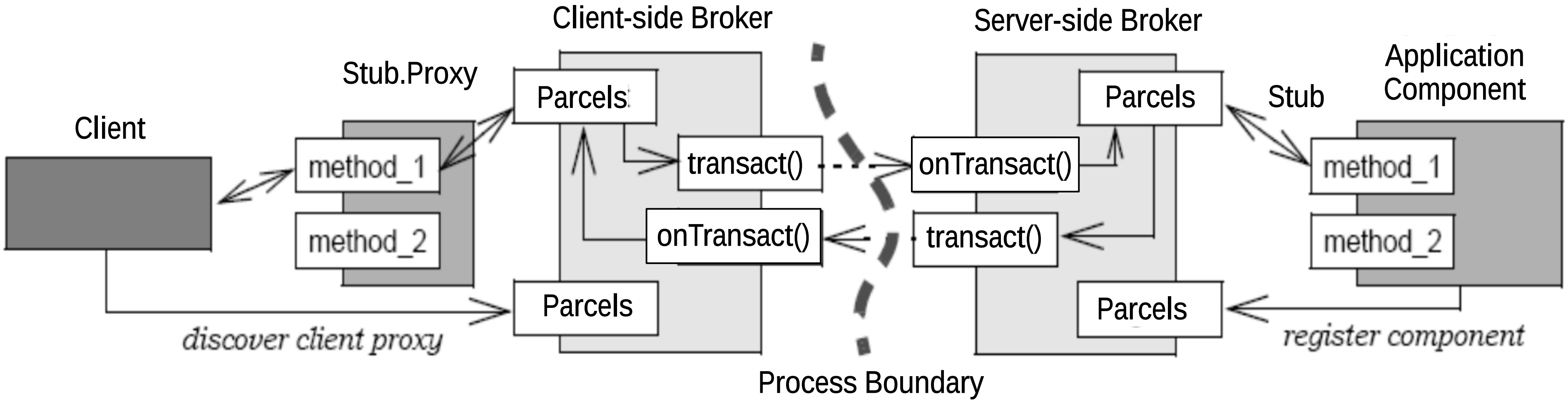
# Summary

- Android Bound Services uses *Broker* to invoke methods across processes

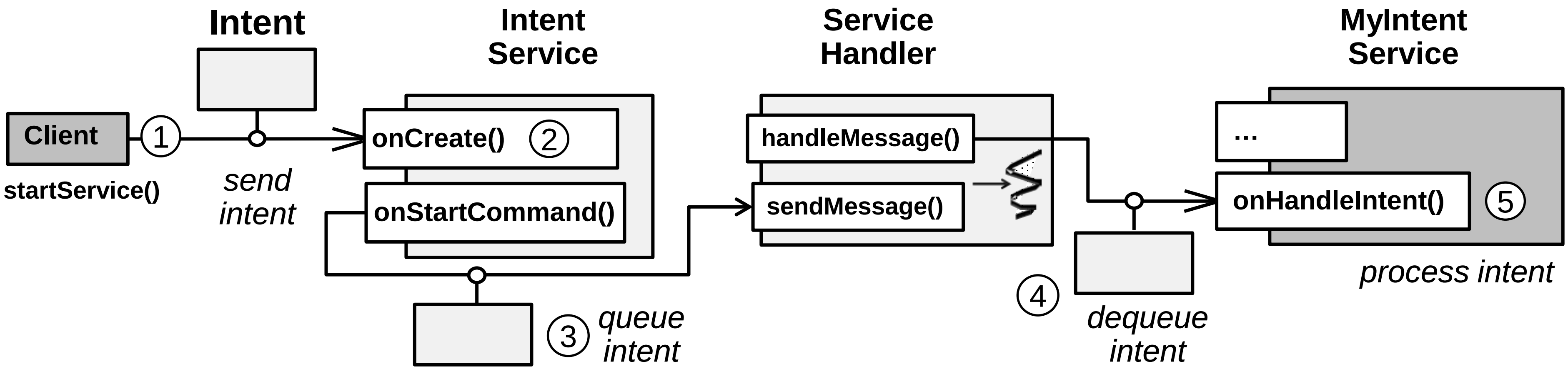


# Summary

- Android Bound Services uses *Broker* to invoke methods across processes



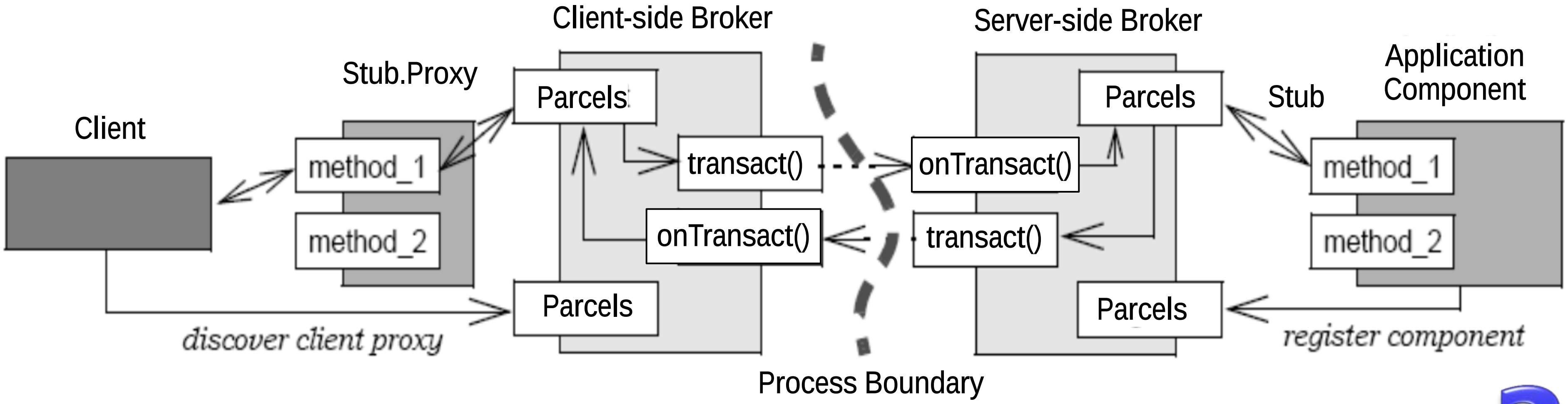
- Android Started Services use *Command Processor* to pass messages



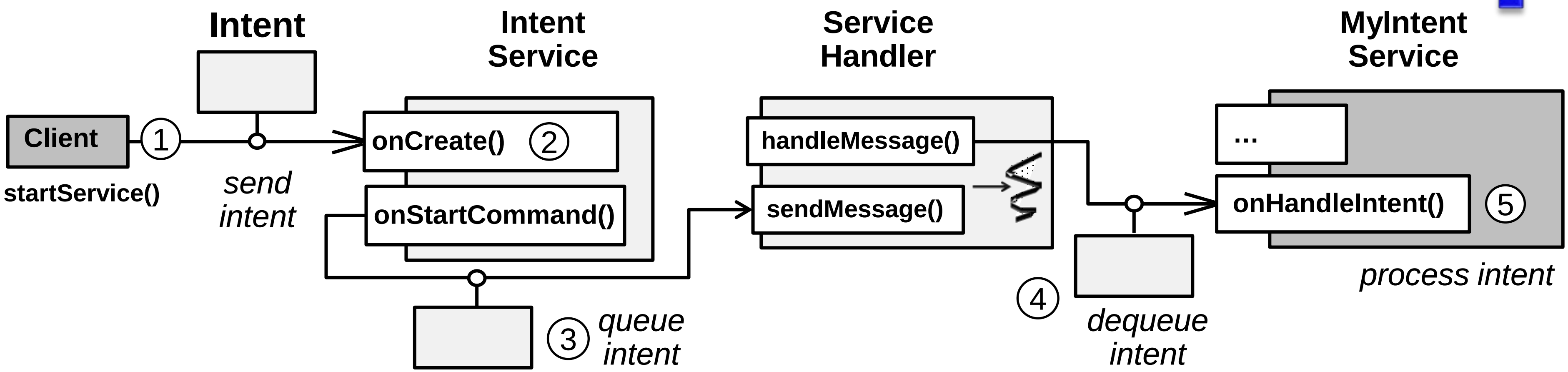
*Command Processor* & *Broker* are “pattern complements”

# Summary

- Android Bound Services uses *Broker* to invoke methods across processes



- Android Started Services use *Command Processor* to pass messages



- Software architects must understand the trade-offs between these patterns

# Android Services & Local IPC: The Publisher/Subscriber Pattern (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

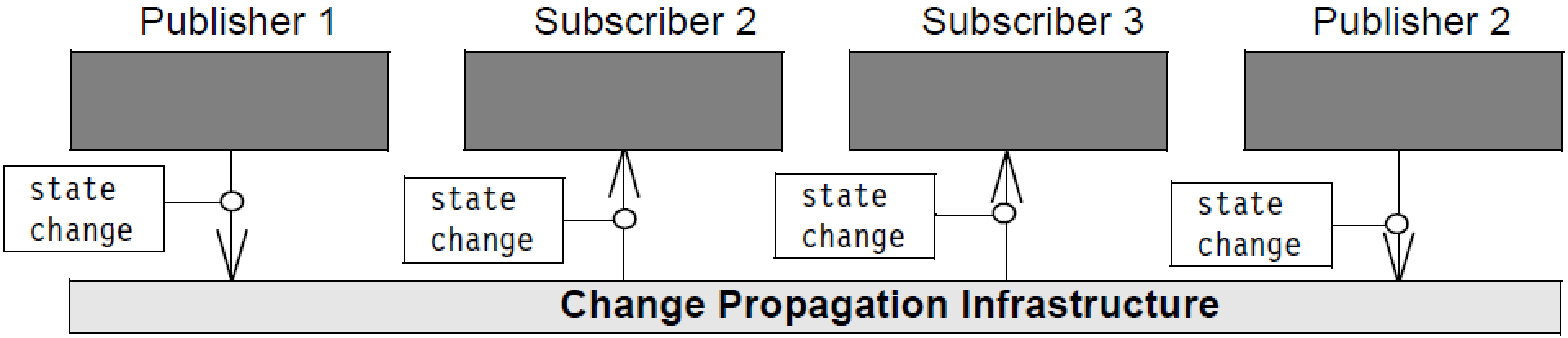
Professor of Computer Science

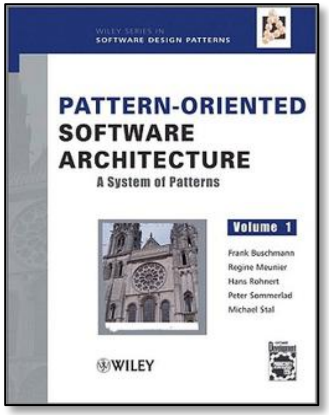
Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Module

- Understand the *Publisher/Subscriber* pattern



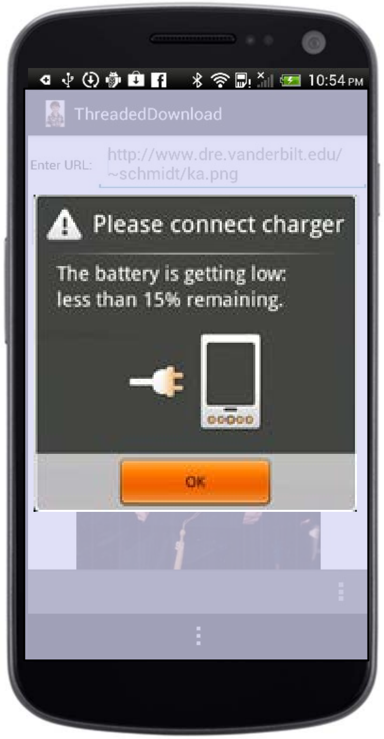


en.wikipedia.org/wiki/Publish-subscribe_pattern has more info

# Challenge: Managing Dependencies Efficiently

## Context

- Smartphone platforms keep track of system-related status info that is of interest to apps
  - e.g., Android tracks & report low battery status



ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

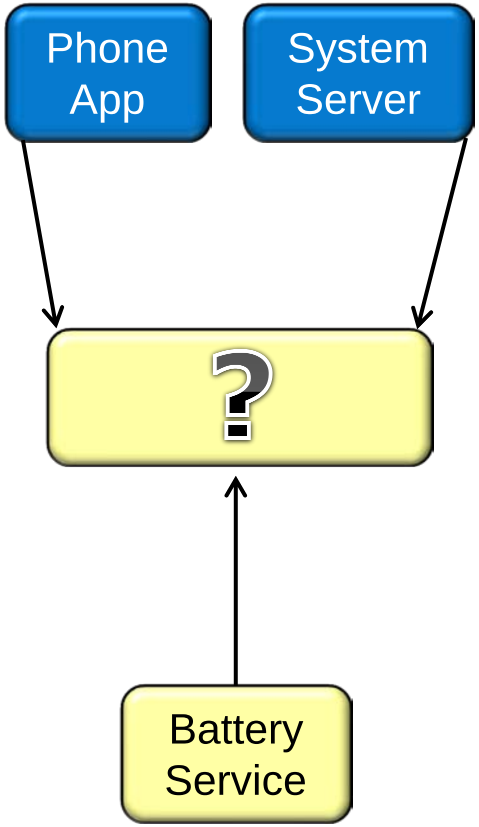
Please connect charger

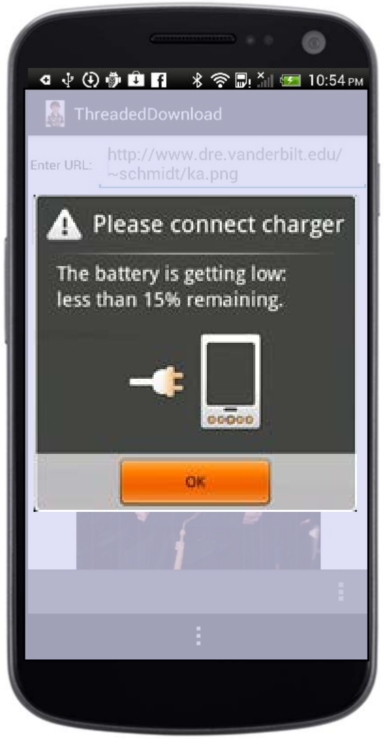
The battery is getting low: less than 15% remaining.

# Challenge: Managing Dependencies Efficiently

## Problems

- Multiple apps/services may be interested in system status info
  - Coupling status info w/app presentation violates modularity





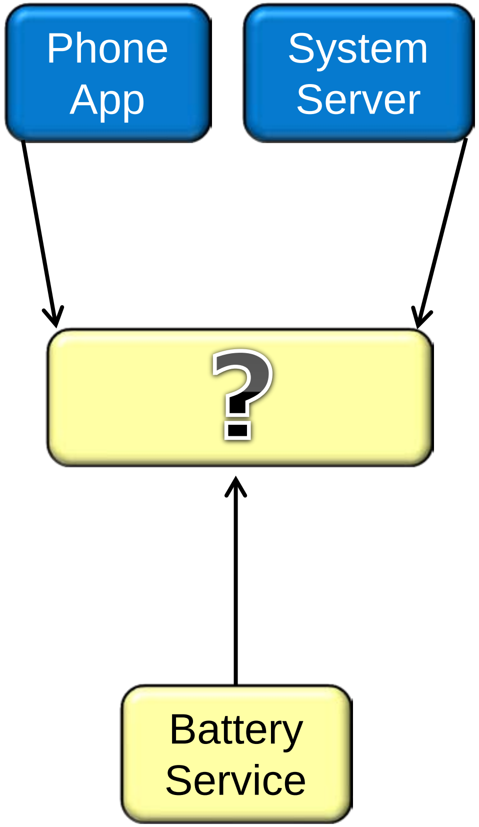
Please connect charger

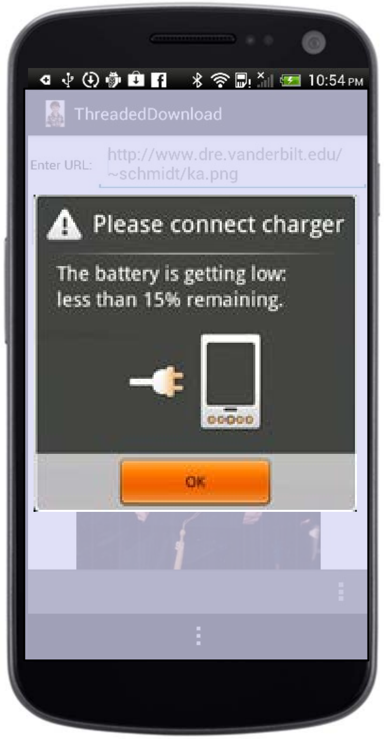
The battery is getting low:
less than 15% remaining.

# Challenge: Managing Dependencies Efficiently

**Problems**

- Multiple apps/services may be interested in system status info
  - Coupling status info w/app presentation violates modularity
  - Apps polling for changes to status information is inefficient




10:54 PM

ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

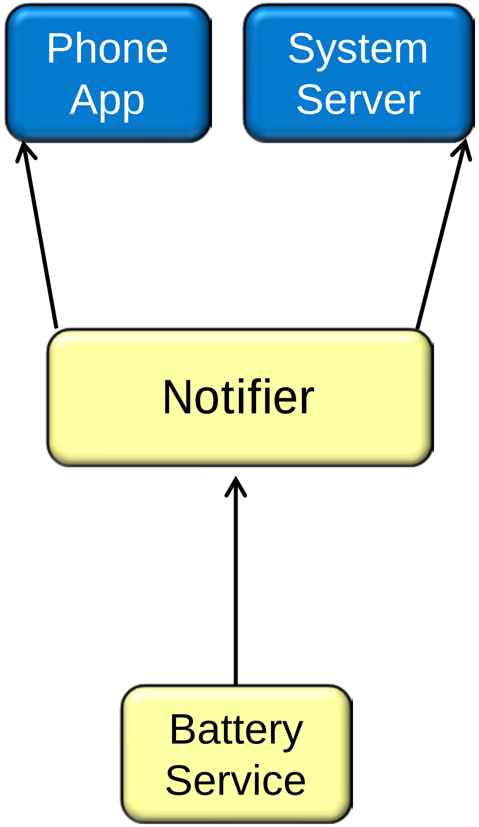
The battery is getting low: less than 15% remaining.

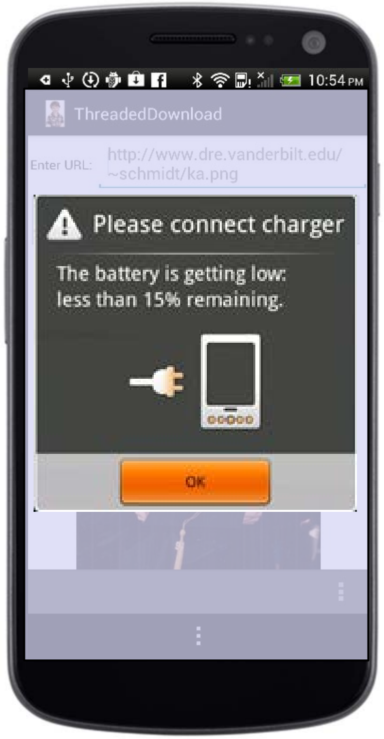
OK

# Challenge: Managing Dependencies Efficiently

**Solution**

- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes





http://www.dre.vanderbilt.edu/~schmidt/ka.png

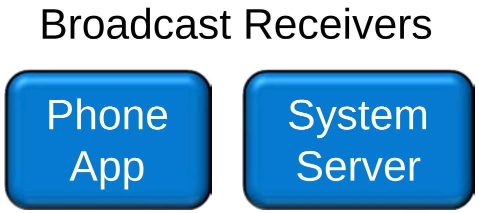
Please connect charger

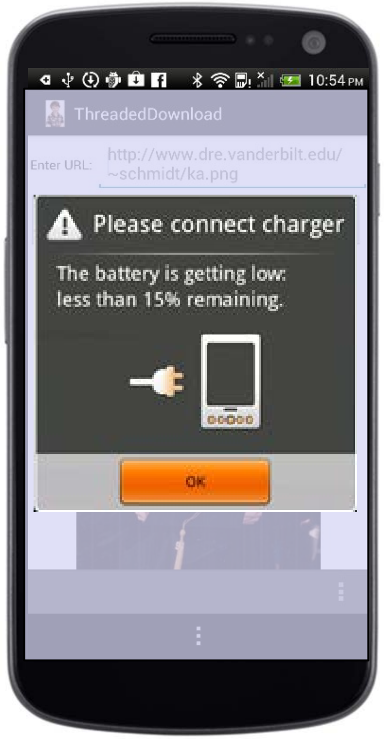
The battery is getting low: less than 15% remaining.

# Challenge: Managing Dependencies Efficiently

## Solution

- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info





http://www.dre.vanderbilt.edu/~schmidt/ka.png

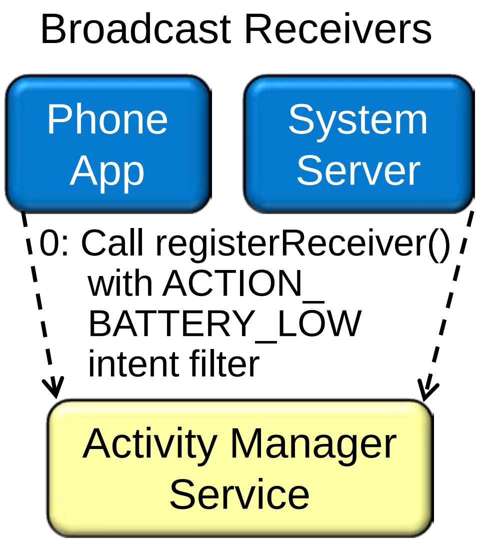
Please connect charger

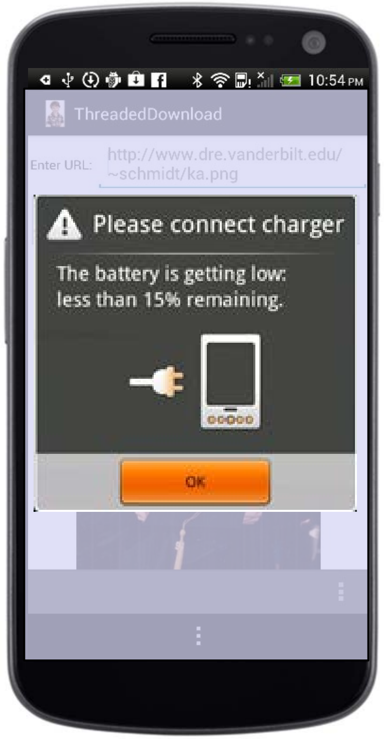
The battery is getting low: less than 15% remaining.

# Challenge: Managing Dependencies Efficiently

## Solution

- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info
  - Use registerReceiver() in an activity to attach BroadcastReceiver that's called back when intent is broadcast
    - e.g., ACTION_BATTERY_LOW


Broadcast Receivers
Phone App
System Server
0: Call registerReceiver() with ACTION_ BATTERY_LOW intent filter
Activity Manager Service


10:54 PM

ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

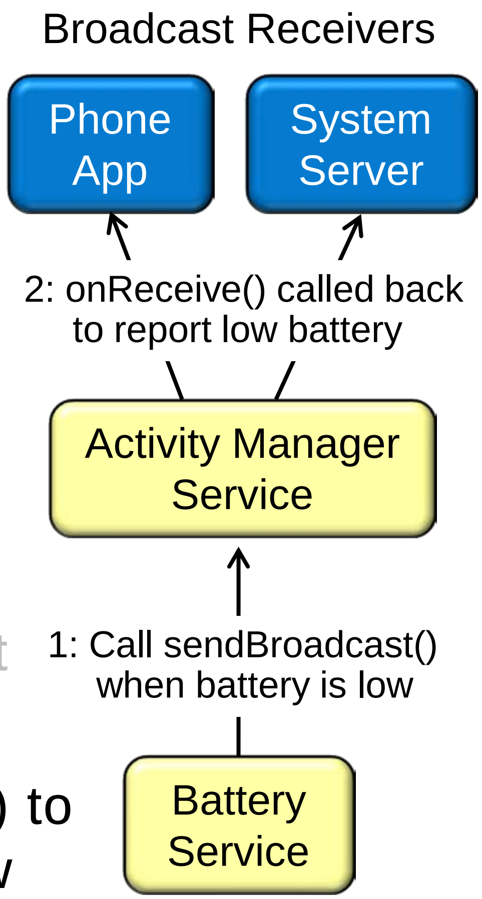
The battery is getting low: less than 15% remaining.

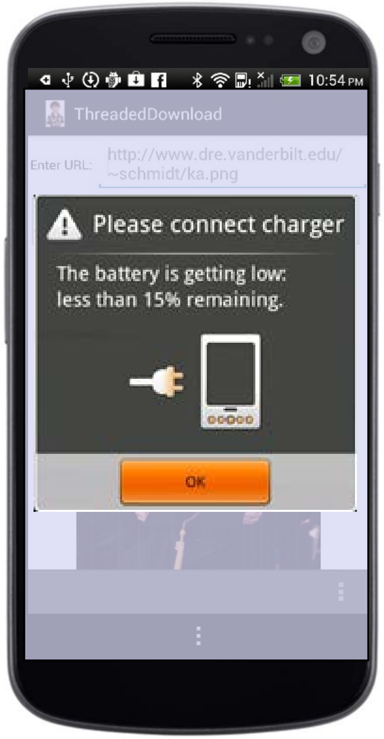
OK

# Challenge: Managing Dependencies Efficiently

## Solution

- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info
  - Use registerReceiver() in an activity to attach BroadcastReceiver that's called back when intent is broadcast
    - e.g., ACTION_BATTERY_LOW
  - BatteryService calls sendBroadcast() to tell BroadcastReceivers battery's low


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service



http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

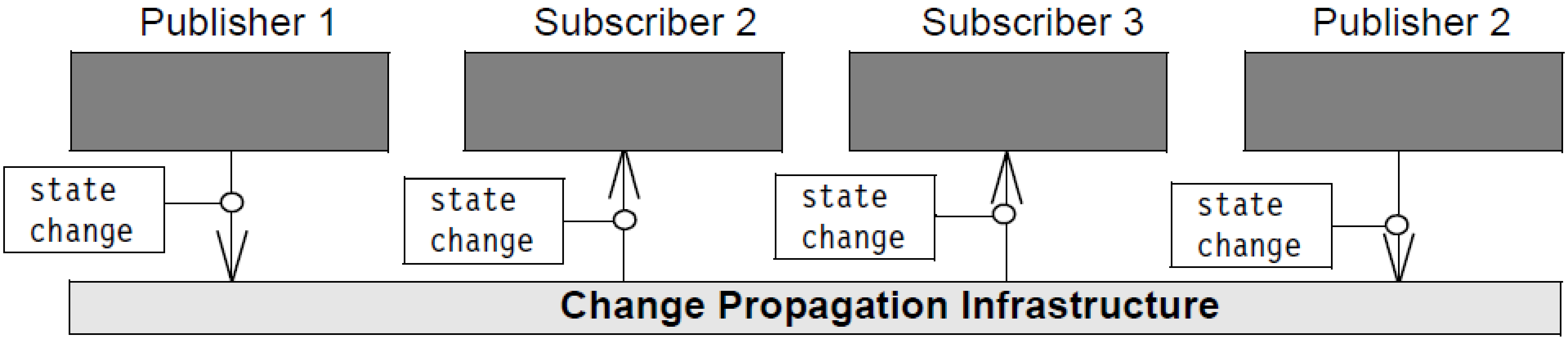
The battery is getting low:
less than 15% remaining.

Android also uses the *Proxy*, *Broker, & Activator* patterns in this scenario

# Publisher-Subscriber POSA1 Architectural

**Intent**

Notify event handlers (Subscribers or Observers) when some interesting object (Publisher or Observable) changes state

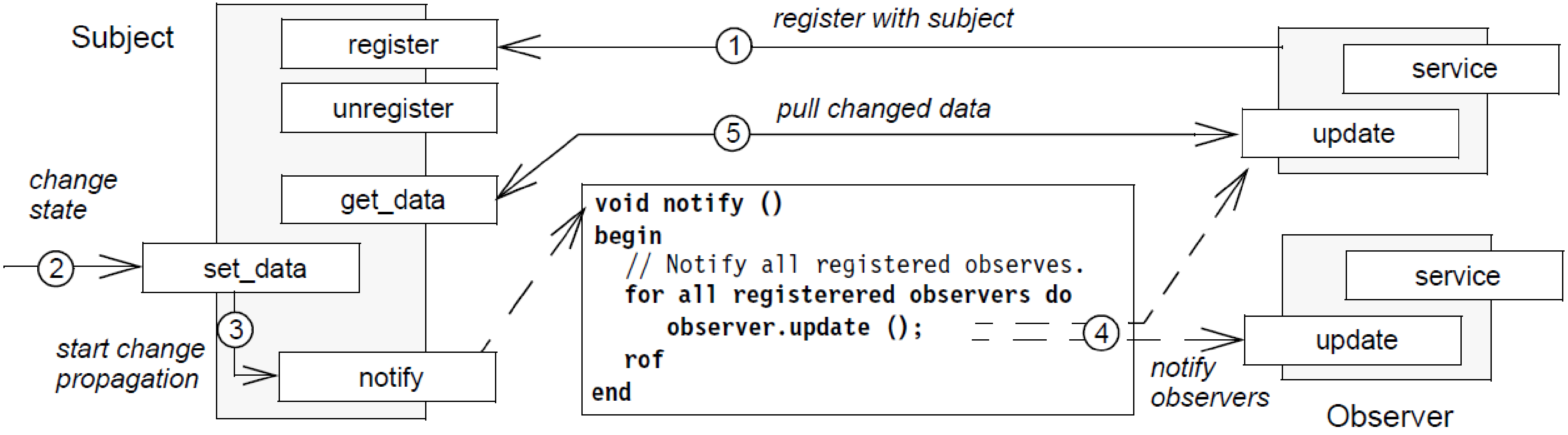


# Publisher-Subscriber POSA1 Architectural

**Intent**

GoF contains similar *Observer* pattern

Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated



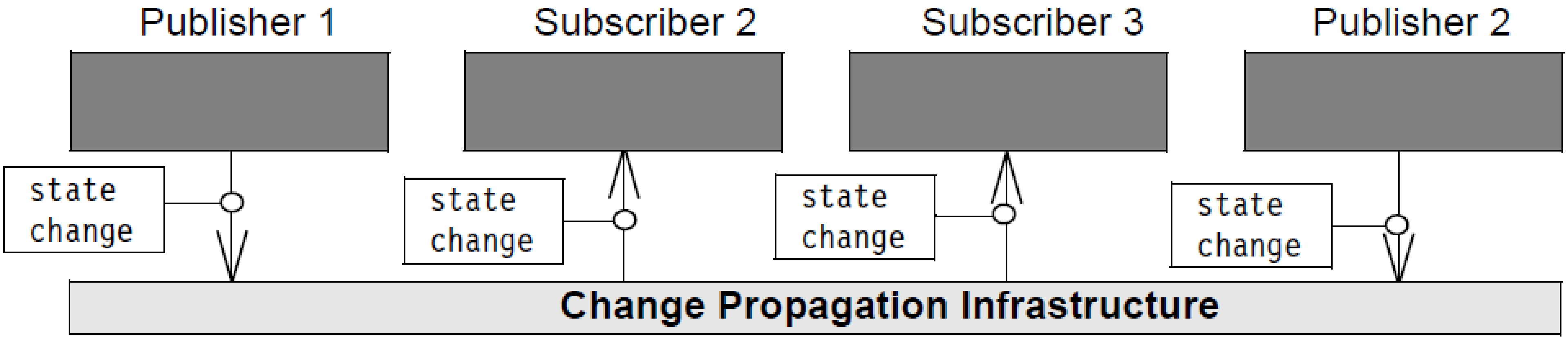
```
void notify ()
begin
  // Notify all registered observes.
  for all registerered observers do
     observer.update ();
  rof
end
```

See en.wikipedia.org/wiki/Observer_pattern for more on *Observer* pattern

# Publisher-Subscriber POSA1 Architectural

## Applicability

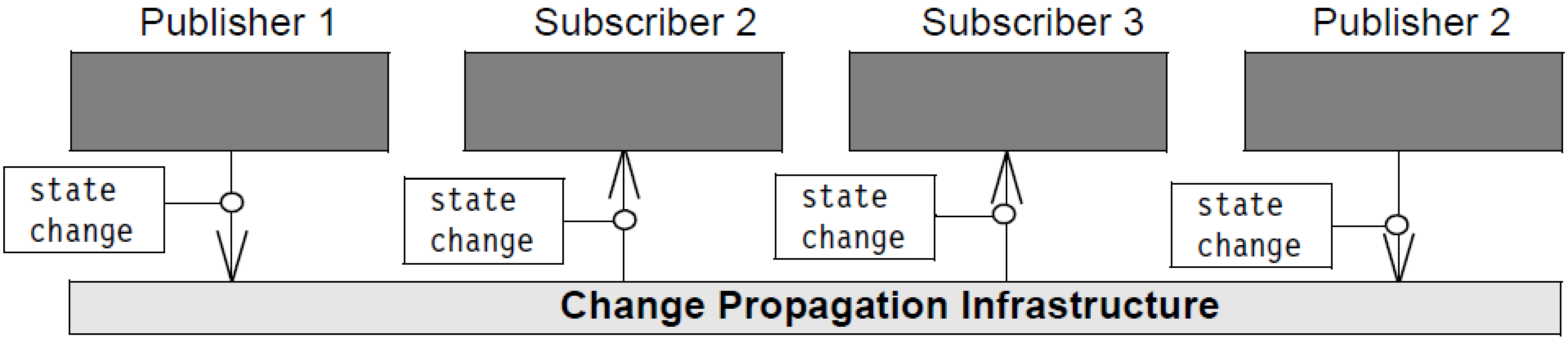
- An abstraction has two aspects, one dependent on the other



# Publisher-Subscriber POSA1 Architectural

**Applicability**

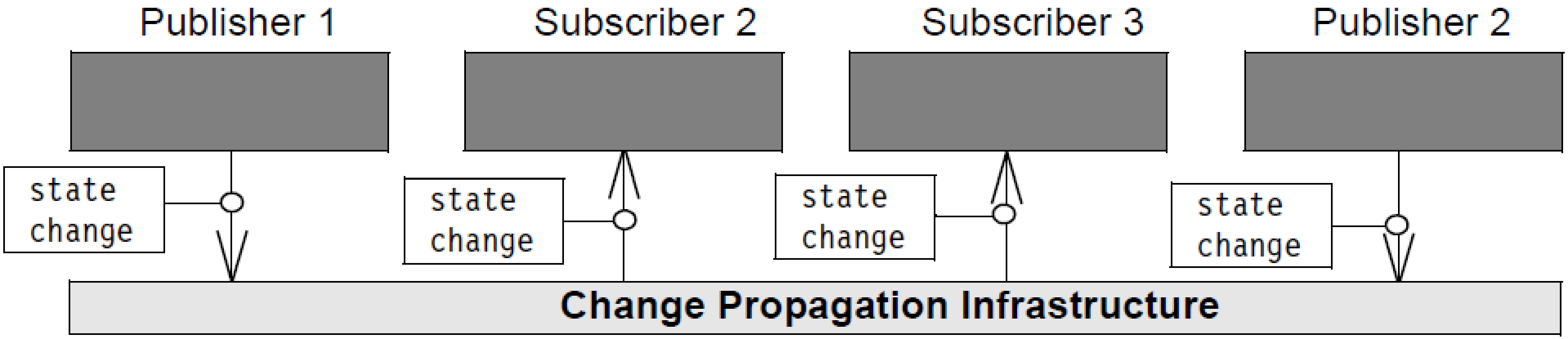
- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others



# Publisher-Subscriber POSA1 Architectural

## Applicability

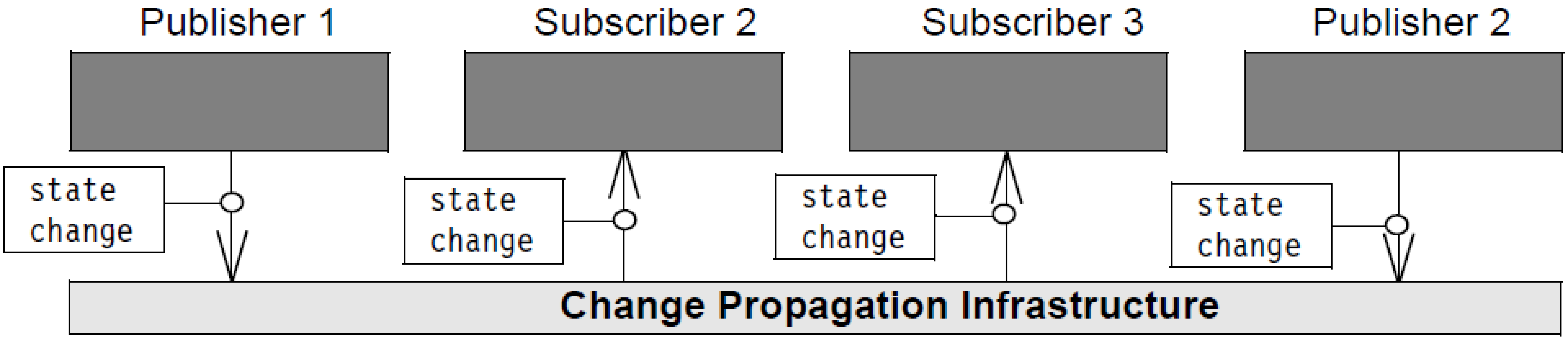
- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others
- An object should notify an unknown number of other objects



# Publisher-Subscriber POSA1 Architectural

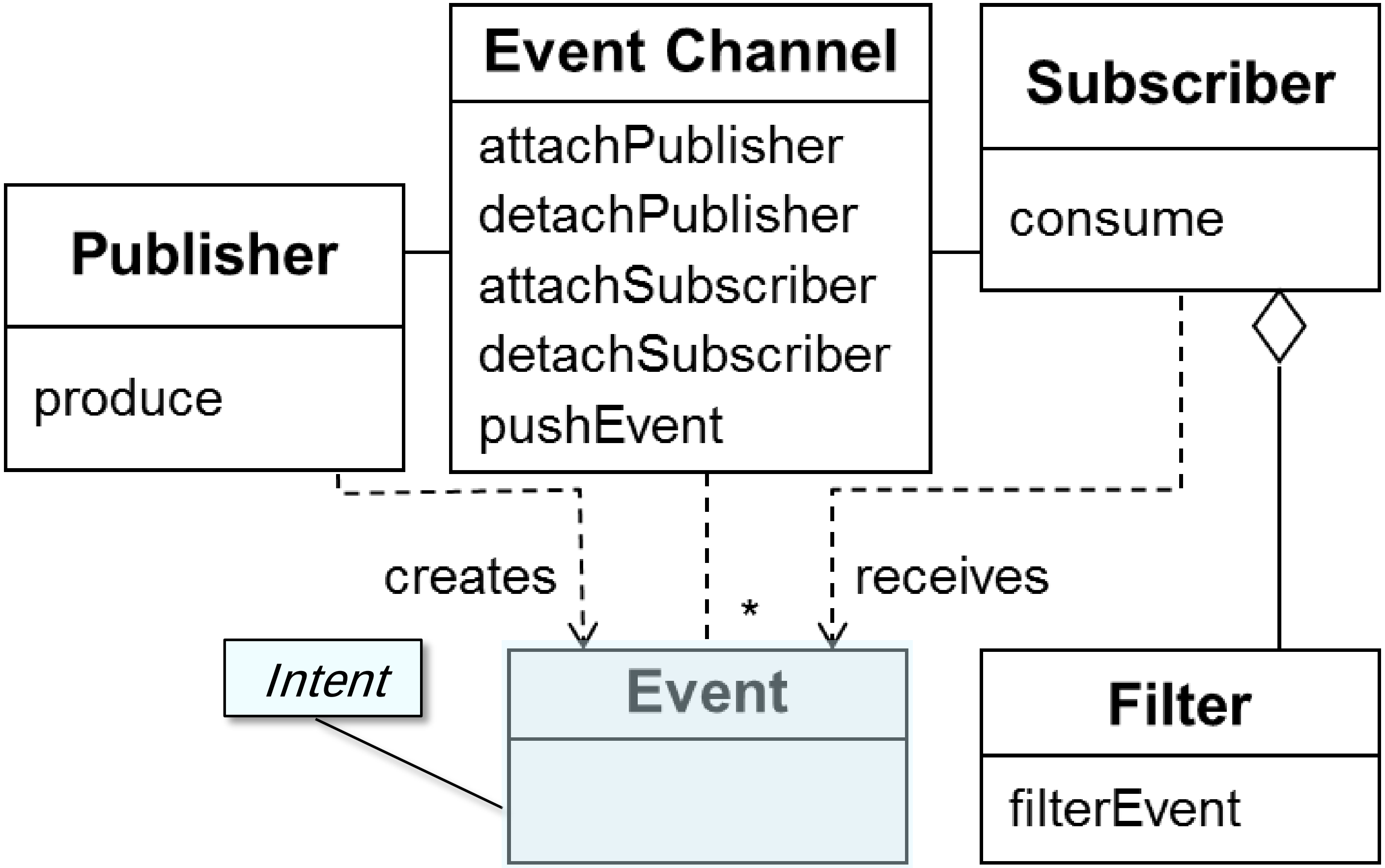
## Applicability

- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others
- An object should notify an unknown number of other objects
- Not every objects is always interested in receiving notifications when an object changes state



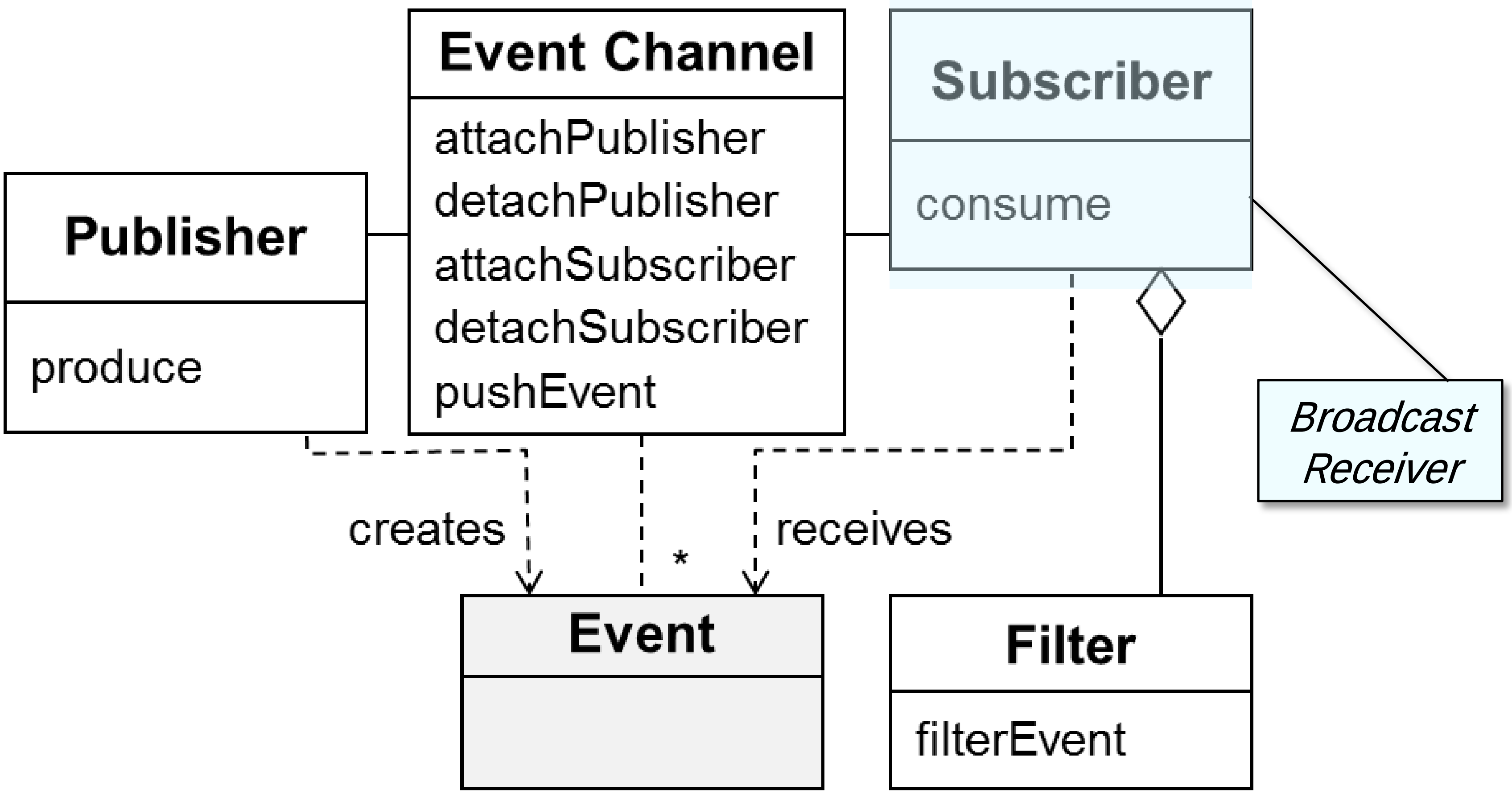
# Publisher-Subscriber POSA1 Architectural

## Structure & Participants



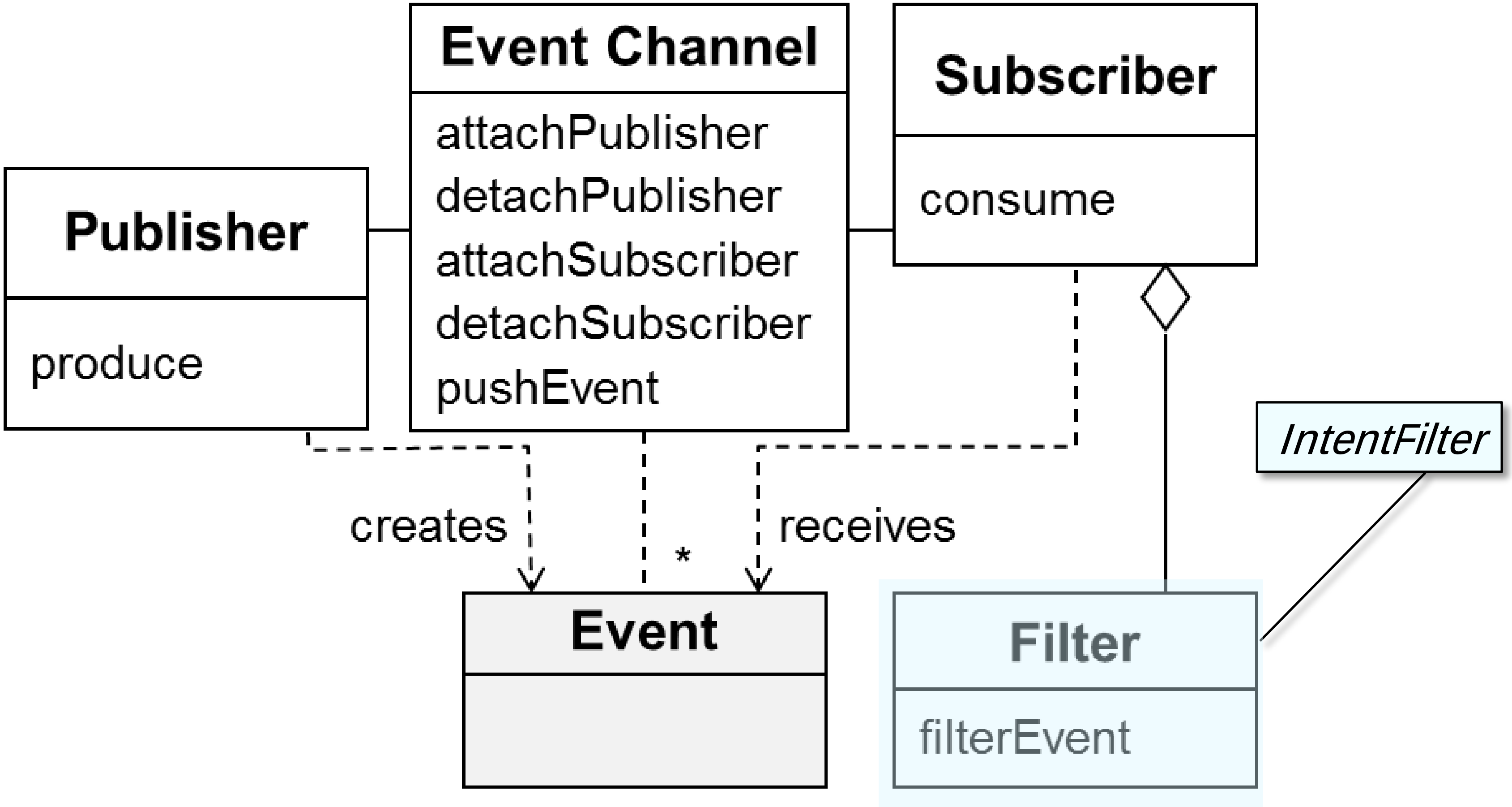
# Publisher-Subscriber POSA1 Architectural

## Structure & Participants



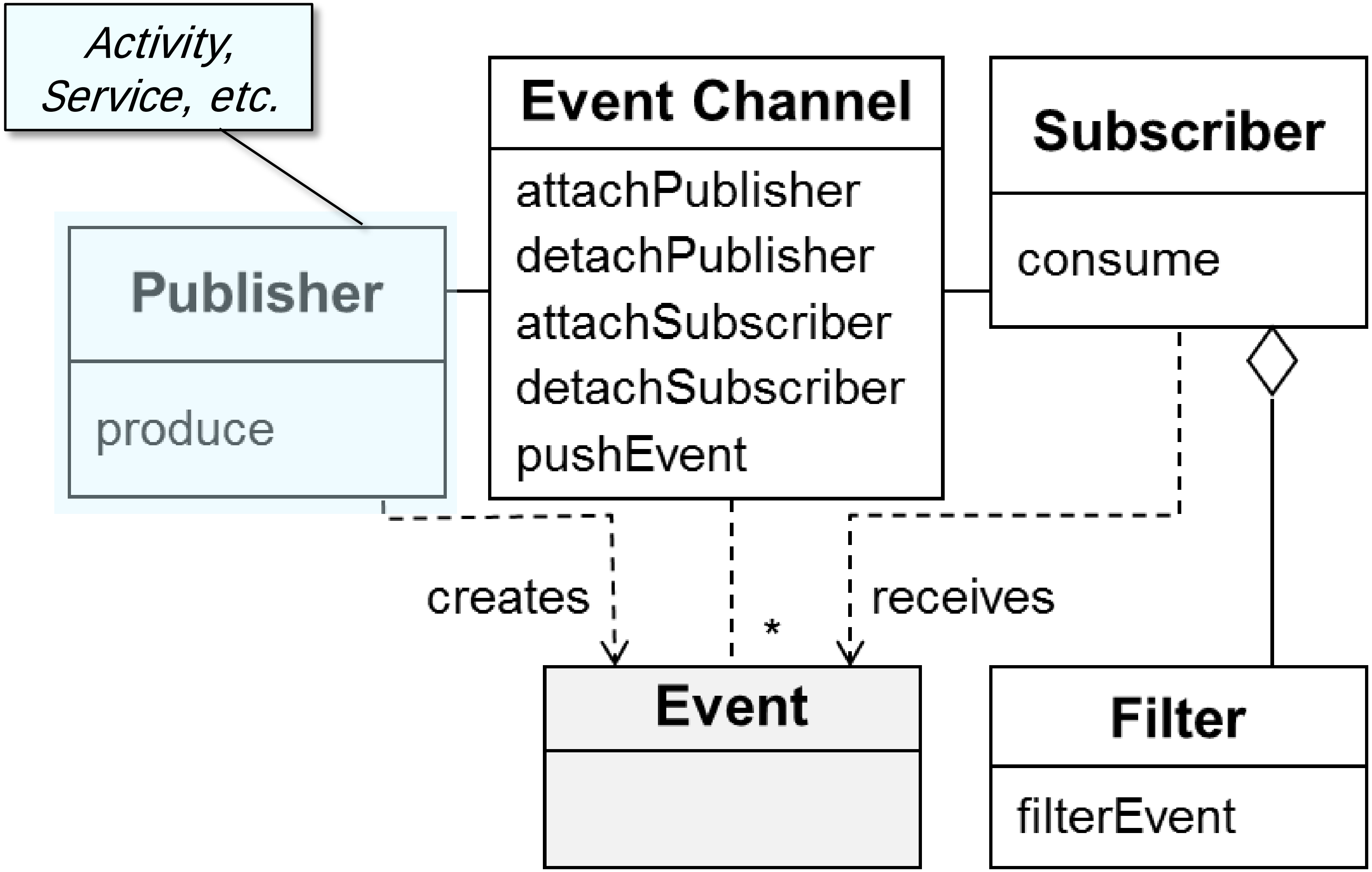
# Publisher-Subscriber POSA1 Architectural

## Structure & Participants



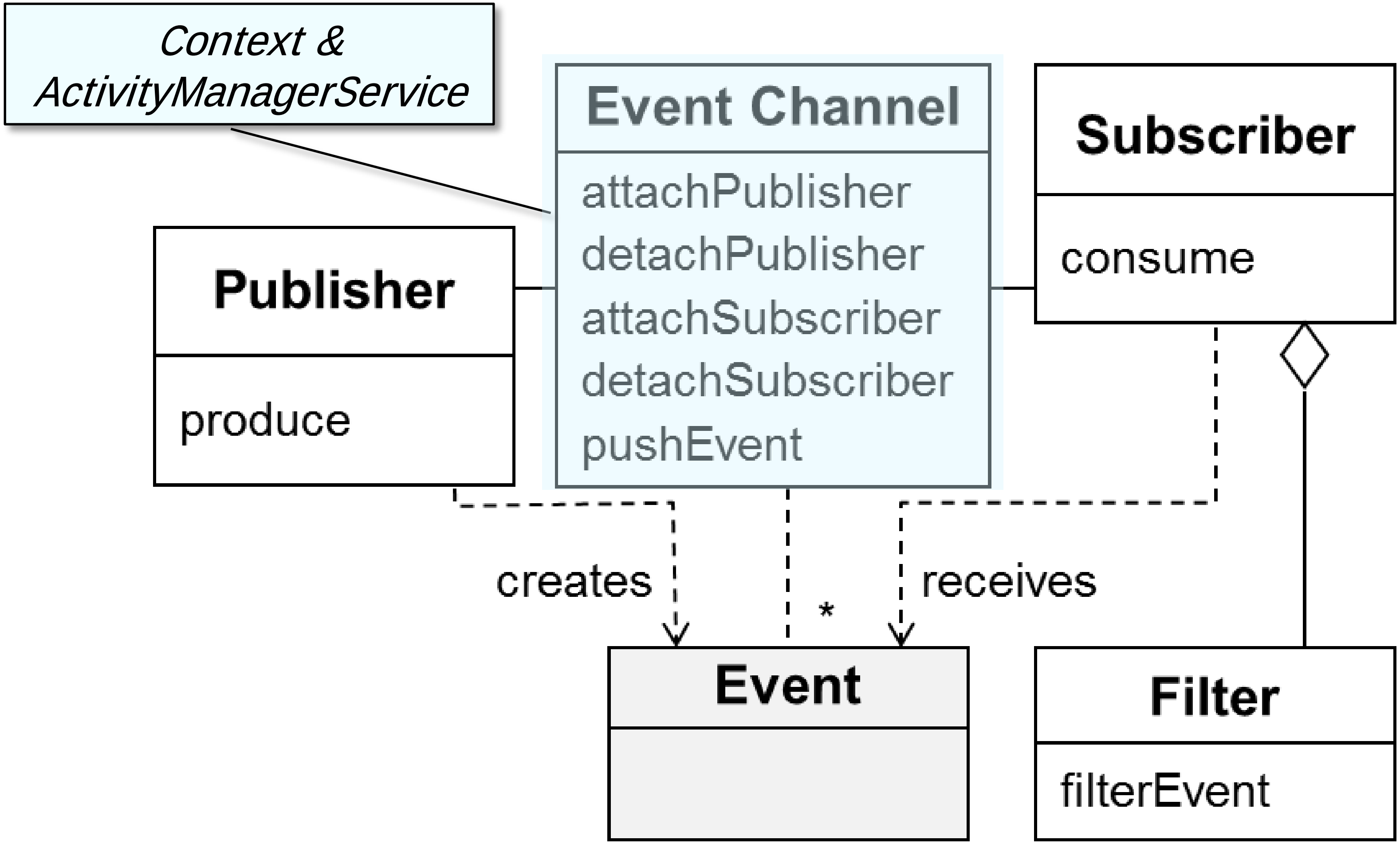
# Publisher-Subscriber POSA1 Architectural

## Structure & Participants



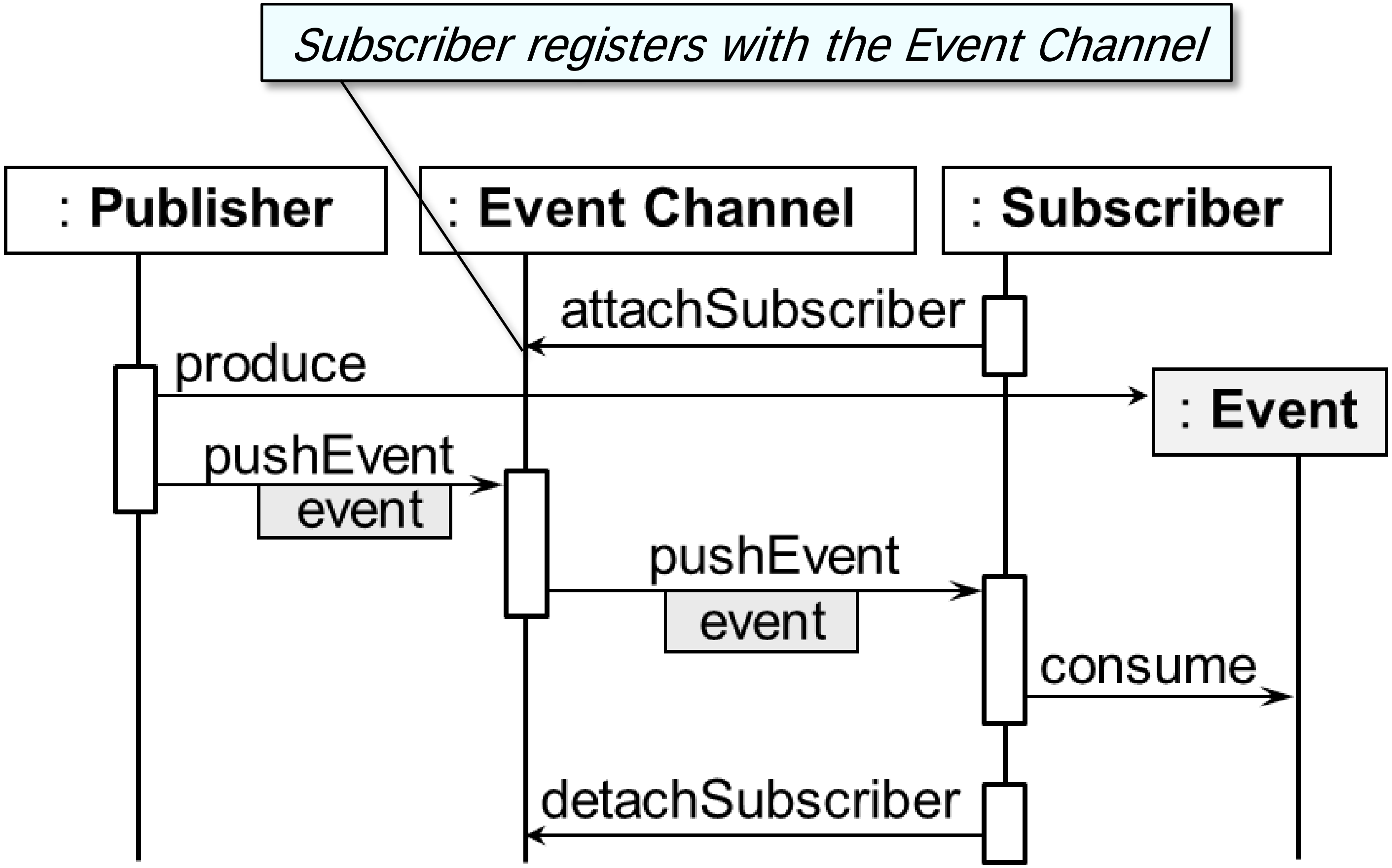
# Publisher-Subscriber POSA1 Architectural

## Structure & Participants



# Publisher-Subscriber POSA1 Architectural

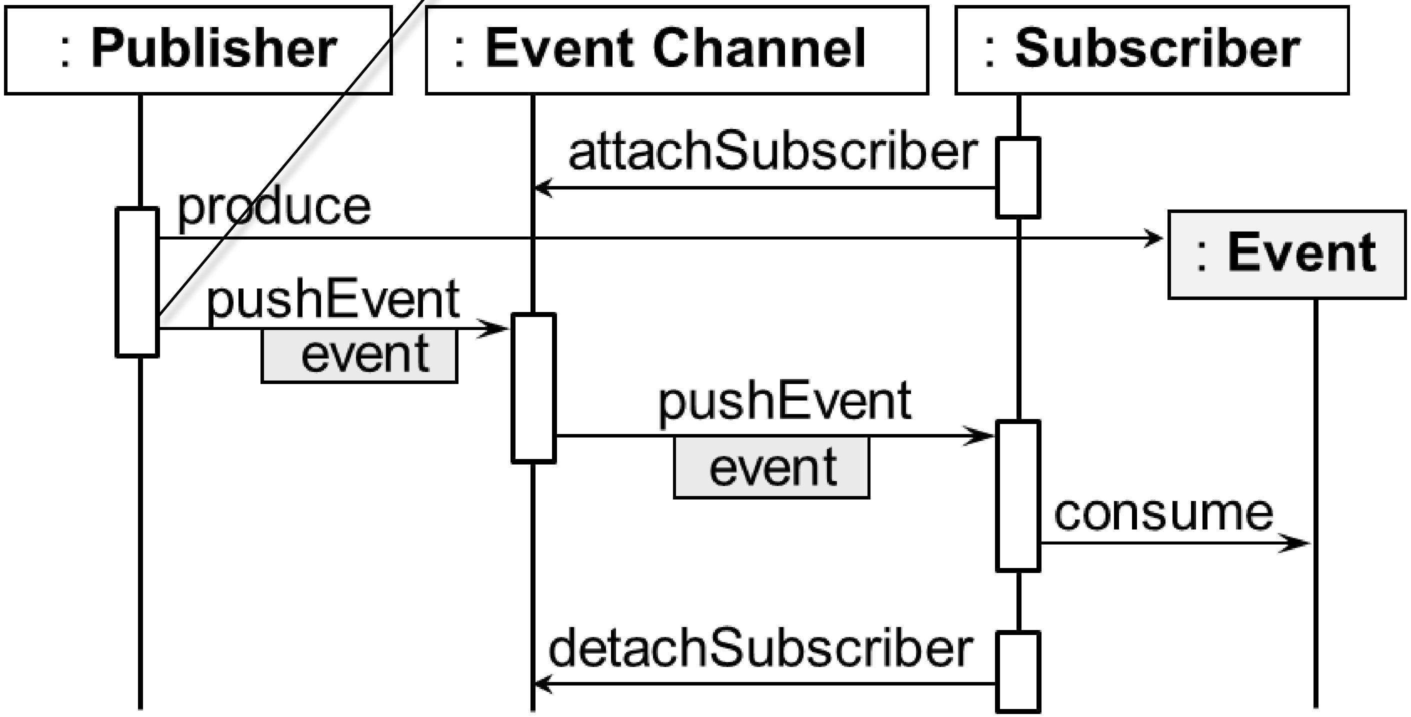
## Dynamics



# Publisher-Subscriber POSA1 Architectural

## Dynamics

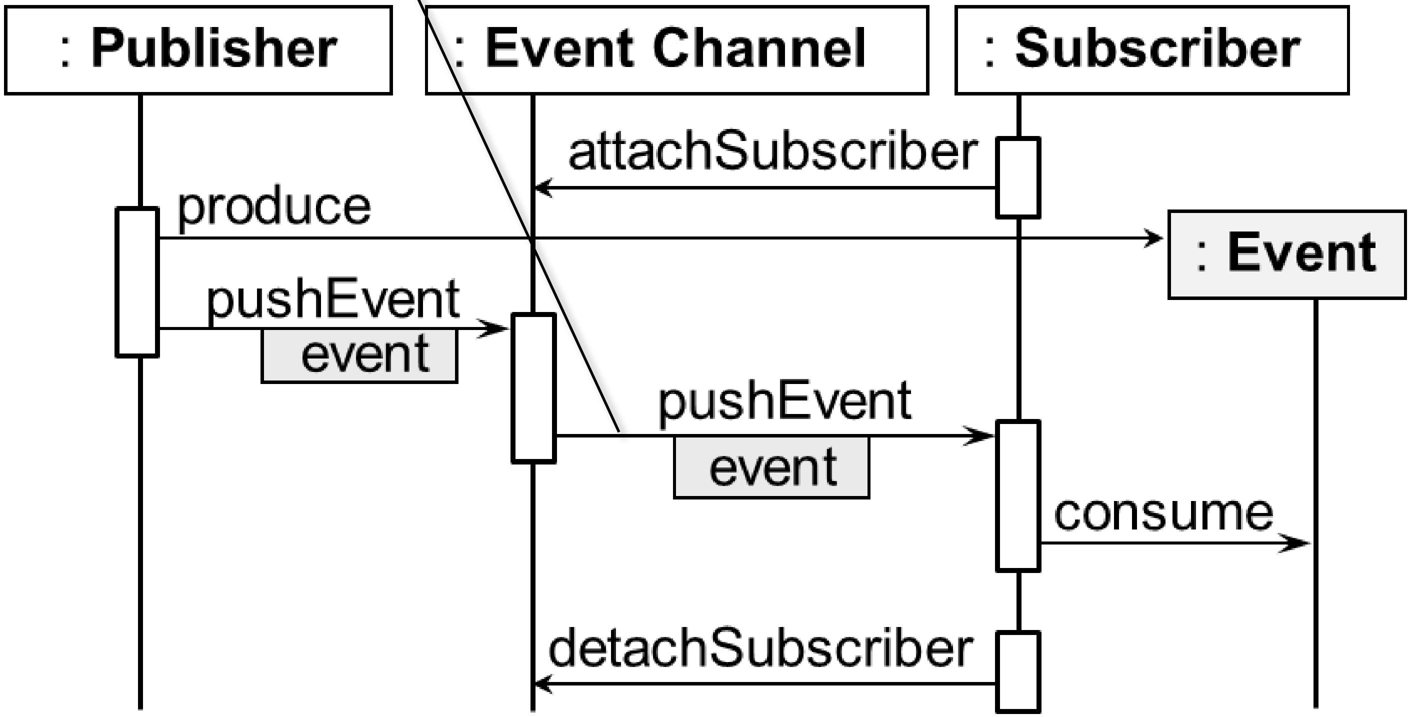
Notify EventChannel when changes occur



# Publisher-Subscriber POSA1 Architectural

## Dynamics

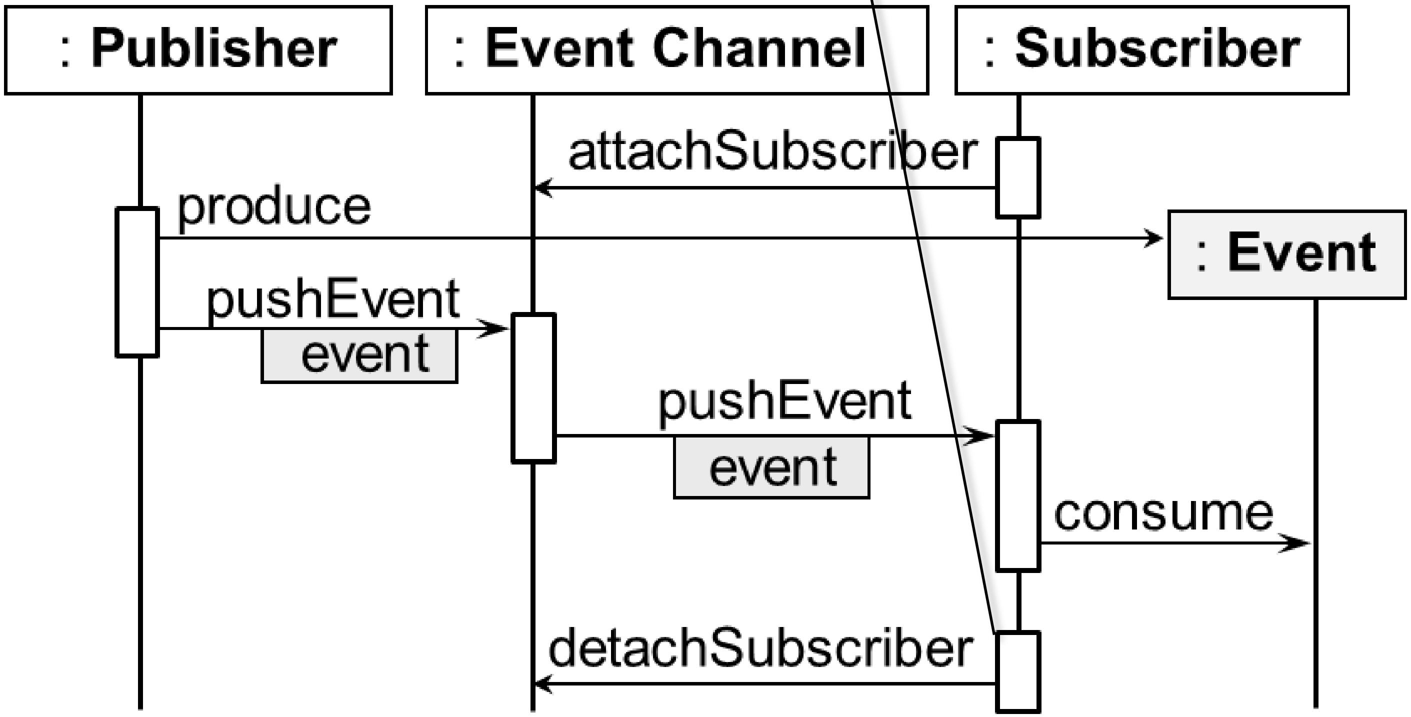
EventChannel propagates event to all registered Subscribers



# Publisher-Subscriber POSA1 Architectural

## Dynamics

*Subscribers can detach when they aren't interested in being notified any more*

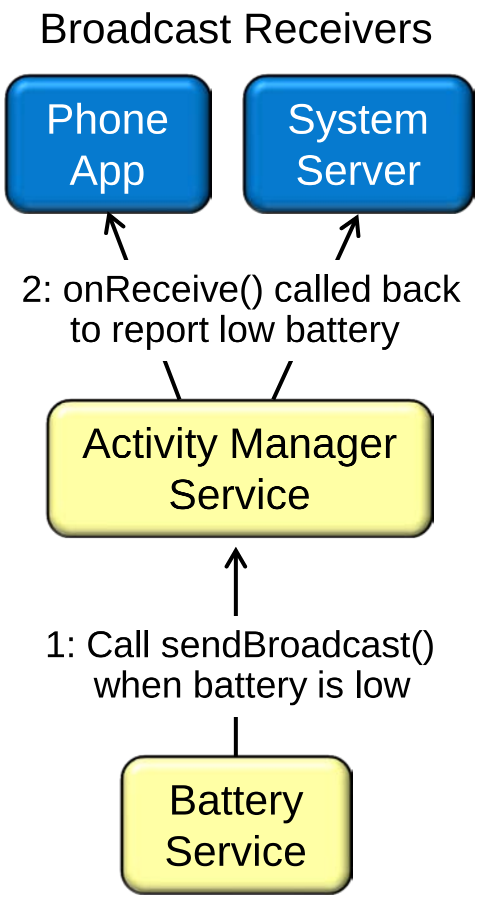


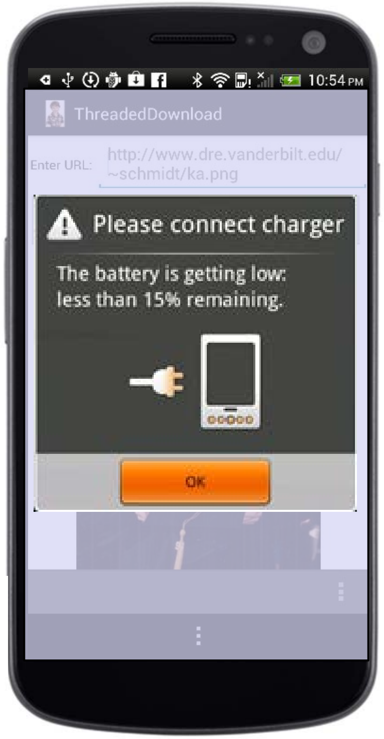
# Publisher-Subscriber POSA1 Architectural

## Consequences

+ Modularity

- Publishers & subscribers may vary independently


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service



10:54 PM

ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

The battery is getting low: less than 15% remaining.

OK

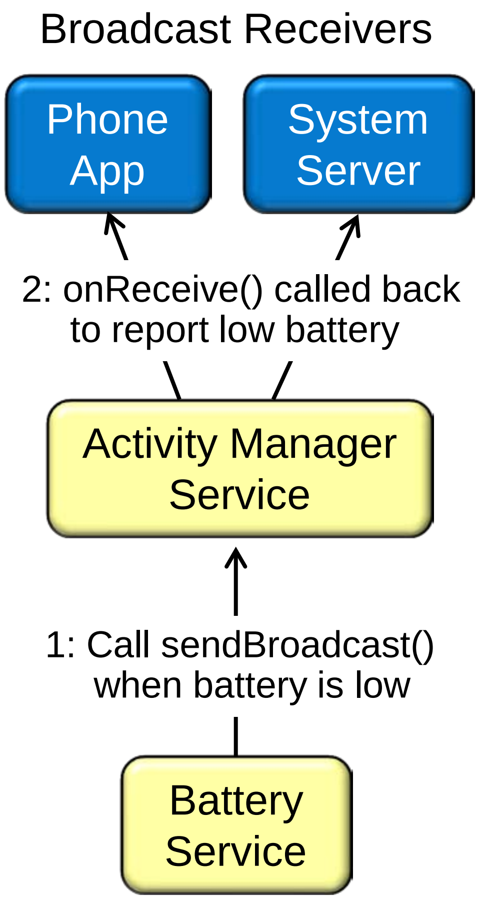
# Publisher-Subscriber POSA1 Architectural

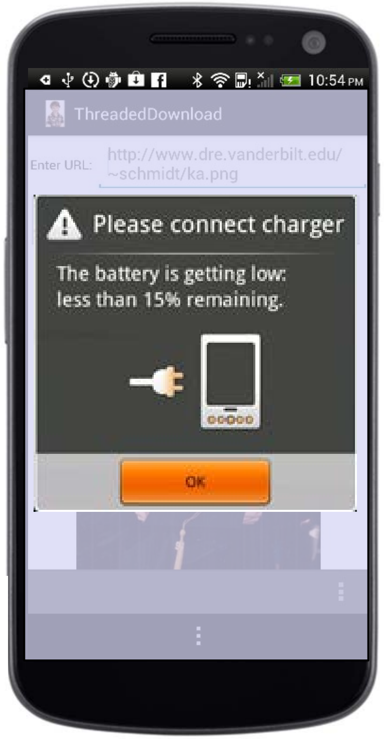
## Consequences

+ Modularity

+ Extensibility

- Can define/add any number of subscribers


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service


10:54 PM

ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

The battery is getting low: less than 15% remaining.

OK

# Publisher-Subscriber POSA1 Architectural

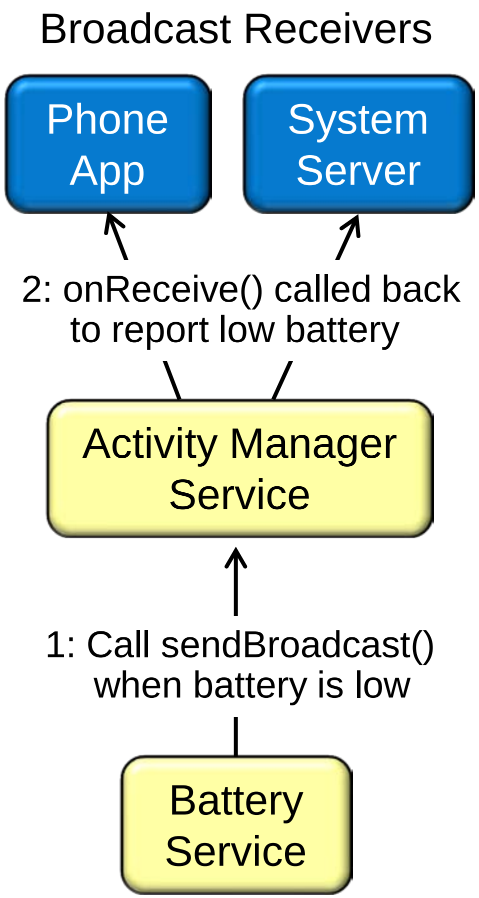
## Consequences

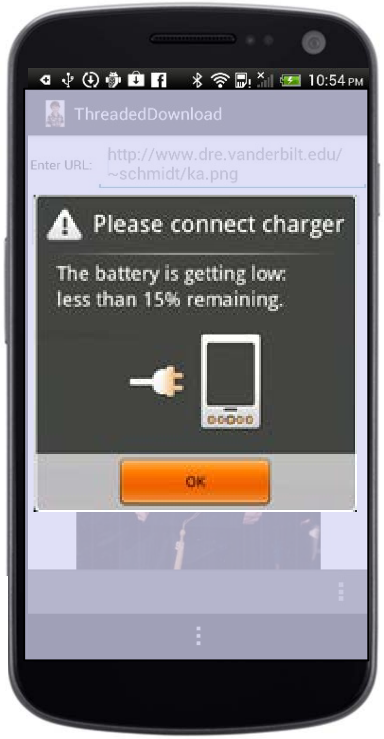
+ Modularity

+ Extensibility

+ Customizability

- Different subscribers offer different views of subject


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service



Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

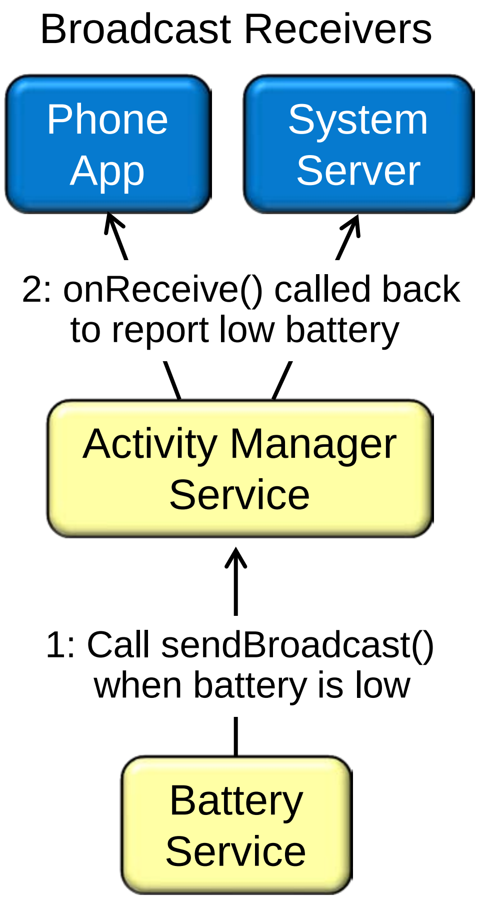
The battery is getting low: less than 15% remaining.

# Publisher-Subscriber POSA1 Architectural

## Consequences

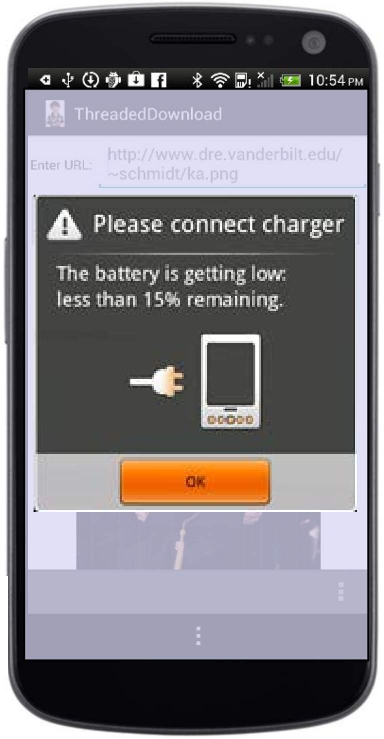
– Unexpected updates

- Subscribers don’t know about each other



Broadcast Receivers

Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service



ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

The battery is getting low: less than 15% remaining.

en.wikipedia.org/wiki/Email_storm has more info on “update storms”

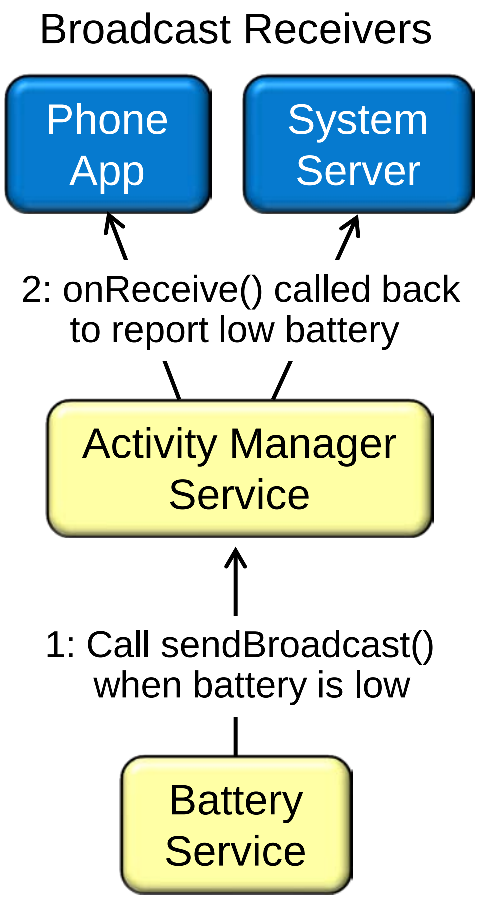
# Publisher-Subscriber POSA1 Architectural

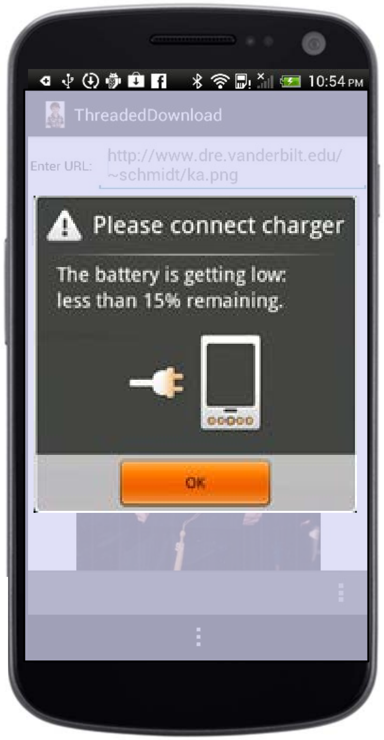
## Consequences

– Unexpected updates

– Update overhead

- Too many irrelevant updates


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service


10:54 PM

ThreadedDownload

Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

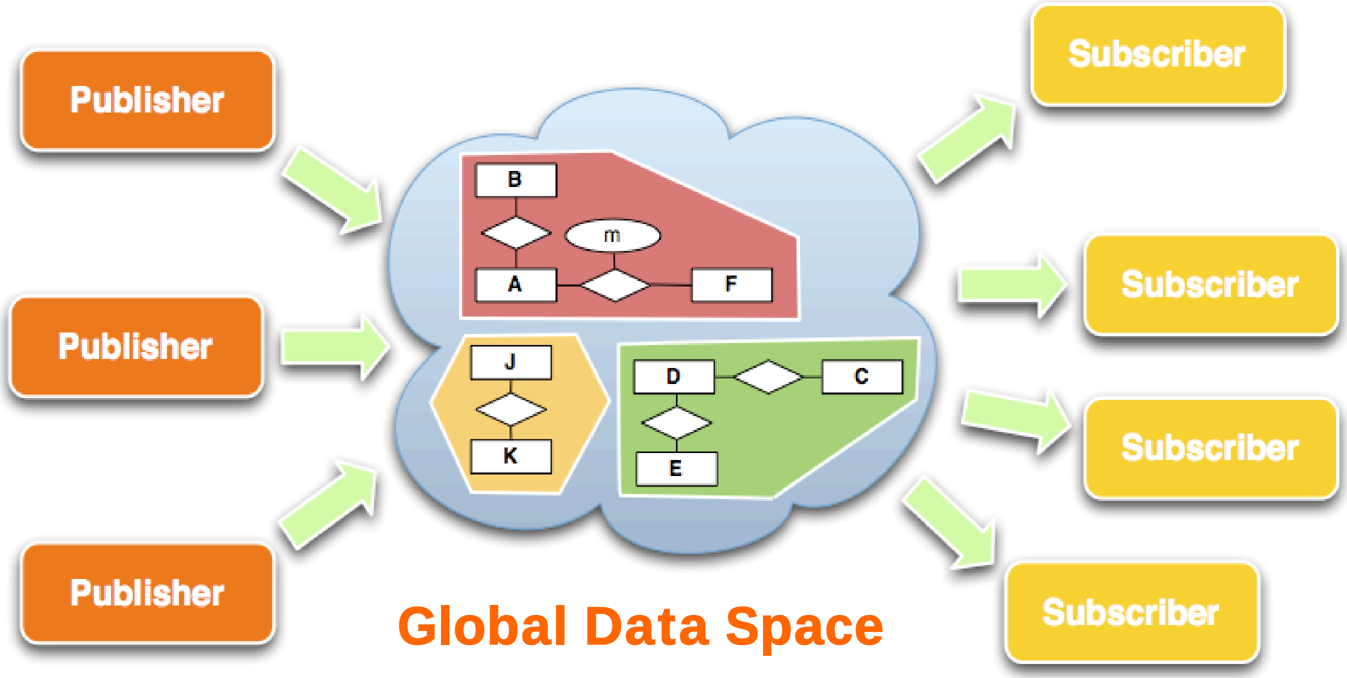
The battery is getting low: less than 15% remaining.

OK

# Publisher-Subscriber POSA1 Architectural

## Known Uses

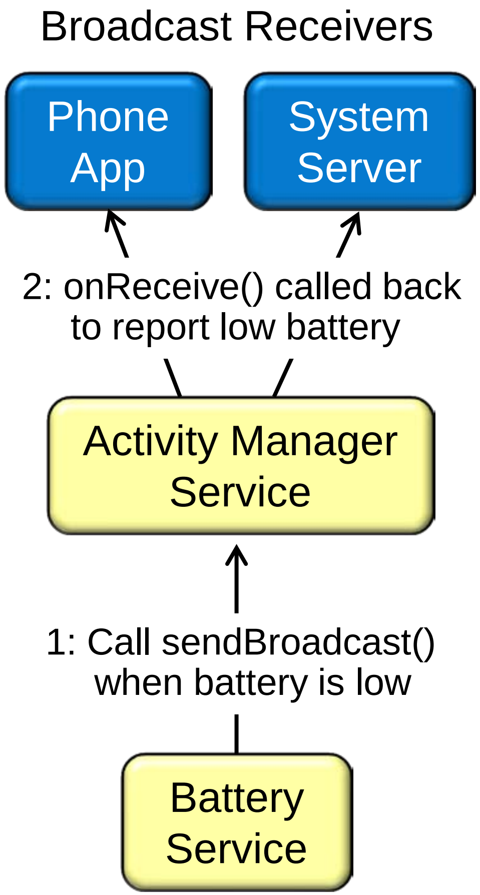
- Pub/sub middleware
  - e.g., Data Distribution Service (DDS), Java Message Service (JMS), CORBA Notification Service, Web Service Notification, etc.

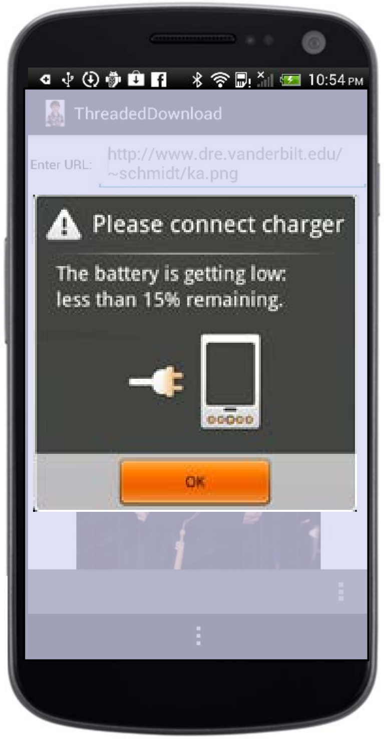


# Publisher-Subscriber POSA1 Architectural

## Known Uses

- Pub/sub middleware
- Smart phone event notification
  - e.g., Android Intents framework & Content Providers


Broadcast Receivers
Phone App
System Server
2: onReceive() called back to report low battery
Activity Manager Service
1: Call sendBroadcast() when battery is low
Battery Service


10:54 PM

ThreadedDownload

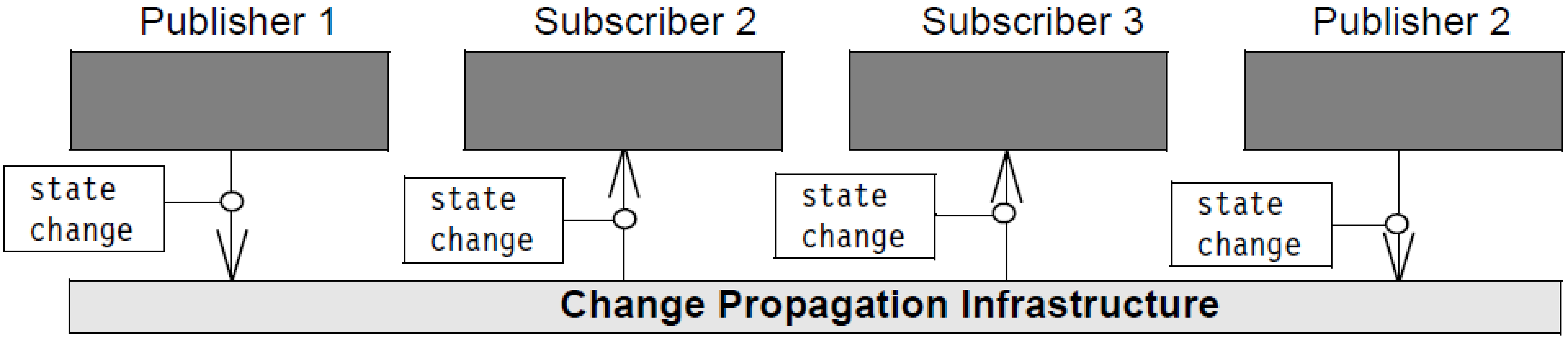
Enter URL: http://www.dre.vanderbilt.edu/~schmidt/ka.png

Please connect charger

The battery is getting low: less than 15% remaining.

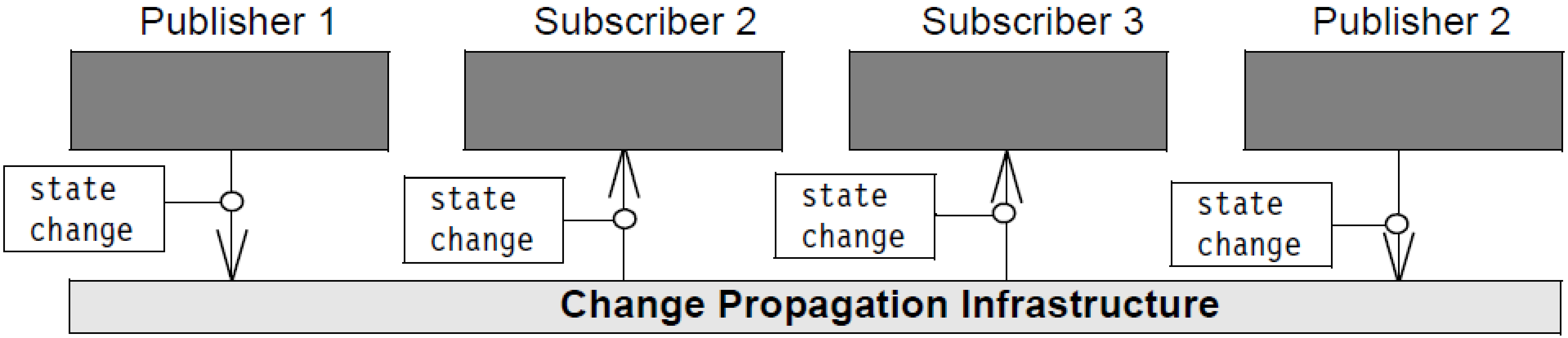
OK

# Summary



- Hard-coding dependencies between publishers & subscribers is avoided by dynamically registering subscribers with the change notification infrastructure
  - Subscribers can join & leave at any time & new types of subscribers that implement the update interface can be integrated without changing the publisher

# Summary



- Hard-coding dependencies between publishers & subscribers is avoided by dynamically registering subscribers with the change notification infrastructure
- The active propagation of changes by the publisher via the event channel avoids polling & ensures that subscribers can update their own state immediately in response to state changes in the publisher

# Android Services & Local IPC: The Publisher/Subscriber Pattern (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

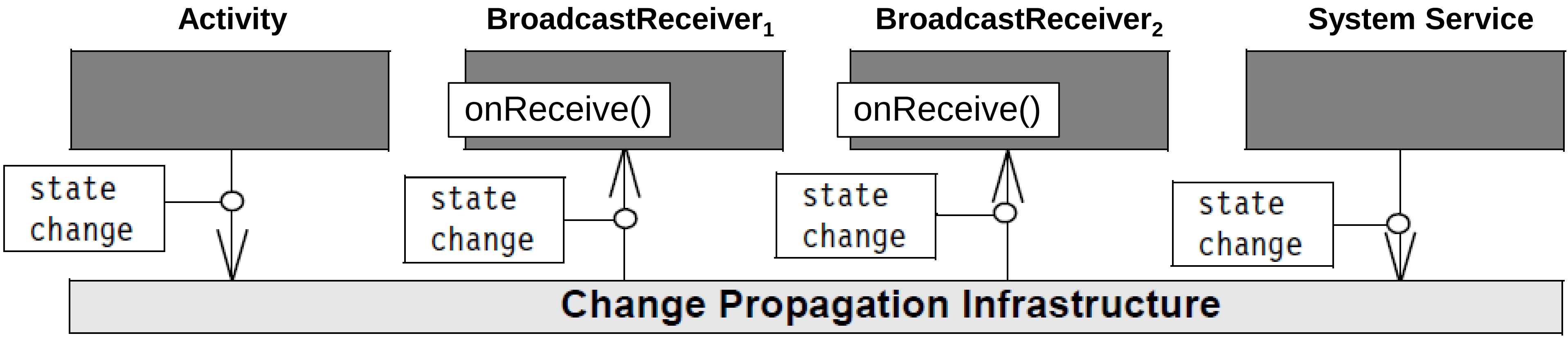
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Module

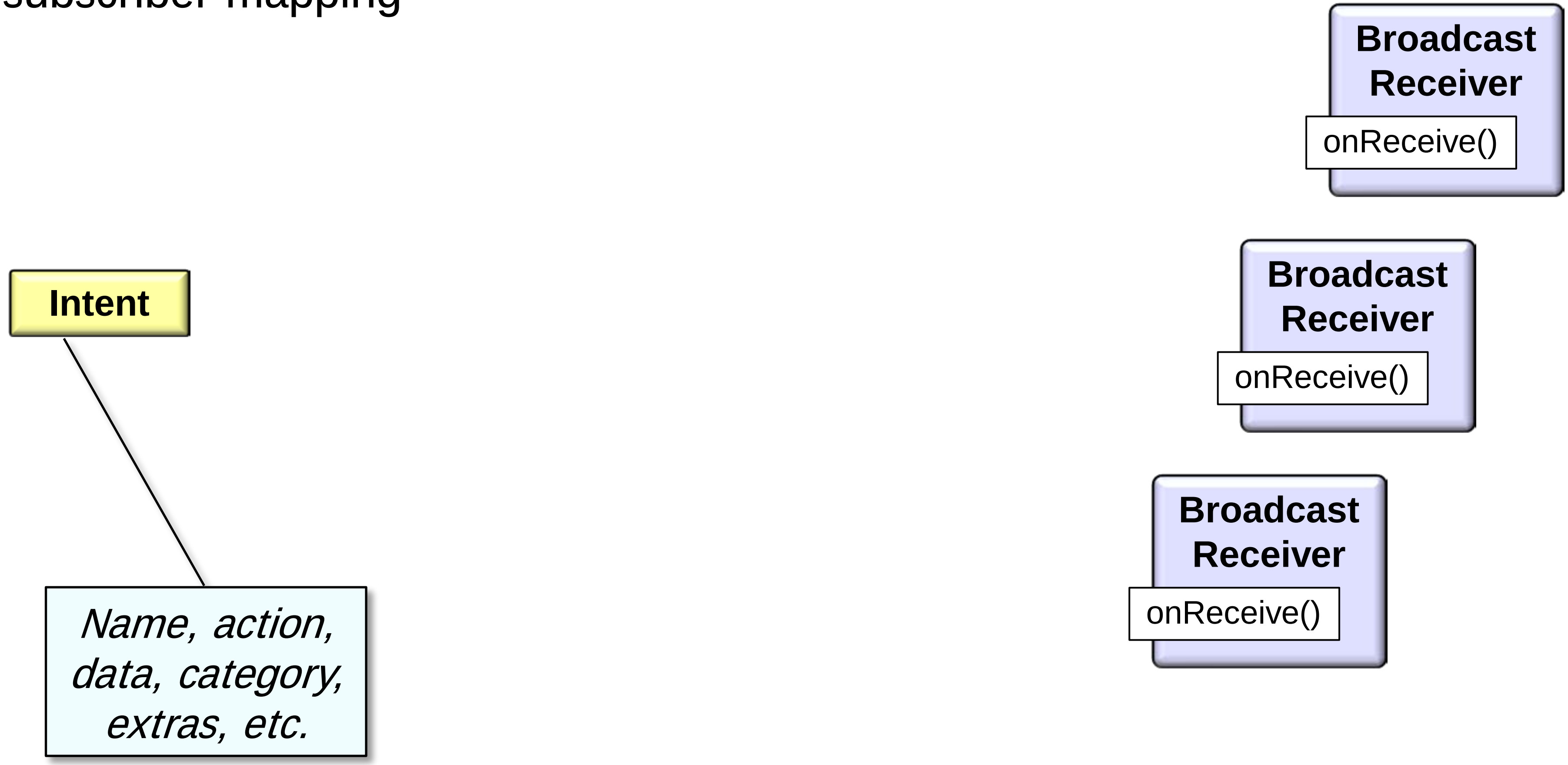
- Understand how the *Publisher-Subscriber* pattern is applied in Android



# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping



developer.android.com/reference/android/content/Intent.html

# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping
- Consider adding filters to narrow interests efficiently

**Broadcast Receiver**

onReceive()

```
<receiver
  android:name=
    ".transaction.PrivilegedSmsReceiver"
  android:permission="android.permission.BROADCAST_SMS">
  <intent-filter>
    <action android:name=
       "android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

AndroidManifest.xml

**Intent**

*Name, action, data, category, extras, etc.*

developer.android.com/guide/components/intents-filters.html

# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping
- Consider adding filters to narrow interests efficiently
- Define/implement the subscriber registration API
  - Provide method(s) for registering receives & (optionally) filters

```
public abstract class Context {
  ...

  public abstract Intent
    registerReceiver
      (BroadcastReceiver receiver,
       IntentFilter filter);

  public abstract Intent
    registerReceiver
      (BroadcastReceiver receiver,
       IntentFilter filter,
       String broadcastPermission,
       Handler scheduler);
  ...
```

frameworks/base/core/java/android/content/Context.java has source code

# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping
- Consider adding filters to narrow interests efficiently
- Define/implement the subscriber registration API
  - Provide method(s) for registering receives & (optionally) filters
  - Registered subscribers are typically stored in an internal data structure

```
class ActivityManagerService
    extends ActivityManagerNative ... {
  ...
  final HashMap mRegisteredReceivers
    = new HashMap();

  public Intent registerReceiver
    (IApplicationThread caller,
     String callerPackage,
     IIntentReceiver receiver,
     IntentFilter filter,
     String permission) {
    ...
    ReceiverList rl = (ReceiverList)
        mRegisteredReceivers.
          get(receiver.asBinder());
    ...
    mRegisteredReceivers.
        put(receiver.asBinder(), rl);
    ...
```

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping
- Consider adding filters to narrow interests efficiently
- Define/implement the subscriber registration API
- Define/implement the subscriber notification API
  - Provide method(s) for controlling how notifications are delivered

```
public abstract class Context {
  public abstract void
    sendBroadcast(Intent intent);

  public abstract void
    sendOrderedBroadcast
        (Intent intent,
         String receiverPermission);
  ...
```

# Publisher-Subscriber POSA1 Architectural

## Implementation

- Determine the publisher-subscriber mapping
- Consider adding filters to narrow interests efficiently
- Define/implement the subscriber registration API
- Define/implement the subscriber notification API
  - Provide method(s) for controlling how notifications are delivered
  - Handle concurrent & sequential deliveries

```
class ActivityManagerService
    extends ActivityManagerNative ... {
  ...
  private final int
    broadcastIntentLocked
      (..., Intent intent, ...) {
    ...
    receivers = AppGlobals.
      getPackageManager().
        queryIntentReceivers(intent,
                             ...);
    ...
    registeredReceivers =
      mReceiverResolver.queryIntent
        (intent, ...);
    ...
  }
```

Static receivers

Dynamic receivers

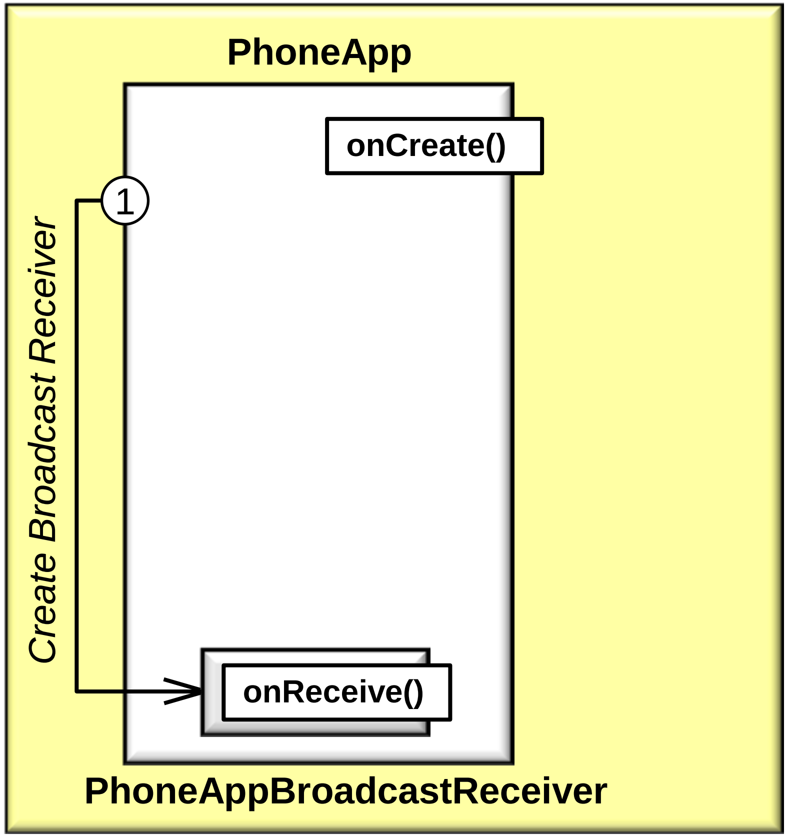
Broadcast intent to receivers

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

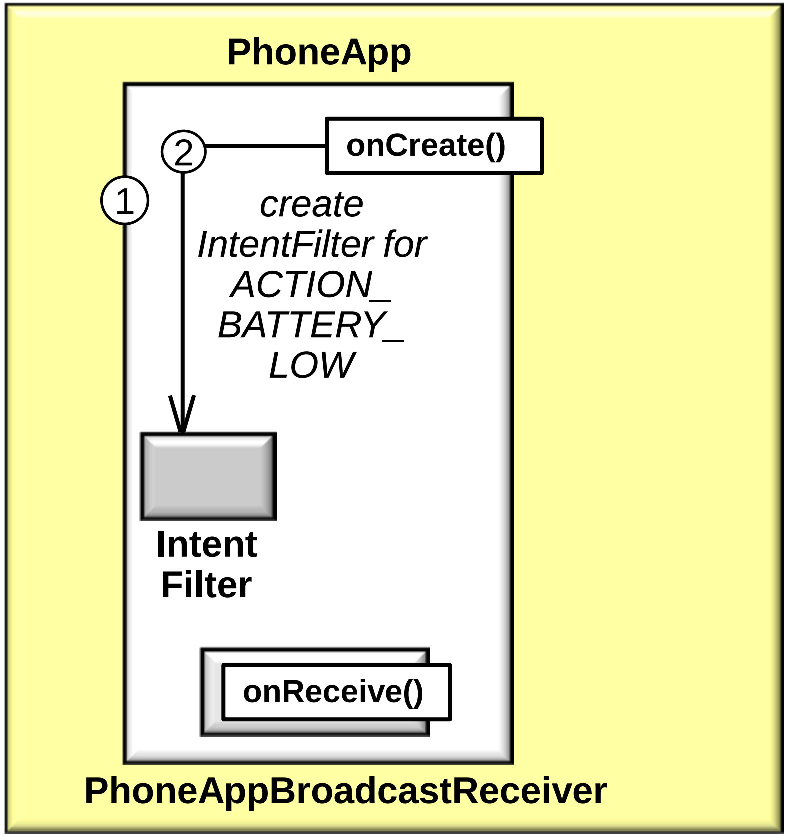


packages/apps/Phone/src/com/android/phone/PhoneApp.java has source code

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

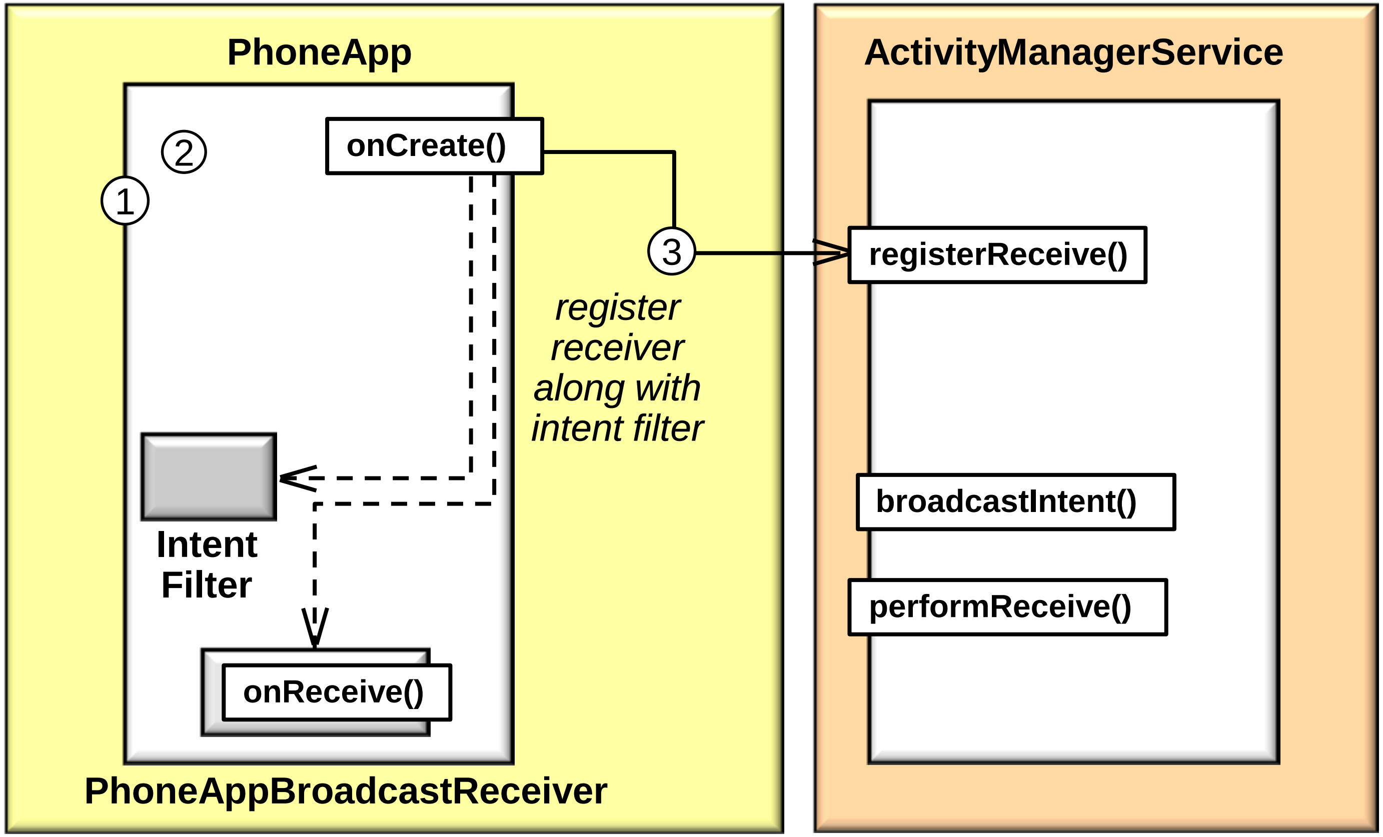

PhoneApp
onCreate()
2
1
create IntentFilter for ACTION_ BATTERY_ LOW
Intent Filter
onReceive()
PhoneAppBroadcastReceiver

[packages/apps/Phone/src/com/android/phone/PhoneApp.java](packages/apps/Phone/src/com/android/phone/PhoneApp.java) has source code

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

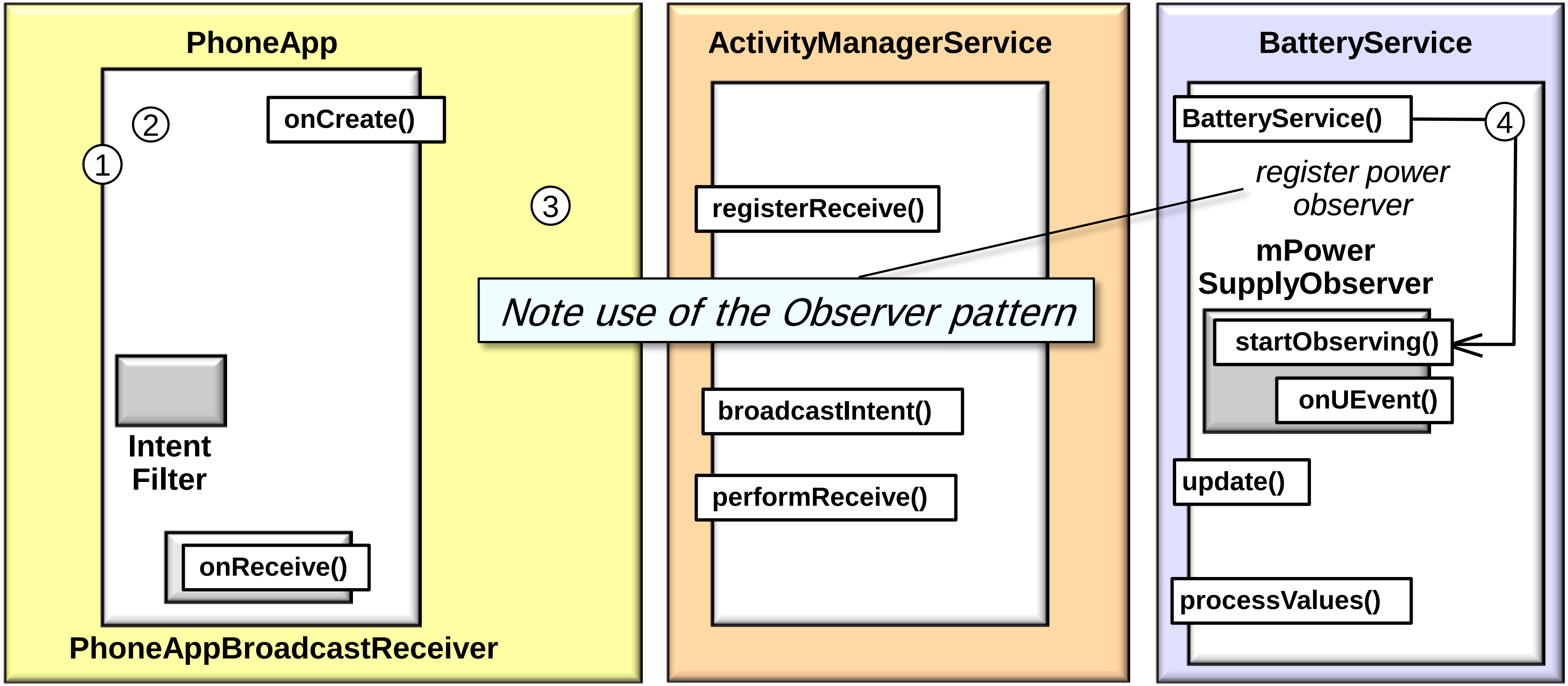


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device



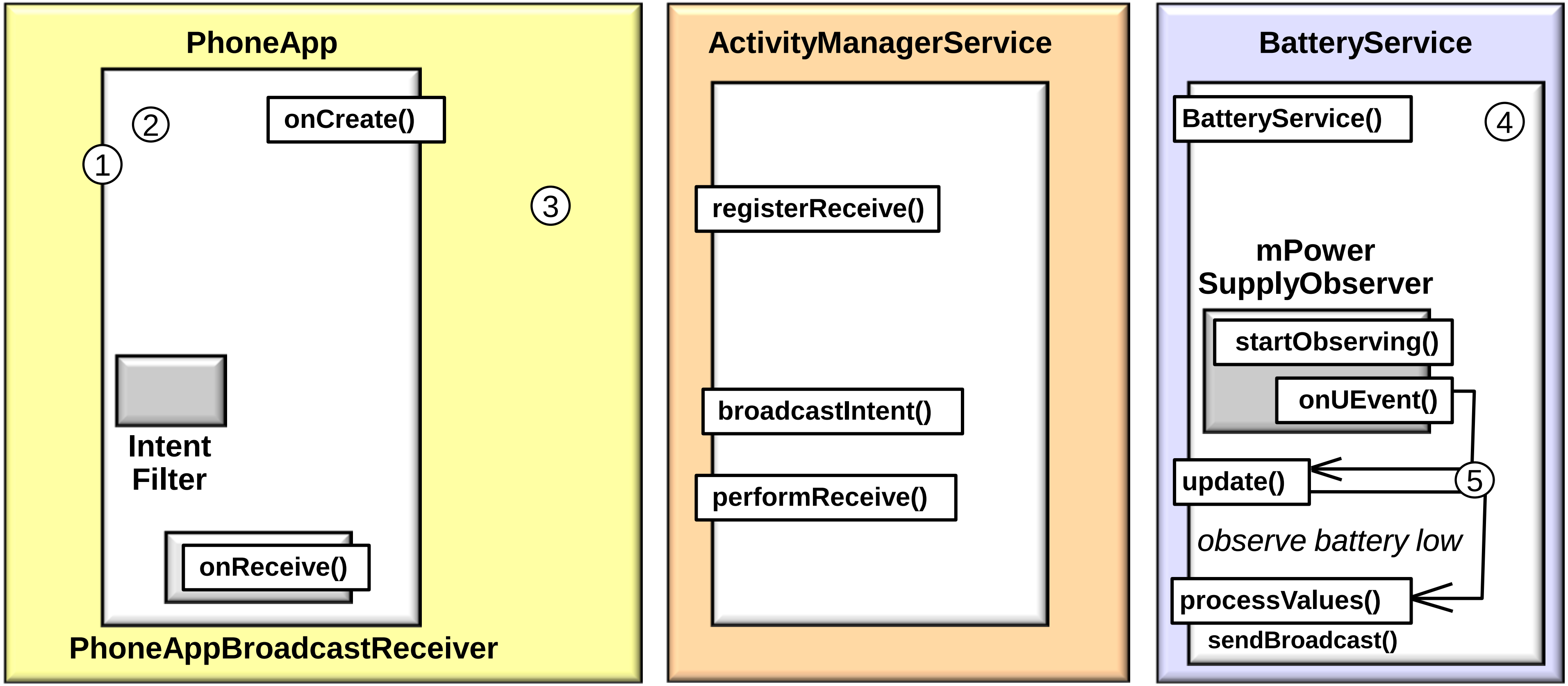
*Note use of the Observer pattern*

See frameworks/base/services/java/com/android/server/BatteryService.java

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

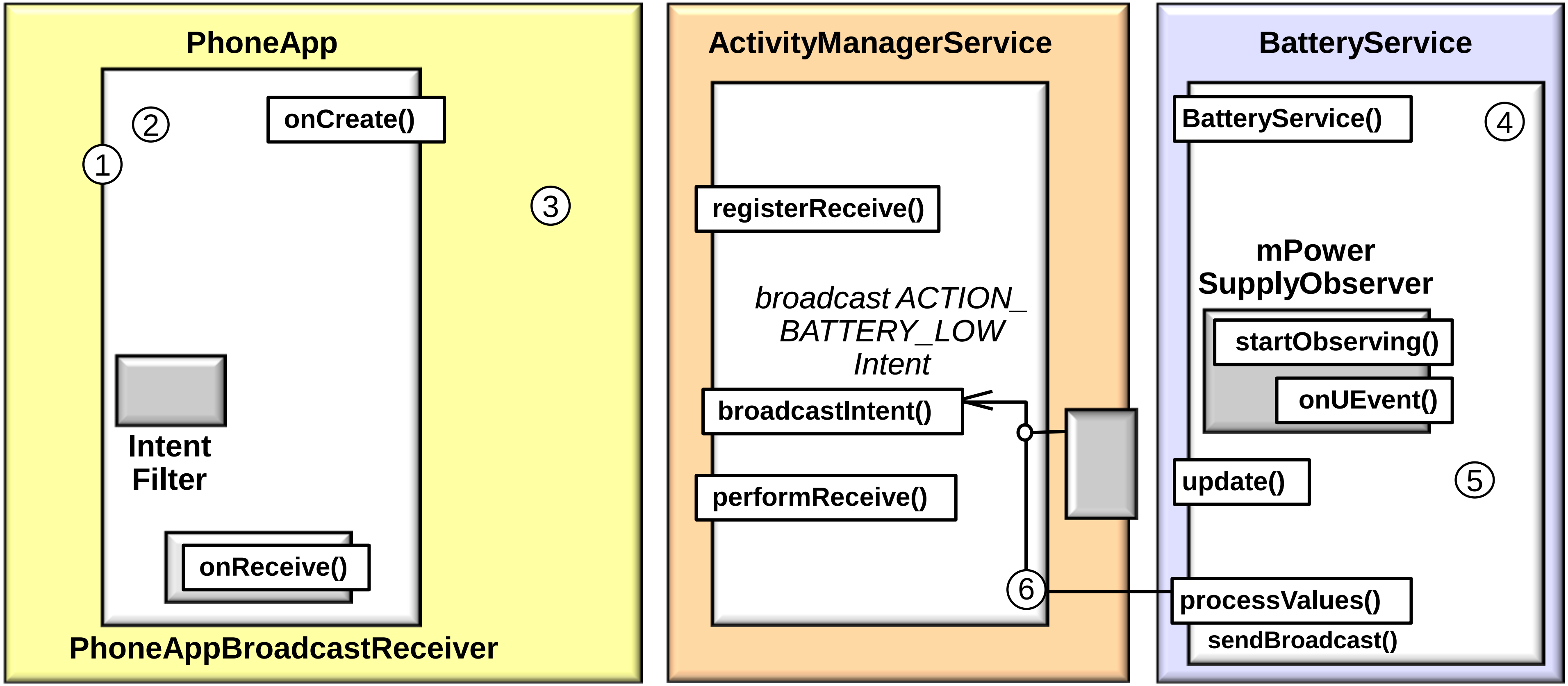


See frameworks/base/services/java/com/android/server/BatteryService.java

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

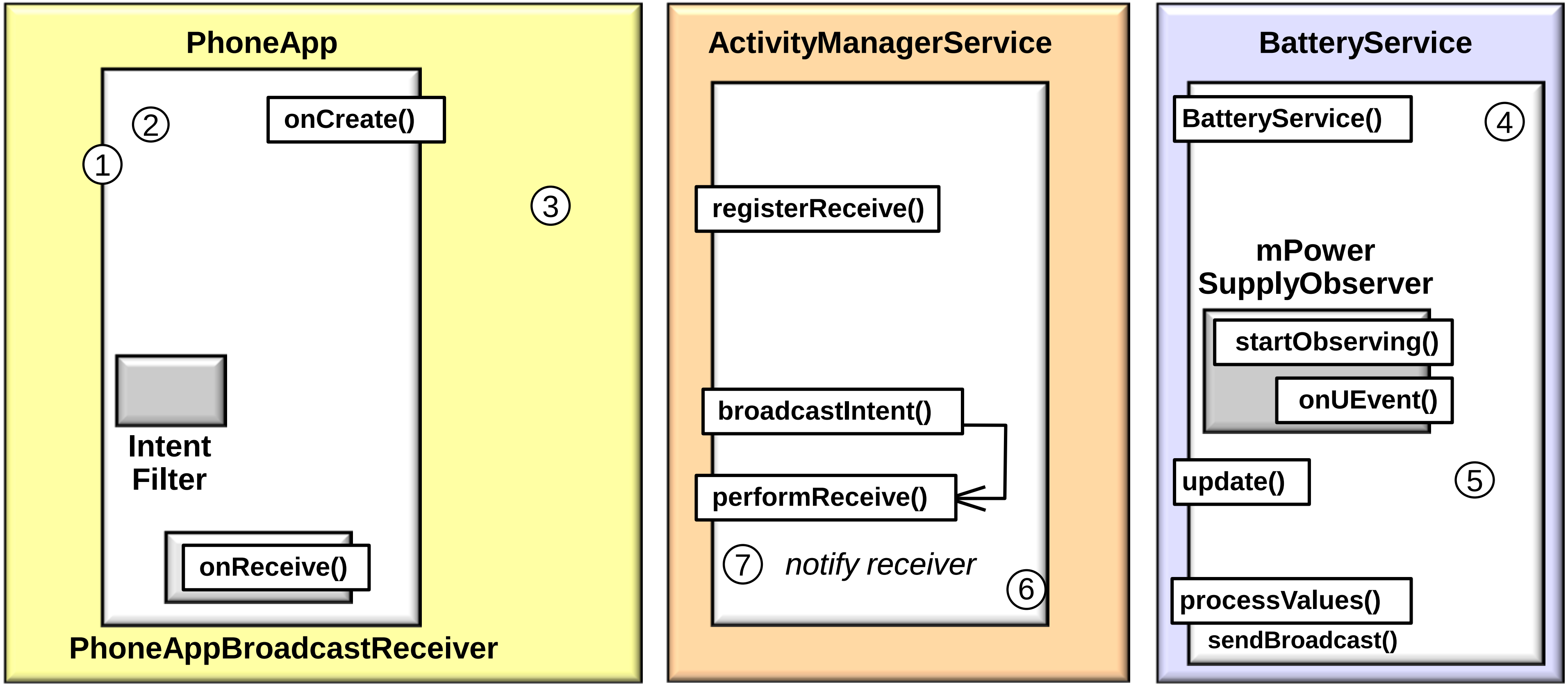


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Publisher-Subscriber POSA1 Architectural

## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device

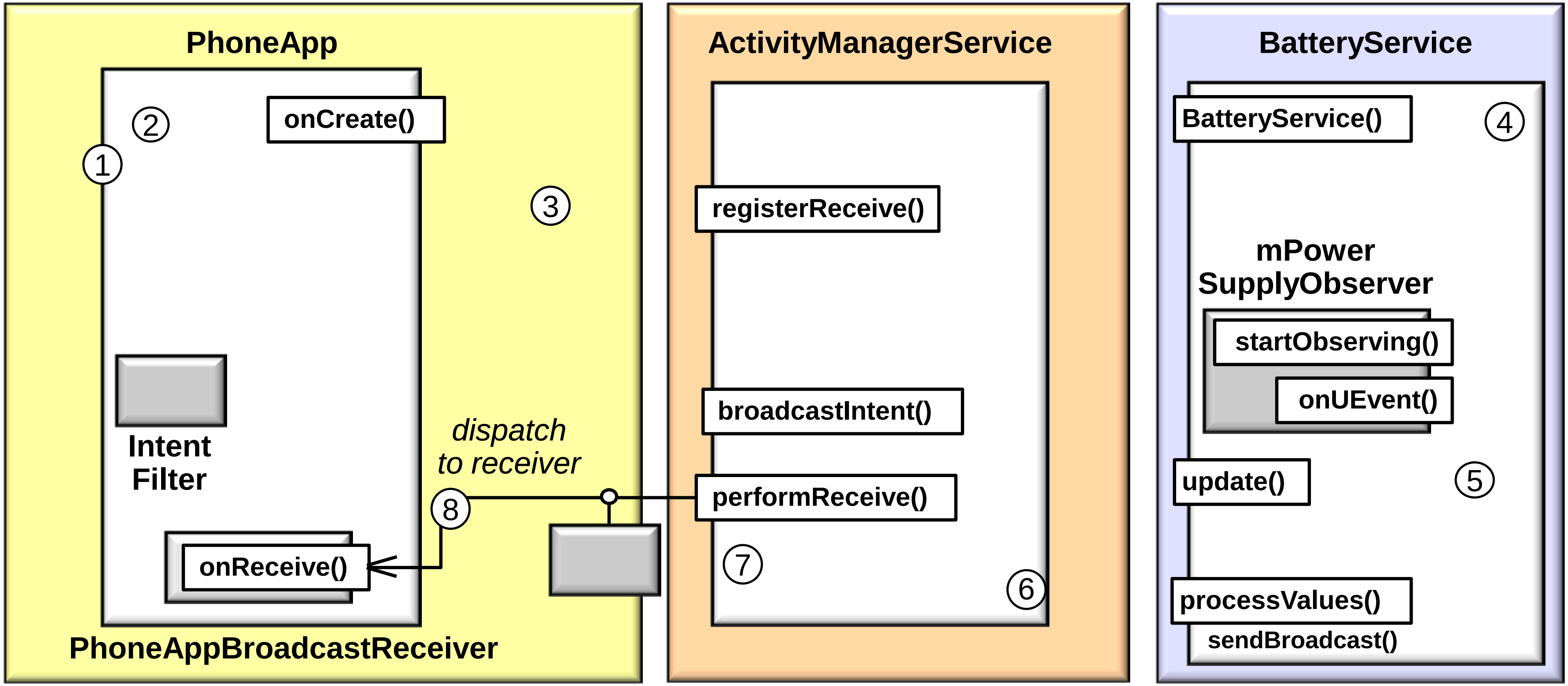


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

# Publisher-Subscriber POSA1 Architectural

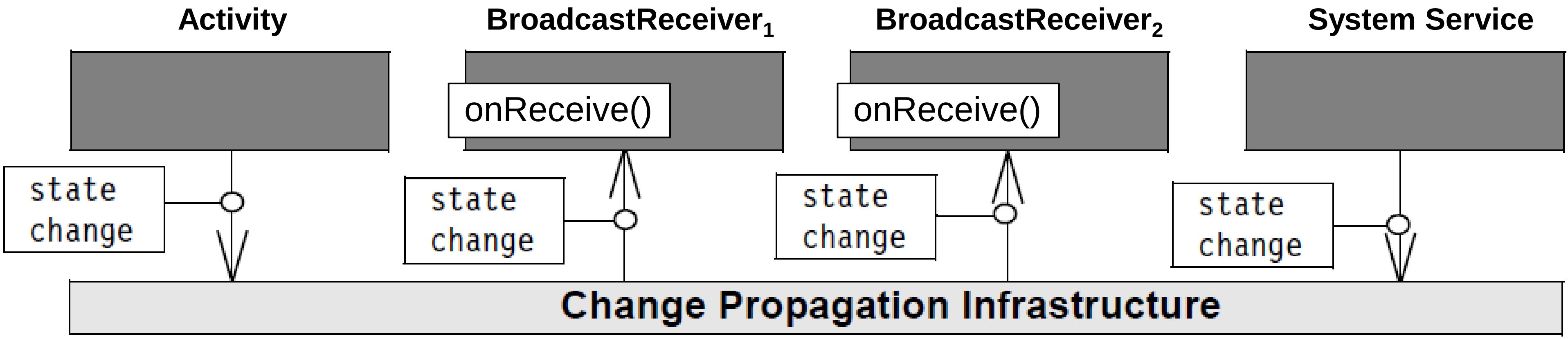
## Applying the Publisher-Subscriber pattern in Android

- Use the Intents framework to report low battery status on an Android device



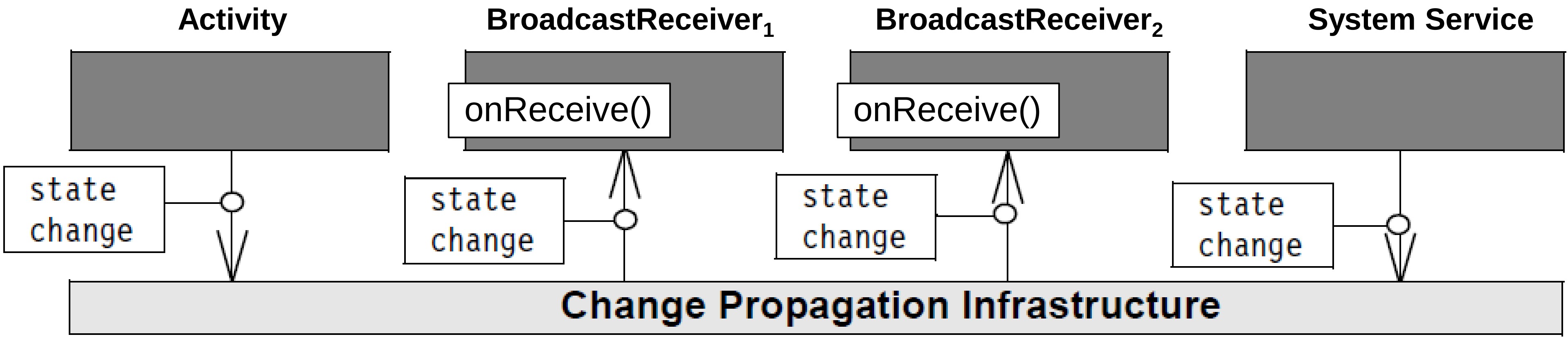
packages/apps/Phone/src/com/android/phone/PhoneApp.java has source code

# Summary



- Android implements the *Publisher-Subscriber* pattern via the Intents framework to enable late run-time binding between components in the same or different Apps
  - The Intent object is a passive data structure holding an abstract description of some change that has occurred & is being announced

# Summary



- Android implements the *Publisher-Subscriber* pattern via the Intents framework to enable late run-time binding between components in the same or different Apps
- Intent objects passed to any of the broadcast methods (such as Context. sendBroadcast() or Context.sendOrderedBroadcast()) are delivered to all interested broadcast receivers