Services, Processes and Binder

Agenda

- Services
    - Essentials
    - Started
    - Intents
    - Bound
- Processes
- Binder and IPC
    - Basic
    - Advanced

Services

Why Services?

- To separate the model from UI
- A service is the only way, other than a visible Activity, to communicate to the system, that useful work is being done
- Inter-process communications

Service Basics

What is a Service?

- “Activity with no UI”
- Must subclass `Service`
- Must be declared in the Manifest

Service Basics

```
public class CookieService extends Service {
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    // …
}
```

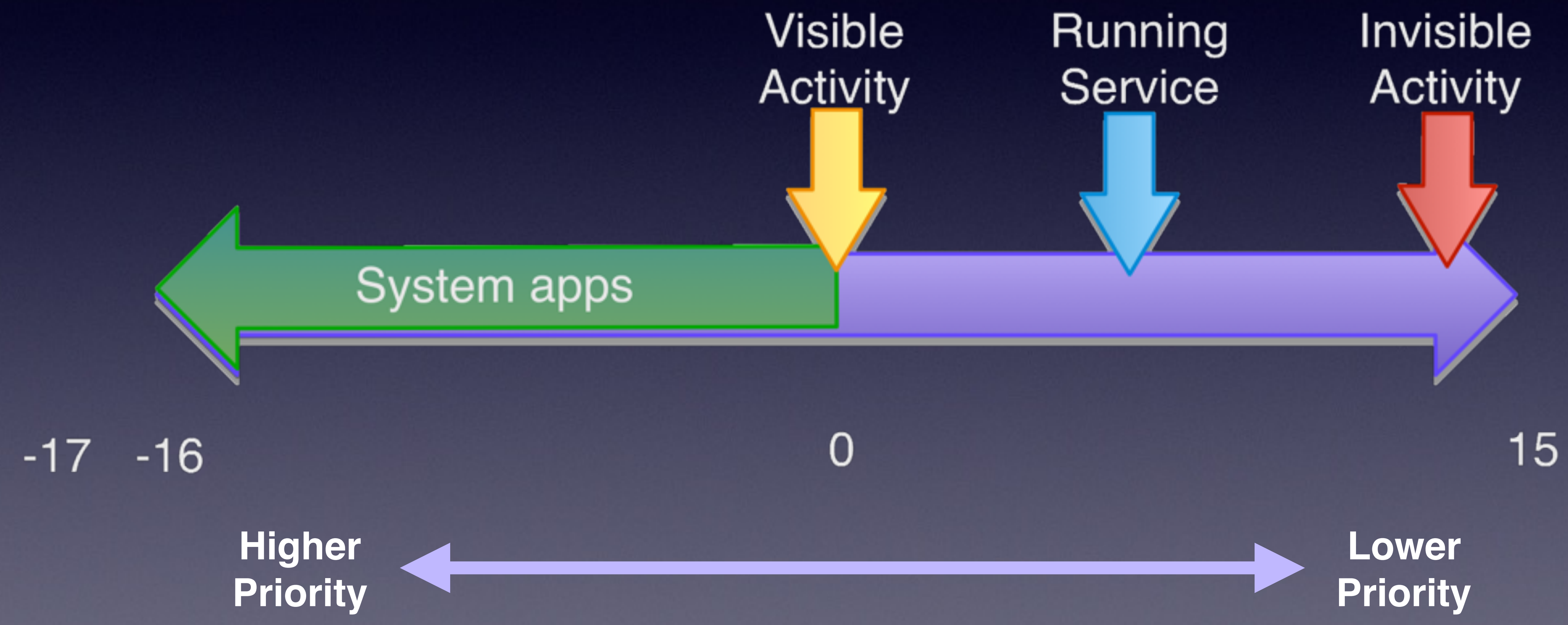
Service Basics

```
<service
    android:name=".CookieService"
    <!-- default -->
    android:enabled="true"
    <!-- default -->
    android:exported="false"
    />
```

Service Visibility

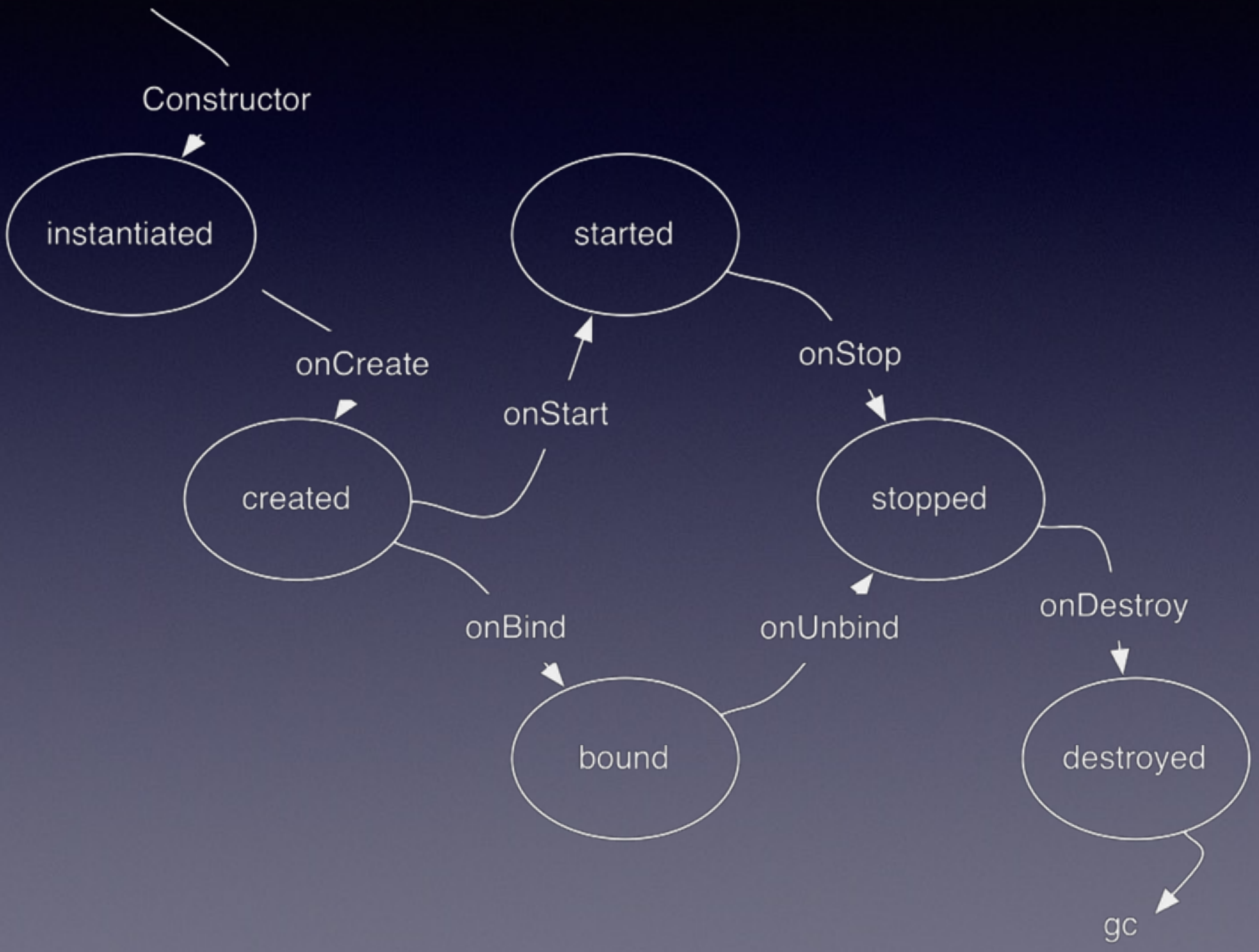
- By default, not visible outside hosting process
- To make it visible:
  - `exported="true"`
  - `<intent filter>`

Service Priority



oom_adj

Service Lifecycle



Services:
Two <u>Unrelated</u> Components:

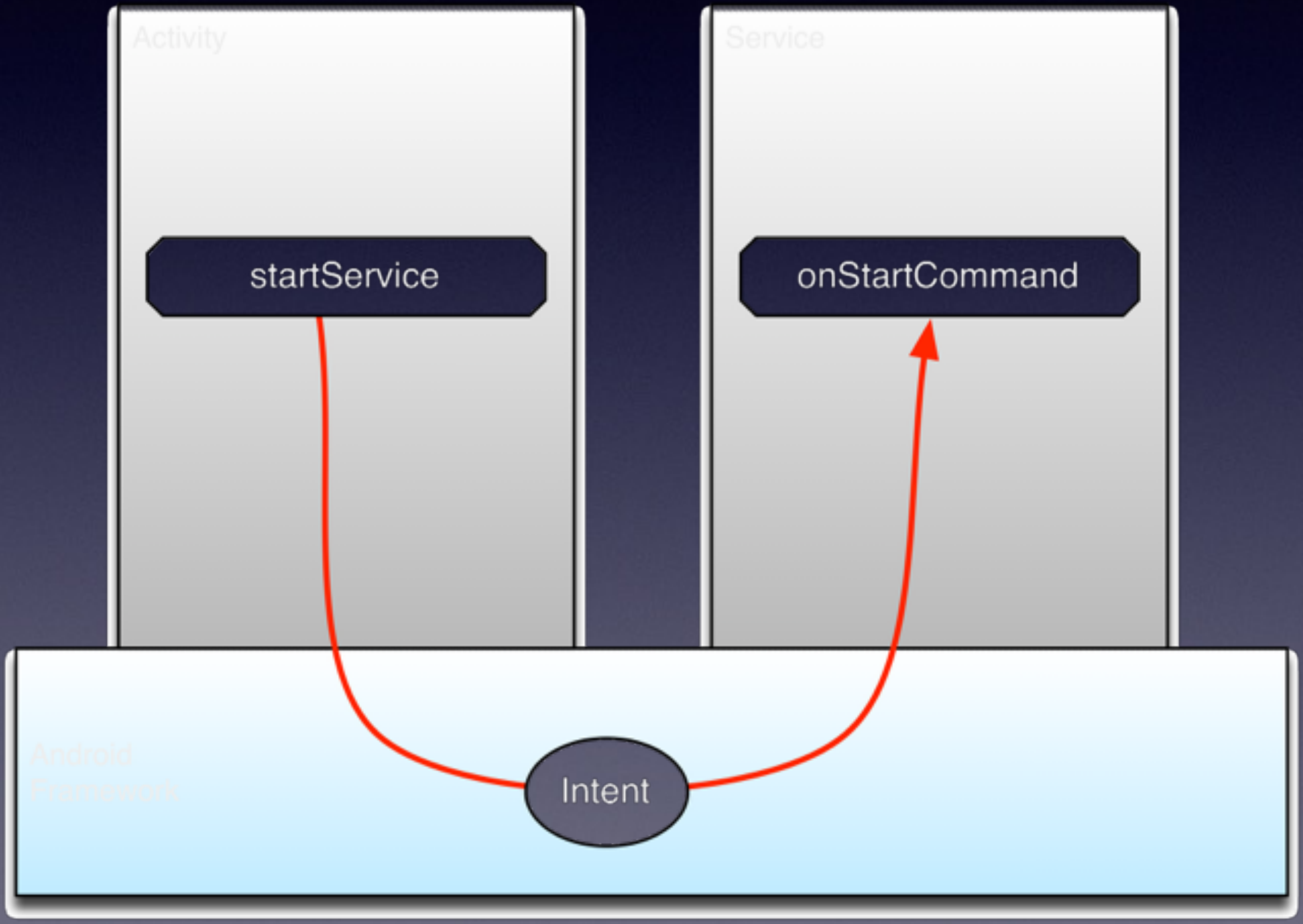
- Started service
- Bound service

Started Services

Started Service

- Client:
  - `Context.startService(intent)`
  - `Context.stopService(intent)`
- Service:
  - `onStartCommand(Intent cmd)`
  - `stopSelf()`

Started Services



Demo!

The Cookie Service

Service Helpers

```
public static void eatCookie(
    @NonNull Context ctxt,
    @NonNull String cookie)
{
    Intent intent = new Intent(
            ctxt, CookieService.class);
    intent.setAction(ACTION_EAT);
    intent.putExtra(PARAM_COOKIE, cookie);
    ctxt.startService(intent);
}
```

Simple Started Service

```
// Runs on the UI thread!!
public int onStartCommand(
    Intent intent, int flags, int startId) {
    String action = intent.getAction();
    switch (action) {
        case ACTION_EAT:
            doEatACookie(
                intent.getStringExtra(PARAM_COOKIE));
            break;

        // ... other ACTIONS.
        default:
            Log.w(TAG,
            "unexpected action: " + action);
    }

    return Service.START_NOT_STICKY;
}
```

Service Restart

- `START_NOT_STICKY`

  Stopped and forgotten

- `START_REDELIVER_INTENT`

  Restarted after interruption; last intent re-delivered.

- `START_STICKY`

  Not marked stopped! `onStartCommand` *called with null!*

Lab!

The Cookie Service runs on the Main Thread!

…Use our old friend AsyncTask to get it off!

Lab!

- Started Service
- Process 2 Actions
  - `eatCookie(String cookie)`
  - `eatCookieNoisily(String cookie)`
- Use AsyncTask to move processing off the main thread
  - Create the task in onStartCommand
  - from the AsyncTask, call Service method `processIntent`
- Action implementation:
  - Log method name and cookie
  - `Thread.sleep(60 * 1000)`
- Demonstrate that it works.
- Stretch: stop service when idle

Async Task Service

- AsyncTasks do work in Services!
- In a Service, an AsyncTask cannot leak the Activity
- No way to return a value (yet)

Custom Task Services

```
executeOnExecutor(
    NETWORK_QUERY_EXECUTOR,
    intent);
```

- Stop the service when idle
- Stop the custom Executor, in `onDestroy`

Intent Service

- Similar to the AsyncTask service
    - Simpler: uses a HandlerThread (Looper)
- One thread per service:
    - In order execution
    - *Tasks can communicate wo/ synchronization*

Invoking Services

Whether started or bound, services are invoked with:

Intents

(a brief aside)

Intents

- Parcelable (therefore, duplicatable)
- The Intent received by service is *never* the intent sent by the client (therefore thread safe)

Demo!

Is the Intent that the Service receives the one the Activity sent?

Intents

- Implicit
  - Bound at runtime
  - May name several services
- Explicit
  - Bound at compile time
  - Names at most one service

Implicit Intent

```
Intent intent = new Intent();
intent.setAction(ACTION_EAT_NOISILY);
intent.addCategory(CATEGORY_COOKIES)
```

Intent Filter

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.callmeike.android.x">

    <application>
        ...
        <service android:name=".ExampleService">
            <intent-filter>
                <action
                   android:name=
                       "net.callmeike.android.action.PING" />
                <category
                   android:name=
                       "net.callmeike.android.category.NET" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

Implicit Intents

- Match an <intent-filter>: ACTION, CATEGORY, etc…
- These fields are compared by intent's `.equals`! (Extras are not)
- By registering an intent filter, a service becomes **`exported`**! Use **`exported="false"`**

Disambiguation

- Completely Unsafe
- Nearly random (unlike Activities)
- Post KitKat, service intents must be explicit
- … well, at least must name a package

Demo!

What makes an Intent explicit?

Explicit Intents

```
Intent intent1
    = new Intent(ctxt, Svc.class);

Intent intent2 = new Intent();
intent2.setComponent(new ComponentName(
    ctxt.getPackageName(),
    Svc.class.getName());

Intent intent3 = new Intent();
intent2.setComponent(new ComponentName(
    "net.callmeike.android.svc",
    "net.callmeike.android.svc.Svc");
```

Explicit Intents

- Must include the package name
- Does *not* include a Java reference to the class

Late-binding Explicit Intents

```
Intent intent = new Intent();
intent.setPackageName(
    "net.callmeike.android.svc");
intent.setAction(
    "net.callmeike.android.action.PING");

<service android:name=".ExampleService">
    <intent-filter>
        <action android:name=
            "net.callmeike.android.action.PING" />
   </intent-filter>
</service>
```

Contracts and Libraries

Contracts are stand alone code that defines necessary symbols and types.

Clients must know, at the very least, the exact package name of the service

Demo!

Convert the Cookie service to run in a remote process

Break!

Convert to using the remote CookieService and then use the `ProcessStats` class from the `lib` module, to explore process `oom_adj`.

- Start logging in `onCreate`
- Stop logging in `onDestroy`
- What is the priority when:
  - Activity is in front?
  - Service is started?
  - Service stopped (hint: see SlowService)?
  - Is `onDestroy` always called?
  - Other states?

Bound Services

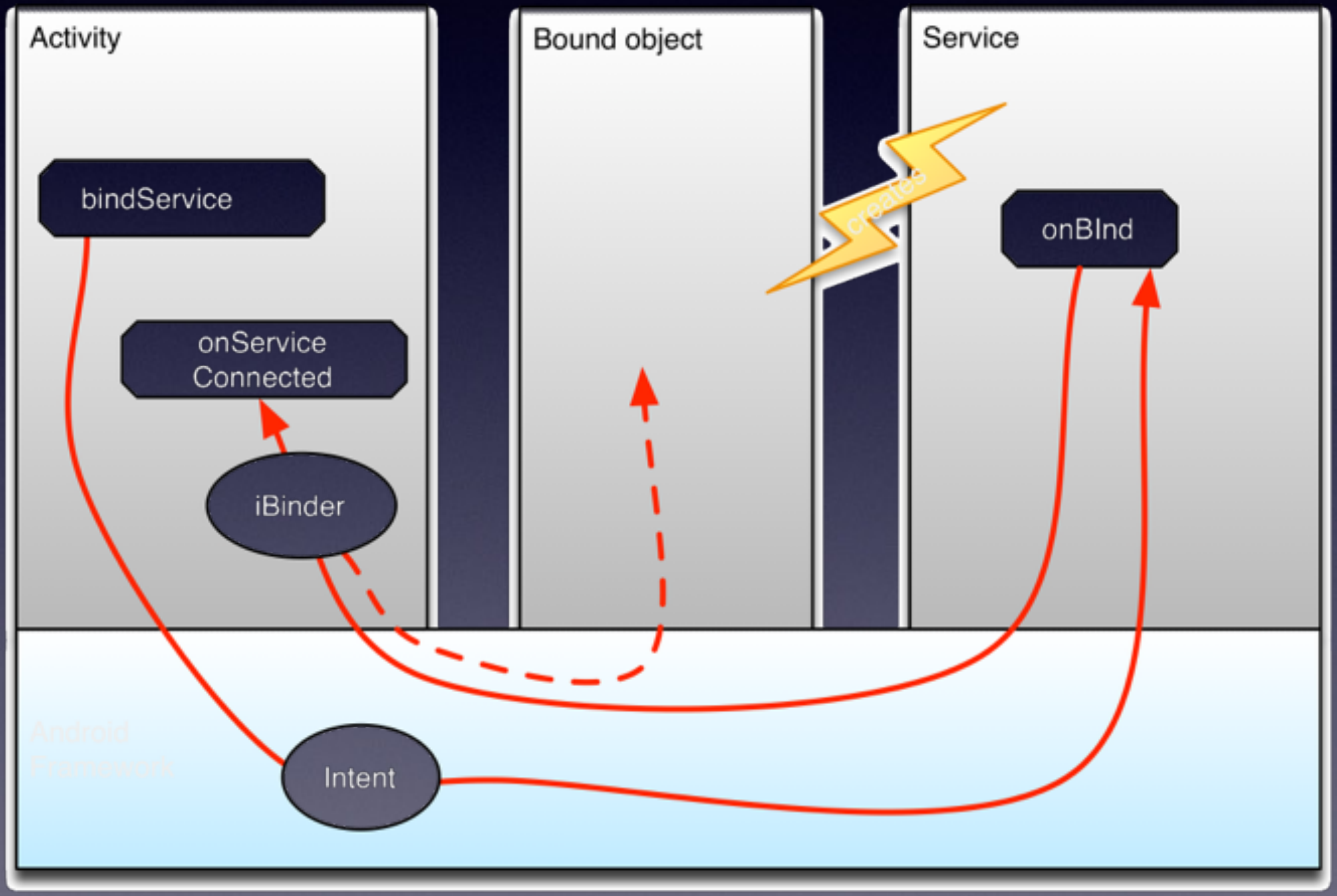
Bound Services

- Among the more complicated features of Android
- Something like a Factory, for a business object.
- Faster, within a process, than sending Intents
- Easier for methods to return values

Bound Service

- Client:
  - `Context.bindService(intent)`
  - `Context.unbindService(connection)`
  - `ServiceConnection`
- Service:
  - `onBind(Intent cmd)`
  - `onUnbind(Intent cmd)`

Bound Services



Binding a Service

- `bindService` returns false if no service can be identified
- Target service receives a a call to `onBind`
- `onBind` returns a Binder object
- `ServiceConnection.onServiceConnected` receives an iBinder object

Basic Bound Service

```java
public class BoundService extends Service {
   private static Binder service;

    @Override
    public void onCreate() {
        super.onCreate();
        if (null == service) {
            service = new BoundServiceImpl(
                 getApplicationContext);
        }
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return service;
    }

    // …
}
```

Basic Bound Service Client

```
public class MainActivity extends Activity
    implements ServiceConnection {
    @Override
    public void onServiceConnected(
        ComponentName name,
        IBinder iBinder) {
        svc1 = iBinder;
    }

    @Override
    public void onServiceDisconnected(
        ComponentName name) {
        svc1 = null;
    }
   // …
```

Basic Bound Service Client

```
public class MainActivity
    extends Activity implements ServiceConnection {

    // …

    @Override
    protected void onStop() {
        super.onStop();
        unbindService(this);
        onServiceDisconnected(null);
    }

    @Override
    protected void onStart() {
        super.onStart();
        Intent i = new Intent(this, BoundService.class);
        bindService(i, this, Context.BIND_AUTO_CREATE);
    }
}
```

Unbinding a Service

- Unbind what you bind!
- Bindings per Context and ServiceConnection
    - You must unbind from the same context from which you bound!
    - You can bind one Service per ServiceConnection

Forgetting a Service

When Android kills an Activity that is still holding a reference to a Service, it knows that the service cannot be correctly unbound.

```
Activity net.callmeike.android.x.MainActivity
  has leaked ServiceConnection
  net.callmeike.android.x.MainActivity@1d77f05
  that was originally bound here
```

Local Service Trick

```
public class LocalService extends Service {
    private static final serviceHolder
        = new ServiceHolder() ;

    public class ServiceHolder extends Binder {
        public LocalService getService() {
            return LocalService.this;
        }
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return serviceHolder;
    }
}
```

example code

Local Service Client

```
private LocalService svc;

@Override
public void onServiceConnected(
    ComponentName componentName,
    IBinder b) {
    svc = ((LocalService.ServiceHolder) b)
          .getService();
}
```

example code

The Local Service Trick

- Works because the binder the client receives is exactly the object that the service returned (`Binder` implements `iBinder`)
- Works only for local services!
- The Service is the "factory"
- The service is, frequently, the object the factory returns
- Better to inject the returned object

Demo!

The Prefix Service

Lab!

Using the Activity as the ServiceConnection allows us to bind only a single service

What if we want to bind two?

Lab!

- Bind both local services (`LocalService1`, `LocalService2`)
- Both are prefix services
  - one prefixes "ONE: "
  - two prefixes "TWO: "

hint: you will need two `ServiceConnections`

Demo!

Binding a remote service

Priorities and Flags

- WTF:
  - BIND_DEBUG_UNBIND
- Connection
  - BIND_AUTO_CREATE *<- This one is IMPORTANT*
- Priority
  - BIND_NOT_FOREGROUND
  - BIND_ABOVE_CLIENT
  - BIND_ALLOW_OOM_ MANAGEMENT
  - BIND_WAIVE_PRIORITY
  - BIND_IMPORTANT
  - BIND_ADJUST_WITH_ACTIVITY

Quiz!

What happens if you kill a bound service?

Hint:

```
adb shell ps
adb shell kill -9 <pid>
```

Service Lifecycles

```
class SingletonService extends Service {
    private static int counter;

    public static int getCount() {
        return counter;
    }

    @Override
    public IBinder onBind(Intent intent) {
        return new ManagedObject(++counter);
    }
}
```

Quiz!

What does `counter` count?

Hint: when does `getCounter` **return** `1`?

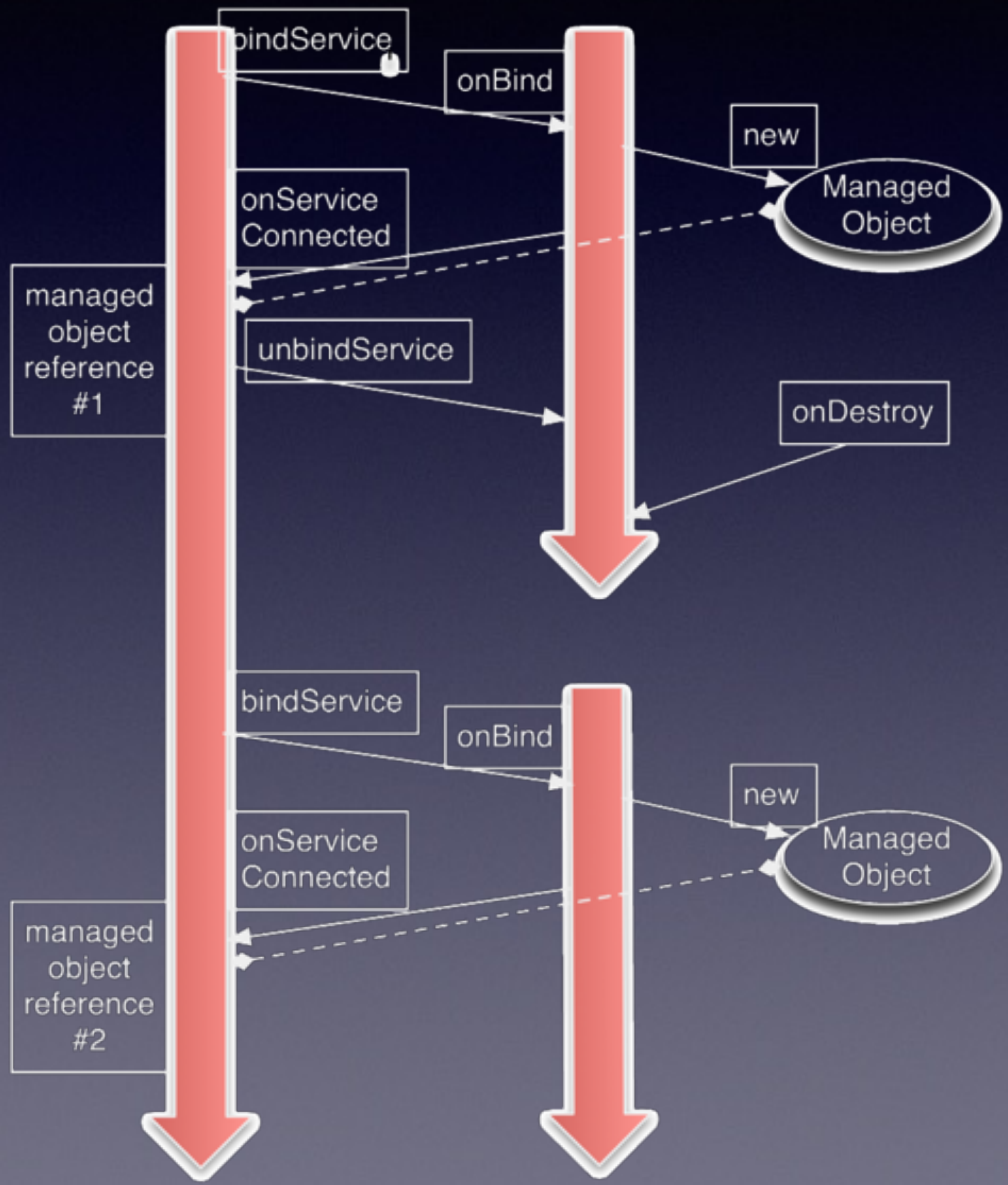
Service Lifecycles

- `onBind` is called only once, for .equals Intents
- A Service may be in a different process! Its statics may be completely reset!

Quiz!

Are the managed objects returned by a Service, singletons?

Service Lifecycles



Service as Factory

- Neither type safe nor synchronous
- Does not return a new instance for each request.
- Requires client collaboration to prevent multiple instances
- You just gotta get used to their peculiarities!

Processes

Processes

- A distinct "application"
  - Doesn’t have to have a UI
- Manifest: `android:process`
  - local process
  - remote process

Local Process

- Name begins with ":"

```
android:process = ":other"
```

- Inaccessible from other applications

Remote Process

- Name begins with lower-case alpha, and contains at least one ".' (period).

```
android:process
    ="com.callmeike.android.shared"
```

- Two applications signed with the same cert and running with the same userId

Demo!

Shared process

Quiz!

Can we use the LocalService trick for Services in a shared process?

Shared Process

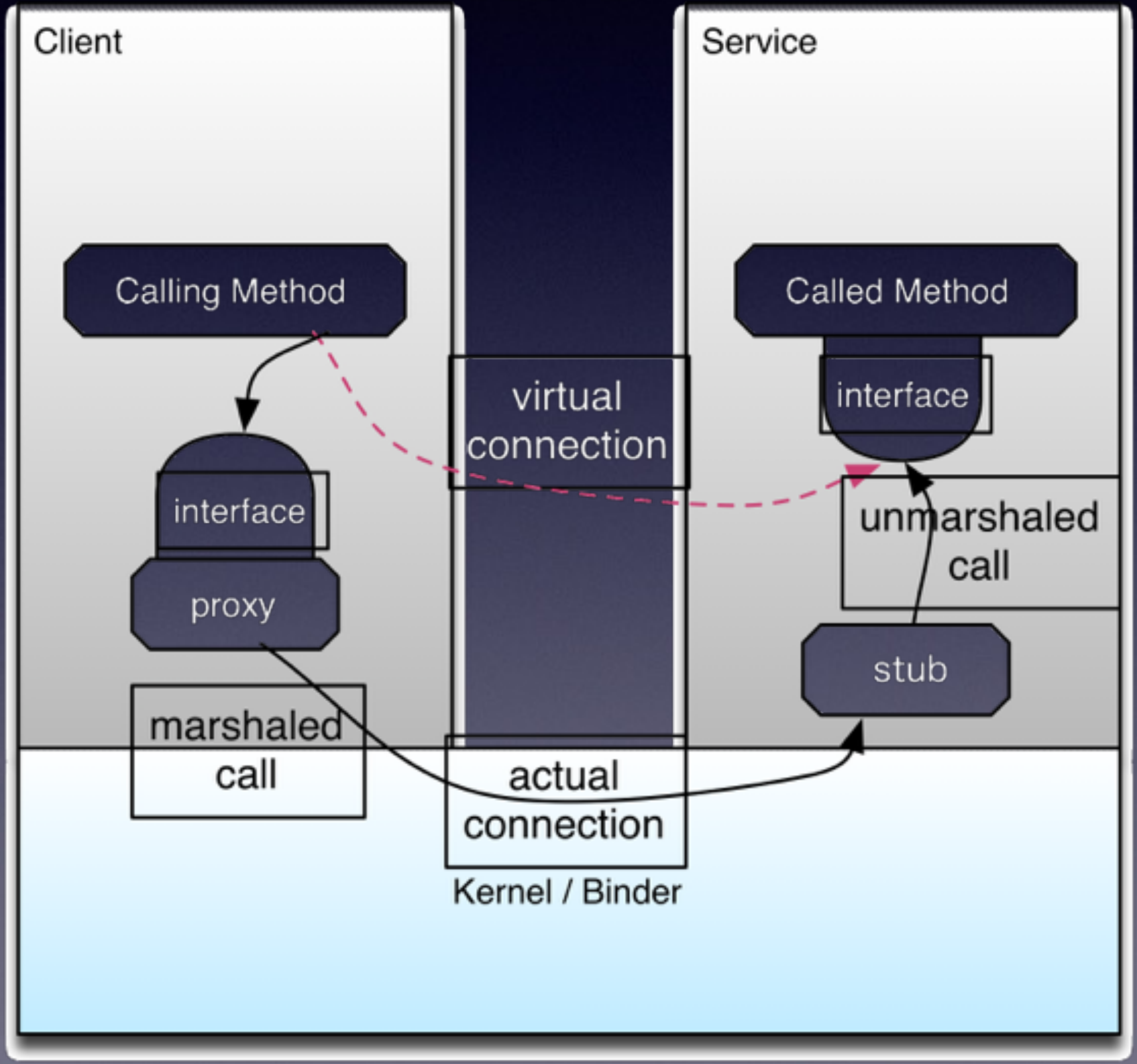
While the processes are the same, the ClassLoaders are not.

Interprocess Communication

IPC

- Only mentioned process boundaries once.
- Services are, generally, agnostic to process boundaries
- Avoid Java `Serializable`; Android `Parcelable` is typically about twice as fast

Generic IPC


Client
Service
Calling Method
Called Method
interface
proxy
virtual connection
interface
unmarshaled call
stub
marshaled call
actual connection
Kernel / Binder

Parcelable

- Implement Parcelable Interface:
    - `describeContents`
    - `writeToParcel`
- `public static CREATOR`
    - implements `Parcelable.Creator`

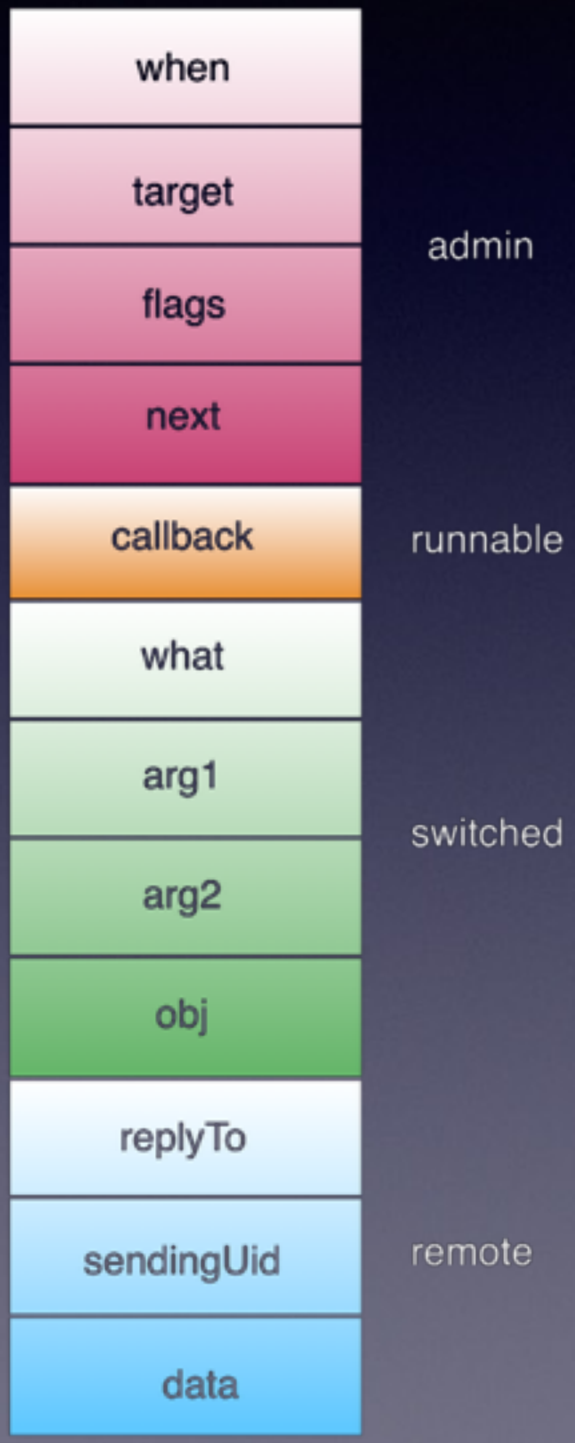
Parcelable

- Marshaling
  - Holding the object
  - call its `writeToParcel`, passing a `Parcel`
- Unmarshaling
  - Holding the class
  - call class `CREATOR.createFromParcel` passing Parcel

Messengers

- Parcelable Reference to a Handler
- Bundle your Strings!

Messages



- Low overhead: pooled, not GCed
- Basis for Messenger IPC

Demo!

Intent Service with Messengers

AIDL

- Compiler
- Source: Java-like language
- Output: Java code for Binder Stub/ Proxy

Using AIDL: client

```
public void onServiceConnected(
    ComponentName name,
    IBinder b) {
    service = IAPI.Stub.asInterface(b);
}
```

Using AIDL: service

```
public class DemoServiceImpl
    extends DemoService.Stub {
    // implement interface…
}

private static DemoServiceImpl service
    = new DemoServiceImpl();

@Override
public IBinder onBind(Intent intent) {
   return service;
}
```

Using AIDL

- Define the interface
- Compile the interface
- Implement the Interface Stub
- Return the implementation from the Service
- Wrap the received binder with Stub.asInterface

Contracts and Libraries

If you have to ship AIDL, you will probably need to provide an aar, instead of/in addition to, a Contract

Demo!

Looking at AIDL output

Advanced Binder

Binding Process

- Find the target app (package name + filter)
- Start it, if it is not started
- Return from bind call (true on success)
- Intent delivered to service
- Service returns binder
- Corresponding IBinder delivered to client

A Connection Broker

- Scanning for Registered packages
- Requesting connections
- Processing connection requests

Demo!

A Binding Broker

Summary

Service

- Services come it two, very distinct flavors:
  - started: `startService`, `stopSelf`
  - bound: `bindService`, `unbindService`
- They are relatively agnostic to process boundaries
- They affect the priority of the host process
- With two important exceptions, service methods run on the Main thread

Started Service

- Use Helper methods to marshal requests
- A good way to implement `void` methods
- IntentServices provide in order, asynchronous execution
- Custom services provide whatever you need.

Bound Services

- They are architecturally weird:
  - The aren't really Singletons
  - They aren't really Factories
  - A bit like WeakReferences?

Bound Services

- Binding
  - One binding per ServiceConnection
  - Bindings are in a single Context
- Unbinding
  - Doesn't disconnect the binding
  - Does affect process priority and notification

IPC

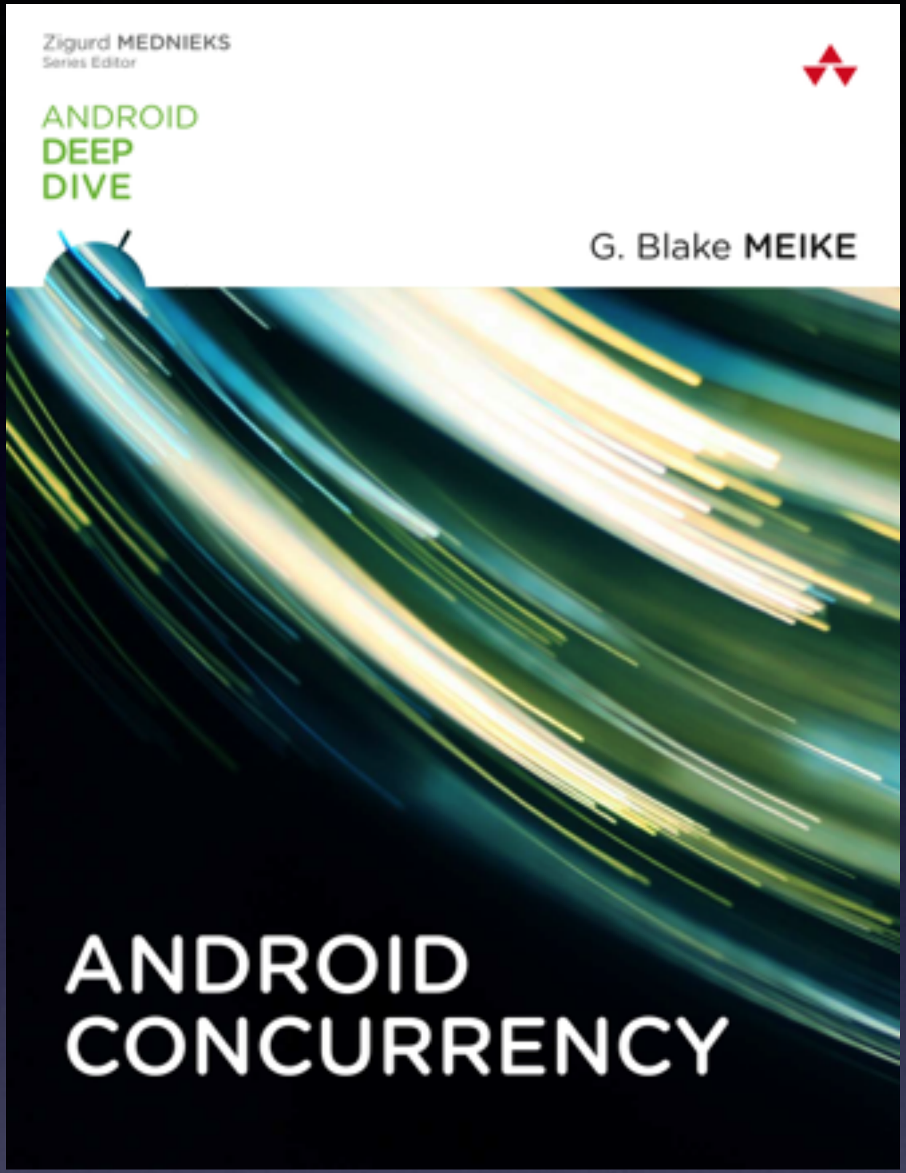
- Parcelables
- AIDL
    - Define the interface
    - Implement the Stub in the Service
    - Use Stub.asInterface in the client
    - Quite brittle

Summary: Binder

- Mind the Binder Threads
- Mind the Data Transfer limit
- Roll your own! Don't need no stinkin' AIDL
- Know when you've been unfriended

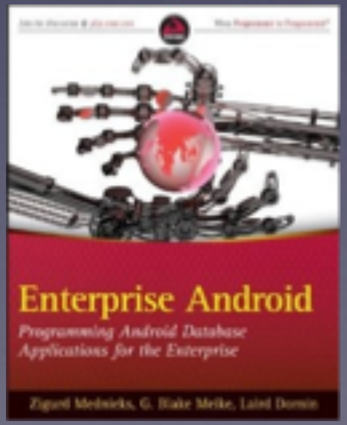
Thank you!

Check out Android Concurrency (Addison-Wesley, 2016)



`blake.meike@gmail.com`

`twitter: @callmeike`


Enterprise Android
Programming Android Database Applications for the Enterprise
Zigurd Mednieks, G. Blake Meike, Laird Dornin

O'REILLY
Programming Android
Zigurd Mednieks, Blake Meike, Masumi Nakamura & Laird Dornin

Presentation code is at:

https://github.com/bmeike/ServiceExperiment.git