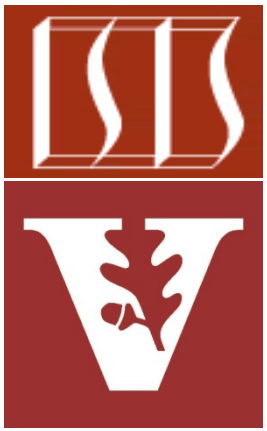
Android Services & Local IPC: Implementing AIDL Interfaces

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



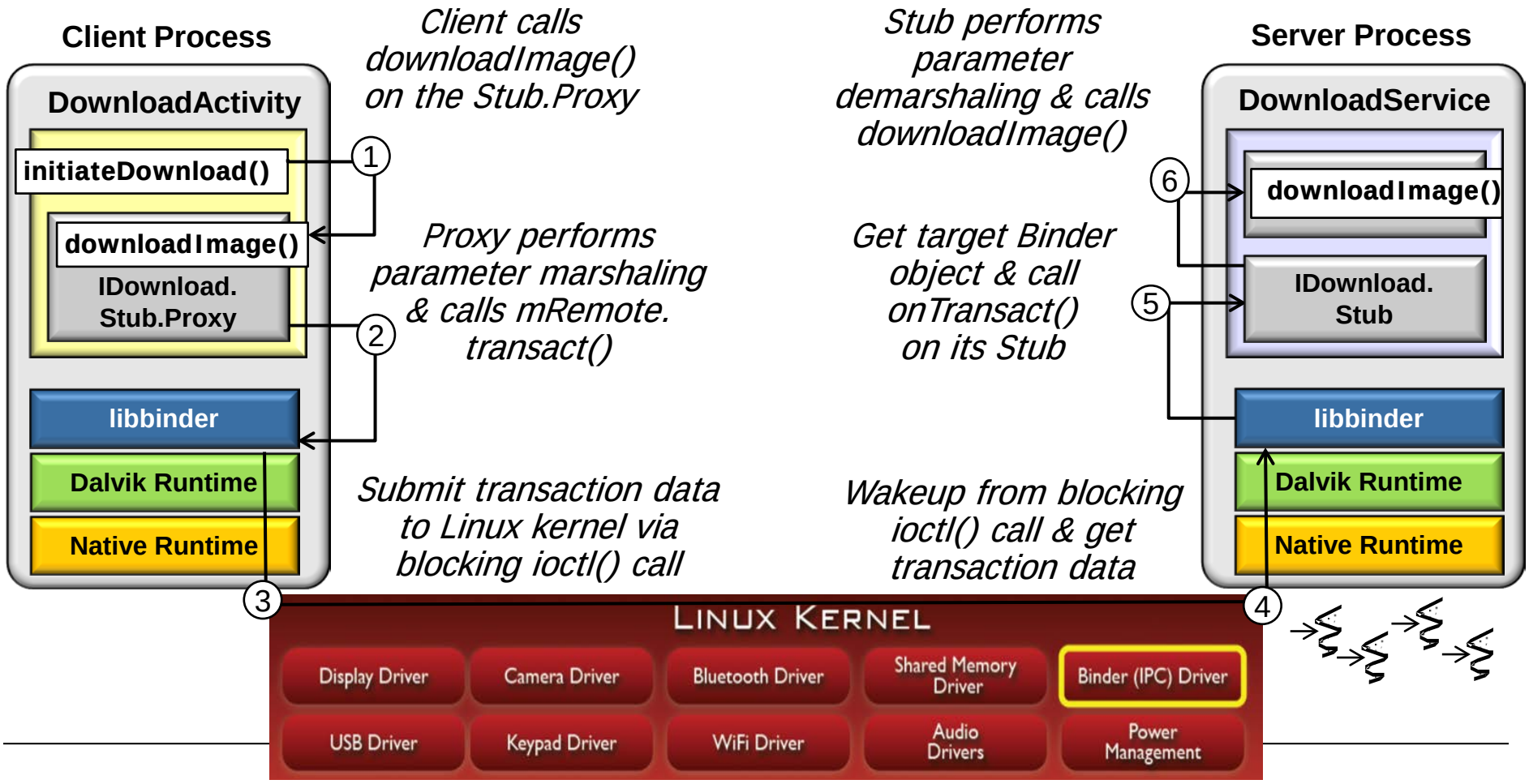
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

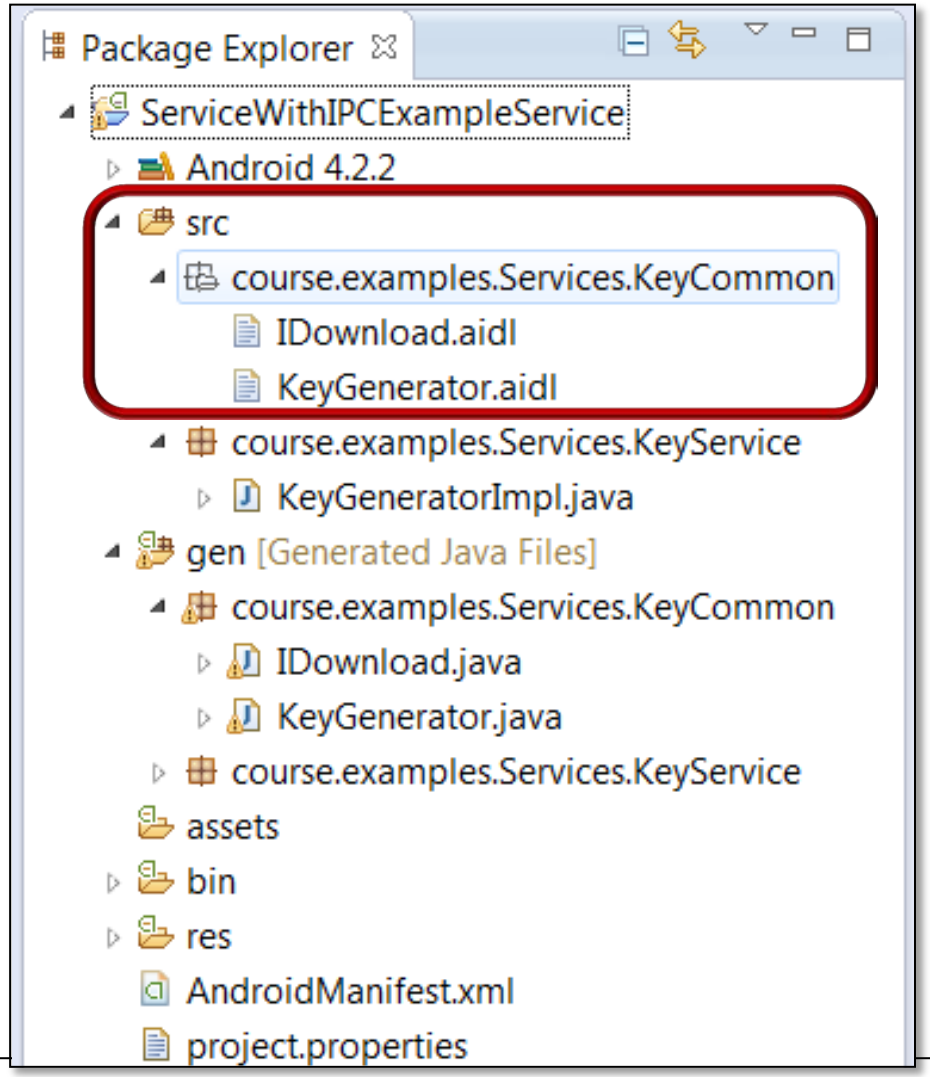
Learning Objectives in this Part of the Module

- Understand how to implement AIDL interfaces via Eclipse



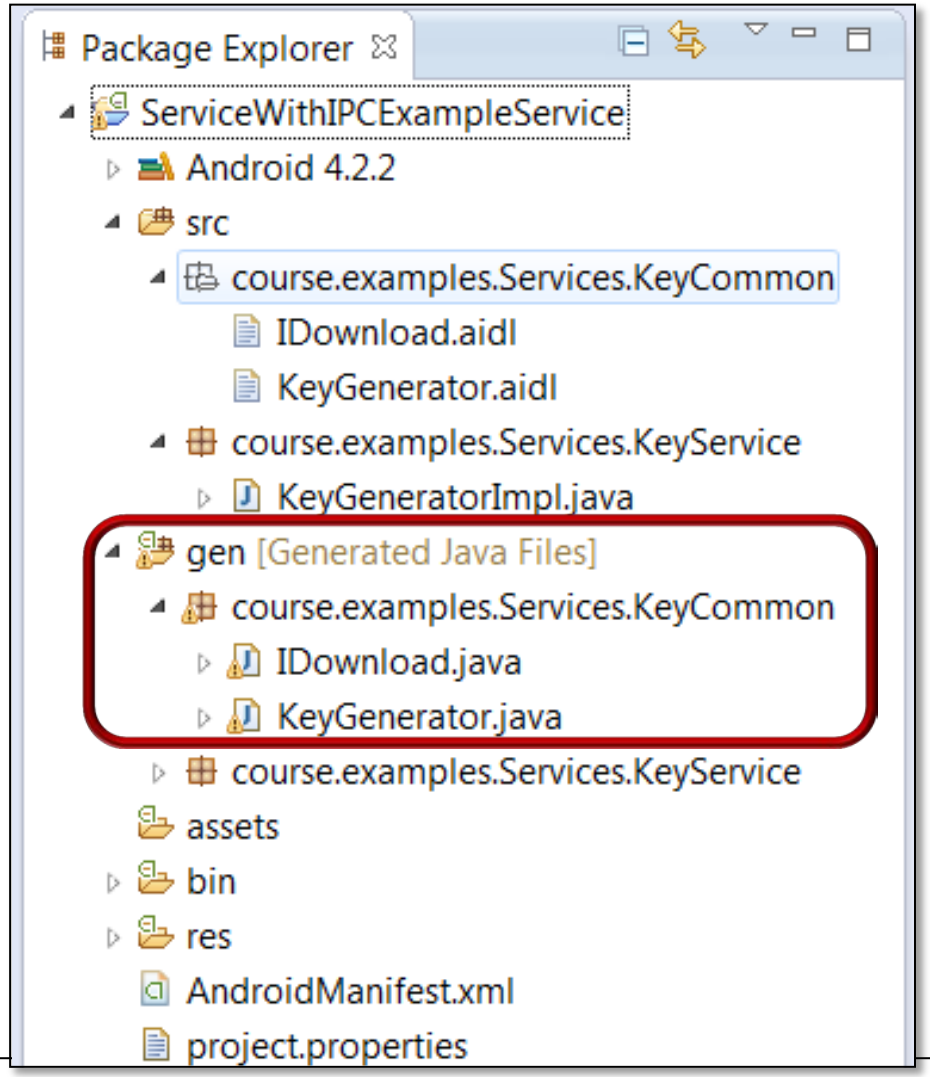
Developing with AIDL on Eclipse

- Each Binder-based service is defined in a separate .aidl file & saved in a **`src`** directory
  - Eclipse ADT automatically calls aidl for each .aidl file it finds in a **`src`** directory



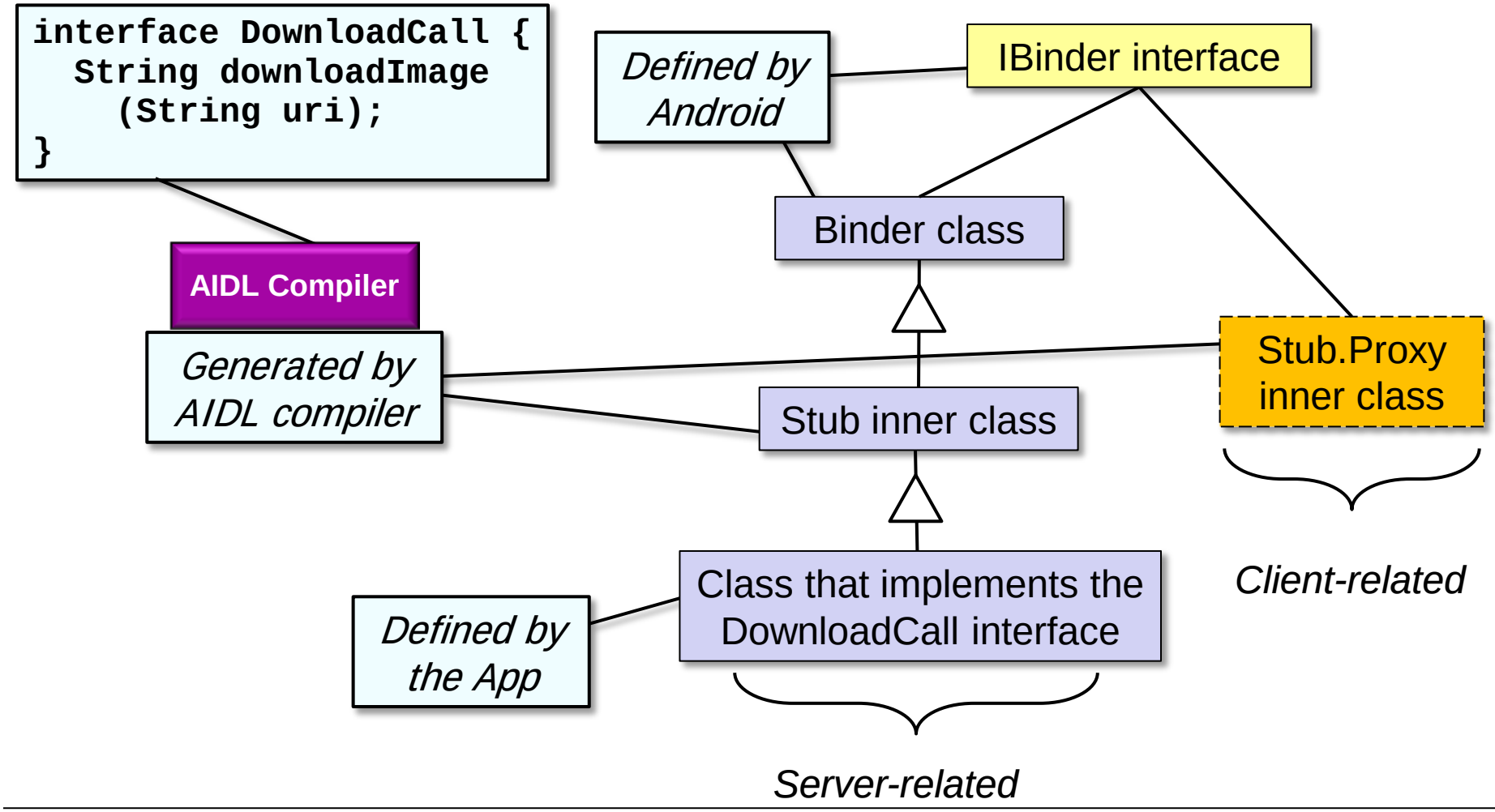
Developing with AIDL on Eclipse

- Each Binder-based service is defined in a separate .aidl file & saved in a **`src`** directory
- The Android aidl build tool extracts a real Java interface from each .aidl file & places it into a *.java file in the **gen** directory
  - This *.java file also contains
    - A generated Stub that extends Android's android.os.Ibinder
    - A Proxy that inherits from the AIDL interface



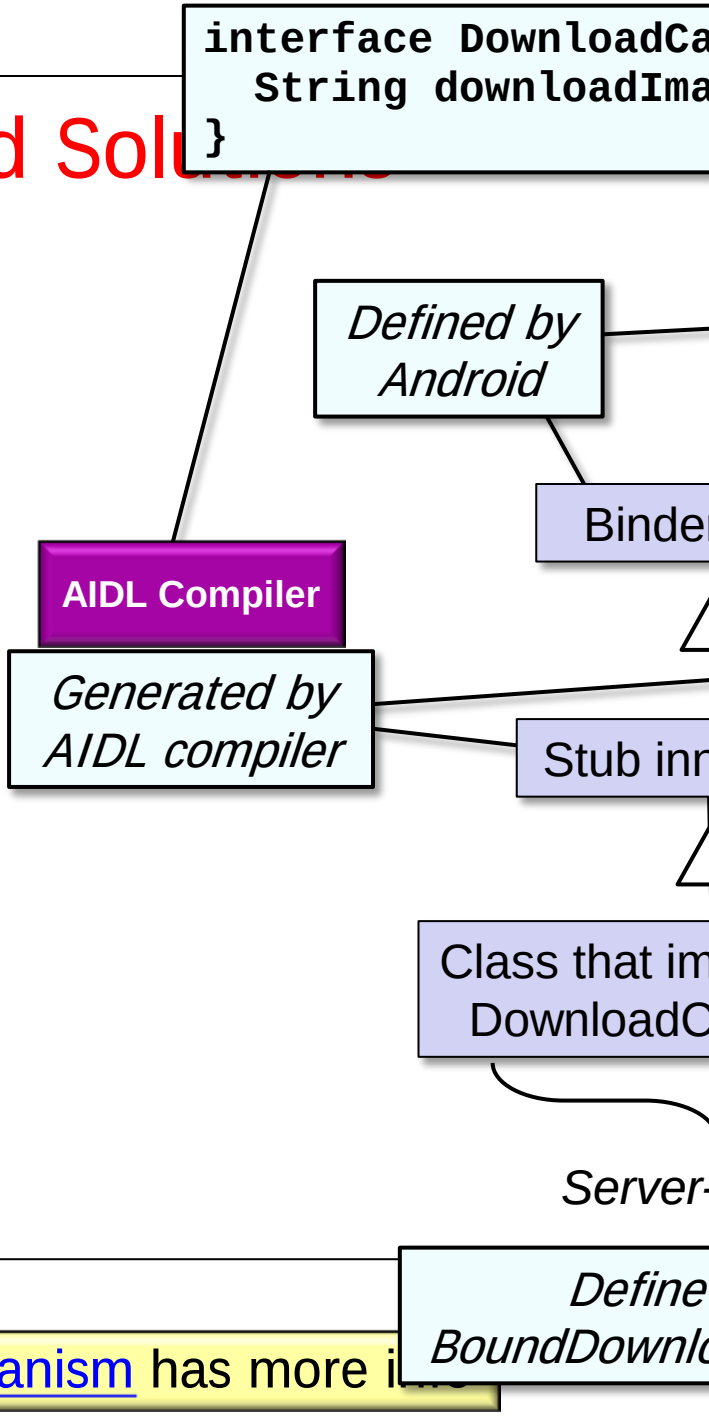
developer.android.com/guide/components/aidl.html has more info

Structure of AIDL-based Solutions



www.slideshare.net/jserv/android-ipc-mechanism has more info

Structure of AIDL-based Solutions



www.slideshare.net/jserv/android-ipc-mechanism has more i

Implementing an AIDL Interface

- Given an auto-generated AIDL stub, you must implement certain methods

```
public interface IDownload extends android.os.Iinterface {
  public static abstract class Stub extends android.os.Binder
                                          implements IDownload {
    ...
    public String downloadImage(String uri)
                              throws android.os.RemoteException;
  }
}
```

  - Either implement downloadImage() directly in the stub or by forwarding the stub to some other implementation method

Implementing an AIDL Interface

- Given an auto-generated AIDL stub, you must implement certain methods
- Implementation steps:
  1. Create a private instance of AIDL-generated Stub class

```
public class DownloadService
                extends Service {
  private IDownload.Stub
    binder = null;
  public void onCreate() {
    binder = new IDownload.Stub(){
      public String downloadImage
        (String uri)  {
         ...
      }
  };

  public IBinder onBind
                    (Intent intent)
  { return this.binder; }
}
```

Implementing an AIDL Interface

- Given an auto-generated AIDL stub, you must implement certain methods
- Implementation steps:
  1. Create a private instance of AIDL-generated Stub class
  2. Implement Java methods for each method in the AIDL file

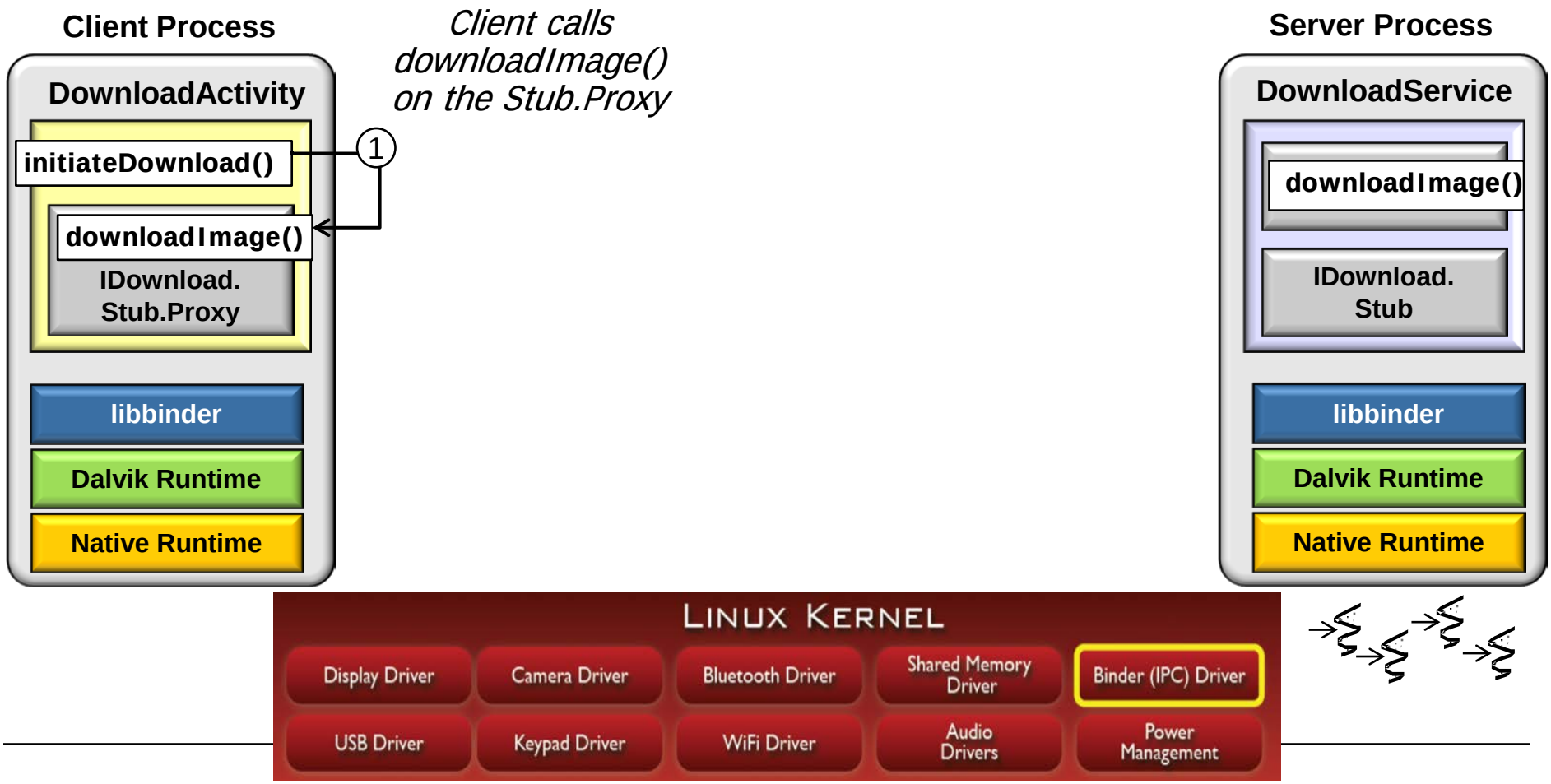
```
public class DownloadService
             extends Service {
  private IDownload.Stub
    binder = null;
  public void onCreate() {
    binder = new IDownload.Stub(){
      public String downloadImage
        (String uri)  {
         ...
      }
  };

  public IBinder onBind
                   (Intent intent)
  { return this.binder; }
}
```

Implementing an AIDL Interface

- Given an auto-generated AIDL stub, you must implement certain methods
- Implementation steps:
  1. Create a private instance of AIDL-generated Stub class
  2. Implement Java methods for each method in the AIDL file
  3. Return this private instance from your onBind() method in the Service subclass

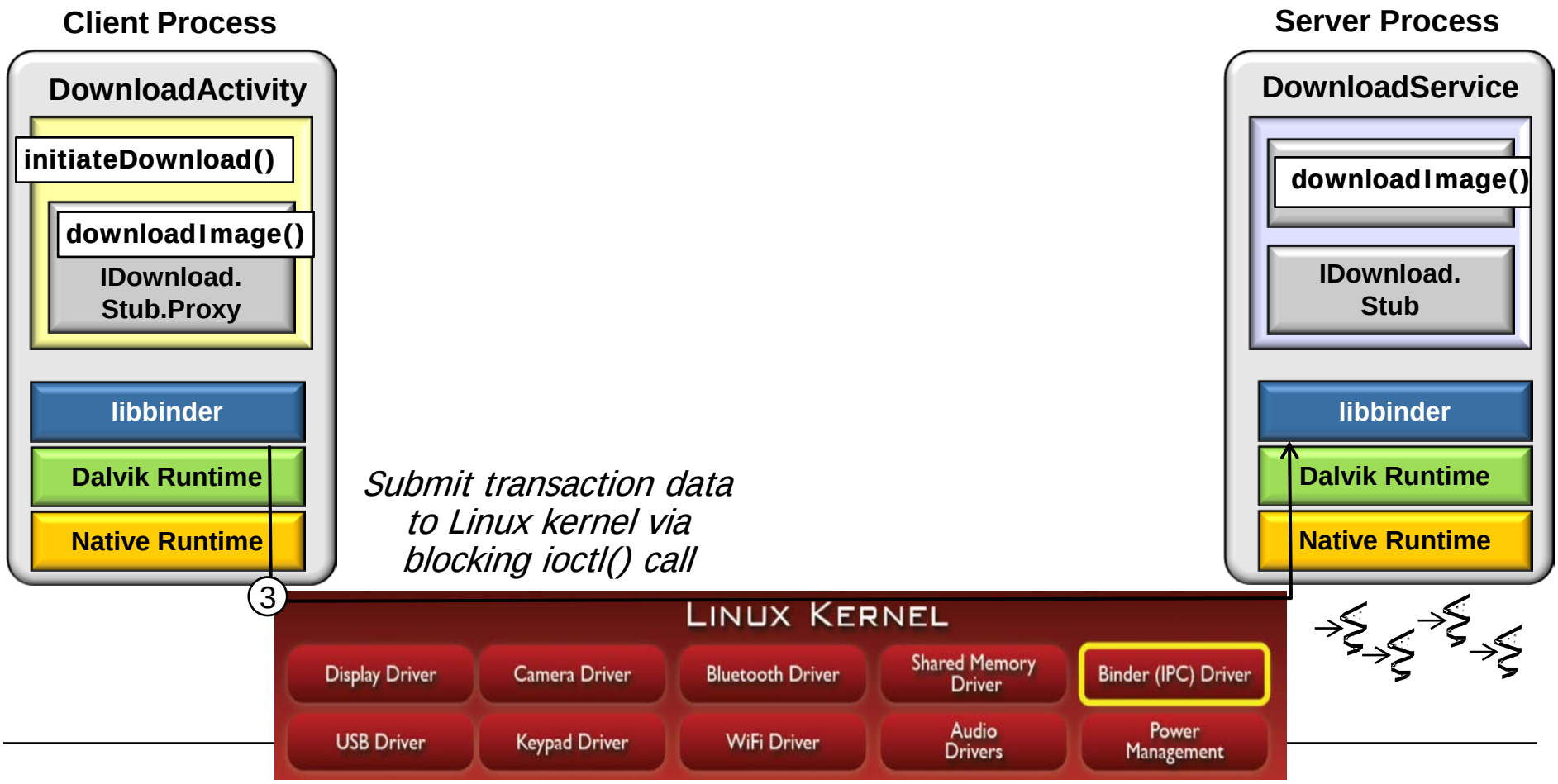
```
public class DownloadService
             extends Service {
  private IDownload.Stub
    binder = null;
  public void onCreate() {
    binder = new IDownload.Stub(){
      public String downloadImage
        (String uri)  {
         ...
      }
  };

  public IBinder onBind
                   (Intent intent)
  { return this.binder; }
}
```

Synchronous Two-way Communication w/AIDL

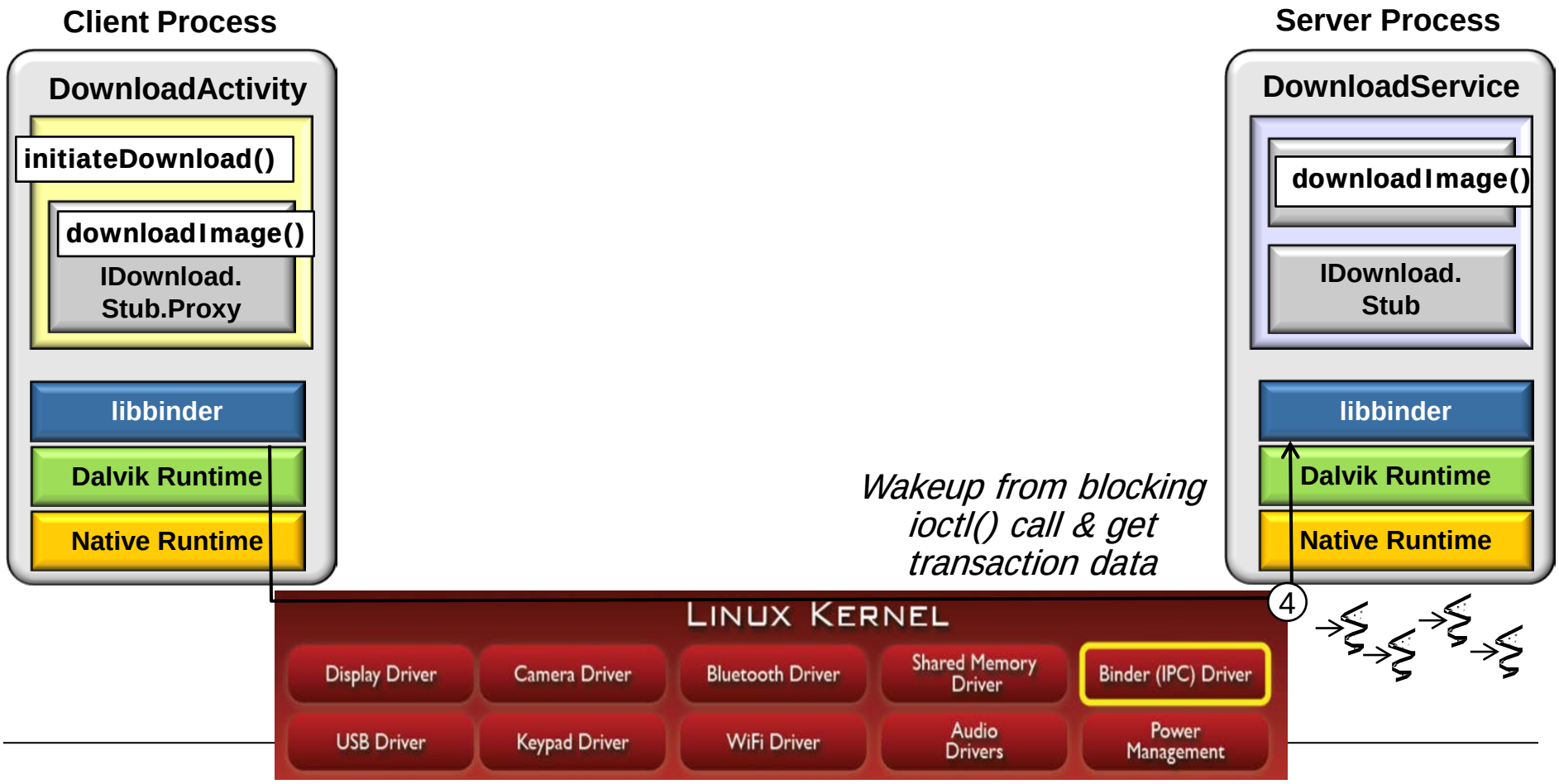


Synchronous Two-way Communication w/AIDL

Synchronous Two-way Communication w/AIDL

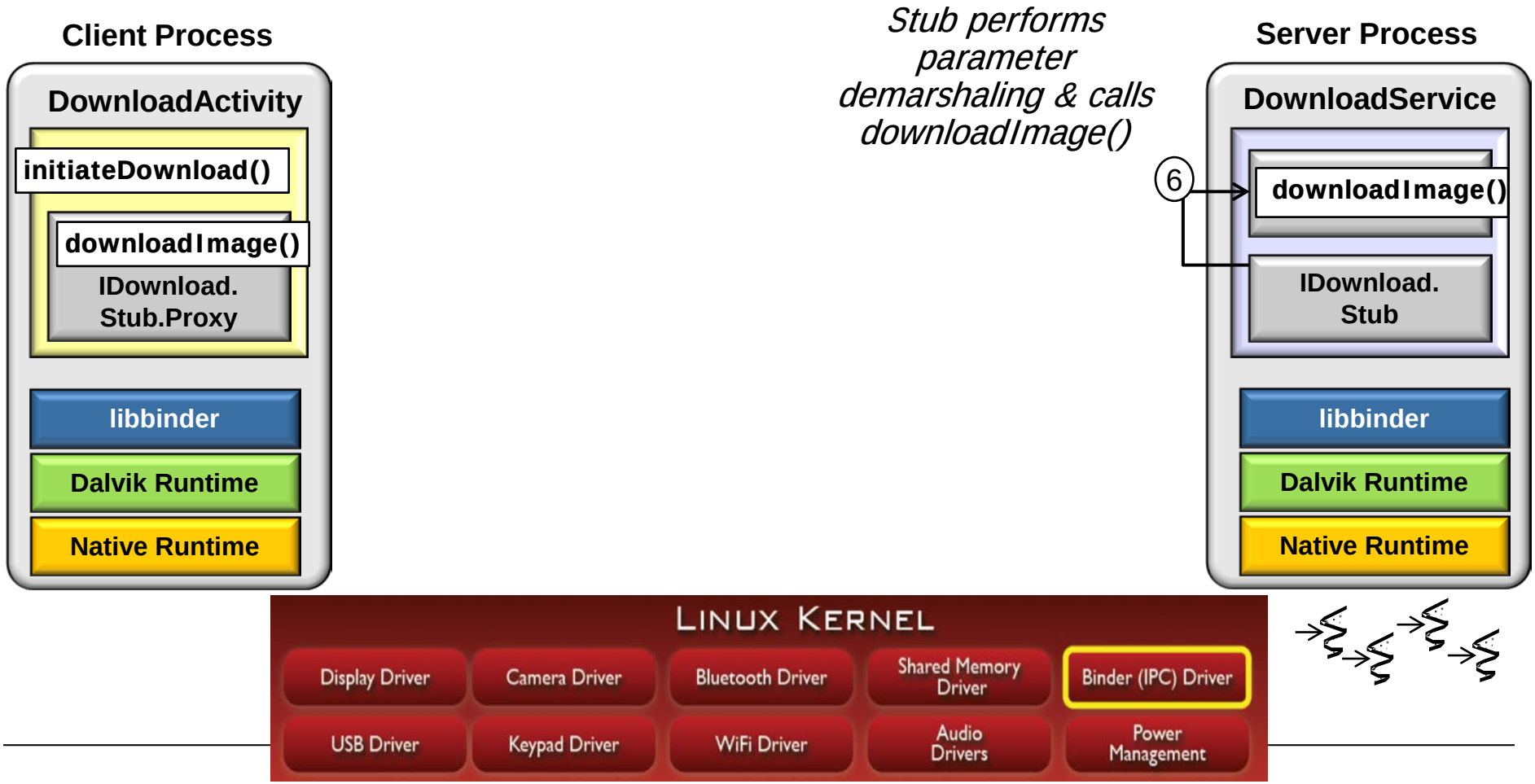


Synchronous Two-way Communication w/AIDL



Synchronous Two-way Communication w/AIDL

Synchronous Two-way Communication w/AIDL

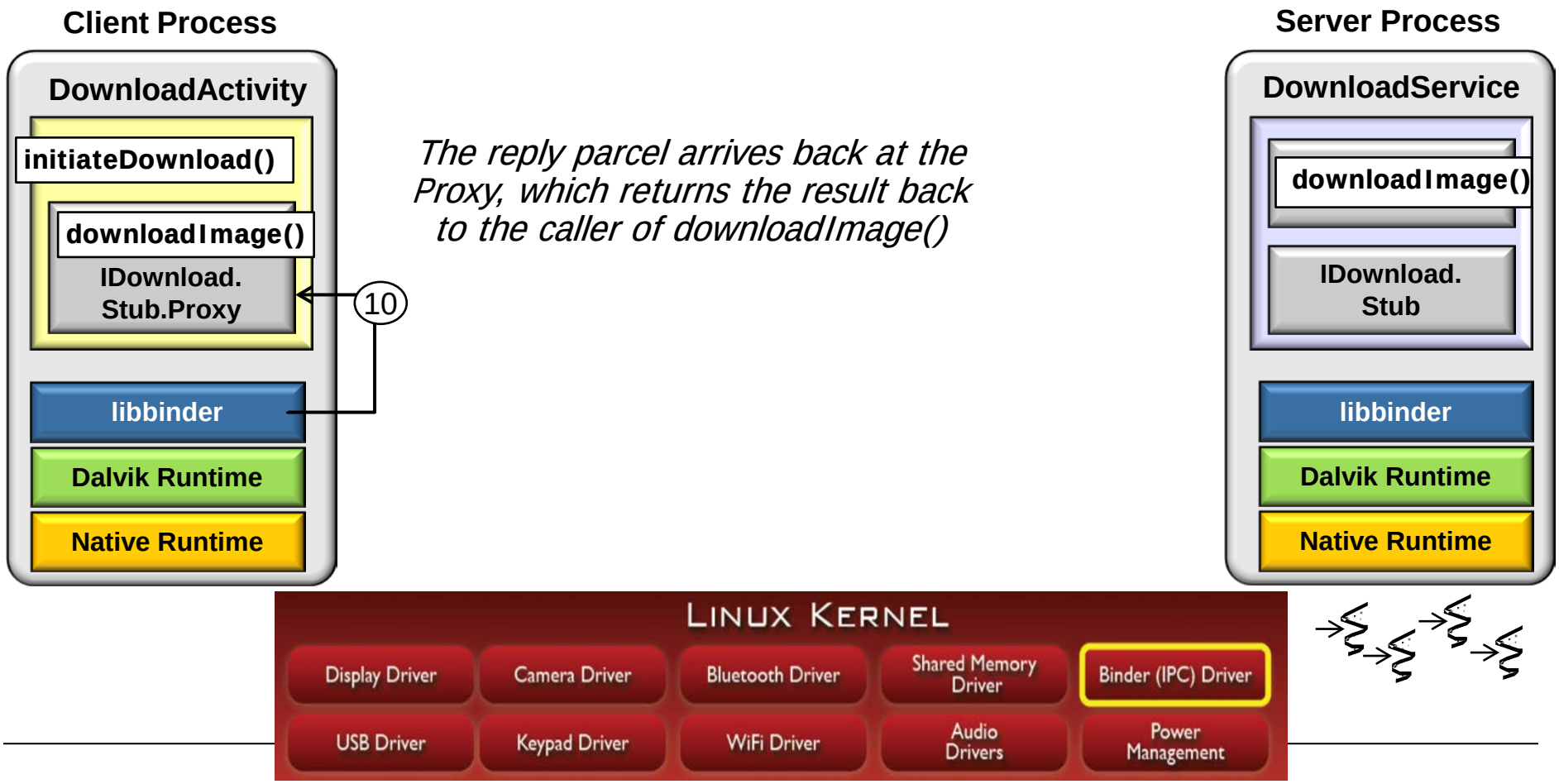


Synchronous Two-way Communication w/AIDL

Synchronous Two-way Communication w/AIDL

Synchronous Two-way Communication w/AIDL

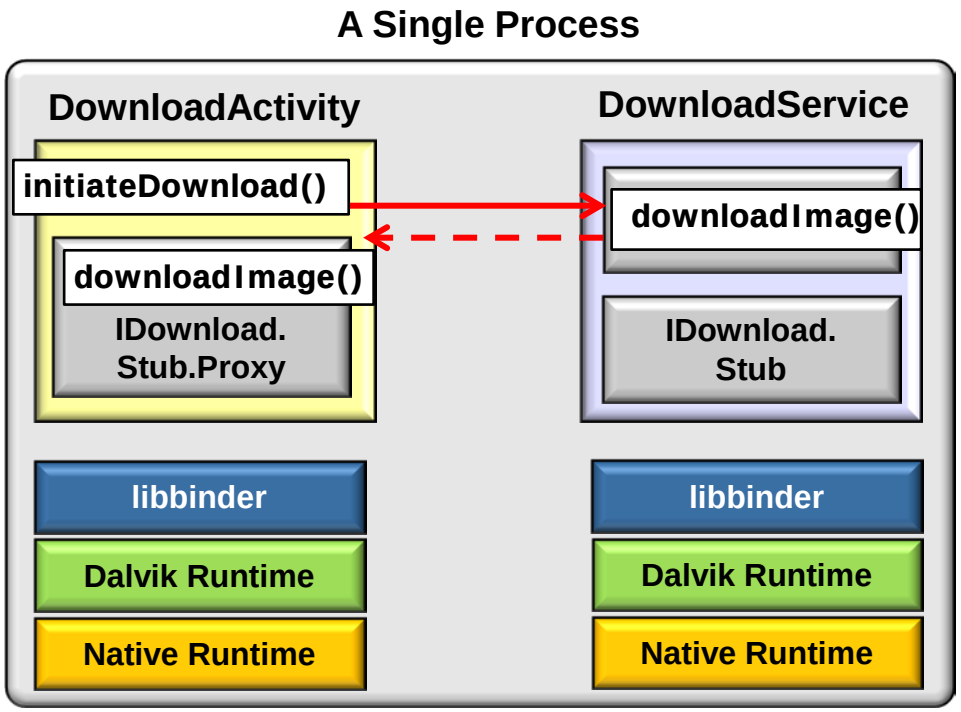
Synchronous Two-way Communication w/AIDL



thenewcircle.com/s/post/1340/Deep_Dive_Into_Binder_Presentation.htm

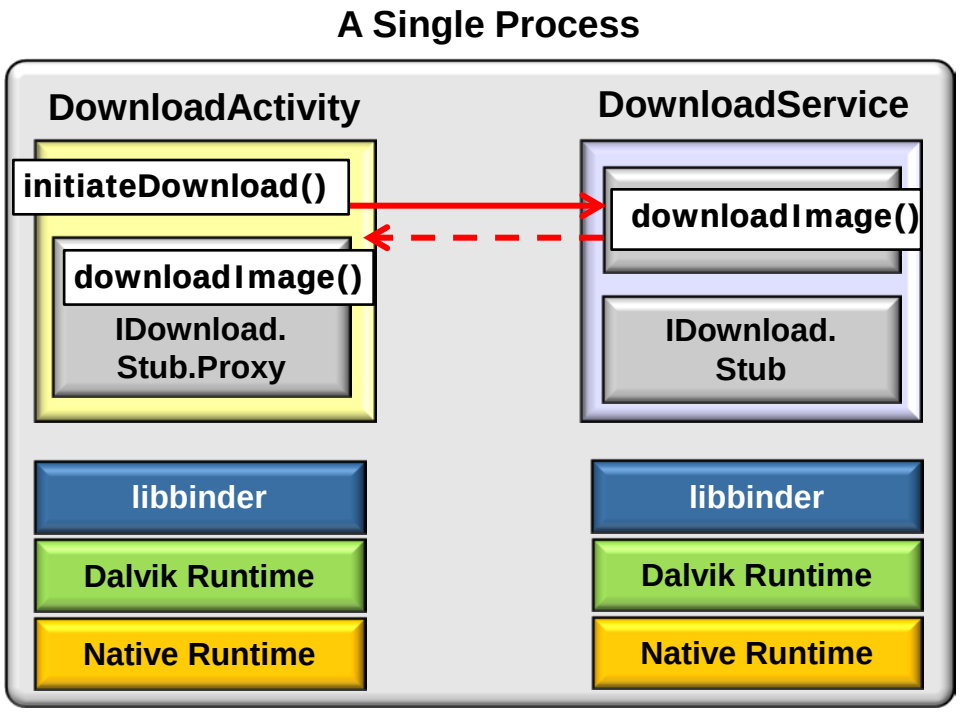
AIDL & Binder RPC Call Semantics

- Calls made from a local process are executed in the same thread that makes the call
  - If this is the main UI thread, that thread continues to execute in the AIDL interface



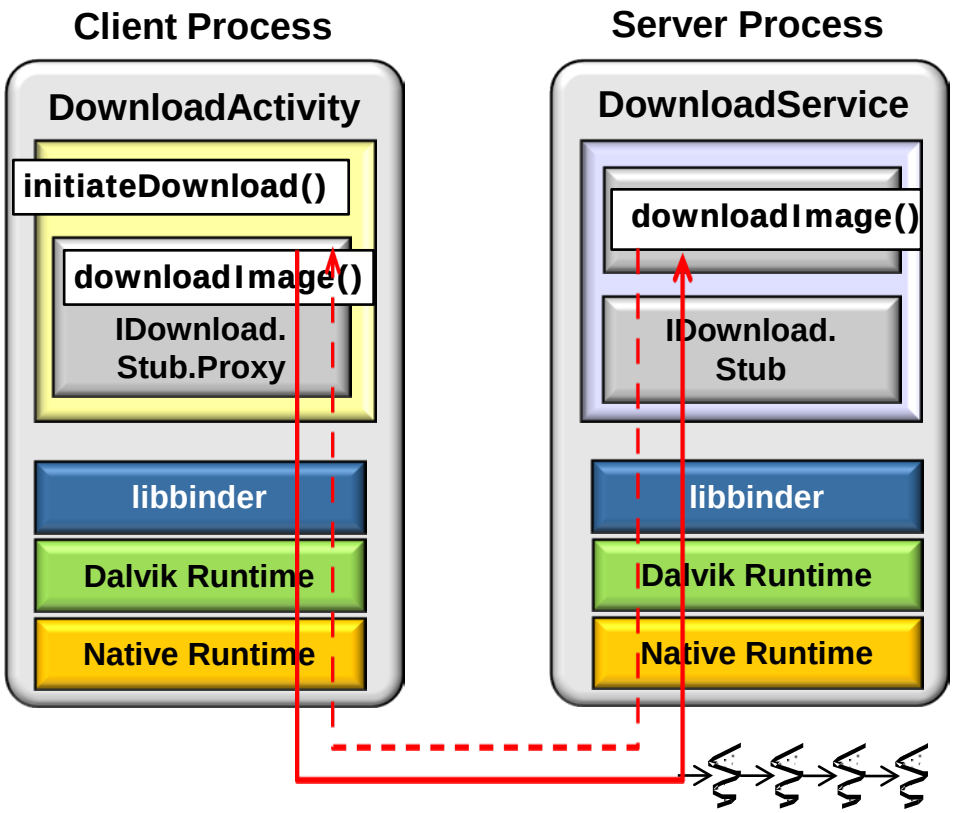
AIDL & Binder RPC Call Semantics

- Calls made from a local process are executed in the same thread that makes the call
  - If this is the main UI thread, that thread continues to execute in the AIDL interface
  - If it is another thread, that is the one that executes the code in the Service



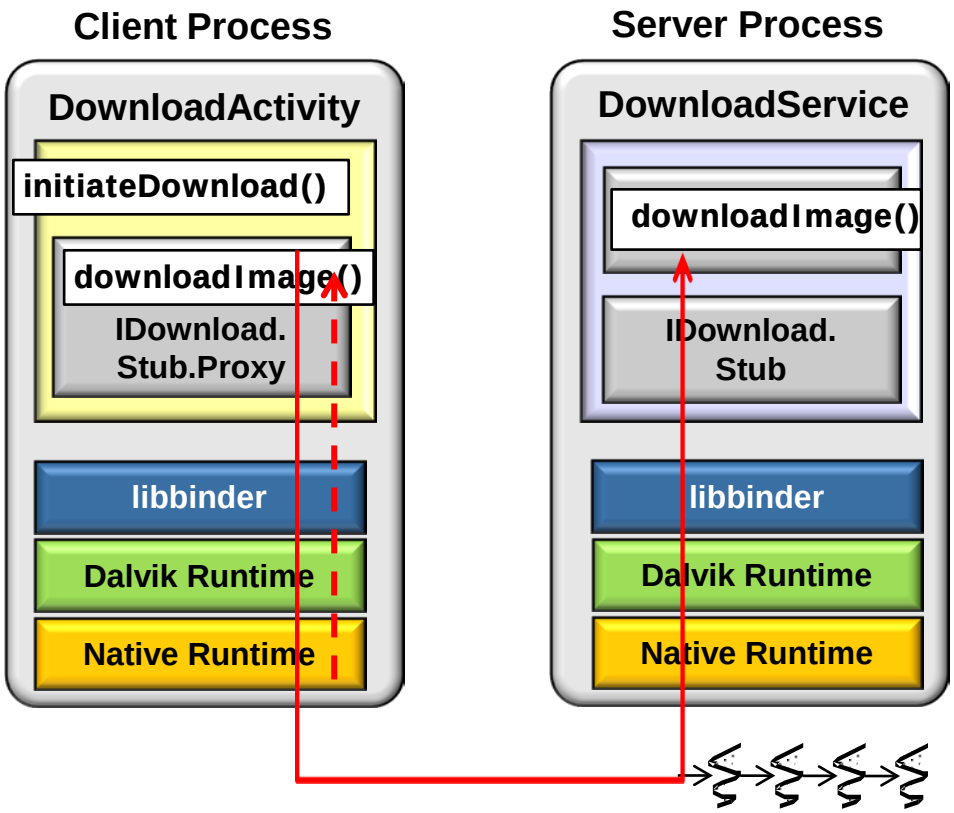
AIDL & Binder RPC Call Semantics

- Calls made from a local process are executed in the same thread that makes the call
- Calls from a remote process are dispatched from a thread pool the platform maintains inside of a process
  - An implementation of an AIDL interface must therefore be completely thread-safe



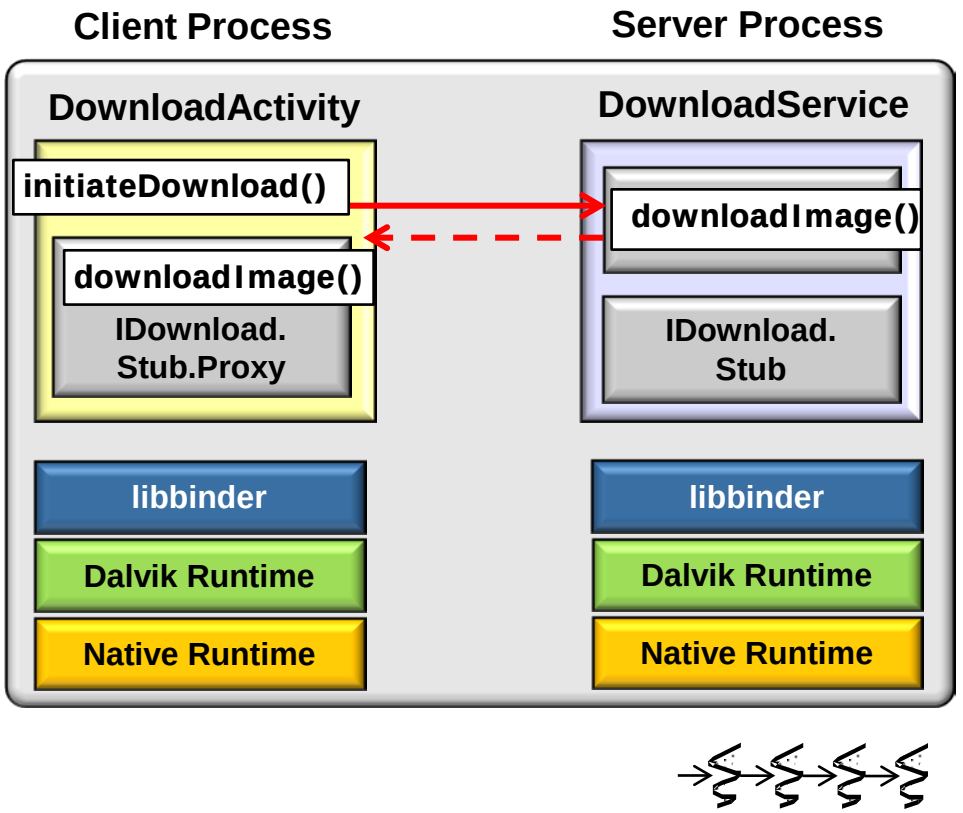
AIDL & Binder RPC Call Semantics

- Calls made from a local process are executed in the same thread that makes the call
- Calls from a remote process are dispatched from a thread pool the platform maintains inside of a process
- The **oneway** keyword modifies the behavior of remote calls
  - When used, a remote call does not block—it simply sends the transaction data & returns immediately



AIDL & Binder RPC Call Semantics

- Calls made from a local process are executed in the same thread that makes the call
- Calls from a remote process are dispatched from a thread pool the platform maintains inside of a process
- The **oneway** keyword modifies the behavior of remote calls
  - When used, a remote call does not block—it simply sends the transaction data & returns immediately
  - If **oneway** is used with a local call, the call is still synchronous
    - But no results are returned



developer.android.com/guide/components/aidl.html

Summary

- AIDL is an interface definition language used to generate code that enables two processes on an Android device to interact using IPC
  - If code in a process calls methods on an object in another process, AIDL can generate code to (de)marshal parameters passed between processes

Summary

- AIDL is an interface definition language used to generate code that enables two processes on an Android device to interact using IPC
- AIDL is interface-based, similar to CORBA & Java, but lighter weight
  - It uses a proxy to pass values between a client & Bound Service

Summary

- AIDL is an interface definition language used to generate code that enables two processes on an Android device to interact using IPC
- AIDL is interface-based, similar to CORBA, but lighter weight
- There are hundreds of *.aidl files used in Android
- AIDL is often used to mediate two-way communication between Activity & Service or Service & Service