Android Services & Security: Activity & Service Communication

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

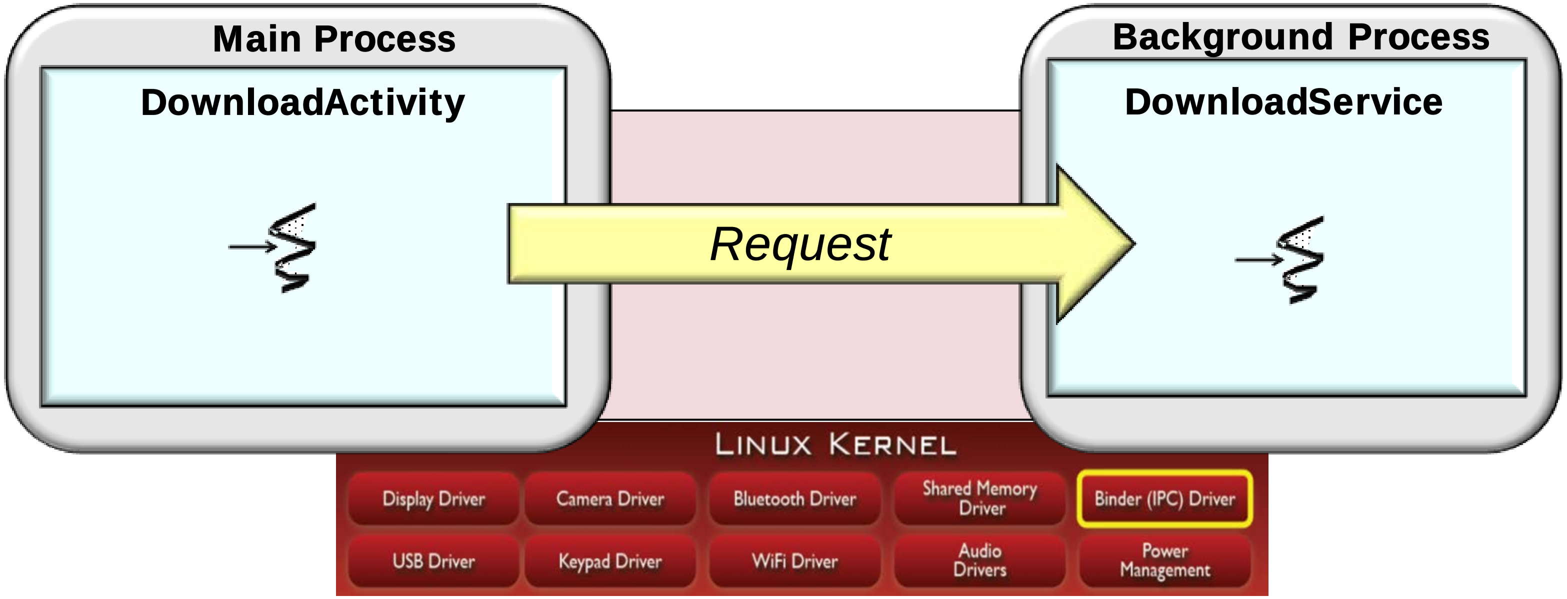
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

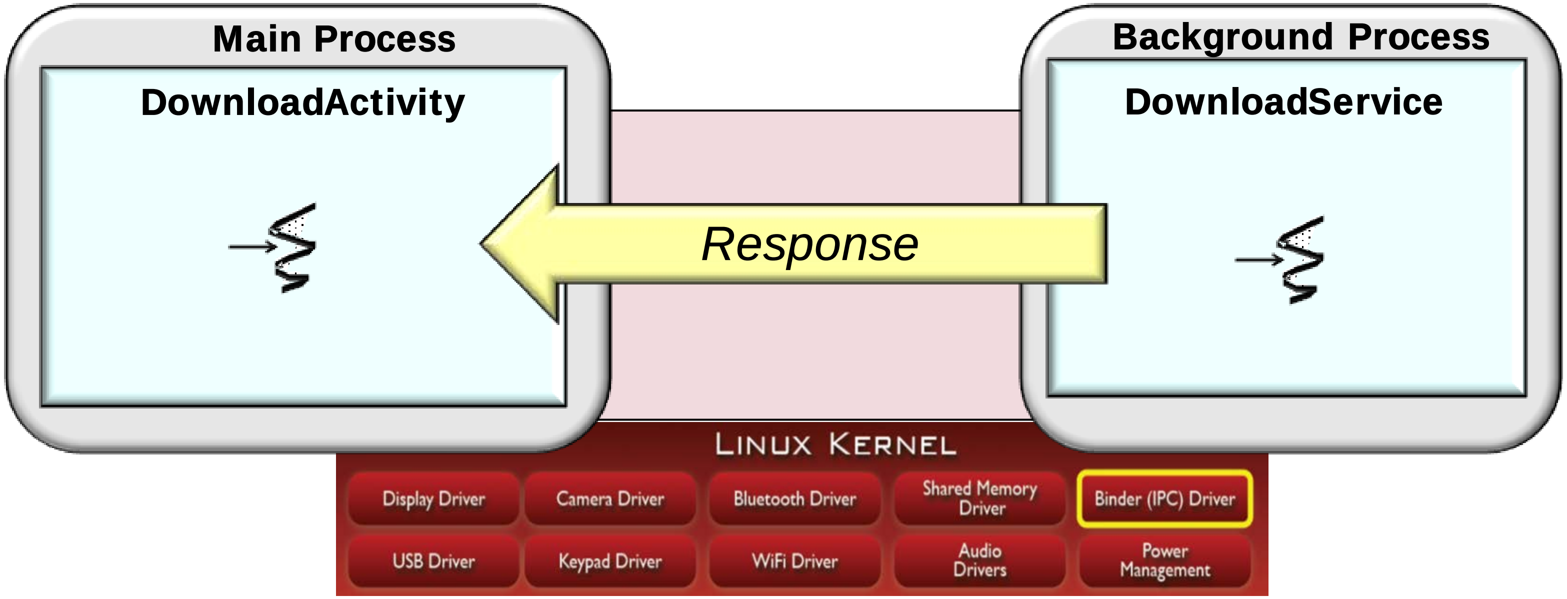
Learning Objectives in this Part of the Module

- Understand various mechanisms that Activities & Services use to communicate



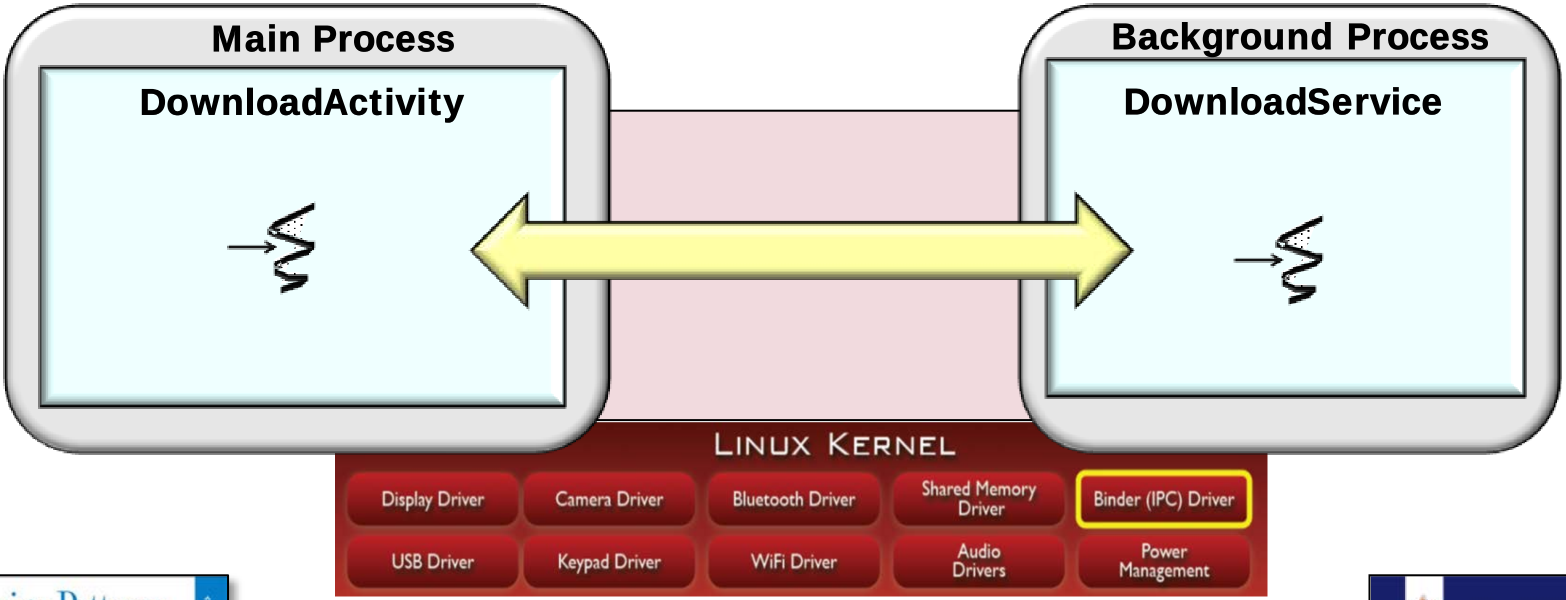
Learning Objectives in this Part of the Module

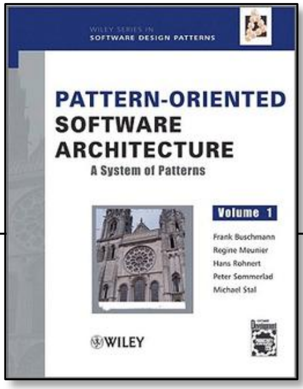
- Understand various mechanisms that Activities & Services use to communicate

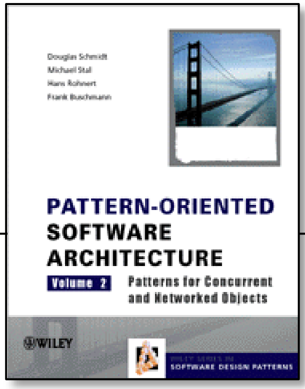


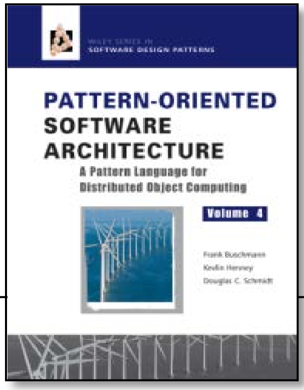
Learning Objectives in this Part of the Module

- Understand various mechanisms that Activities & Services use to communicate
- Recognize the common patterns used to implement communication with Services



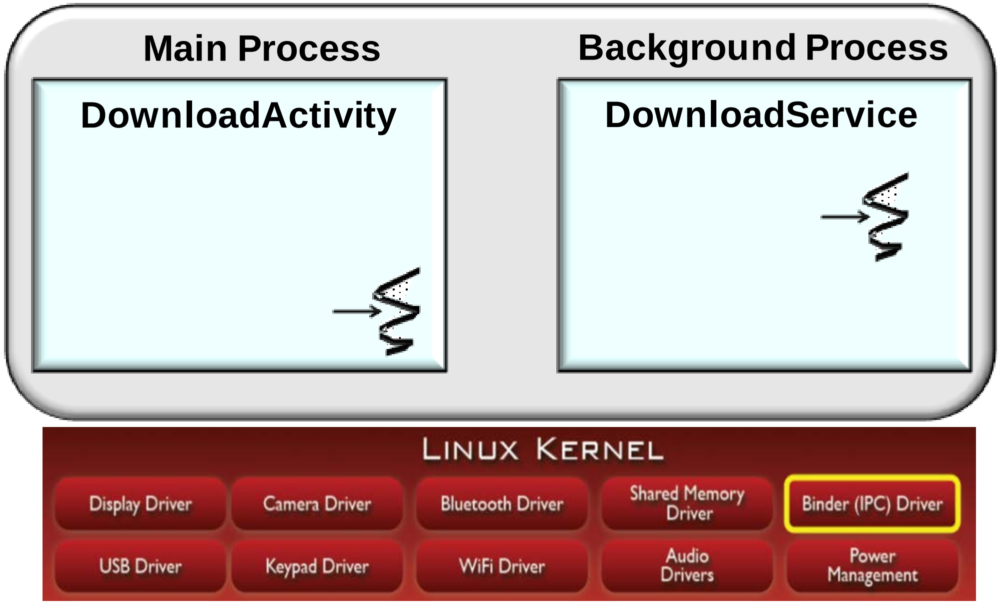



PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY


PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Object Computing
Volume 4

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients



Main Process

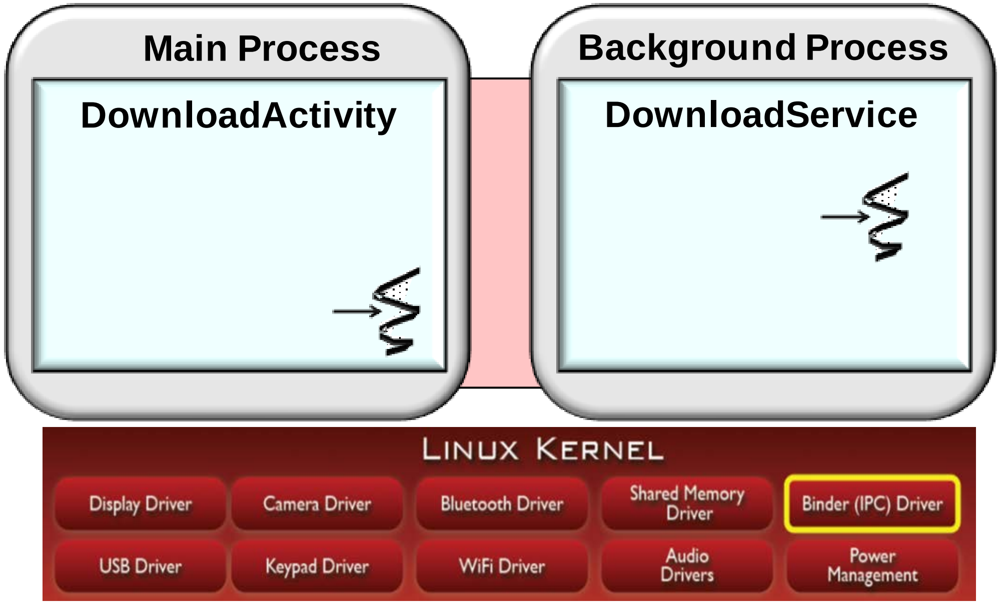
DownloadActivity

Background Process

DownloadService

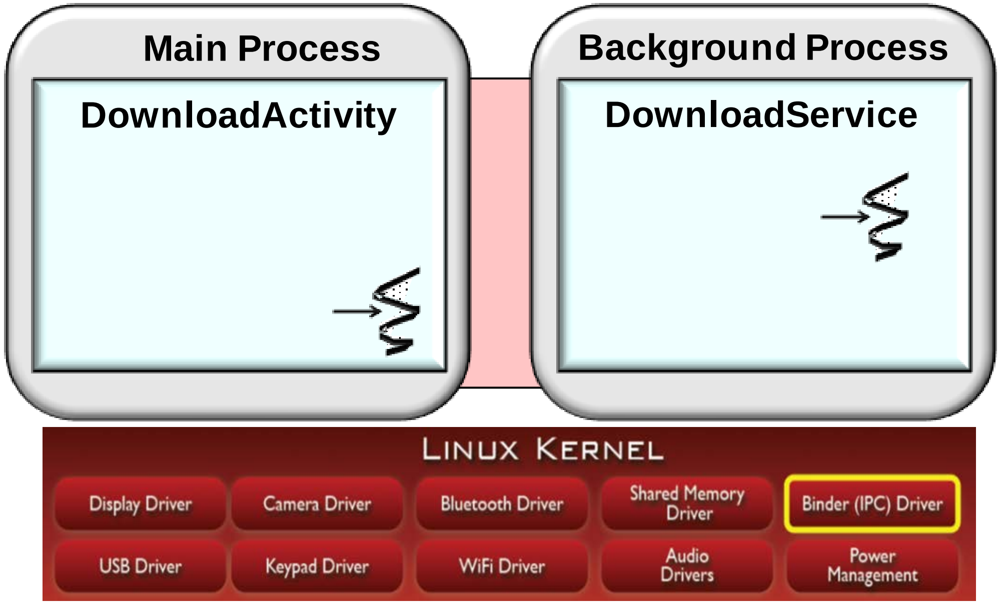
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients



Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml

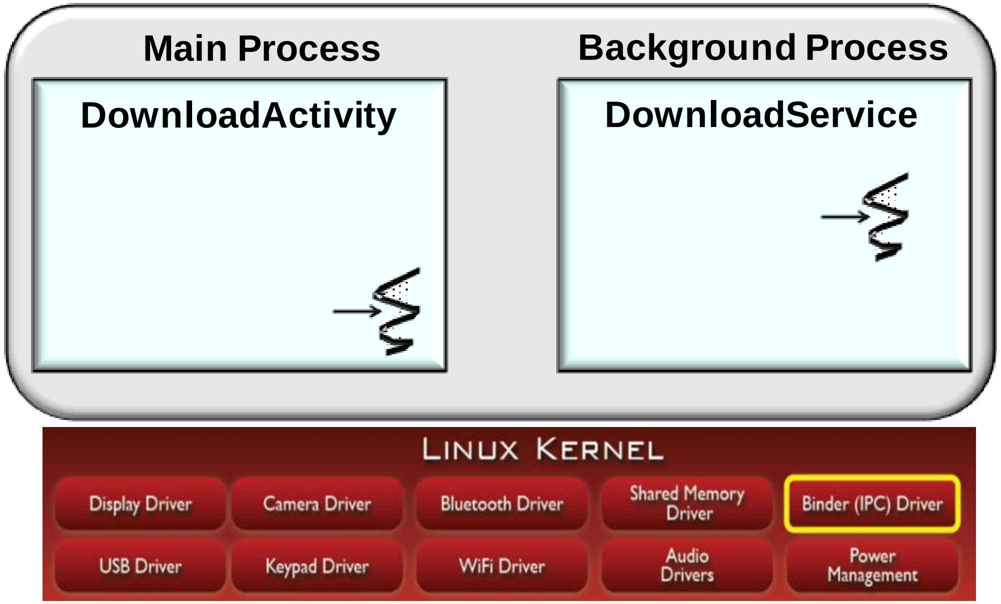


```
<service
    android:enabled
      =["true" | "false"]
    android:exported
      =["true" | "false"]
    android:icon="drawable resource"
    android:isolatedProcess=["true" | "false"]
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
    ...
</service>
```

developer.android.com/guide/topics/manifest/service-element.html has more

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml



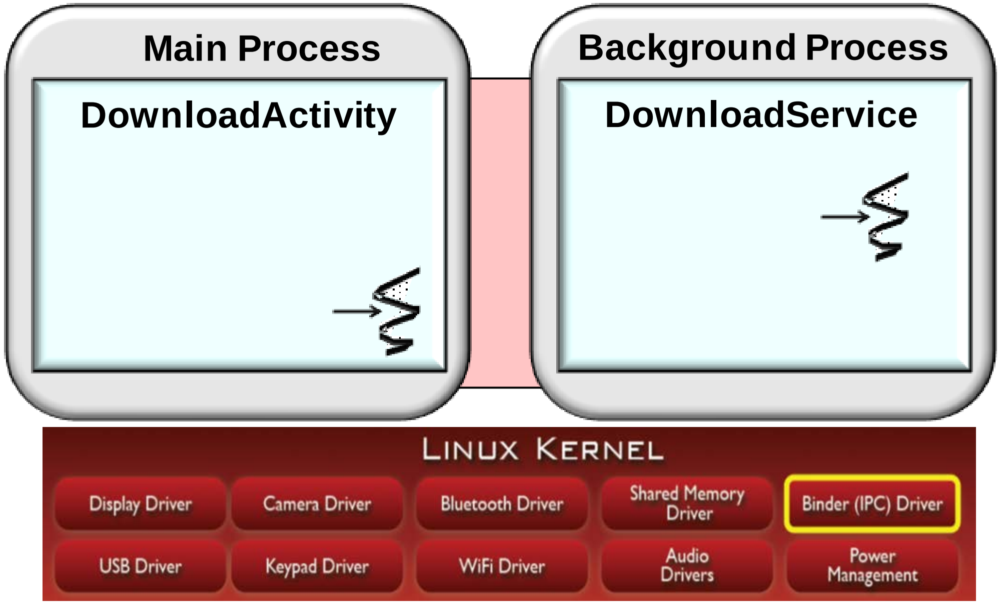
AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"

                      />
...
```

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml

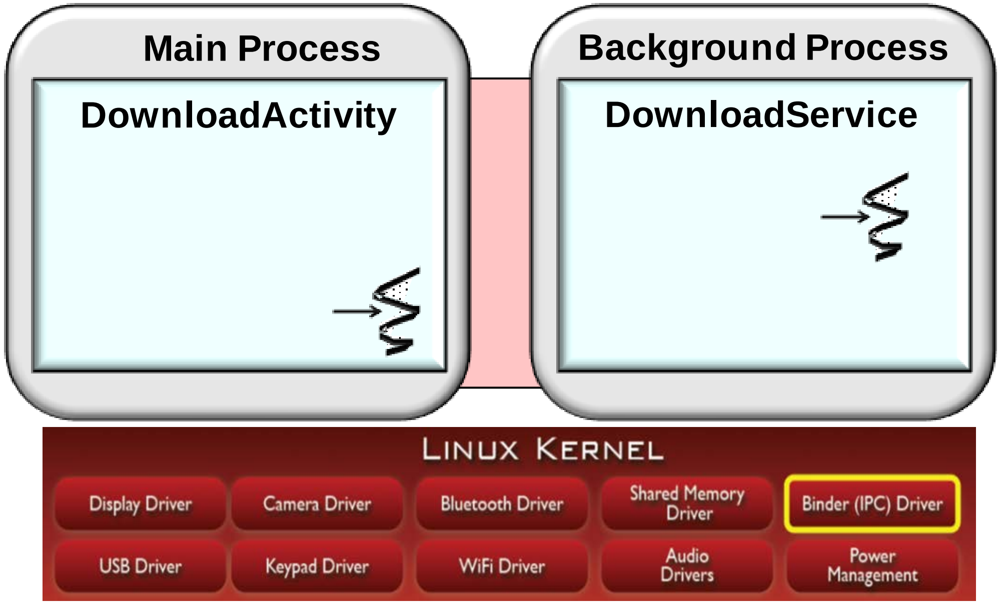


AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"
         android:process=
         ":myProcess"/>
...
```

Overview of Service Deployment Models

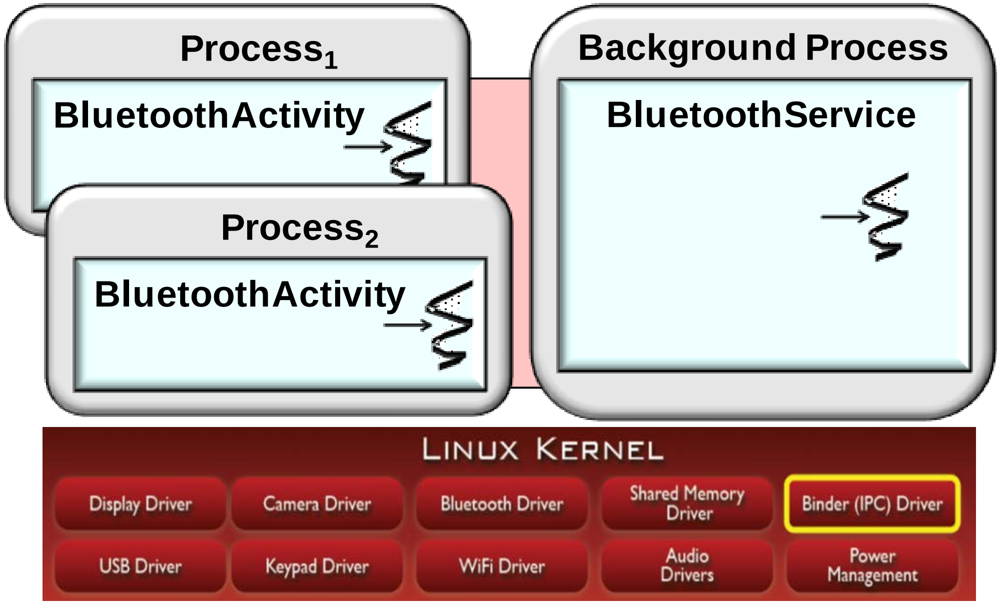
- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process



See www.vogella.com/tutorials/AndroidServices/article.html#service_advice

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes

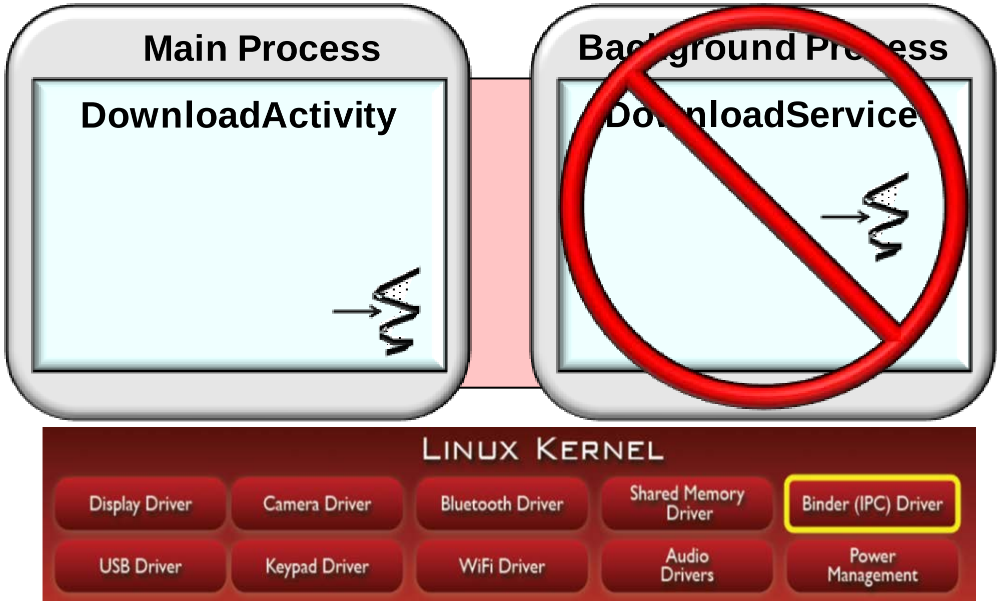


```
<service android:process="@string/process"
         android:name=".opp.BluetoothOppService"
         android:permission=
            "android.permission.ACCESS_BLUETOOTH_SHARE" />
```

See packages/apps/Bluetooth/AndroidManifest.xml

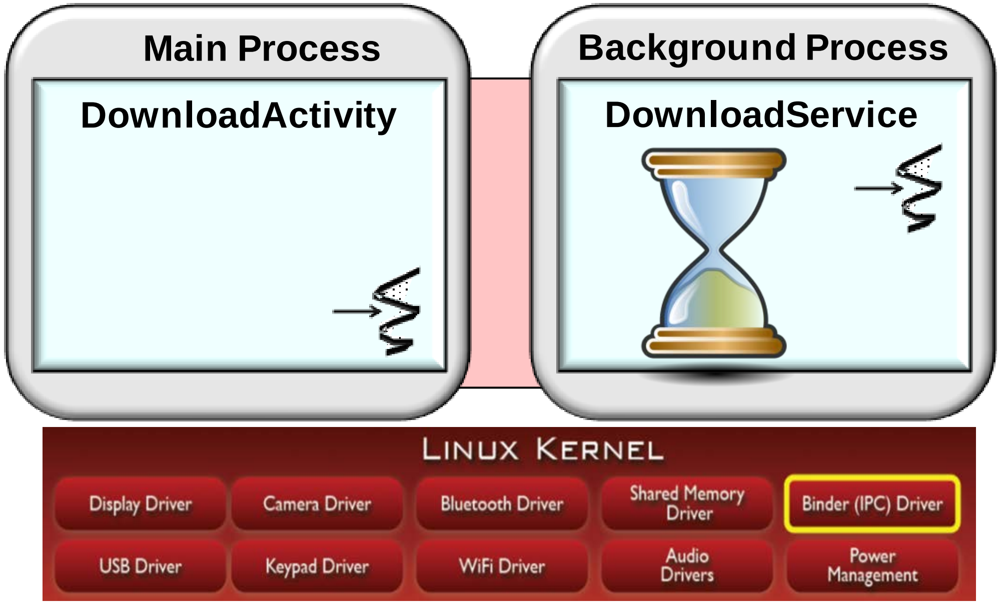
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes
  - Giving a Service its own address space can make applications more robust if failures or hangs occur



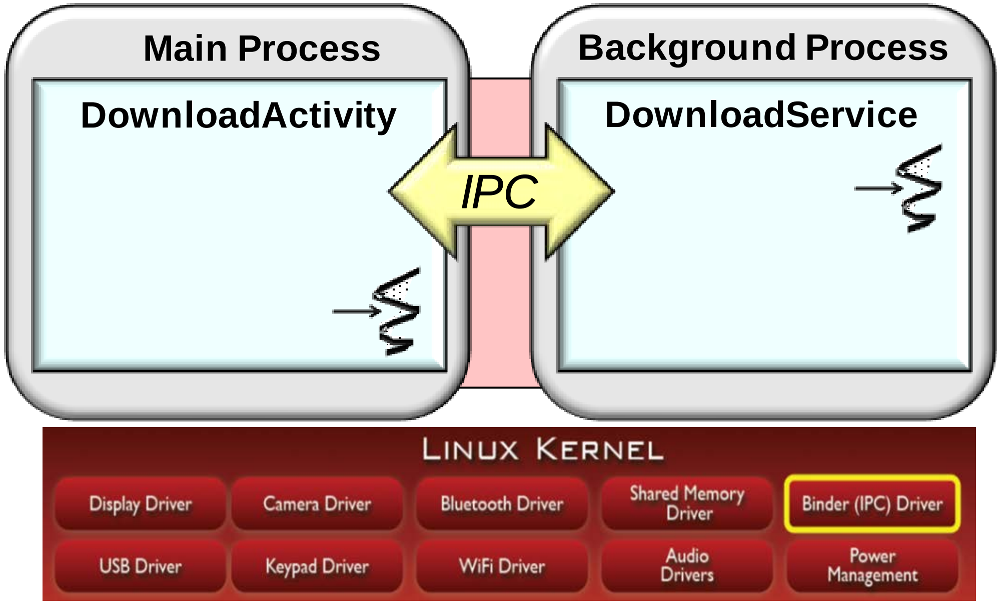
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes
  - Giving a Service its own address space can make applications more robust if failures or hangs occur
  - Garbage collection of the virtual machine in a separate Service process doesn't affect the Application process



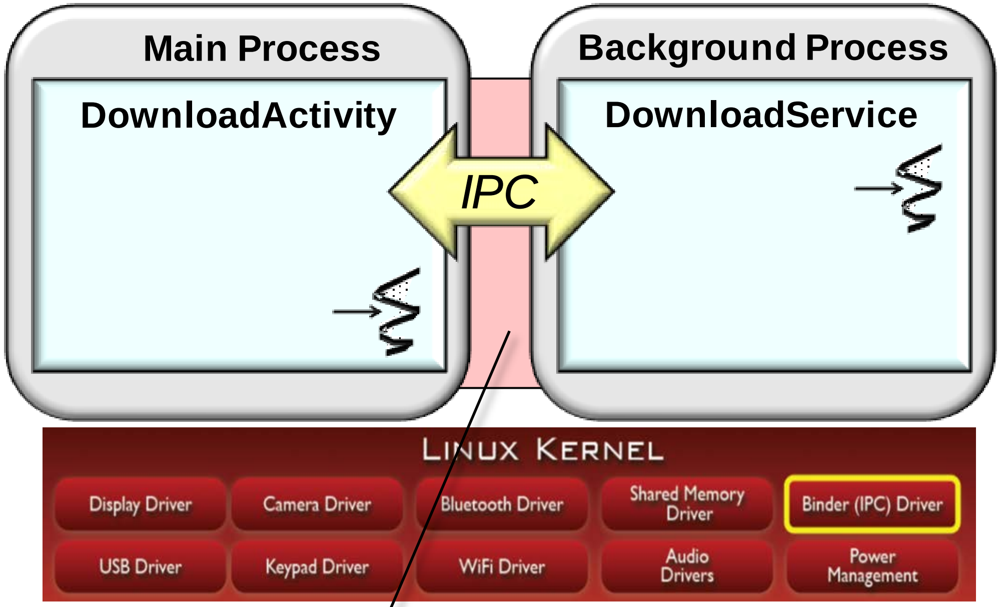
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes



Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - The Android Binder RPC framework underlies the various IPC mechanisms

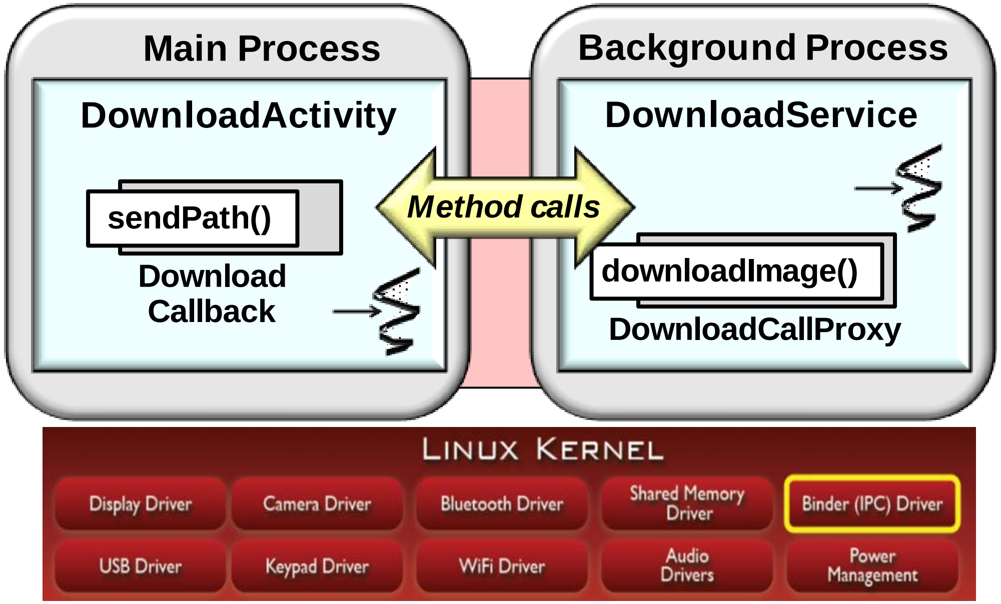


The Binder supports two-way or one-way client-service communication models

See elinux.org/Android_Binder for more on Binder

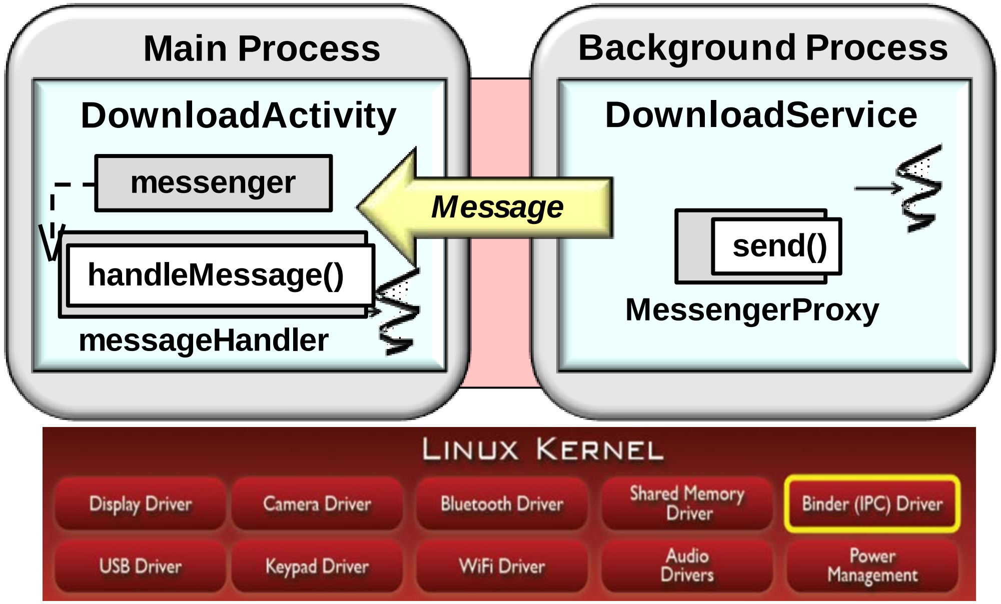
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - The Android Binder RPC framework underlies the various IPC mechanisms



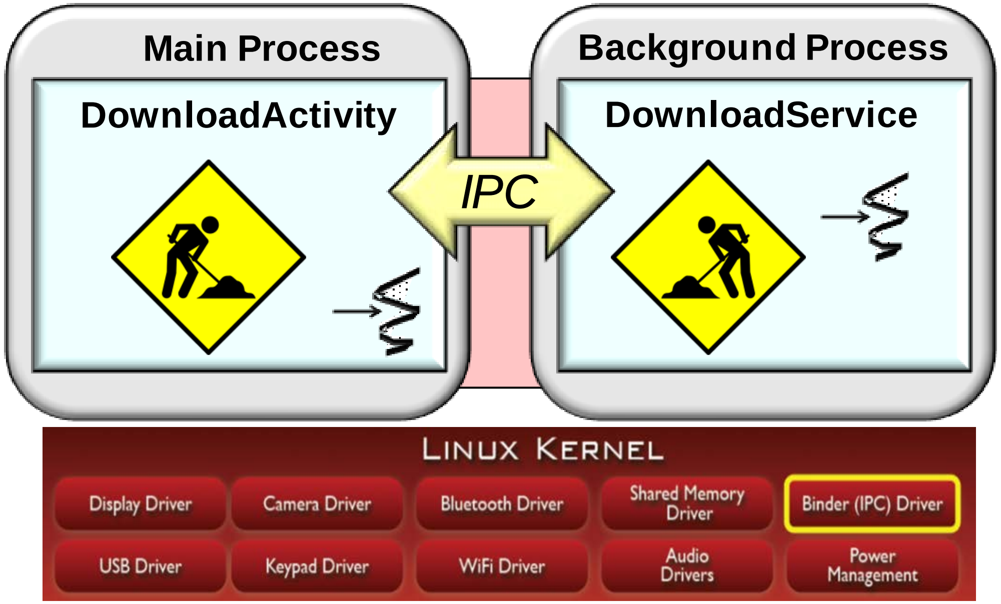
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - The Android Binder RPC framework underlies the various IPC mechanisms



Overview of Service Deployment Models

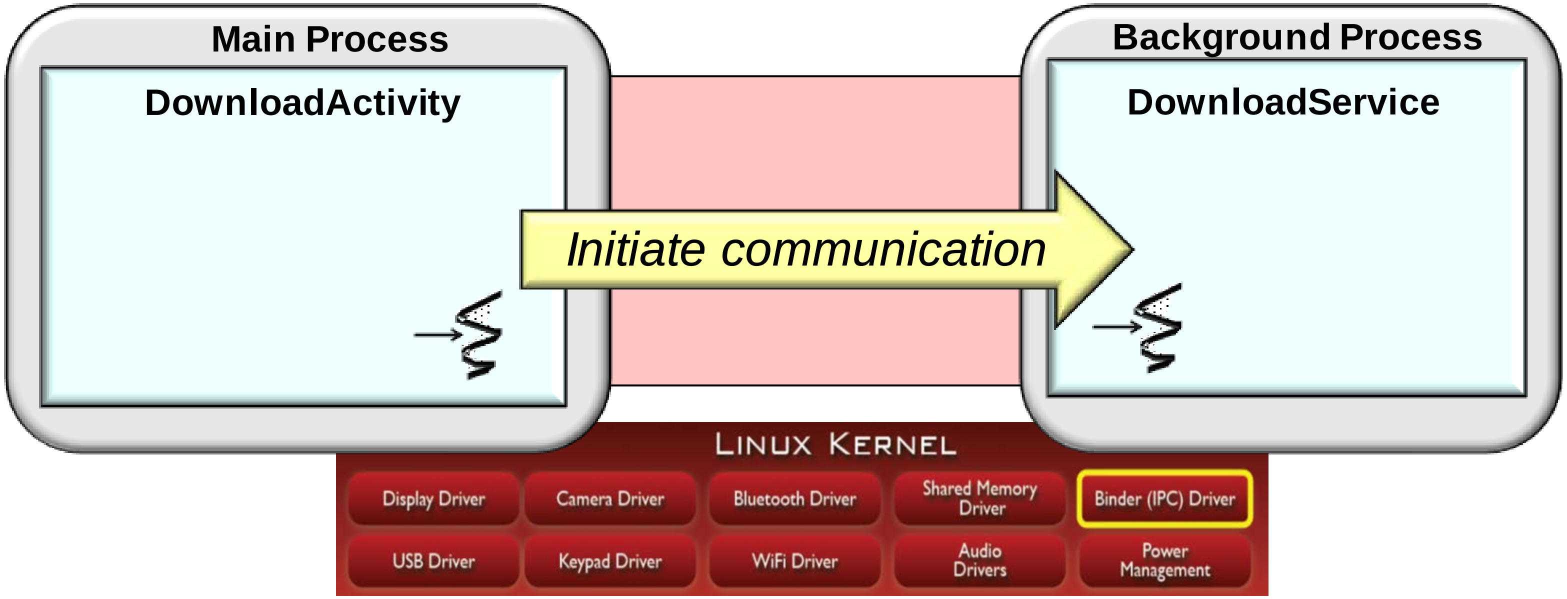
- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - The Android Binder RPC framework underlies the various IPC mechanisms



Running a Service in its own process may also require modifications to how data is exchanged

Communicating from Activities to Services

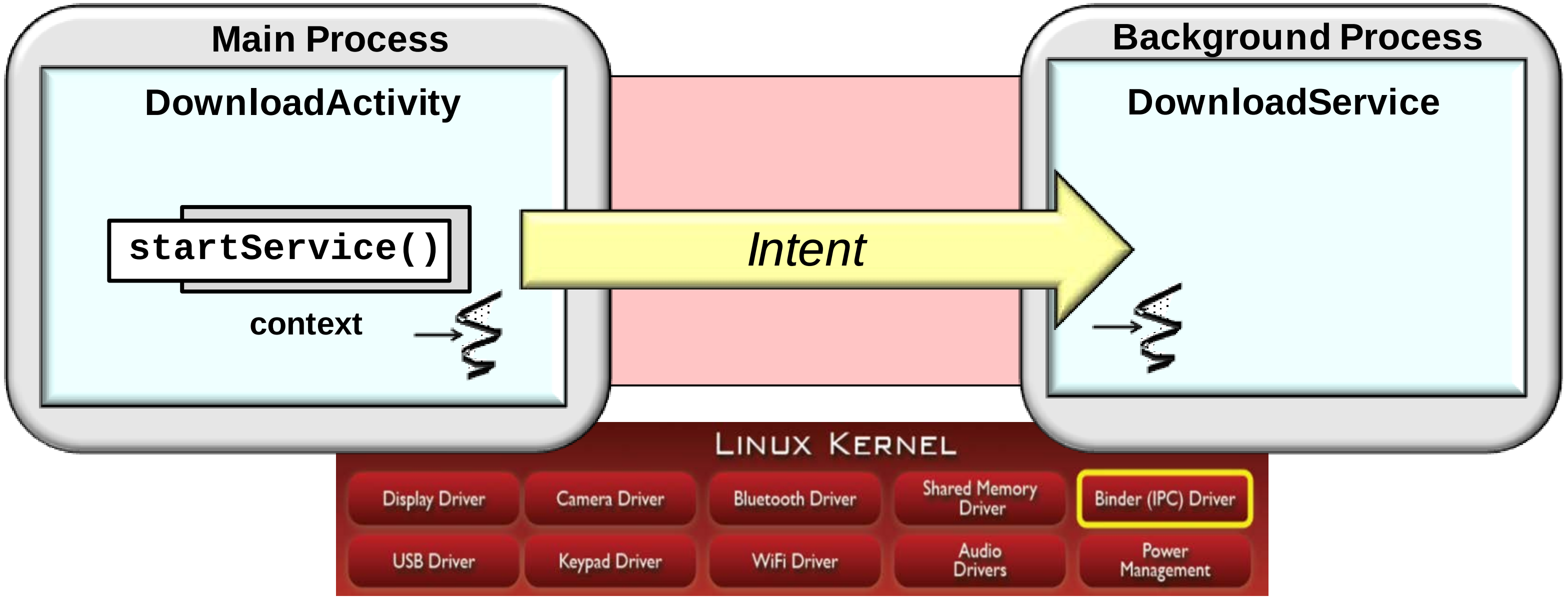
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service

Mechanism selection depends on factors like Started vs. Bound Services or message- vs. method-oriented

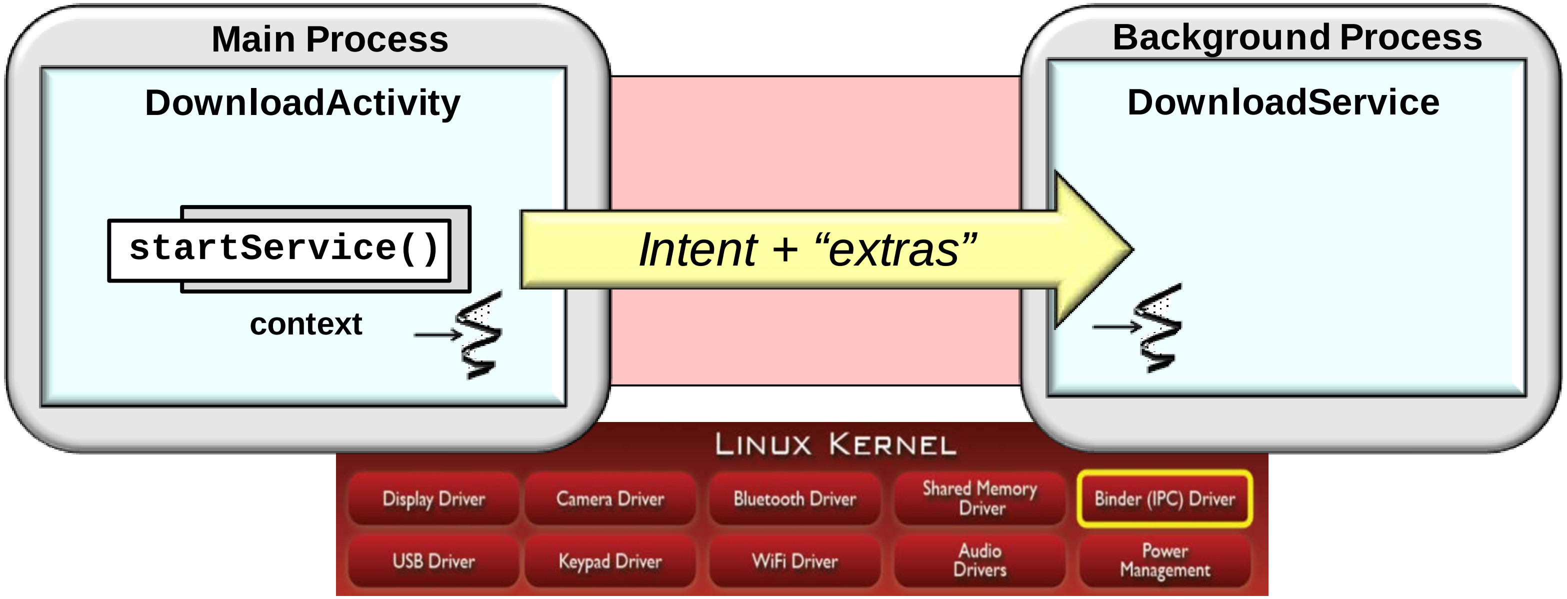
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()

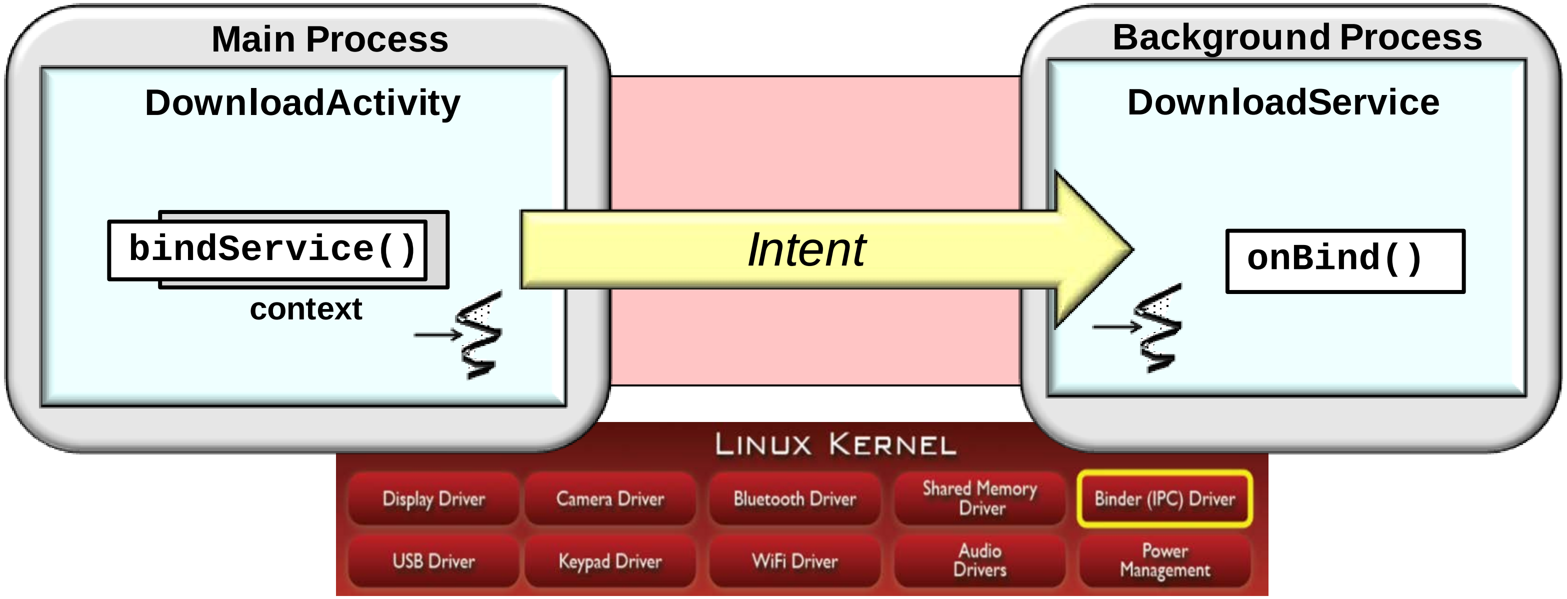
See earlier parts on “Programming Started Services” & “Android Intent Service”

Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
    - Parameters can be added as "extras" to the Intent used to start a Service

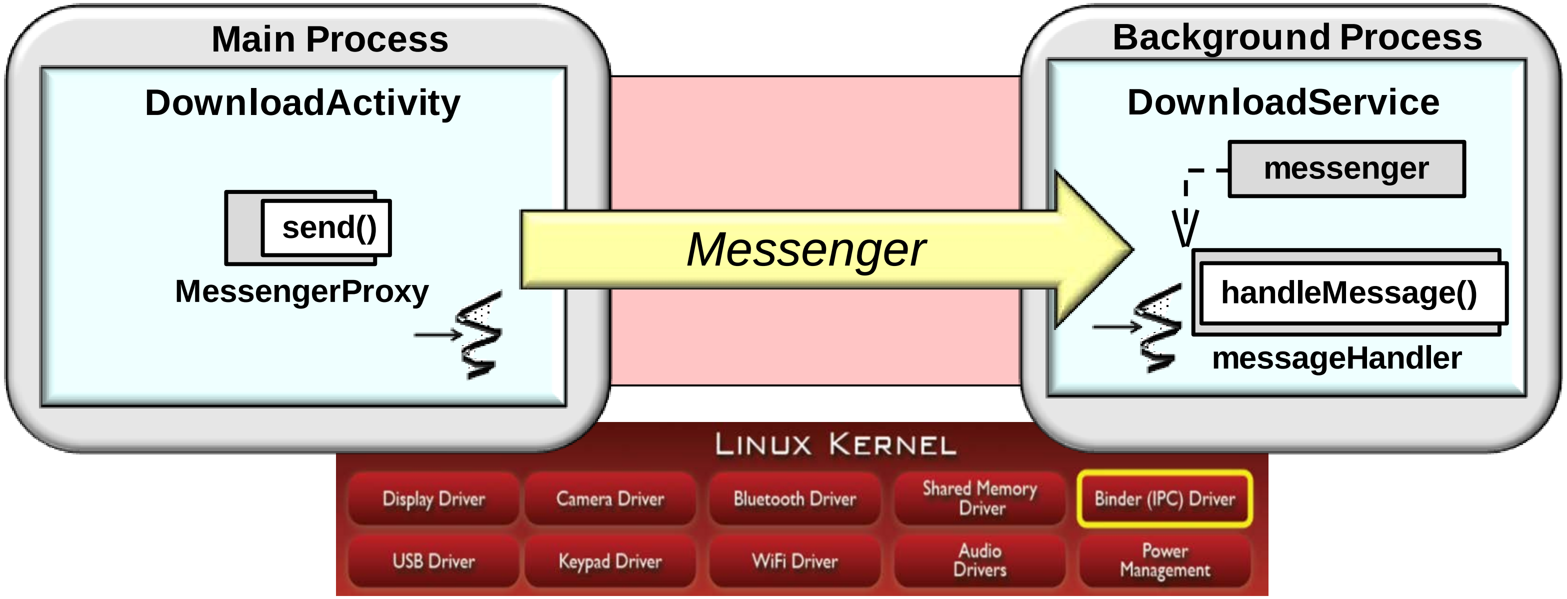
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()

See earlier part on “Overview of Android Services”

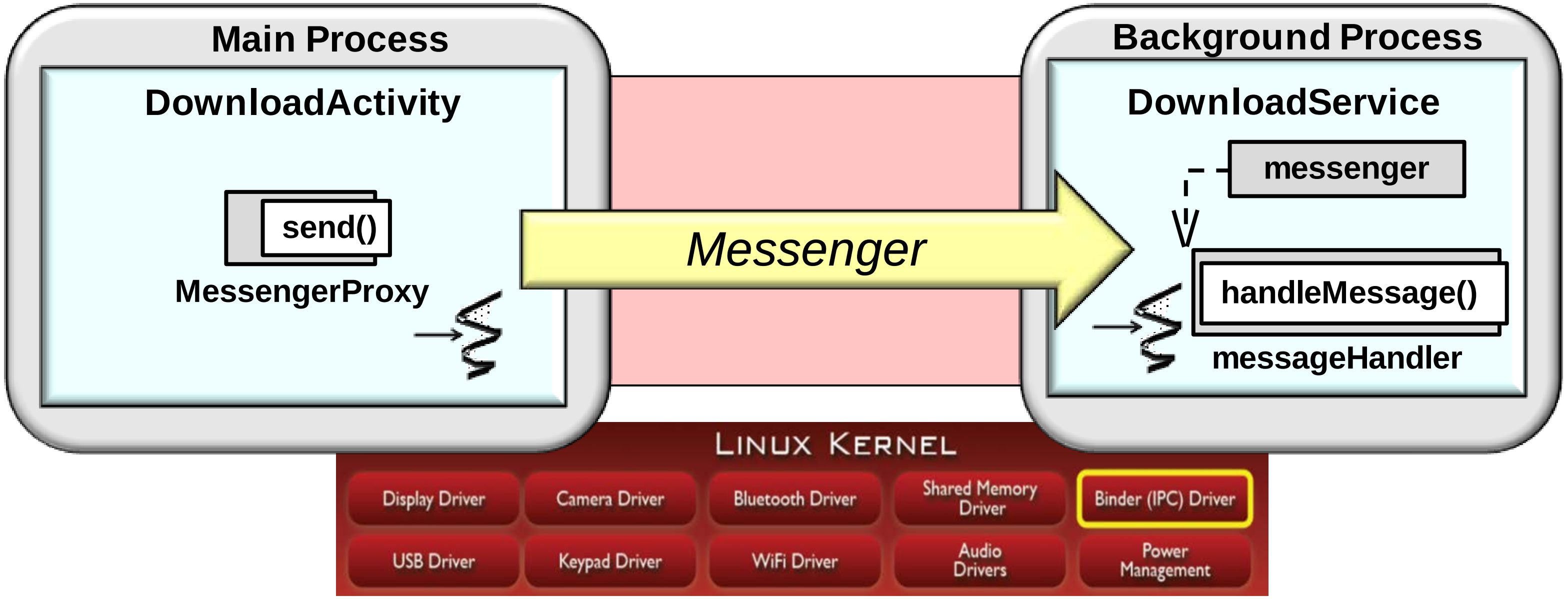
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger

See developer.android.com/reference/android/os/Messenger.html

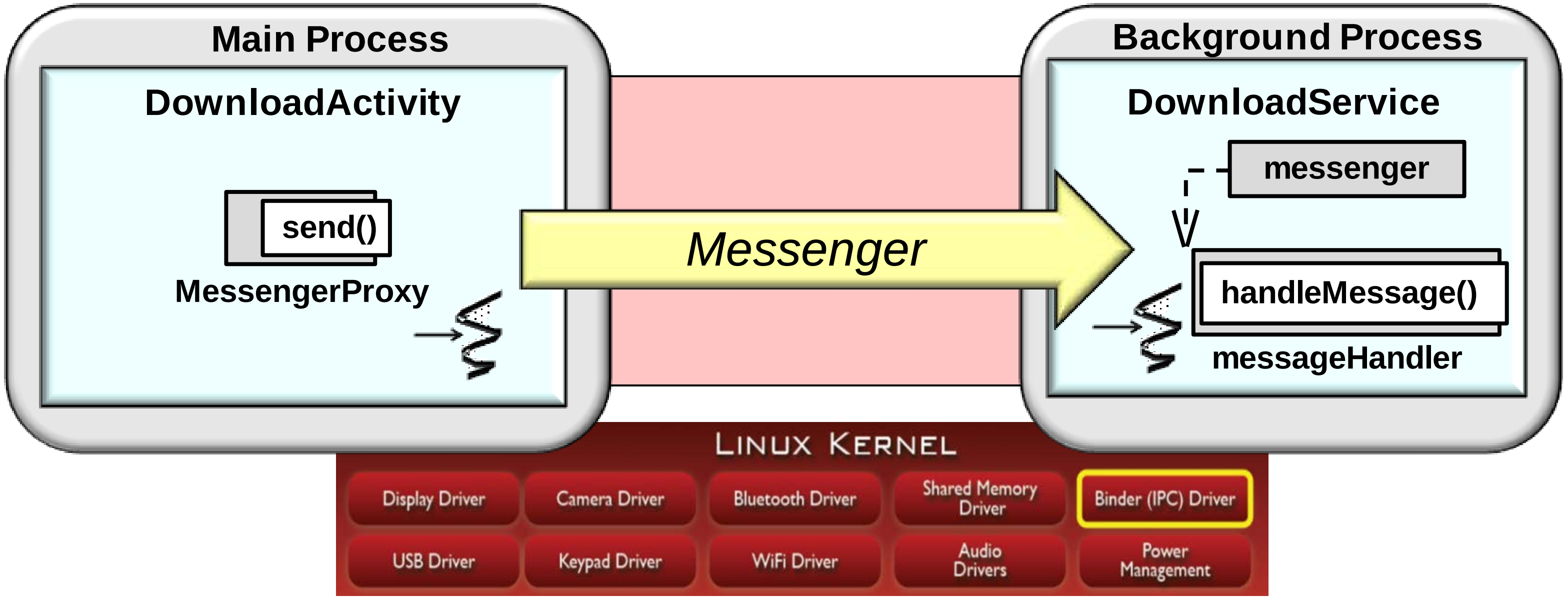
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger

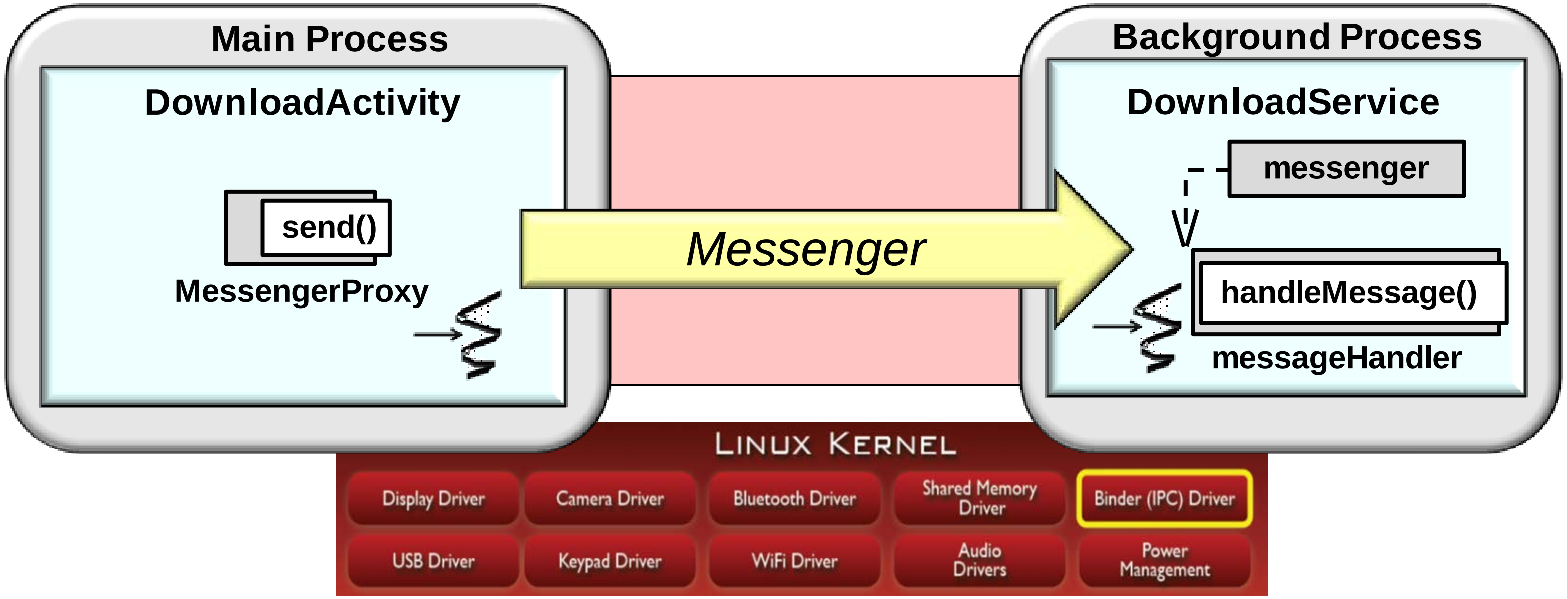
See earlier part on “Sending & Handling Messages with Android Handler”

Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger
      - A Messenger encapsulates a Handler implemented within a Service

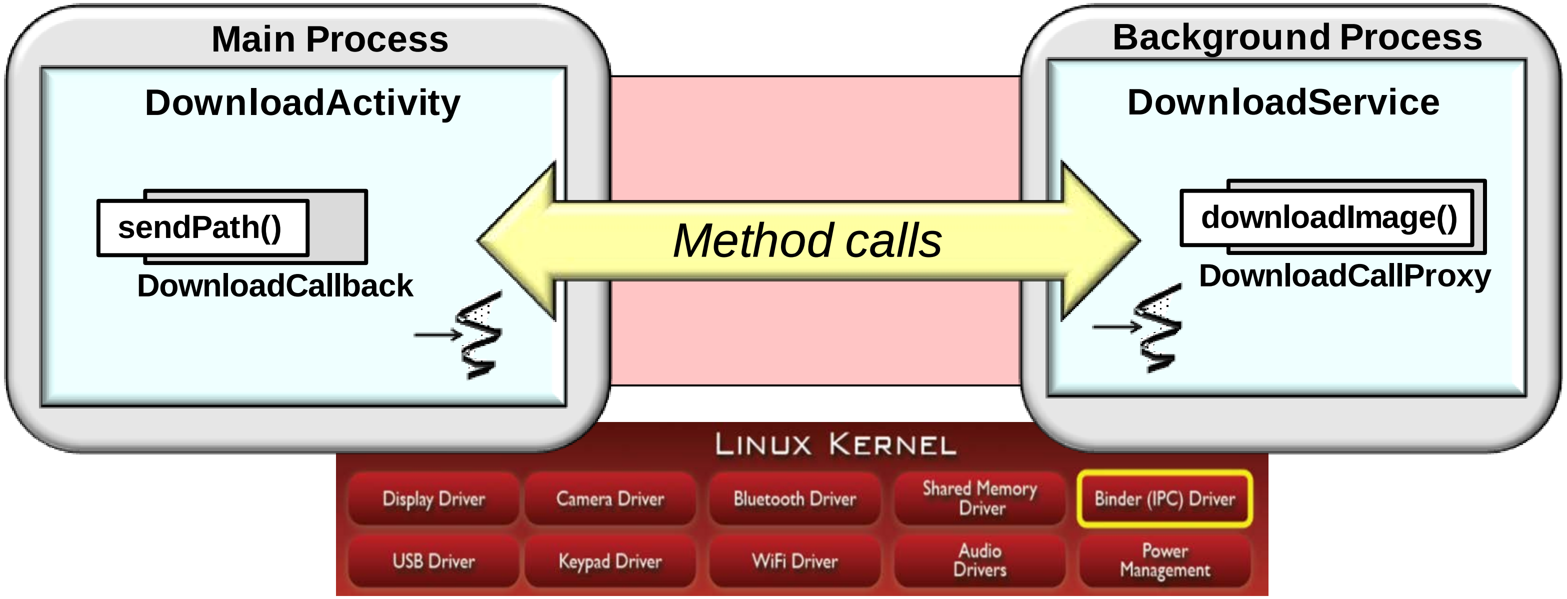
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger
      - A Messenger encapsulates a Handler implemented within a Service
      - Enables passing Messages to a Handler across process boundaries

See upcoming part on "Service to Activity Communication via Android Messenger"

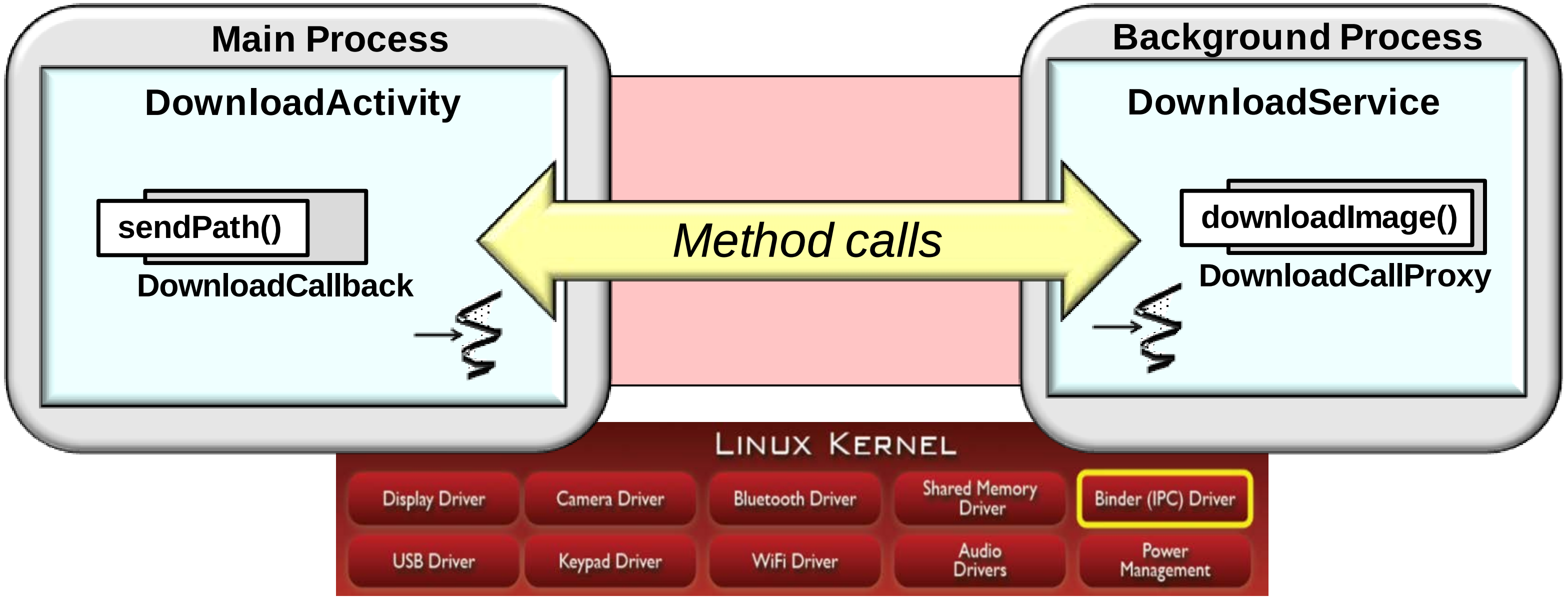
Communicating from Activities to Services



- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger
    - Invoke method calls
      - Use stubs generated by the AIDL compiler

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

Communicating from Activities to Services

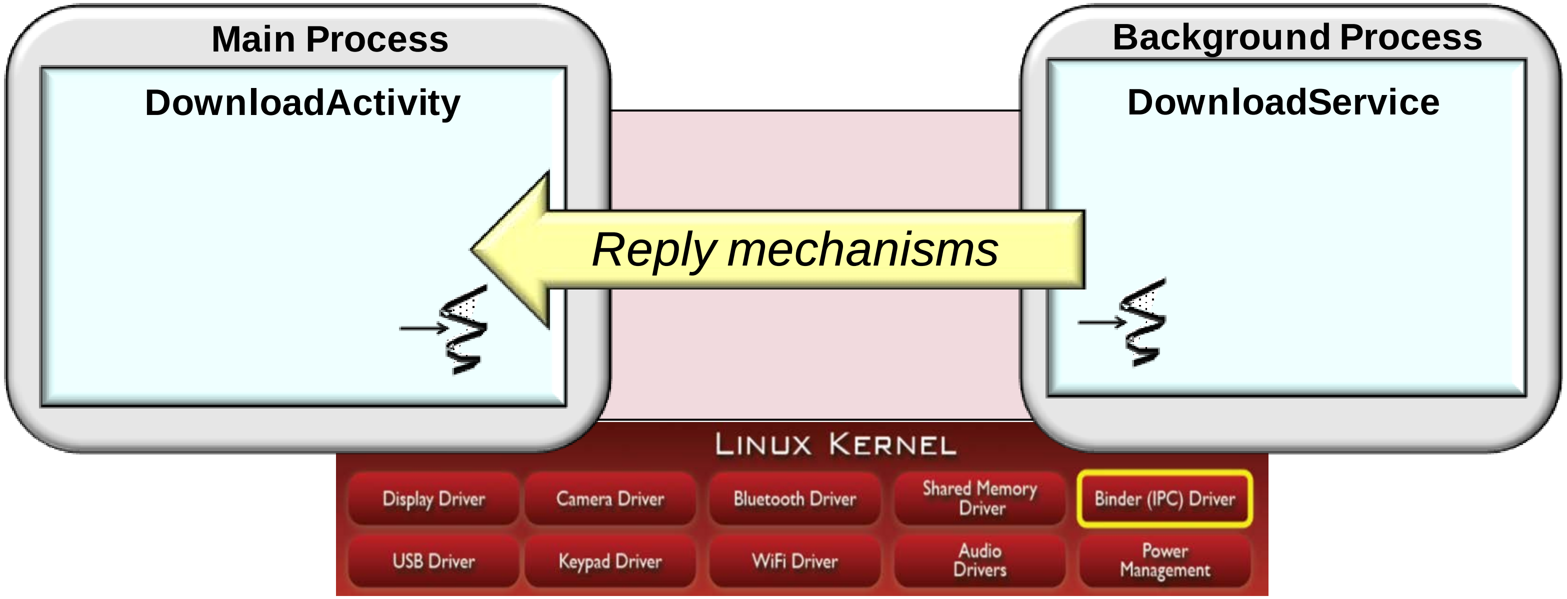


- Activities can use several mechanisms to communicate to a Service
  - Send an Intent command to Started Service via startService()
  - Bind to a Bound Service via BindService()
    - Call send() on a reference to a Messenger
    - Invoke method calls
      - Use stubs generated by the AIDL compiler
      - These methods can be programmed to implement various behaviors

See upcoming part on "Programming Bound Services"

Communicating from Services to Activities

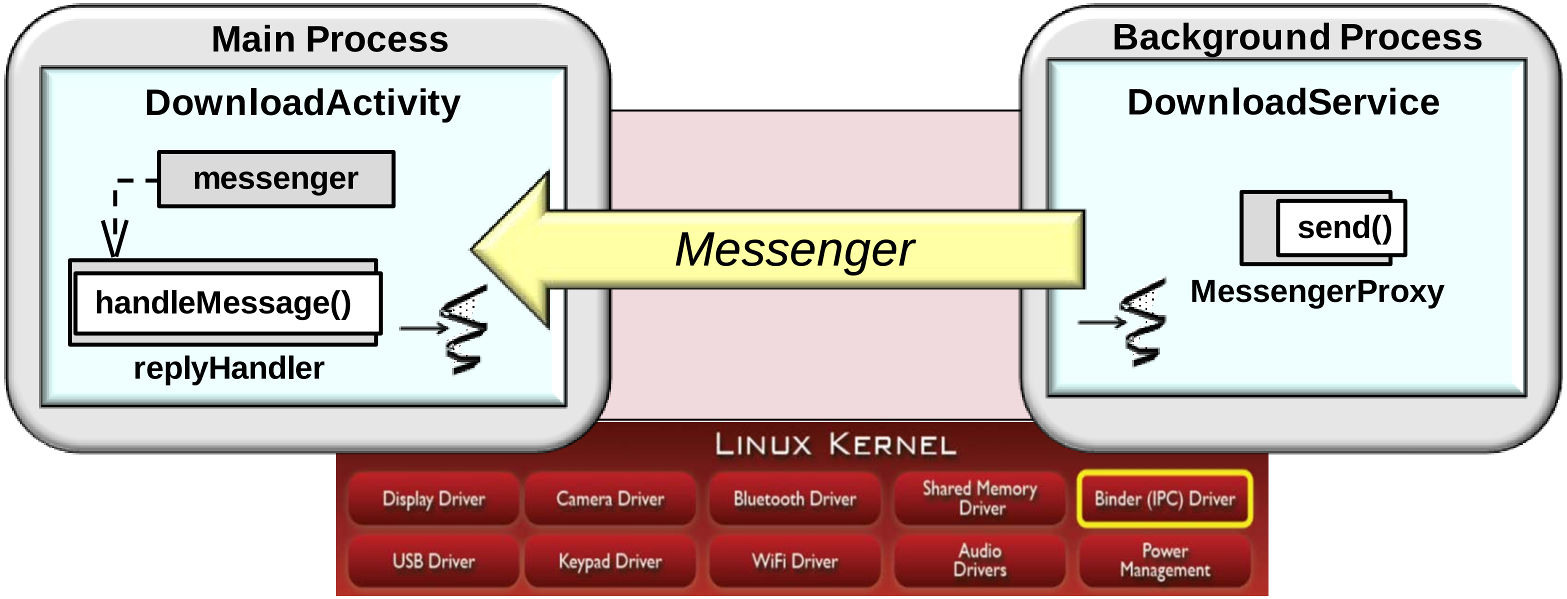
Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them

The Activity initiating the communication typically dictates the reply mechanism

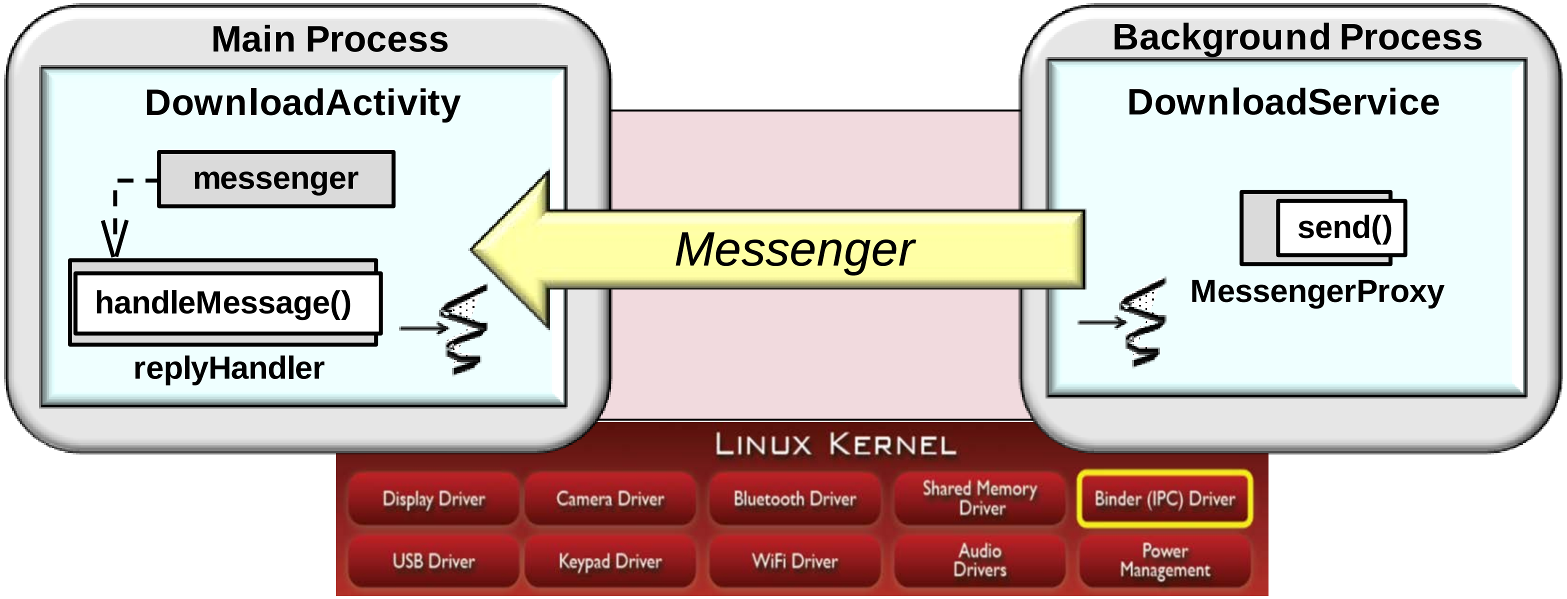
Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service

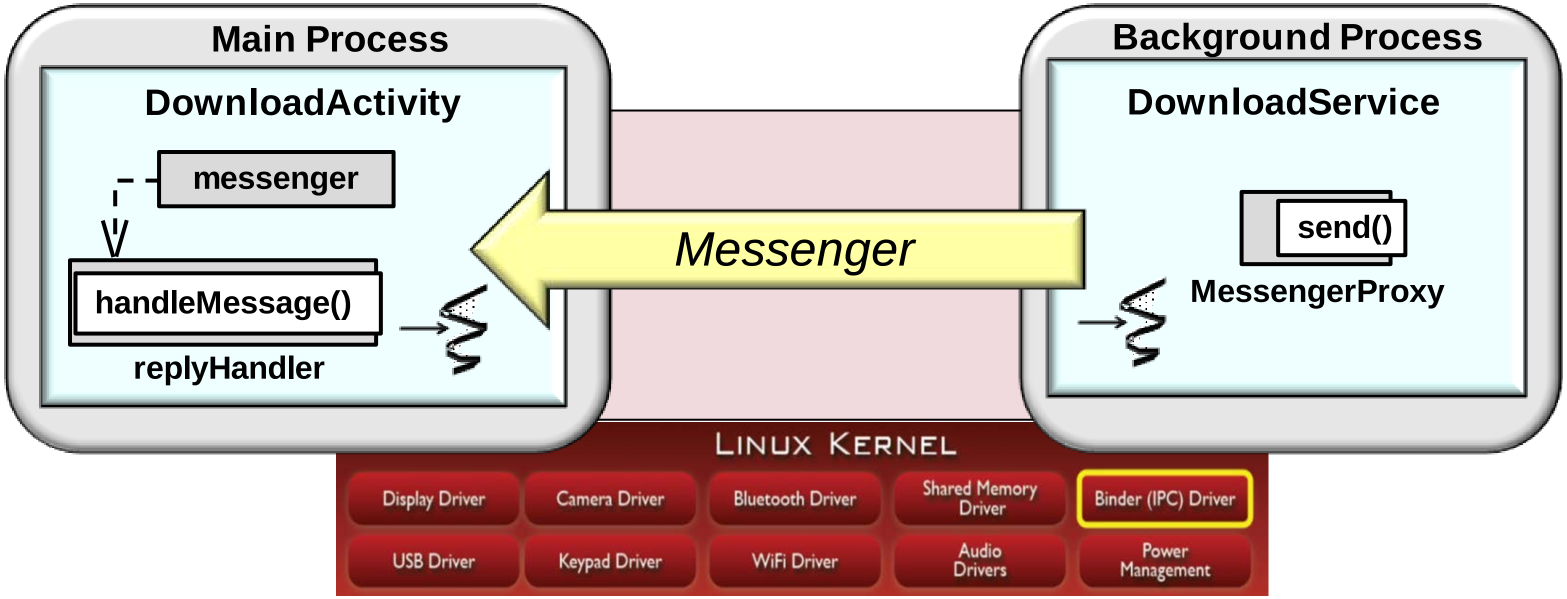
See developer.android.com/reference/android/os/Messenger.html

Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service
    - The Activity creates a Messenger Service & gives a reference to it to the Service

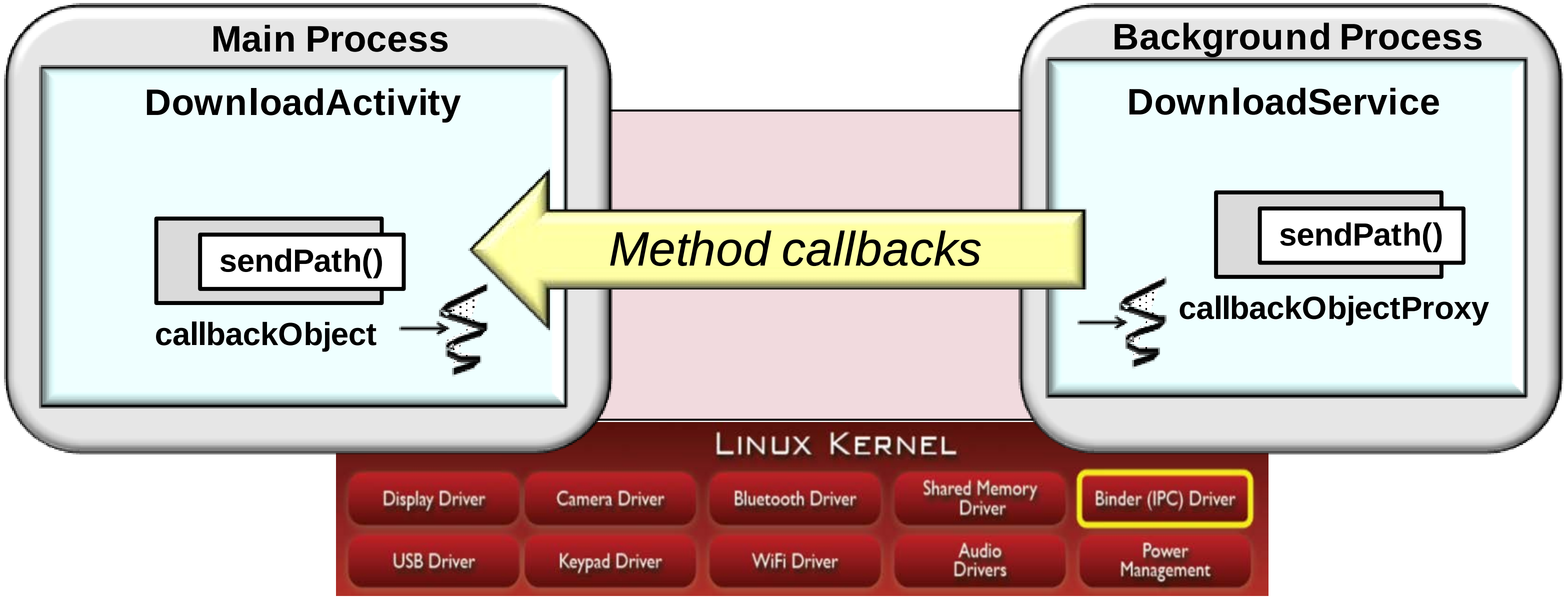
Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service
    - The Activity creates a Messenger Service & gives a reference to it to the Service
    - The Service then uses this Messenger to send reply Messages back to the Activity’s Handler

See upcoming part on “Service to Activity Communication via Android Messenger”

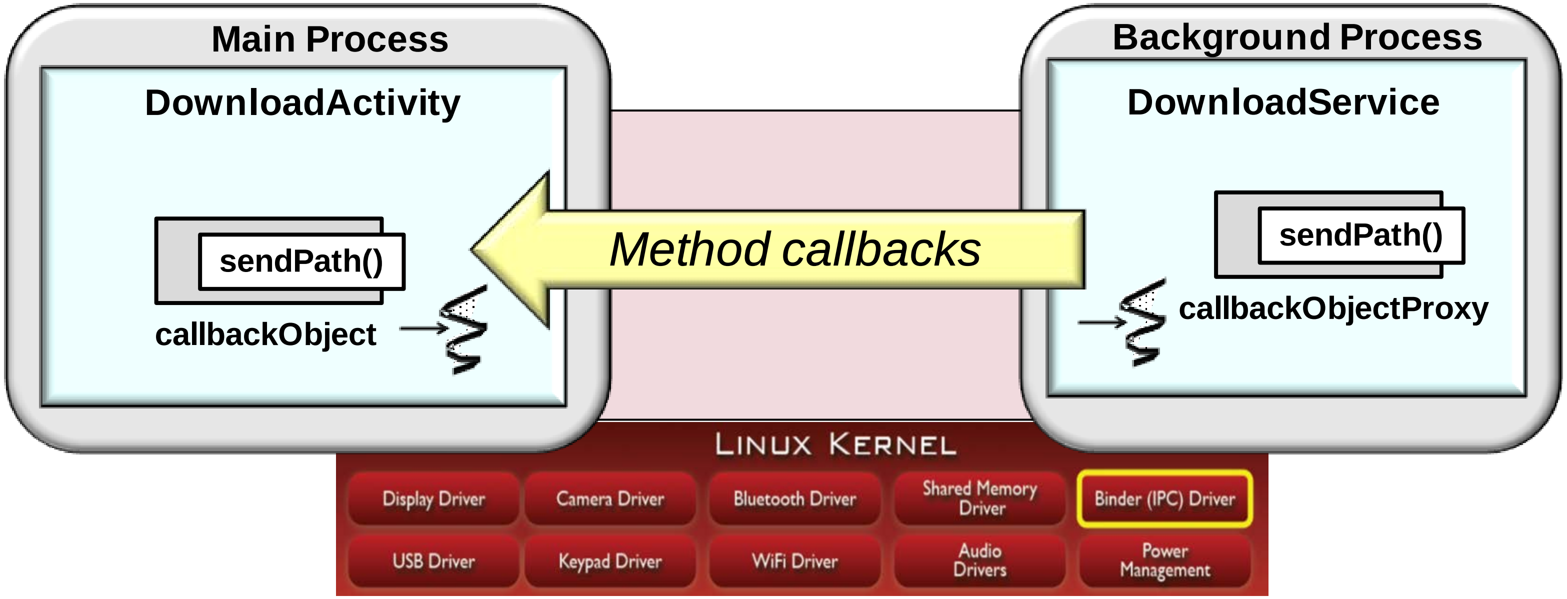
Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service
  - Use an AIDL-based callback object passed from the Activity to the Service

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

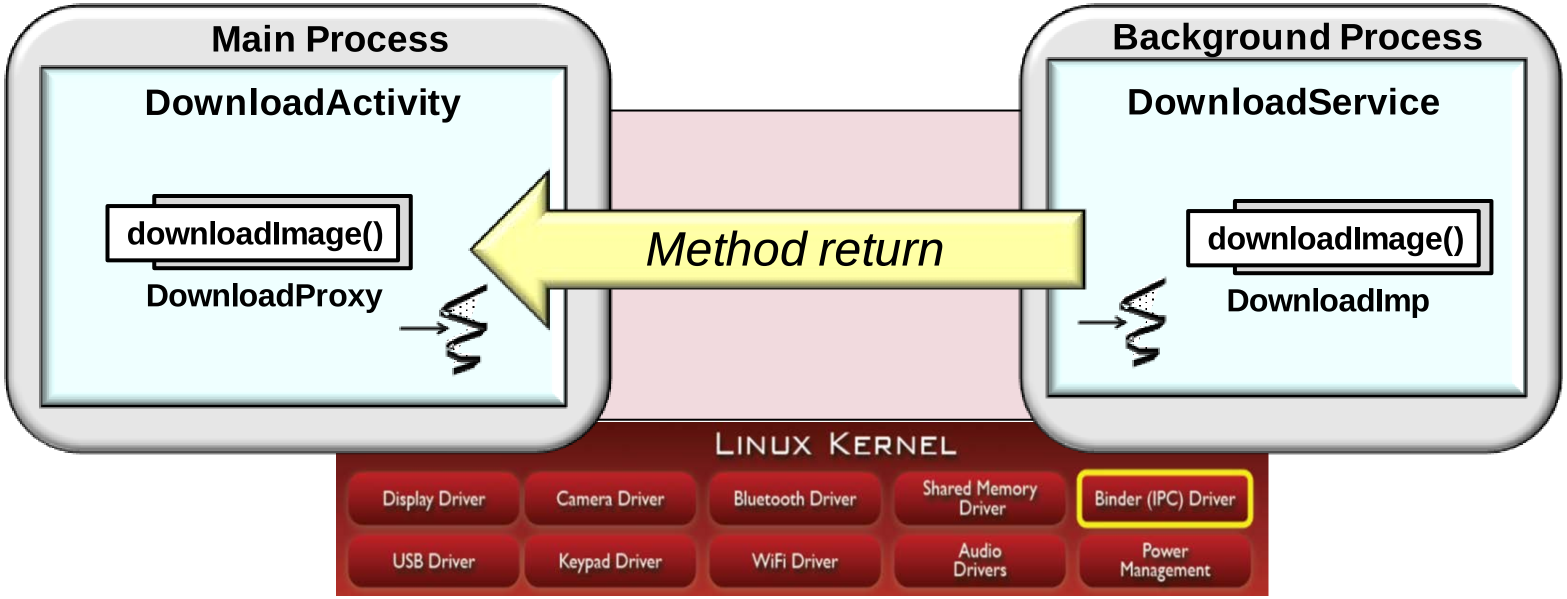
Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
    - Use a Messenger passed from the Activity to the Service
    - Use an AIDL-based callback object passed from the Activity to the Service
        - Invoke oneway method to return the reply to the Activity

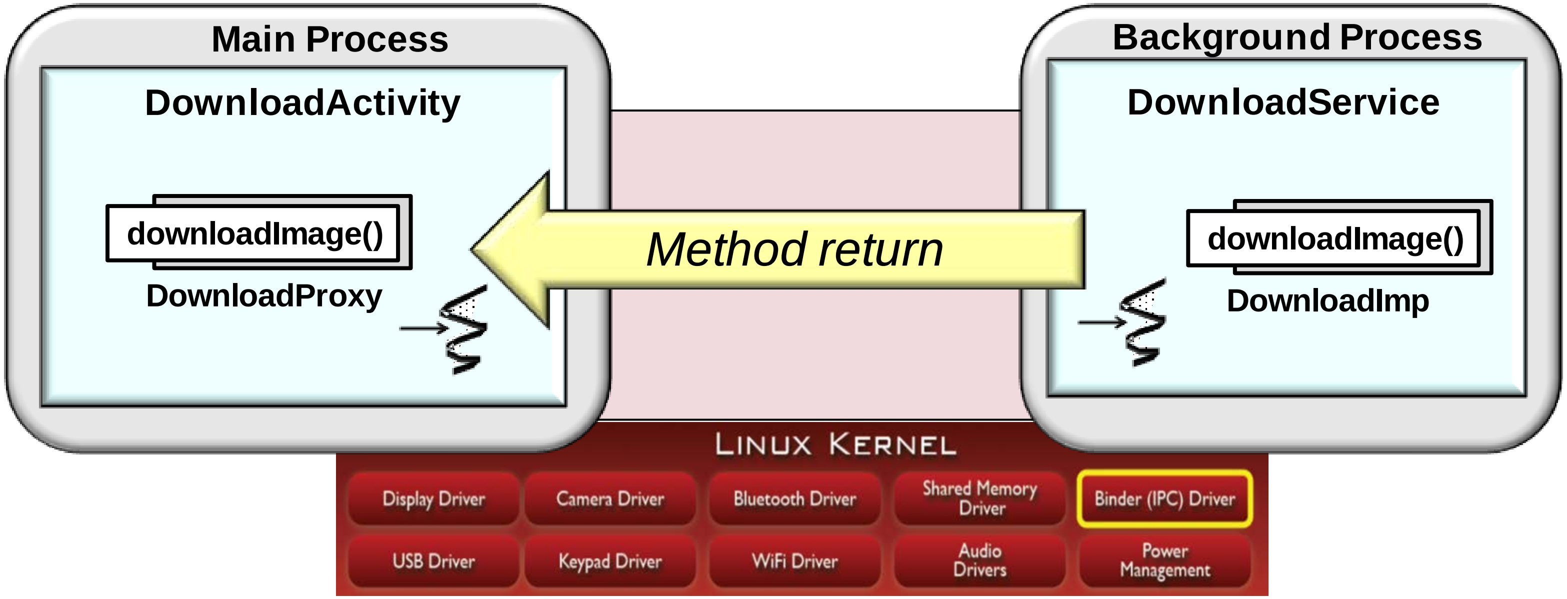
See upcoming part on
"Programming Bound Services"

Communicating from Services to Activities



- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service
  - Use an AIDL-based callback object passed from the Activity to the Service
  - Use an AIDL-based twoway method called from the Activity on the Service

Communicating from Services to Activities



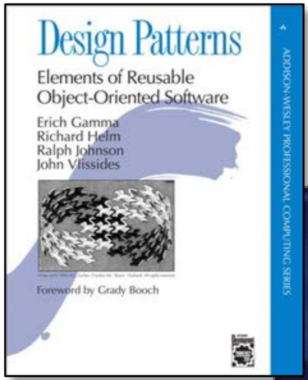
- Services can reply to Activities that initiated communication with them
  - Use a Messenger passed from the Activity to the Service
  - Use an AIDL-based callback object passed from the Activity to the Service
  - Use an AIDL-based twoway method called from the Activity on the Service
    - The return value and/or out parameters of the twoway method implementation implicitly sends a reply from the Service back to the Activity

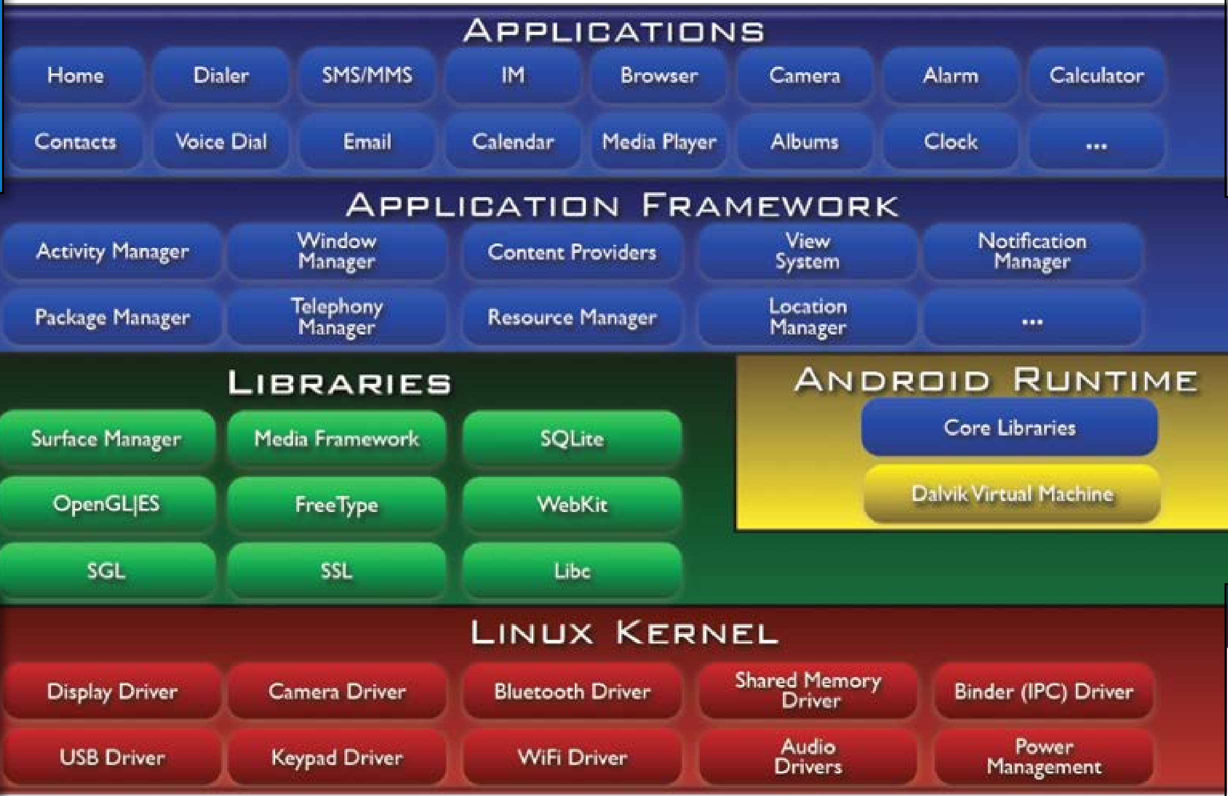
Although twoway method calls seem convenient, they are problematic..

Patterns Used by Communication & Service Frameworks (Part 1)

Patterns in Android's Frameworks

- Android's frameworks & applications of these frameworks are designed, implemented, & integrated in accordance with many POSA & GoF patterns


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



APPLICATIONS

Home | Dialer | SMS/MMS | IM | Browser | Camera | Alarm | Calculator

Contacts | Voice Dial | Email | Calendar | Media Player | Albums | Clock | ...

APPLICATION FRAMEWORK

Activity Manager | Window Manager | Content Providers | View System | Notification Manager

Package Manager | Telephony Manager | Resource Manager | Location Manager | ...

LIBRARIES

Surface Manager | Media Framework | SQLite

OpenGL|ES | FreeType | WebKit

SGL | SSL | Libc

ANDROID RUNTIME

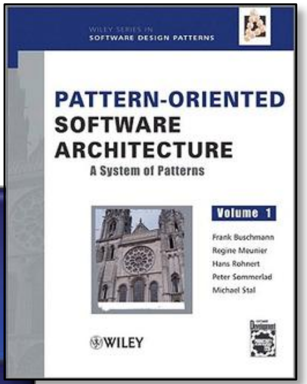
Core Libraries

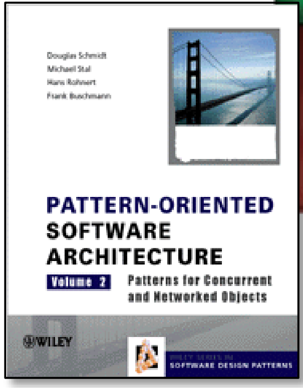
Dalvik Virtual Machine

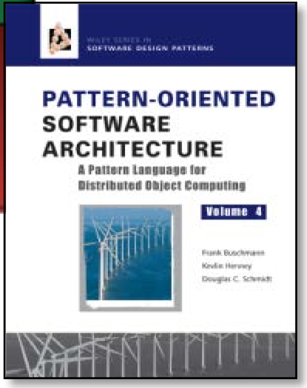
LINUX KERNEL

Display Driver | Camera Driver | Bluetooth Driver | Shared Memory Driver | Binder (IPC) Driver

USB Driver | Keypad Driver | WiFi Driver | Audio Drivers | Power Management

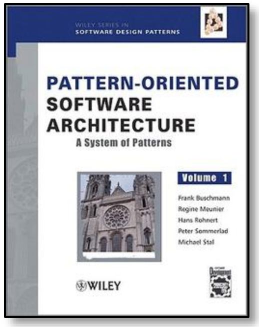



Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS

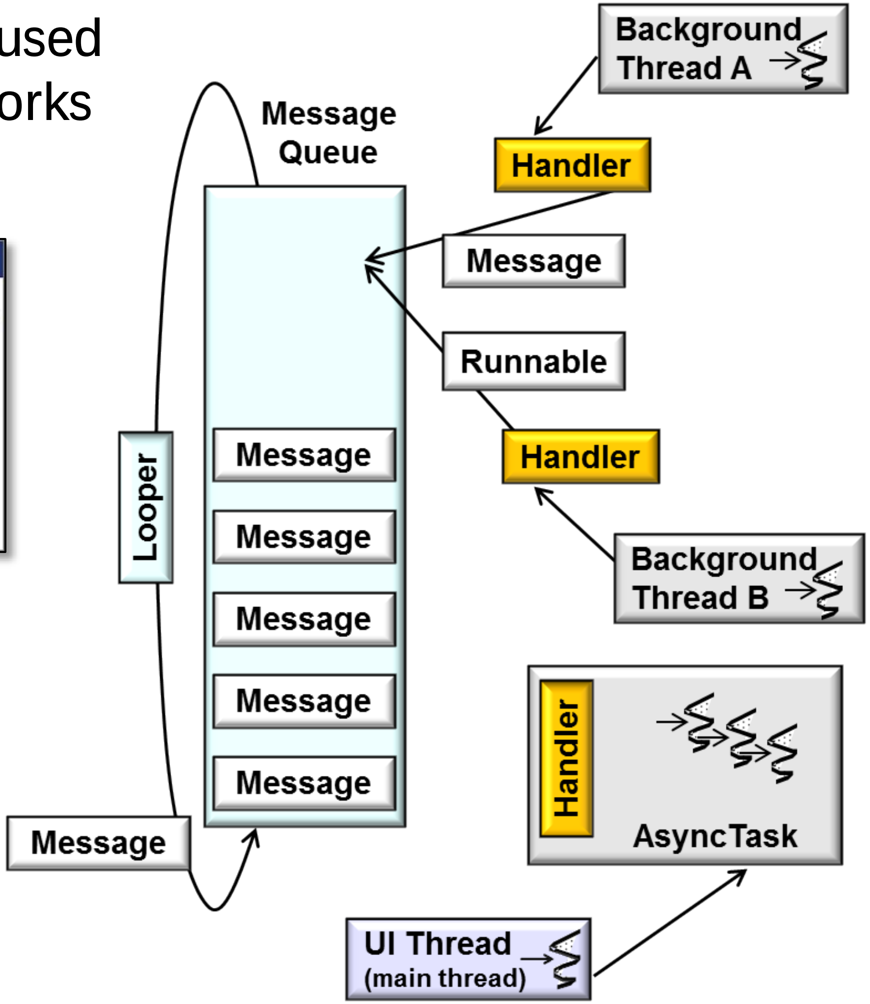

WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Object Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

Patterns in Android's Frameworks

- Android's frameworks & applications of these frameworks are designed, implemented, & integrated in accordance with many POSA & GoF patterns
  - We've summarized key patterns used in Android's concurrency frameworks



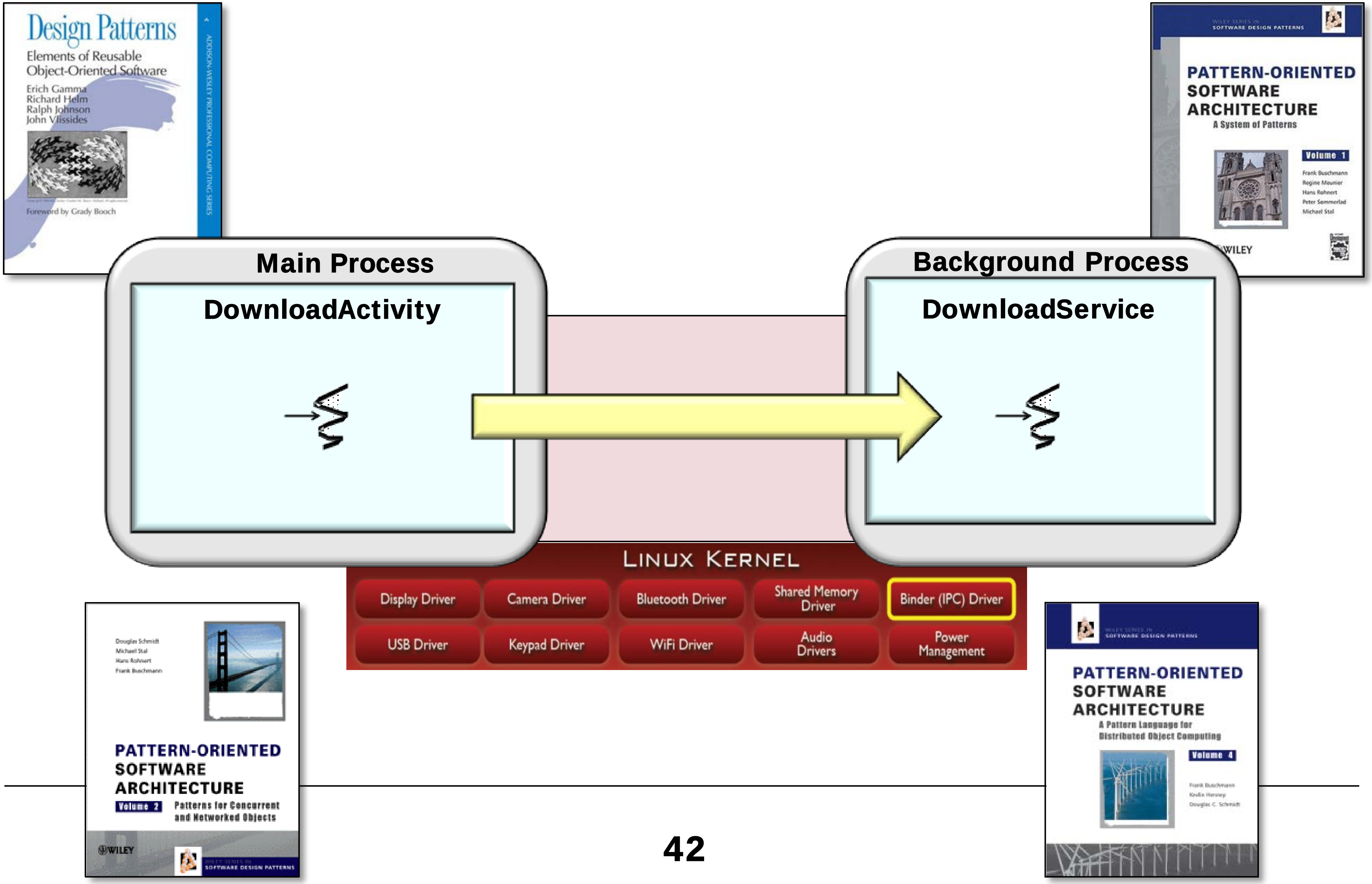
Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS



See earlier part on "Overview of Patterns & Frameworks (Part 2)"

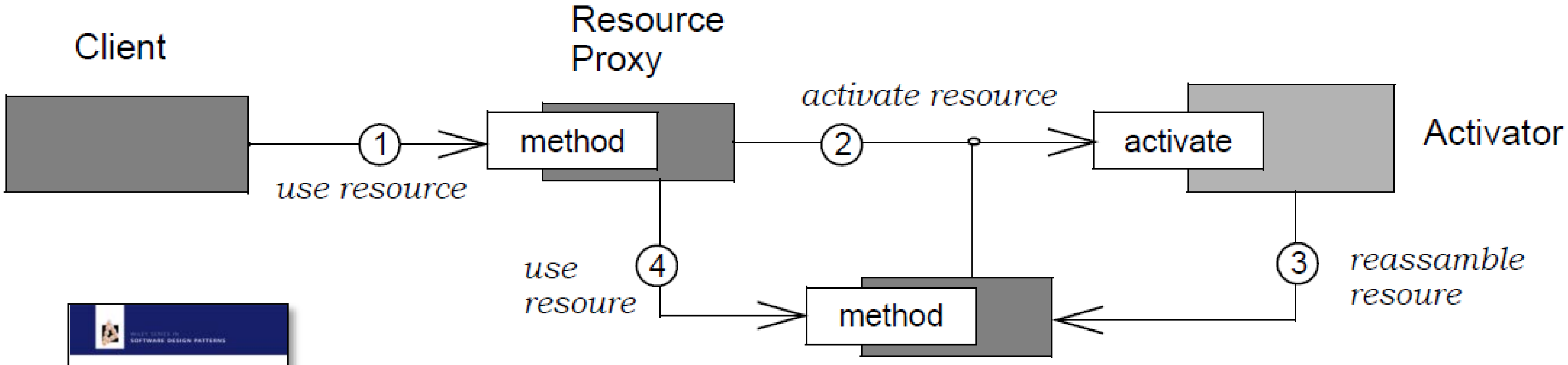
Patterns in Communication & Service Frameworks

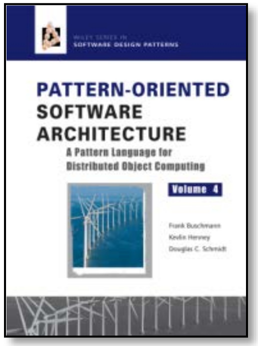
- Android's communication & service frameworks apply POSA & GoF patterns



Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns
  - *Activator* – Automate scalable on-demand activation & deactivation of service execution contexts to run services accessed by multiple clients without consuming resources unnecessarily

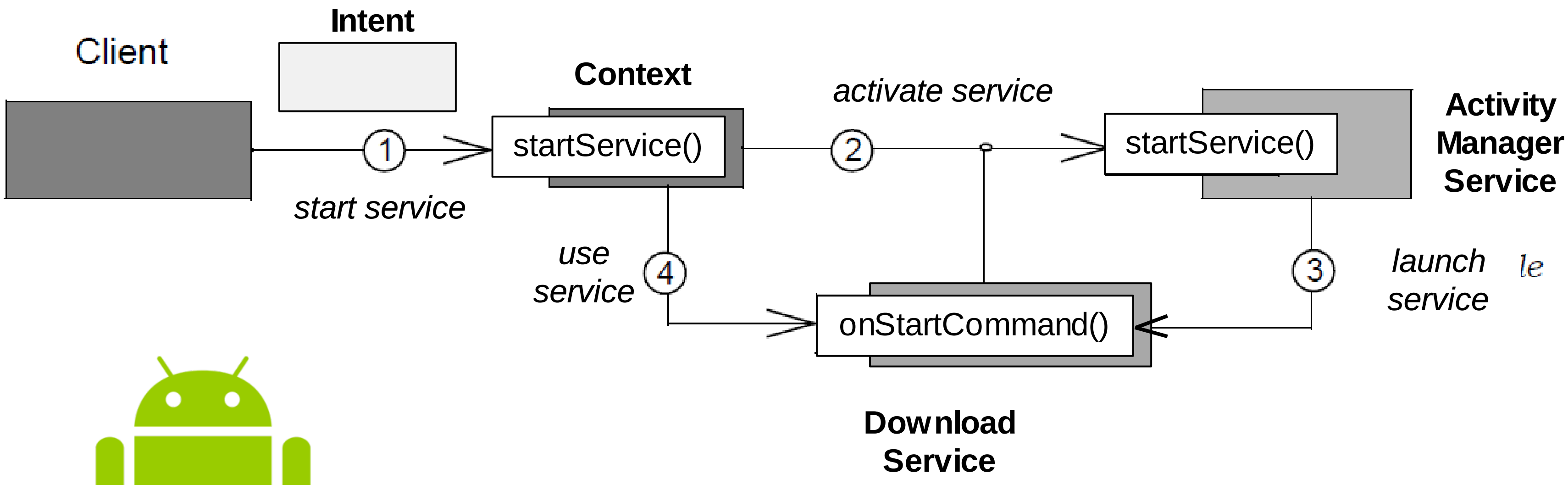




www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf has more info

Patterns in Communication & Service Frameworks

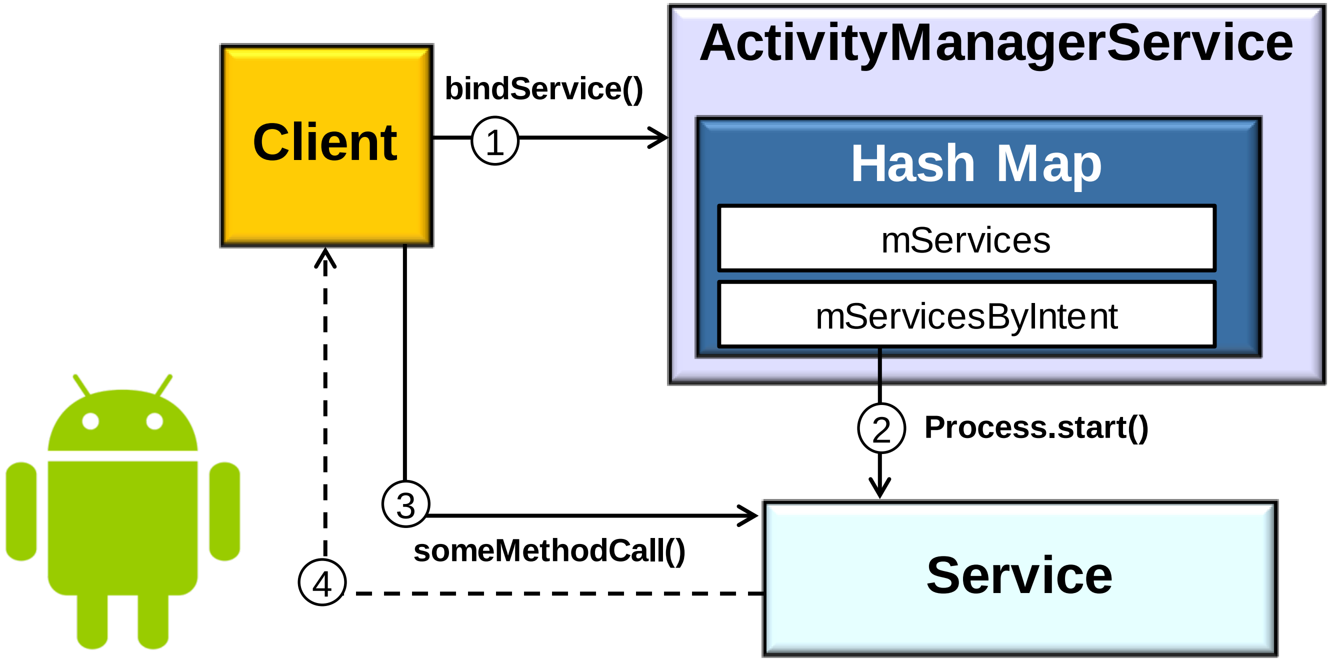
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Activator* – Automate scalable on-demand activation & deactivation of service execution contexts to run services accessed by multiple clients without consuming resources unnecessarily



See upcoming part on "The Activator Pattern"

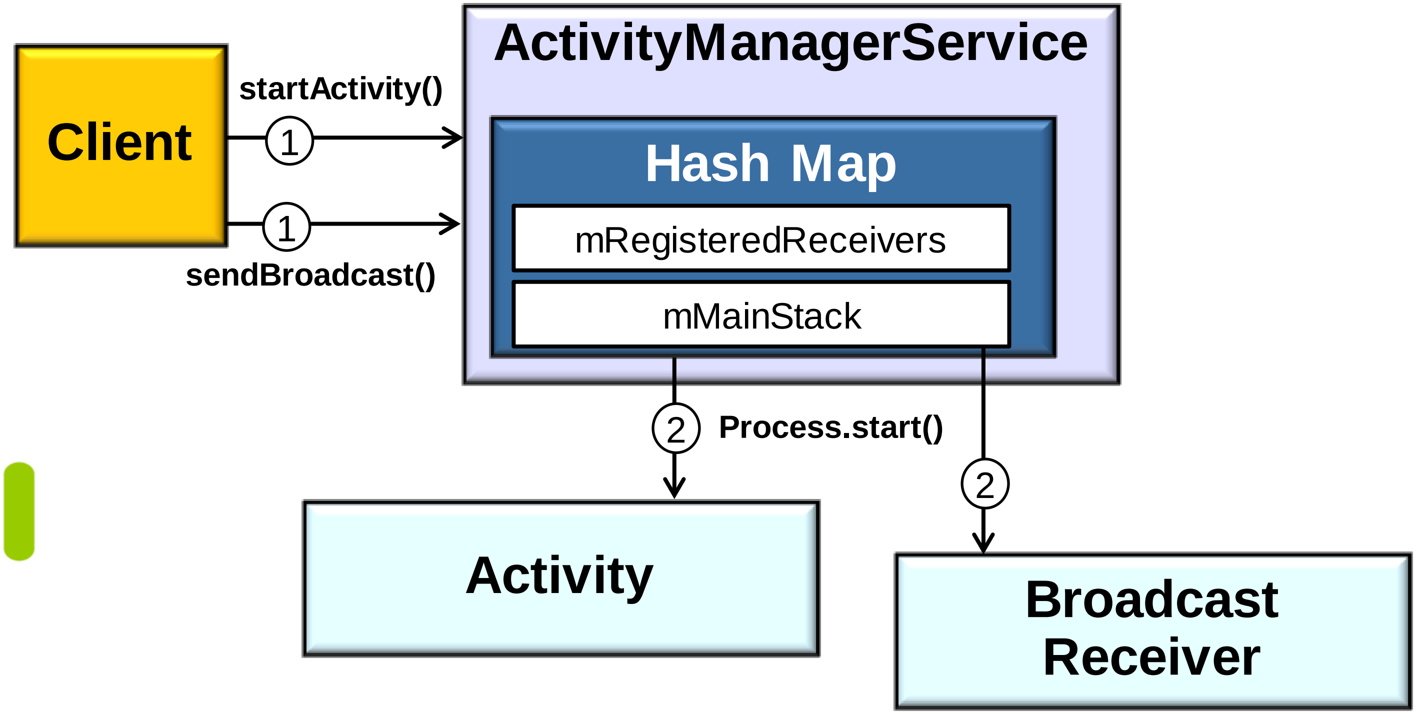
Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns
  - *Activator* – Automate scalable on-demand activation & deactivation of service execution contexts to run services accessed by multiple clients without consuming resources unnecessarily



Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns
  - *Activator* – Automate scalable on-demand activation & deactivation of service execution contexts to run services accessed by multiple clients without consuming resources unnecessarily

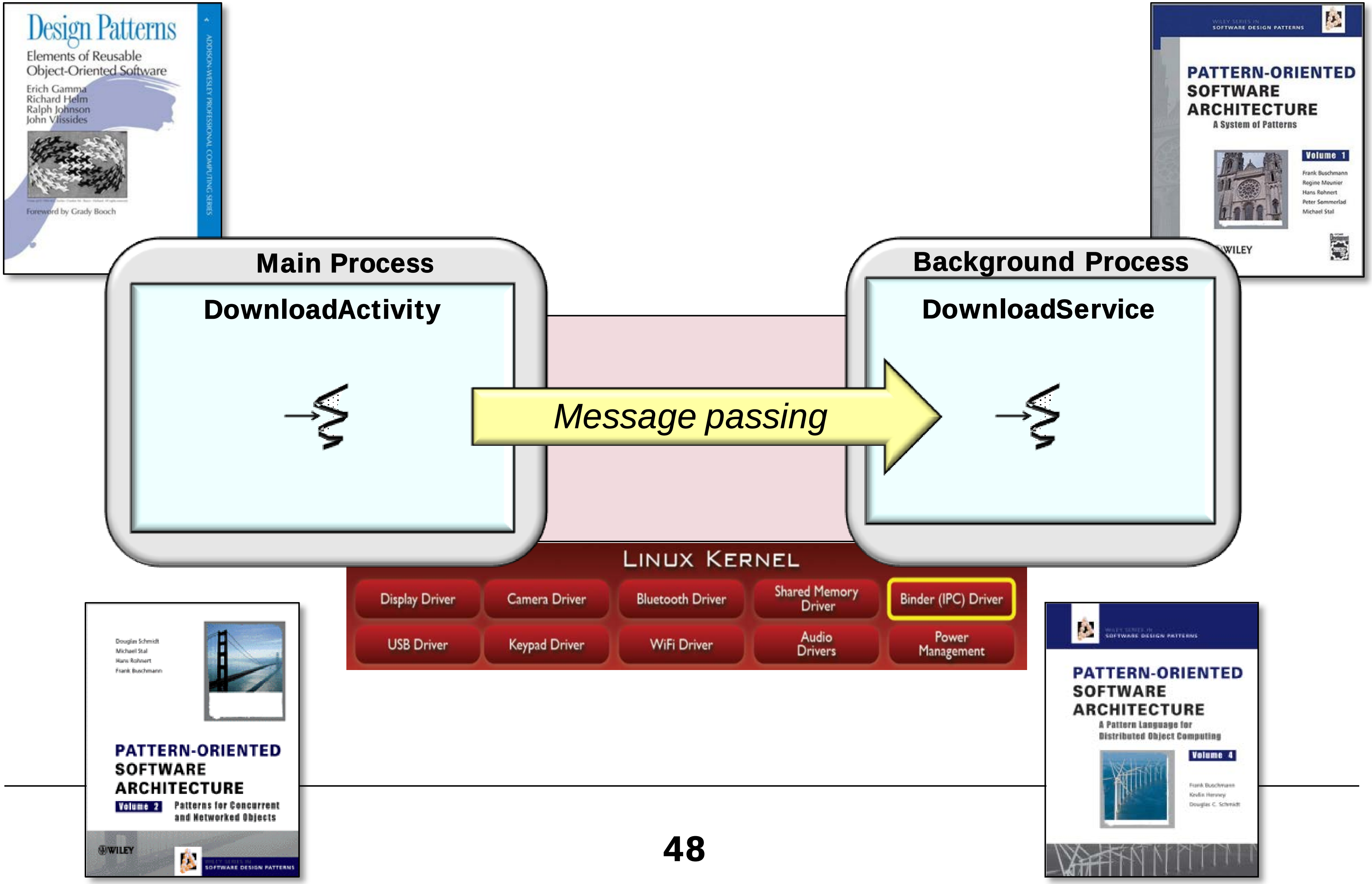


See frameworks/base/services/java/com/android/server/am for source code

Patterns Used by Communication & Service Frameworks (Part 2)

Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns



Main Process

DownloadActivity

Message passing

Background Process

DownloadService

LINUX KERNEL

Display Driver

Camera Driver

Bluetooth Driver

Shared Memory Driver

Binder (IPC) Driver

USB Driver

Keypad Driver

WiFi Driver

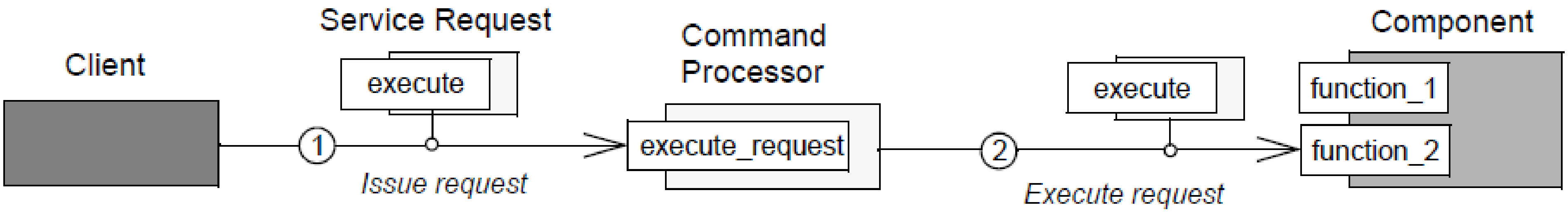
Audio Drivers

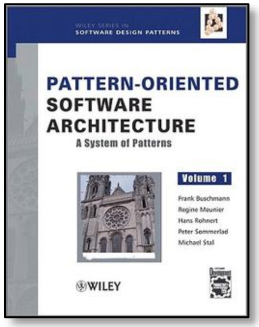
Power Management

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A System of Patterns
Volume 1
Frank Buschmann
Regine Meunier
Hans Rohnert
Peter Sommerlad
Michael Stal
WILEY
Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS
WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Object Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns
  - *Command Processor* – package a piece of application functionality—as well as its parameterization in an object—to execute it in another context
    - e.g., at a later point in time, in a different process or thread, etc.

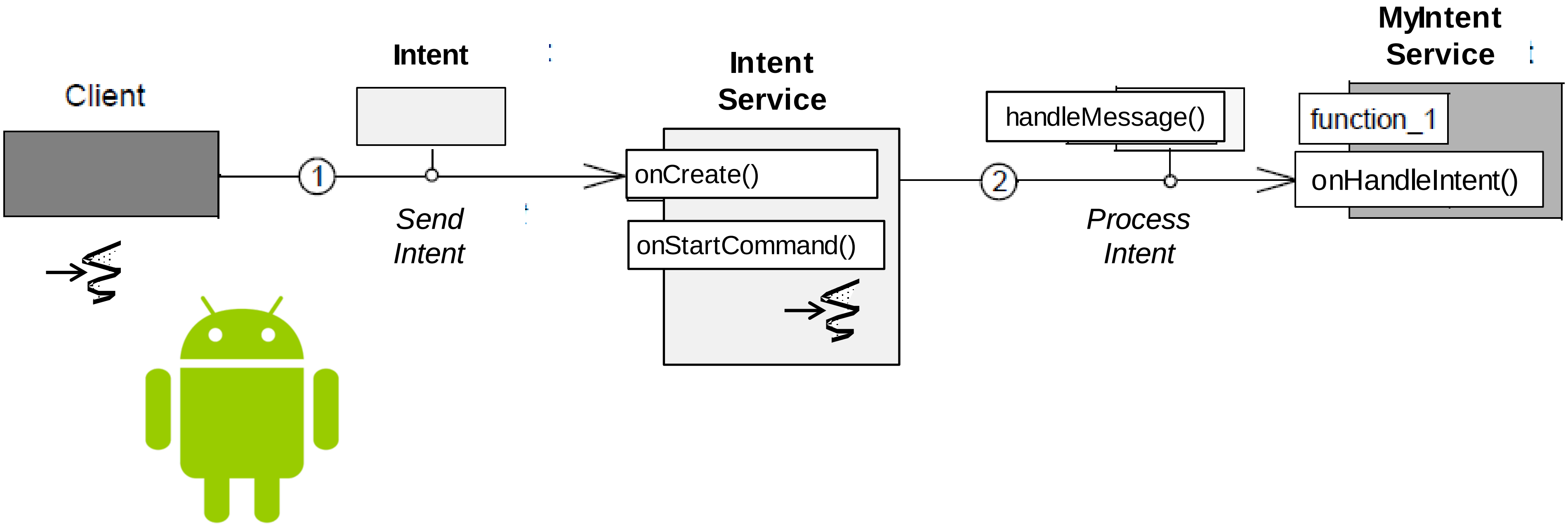




www.dre.vanderbilt.edu/~schmidt/PDF/CommandRevisited.pdf has more info

Patterns in Communication & Service Frameworks

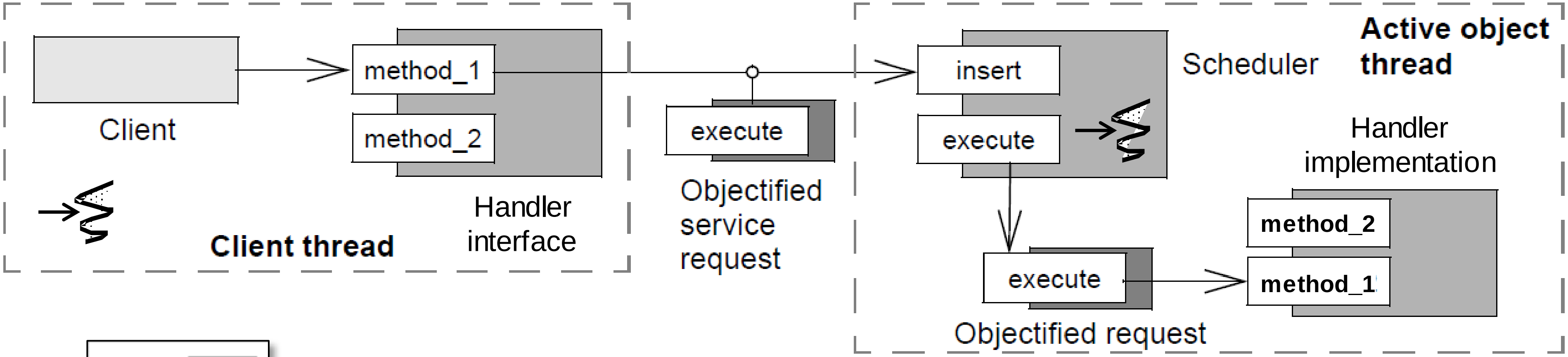
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Command Processor* – package a piece of application functionality—as well as its parameterization in an object—to execute it in another context
    - e.g., at a later point in time, in a different process or thread, etc.



See earlier part on the "Android IntentService"

Patterns in Communication & Service Frameworks

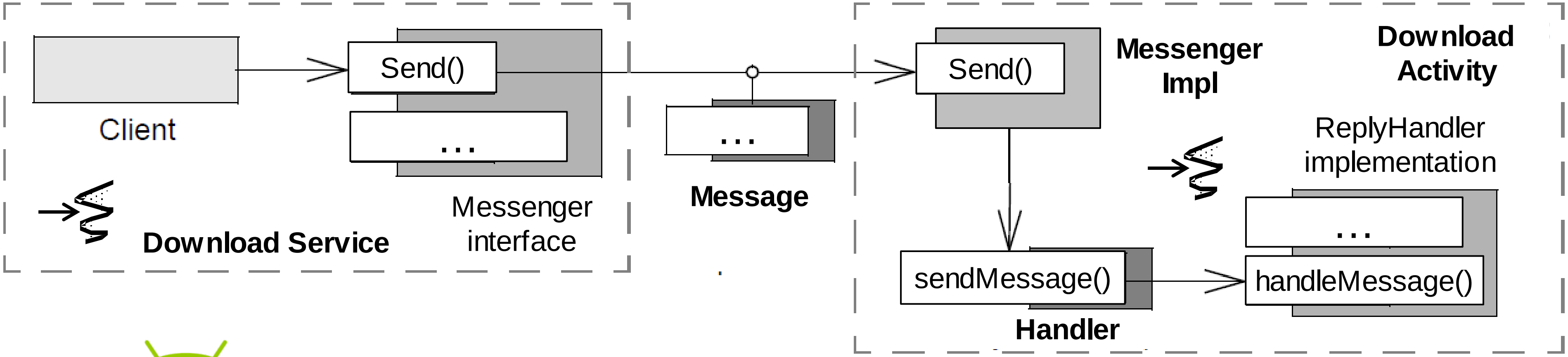
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Active Object* – define service requests on components as the units of concurrency & run service requests on a component in different thread(s) from the requesting client thread



www.dre.vanderbilt.edu/~schmidt/PDF/Act-Obj.pdf has more info

Patterns in Communication & Service Frameworks

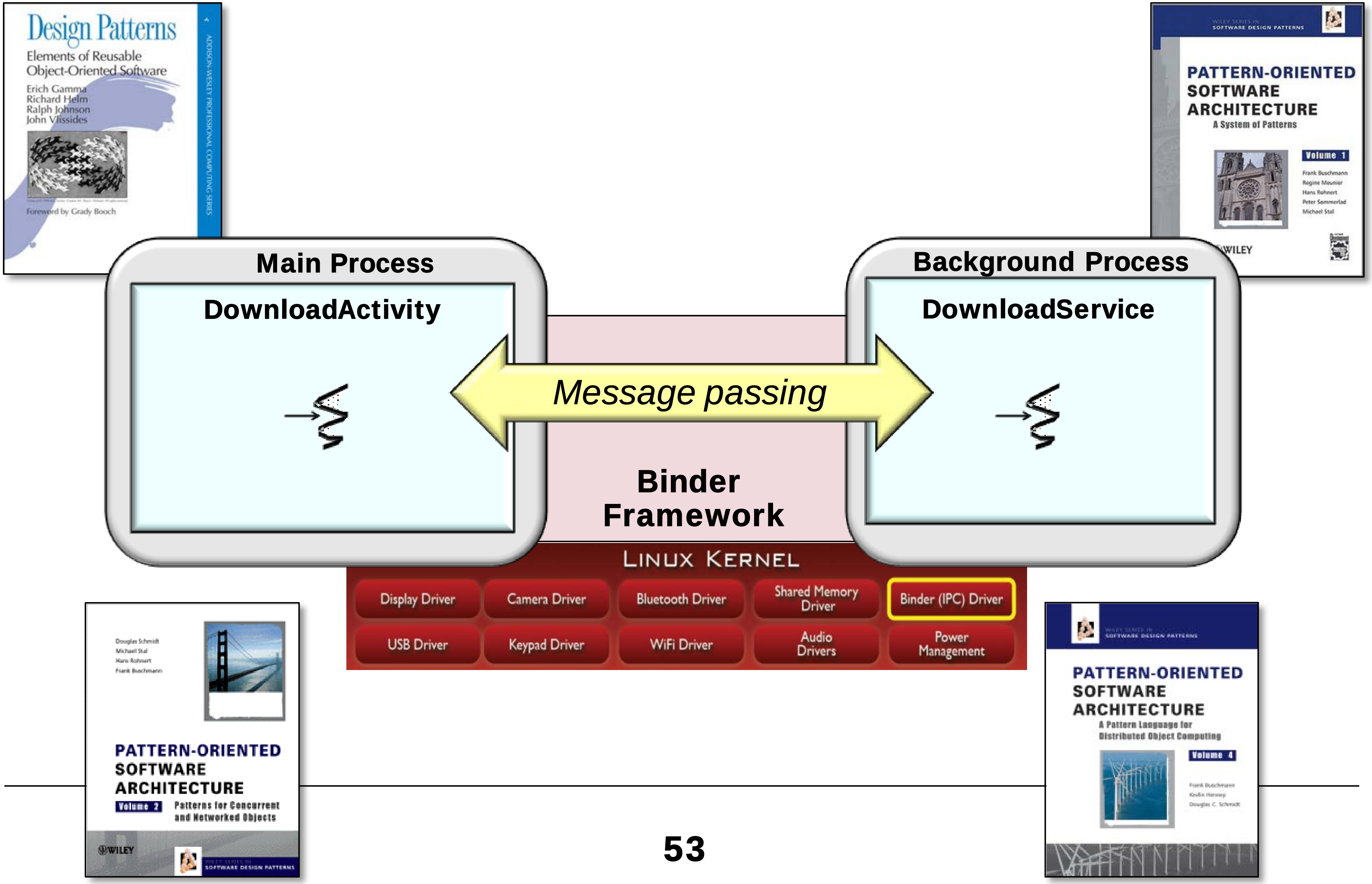
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Active Object* – define service requests on components as the units of concurrency & run service requests on a component in different thread(s) from the requesting client thread



See upcoming part on "Service to Activity Communication via Android Messenger"

Patterns in Communication & Service Frameworks

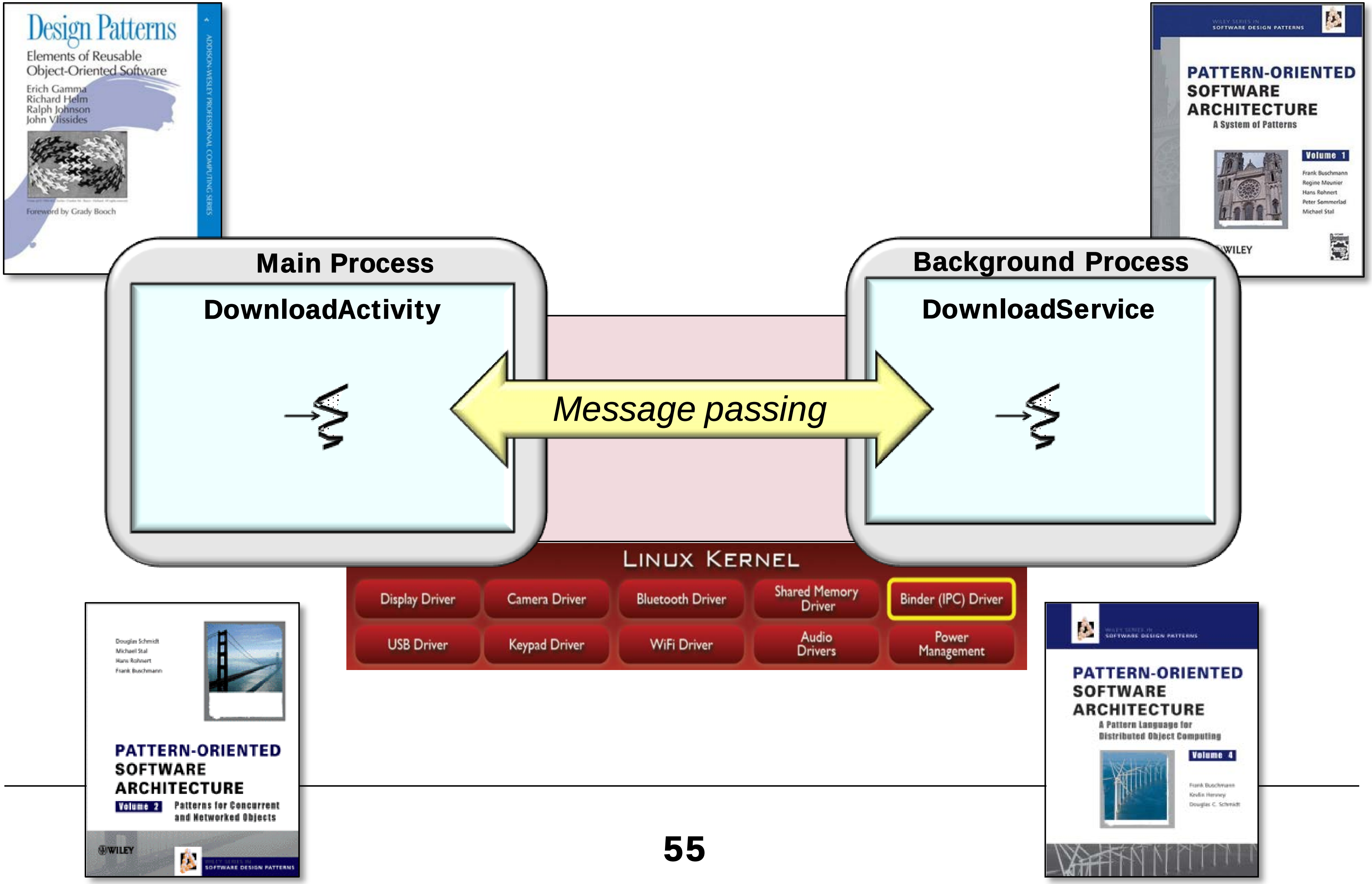
- Android's communication & service frameworks apply POSA & GoF patterns



Patterns Used by Communication & Service Frameworks (Part 3)

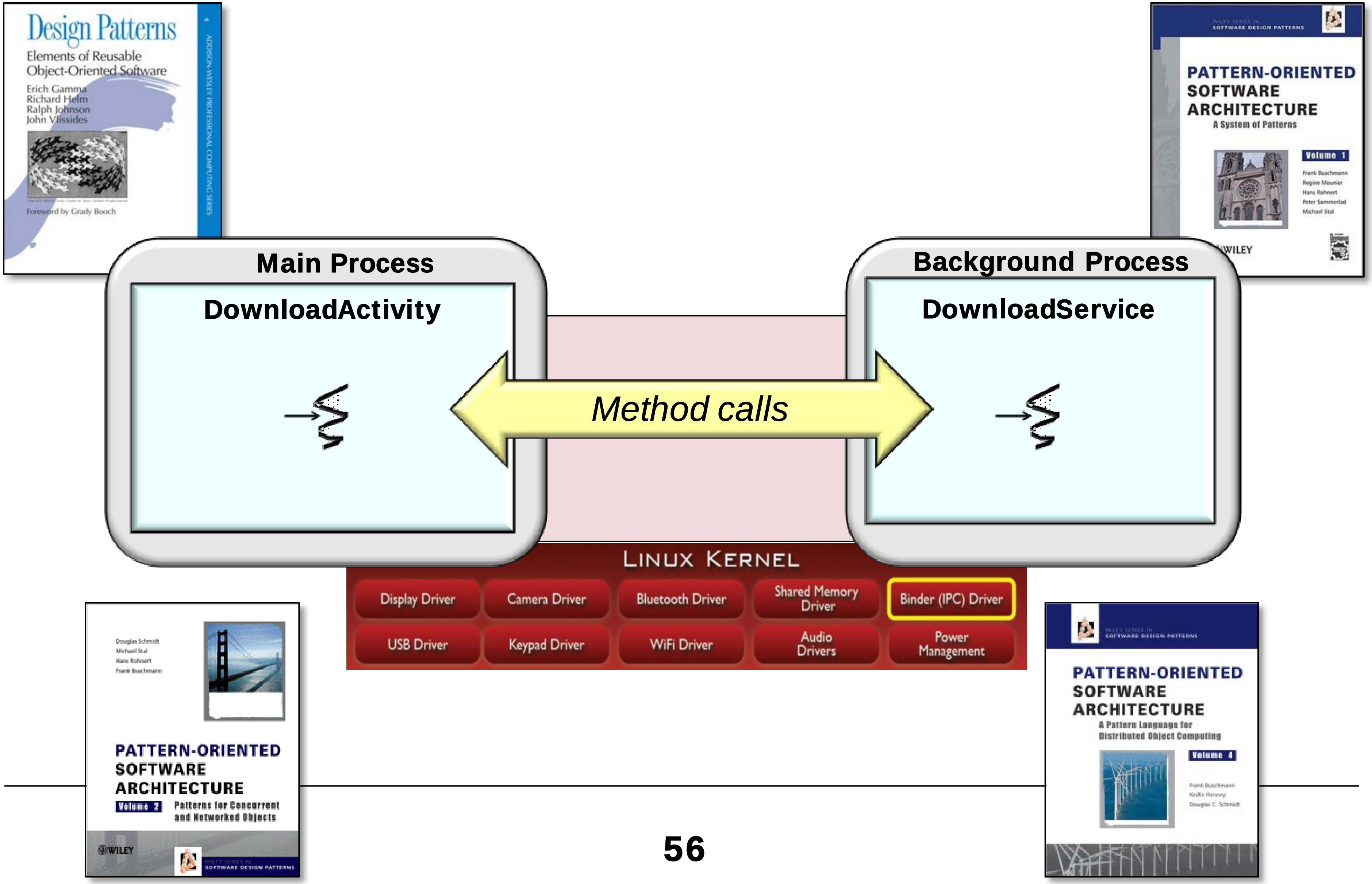
Patterns in Communication & Service Frameworks

- Android’s communication & service frameworks apply POSA & GoF patterns



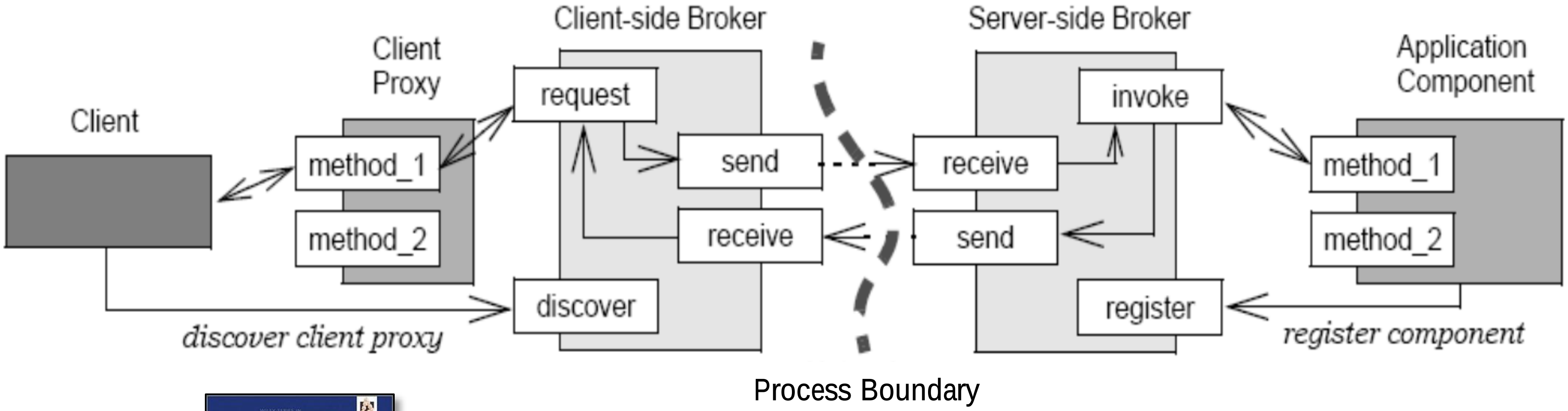
Patterns in Communication & Service Frameworks

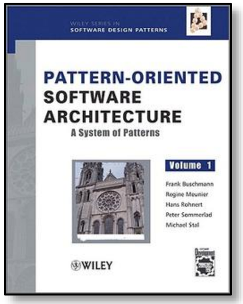
- Android's communication & service frameworks apply POSA & GoF patterns



Patterns in Communication & Service Frameworks

- Android’s communication & service frameworks apply POSA & GoF patterns
  - *Broker* – Connect clients with remote objects by mediating invocations from clients to remote objects, while encapsulating the details of local and/or remote inter-process communication (IPC)




WILEY SERIES IN SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A System of Patterns

Volume 1

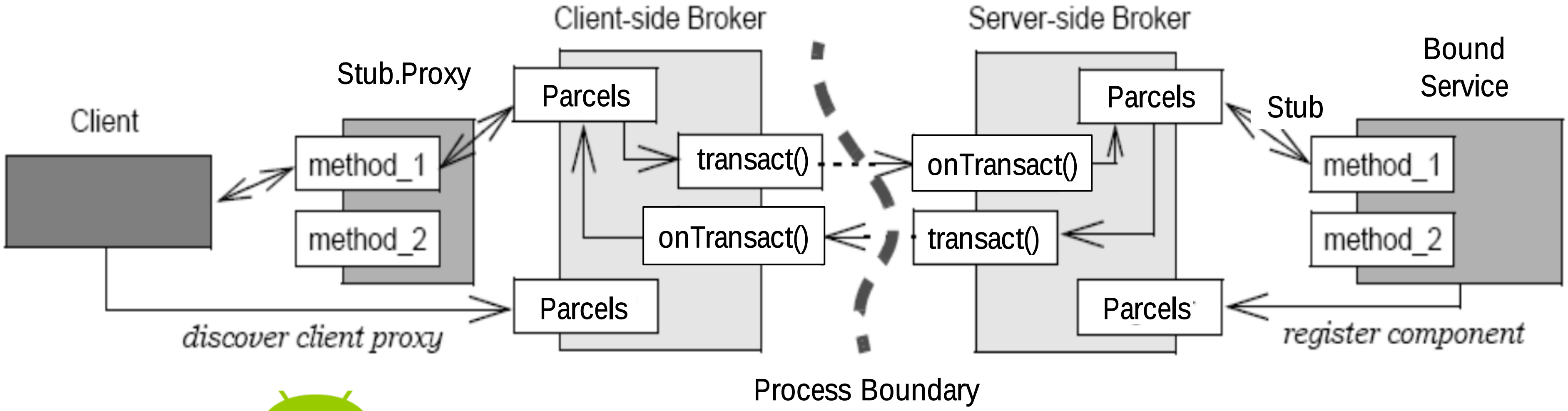
Frank Buschmann
Regine Meunier
Hans Rohnert
Peter Sommerlad
Michael Stal

WILEY

See www.kircher-schwanninger.de/michael/publications/BrokerRevisited.pdf

Patterns in Communication & Service Frameworks

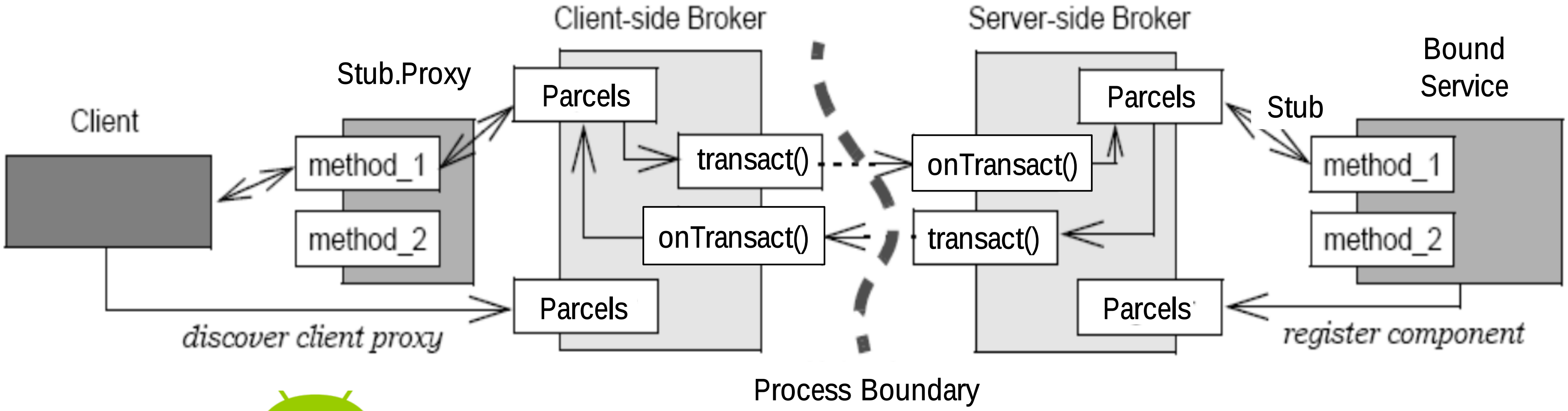
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Broker* – Connect clients with remote objects by mediating invocations from clients to remote objects, while encapsulating the details of local and/or remote inter-process communication (IPC)



See upcoming parts on "Programming Bound Services"

Patterns in Communication & Service Frameworks

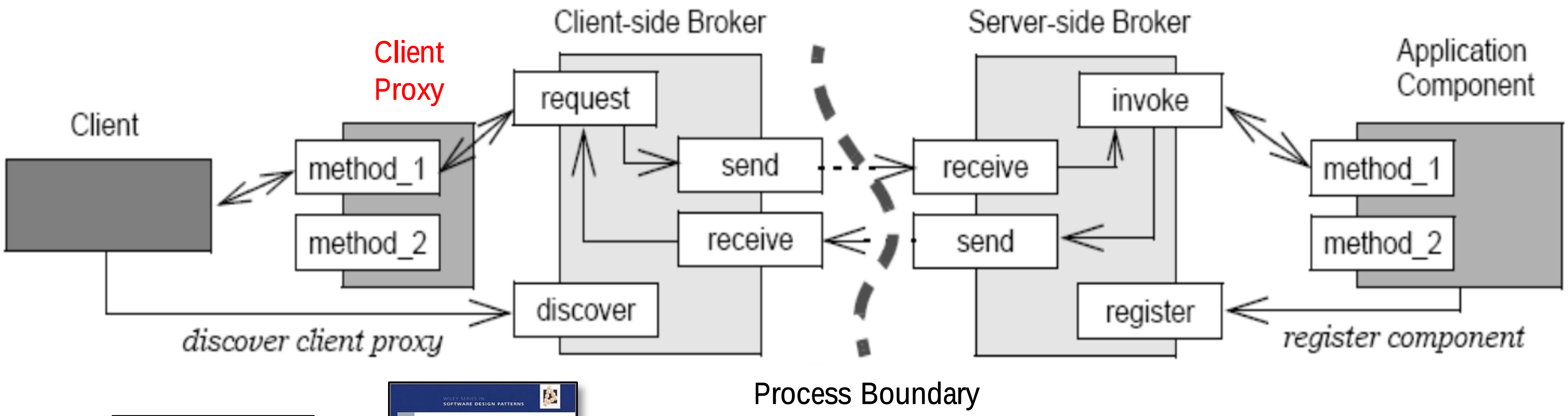
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Broker* – Connect clients with remote objects by mediating invocations from clients to remote objects, while encapsulating the details of local and/or remote inter-process communication (IPC)



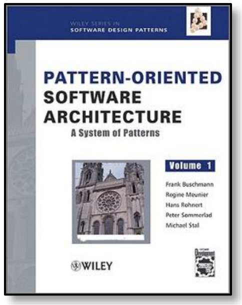
See www.dre.vanderbilt.edu/~schmidt/PDF/remoting-patterns.pdf

Patterns in Communication & Service Frameworks

- Android's communication & service frameworks apply POSA & GoF patterns
  - *Proxy* – Provide a surrogate or placeholder for another object to control access to it



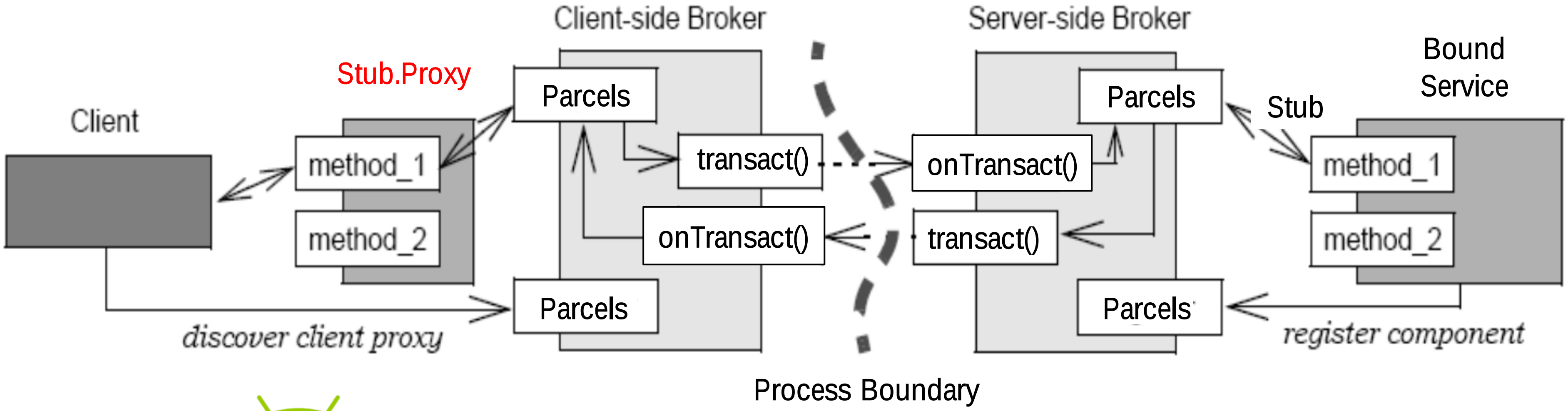
Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



en.wikipedia.org/wiki/Proxy_pattern has more info

Patterns in Communication & Service Frameworks

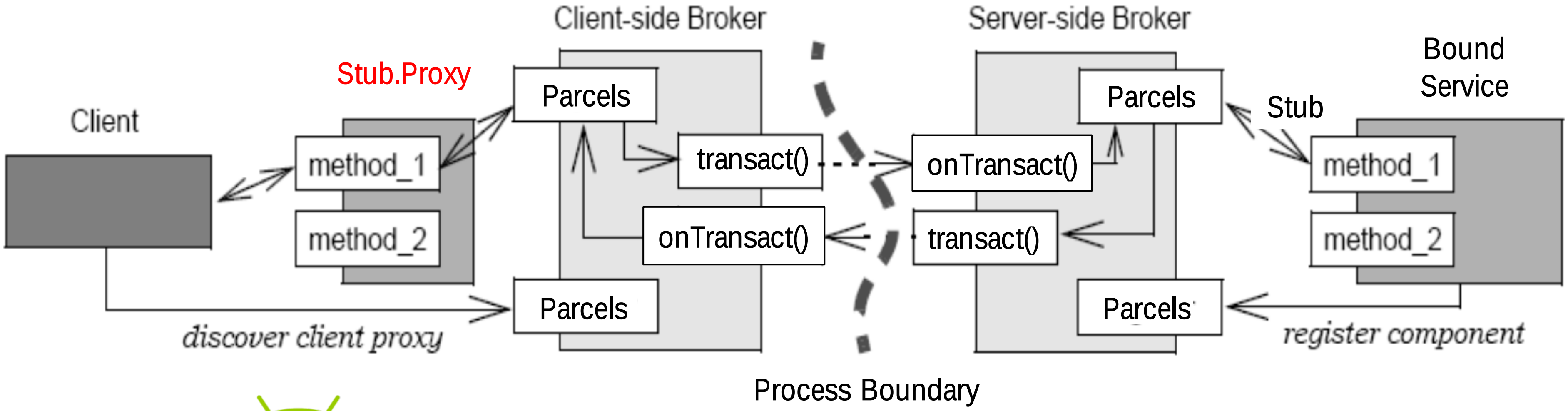
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Proxy* – Provide a surrogate or placeholder for another object to control access to it



developer.android.com/guide/components/aidl.html explains (un)marshaling

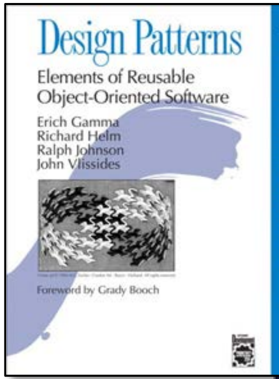
Patterns in Communication & Service Frameworks

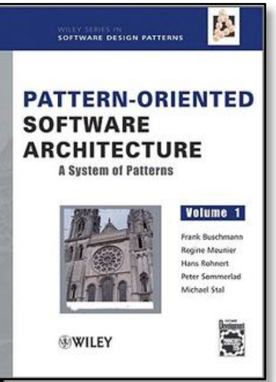
- Android's communication & service frameworks apply POSA & GoF patterns
  - *Proxy* – Provide a surrogate or placeholder for another object to control access to it

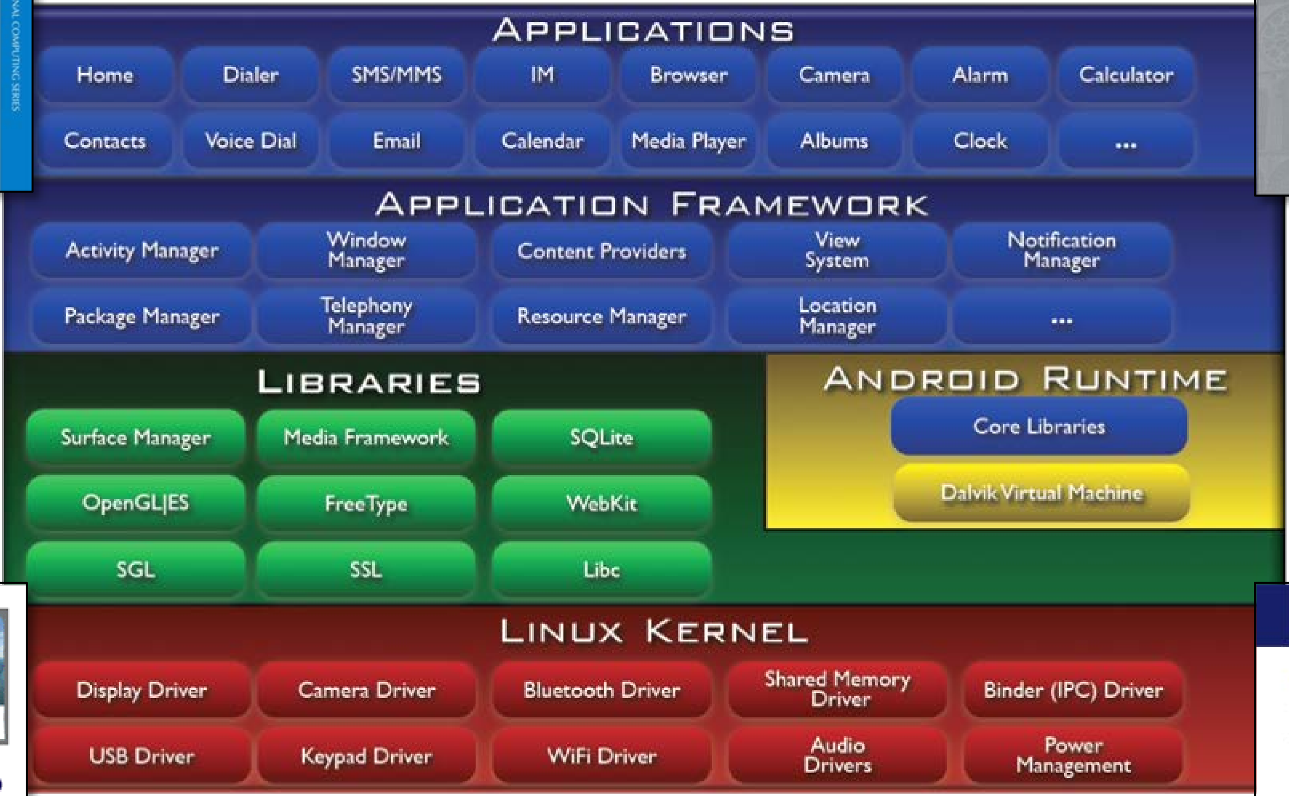


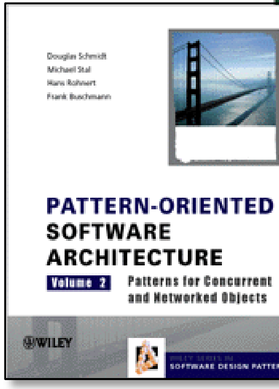
See upcoming part on "The Proxy Pattern"

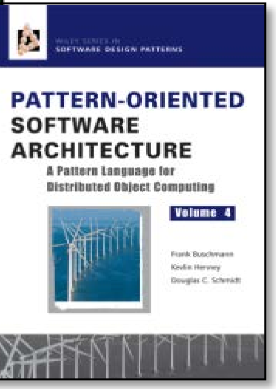
Patterns in Communication & Service Frameworks


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES






Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS

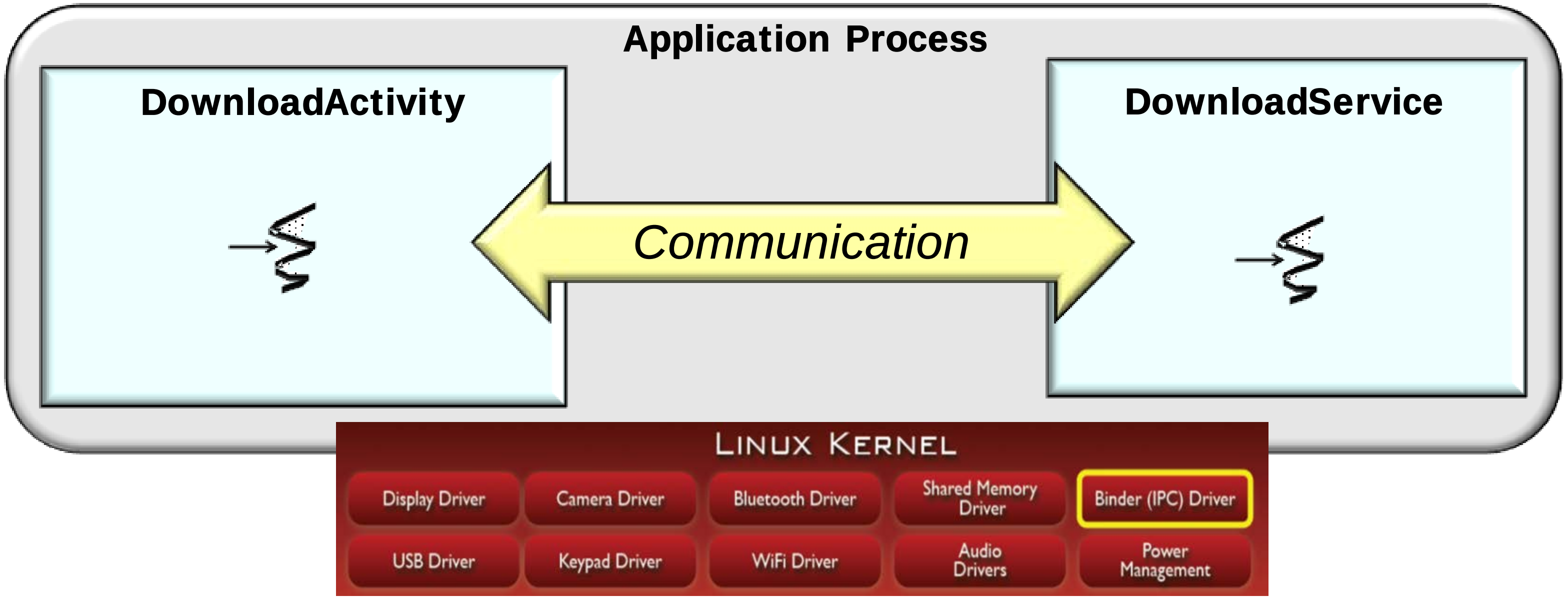

WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Object Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

See upcoming section on “Concurrency & Communication Patterns”

Summary

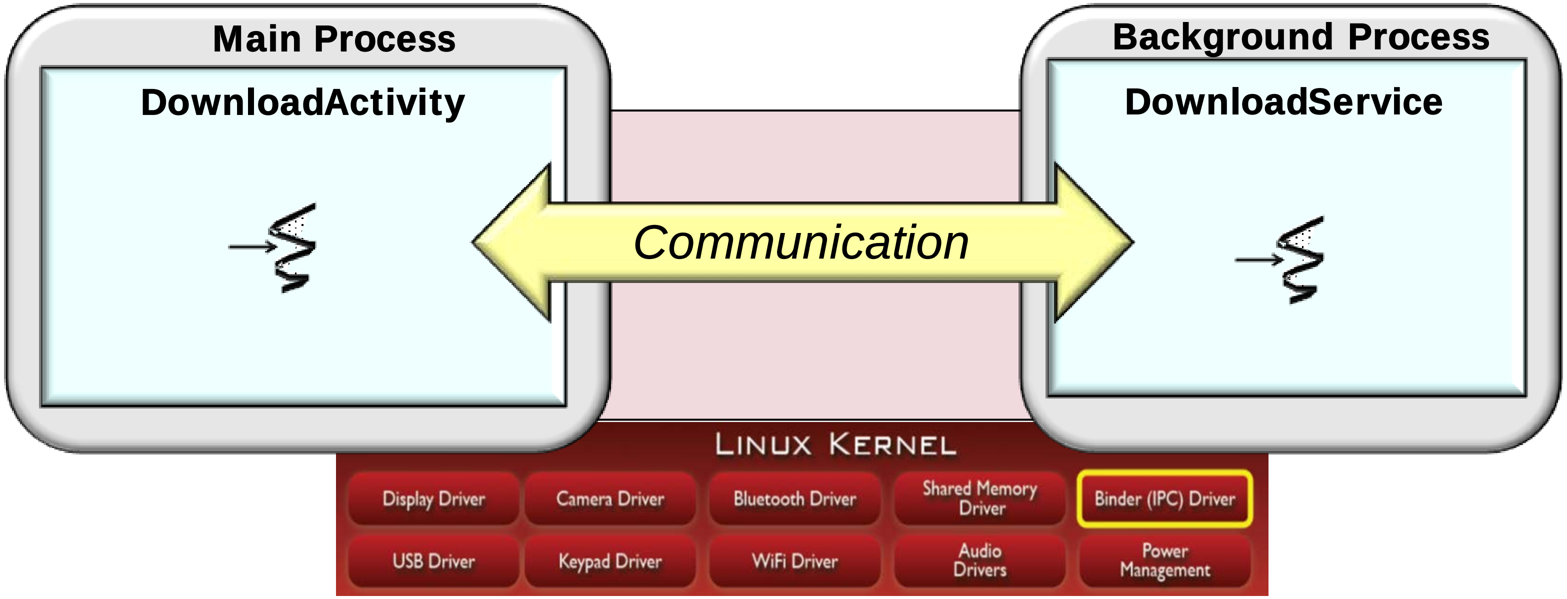
Summary

- Android provides mechanisms that enable Activity & Service communication



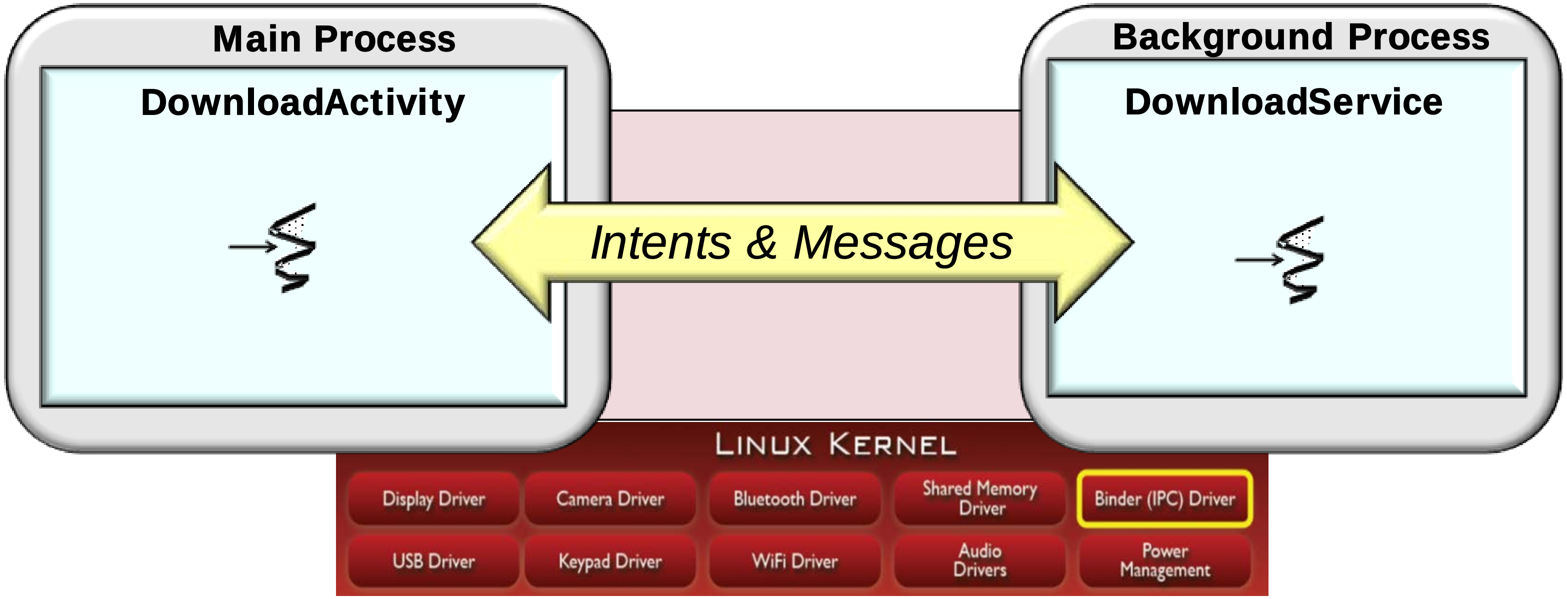
Summary

- Android provides mechanisms that enable Activity & Service communication



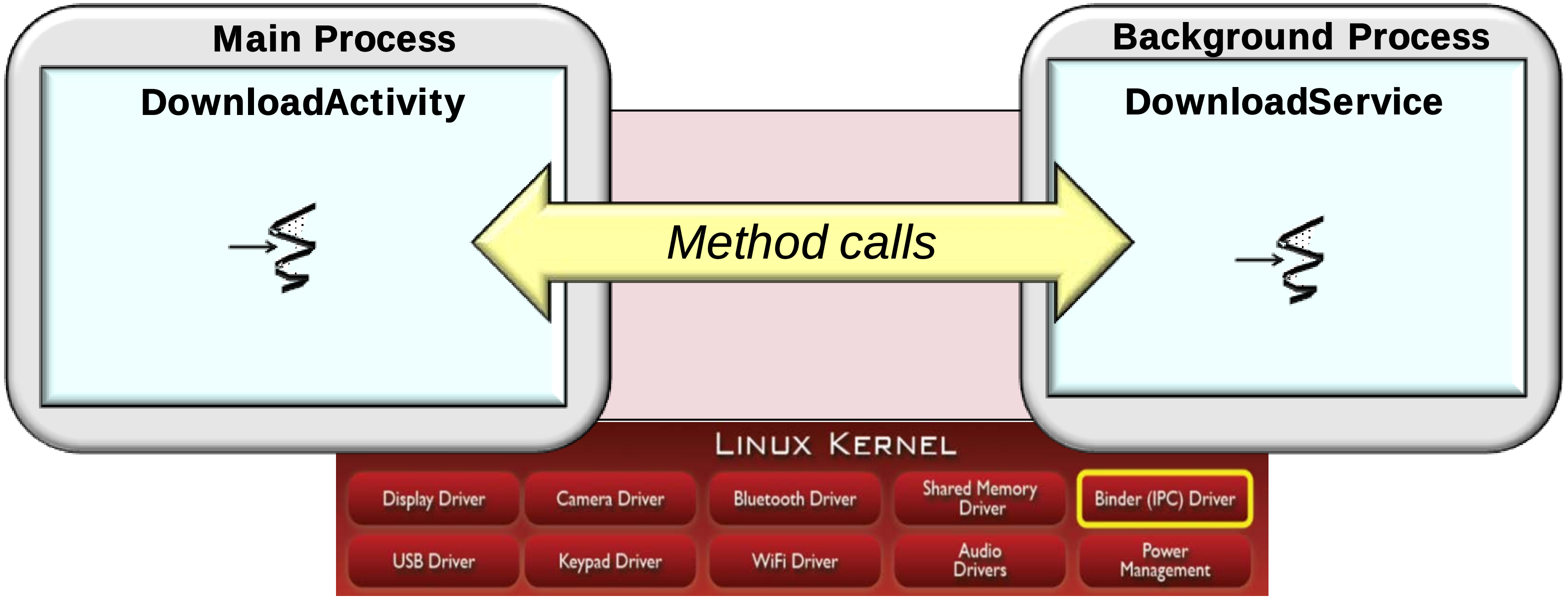
Summary

- Android provides mechanisms that enable Activity & Service communication



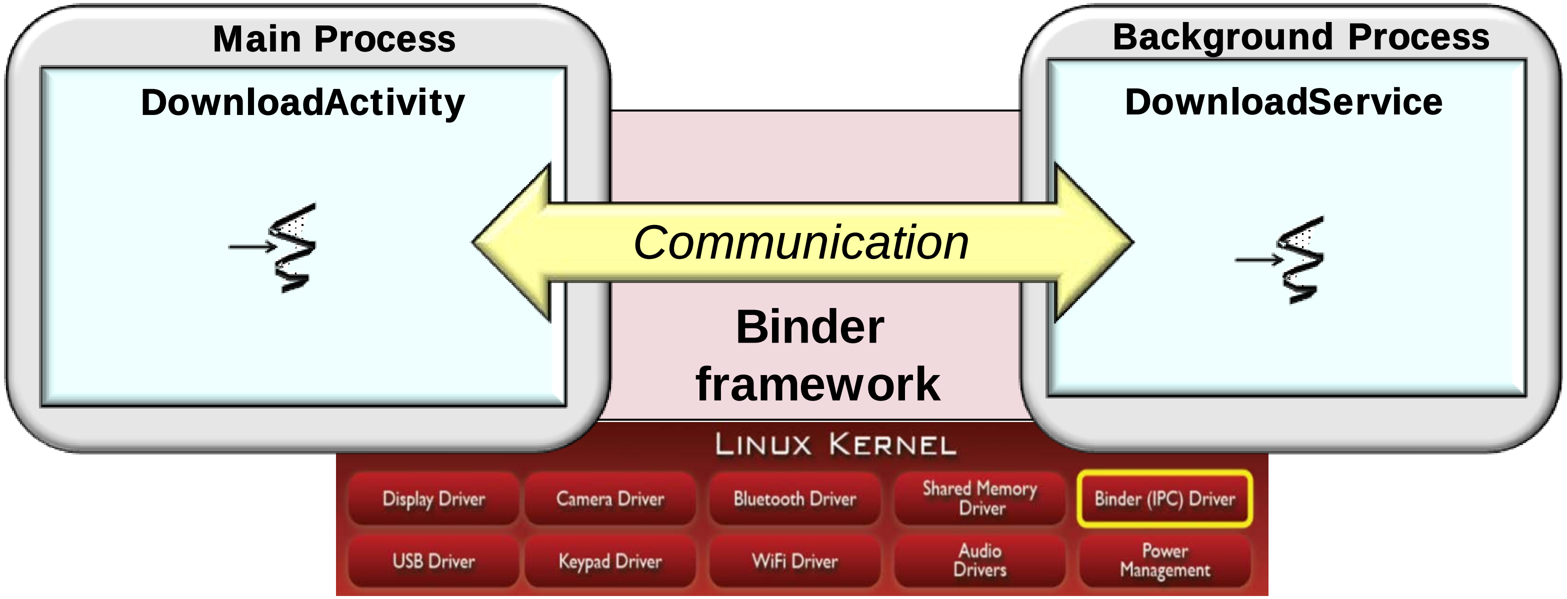
Summary

- Android provides mechanisms that enable Activity & Service communication

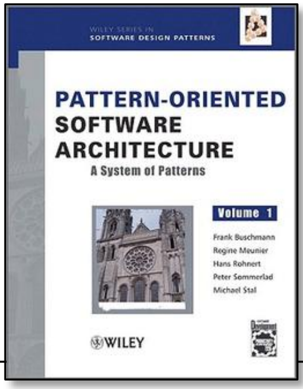


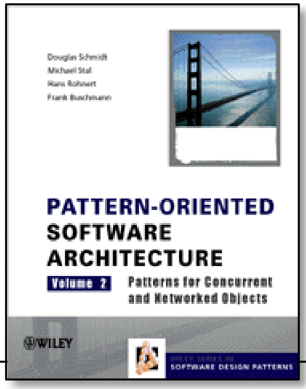
Summary

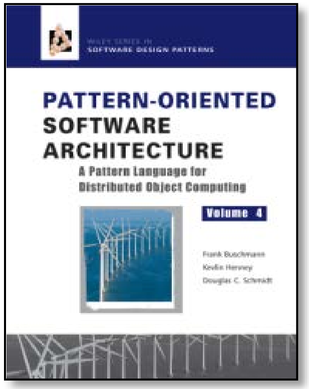
- Android provides mechanisms that enable Activity & Service communication



Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

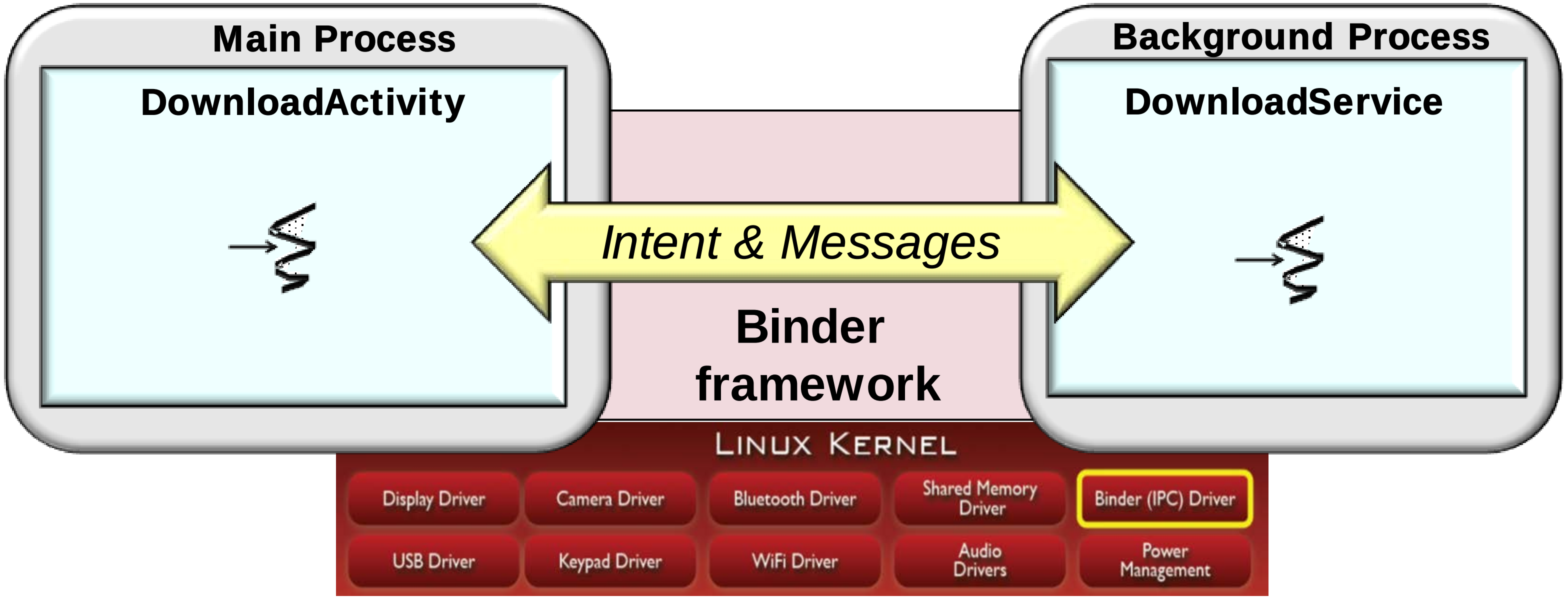

WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A System of Patterns
Volume 1
Frank Buschmann
Regine Meunier
Hans Rohnert
Peter Sommerlad
Michael Stal
WILEY

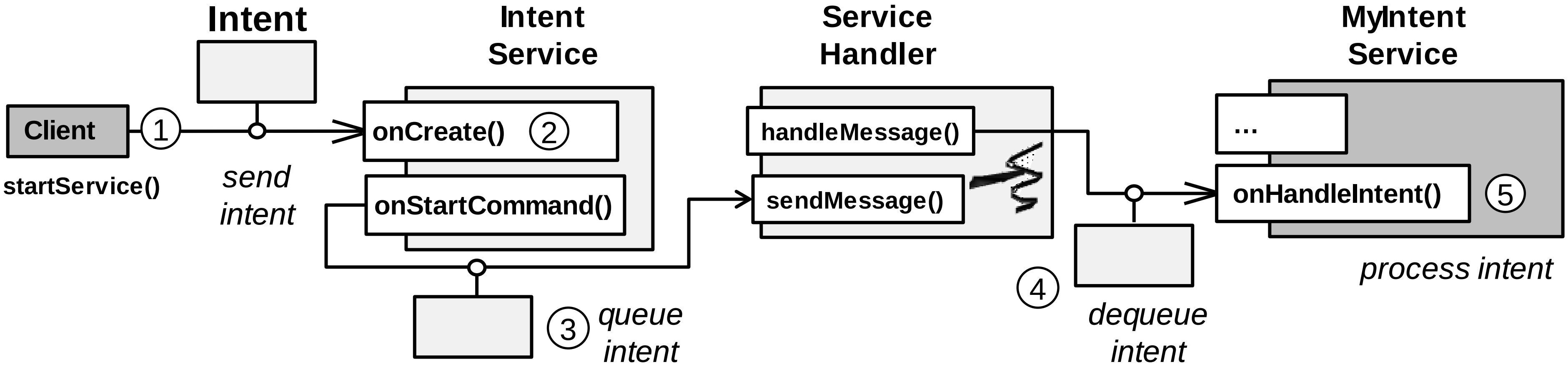

Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS


WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Object Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

Summary

- Android provides mechanisms that enable Activity & Service communication

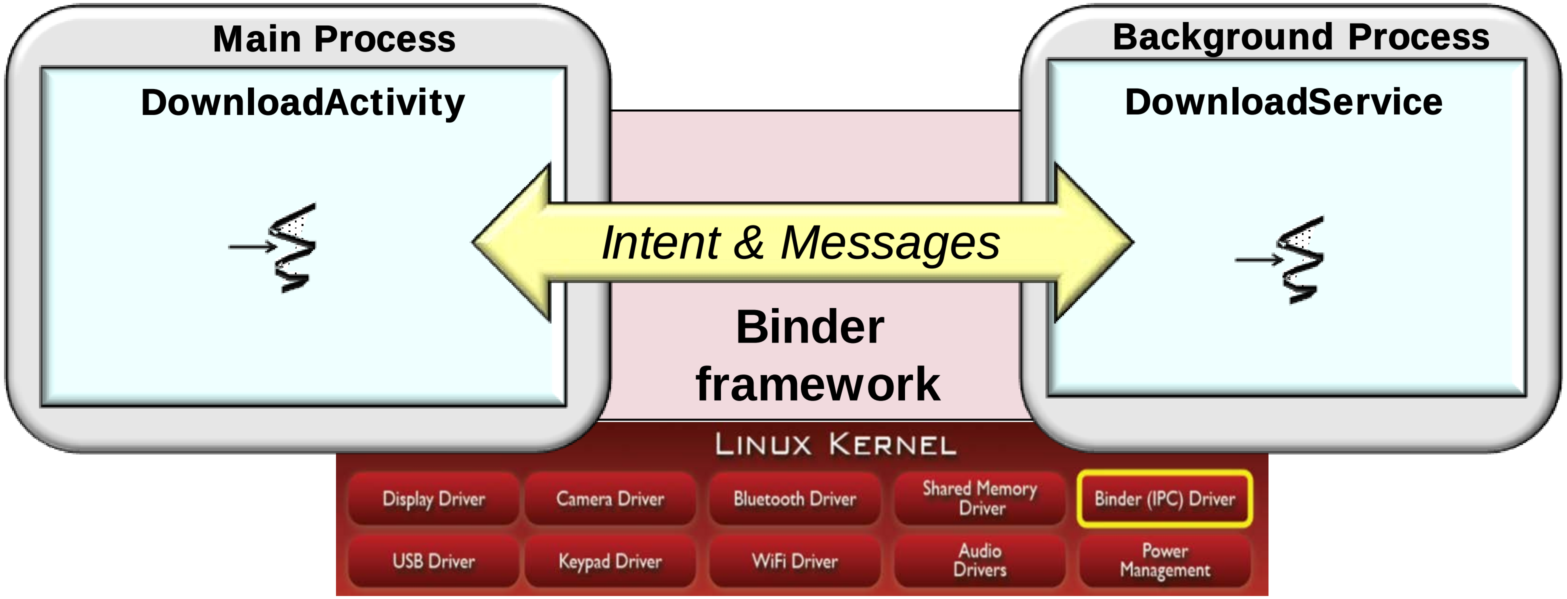


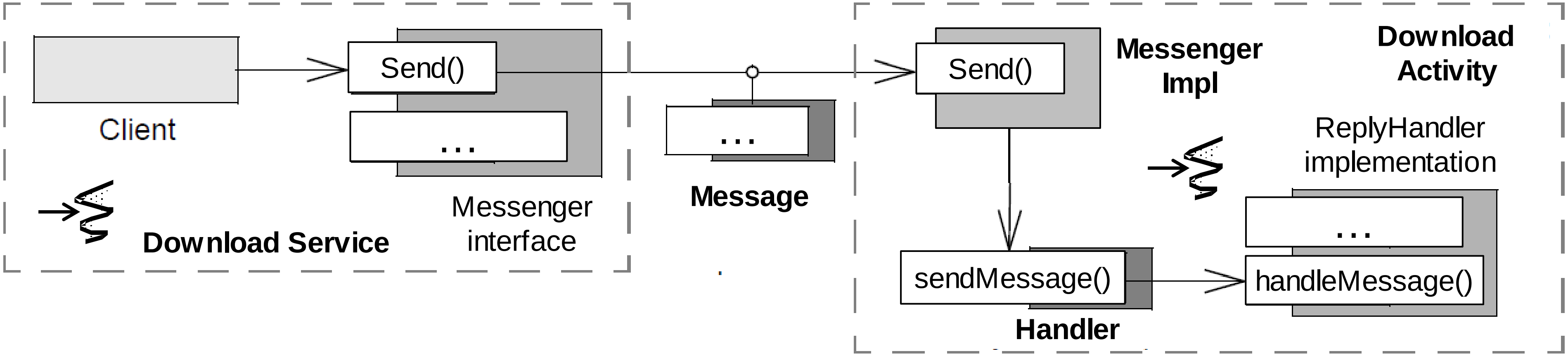


IntentService-based programs apply the *Command Processor* pattern

Summary

- Android provides mechanisms that enable Activity & Service communication

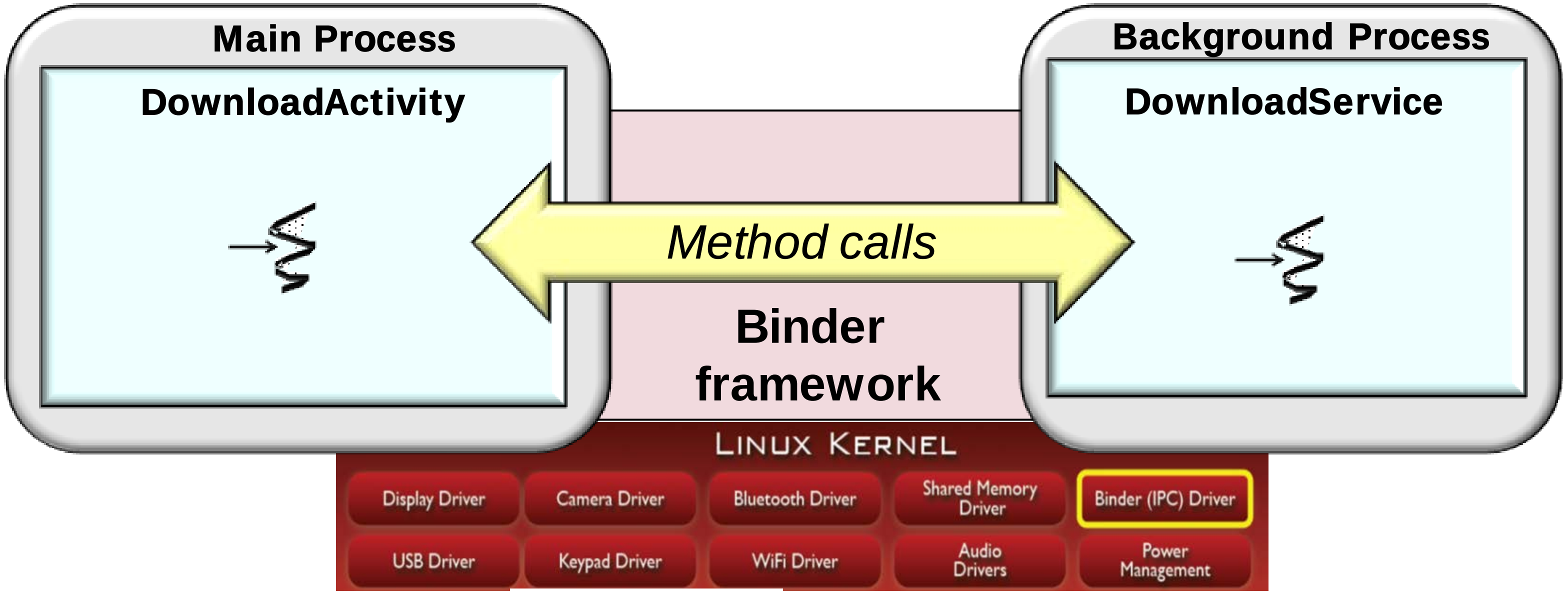


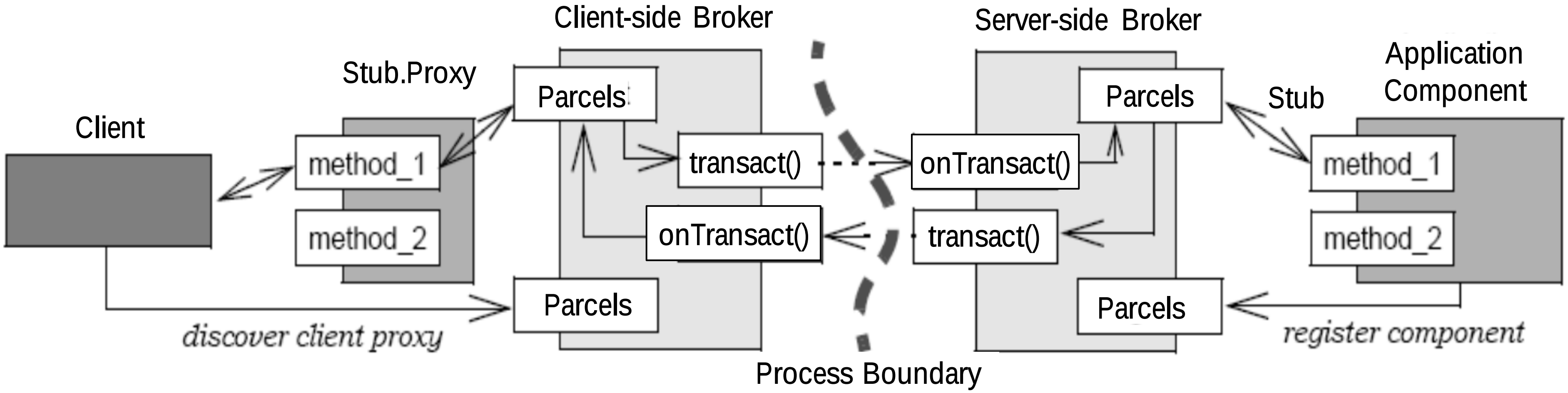


Messenger-based programs apply the *Active Object* pattern

Summary

- Android provides mechanisms that enable Activity & Service communication

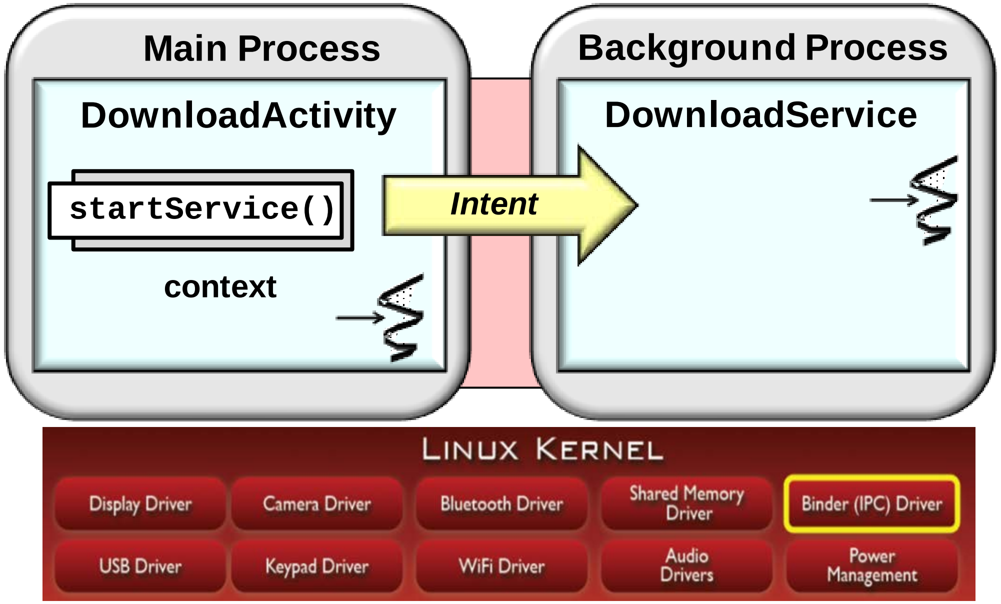




AIDL-based solutions apply the Broker pattern

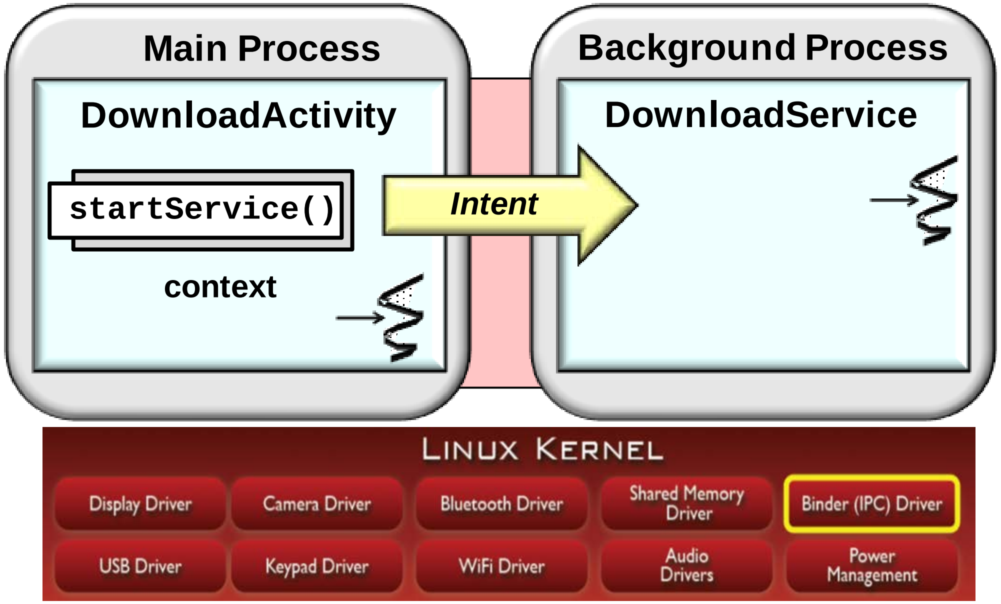
Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services



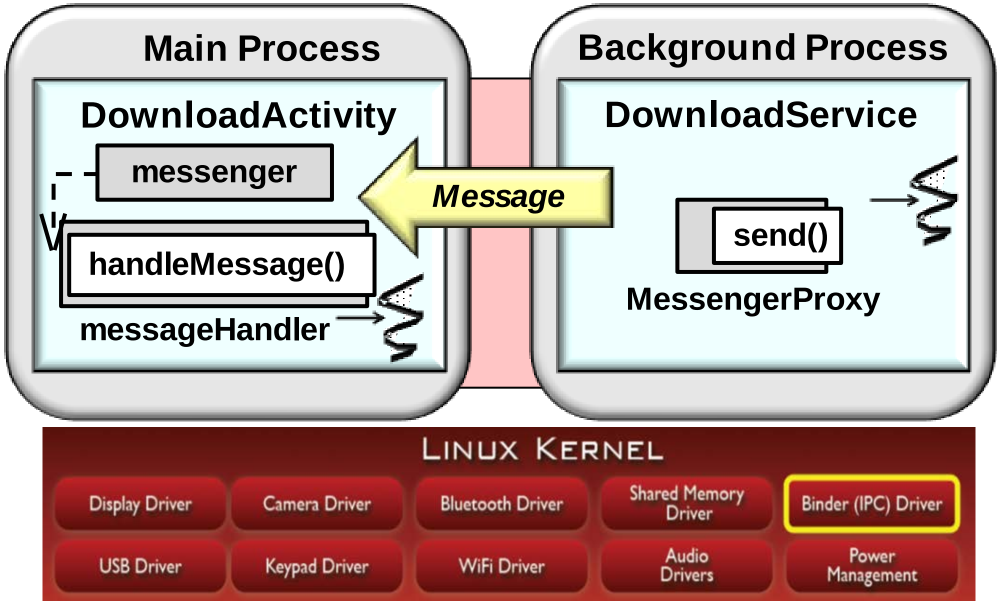
Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
  - However, it's also limited since there's no equivalent interface for the Service to pass an Intent reply back to the Activity



Summary

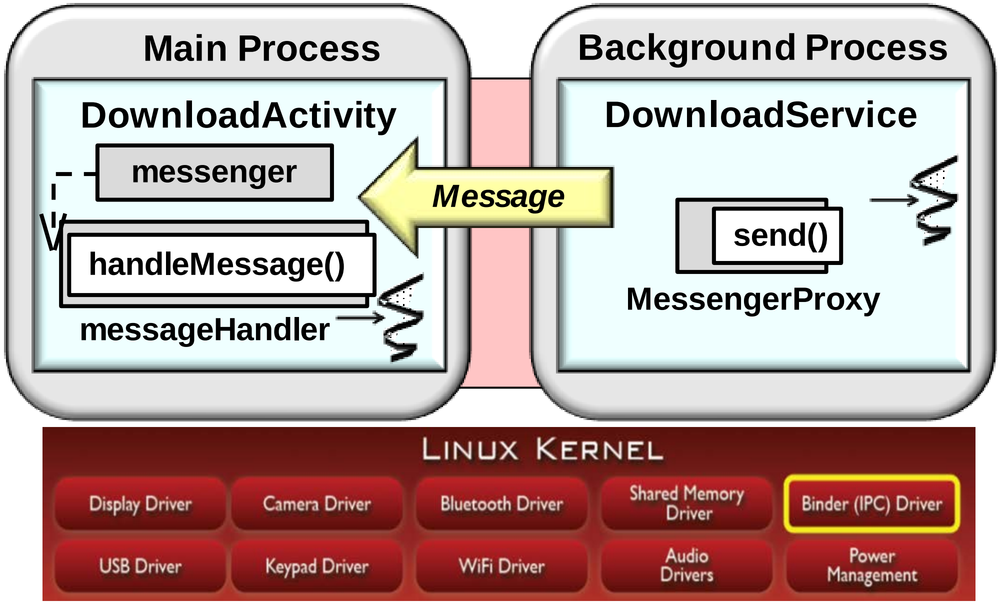
- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services



- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services

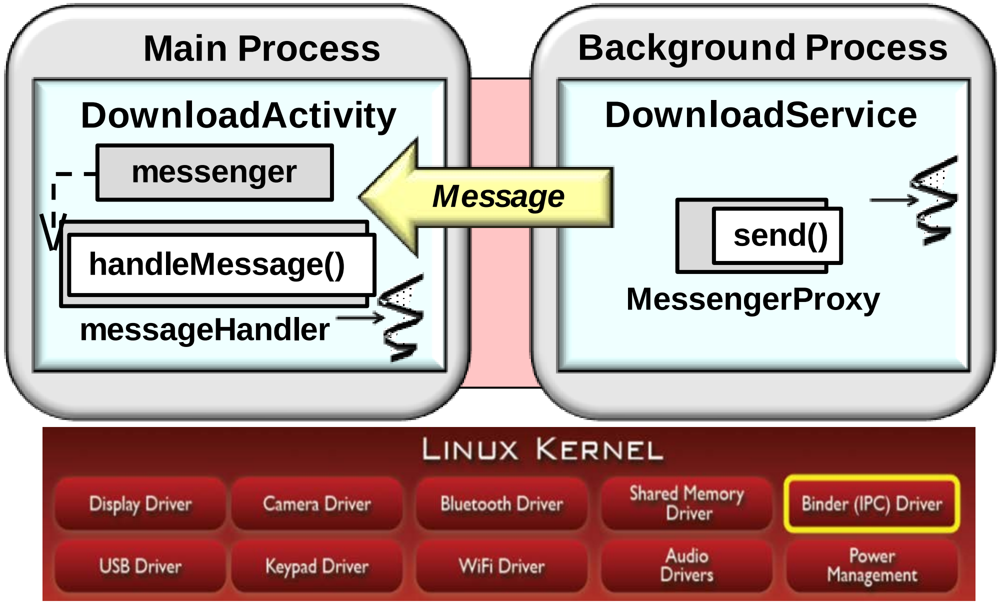
Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services
  - Often used to send replies from a Started Service back to the Activity that initiated it



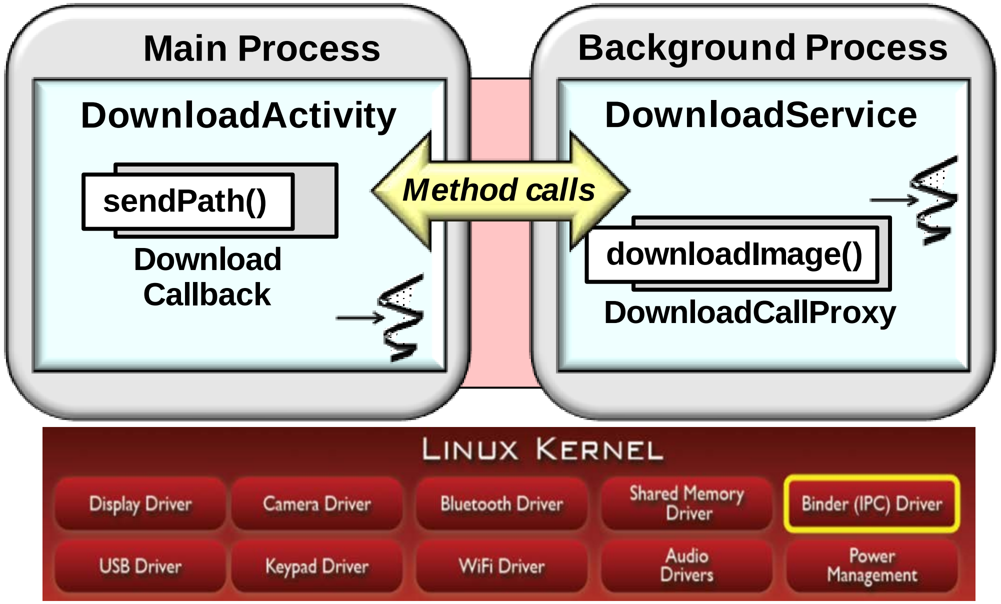
Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services
    - Often used to send replies from a Started Service back to the Activity that initiated it
    - Harder to use for more complex interactions involving complex data types



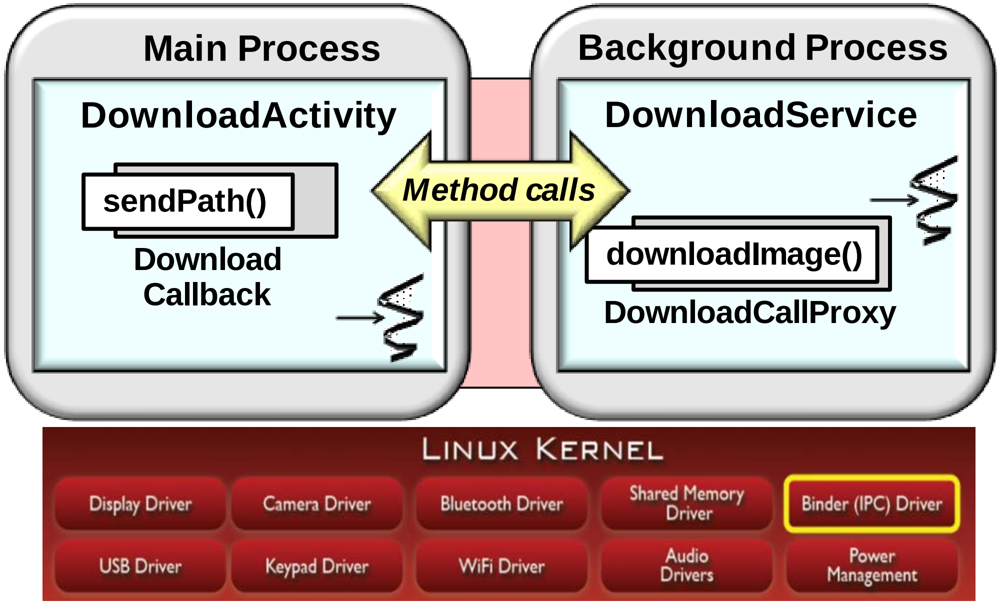
Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services
- Invoking methods via AIDL Stubs is often more effective & efficient for complex interactions



Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services

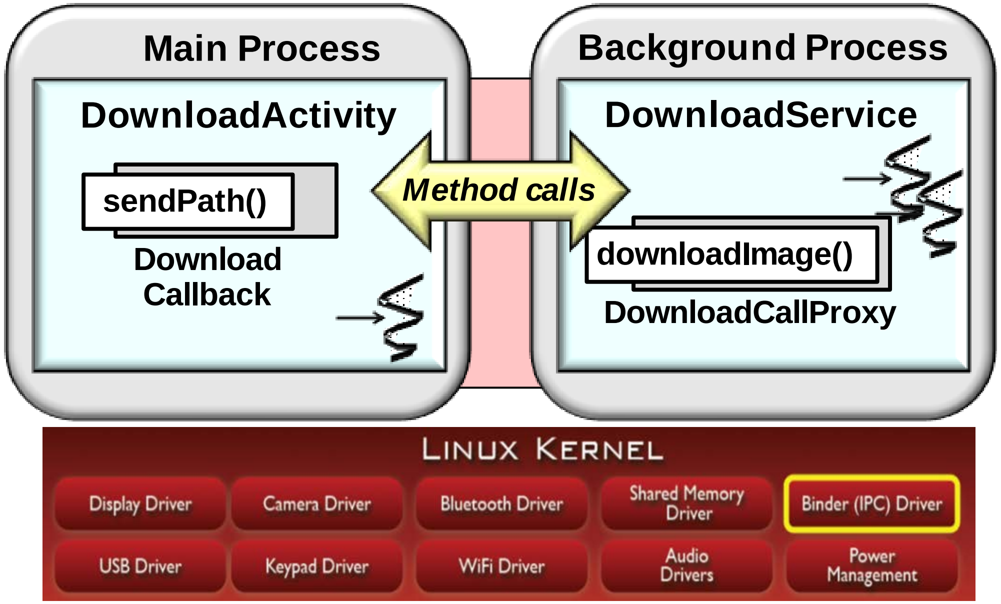


- Invoking methods via AIDL Stubs is often more effective & efficient for complex interactions
  - AIDL compiler generates Stubs that perform (de)marshaling

See en.wikipedia.org/wiki/Marshalling_(computer_science)

Summary

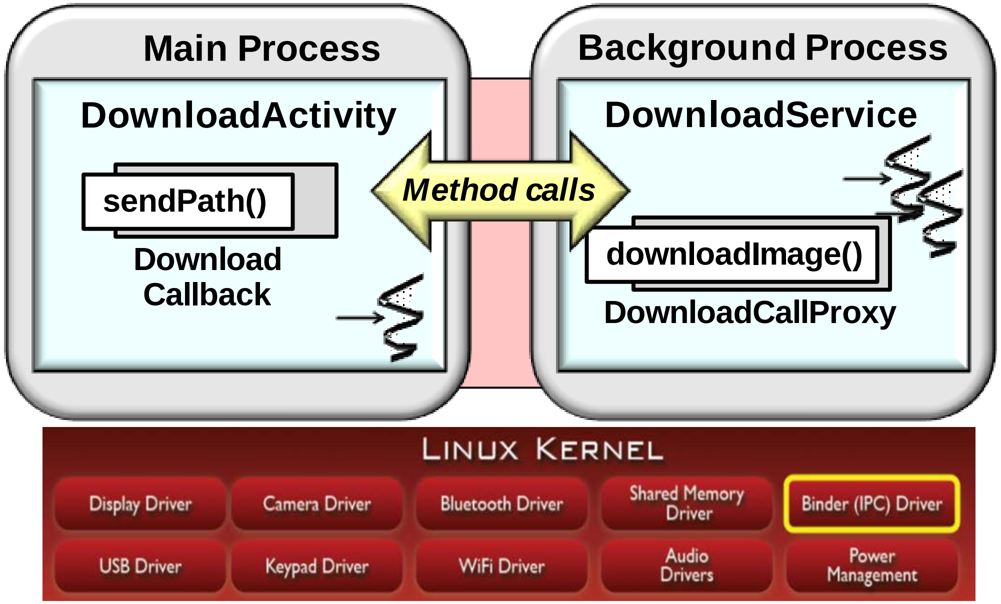
- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services
- Invoking methods via AIDL Stubs is often more effective & efficient for complex interactions
  - AIDL compiler generates Stubs that perform (de)marshaling
  - AIDL-based method calls run concurrently in a pool of Threads



developer.android.com/guide/components/aidl.html has info on AIDL thread pools

Summary

- Android provides mechanisms that enable Activity & Service communication
- Passing Intents via startService() or bindService() is straightforward for oneway communication from Activities to Services
- Sending Messages via Messengers is also straightforward for simple interactions between Activities & Services
- Invoking methods via AIDL Stubs is often more effective & efficient for complex interactions
  - AIDL compiler generates Stubs that perform (de)marshaling
  - AIDL-based method calls run concurrently in a pool of Threads
  - ,



- In contrast, Messengers don't require any particular concurrency model