The Android Linux Kernel (Part 3): Android Kernel Extensions

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

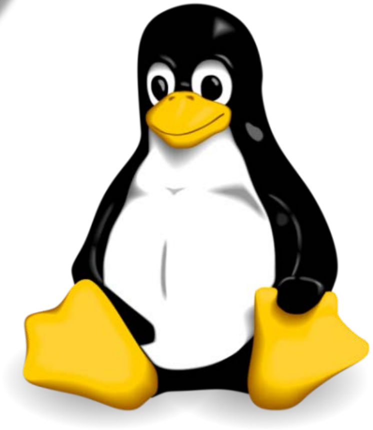
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

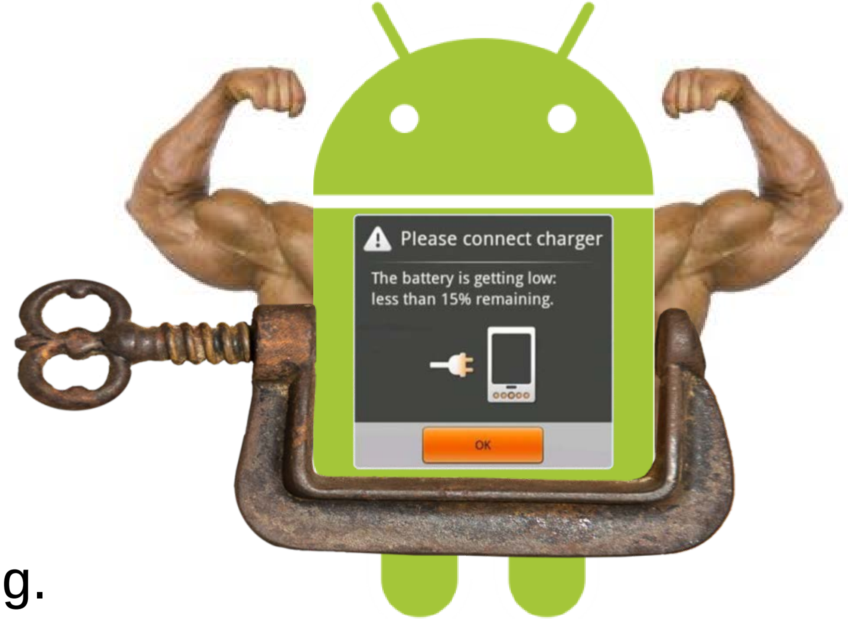
Learning Objectives in this Part of the Lesson

1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux's local & remote communication mechanisms
3. Know how Android Linux's processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel



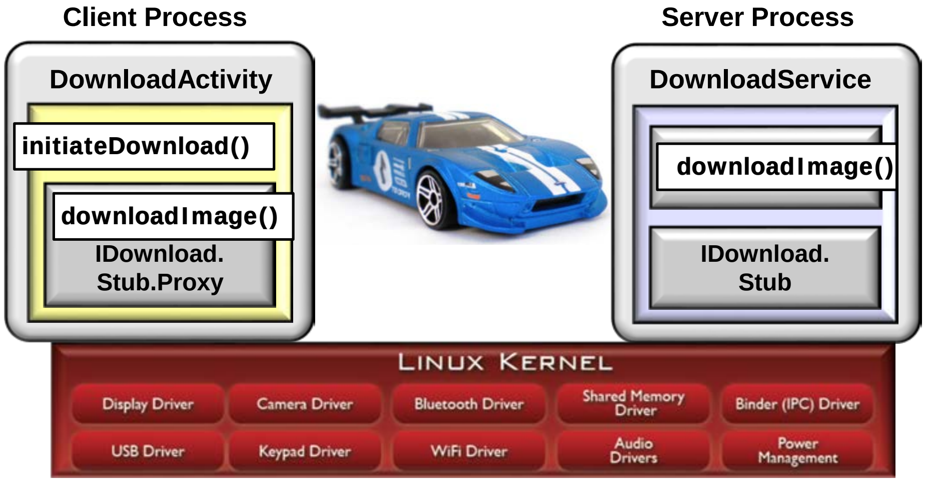
Learning Objectives in this Part of the Lesson

1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux's local & remote communication mechanisms
3. Know how Android Linux's processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel, e.g.
   - Manage memory & power effectively on mobile devices



Learning Objectives in this Part of the Lesson

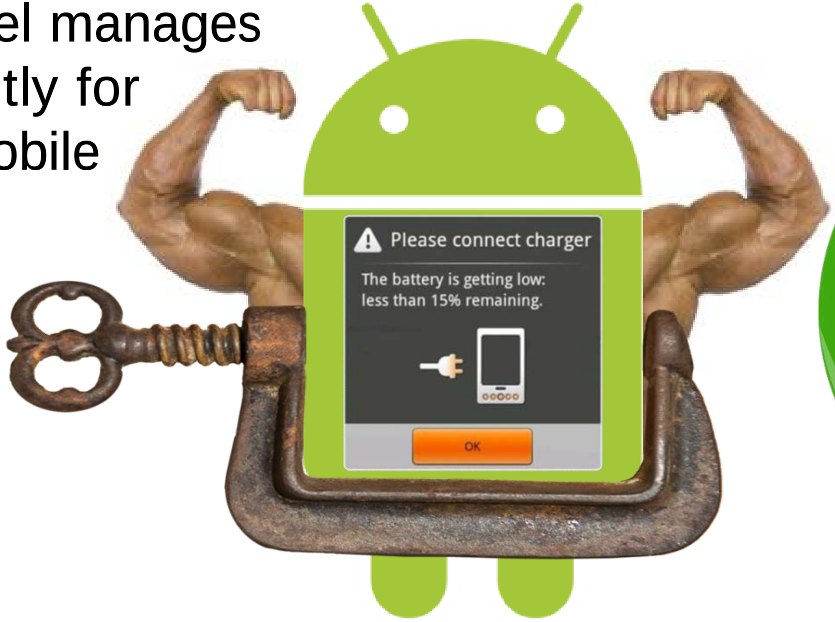
1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux's local & remote communication mechanisms
3. Know how Android Linux's processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel, e.g.
   - Manage memory & power effectively on mobile devices
   - Accelerate performance for local inter-process communication (IPC)



Android Linux Extensions: Memory Management

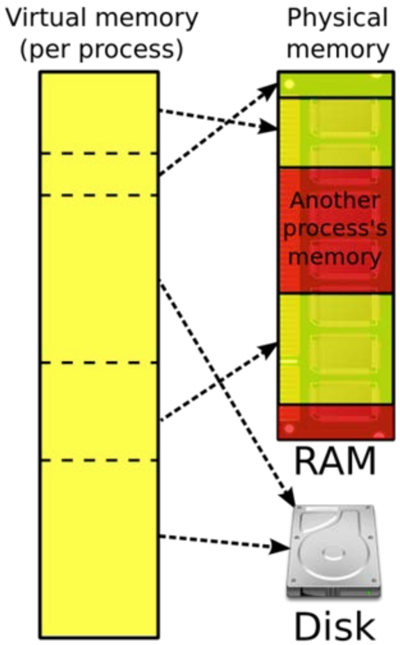
Android Linux Extensions: Memory Management

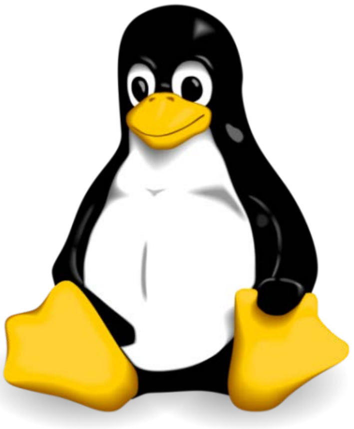
- The Android Linux kernel manages primary storage efficiently for memory-constrained mobile devices



Android Linux Extensions: Memory Management

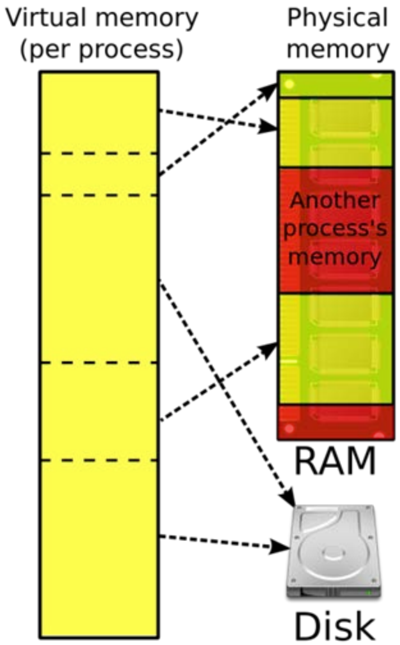
- GNU Linux kernel shields programmers from primary storage constraints

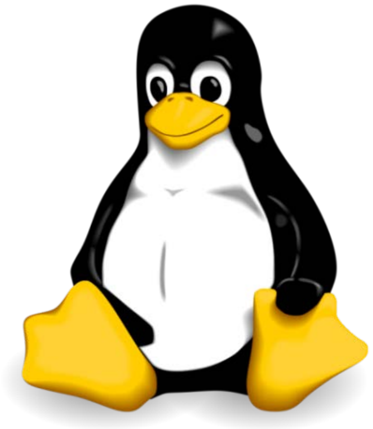




Android Linux Extensions: Memory Management

- GNU Linux kernel shields programmers from primary storage constraints, e.g.
  - Linux's virtual memory manager allows applications to access an address space larger than RAM

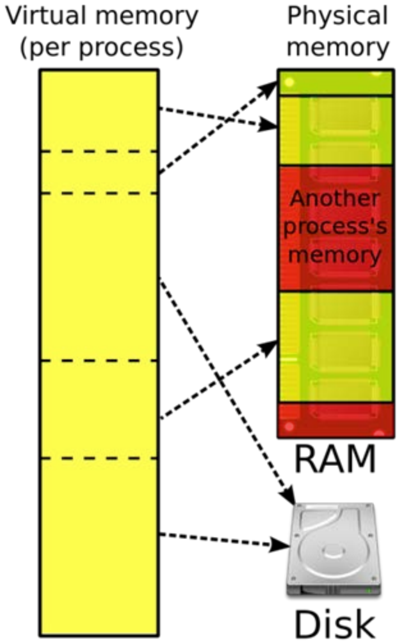


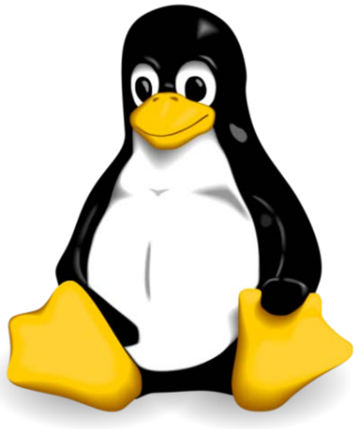


See Part 1 of this lesson on "Overview of the Android Linux Kernel"

Android Linux Extensions: Memory Management

- GNU Linux kernel shields programmers from primary storage constraints, e.g.
  - Linux's virtual memory manager allows applications to access an address space larger than RAM
  - Swapping is used to automatically exchange the contents of primary storage with secondary storage

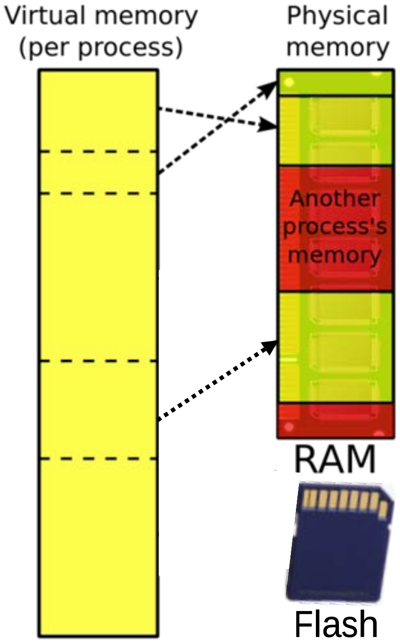




See stackoverflow.com/a/4420560/3312330

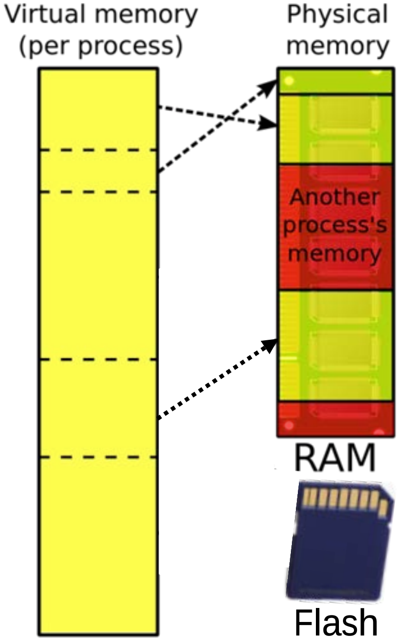
Android Linux Extensions: Memory Management

- The Android Linux virtual memory manager behaves differently than GNU Linux



Android Linux Extensions: Memory Management

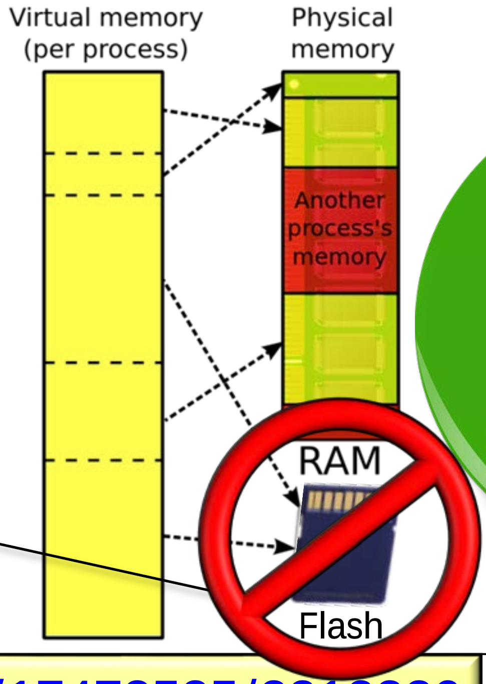
- The Android Linux virtual memory manager behaves a bit differently than GNU Linux
  - e.g., to conserve power Android Linux doesn't provide swap space for RAM



See developer.android.com/training/articles/memory.html#Android

Android Linux Extensions: Memory Management

- The Android Linux virtual memory manager behaves a bit differently than GNU Linux
  - e.g., to conserve power Android Linux doesn't provide swap space for RAM

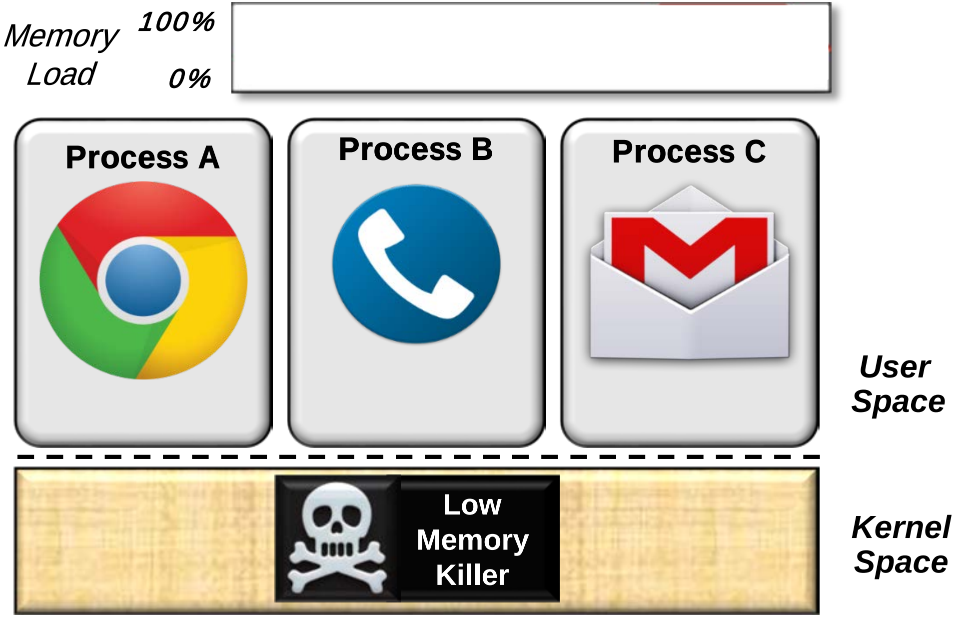


RAM modified by an app is not written automatically out to swap space

See stackoverflow.com/a/17478535/3312330

Android Linux Extensions: Memory Management

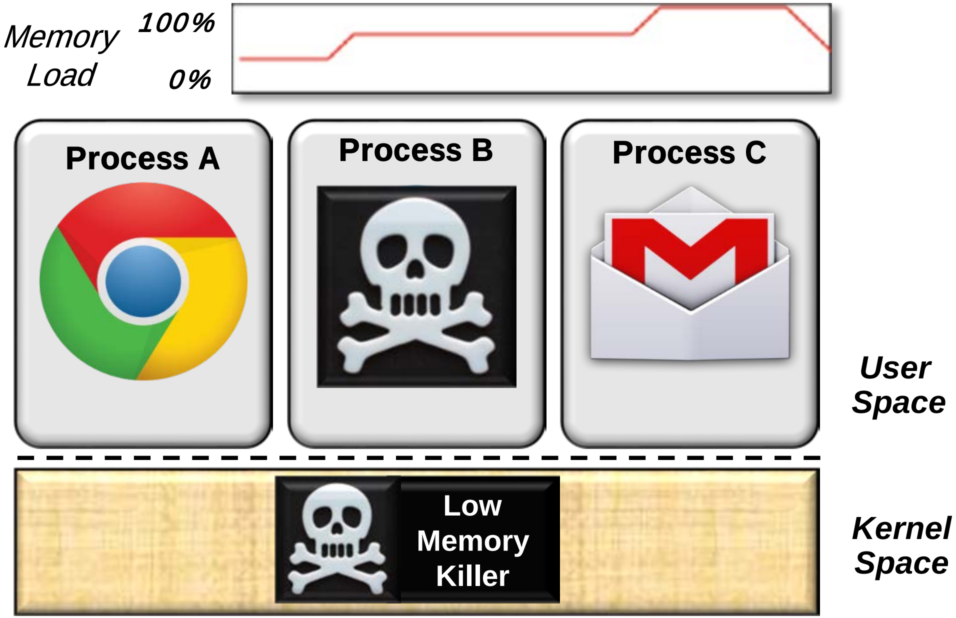
- Android Linux defines a “low memory killer” extension



See www.programering.com/a/MjNzADMwATE.html

Android Linux Extensions: Memory Management

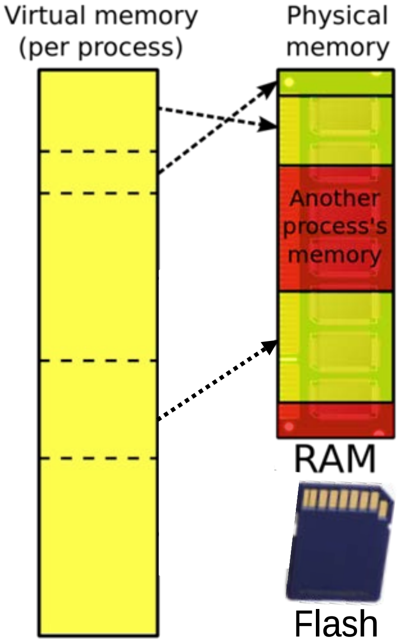
- Android Linux defines a "low memory killer" extension
  - Terminates app components when available RAM falls below a given threshold



See www.programering.com/a/MjNzADMwATE.html

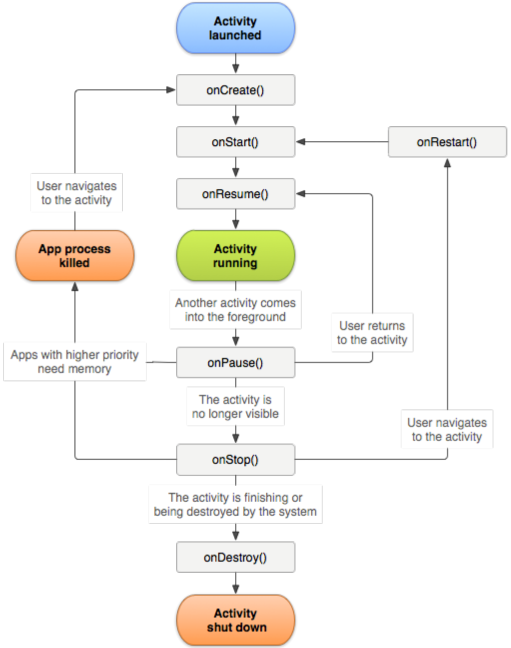
Android Linux Extensions: Memory Management

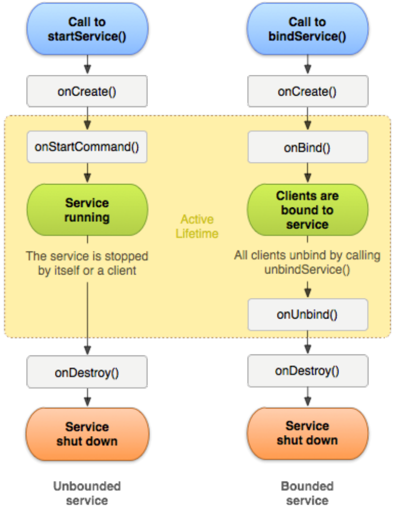
- App developers must know Android Linux kernel storage extensions



Android Linux Extensions: Memory Management

- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods

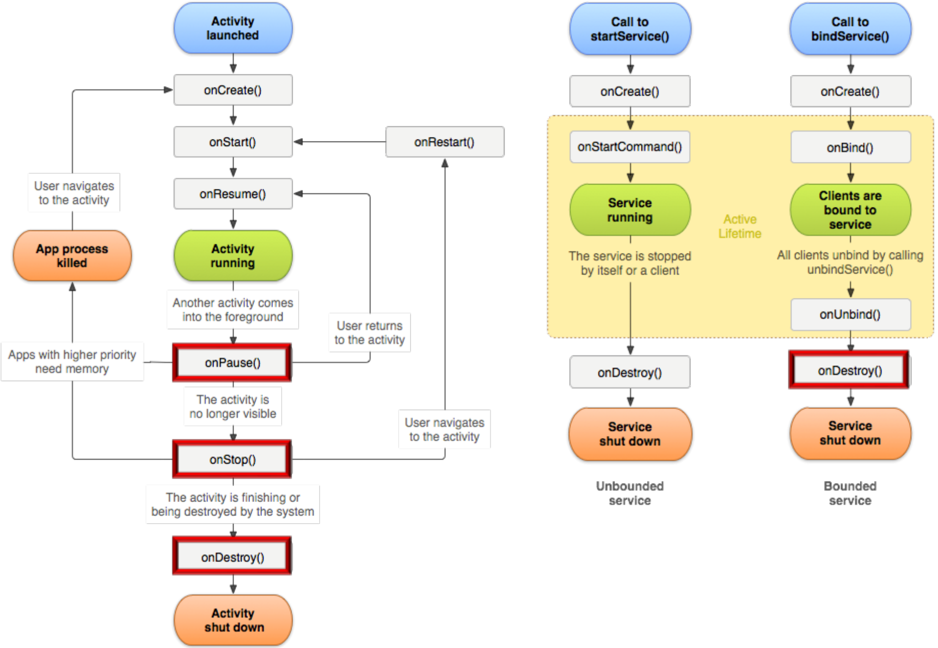




See www.vogella.com/tutorials/AndroidLifeCycle/article.html

Android Linux Extensions: Memory Management

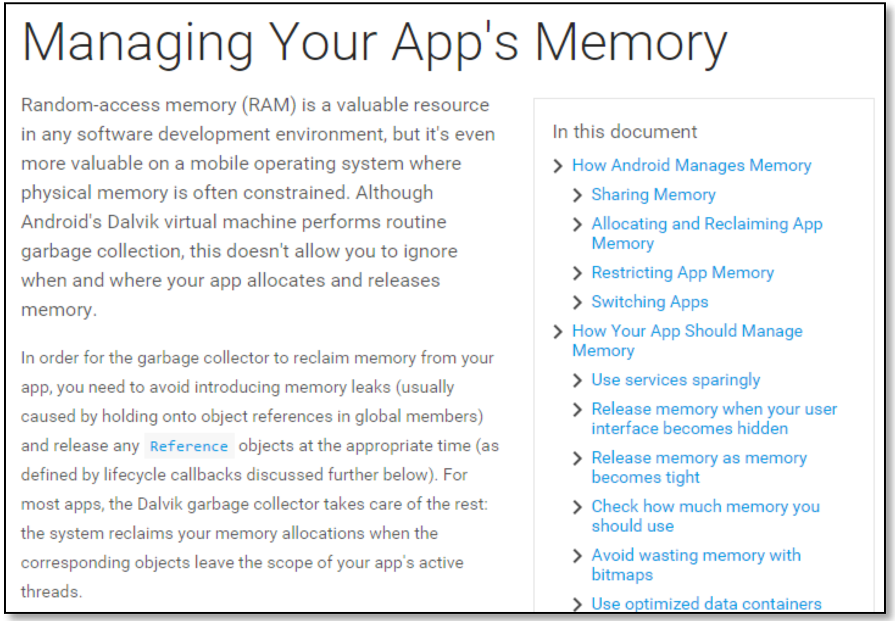
- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods
    - e.g., cleanup dynamically allocated resources in the onPause(), onStop(), & onDestroy() methods



See www.vogella.com/tutorials/AndroidLifeCycle/article.html

Android Linux Extensions: Memory Management

- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods
  - Apply common Android idioms for managing app RAM



Managing Your App's Memory

Random-access memory (RAM) is a valuable resource in any software development environment, but it's even more valuable on a mobile operating system where physical memory is often constrained. Although Android's Dalvik virtual machine performs routine garbage collection, this doesn't allow you to ignore when and where your app allocates and releases memory.

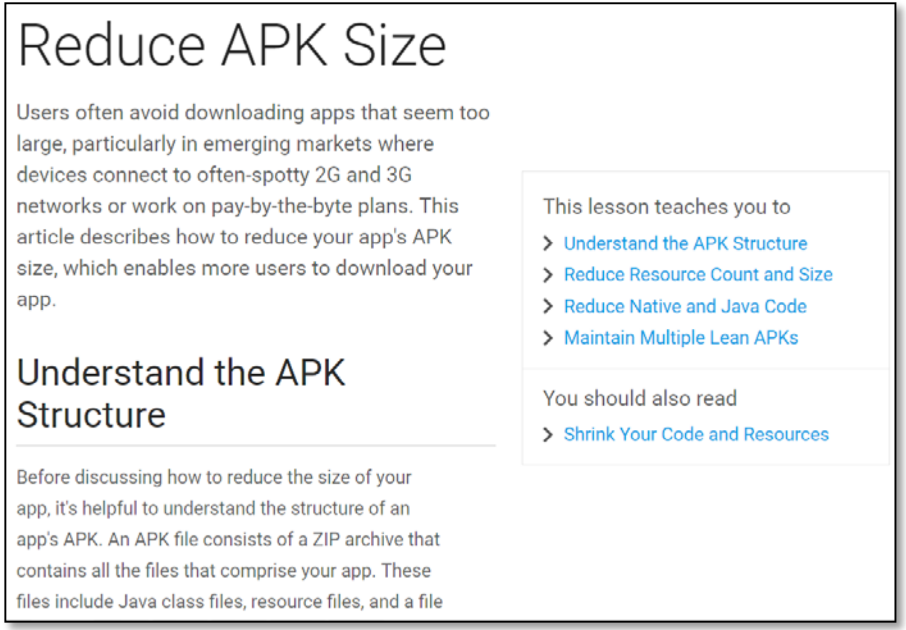
In order for the garbage collector to reclaim memory from your app, you need to avoid introducing memory leaks (usually caused by holding onto object references in global members) and release any `Reference` objects at the appropriate time (as defined by lifecycle callbacks discussed further below). For most apps, the Dalvik garbage collector takes care of the rest: the system reclaims your memory allocations when the corresponding objects leave the scope of your app's active threads.

In this document

See developer.android.com/training/articles/memory.html

Android Linux Extensions: Memory Management

- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods
  - Apply common Android idioms for managing app RAM
    - e.g., use long-running services sparingly & reduce size of Android package kits (APKs)



See developer.android.com/topic/performance/reduce-apk-size.html

Android Linux Extensions: Memory Management

- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods
  - Apply common Android idioms for managing app RAM
  - Apply common Android tools for managing app RAM

Tools for analyzing RAM usage

Before you can fix the memory usage problems in your app, you first need to find them. Android Studio and the Android SDK include several tools for analyzing memory usage in your app:

1. The Device Monitor has a Dalvik Debug Monitor Server (DDMS) tool that allows you to inspect memory allocation within your app process. You can use this information to understand how your app uses memory overall. For example, you can force a garbage collection event and then view the types of objects that remain in memory. You can use this information to identify operations or actions within your app that allocate or leave excessive amounts of objects in memory.

   For more information about how to use the DDMS tool, see Using DDMS.

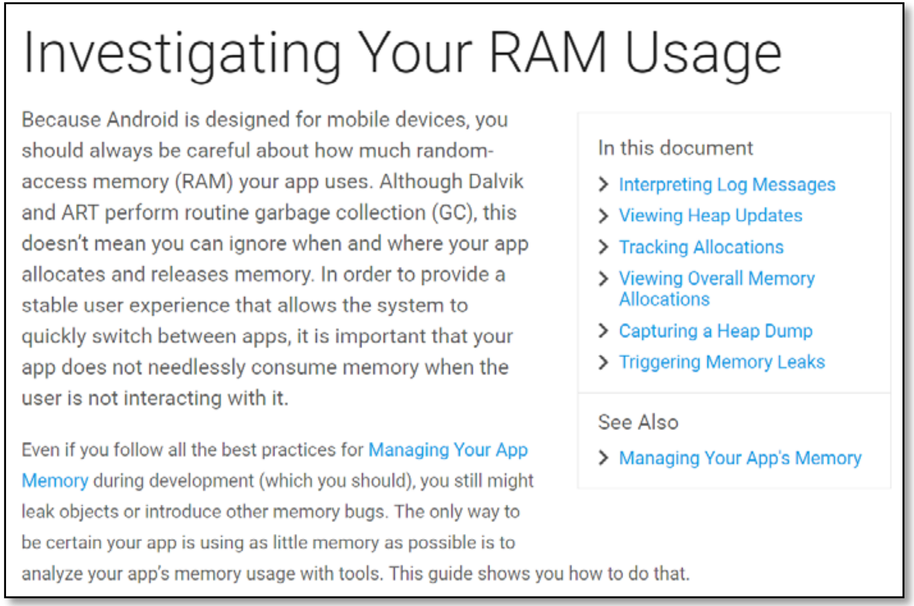
2. The Memory Monitor in Android Studio shows you how your app allocates memory over the course of a single session. The tool shows a graph of available and allocated Java memory over time, including garbage collection events. You can also initiate garbage collection events and take a snapshot of the Java heap while your app runs. The output from the Memory Monitor tool can help you identify points when your app experiences excessive garbage collection events, leading to app slowness.

   For more information about how to use Memory Monitor tool, see Viewing Heap Updates.

See developer.android.com/topic/performance/memory.html#AnalyzeRam

Android Linux Extensions: Memory Management

- App developers must know Android Linux kernel storage extensions, e.g.
  - Properly implement Activity & Service lifecycle methods
  - Apply common Android idioms for managing app RAM
  - Apply common Android tools for managing app RAM
    - e.g., Android Studio tools



Investigating Your RAM Usage

Because Android is designed for mobile devices, you should always be careful about how much random-access memory (RAM) your app uses. Although Dalvik and ART perform routine garbage collection (GC), this doesn't mean you can ignore when and where your app allocates and releases memory. In order to provide a stable user experience that allows the system to quickly switch between apps, it is important that your app does not needlessly consume memory when the user is not interacting with it.

Even if you follow all the best practices for Managing Your App Memory during development (which you should), you still might leak objects or introduce other memory bugs. The only way to be certain your app is using as little memory as possible is to analyze your app's memory usage with tools. This guide shows you how to do that.

In this document

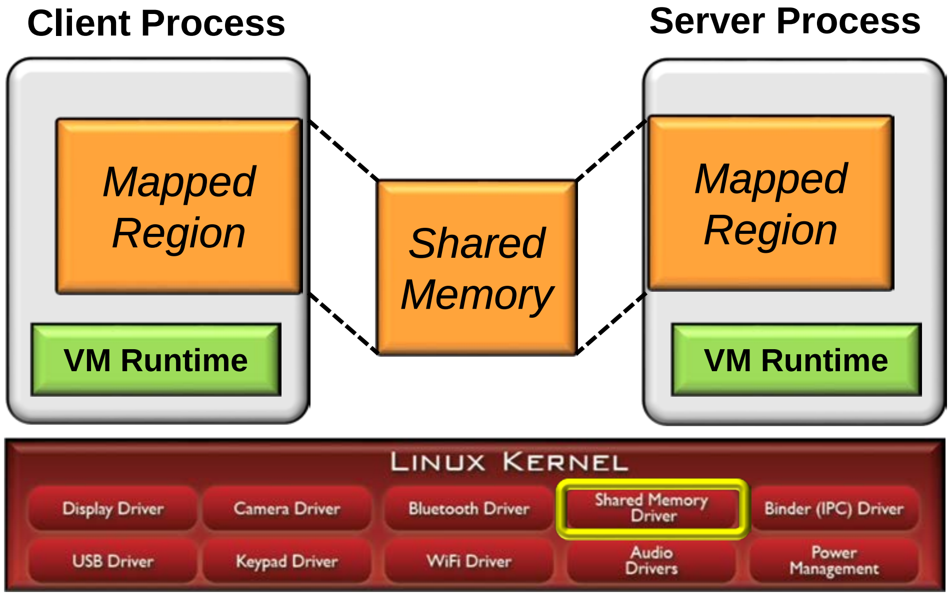
See Also

- Managing Your App's Memory

See developer.android.com/studio/profile/investigate-ram.html

Android Linux Extensions: Memory Management

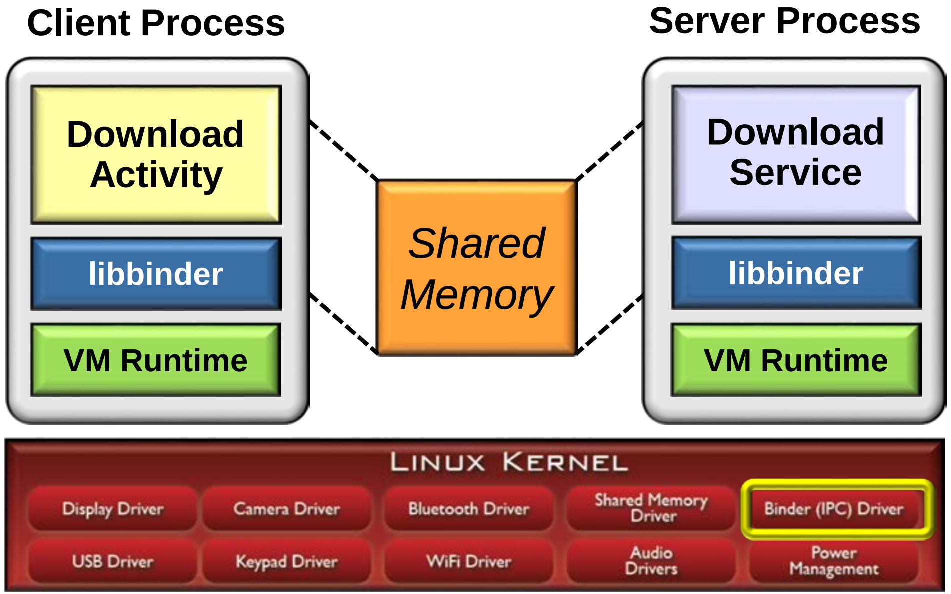
- Android's anonymous shared memory (ashmem) kernel extension allows multiple processes to share memory



See elinux.org/Android_Kernel_Features#ashmem

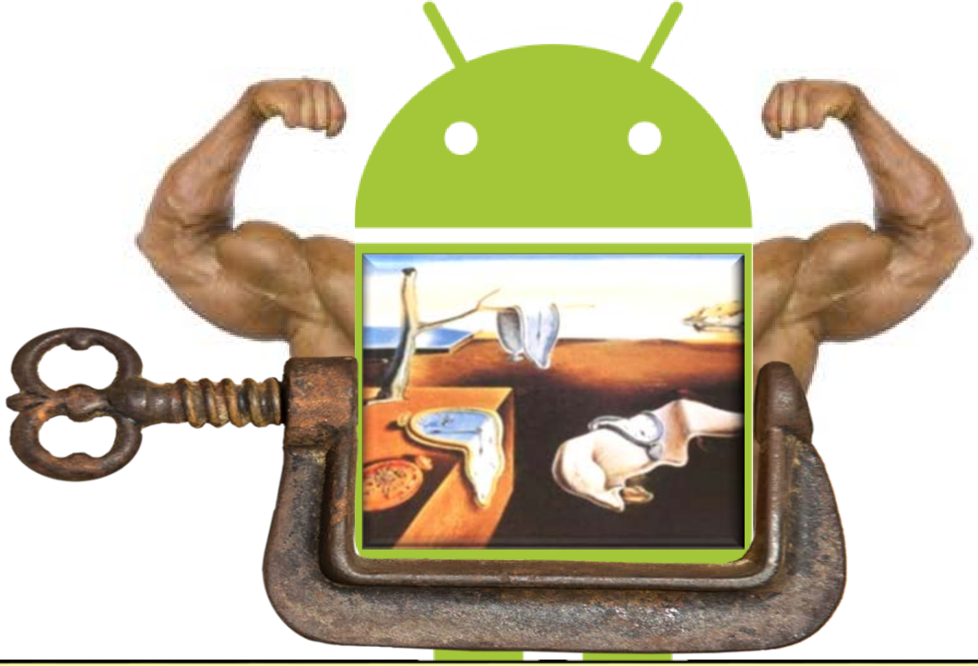
Android Linux Extensions: Memory Management

- Android's anonymous shared memory (ashmem) kernel extension allows multiple processes to share memory
  - e.g., the Binder framework uses ashmem to optimize transfer of "blobs" by avoiding data copying



Android Linux Extensions: Memory Management

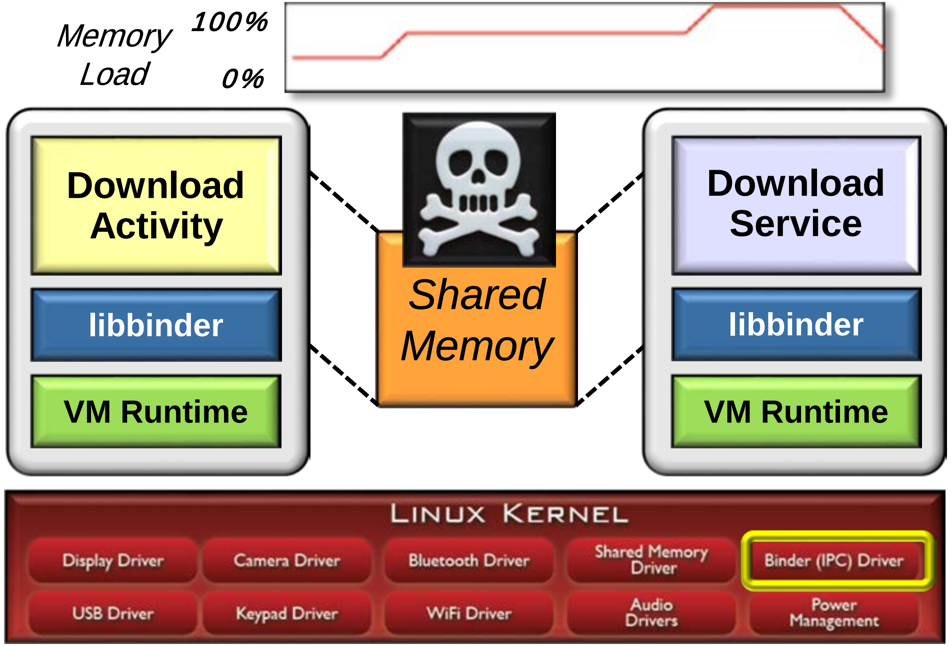
- ashmem is targeted for mobile devices with limited RAM



See notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory

Android Linux Extensions: Memory Management

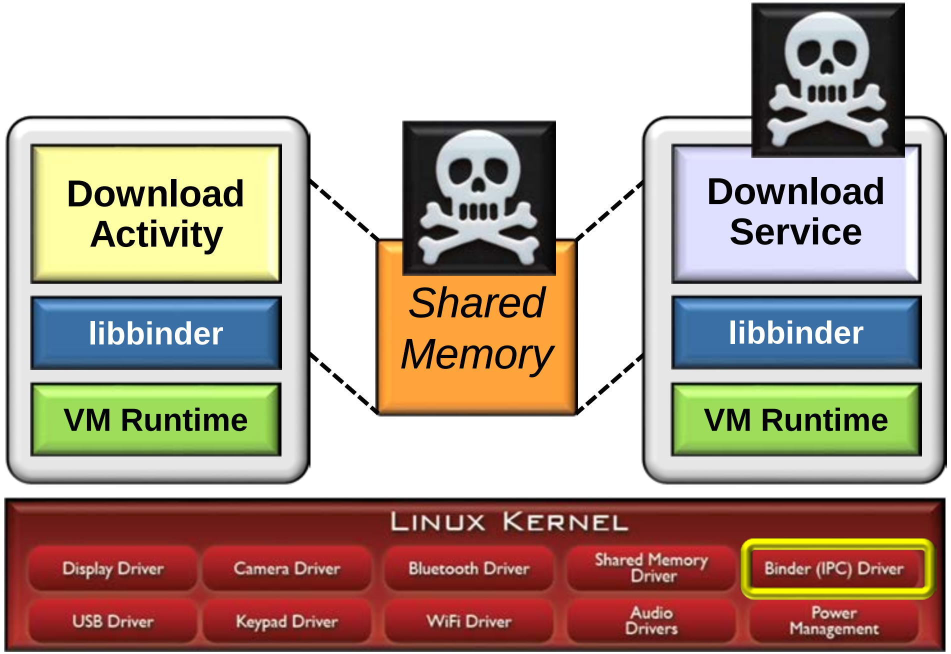
- ashmem is targeted for mobile devices with limited RAM, e.g.
  - Shared memory regions can be discarded when RAM runs low



See notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory

Android Linux Extensions: Memory Management

- ashmem is targeted for mobile devices with limited RAM, e.g.
  - Shared memory regions can be discarded when RAM runs low
  - Memory allocated by ashmem is released when the process that creates it exits



See notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory

Android Linux Extensions: Power Management

Android Linux Extensions: Power Management

- Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services

Android Linux Extensions: Power Management

- Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services
  - e.g., mobile devices require kernel-level power management capabilities

See developer.android.com/training/monitoring-device-state

Android Linux Extensions: Power Management

- Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services
  - e.g., mobile devices require kernel-level power management capabilities
    - Common problems include keeping the screen on too long or running the CPU for extended periods of time

Android Linux Extensions: Power Management

- Android Linux's PowerManagement kernel extensions control when a device sleeps & wakes

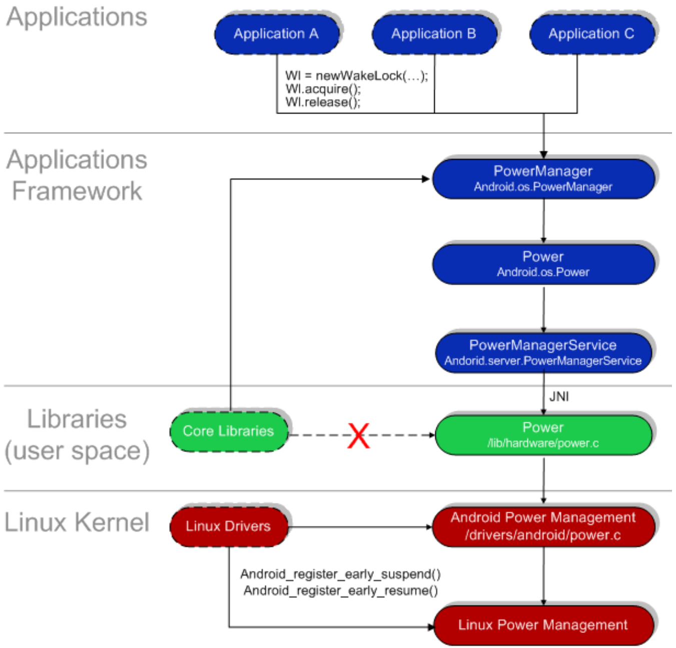
See elinux.org/Android_Power_Management

Android Linux Extensions: Power Management

- Android Linux's PowerManagement kernel extensions control when a device sleeps & wakes
  - Android middleware manages app power consumption via these kernel extensions

Android Linux Extensions: Power Management

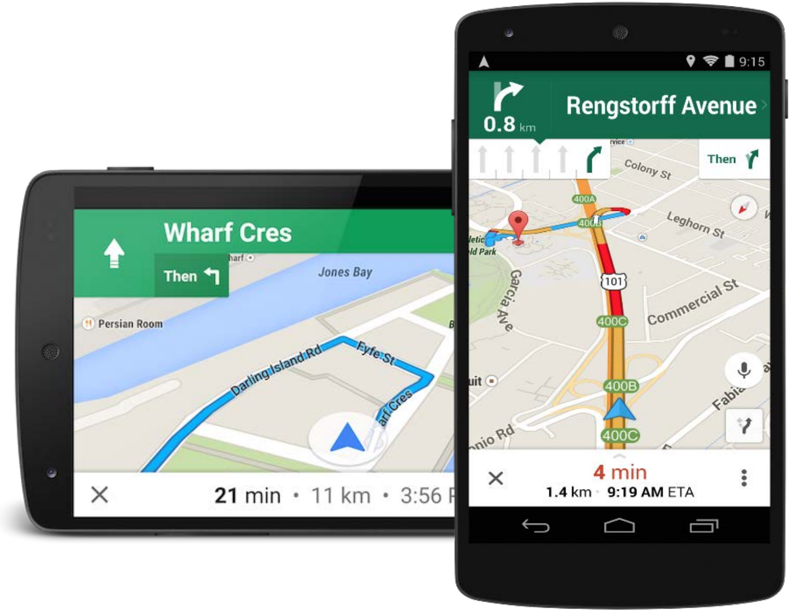
- Android Linux's PowerManagement kernel extensions control when a device sleeps & wakes
  - Android middleware manages app power consumption via these kernel extensions
    - e.g., the PowerManager system service provides wake locks to apps that can't sleep when they are idle



See developer.android.com/training/scheduling/wakelock.html

Android Linux Extensions: Power Management

- Android Linux's PowerManagement kernel extensions control when a device sleeps & wakes
  - Android middleware manages app power consumption via these kernel extensions
    - e.g., the PowerManager system service provides wake locks to apps that can't sleep when they are idle

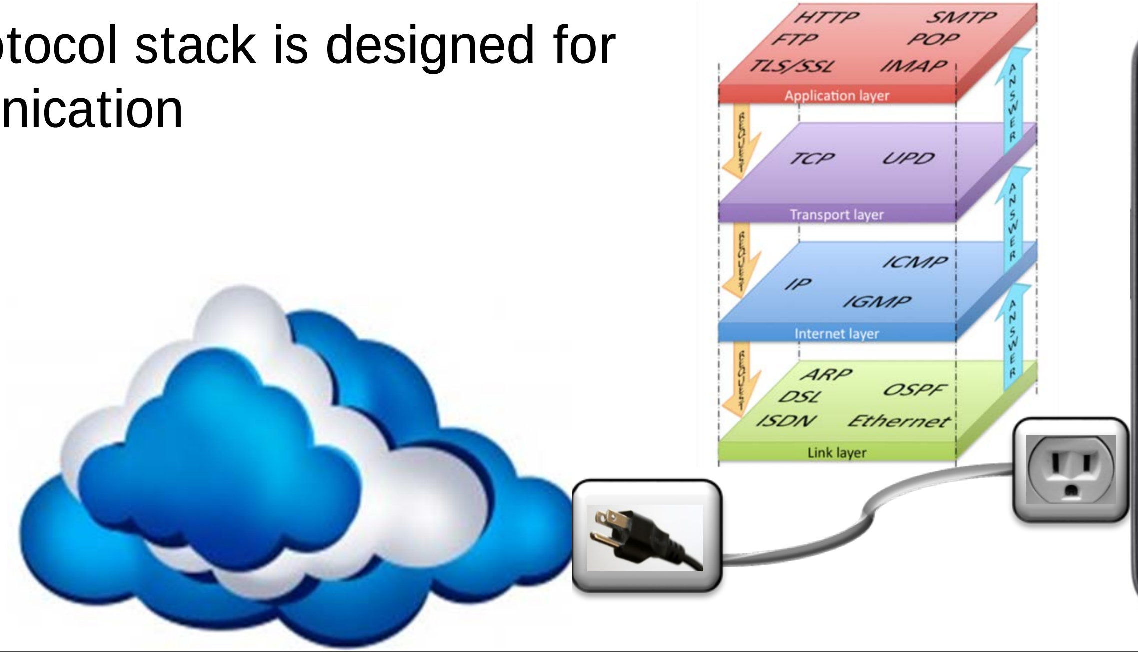


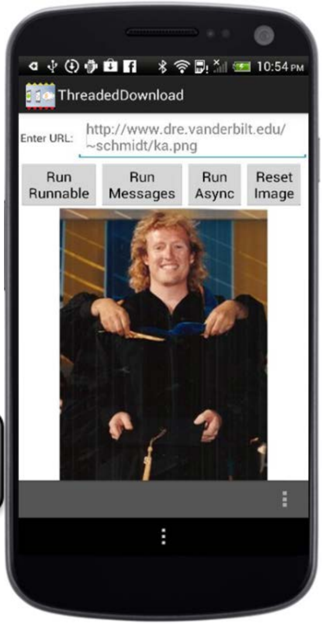
Examples include Google Maps navigation

Android Linux Extensions: Local Inter-Process Communication

Android Linux Extensions: Local IPC

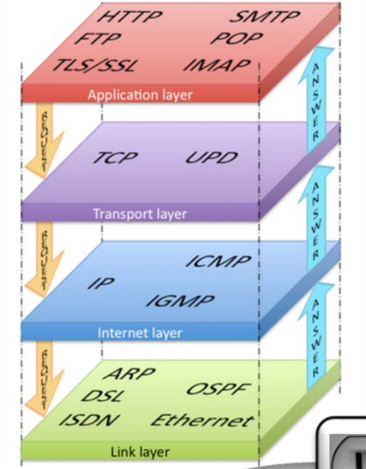
- The TCP/IP protocol stack is designed for remote communication

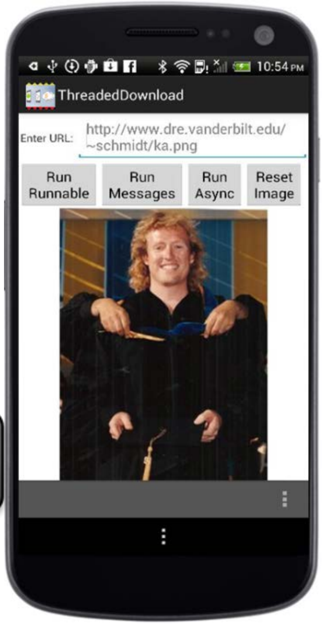




Android Linux Extensions: Local IPC

- The TCP/IP protocol stack is designed for remote communication
  - e.g., handles network congestion via "slow start" & exponential back-off algorithms

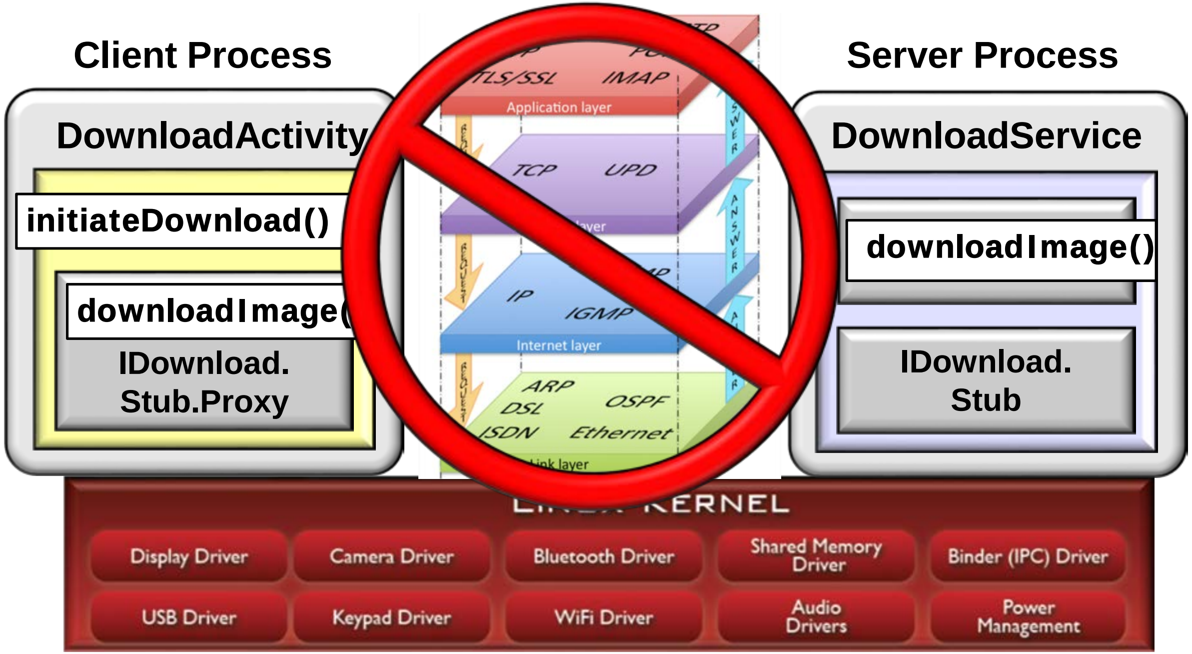




See en.wikipedia.org/wiki/TCP_congestion_control

Android Linux Extensions: Local IPC

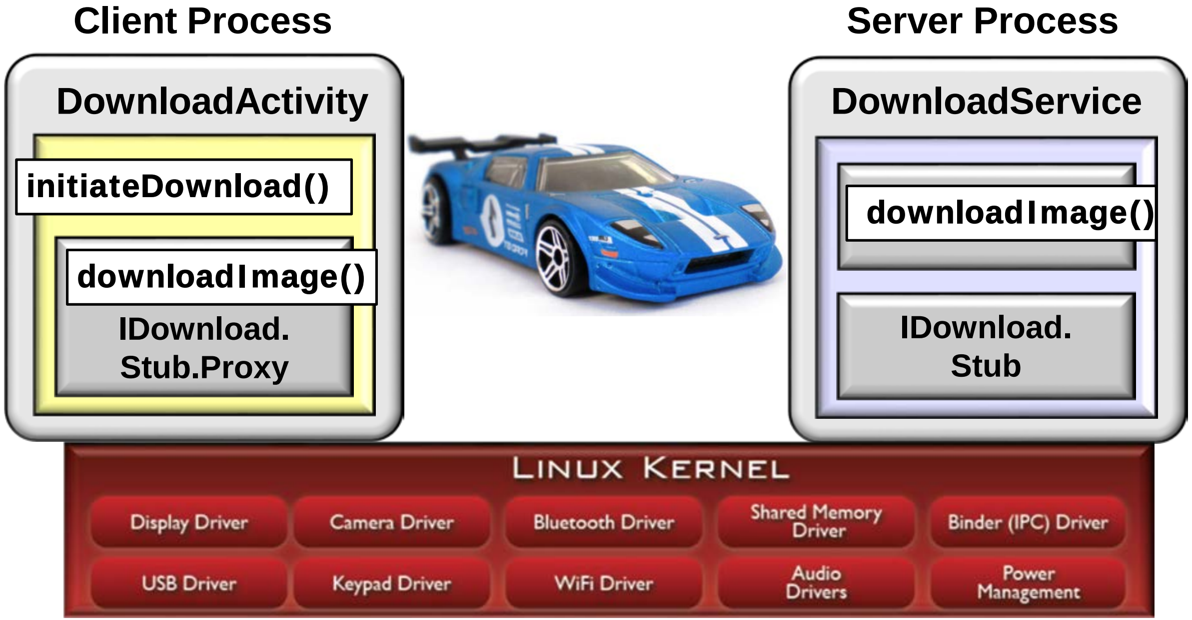
- TCP/IP incurs gratuitous overhead when used for IPC between processes on a mobile device
  - e.g., unnecessary checksums & code that handles network congestion



See bhavin.directi.com/unix-domain-sockets-vs-tcp-sockets

Android Linux Extensions: Local IPC

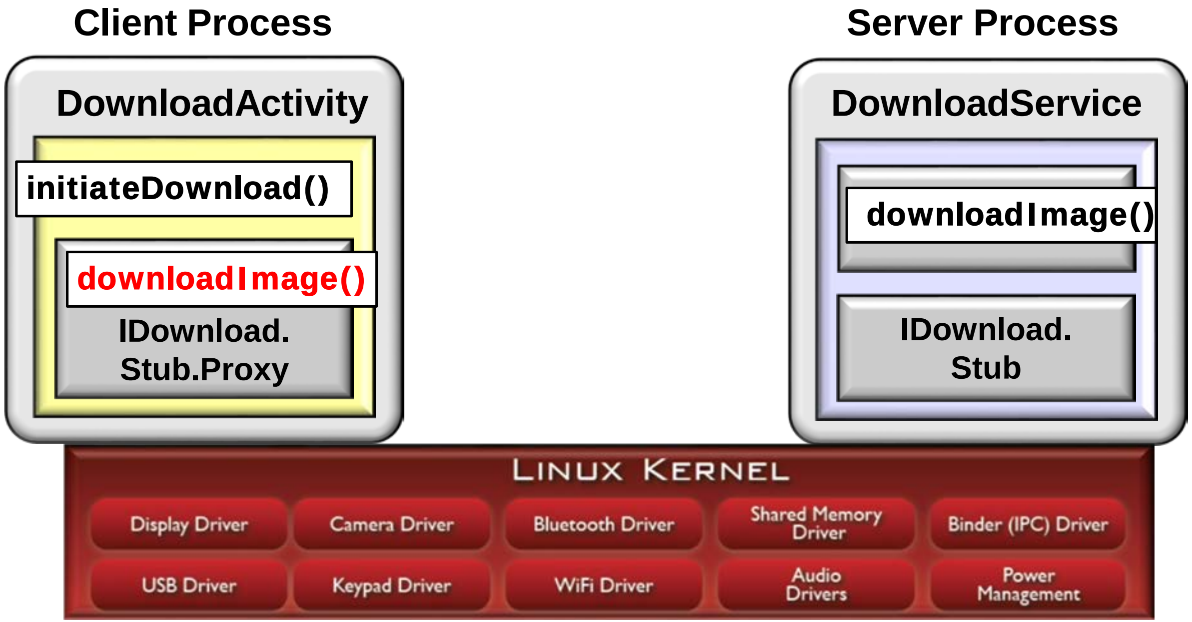
- Android's Binder driver is optimized for IPC on the same device



See elinux.org/Android_Binder

Android Linux Extensions: Local IPC

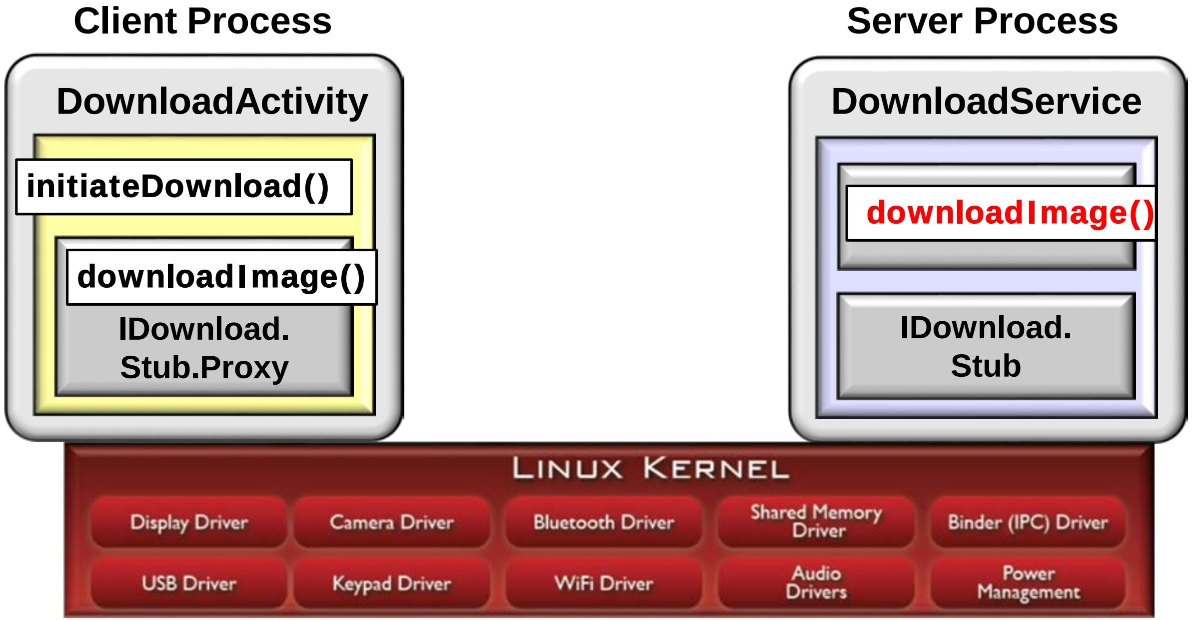
- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



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

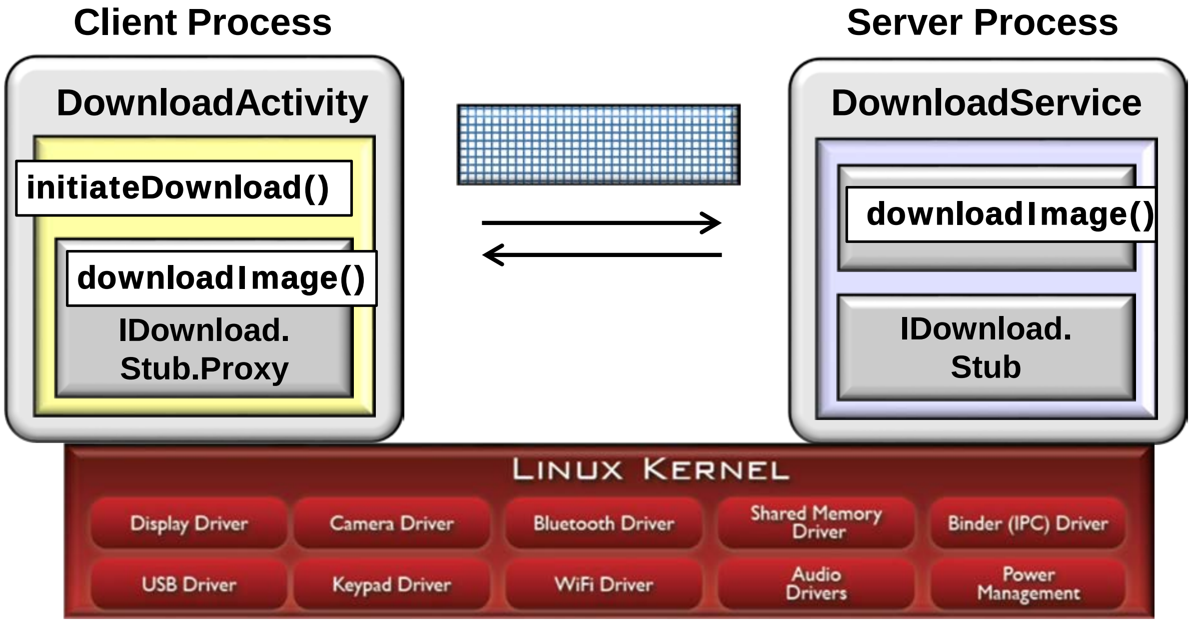
Android Linux Extensions: Local IPC

- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



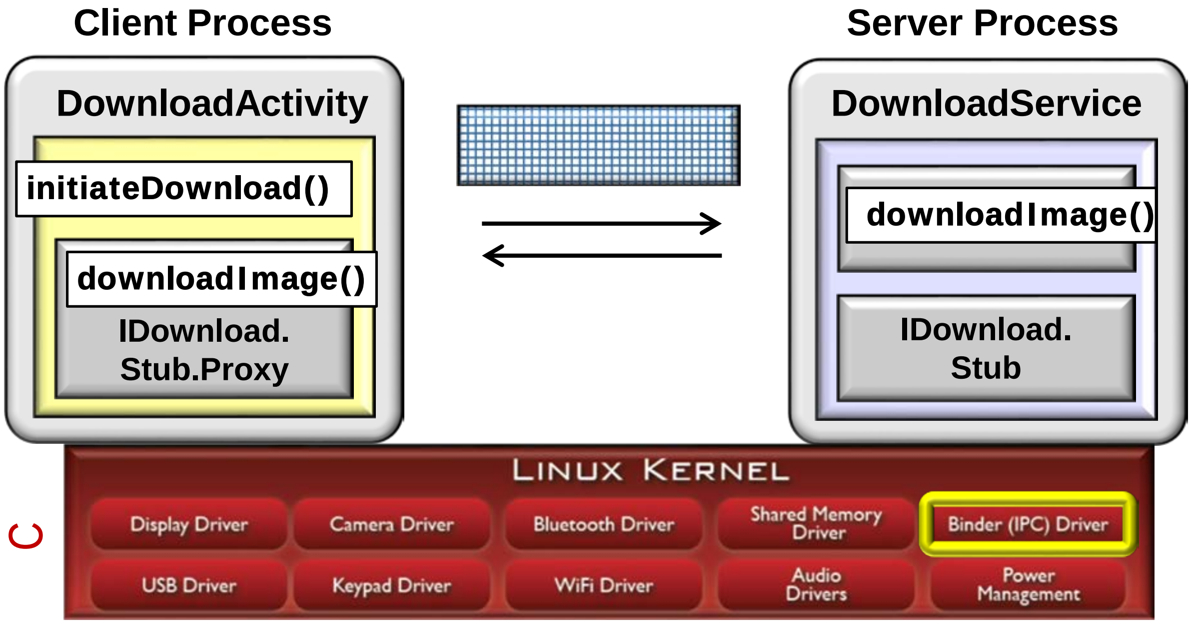
Android Linux Extensions: Local IPC

- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



Android Linux Extensions: Local IPC

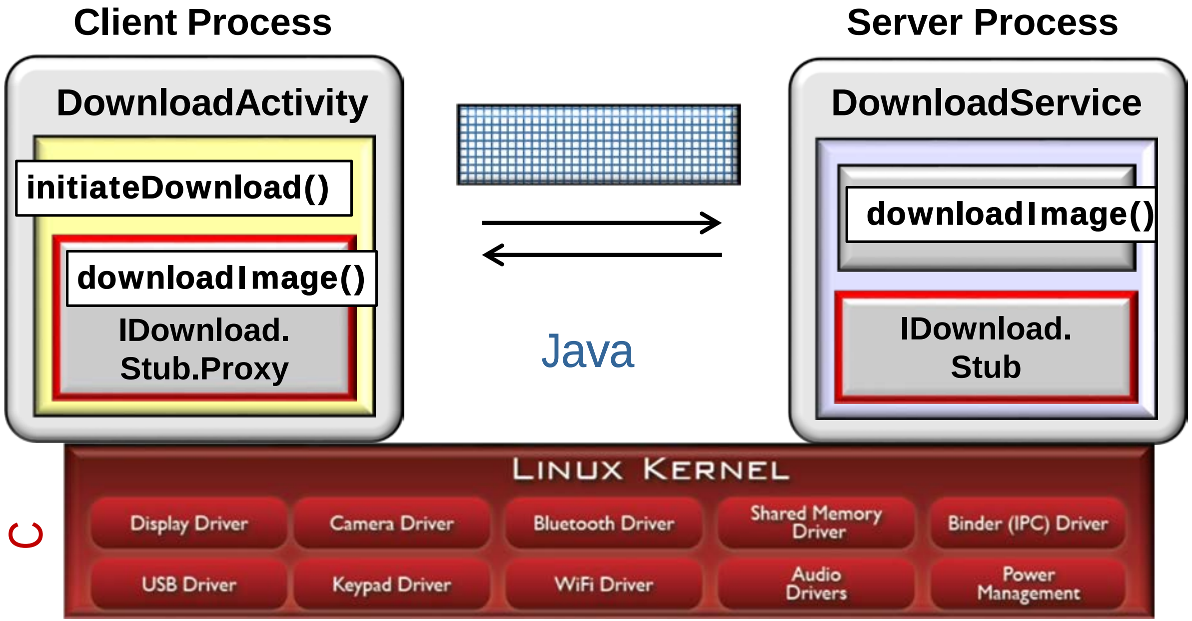
- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



Binder driver is written in C/C++ & runs in the kernel to enhance performance

Android Linux Extensions: Local IPC

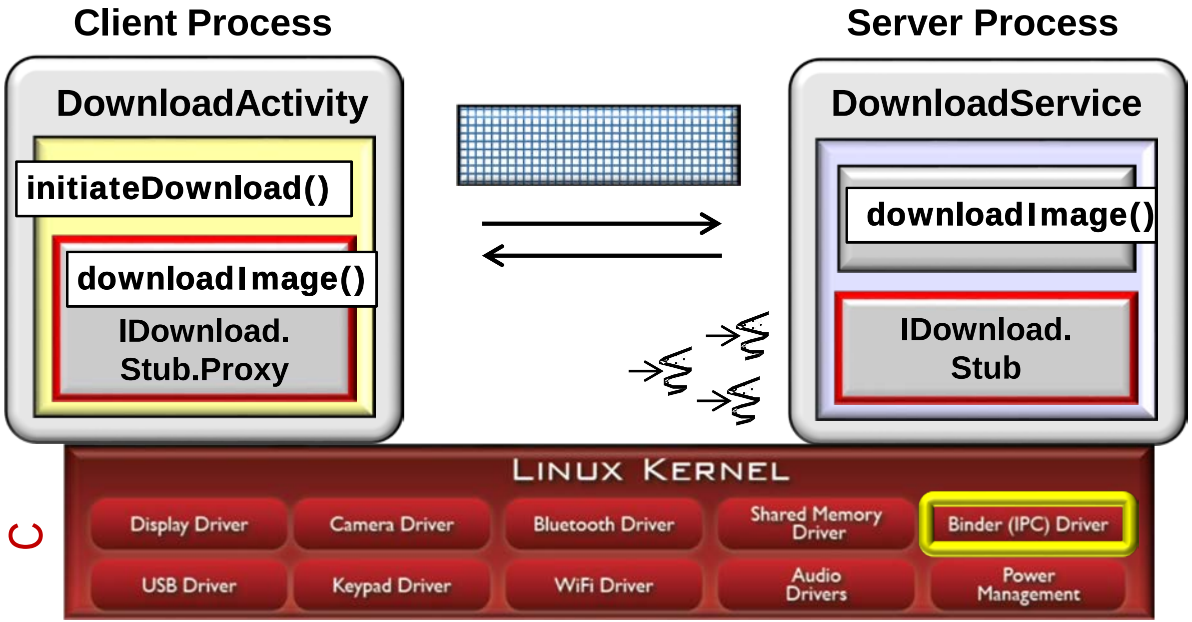
- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



Binder driver collaborates with higher-layer Binder framework written in Java

Android Linux Extensions: Local IPC

- Android's Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke "remote" calls



These layers of software simplify/optimizre object-oriented local IPC

End of the Android Linux Kernel (Part 3): Android Kernel Extensions