G54MDP
Mobile Device Programming

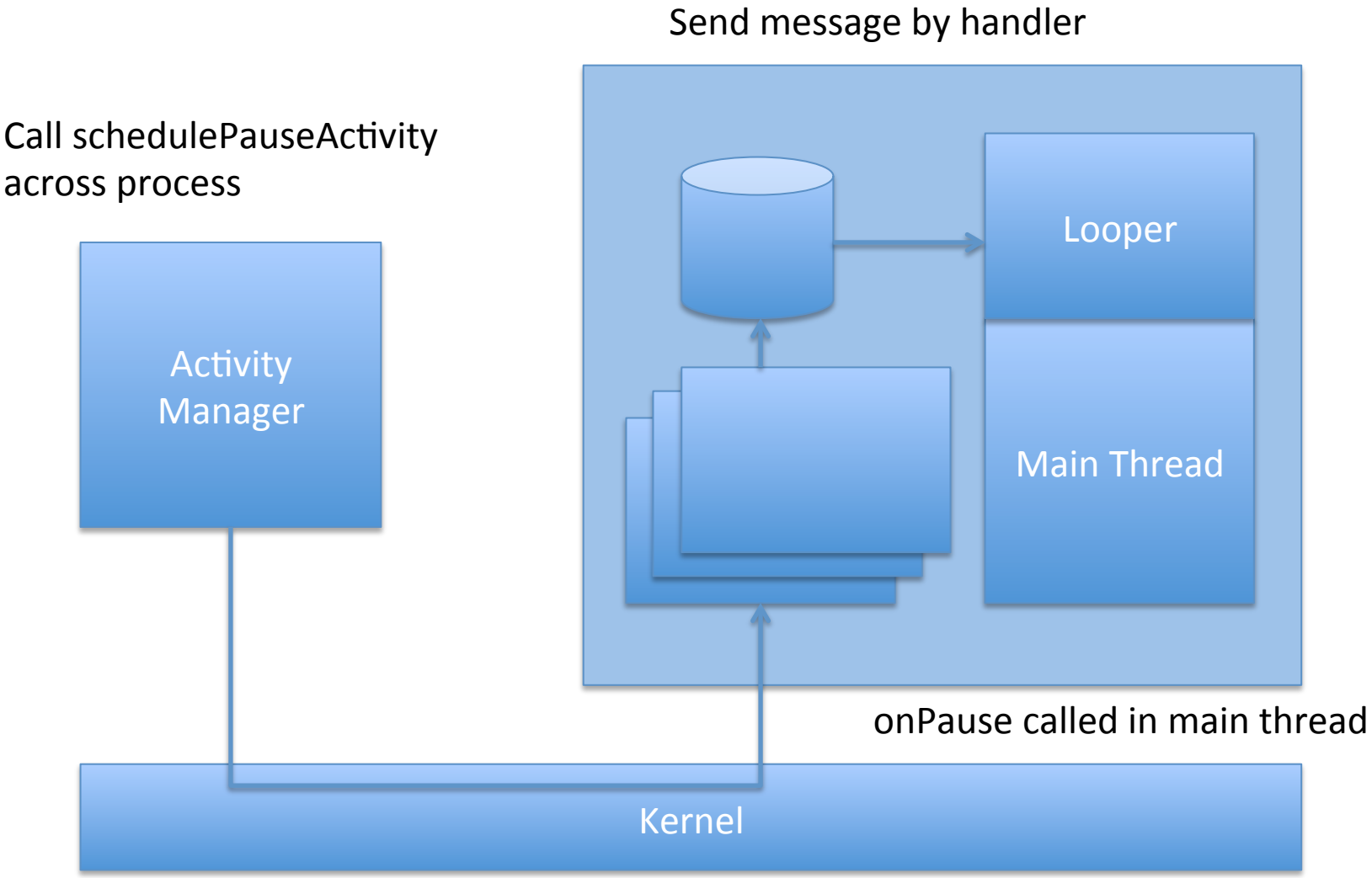
Lecture 9 – IPC Internals / Storage

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

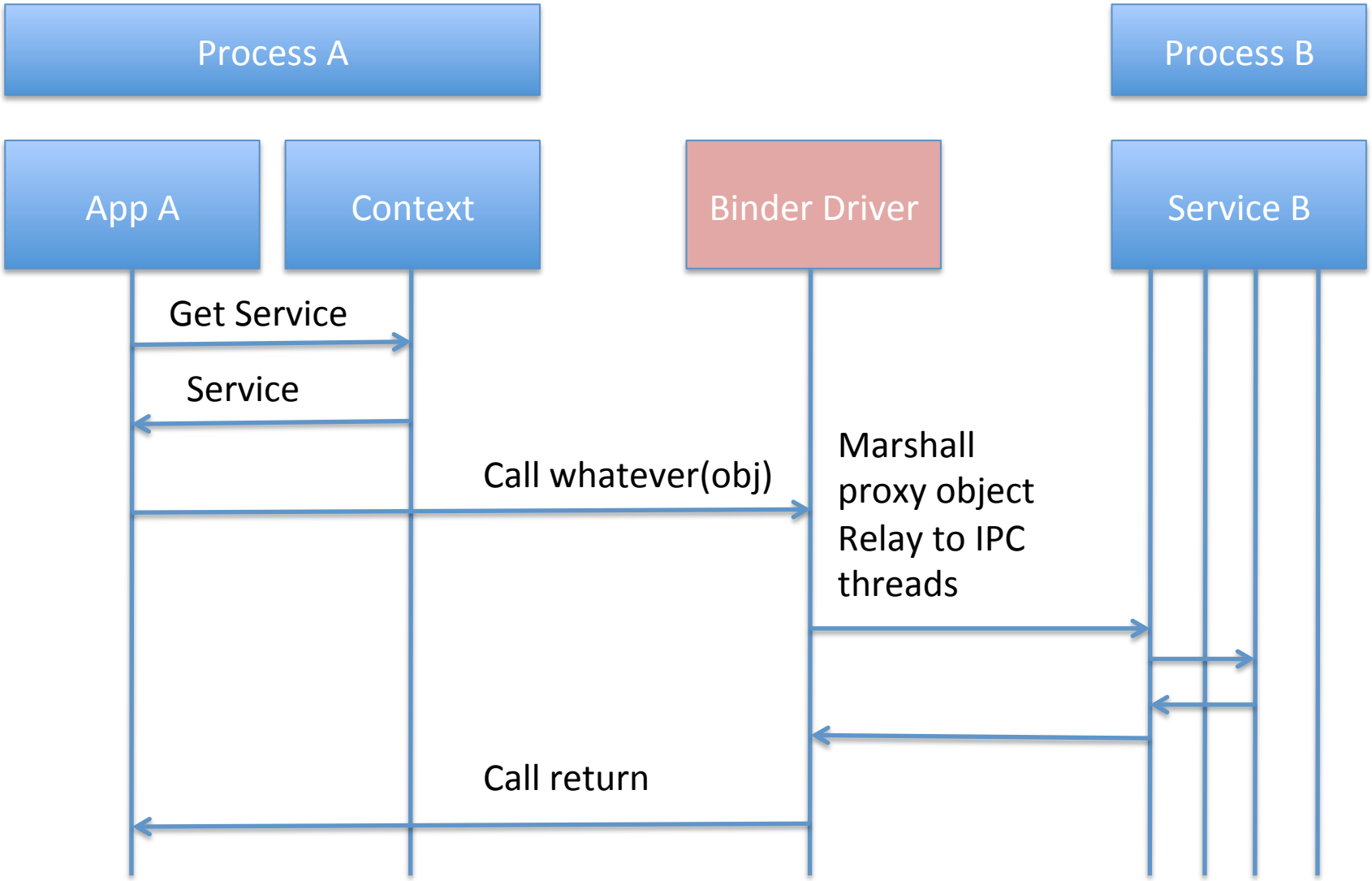
Coursework Questions

- Why is it harder than last year?
- What kind of Service?
  - Think about abstractions
- What kind of timer?
  - I don't care
  - Threads, how many?

onPause()


Send message by handler
Call schedulePauseActivity
across process
Activity
Manager
Looper
Main Thread
onPause called in main thread
Kernel

Binder in action



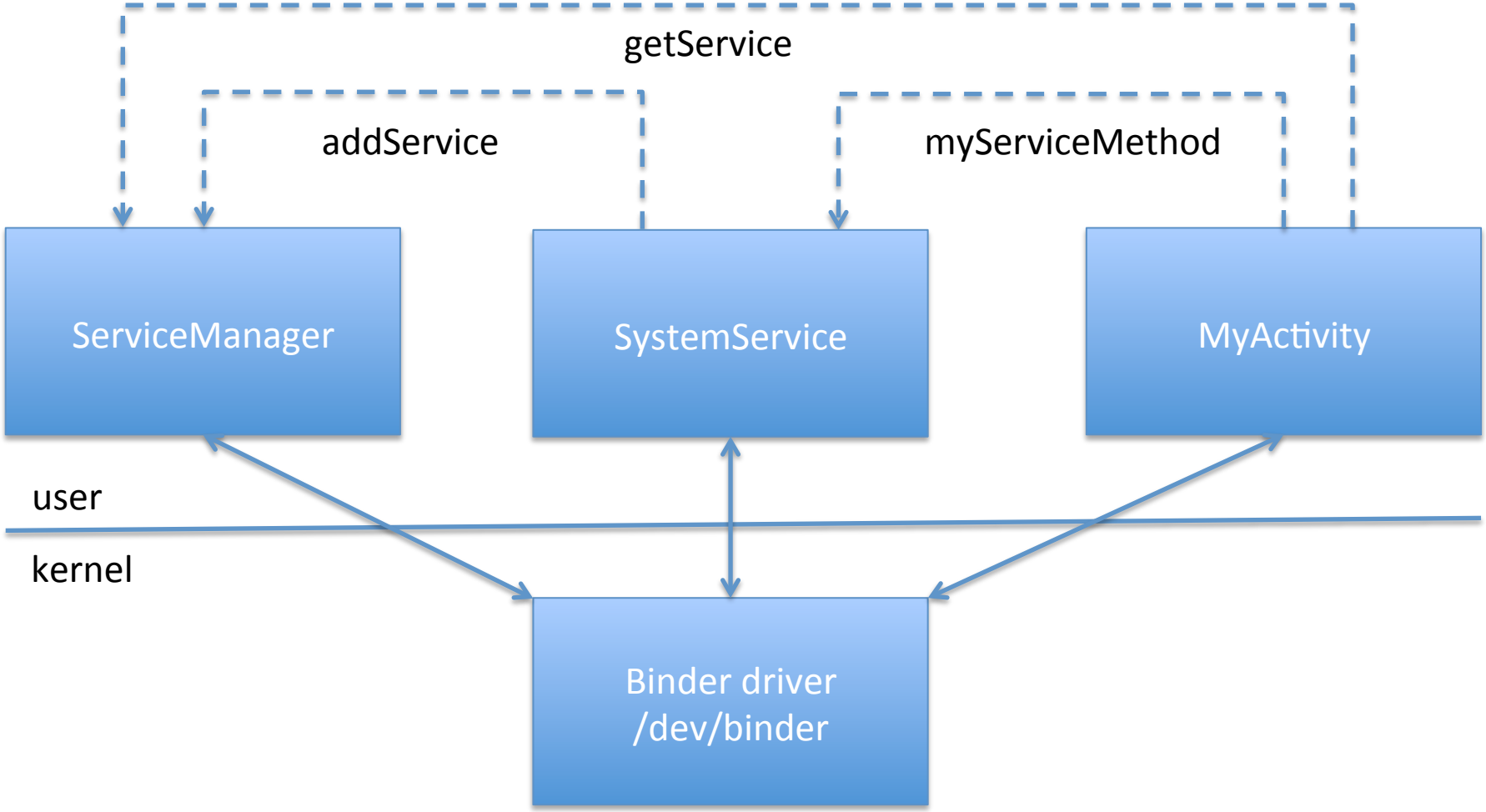
Binder Functionality

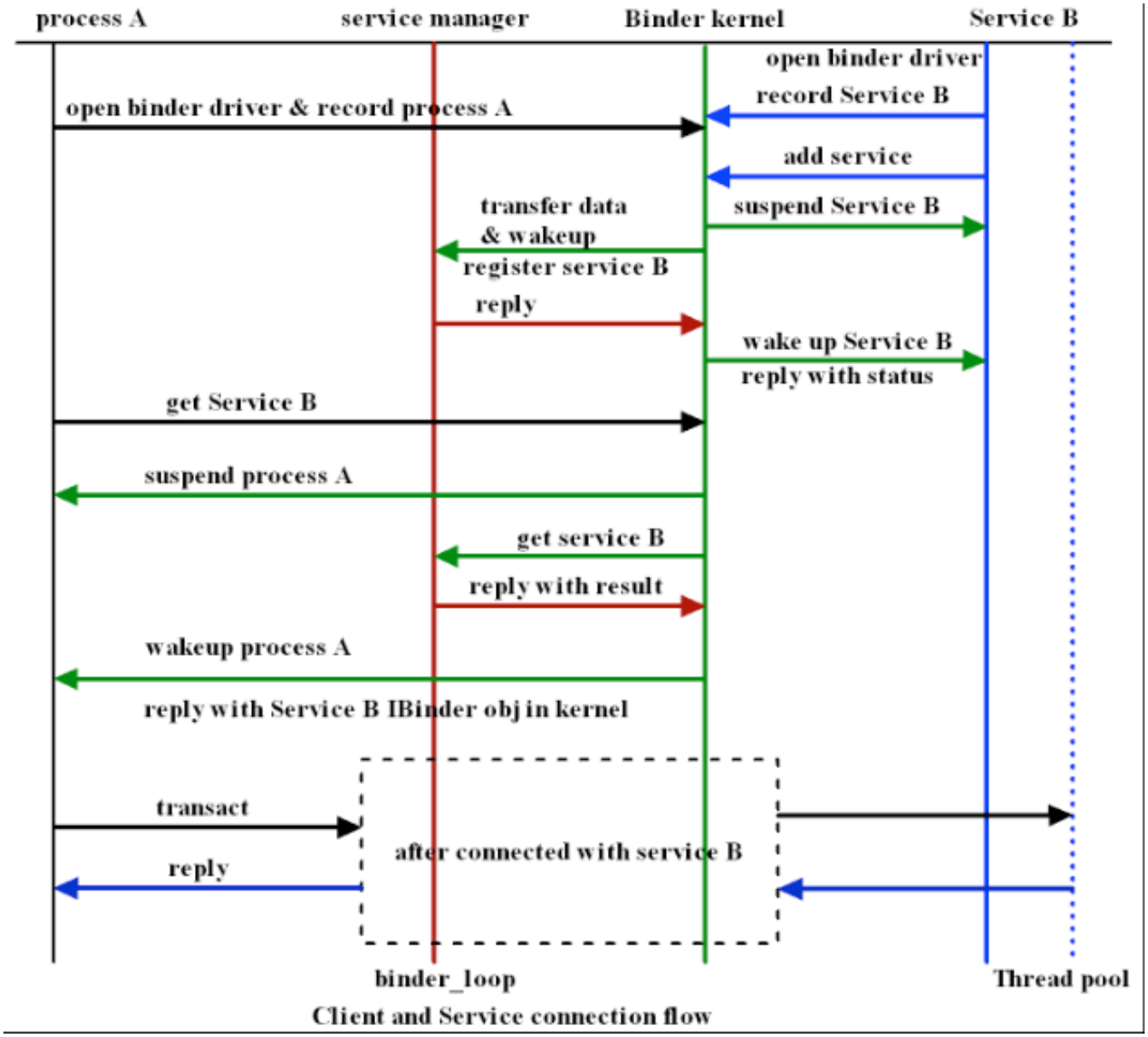
- Architecture
  - Binder kernel driver
  - Instance of Binder objects within user-space
    - Implements the IBinder interface
- Managing communication between processes
  - Simple inter-process messaging
    - Parcelable objects
  - Inter-process message calls
    - Call methods on remote objects as if they were local
  - Notifying processes of service events
- Identifying processes and services
  - Binder Token
    - Numerically uniquely identify a Binder instance
  - Basis of Android's **permissions** model
    - What are processes allowed to do?

ServiceManager

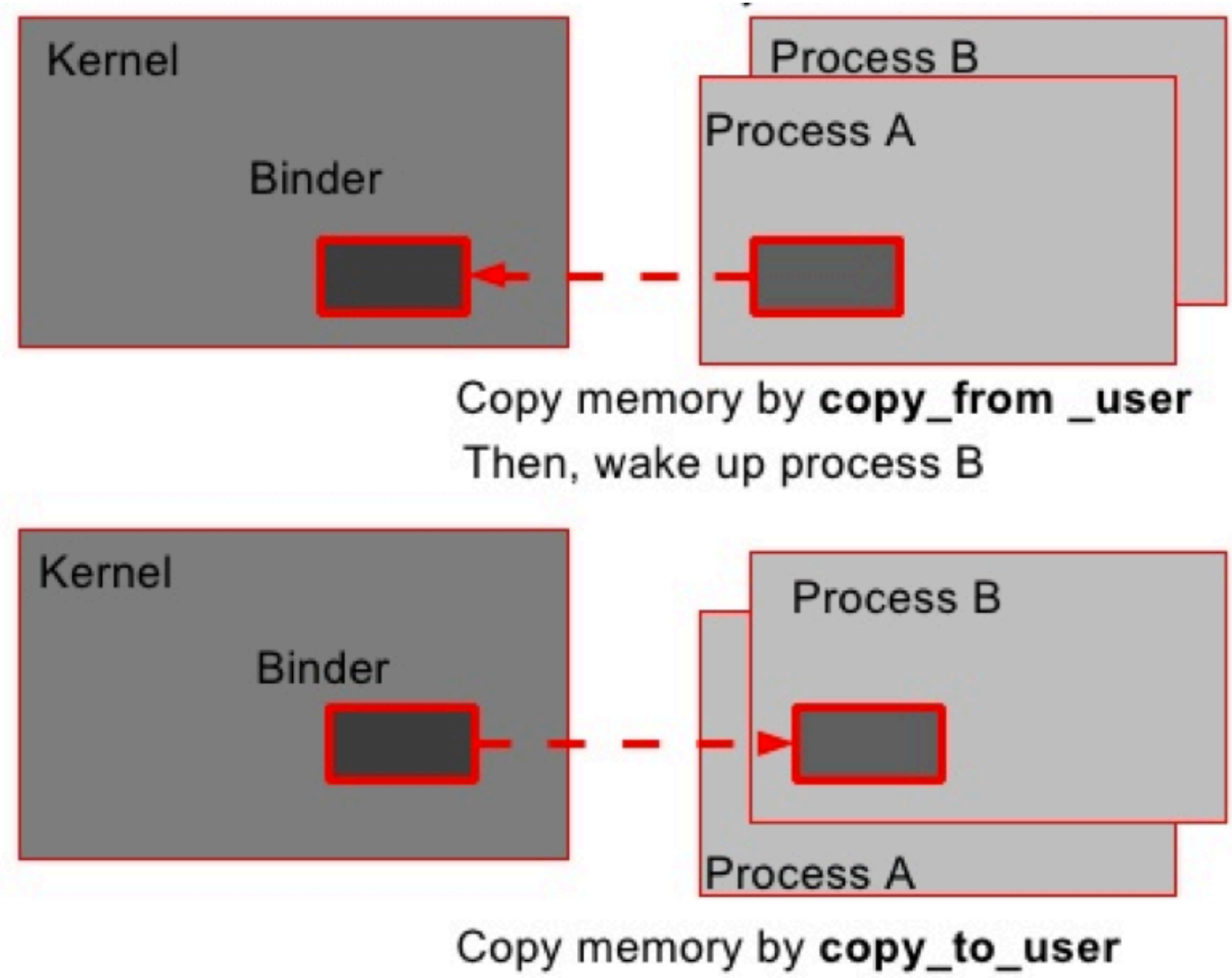
- A special Binder instance with a known Binder address
  - Hosts many system services within its process
  - Knows about other remote services
- Client does not know the token of remote Binder
  - Only the Binder interface knows its own address
- Binder submits a service name and its Binder token to the ServiceManager via IPC
  - Client retrieves remote service Binder address with service name
  - Client communicates with remote service

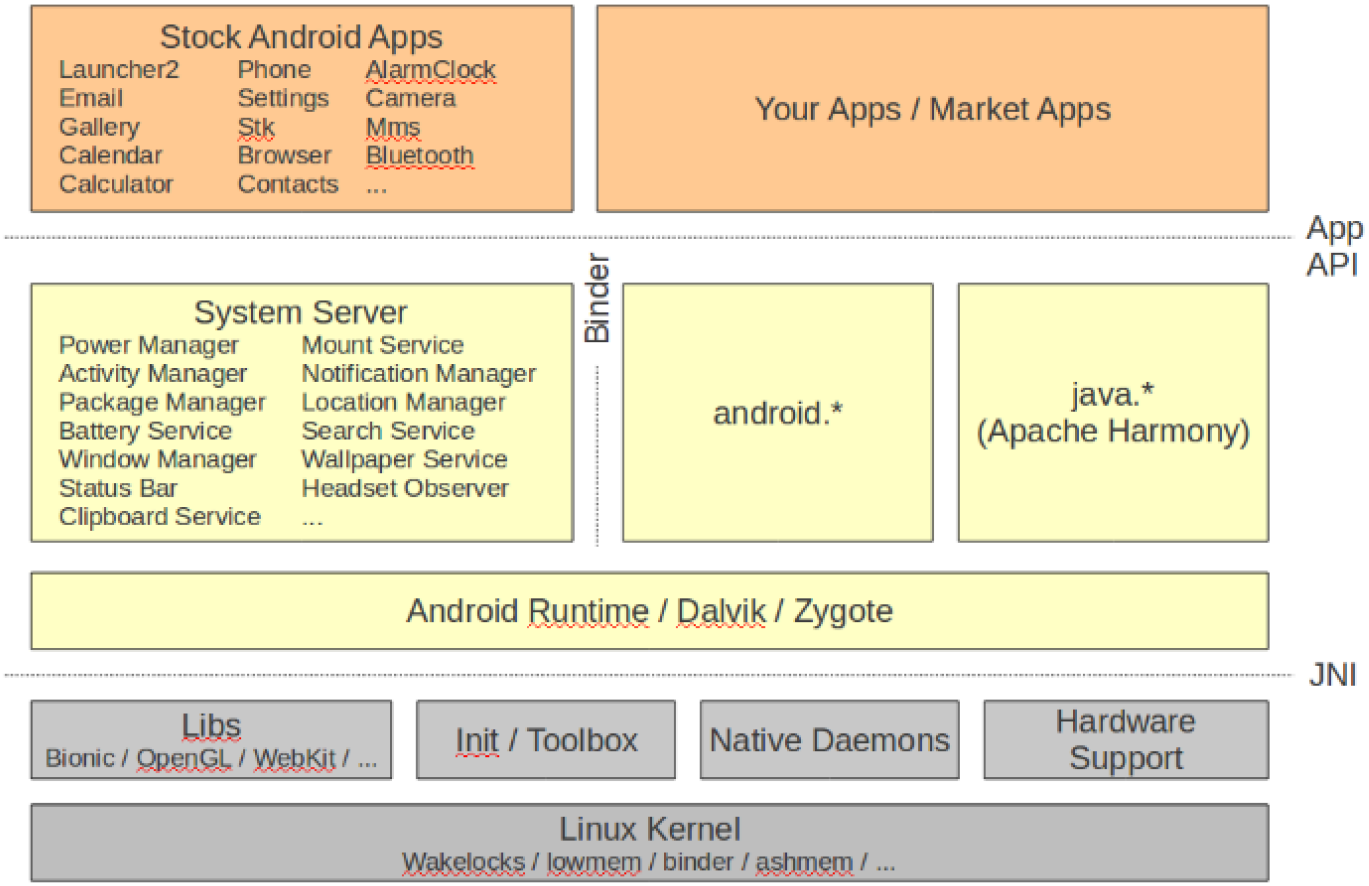
ServiceManager




process A
service manager
Binder kernel
Service B
open binder driver
record Service B
open binder driver & record process A
add service
transfer data
& wakeup
suspend Service B
register service B
reply
wake up Service B
reply with status
get Service B
suspend process A
get service B
reply with result
wakeup process A
reply with Service B IBinder obj in kernel
transact
after connected with service B
reply
binder_loop
Thread pool
Client and Service connection flow

Binder Transactions





SDK, Eclipse, .apk

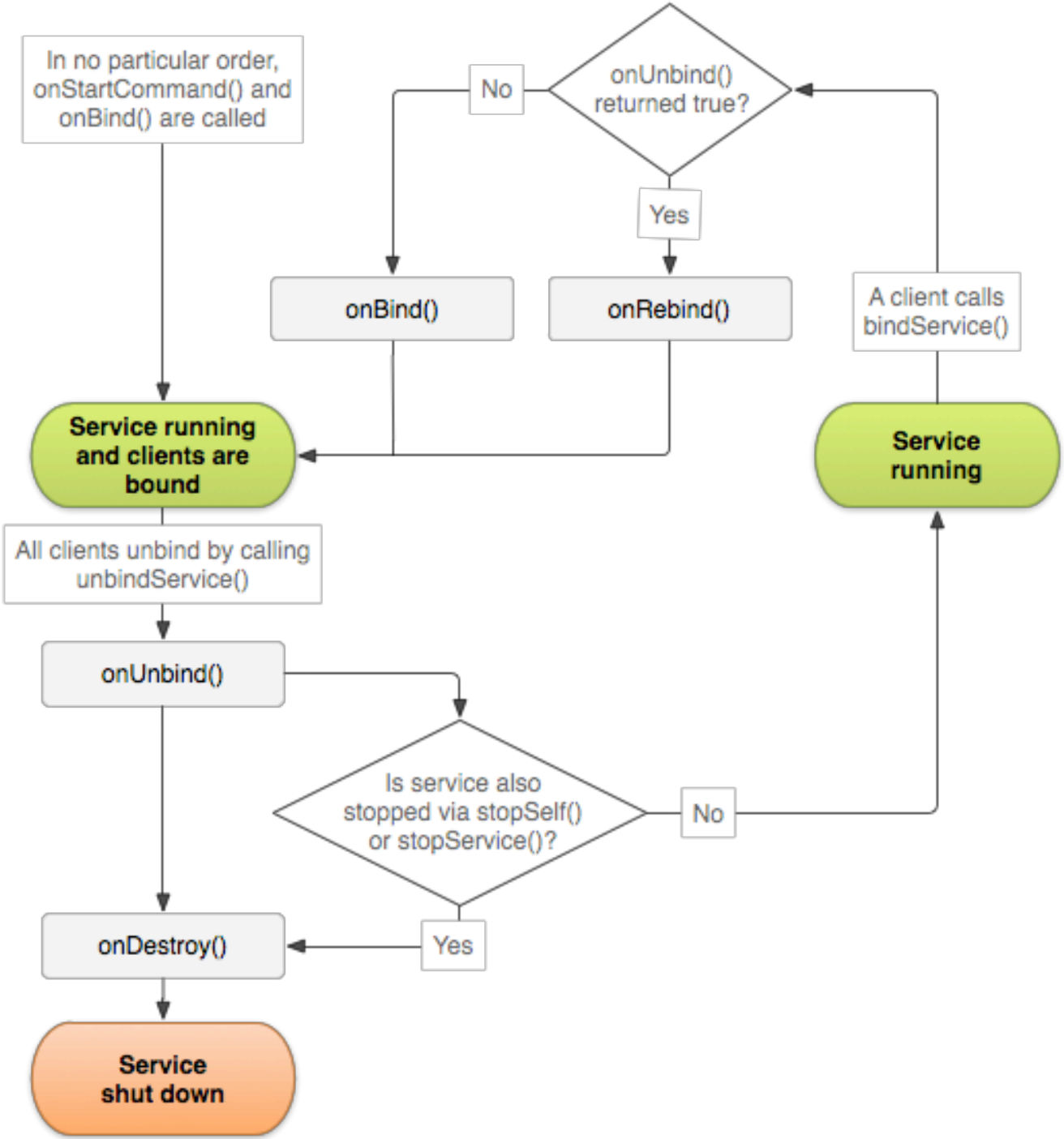
Manifest:
Perms / SDK ver.

.dex, ddms

NDK, rootfs, initrc, adb

GNU toolchain

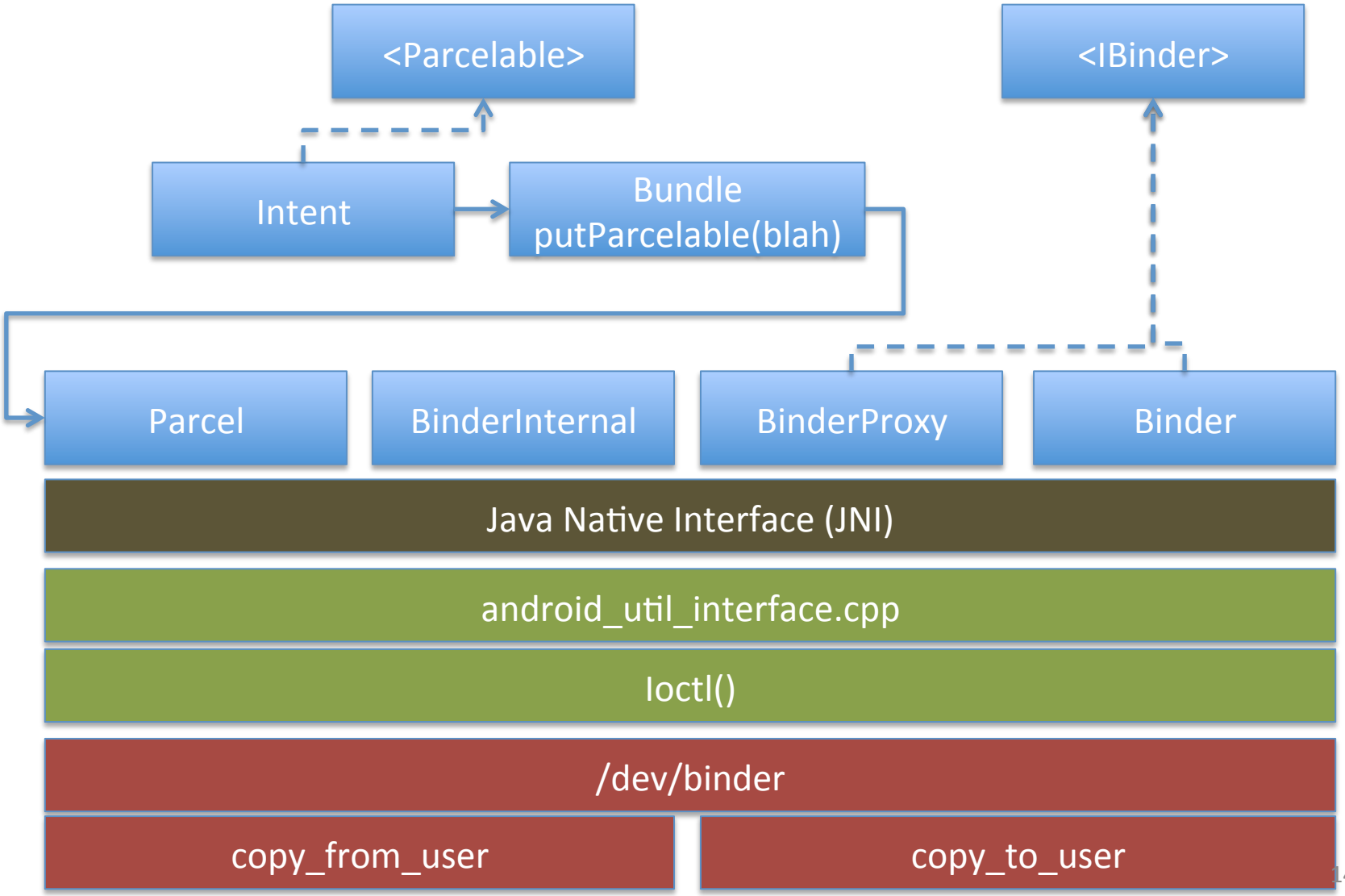
(fastboot)


In no particular order, onStartCommand() and onBind() are called
onUnbind() returned true?
No
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
Yes
onDestroy()
Service shut down

Binder Implementation

- API for apps
  - Written in Java
  - AIDL
  - Java API wrapper
    - Exposes the IBinder interface
    - Wraps the middleware layer
    - Parcelable object marshalling interface
- Middleware
  - Written in C++
  - Implements the user space (i.e. within a process) facilities of the Binder framework
  - Marshalling and unmarshalling of specific data to primitives
  - Provides interaction with the Binder kernel driver
- Kernel drivers
  - Written in C
  - Supports ioctl system calls from the middleware
  - Supports cross-process file operations, memory mapping
  - Thread pool for each service application for IPC
  - Mapping of objects between processes via copy_from_user, copy_to_user

Binder Implementation



Binder Performance / Limitations

- Binders communicate over process boundaries
  - Processes do not share a common virtual machine context
    - No direct access to objects
  - Not ideal of large data-streams
    - i.e. audio/video
    - Parcelable overhead
  - Good enough for window / activity / surface management
- Advantages
  - Native binary marshalling
    - Not java serialisation
  - Support of ashmem shared memory
- Disadvantages
  - Overhead of Dalvik Parcel marshalling
  - Ioctl() not optimal
  - Passes file descriptors for faster binary data transfer

Binder Security

- Binder Security Features
  - Client identity managed by the kernel
    - Binder.getCallingUid(), Binder.getCallingPid()
  - Interface reference security
    - Client cannot guess “address” of a service without going via the Service Manager
- Service Manager
  - A directory service for system services
    - Mediate access
  - Revoke access based on token
- Server could check client permissions at run-time
  - Context.checkPermission(permission, pid, uid)

Services recap

- A second kind of Android component
  - An abstraction of Binder / IPC
    - Used throughout the Android OS
- Tightly or loosely coupled to Activities
  - Start / destroy
    - Either by the Application
      - If we start it, it will run until we stop it
    - Or by the OS
      - If the OS starts it because it was bound, the OS destroys it when it is unbound
  - Communicate tightly via a Binder instance
    - Locally or remotely across processes
  - Communicate loosely via Notifications / Intents / Messages

References

- http://developer.android.com/guide/components/processes-and-threads.html
- http://developer.android.com/guide/components/services.html
- http://elinux.org/Android_Binder

So far…

- How to create a UI
  - Views defined in XML
  - Decomposing a task into Activities
- Services
  - Long running processes
- Intents
  - Used to send messages between things asynchronously

Data

- Apps generally store, process and display data
  - Maintain state
  - Store media data
  - Normally stored in files, databases
- Android doesn't expose its file system in the conventional manner
- Why not?

Logical Data Storage on Android

- File-based abstractions
  - Shared Preferences
    - Simple key value pairs
  - File-based storage
    - Internal Data Storage
      - Soldered RAM
      - Internal APK resources, temporary files
    - External Data Storage
      - SD Card
      - Large media files
  - SQLite Database
    - Structured data, small binary files
- Network
  - Shared contact lists, backups
  - SyncAdapter

Shared Preferences

- Persistently store primitive key/value pairs for an Application
  - Note, not an Activity
  - Removed when the application is removed

```
SharedPreferences settings = getSharedPreferences(“MY_PREFS”, 0);

boolean silent = settings.getBoolean("silentMode", false);
```

- Set / edit using an Editor object

```
SharedPreferences settings = getSharedPreferences(“MY_PREFS”, 0);

SharedPreferences.Editor editor = settings.edit();

editor.putBoolean("silentMode", mSilentMode);

editor.commit();
```

File Access

- Android uses the standard Java I/O classes to access files
  - File, InputStream, OutputStream, etc…
- However, these are obtained in a different way
  - Depending on the use

Let’s have a look…

References

- http://developer.android.com/guide/topics/data/data-storage.html
- http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html
- http://developer.android.com/reference/android/database/Cursor.html