Android IPC Mechanism

Jim Huang (黃敬群)

Developer, 0xlab

jserv@0xlab.org

March 19, 2012 / 南台科大

Rights to copy

Agenda

(0) IPC: The heart of Android

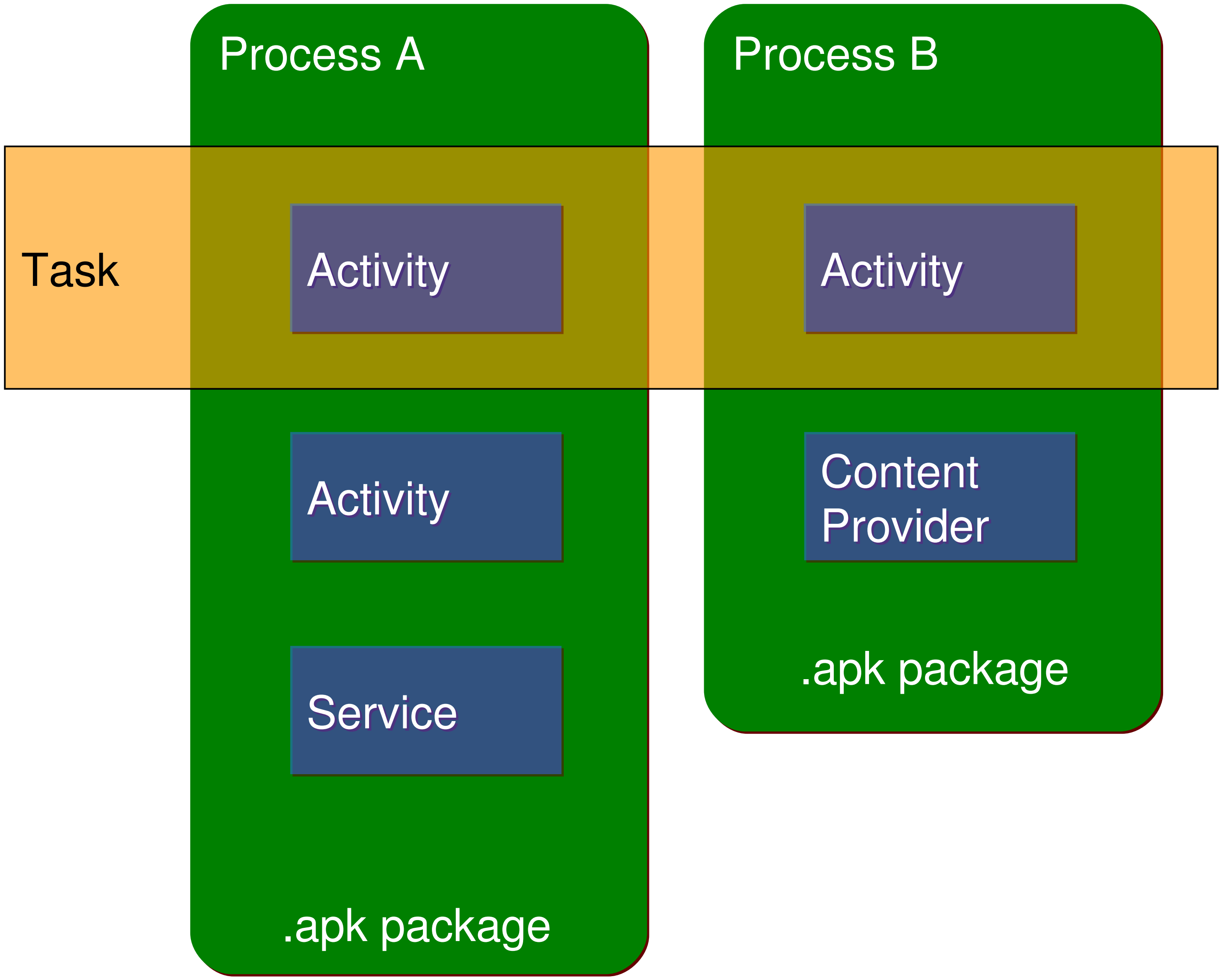
(1) Design Patterns

(2) Binder IPC Internals

(3) Use case: Graphics

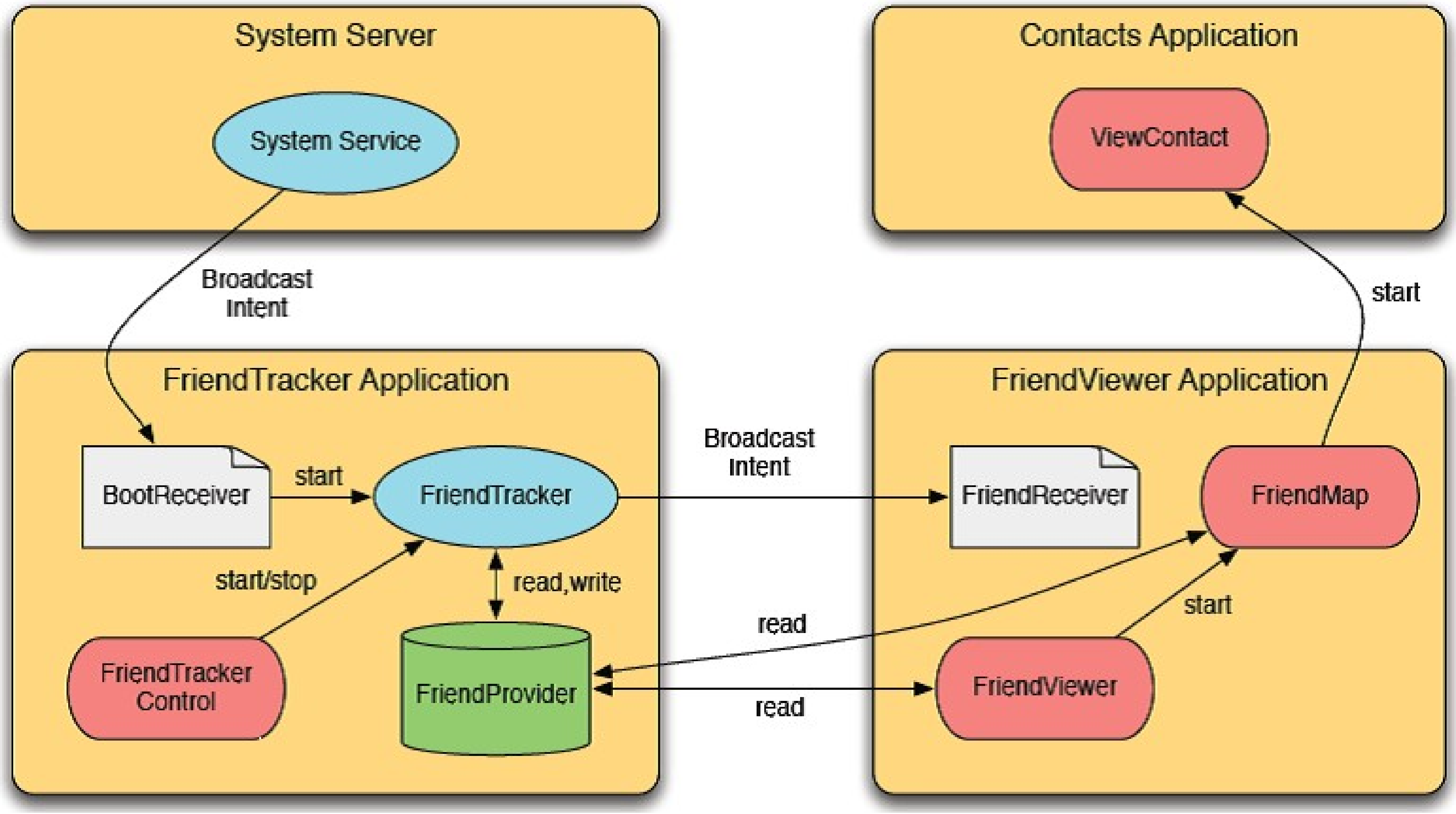
Binder IPC: The heart of Android

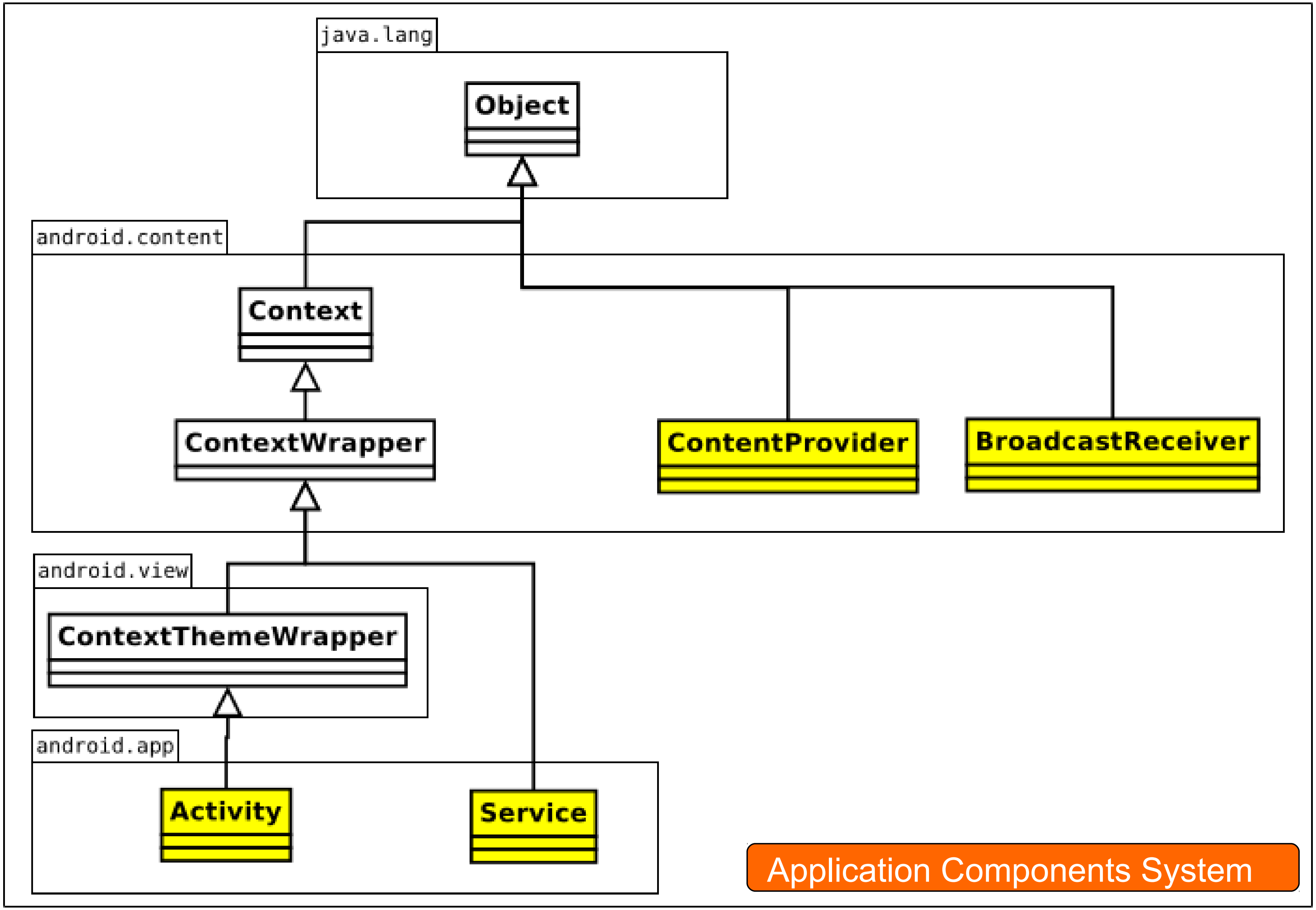
Android Tasks



Component View

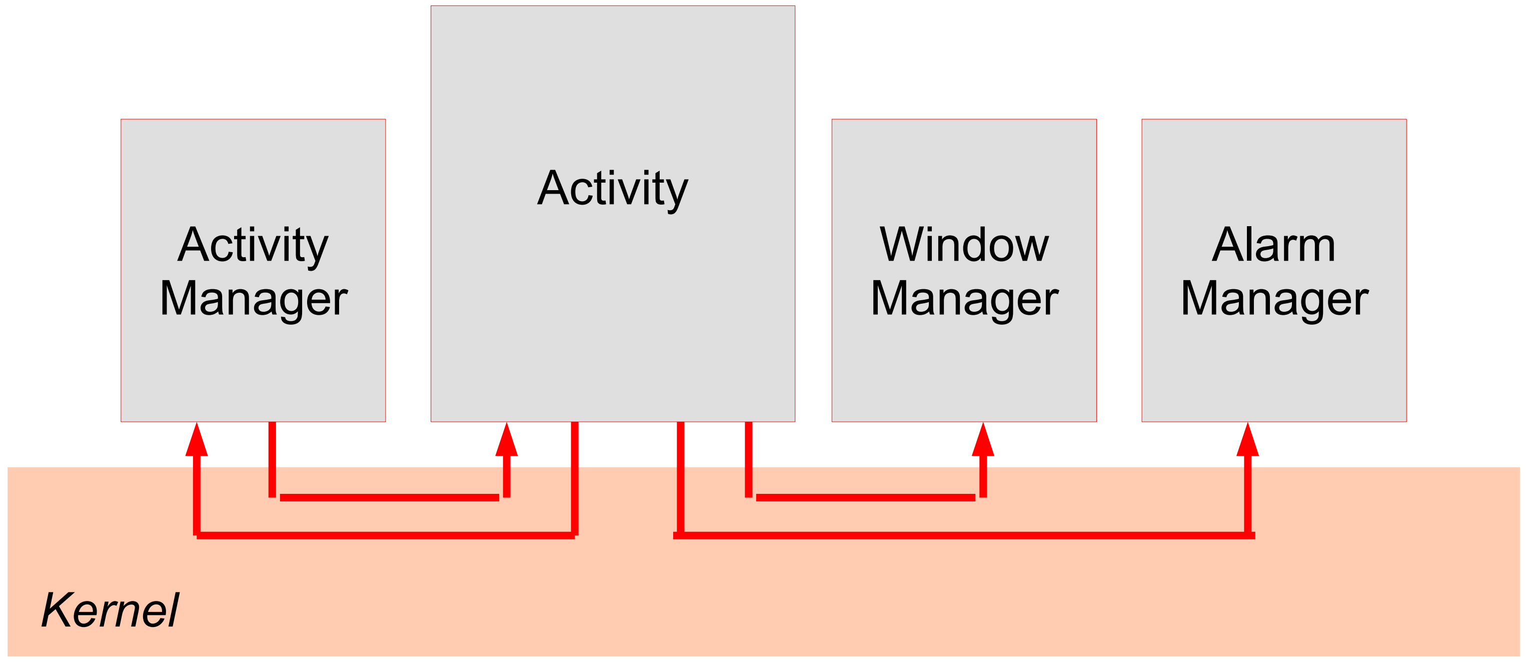
- Different component types
  - Activity
  - Service
  - Content Provider
  - Broadcast Receiver





Application Components System

IPC = Inter-Process Communication



Why IPC?

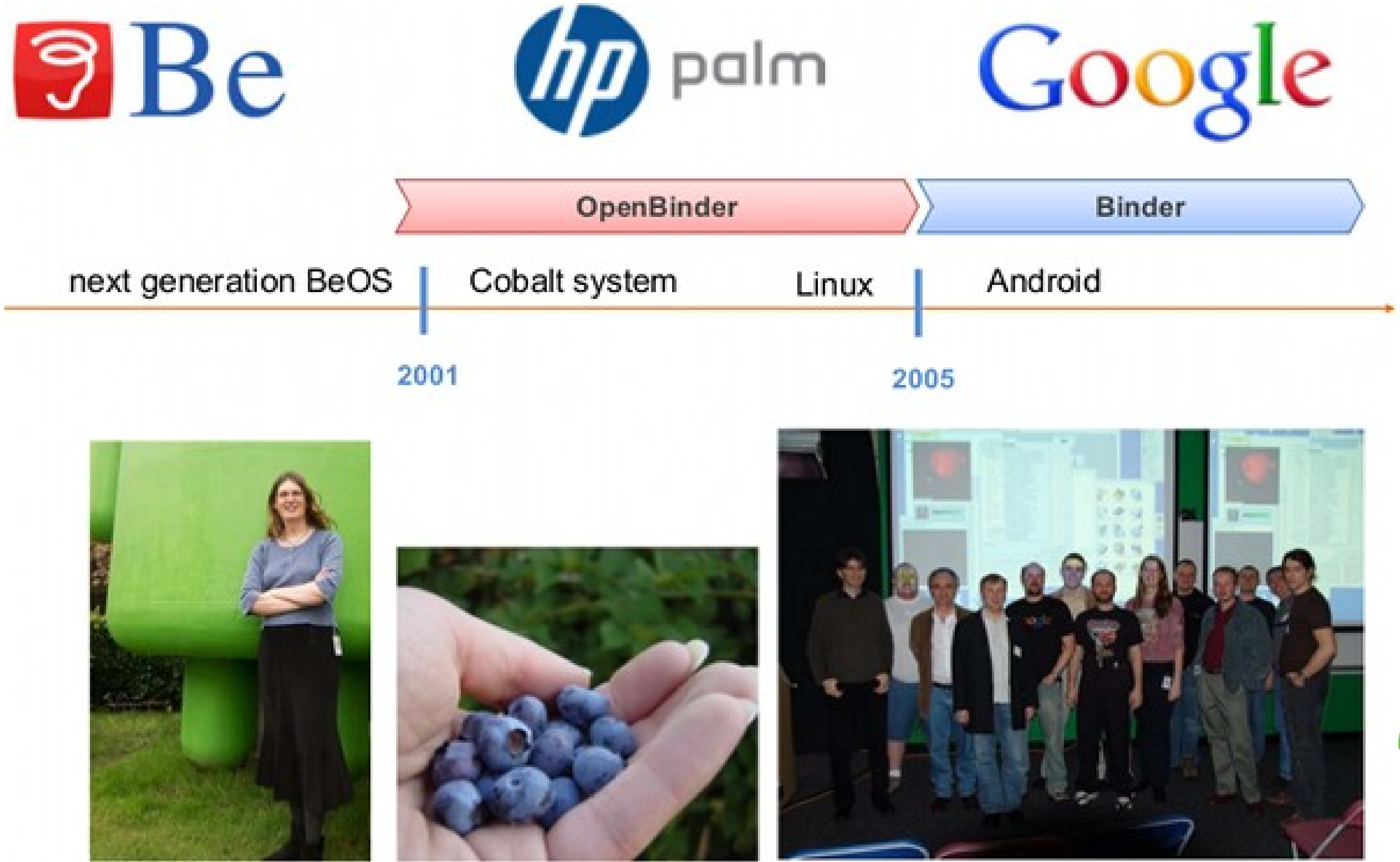
- Each process has its own address space
- Provides data isolation
- Prevents harmful direct interaction between two different processes
  - Sometimes, communication between processes is required for modularization

IPC Mechanisms

- In GNU/Linux
  - Signal
  - Pipe
  - Socket
  - Semaphore
  - Message queue
  - Shared memory
- In Android
  - Binder: lightweight RPC (Remote Procedure Communication) mechanism

Binder History

- Developed under the name OpenBinder by Palm Inc. under the leadership of Dianne Hackborn
- Android Binder is the customized re-implementation of OpenBinder, which provides bindings to functions and data from one execution environment to another



Background Problems

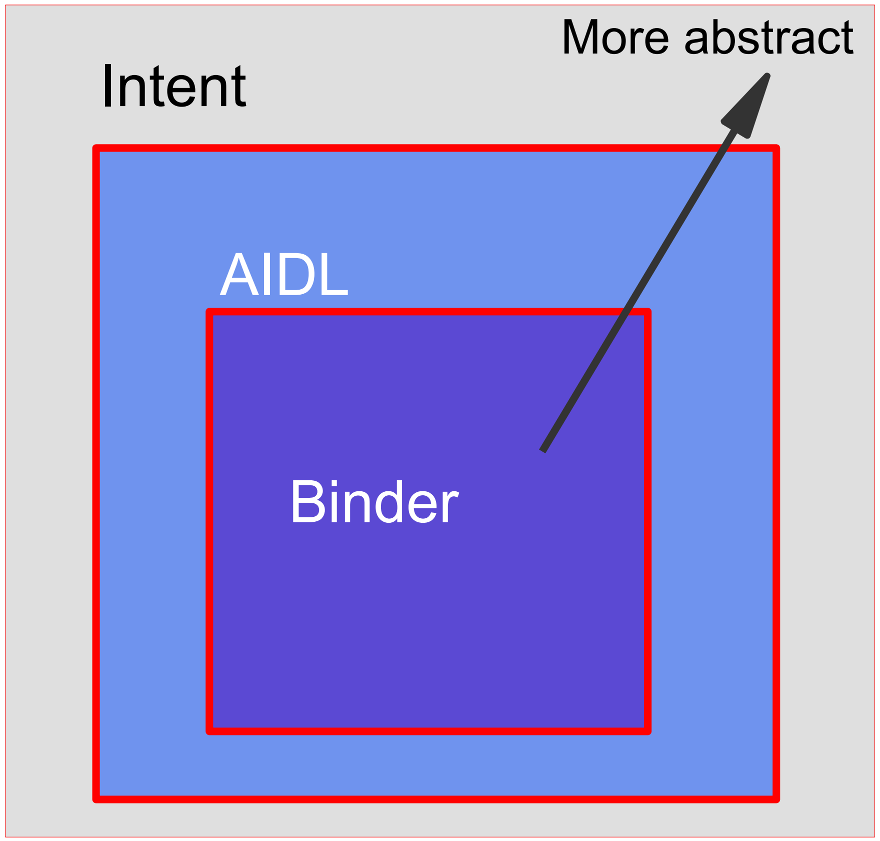
- Applications and Services may run in separate processes but must communicate and share data
- IPC can introduce significant processing overhead andsecurity holes

Binder: Android's Solution

- Driver to facilitate inter-process communication
- High performance through shared memory
- Per-process thread pool for processing requests
- Reference counting, and mapping of object references across processes
- Synchronous calls between processes

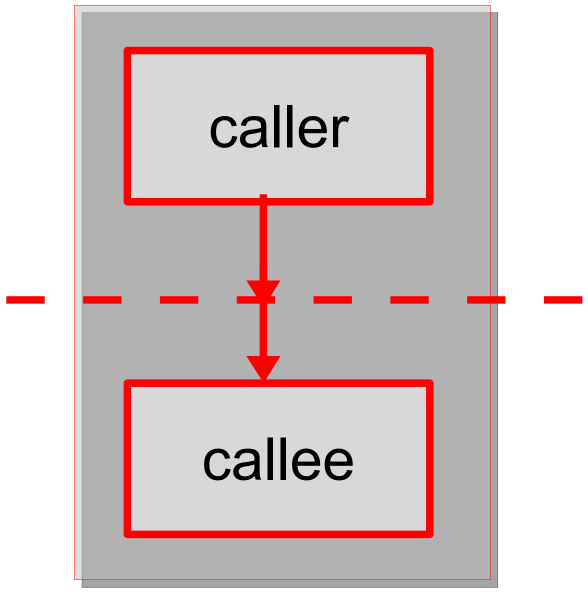
"In the Android platform, the binder is used for nearly everything that happens across processes in the core platform. " – Dianne Hackborn
`https://lkml.org/lkml/2009/6/25/3`

IPC Abstraction



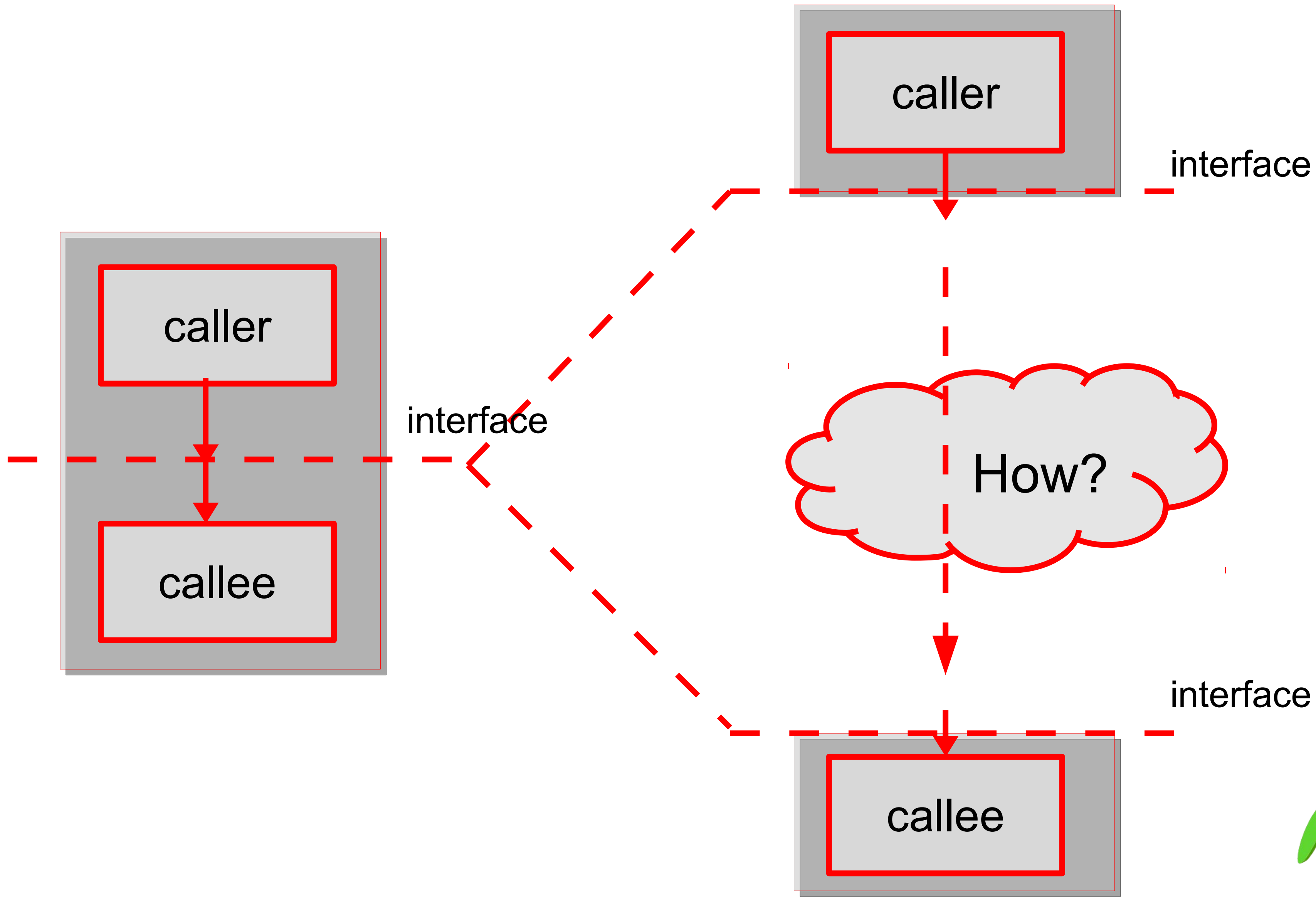
- Intent
  - The highest level abstraction
- Inter process method invocation
  - **AIDL**: Android Interface Definition Language
- binder: kernel driver
- ashmem: shared memory

Method invocation

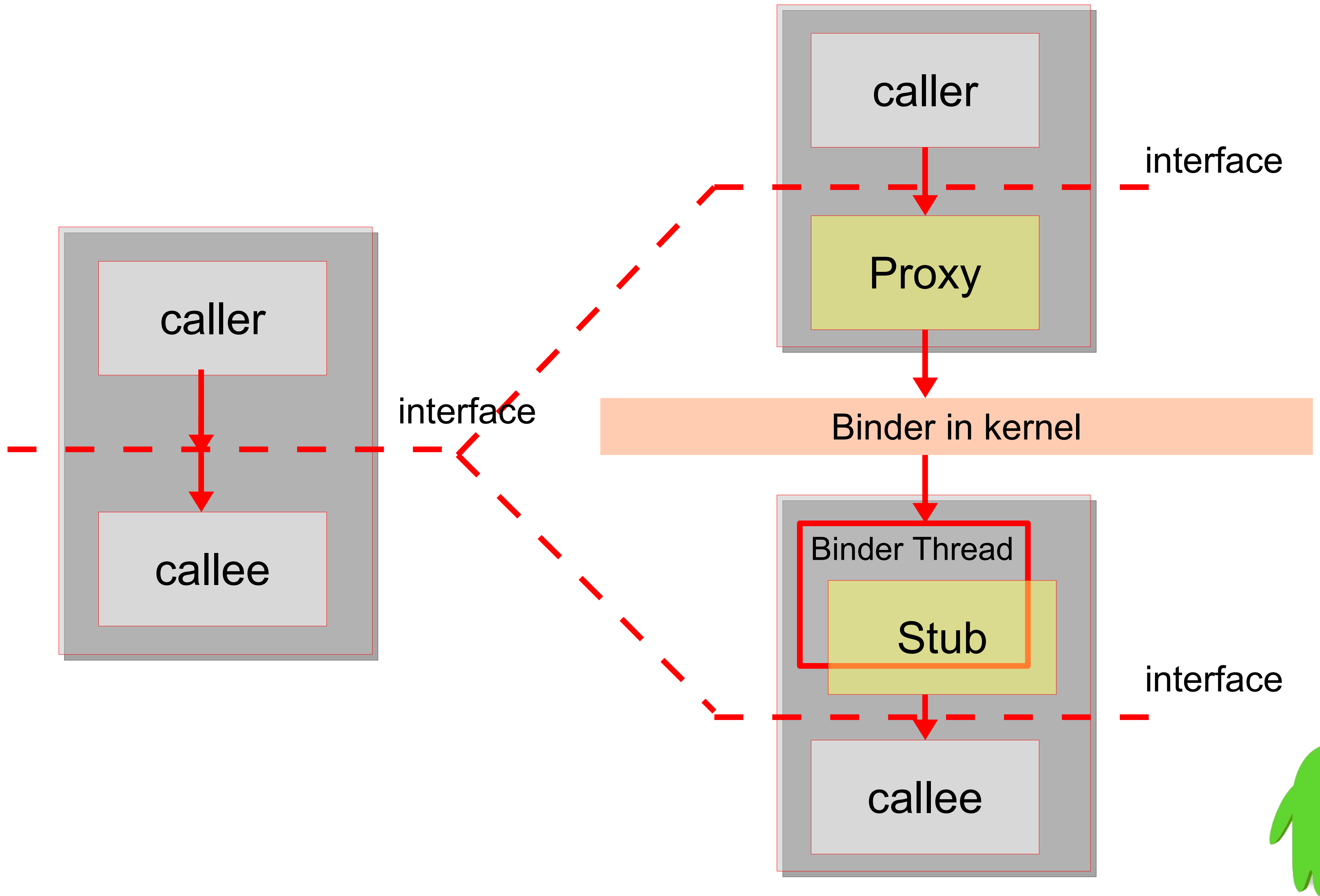


In the same process

Inter-process method invocation



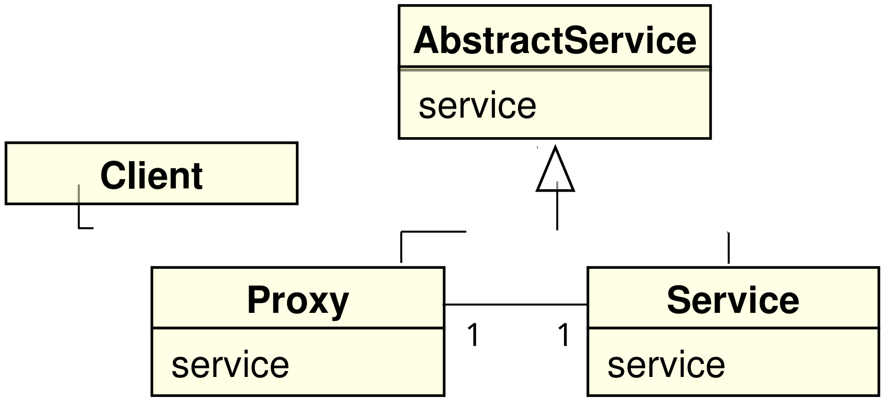
Inter-process method invocation



Design Patterns

Pattern

- Abstracts and names a recurring design structure
- Comprises class and/or object
  - Dependencies
  - Structures
  - Interactions
  - Conventions
- Specifies the design structure explicitly
- is distilled from actual design experience
- Android itself follows object oriented design



The Proxy Pattern

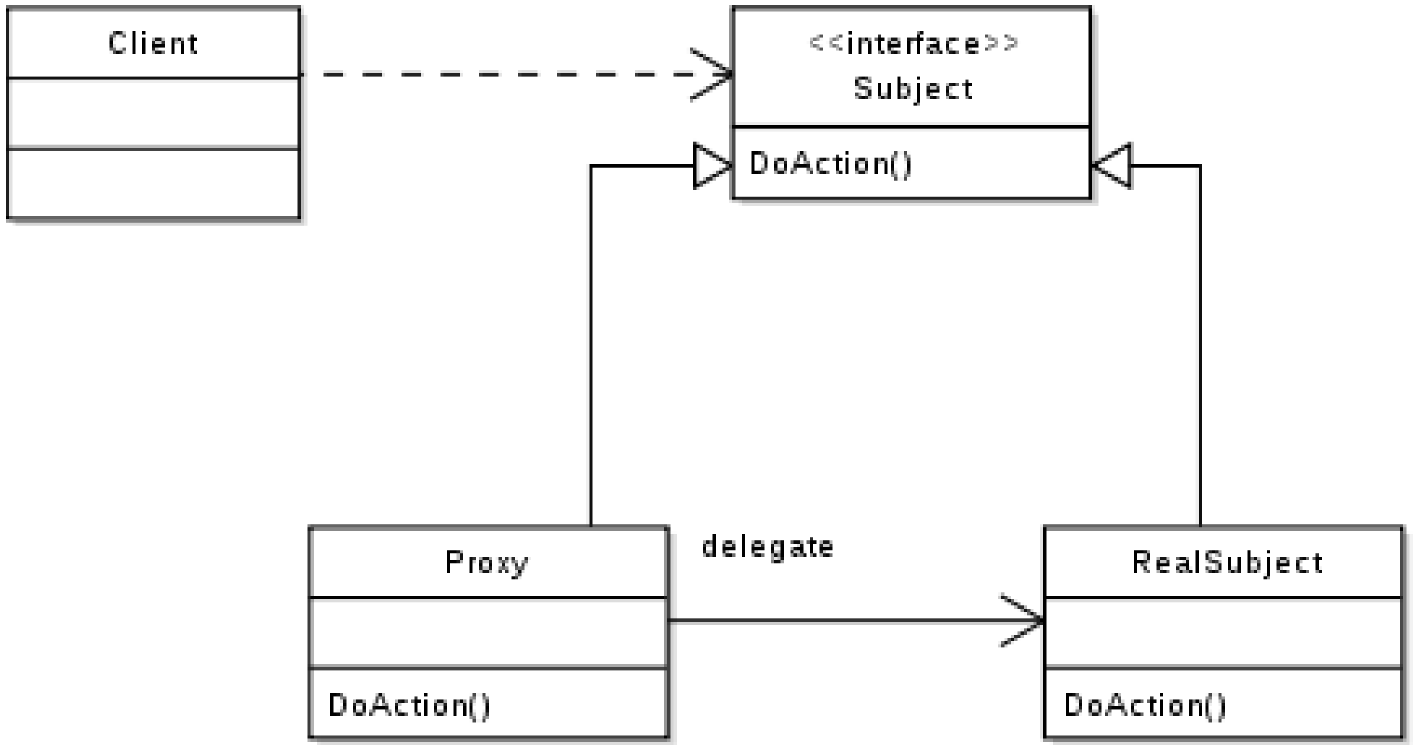
Design Patterns used in Binder

(incomplete)

- Proxy Pattern
- Mediator Pattern
- Bridge Pattern

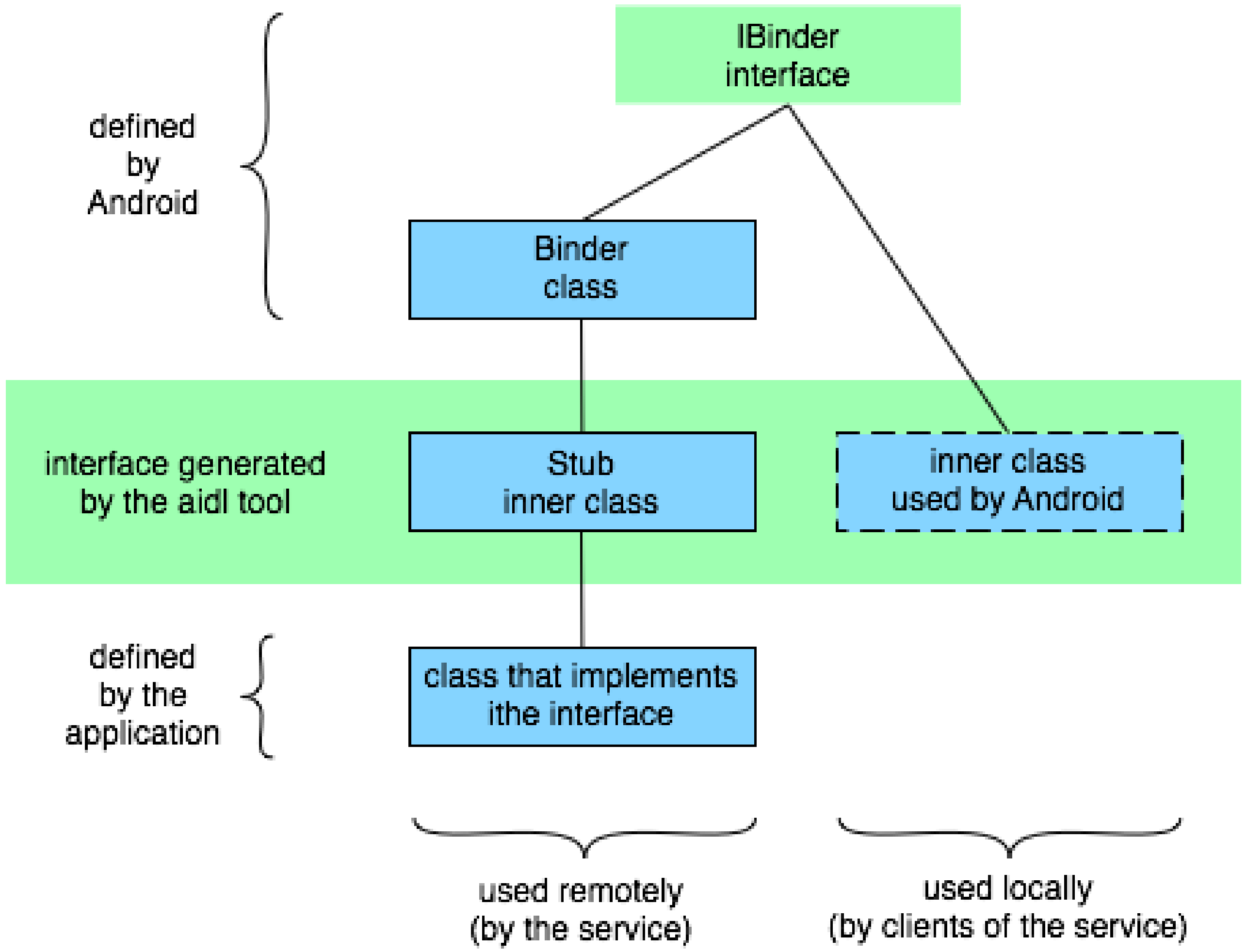
Proxy Pattern

- The proxy could interface to anything: a network connection, a large object in memory, a file, or some other resource that is expensive or impossible to duplicate.



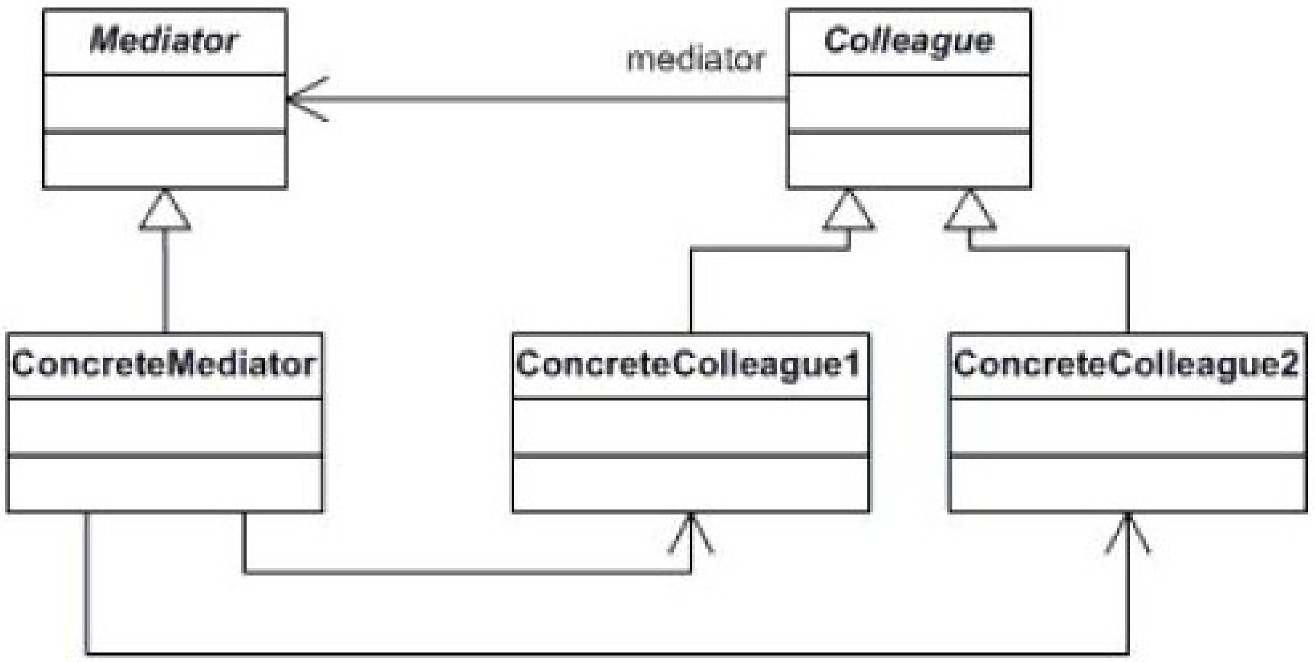
Proxy Pattern in Android

- Binder decomposes the method call and all its corresponding data to a level that Linux can understand, transmitting it from the local process and address space to the remote process and address space, and reassembling and reenacting the call there.


IBinder interface
defined by Android
Binder class
interface generated by the aidl tool
Stub inner class
inner class used by Android
defined by the application
class that implements ithe interface
used remotely (by the service)
used locally (by clients of the service)

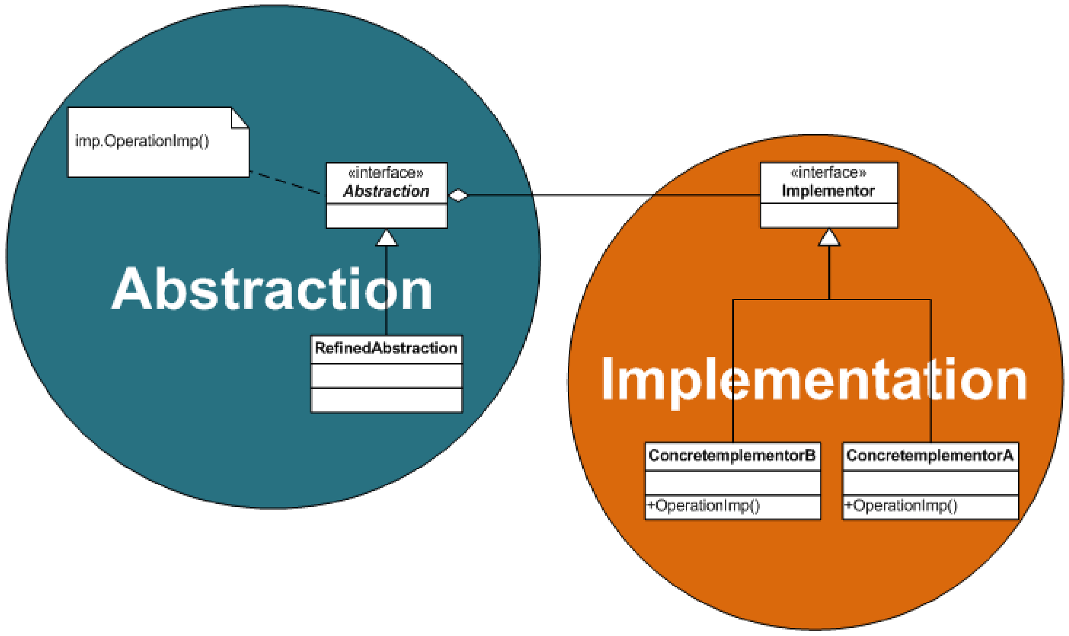
Mediator Pattern

- With the **mediator pattern,** communication between objects is encapsulated with a **mediator** object.



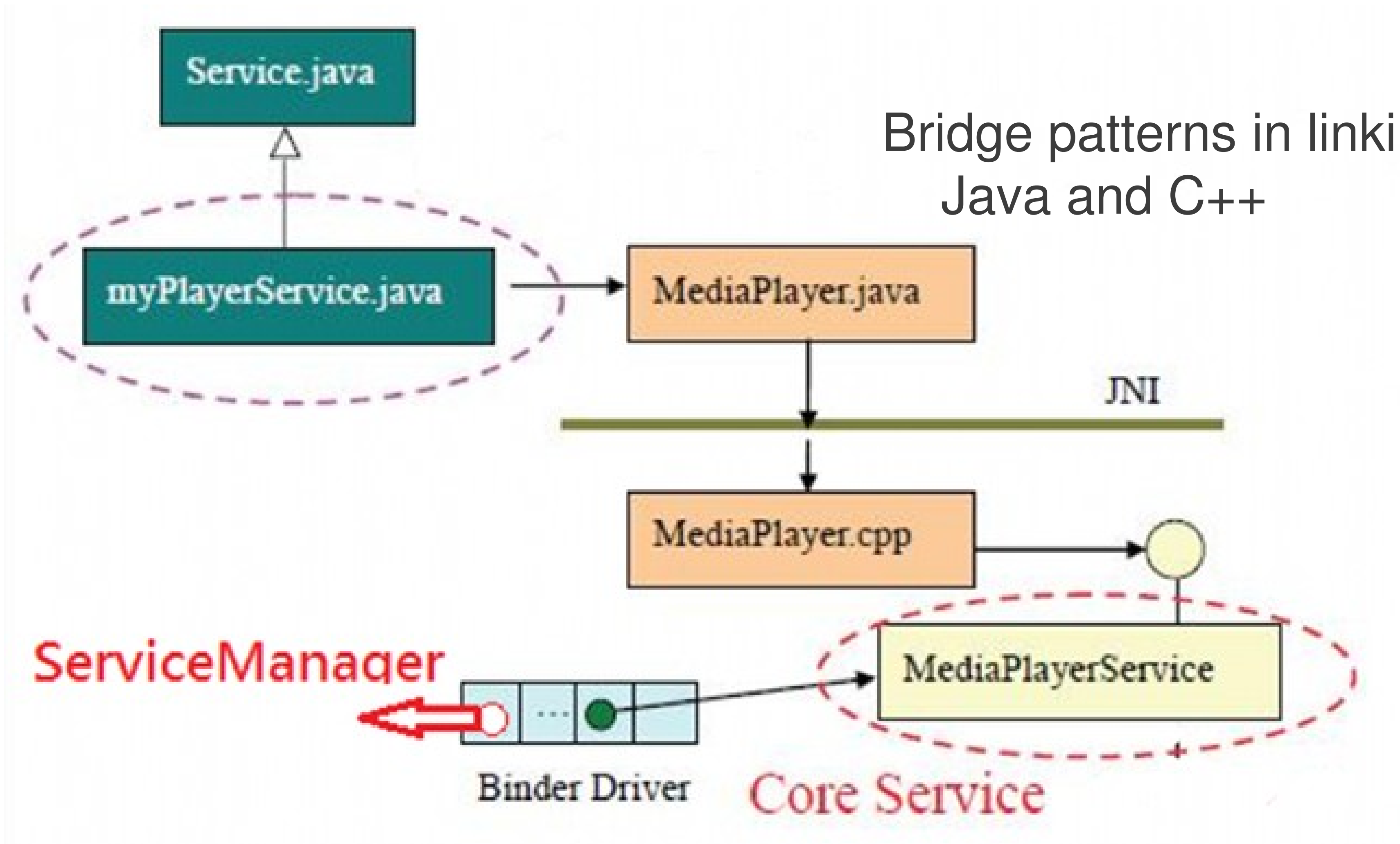
Bridge Pattern

- decouple an abstraction from its implementation so that the two can vary independently



Bridge Pattern - Generic Structure

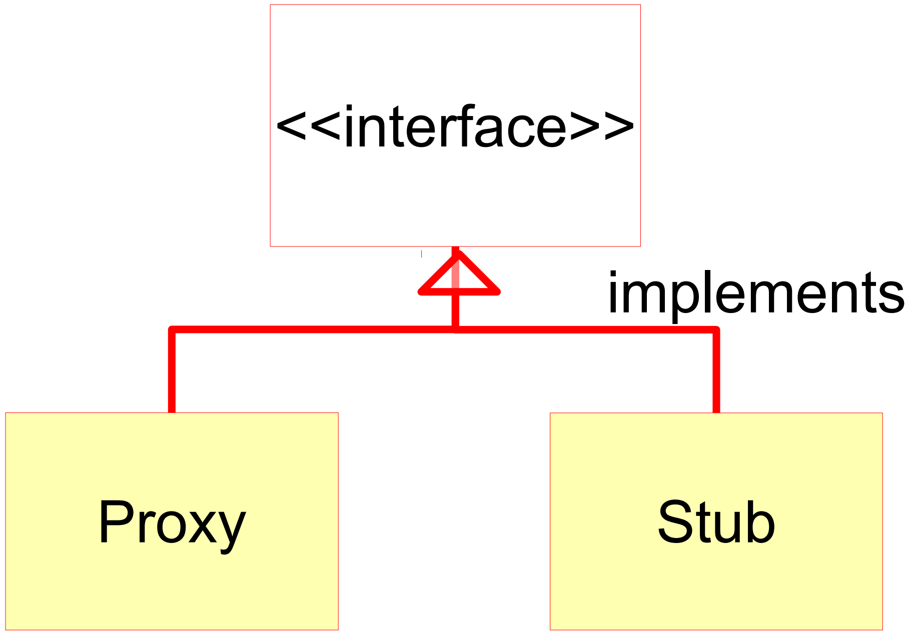
Bridge and Mediator Pattern in Android



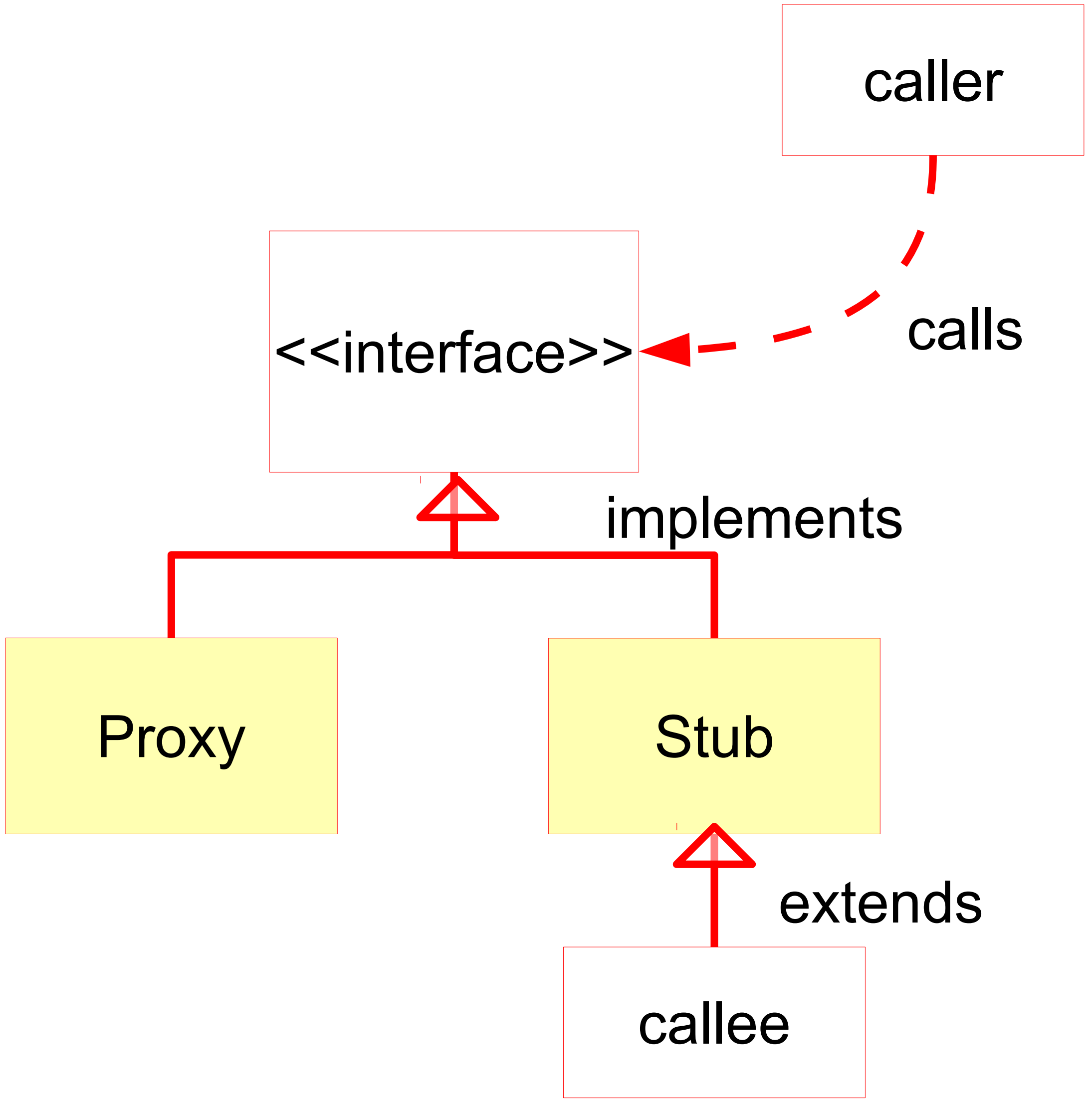
Bridge patterns in linking Java and C++

Mediator pattern

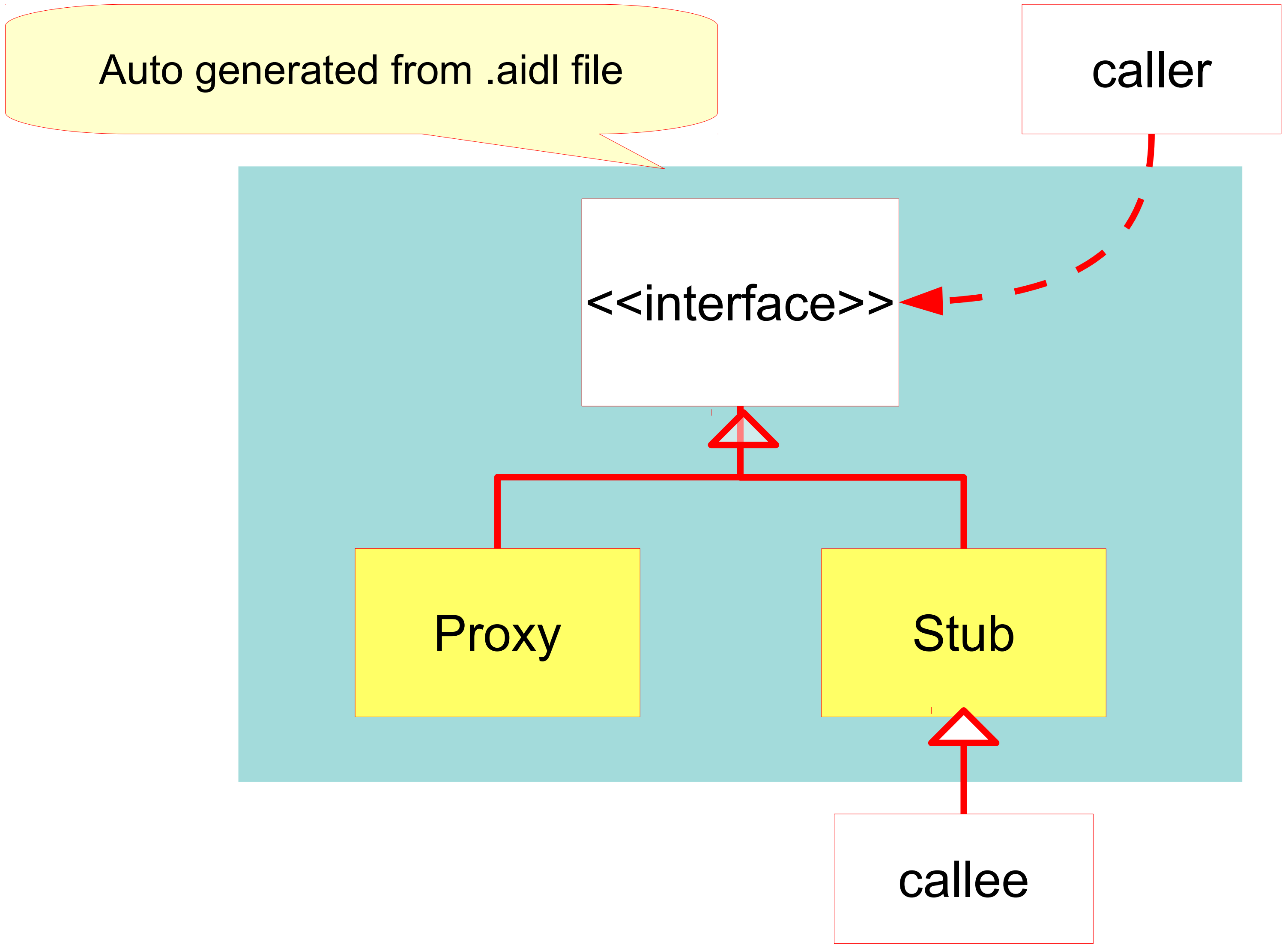
UML Representation



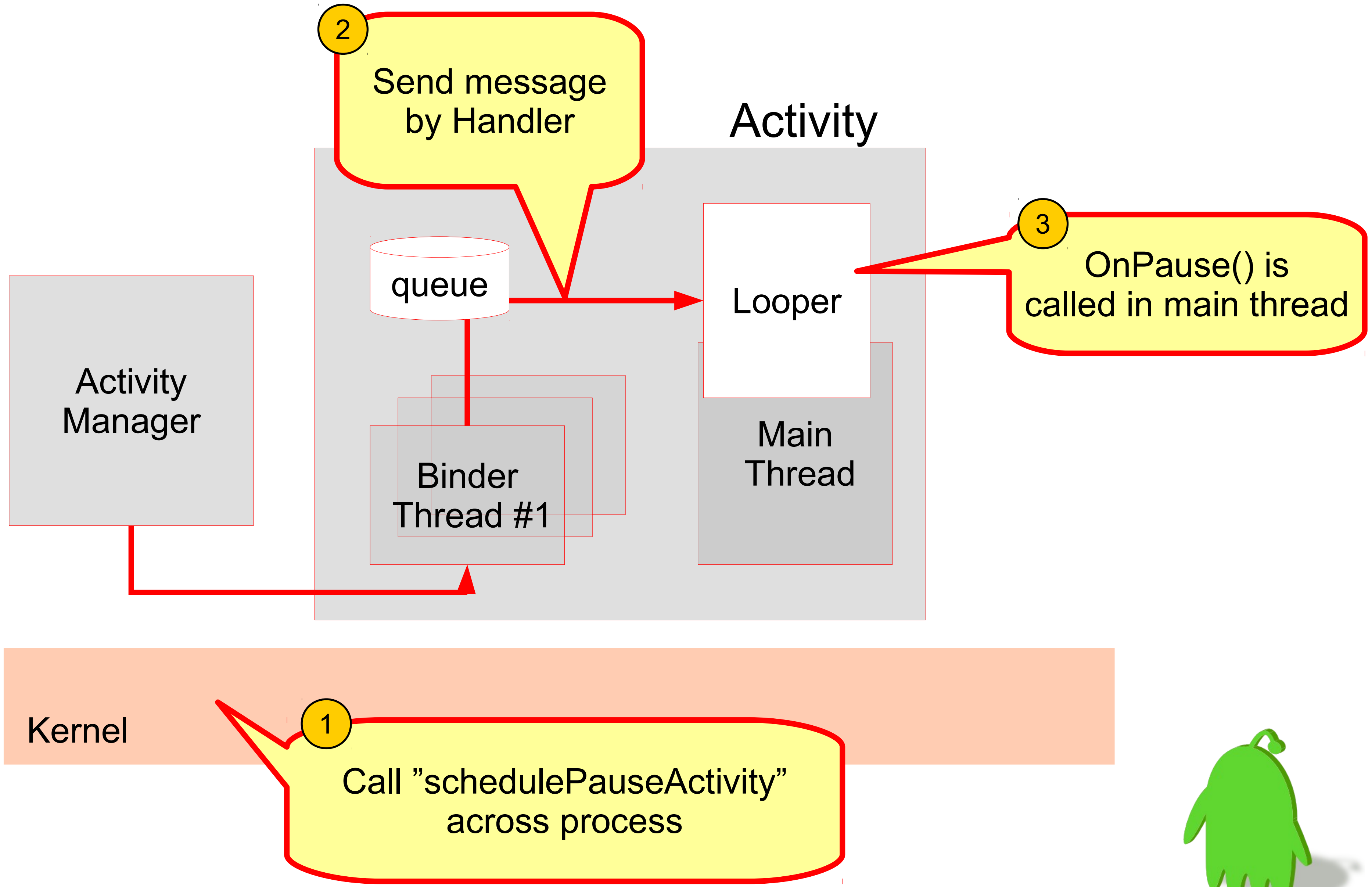
UML Representation



AIDL

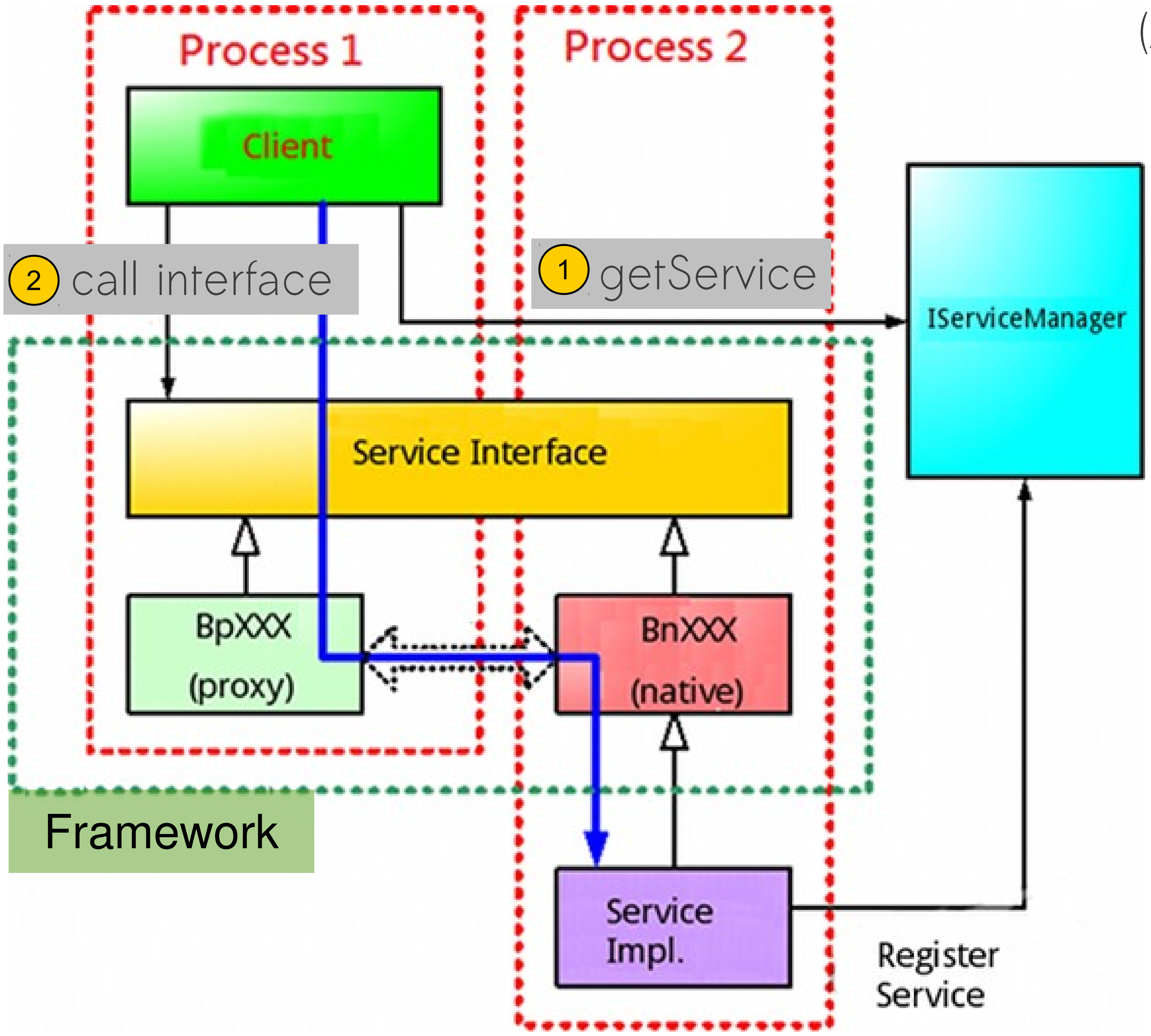


Use Case:
Who calls onPause() in Activity?



IPC Interaction in Android

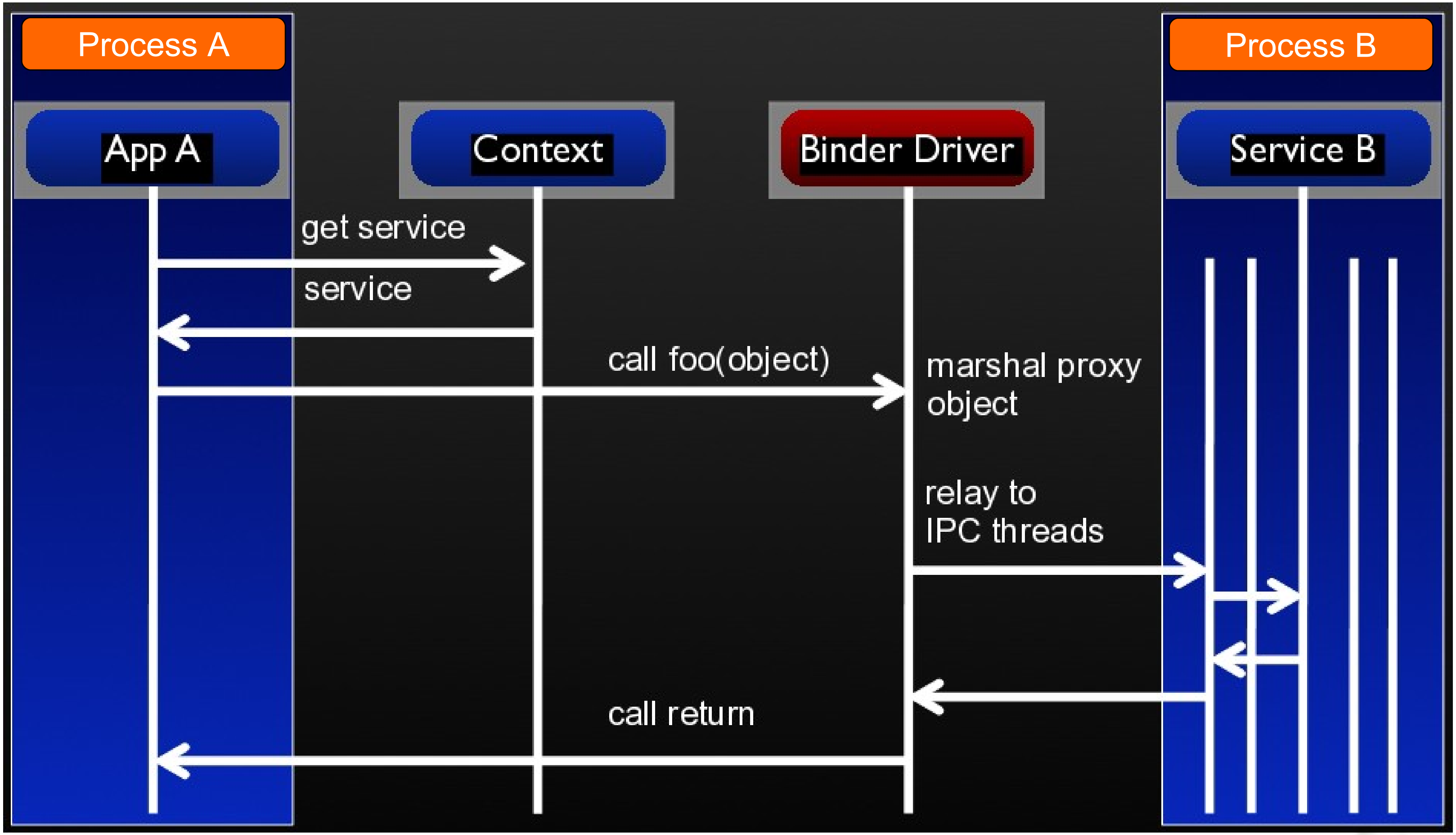
(Application View)



3 parts:

- BnXXX: native
- BpXXX: proxy
- Client Invoke BpXXX

Binder in Action



Binder Internals

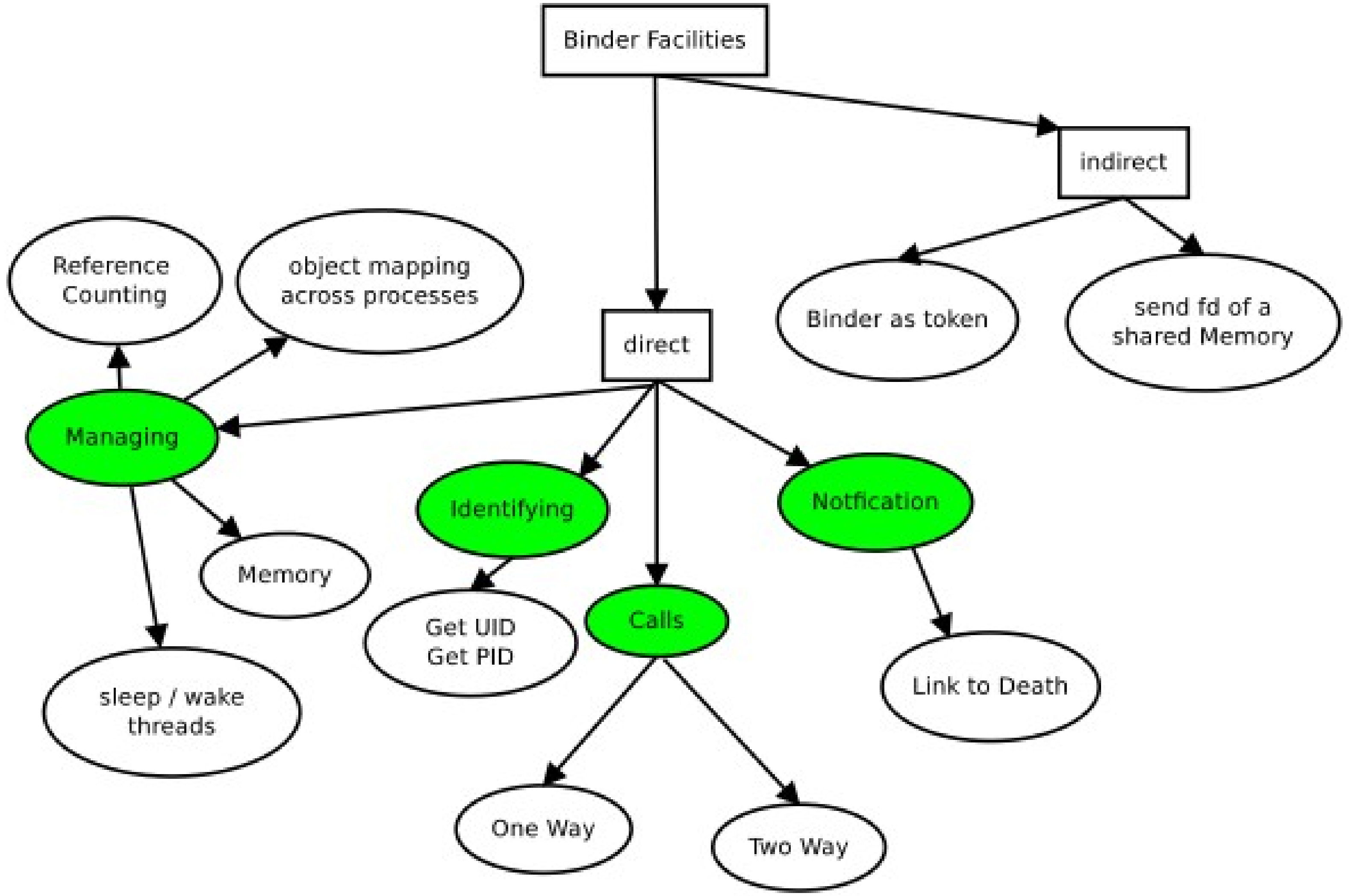
Binder Terminology

- Binder
- Binder Object
  - an instance of a class that implements the Binder interface.
  - One Binder object can implement multiple Binders
- Binder Protocol
- IBinder Interface
  - is a well-defined set of methods, properties and events that a Binder can implement.
- Binder Token
  - A numeric value that uniquely identifies a Binder

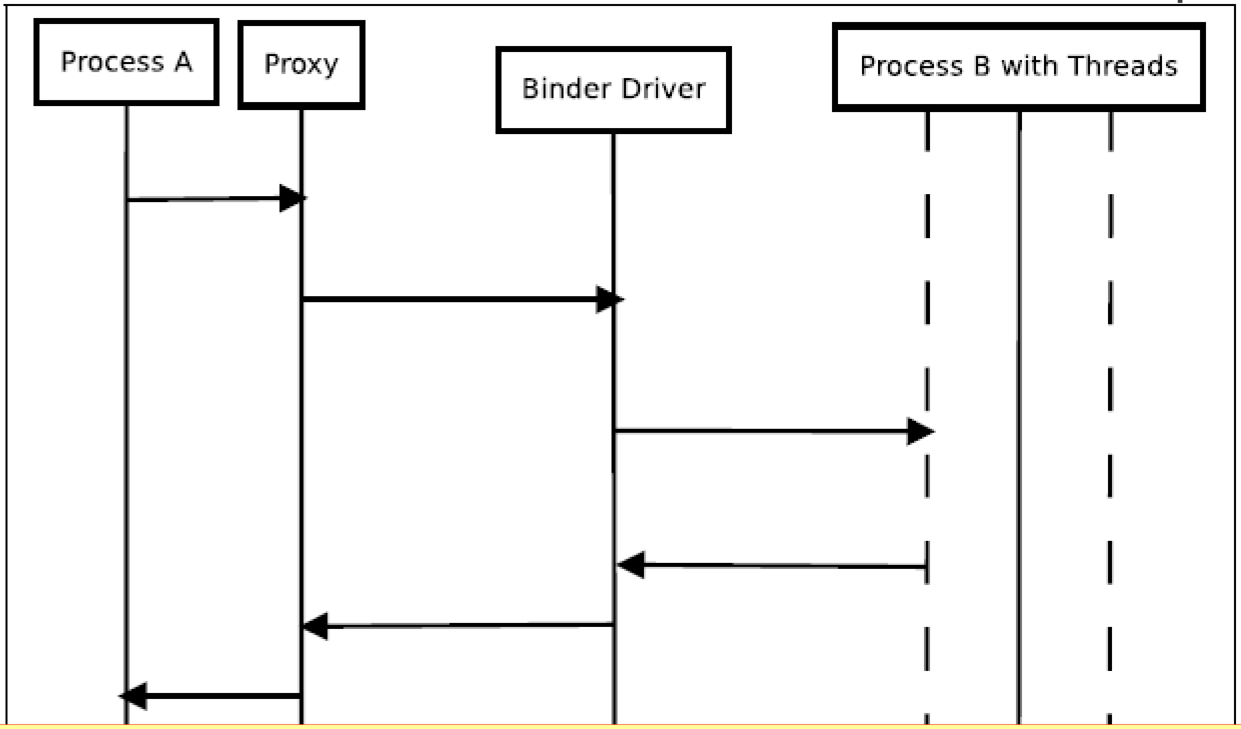
Facilities

- Simple inter process messaging system
- Managing
- Identifying
- Calls
- Notification
- Binder as a security access token

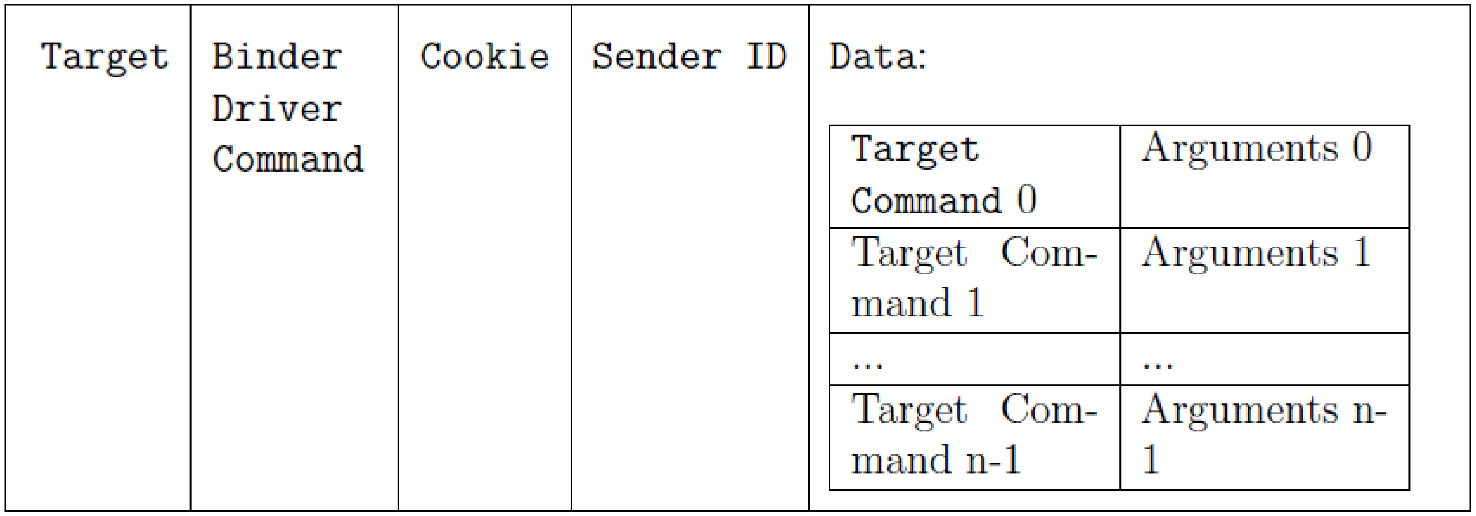
- Binder framework provides more than a simple interprocess messaging system.
- Methods on remote objects can be called as if they where local object methods.



Communication protocol



If one process sends data to another process, it is called transaction.
The data is called transaction data.

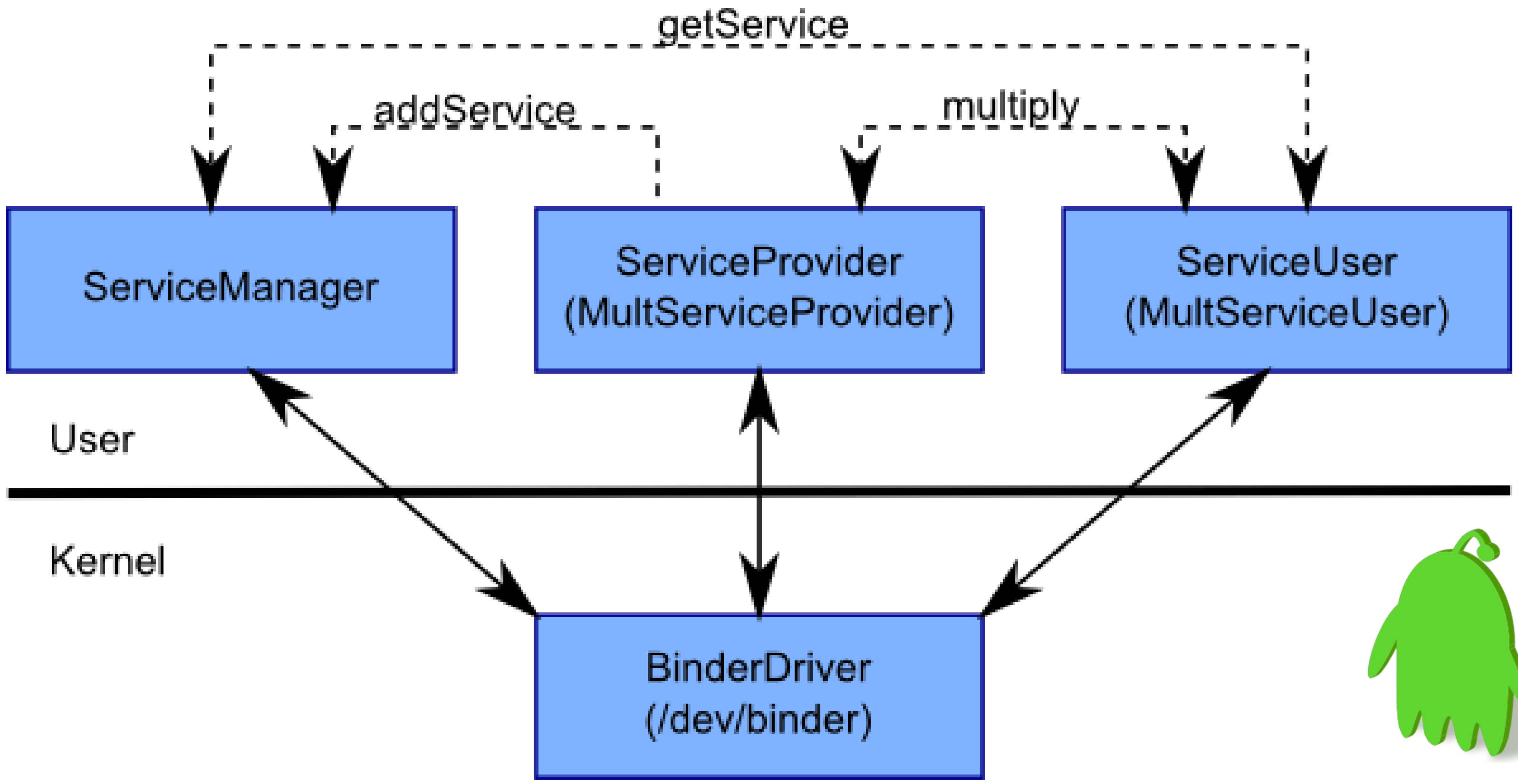


Target	Binder Driver Command	Cookie	Sender ID	Data:

Target Command 0	Arguments 0
Target Command 1	Arguments 1
...	...
Target Command n-1	Arguments n-1

Service Manager (SM)

- Special Binder node with known Binder address
- Client does not know the address of remote Binder
  - only Binder interface knows its own address
- Binder submits a name and its Binder token to SM
  - Client retrieves Binder address with service name from SM



Get Service list from SM

```
$ adb shell service list
Found 71 services:
0  stub_isms: [com.android.internal.telephony.ISms]
1  stub_phone: [com.android.internal.telephony.ITelephony]
2  stub_iphonesubinfo:
               [com.android.internal.telephony.IPhoneSubInfo]
..
5  stub_telephony.registry:
           [com.android.internal.telephony.ITelephonyRegistry]
...
7  stub_activity: [android.app.IActivityManager]
...
9  phone: [com.android.internal.telephony.ITelephony]
…
56 activity: [android.app.IActivityManager]
...
64 SurfaceFlinger: [android.ui.ISurfaceComposer]
...
```

Call remote method in ActivityManager

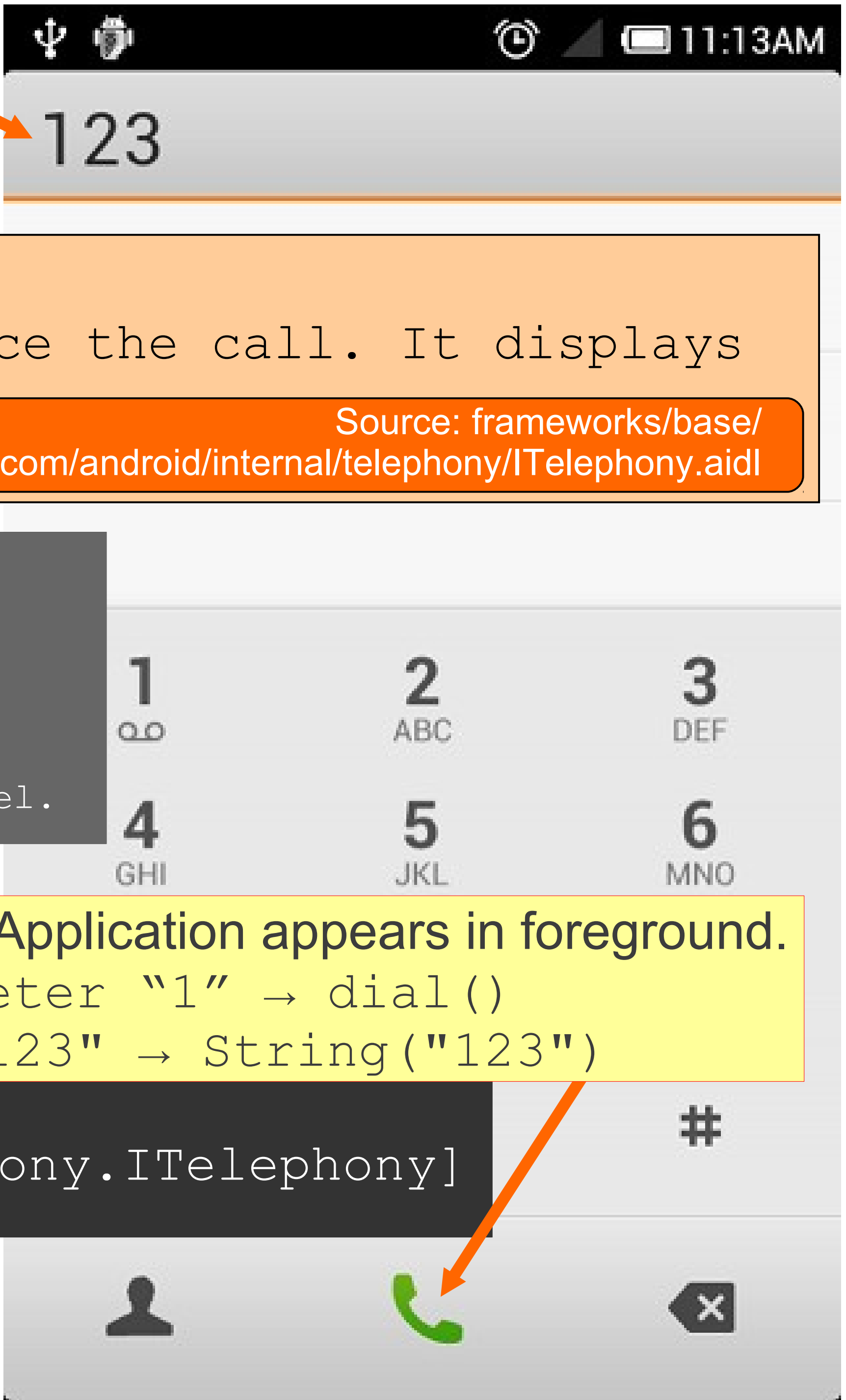
```
$ adb shell service list
...
56 activity: [android.app.IActivityManager]
...
$ adb service call activity 1598968902
Result: Parcel(
  0x00000000: 0000001c 006e0061 00720064 0069006f '....a.n.d.r.o.i.'
  0x00000010: 002e0064 00700061 002e0070 00410049 'd...a.p.p...I.A.'
  0x00000020: 00740063 00760069 00740069 004d0079 'c.t.i.v.i.t.y.M.'
  0x00000030: 006e0061 00670061 00720065 00000000 'a.n.a.g.e.r.....')
```

```
public abstract interface IBinder {
    ...
    field public static final int INTERFACE_TRANSACTION
        = 1598968902; // 0x5f4e5446
    …
}
```

Source: frameworks/base/api/current.txt

Interact with Android Service

```
$ adb service call phone 1 s16 "123"
Result: Parcel(00000000 '....')
```



```
interface ITelephony {
    /* Dial a number. This doesn't place the call. It displays
     * the Dialer screen. */
    void dial(String number);
```

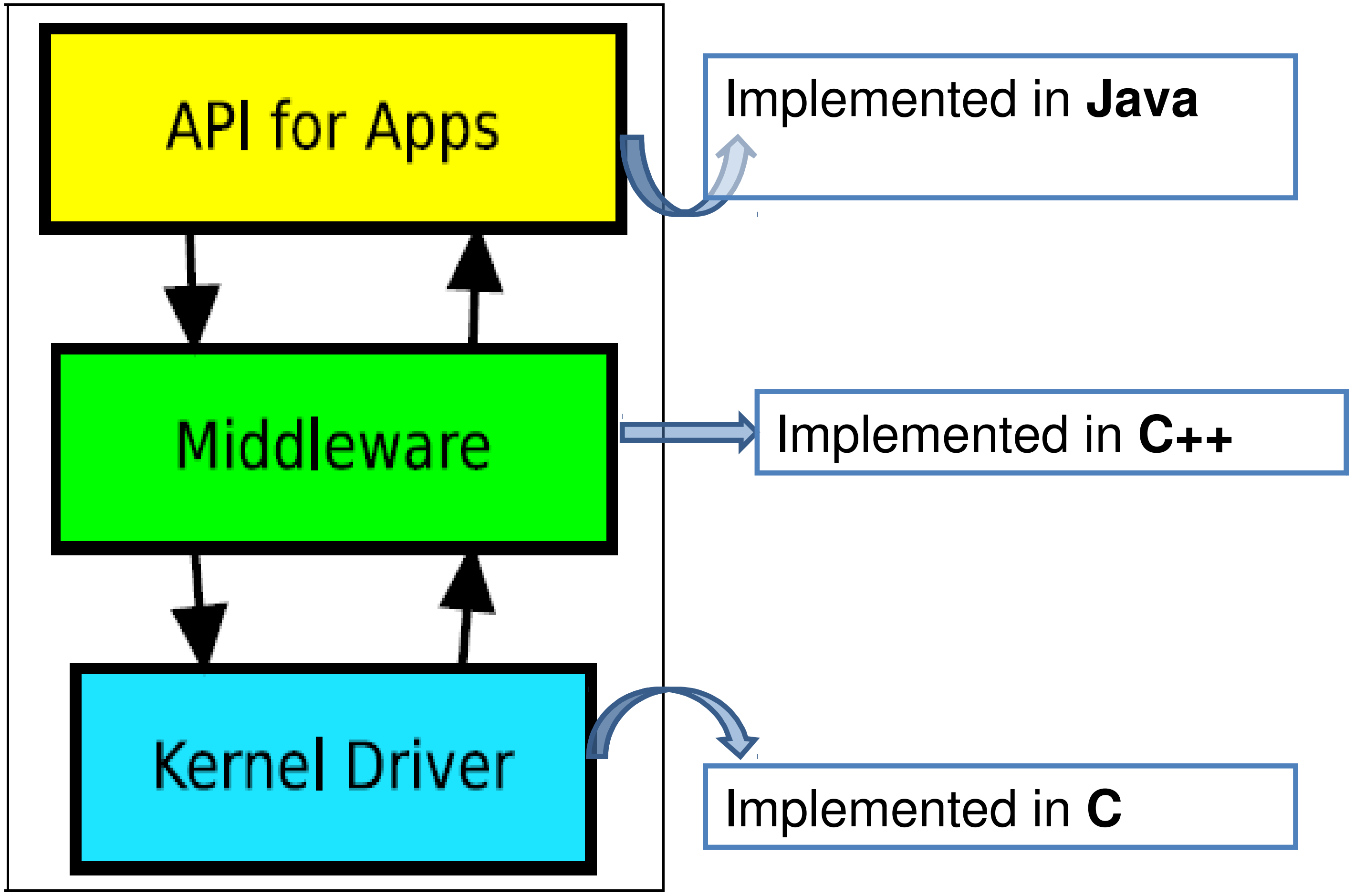
Source: frameworks/base/telephony/java/com/android/internal/telephony/ITelephony.aidl

```
service call SERVICE CODE [i32 INT | s16 STR] …

Options:

   i32: Write the integer INT into the send parcel.

   s16: Write the UTF-16 string STR into the send parcel.
```

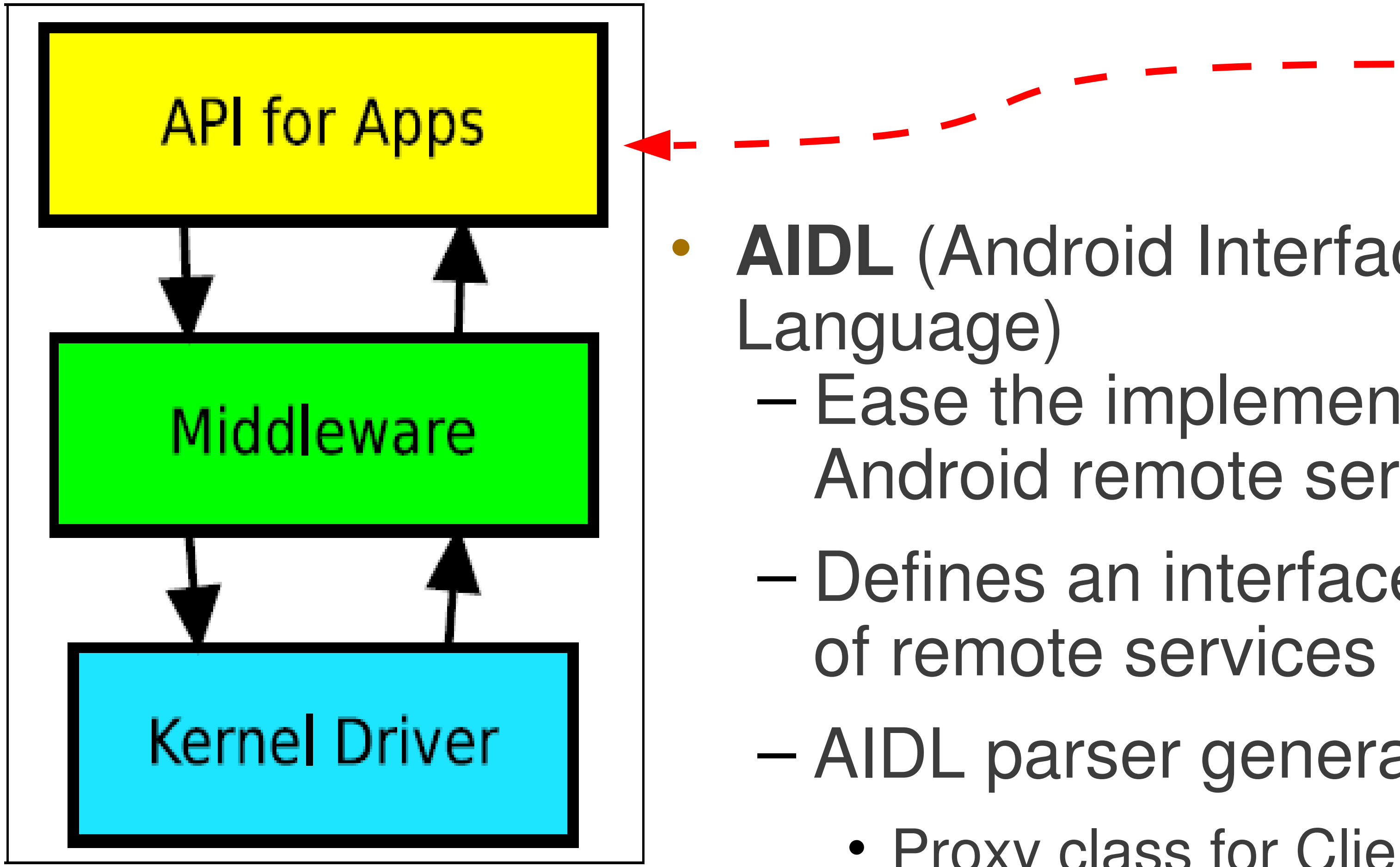
```
$ adb shell service list
Found 71 services:
...
9  phone: [com.android.internal.telephony.ITelephony]
```

Phone Application appears in foreground.
parameter "1" → dial()
s16 "123" → String("123")

Implementation Layers of Binder



API Layer



- **AIDL** (Android Interface Definition Language)
  - Ease the implementation of Android remote services
  - Defines an interface with method of remote services
  - AIDL parser generates Java class
    - Proxy class for Client
    - Stub class for Service
- **Java API Wrapper**
  - Introduce facilities to the binder
  - Wraps the middleware layer

AIDL

- Data Types
  - Java Primitives
  - Containers
    - String, List, Map, CharSequence
    - List<>
    - Multidimensional Array
  - Parcelable
  - Interface Reference
- Direction - in, out, inout
- oneway
  - android.os.IBinder.FLAG_ONEWAY

AIDL Compiler

- Full-fledged Java(-only) Support
- Stub and Proxy Generator

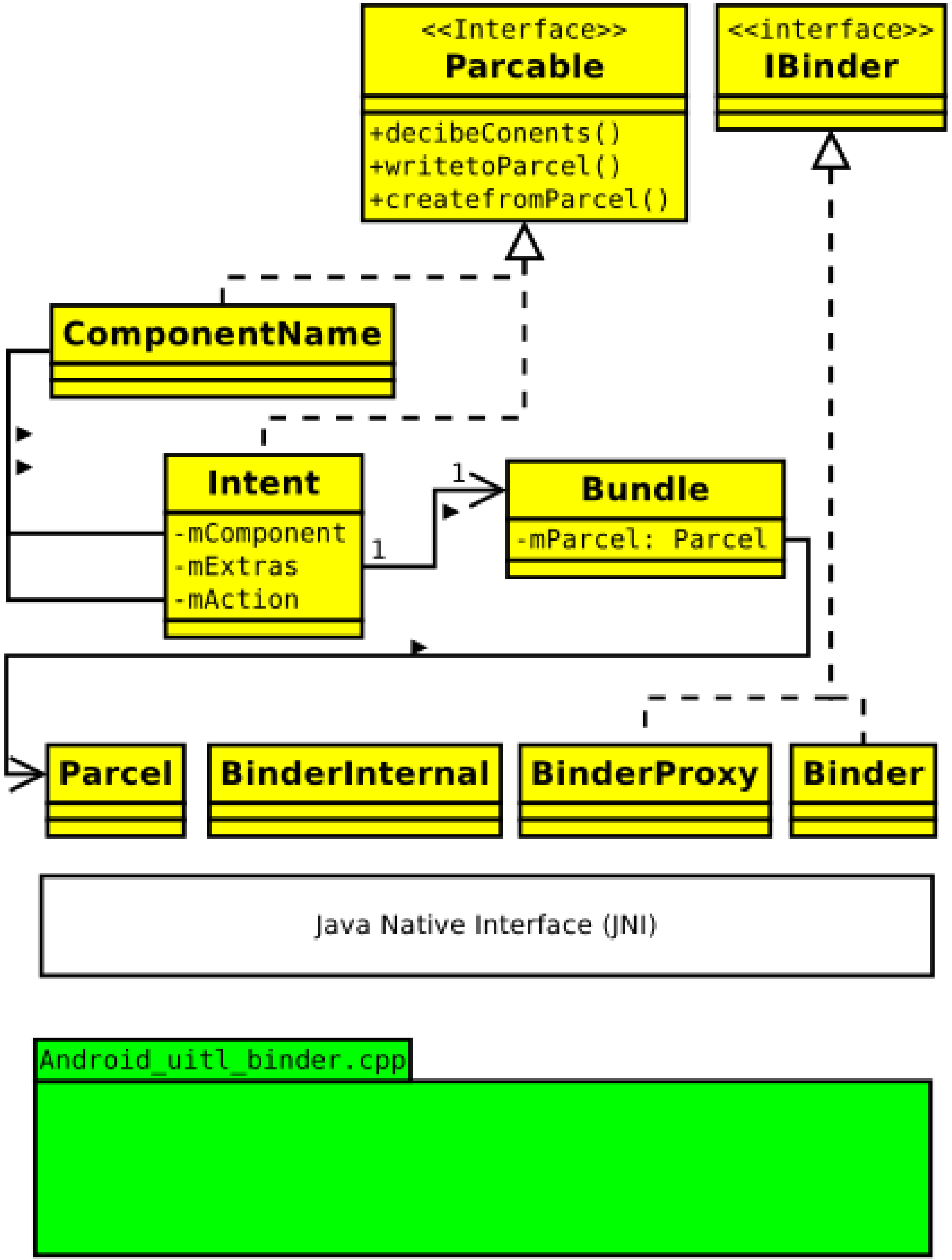
```
// Interface
interface IRemoteService {
    void ping();
}
```

Client

```
IRemoteService mService =
    IRemoteService.Stub.asInterface(service);
```

```
public class RemoteService extends Service {
    public IBinder onBind(Intent intent) { return mBinder; }
    private final IRemoteService.Stub mBinder =
        new IRemoteService.Stub() {
            public void ping() { // Nothing }
    };
}
```

Server



<<Interface>>
Parcable
+decibeConents()
+writetoParcel()
+createfromParcel()
<<interface>>
IBinder
ComponentName
Intent
-mComponent
-mExtras
-mAction
1
1
Bundle
-mParcel: Parcel
Parcel
BinderInternal
BinderProxy
Binder
Java Native Interface (JNI)
Android_uitl_binder.cpp

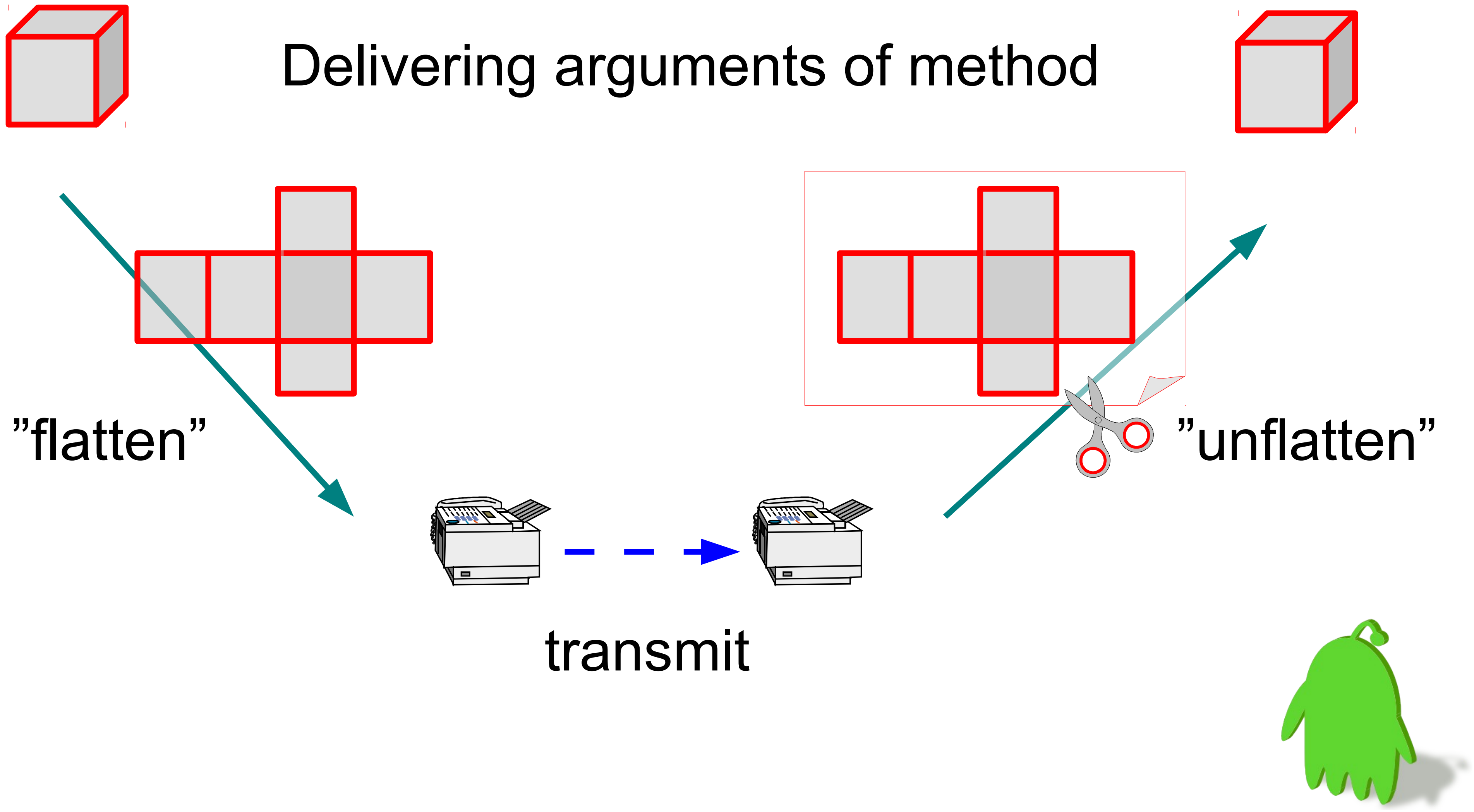
Parcels and Marshalling

- Simple inter process messaging system
- In an object oriented view, the transaction data is called parcel.
- The procedure of building a parcel is called **marshalling** an object.
- The procedure of rebuilding a object from a parcel is called **unmarshalling** an object.

Parcel

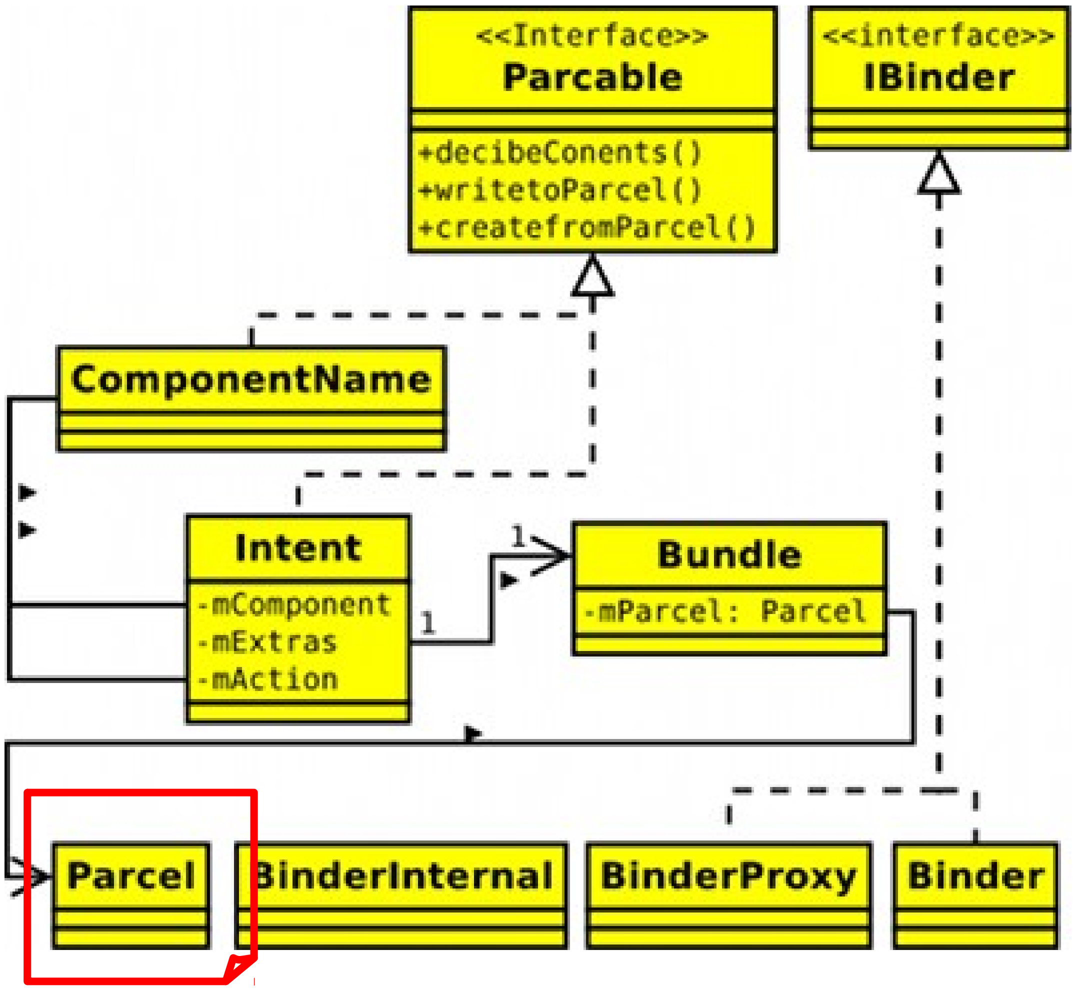
- Marshalling – The transferring of data across process boundaries
  - Represented in native binary encoding
- Mostly handled by AIDL-generated code
- Extensible – Parcelable

android.os.Parcel



Parcel Definition

- Container for a message (data and object references) that can be sent through an IBinder.



- A Parcel can contain both flattened data that will be unflattened on the other side of the IPC (using the various methods here for writing specific types, or the general Parcelable interface), and references to live IBinder objects that will result in the other side receiving a proxy IBinder connected with the original IBinder in the Parcel.

Representation of Parcel

- Parcel is not for general-purpose serialization
  - This class (and the corresponding Parcelable API for placing arbitrary objects into a Parcel) is designed as a high-performance IPC transport.
  - Not appropriate to place any Parcel data into persistent storage
- Functions for writing/reading primitive data types:
  - `writeByte(byte) / readByte()`
  - `writeDouble(double) / readDouble()`
  - `writeFloat(float) / readFloat()`
  - `writeInt(int) / readInt()`
  - `writeLong(long) / readLong()`
  - `writeString(String) / readString()`

Parcelable

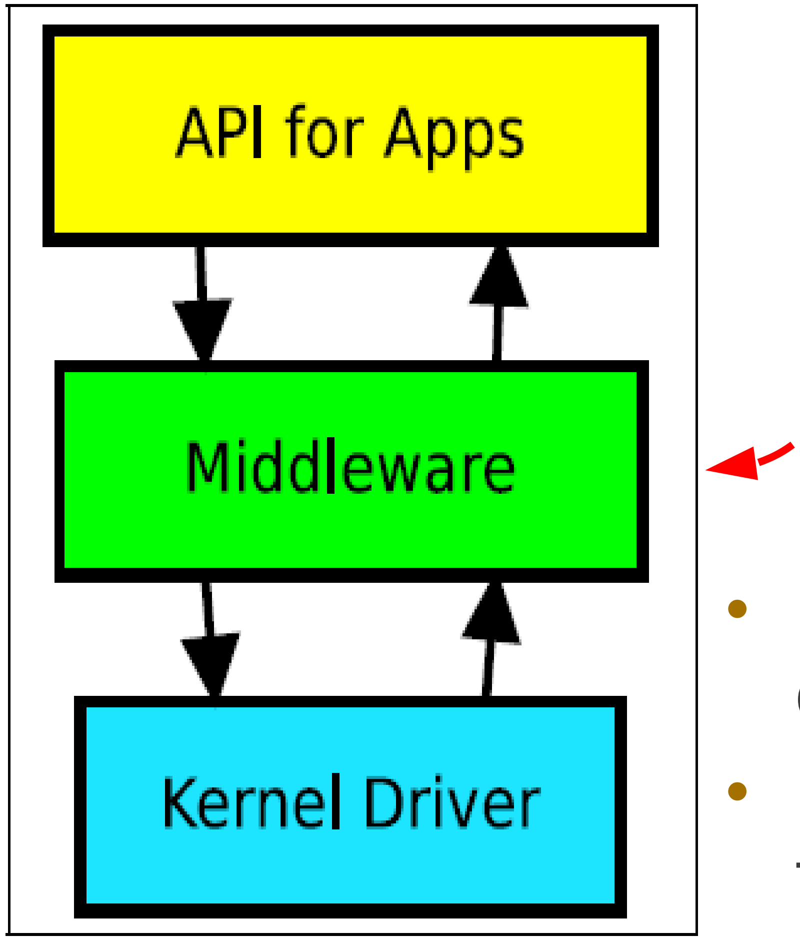
- The Parcelable protocol provides an extremely efficient (but low-level) protocol for objects to write and read themselves from Parcels.
- Use the direct methods to write/read
  - **`writeParcelable`**`(Parcelable, int)`

    `readParcelable``(ClassLoader)`
  - **`writeParcelableArray`**`(T[],int)`

    `readParcelableArray``(ClassLoader)`
- These methods write both the class type and its data to the Parcel, allowing that class to be reconstructed from the appropriate class loader when later reading.

Bundles

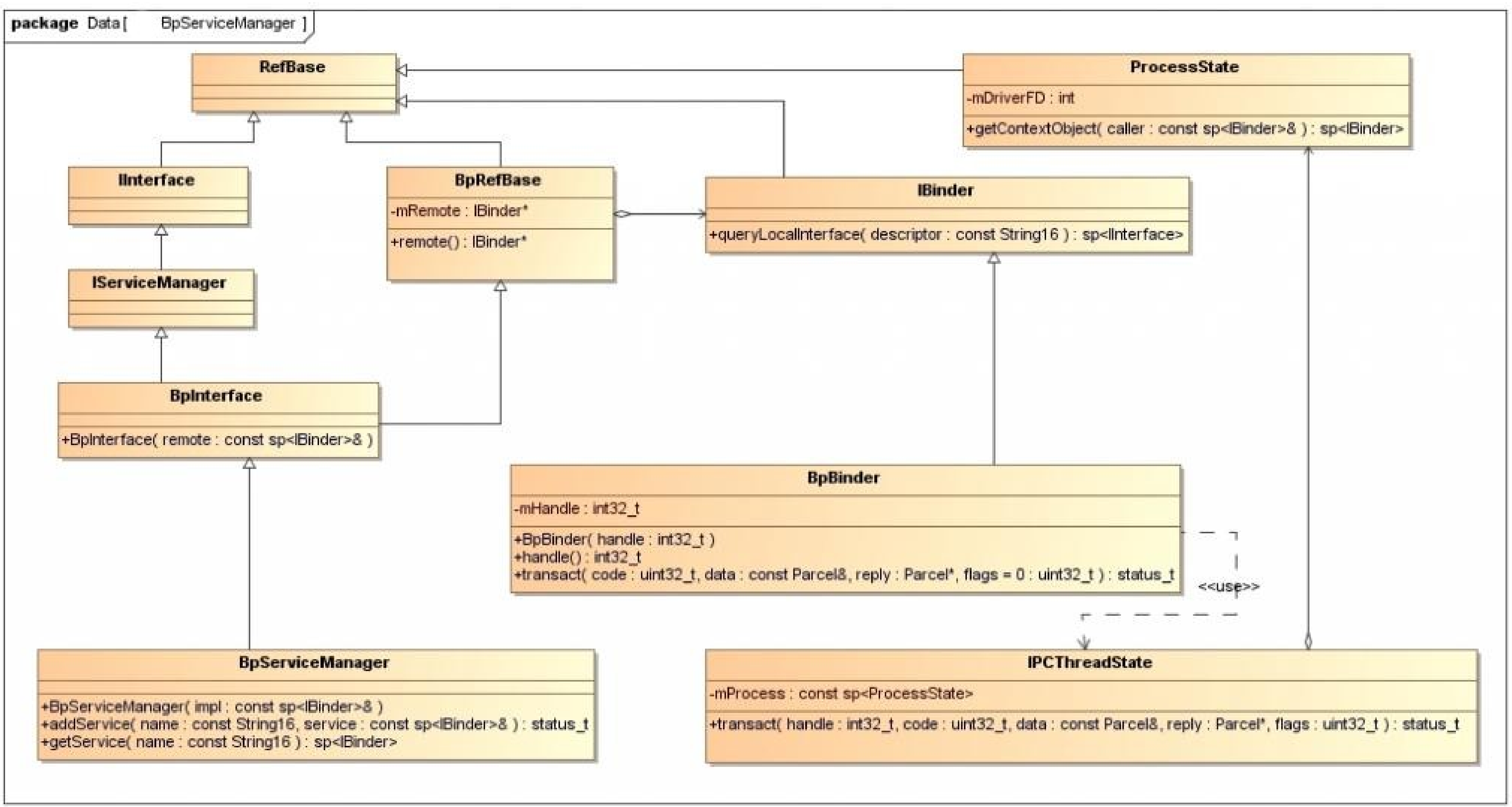
- A special type-safe container, called Bundle, is available for key/value maps of heterogeneous values.
- This has many optimizations for improved performance when reading and writing data, and its type-safe API avoids difficult to debug type errors when finally marshalling the data contents into a Parcel.



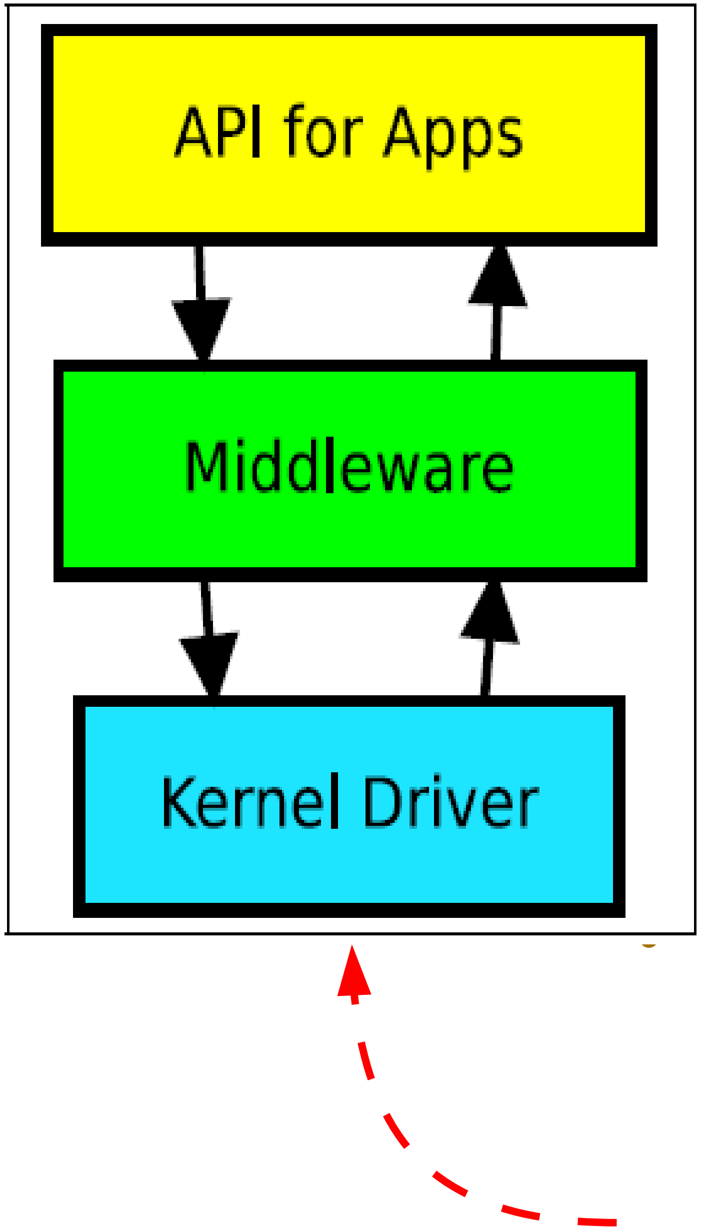
Middleware Layer

- Implements the user space facilities of the Binder framework in C++
- Implements structures and methods to spawn and manage new threads
- Marshalling and unmarshalling of specific data
- Provides interaction with the Binder kernel driver

- frameworks/base/include/binder/IServiceManager.h
  `sp<IServiceManager> defaultServiceManager()`
- frameworks/base/include/binder/IInterface.h
  `template` **`BpInterface`**


package Data [BpServiceManager]
RefBase
ProcessState
-mDriverFD : int
+getContextObject(caller : const sp<IBinder>&) : sp<IBinder>
IInterface
BpRefBase
-mRemote : IBinder*
+remote() : IBinder*
IBinder
+queryLocalInterface(descriptor : const String16) : sp<IInterface>
IServiceManager
BpInterface
+BpInterface(remote : const sp<IBinder>&)
BpBinder
-mHandle : int32_t
+BpBinder(handle : int32_t)
+handle() : int32_t
+transact(code : uint32_t, data : const Parcel&, reply : Parcel*, flags = 0 : uint32_t) : status_t
<<use>>
BpServiceManager
+BpServiceManager(impl : const sp<IBinder>&)
+addService(name : const String16, service : const sp<IBinder>&) : status_t
+getService(name : const String16) : sp<IBinder>
IPCThreadState
-mProcess : const sp<ProcessState>
+transact(handle : int32_t, code : uint32_t, data : const Parcel&, reply : Parcel*, flags : uint32_t) : status_t

Kernel Driver Layer

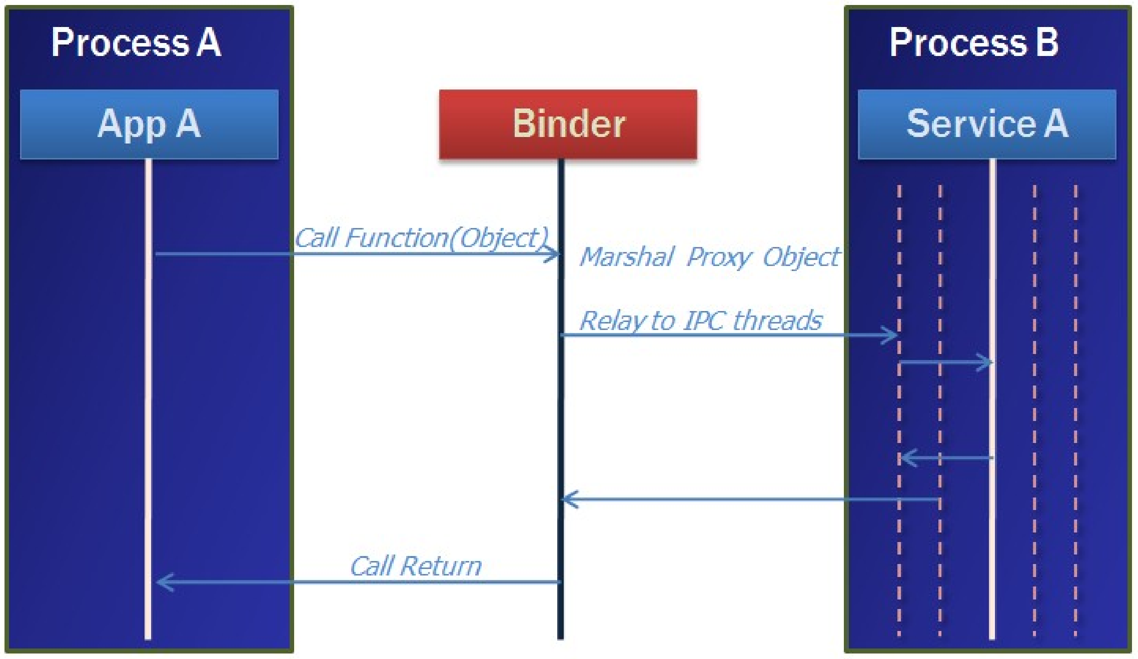


- Binder Driver supports the file operations open, mmap, release, poll and the system call ioctl
- ioctl arguments
  - Binder driver command code
  - Data buffer

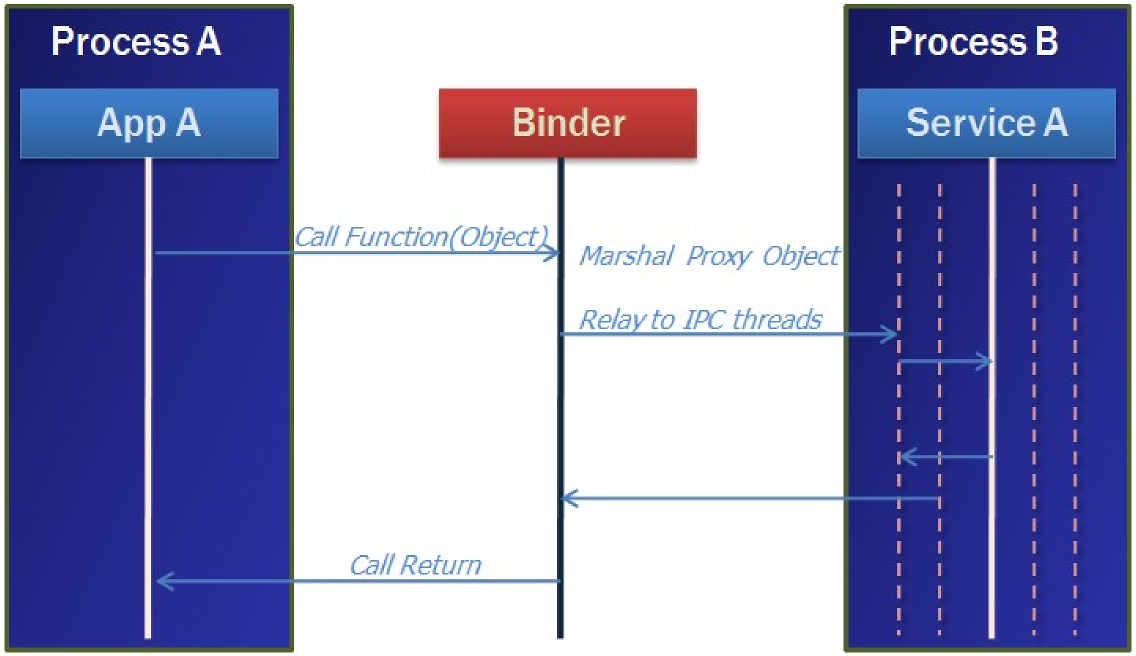
Command codes
- BINDER_WRITE_READ
- BINDER_SET_MAX_THREADS
- BINDER_SET_CONTEXT_MGR
- BINDER_THREAD_EXIT
- BINDER_VERSION

Binder Driver

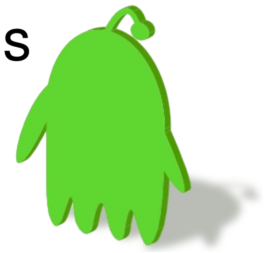
- Multi-thread aware
  - Have internal status per thead
  - Compare to UNIX socket: sockets have internal status per file descriptor (FD)



Binder Driver



- A pool of threads is associated to each service application to process incoming IPC
- Binder performs mapping of object between two processes.
- Binder uses an object reference as an address in a process's memory space.
- Synchronous call, reference counting

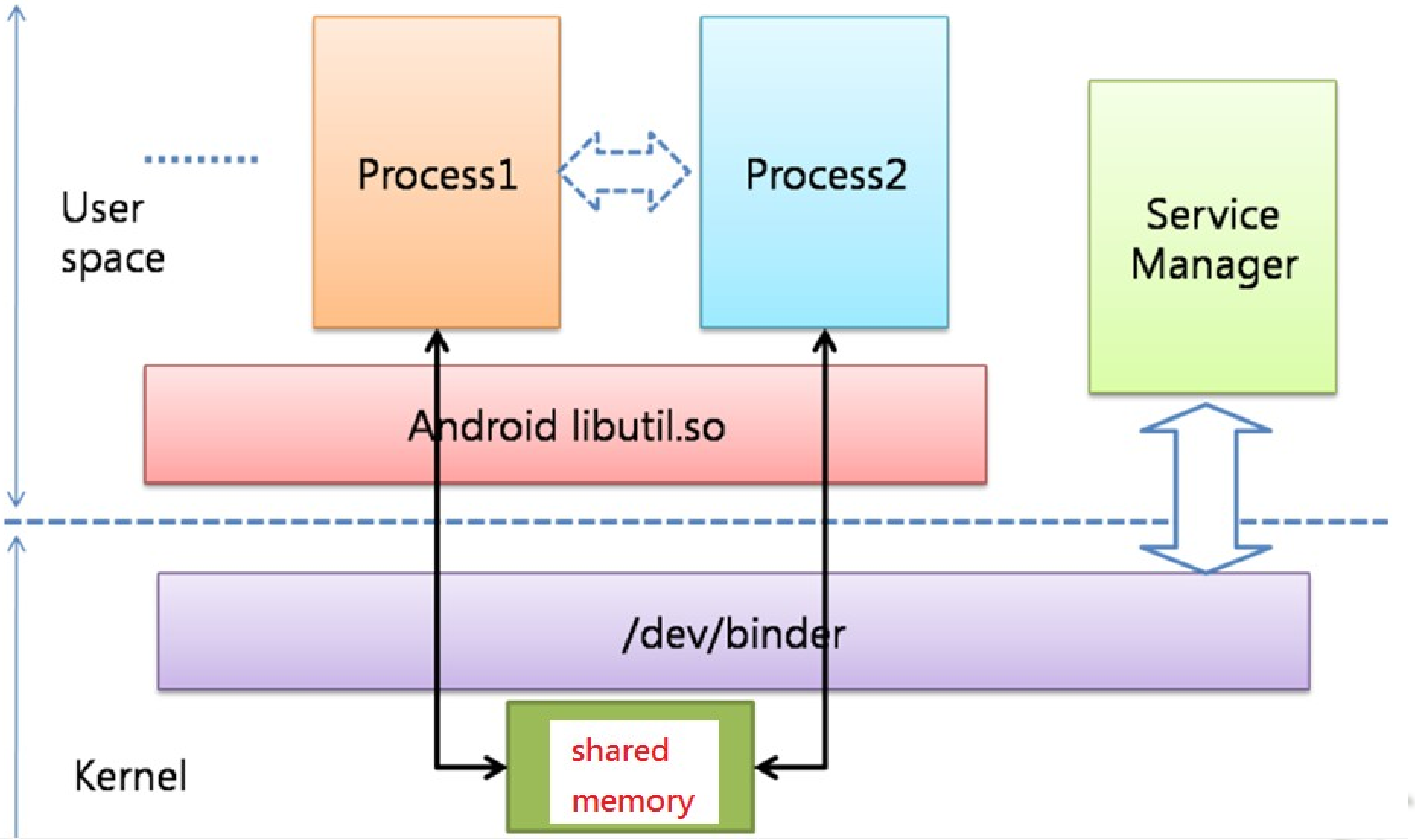


Binder is different from UNIX socket

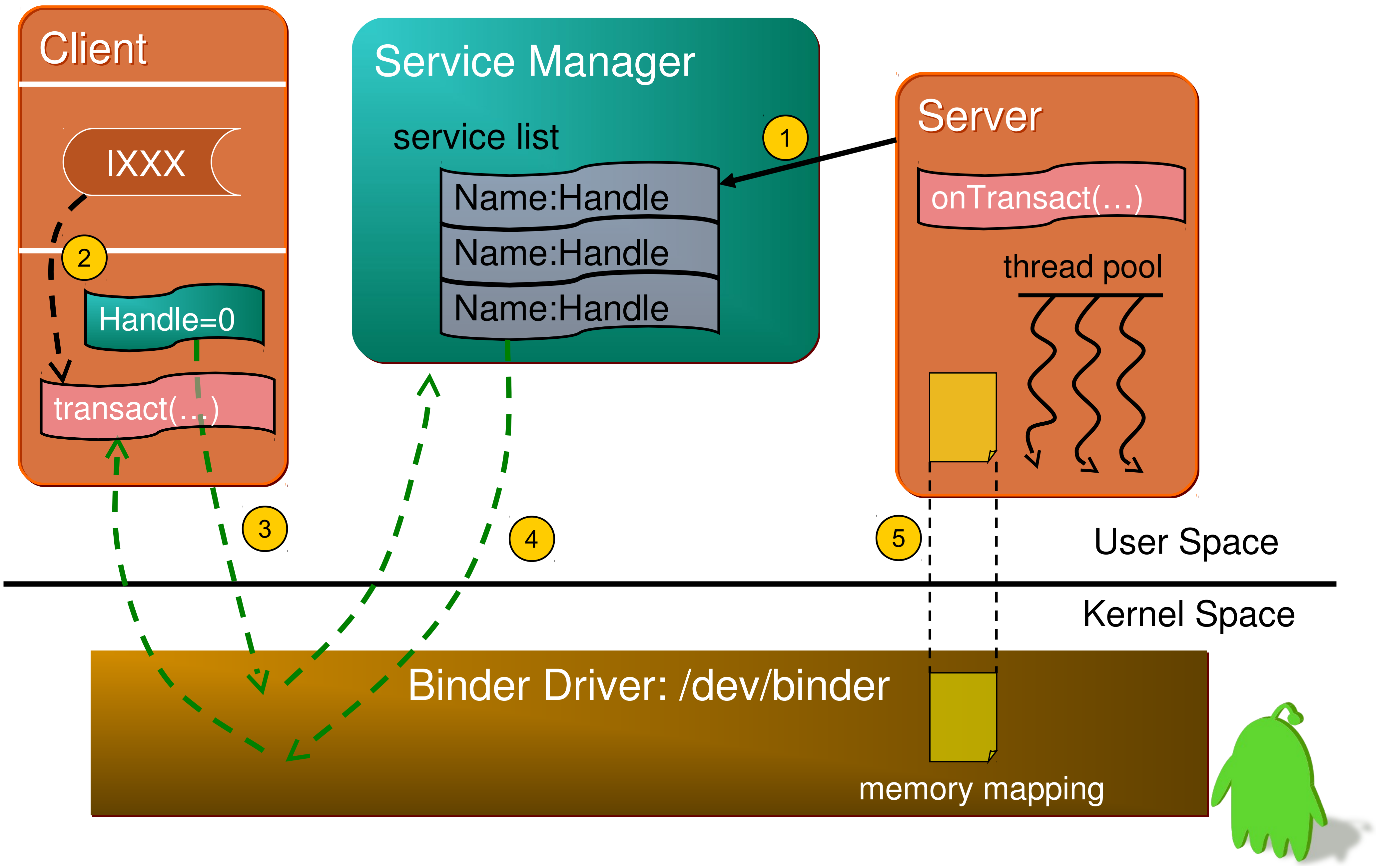
	socket	binder
internal status	associated to FD	associated to PID (FD can be shared among threads in the same process)
read & write operation	stream I/O	done at once by **ioctl**
network transparency	Yes	No expected local only

Binder

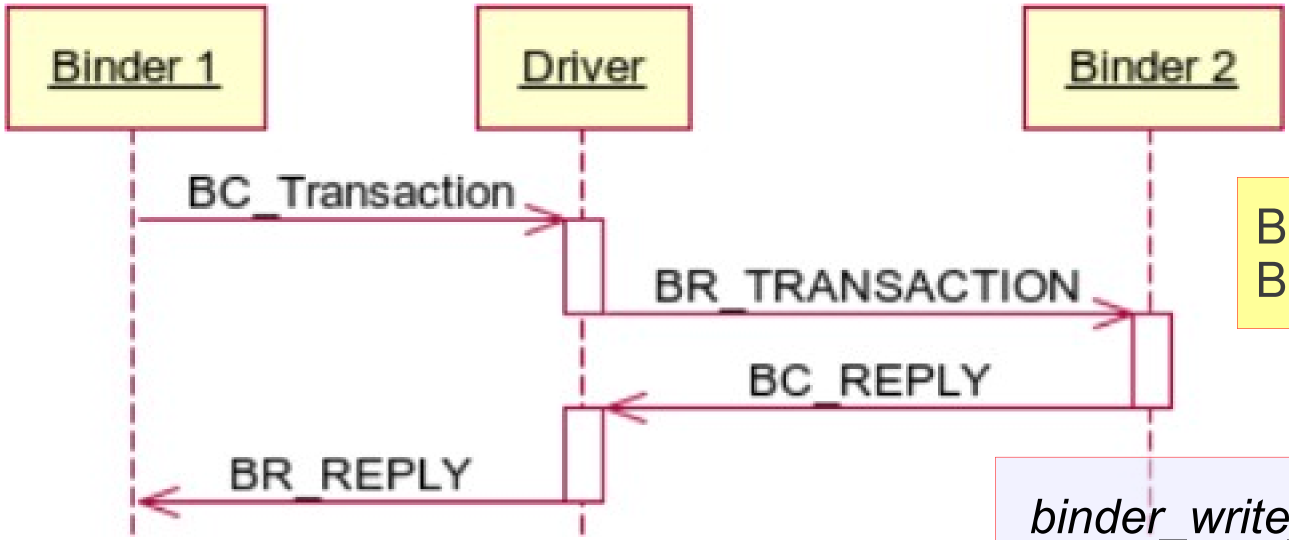
```
$ adb cat /sys/devices/virtual/misc/binder/uevent
MAJOR=10
MINOR=47
DEVNAME=binder
```



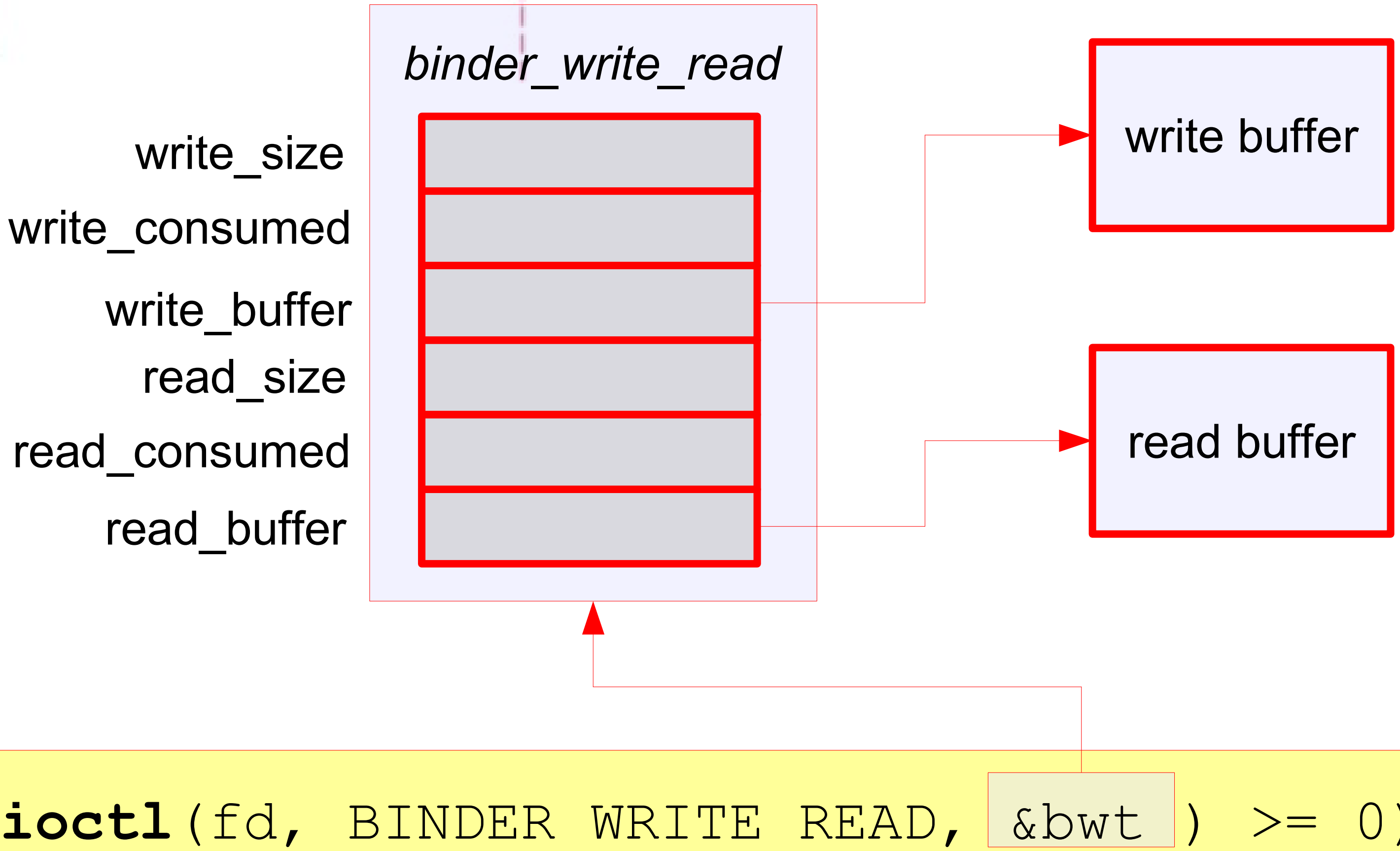
from SM to Binder Driver



Transaction



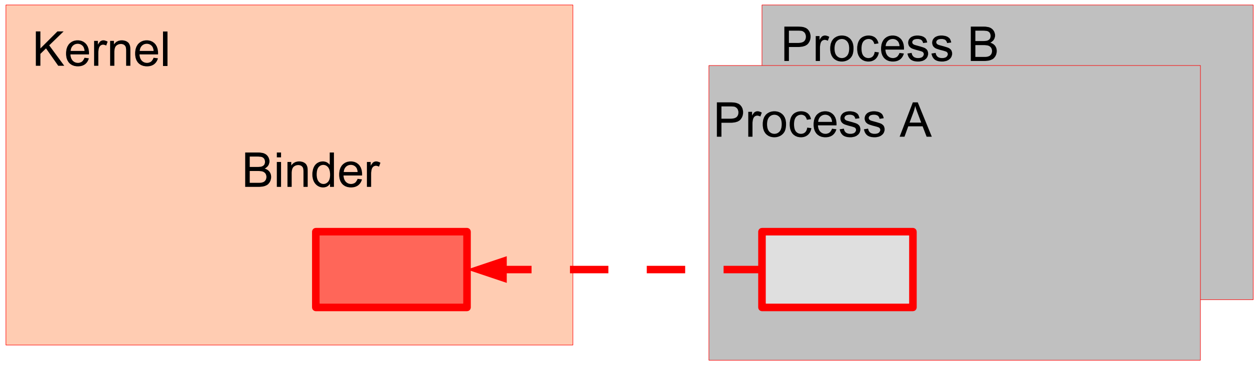
BR → BinderDriverReturnProtocol
BC → BinderDriverCommandProtocol



```
if (ioctl(fd, BINDER_WRITE_READ, &bwt ) >= 0)
    err = NO_ERROR;
else
    err = -errno;
```

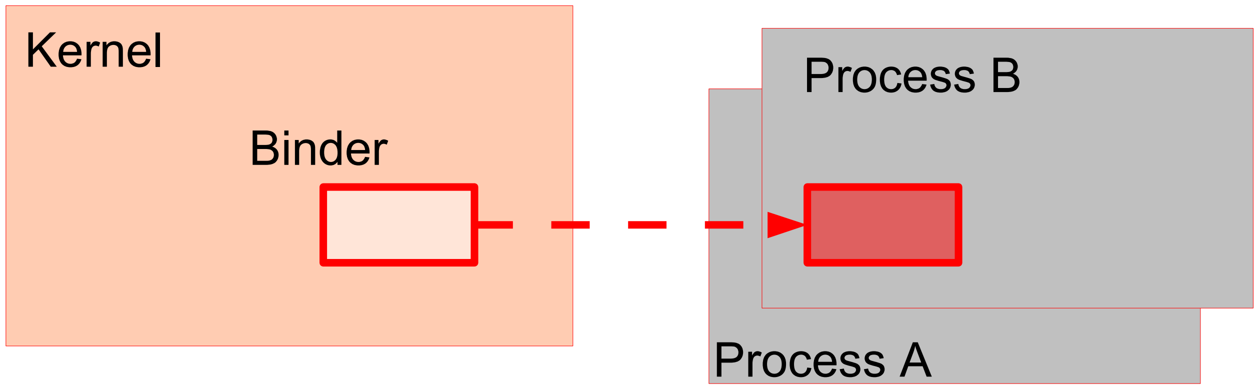
Transaction of Binder

Process A and B have different memory space.
They can not see each other.



Copy memory by **copy_from _user**

Then, wake up process B



Copy memory by **copy_to_user**

Internally, Android uses Binder for graphics data transaction across processes.
It is fairly efficient.

Limitation of Binder IPC

- Binders are used to to communicate over process boundaries since different processes don't share a common VM context
  - no more direct access to each others Objects (memory).
- Binders are not ideal for transferring large data streams (like audio/video) since every object has to be converted to (and back from) a Parcel.

Binder Performance

- Good
  - Compact method index
  - Native binary marshalling
  - Support of ashmem shortcut
  - No GUID
- Bad
  - Dalvik Parcel overhead
  - ioctl() path is not optimal
  - Interface name overhead
  - Global lock

Binder Security

- Binder's Security Features
  - Securely Determined Client Identity
    - Binder.getCallingUid(), Binder.getCallingPid()
    - Similar to Unix Domain Socket

      `getsockopt(..., SO_PEERCRED, ...)`
  - Interface Reference Security
    - Client cannot guess Interface Reference
- Service Manager
  - Directory Service for System Services
- Server should check client permission

  `Context.checkPermission(permission, pid, uid)`

Binder sample program

- Build binder benchmark program

```
cd system/extras/tests/binder/benchmarks
```

```
mm
```

```
adb push \
  ../../../../out/target/product/crespo/data/nativebenchmark/binderAddInts \
   /data/local/
```

- Execute

```
adb shell
```

```
su
```

```
/data/local/binderAddInts -d 5 -n 5 &
```

```
ps
```

```
...
root      17133 16754 4568   860   ffffffff 400e6284 S
/data/local/binderAddInts
root      17135 17133 2520   616   00000000 400e5cb0 R
/data/local/binderAddInts
```

Binder sample program

- Execute

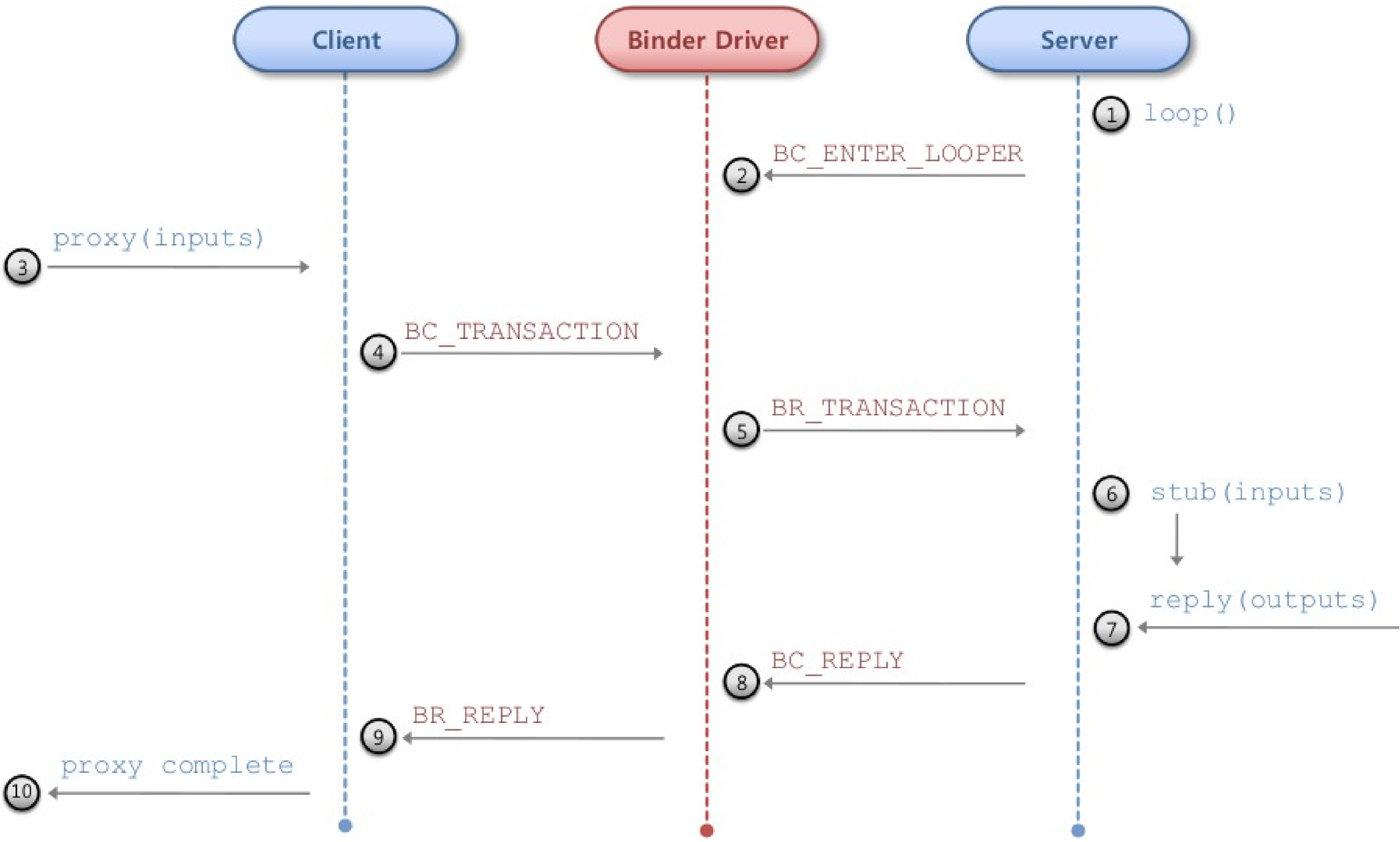
```
/data/local/binderAddInts -d 5 -n 5 &
ps
...
root      17133 16754 4568   860   ffffffff 400e6284 S
/data/local/binderAddInts
root      17135 17133 2520   616   00000000 400e5cb0 R
/data/local/binderAddInts
cat /sys/kernel/debug/binder/transaction_log
transaction_log:3439847: call  from 17133:17133 to 72:0 node
1 handle 0 size 124:4
transaction_log:3439850: reply from 72:72 to 17133:17133 node
0 handle 0 size 4:0
transaction_log:3439855: call  from 17135:17135 to 17133:0
node 3439848 handle 1 size 8:0
...
```

Binder sysfs entries

- `adb shell ls /sys/kernel/debug/binder`

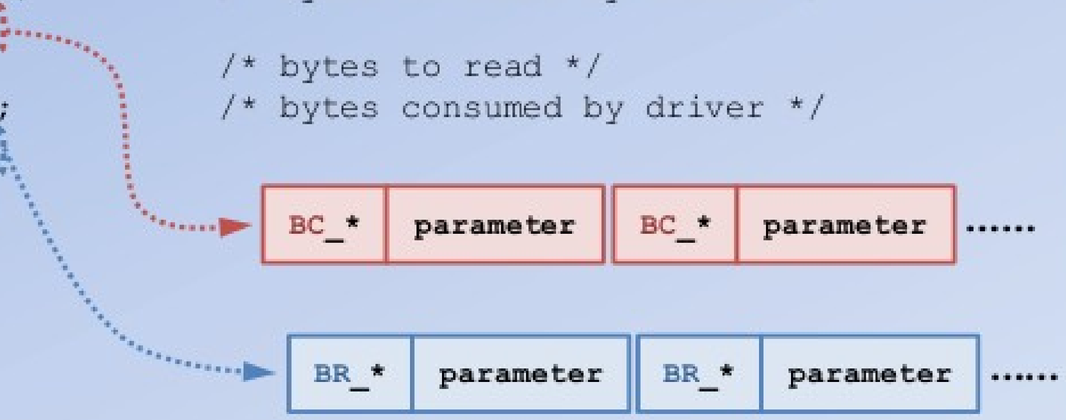
```
failed_transaction_log
proc
state
stats
transaction_log
transactions
```

Remote Procedure Call



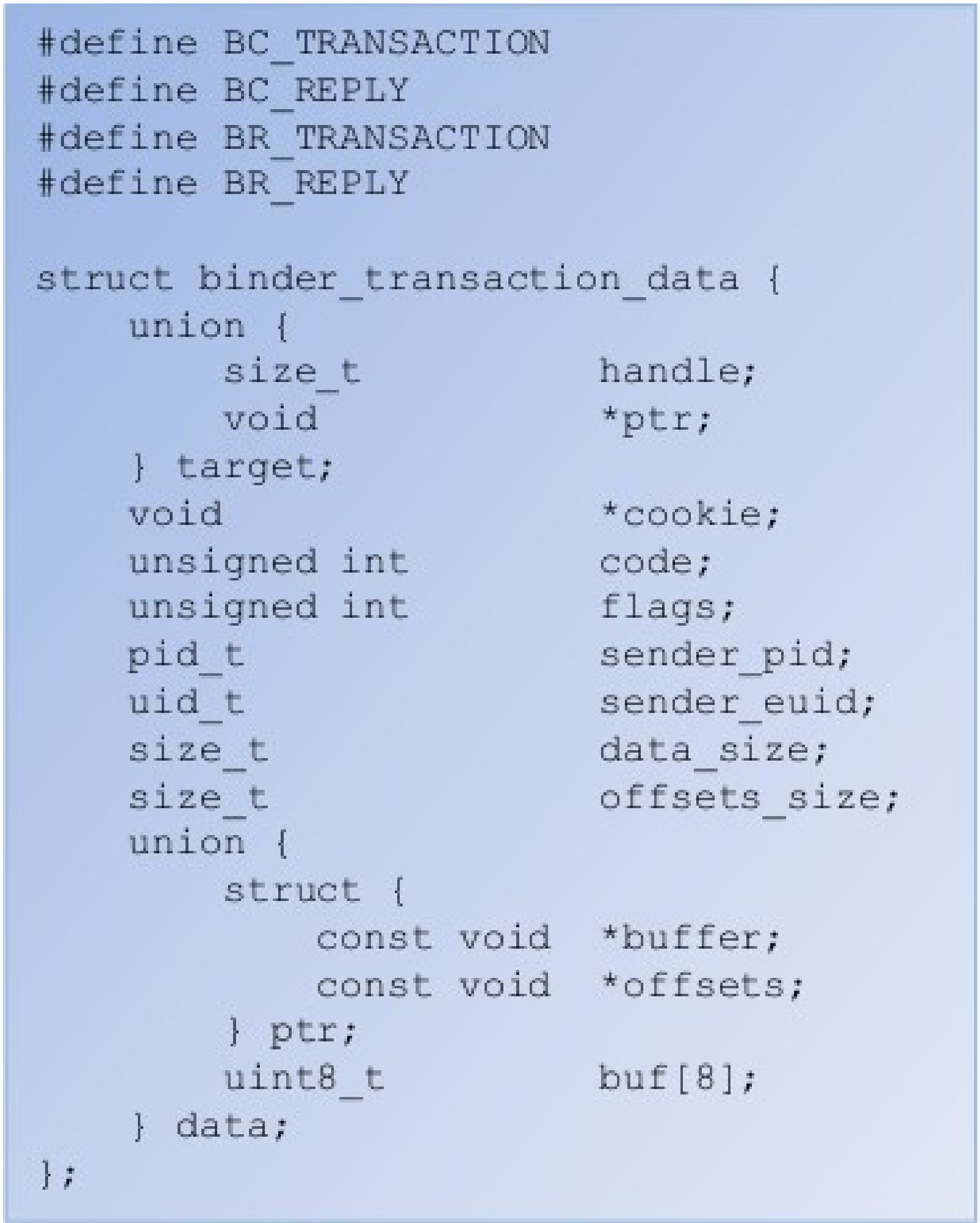
BINDER_WRITE_READ

```
struct binder_write_read {
            long                    write_size;          /* bytes to write */
            long                    write_consumed;      /* bytes consumed by driver */
            unsigned long           write_buffer;
            long                    read_size;           /* bytes to read */
            long                    read_consumed;       /* bytes consumed by driver */
            unsigned long           read_buffer;
};

#include <sys/ioctl.h>
#include <linux/binder.h>

int binder_write(int fd, void *data, long len) {
            struct binder_write_read bwr;

            bwr.write_size = len;
            bwr.write_consumed = 0;
            bwr.write_buffer = (unsigned) data;
            bwr.read_size = 0;
            bwr.read_consumed = 0;
            bwr.read_buffer = 0;
            return ioctl(fd, BINDER_WRITE_READ, &bwr);
}
```



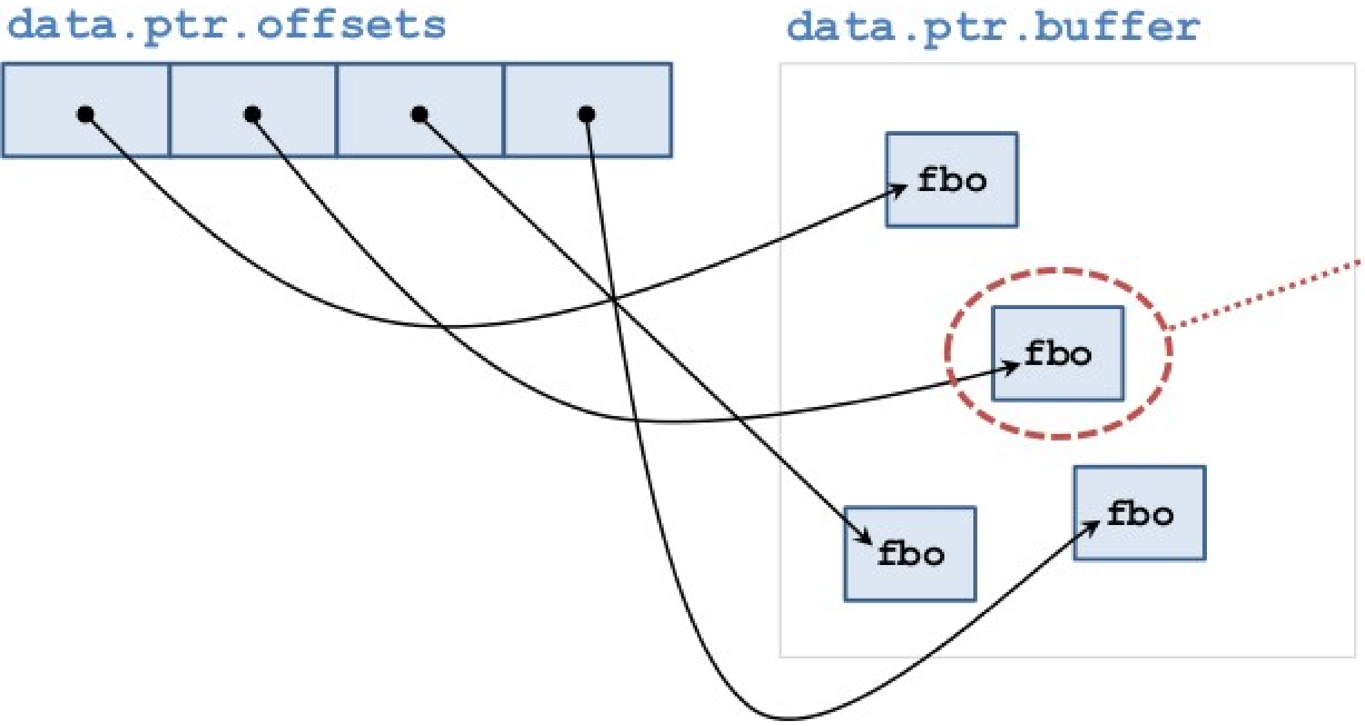
Binder Transaction

- Target Method
  - handle : Remote Interface
  - ptr & cookie : Local Interface
  - code : Method ID
- Parcel - Input/Output Parameters
  - data.ptr.buffer
  - data_size
- Object Reference Management
  - data.ptr.offsets
  - offsets_size
- Security
  - sender_pid
  - sender_euid
- No Transaction GUID
  - Transparent Recursion



```
#define BC_TRANSACTION
#define BC_REPLY
#define BR_TRANSACTION
#define BR_REPLY

struct binder_transaction_data {
    union {
        size_t              handle;
        void                *ptr;
    } target;
    void                    *cookie;
    unsigned int            code;
    unsigned int            flags;
    pid_t                   sender_pid;
    uid_t                   sender_euid;
    size_t                  data_size;
    size_t                  offsets_size;
    union {
        struct {
            const void      *buffer;
            const void      *offsets;
        } ptr;
        uint8_t             buf[8];
    } data;
};
```

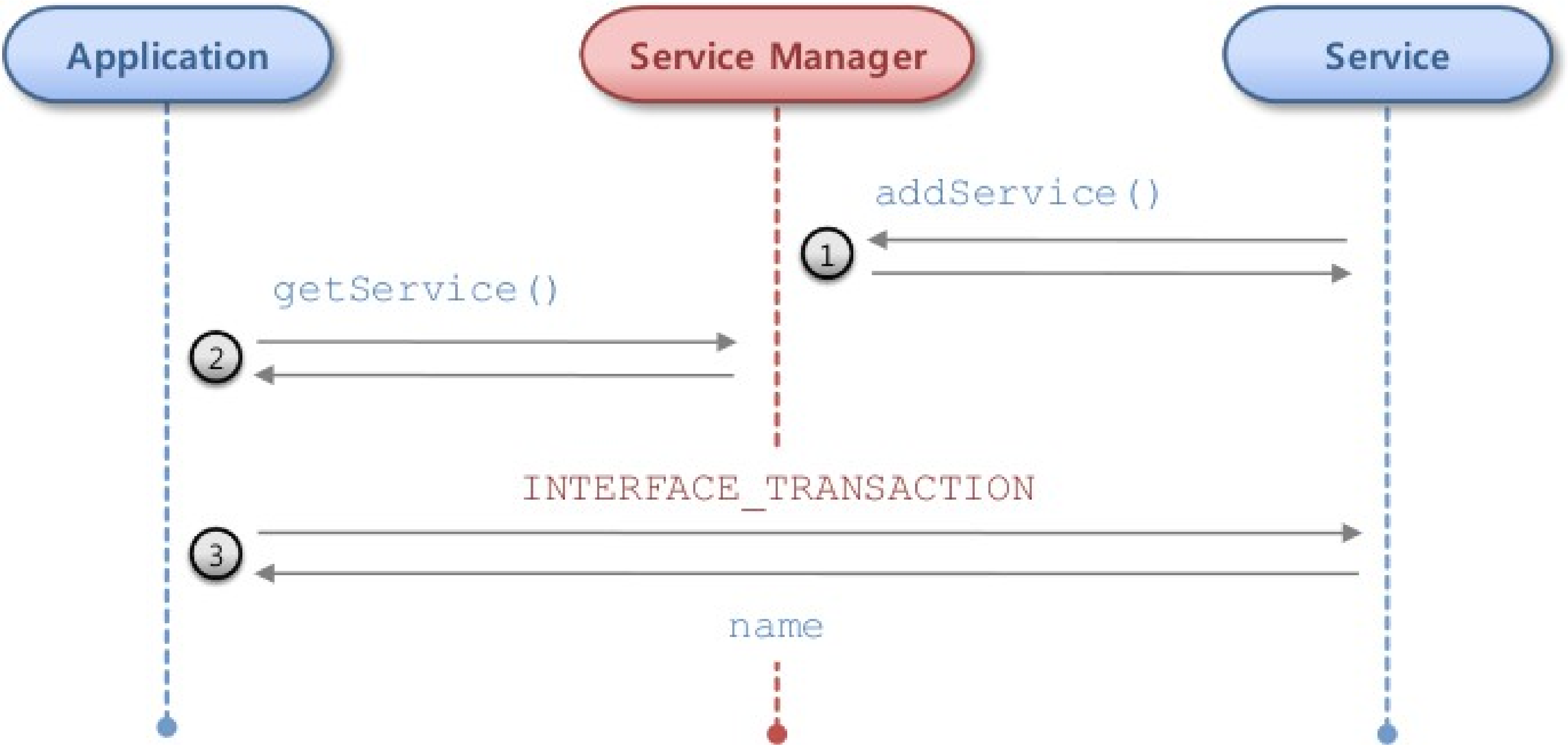
Object Reference Management



```
struct flat_binder_object {
    unsigned long   type;
    unsigned long   flags;
    union {
        void            *binder;
        signed long handle;
    };
    void            *cookie;
};
```

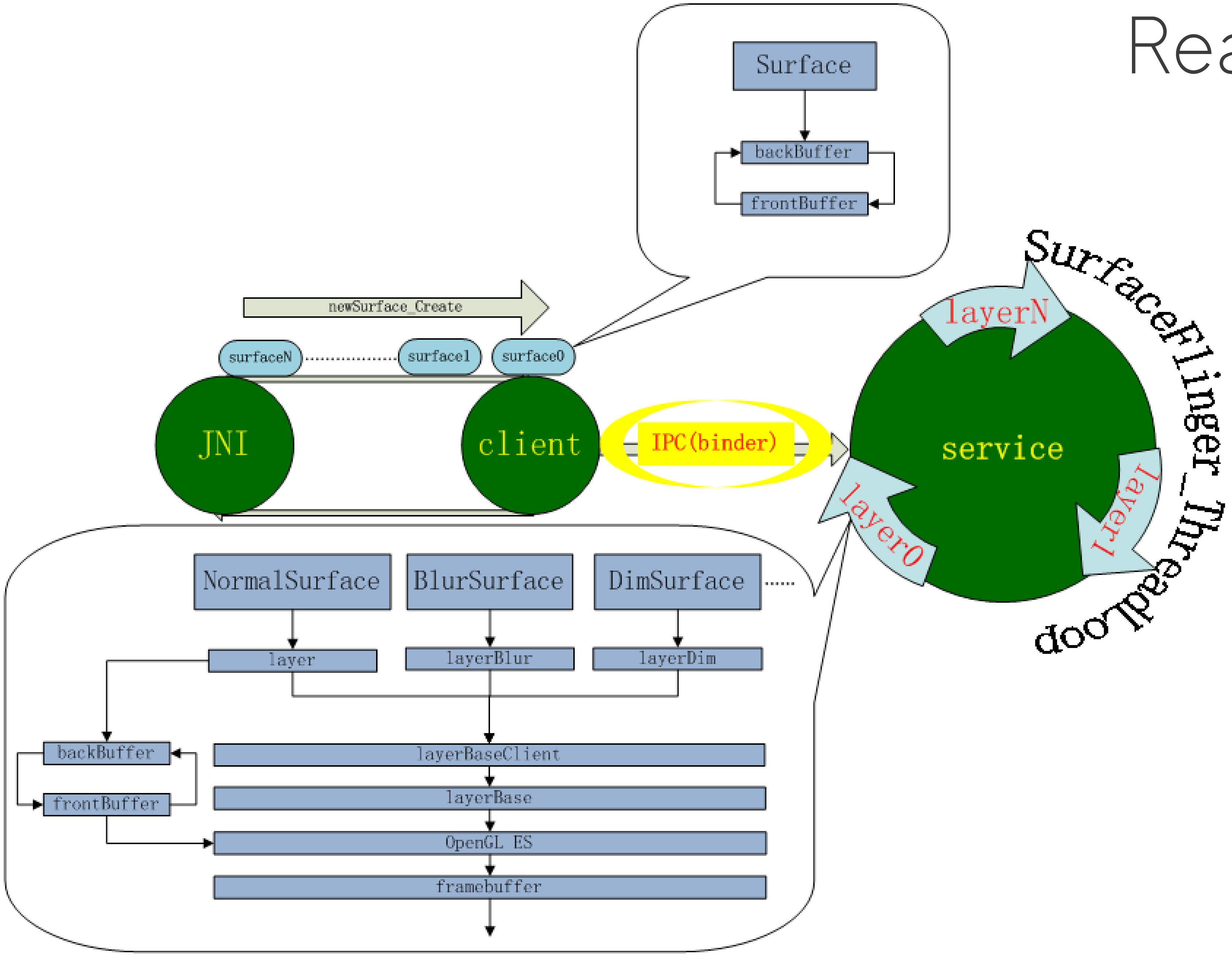
Service Registration and Discovery

- System service is executed by `IServiceManager::addService()` calls.
  - Parameter: handle to Binder Driver
- Look up the name of specific service in Binder Driver Map
  - `IServiceManager::getService()` returns the handle of the found registered services
- `android.os.IBinder.INTERFACE_TRANSACTION`: the actual name



Binder use case: Android Graphics

Real Case



Binder IPC is used for communicating between Graphics client and server.
Taken from http://www.cnblogs.com/xl19862005/archive/2011/11/17/2215363.html

Surface

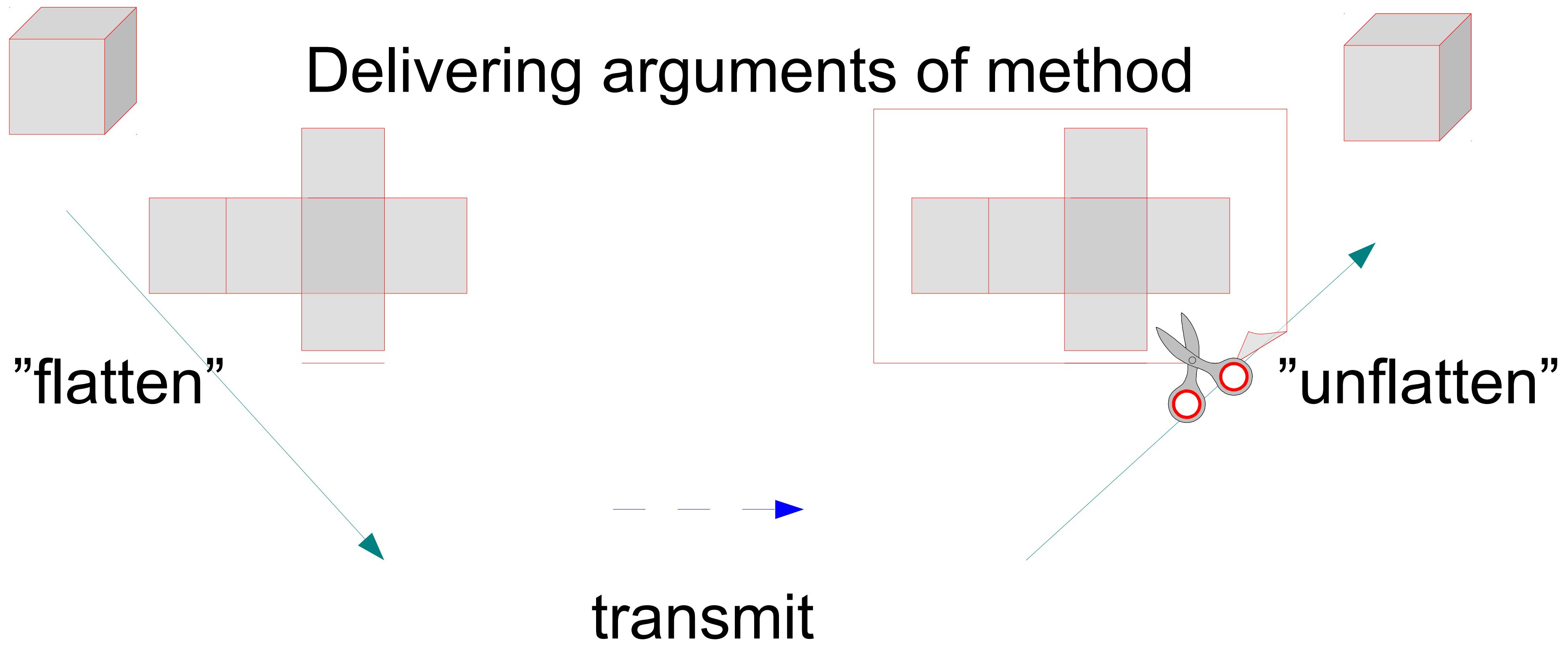
Source: frameworks/base/core/java/android/view/Surface.java

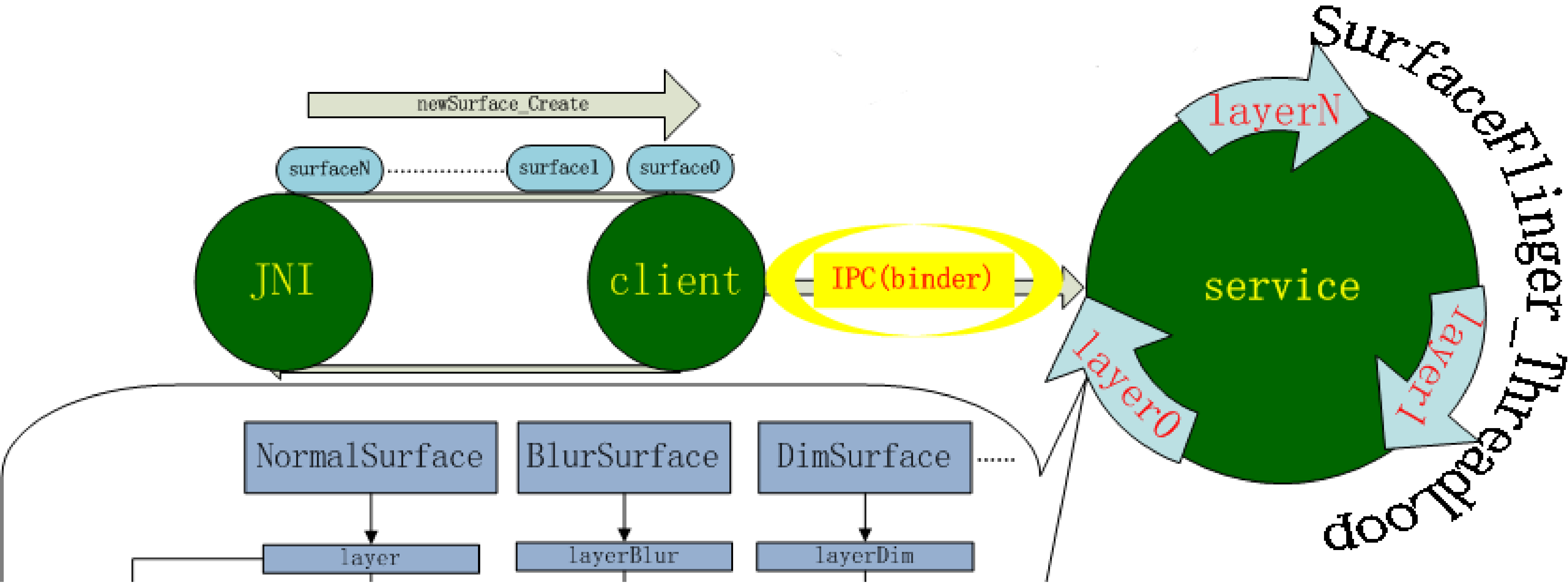
- /* **Handle on to a raw buffer that is being managed by the screen compositor** */

```
public class Surface implements Parcelable {
    public Surface() {
        mCanvas = new CompatibleCanvas();
    }
    private class CompatibleCanvas
                extends Canvas { /* ... */ }
}
```

Surface instances can be written to and restored from a Parcel.

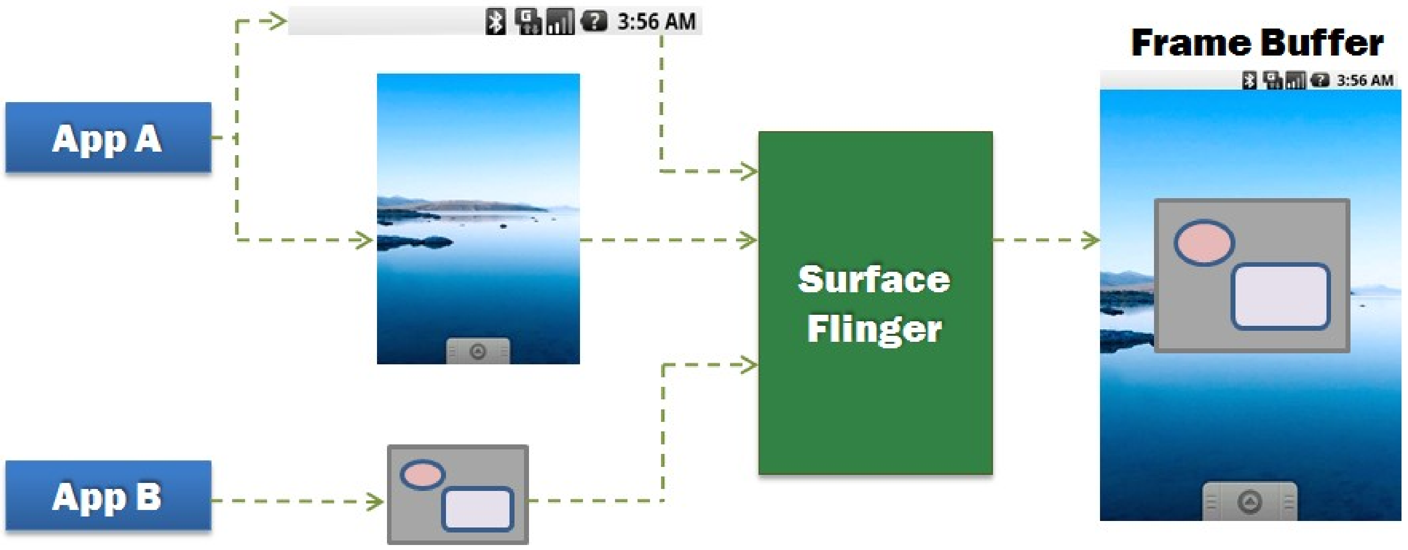
Delivering arguments of method

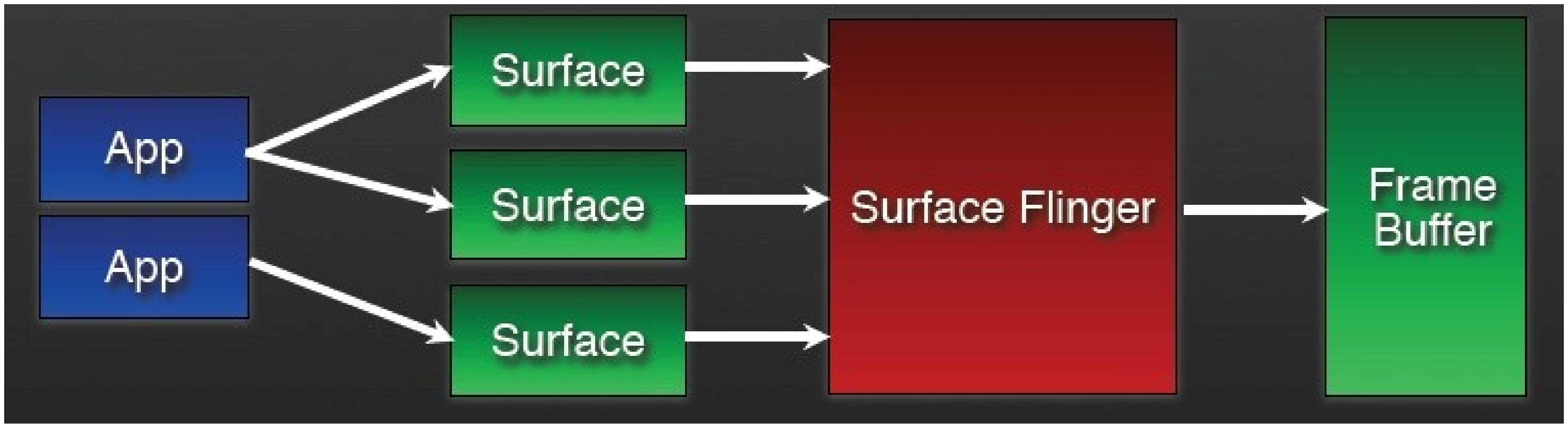


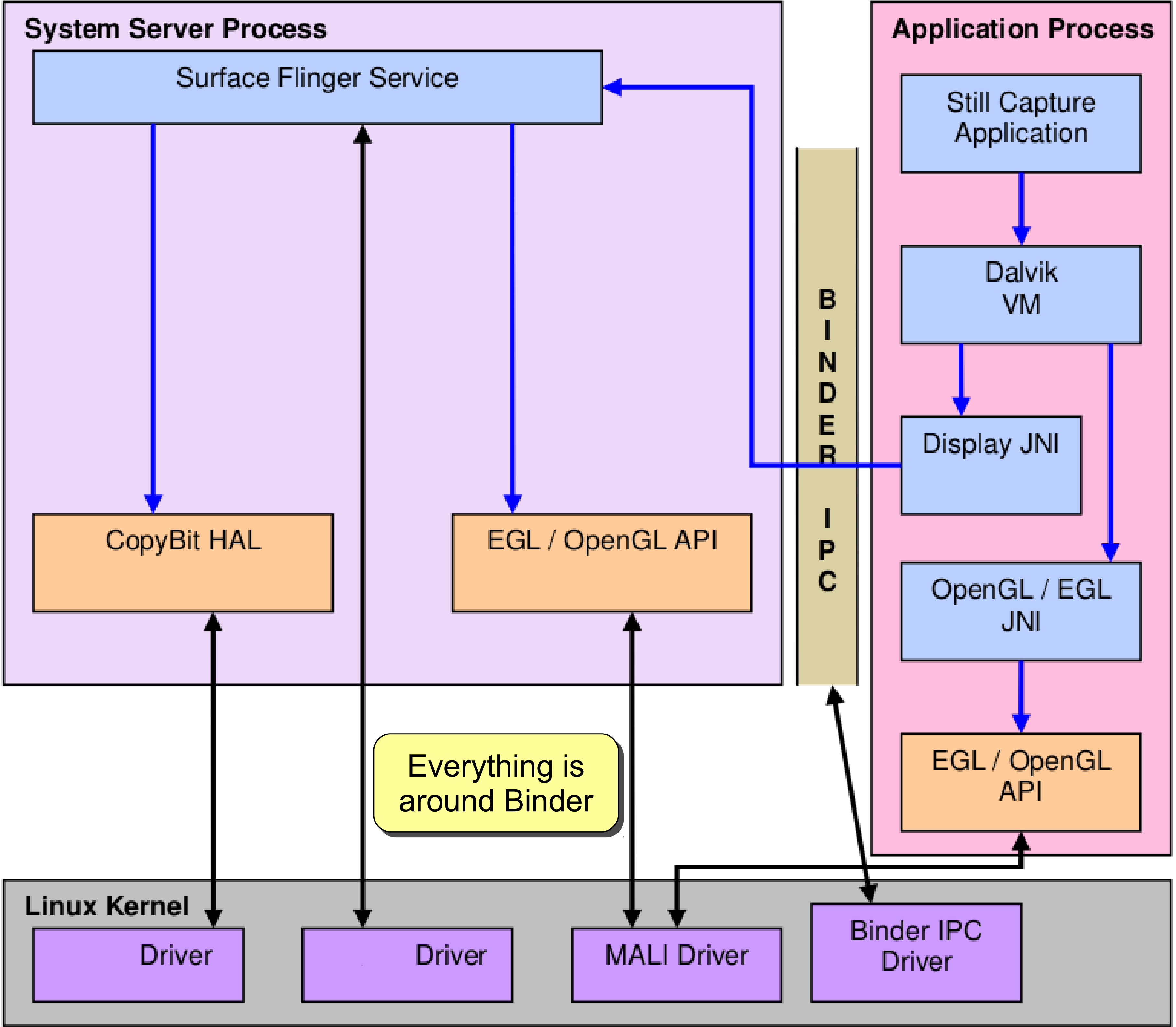


Android SurfaceFlinger

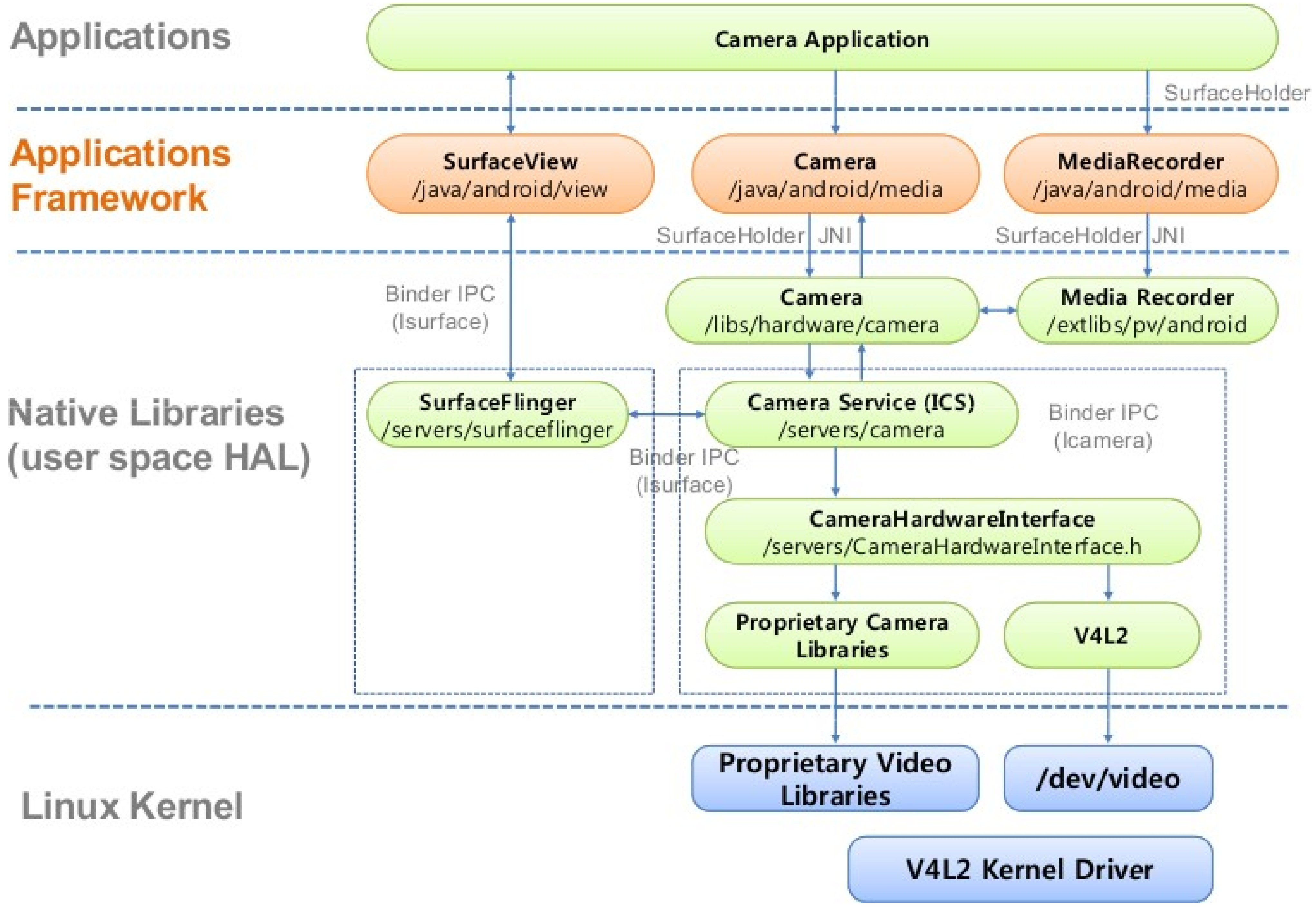
- Properties
  - Can combine 2D/3D surfaces and surfaces from multiple applications
  - Surfaces passed as buffers via Binder IPC calls
  - Can use OpenGL ES and 2D hardware accelerator for its compositions
    - Double-buffering using page-flip






System Server Process
Surface Flinger Service
CopyBit HAL
EGL / OpenGL API
BINDER IPC
Application Process
Still Capture Application
Dalvik VM
Display JNI
OpenGL / EGL JNI
EGL / OpenGL API
Everything is around Binder
Linux Kernel
Driver
Driver
MALI Driver
Binder IPC Driver

Camera + SurfaceFlinger + Binder


Applications
Camera Application
SurfaceHolder
Applications Framework
SurfaceView /java/android/view
Camera /java/android/media
MediaRecorder /java/android/media
SurfaceHolder
JNI
SurfaceHolder
JNI
Binder IPC (Isurface)
Camera /libs/hardware/camera
Media Recorder /extlibs/pv/android
Native Libraries (user space HAL)
SurfaceFlinger /servers/surfaceflinger
Camera Service (ICS) /servers/camera
Binder IPC (Icamera)
Binder IPC (Isurface)
CameraHardwareInterface /servers/CameraHardwareInterface.h
Proprietary Camera Libraries
V4L2
Linux Kernel
Proprietary Video Libraries
/dev/video
V4L2 Kernel Driver

Reference

- Inter-process communication of Android, Tetsuyuki Kobayashi
- 淺談 Android 系統進程間通信（ IPC ）機制 Binder 中的 Server 和 Client 獲得 Service Manager 接口之路

  http://blog.goggb.com/?post=1580
- Android Binder – Android Interprocess Communication, Thorsten Schreiber
- Design Patterns in the Android Framework, Prof. Sheng-De Wang

0xlab

http://0xlab.org