Android Binder IPC

Chethan Palakshamurthy

Outline

- What is Binder IPC?
- High level design
- Communication between participants
- Low level design
- Creation of proxy and native binders

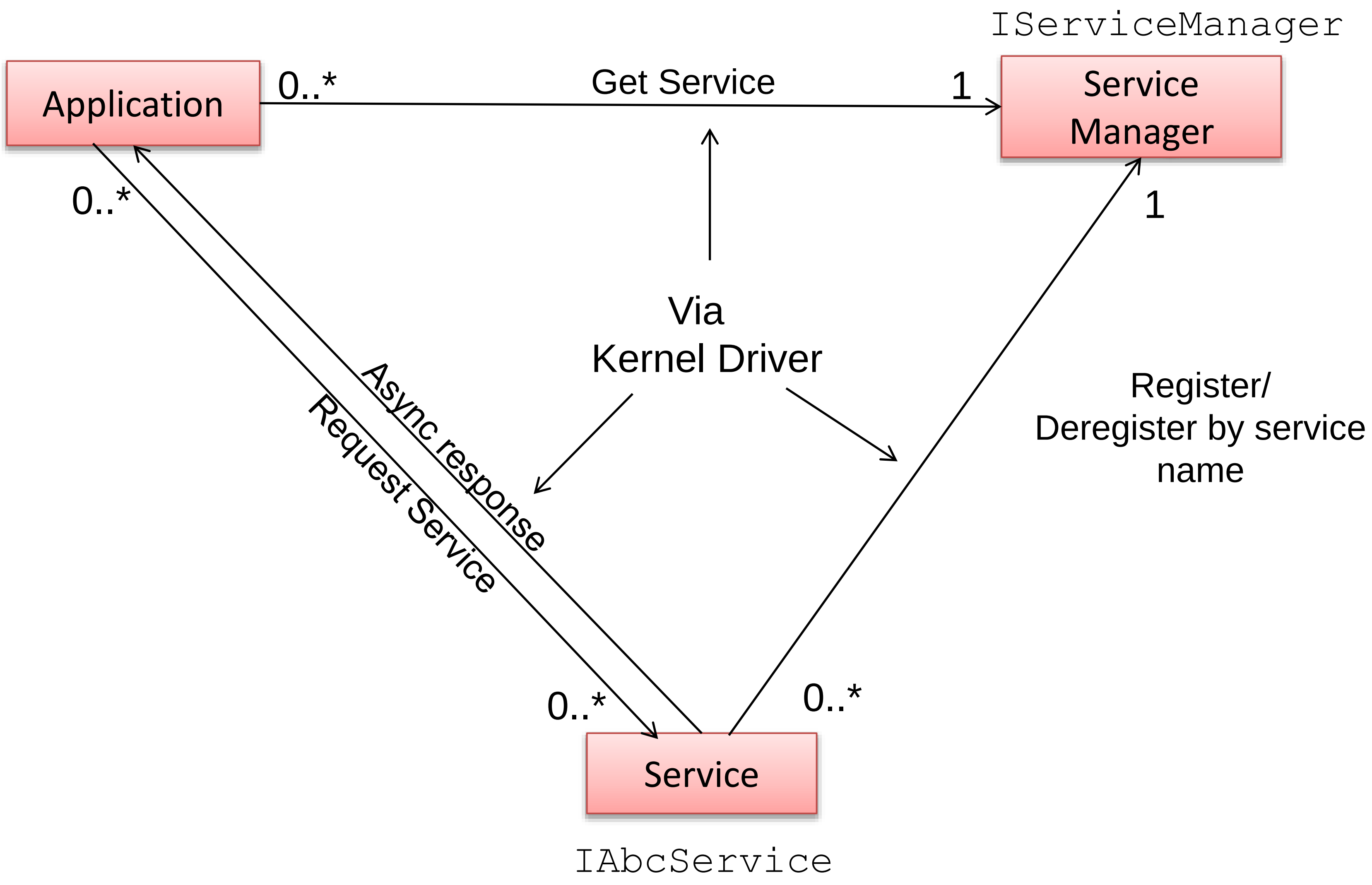
Index

Binder IPC

- The features of Binder are comparable to functionality provided by any mature traditional client/server architecture or IPC mechanisms.
  - Symbian IPC, Linux D-Bus are couple of the examples.
- Binder takes a different approach with the constructs used, to better support Android Interface Definition Language (AIDL) and its implementation
- The main feature of Binder is that, instead of sharing enumerated command/request ids, the client and server sides share a common abstract service interface
  - There exist two objects which implement the same interface. (1) Local proxy – for use by application in the same process and (2) Remote service object – which has the actual service implementation, resides in service's process
  - Invoking an API on the local proxy object, translates to a call on the remote object

Index

Binder IPC – High level design



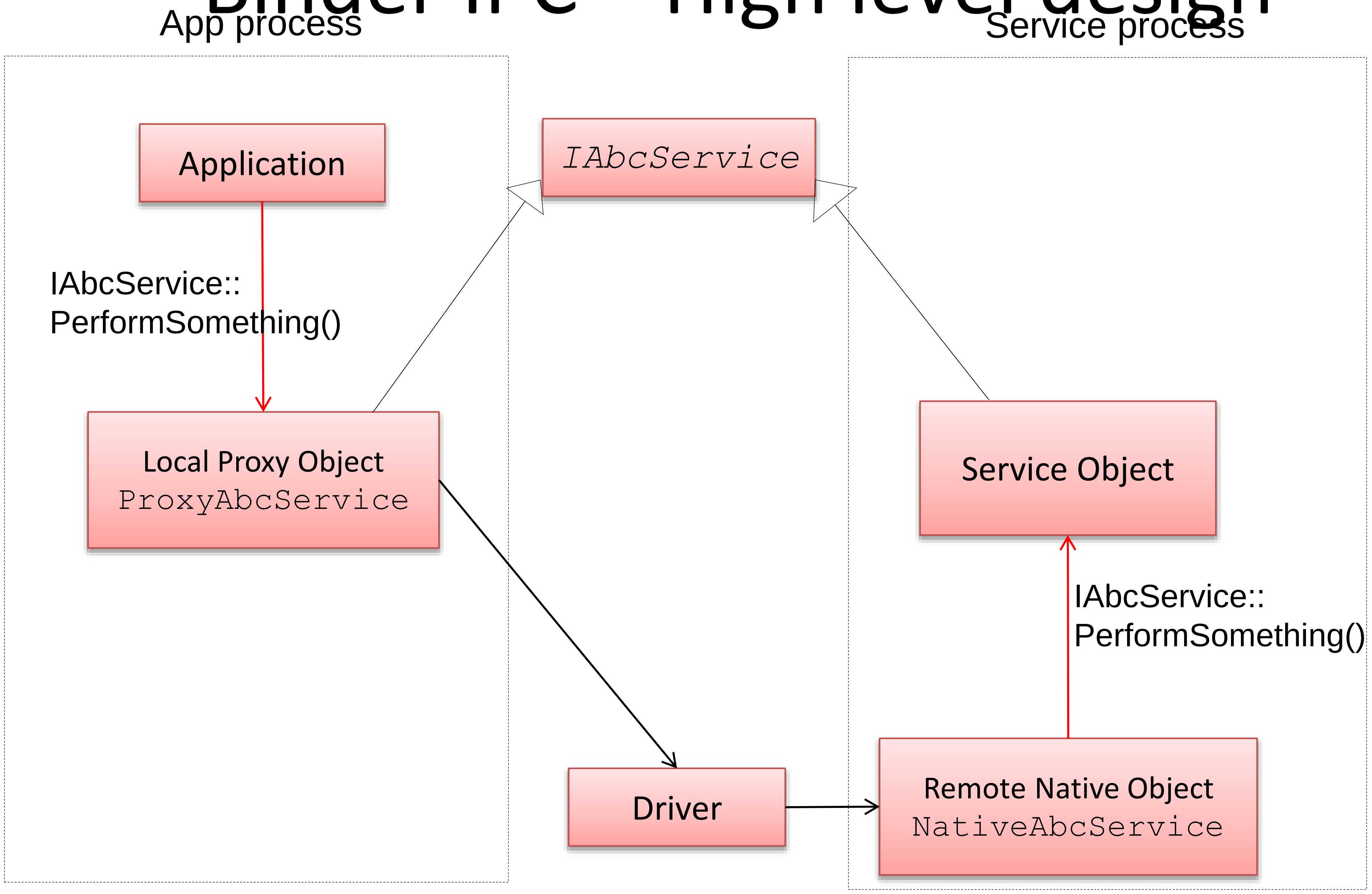
Binder IPC – High level view

- Binder framework uses a kernel driver for IPC - /dev/binder
- Clients to the driver are
  - App (Service user)
  - Services
  - Service Manager (Also a service – a special one)
- Driver assigns and maintains IDs or handles (and much more info) of each.
- Service manager (Id = 0)
  - Registers itself with binder driver, as ‘Manager’ on device startup
  - Manages a list of services.
- Services
  - Services register themselves with SVC manager on service startup
  - Provide an abstract service interface

Binder IPC – Using a service

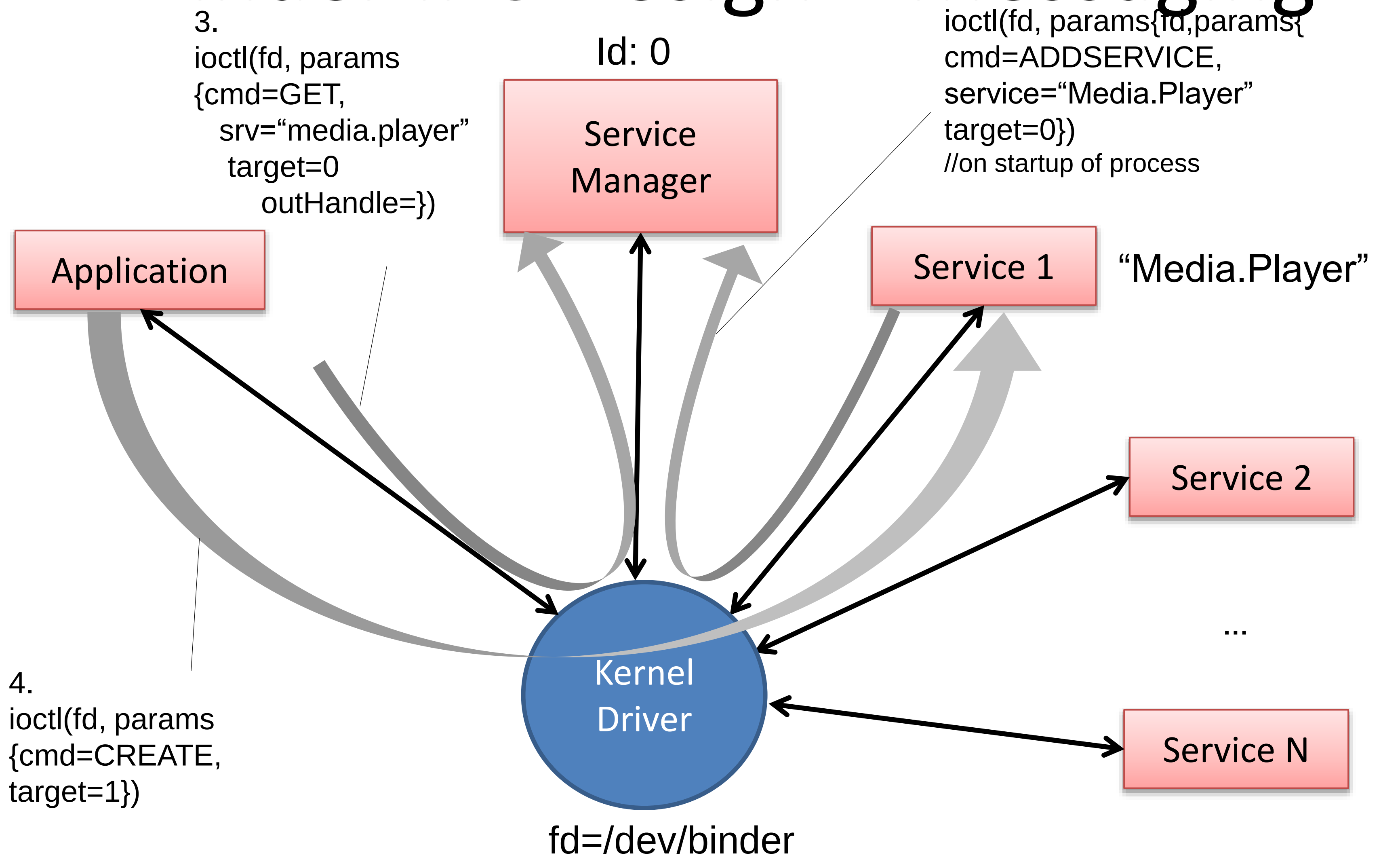
- First, application gets IServiceManager handle.
  - Using the globally known identifier – 0.
  - There are helper functions to get this object
- App invokes IServiceManager::GetService to get a handle of IAbcService for a service “Abc”
  - IServiceManager object is implemented by framework and is part of binder library
- Invokes IAbcService::PerformSomething call
  - The call gets translated to PerformSomething call on the service object
  - Service provider needs to implement the IAbcService

Binder IPC – High level design



Index

Binder IPC Design – Messaging



Binder IPC Design - Messaging

1. Service manager opens ‘/dev/binder’ and registers itself (handle = 0) as manager using ioctl
2. Media Player Service, on process startup, creates an object instance (`MediaPlayerService`) and registers it (instance as handle, say 0x70FF) along with a name, with SVC Mgr.
   – By calling ioctl with target handle = 0, in parameter
   – Driver knows ‘0’. It directs it to SVC Mgr.
   – Seeing ADD_SERVICE in param, SVC Mgr, registers the service along with provided handle.
   – Now, SVC manager knows “Media.Player”. Driver knows media player service handle – 0x70FF.
3. Application asks SVC Mgr for “Media.Player” service
   – By calling ioctl with target handle = 0, cmd=“GET_SERVICE”, name=“Media.Player”
   – SVC Mgr returns the handle associated with “Media.Player”, in ioctl out params.

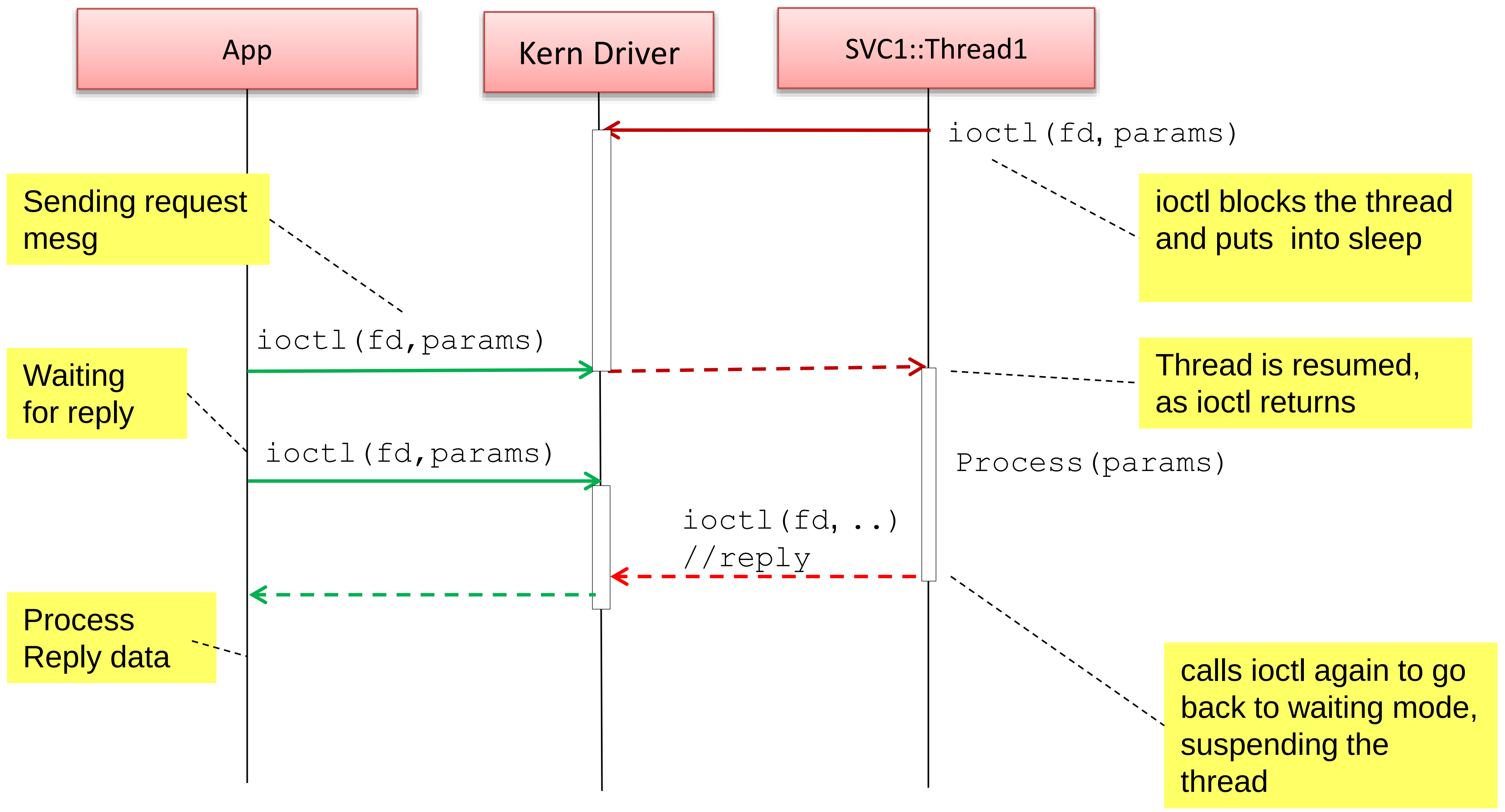
Binder IPC Design - Messaging

4. Application asks the service to create one instance of media player. (Media Player Service supports multiple player instances)
   - By calling ioctl with target handle = ‘0x70FF’ (say)
   - Media Player Service on seeing command ‘CREATE’ creates a player instance and embeds the instance handle in the reply.

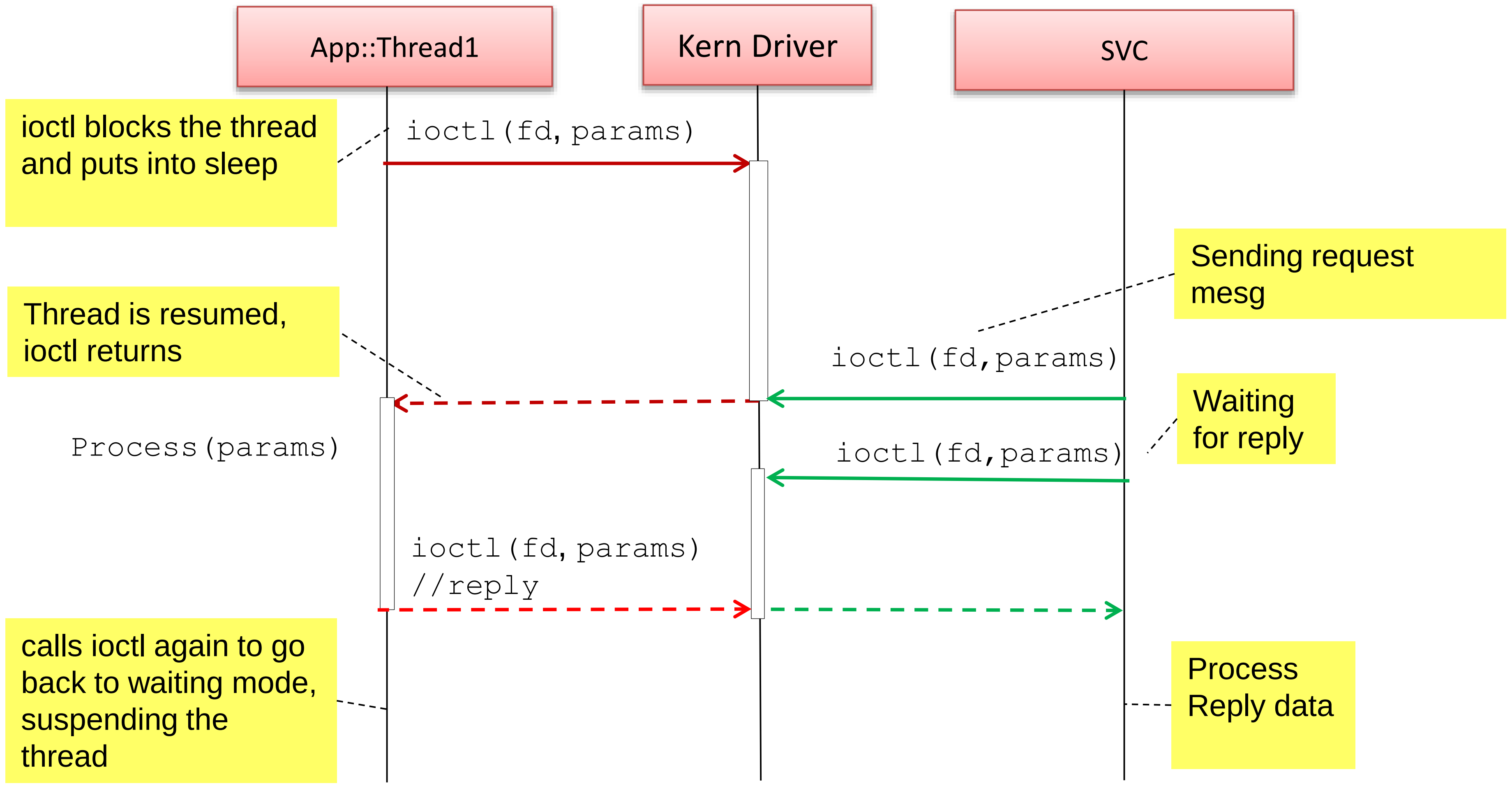
Binder IPC Design – Send/Receive Impl.

- Each client of the driver has 1 or more threads.
- A thread on the server waits on a loop on an ioctl waiting for a service request.
- The driver puts the thread to sleep using wait_event_interruptible.
- When an app calls ioctl on its end targeting a service, the driver wakes up a thread of that service
- ioctl on service end, comes out of the wait, services the request
- Now, if it’s a sync request, app makes another ioctl call waiting for reply.
- The services sends a reply parcel back by calling ioctl, waking up the app; and goes back to sleep with another ioctl call (typically in a loop)
- If the request is Async, service calls ioctl sometime later. But this time, one of the threads waiting with ioctl will pick it up

Binder IPC Design – Send/Receive - Sync



Binder IPC Design – Async call from Service


App::Thread1
Kern Driver
SVC
ioctl blocks the thread and puts into sleep
ioctl(fd, params)
Sending request mesg
Thread is resumed, ioctl returns
ioctl(fd,params)
Waiting for reply
Process(params)
ioctl(fd,params)
ioctl(fd, params) //reply
calls ioctl again to go back to waiting mode, suspending the thread
Process Reply data

Index

Binder IPC – LLD

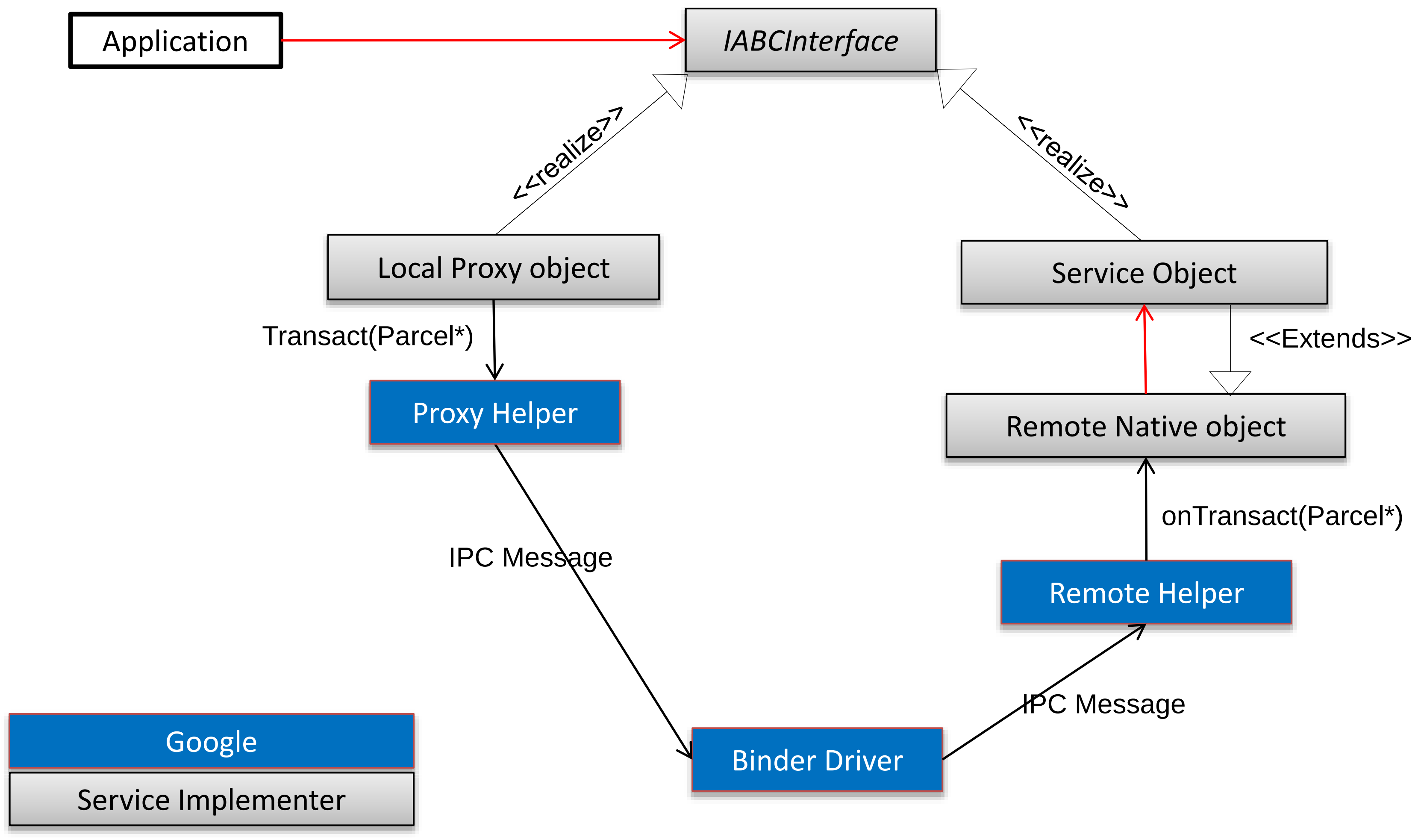
- Application or service do not call ioctl directly.
- There are layers of objects before an application intent gets translated to an ioctl. Some important ones are -

1. Local proxy object → Implements a service specific abstract interface
   - E.g., BpMediaPlayerService (B=binder, p=proxy)
   - Each API implementation creates `Parcels` that encapsulate command/request ID etc.
   - Forwards Parcel to proxy helper.
2. Proxy Helper →
   - Flattens & converts the parcels into ioctl parameter objects and makes the ioctl call.
   - BpBinder, IPCThreadState

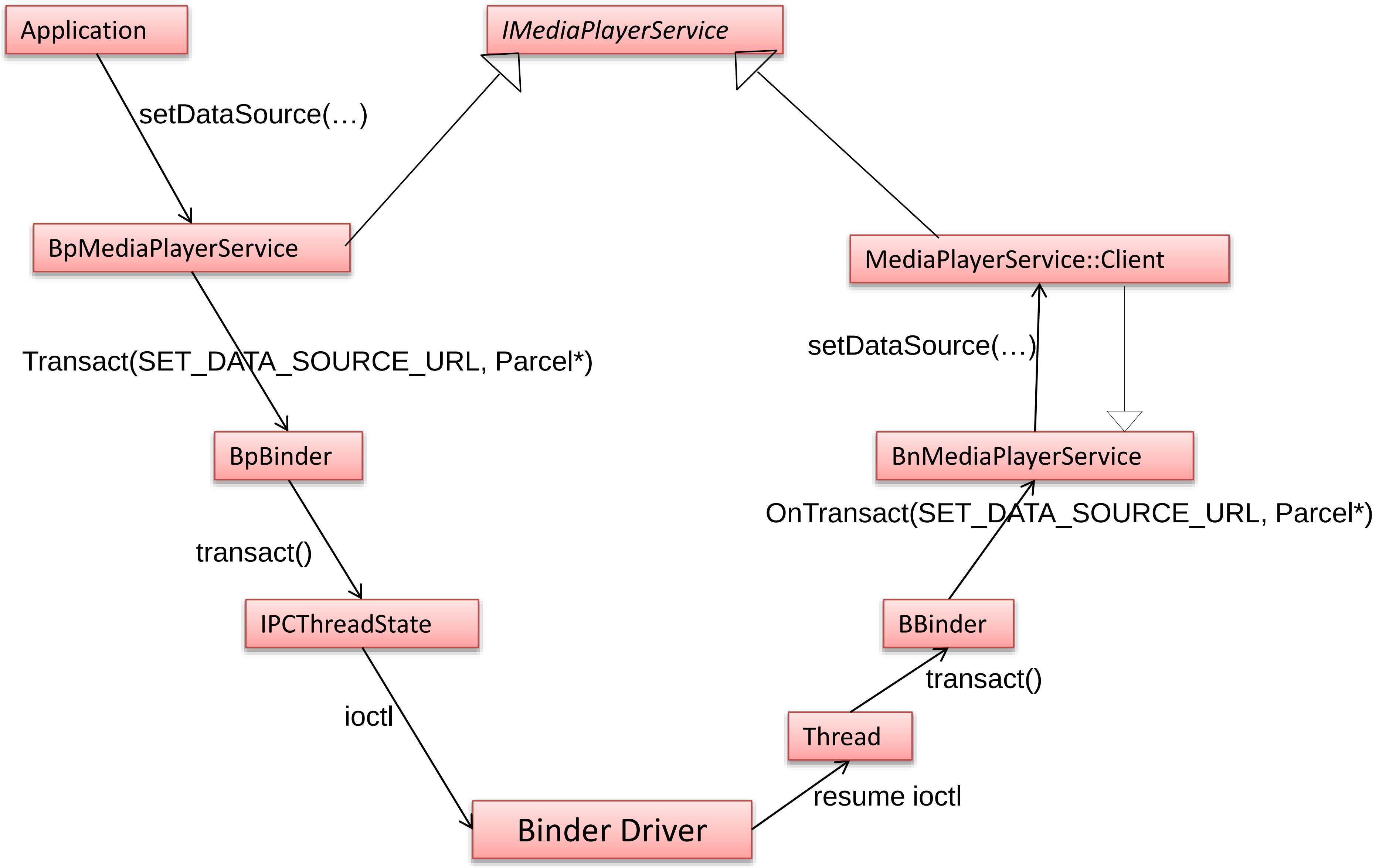
Binder IPC – LLD

3. Remote helper →
   - Receives and unflattens the ioctl parameters
   - Delegates parcel to remote native object.
   - IPCThreadState, BBinder
4. Remote native object
   - Does the exact opposite of local proxy object
   - Receives the parcel and calls the appropriate service object
   - BnMediaPlayerService
5. Service Object
   - Has the ‘real’ implementation of the service
   - E.g., MediaPlayerService : BnMediaPlayerService (B=binder, n=proxy)
   - MediaPlayerService

Binder IPC – LLD



Binder IPC – LLD



Index

Binder IPC – Creation of proxy and native binders

- Getting Service Manager object
  - Use `sp<IServiceManager> defaultServiceManager()` to get handle.
  - This function creates a BpBinder(0) and wraps it with BpServiceManager
  - BpBinder is the helper object which can send IPC to the desired handle. In this case handle = 0.
  - BpServiceManager translates manager calls to IPC using BpBinder object
  - That is, a service proxy object wraps a BpBinder
  - Wrapping is done with interface_cast<>

Binder IPC – Creation of proxy and native binders

- Getting service object
  - App gets a desired service using sp<IBinder> IServiceManager::GetService (“Media.Player”)
  - When GetService calls ioctl, it gets a virtual handle to MediaPlayerService.
  - A BpBinder(handle) is created and wrapped with BpMediaPlayerService
  - Thus sp<BpMediaPlayerService> is obtained for App’s use.

Binder IPC – Creation of proxy and native binders

- Creating a media player instance
  - sp<BpMediaPlayerService>.create(…)
  - create() sends ioctl message to MediaPlayerService instance on Media server process
  - create API is invoked on MediaPlayerService instance.
  - Based on parameters, the service creates a media player instance - BnMediaPlayer.
  - The instance handle is returned embedded in the ioctl call as a ‘cookie’
  - Driver notes the cookie (in binder node inside driver) and in future transactions to Media Player, it sends the cookie, along with any msg from Application.
  - On app side, sp<BpMediaPlayerService>.create() method again creates a BpBinder with that handle of MediaPlayer

Binder IPC – Creation of proxy and native binders

- Calling API on media player instance
  - sp<BpMediaPlayer>.setDataSource(…)
  - The implementation creates a Parcel and passes it on to BpBinder
  - The IPC message is delivered to the media server.
  - The driver adds the ‘proxy’ pointer along with the message
  - The binder framework on the media server on receiving the cookie, fetches the native service instance and passes on the Parcel.
  - The instance eventually calls setDataSource on itself.