Deep dive into Binder

- Prasanna Kumar

Who am I?

- Prasanna Kumar
- Linux kernel enthusiast
- Contributed to open source projects
- Interested in Embedded devices and Android
- Can be reached at prasannatsmkumar@gmail.com
- Follow
  - www.github.com/prasannatsm
  - www.linkedin.com/in/prasannakumartsm/
  - @prasannatsm

What is Android

- OS for Smartphone and embedded devices
- Works on variety of hardware thanks to Linux kernel
- Code available with open source license (available source)
- A bunch of services that Apps rely on to provide their intended functionality

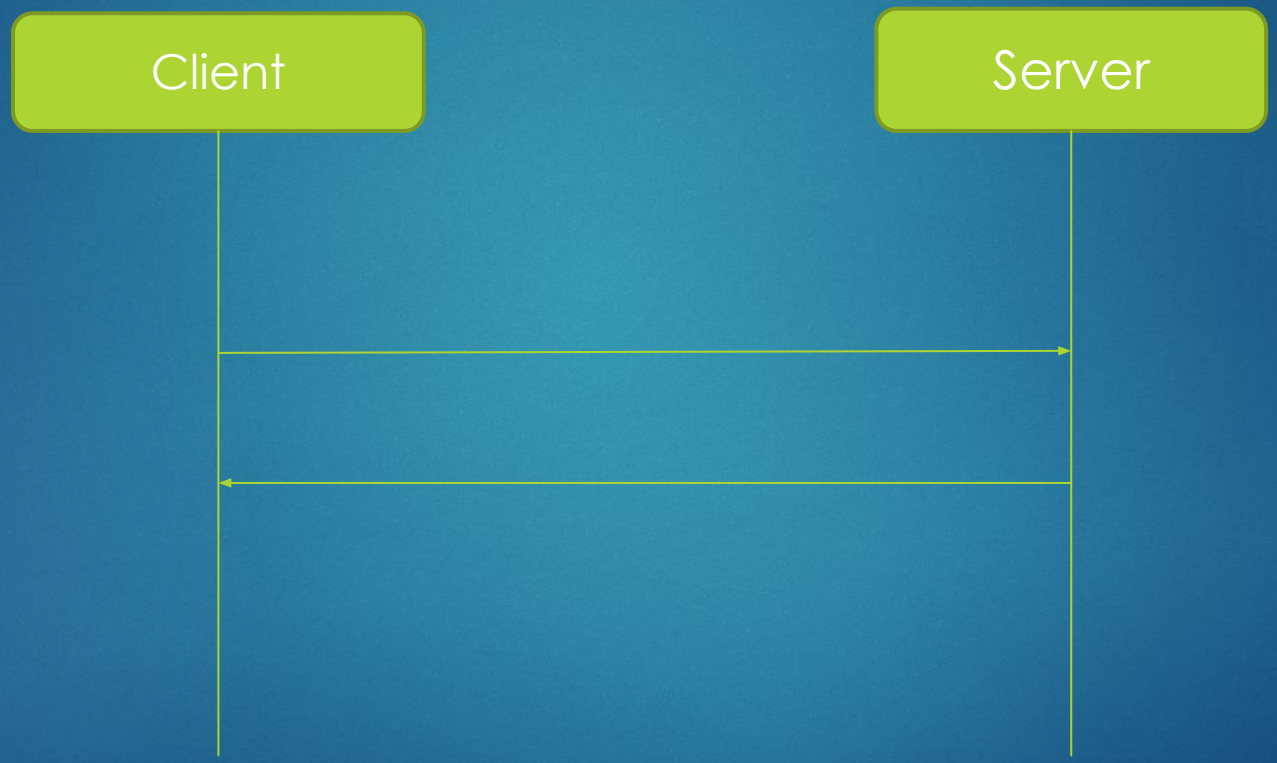
Android App

- An Android app consists of Activities, Services, Content Providers, Broadcast Receivers
- Gets a unique user id to provide data isolation between Apps
- Despite data isolation Apps can expose features and data to other applications in a secure manner
- These requires fast Inter Process Communication (IPC) mechanism
- Android security model requires Remote Procedure Call support
- Welcome to Binder

Binder

- Is a low overhead Inter Process Communication (IPC) and Remote Procedure Call (RPC) mechanism
- Implemented as a Linux kernel driver
- Makes Android framework a set of services which Apps would use
- Allows Android system services / framework components to run as separate processes
- Enables apps to be killed without any resource leaks
- Backbone of Android
- Audio, display, graphics, sensors etc won't work without Binder => Unusable system
- Intents, Content Providers, Messenger / Handler all use Binder under the hood
- Works in client server model

Client Server model



Binder framework

- A process cannot invoke another process's method directly
- Binder framework makes client to feel it is calling server's methods directly
- Binder framework includes Binder driver, libbinder, AIDL, IBinder and Binder interface, Parcel etc
- Binder driver is exposed via /dev/binder

Binder communication

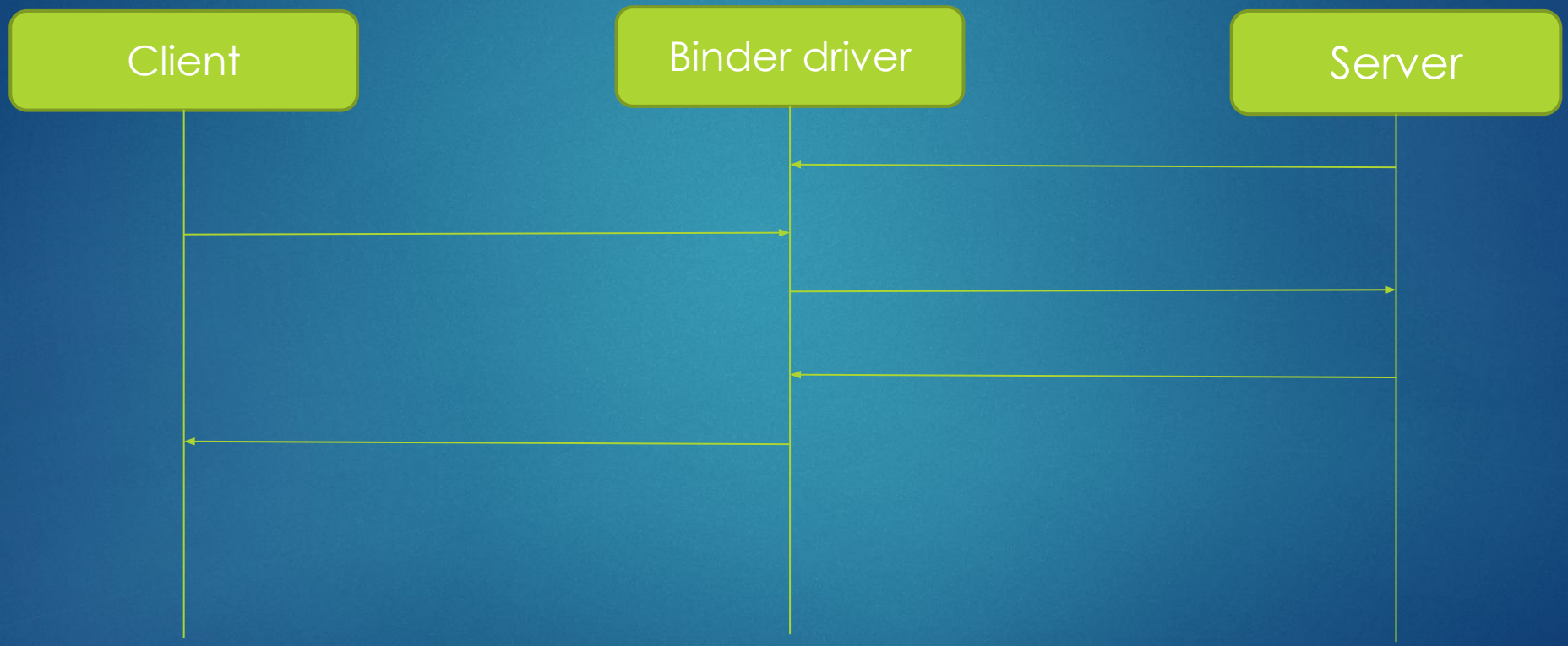
- Most of the communication happens via ioctl call
  - ioctl(binder_fd, BINDER_WRITE_READ, &write_read_obj);
  - write_read_obj is an instance of 'struct binder_write_read'
- write_command has a series of commands to binder driver
- Commands can be for book keeping (increment / decrement reference count), request for client death notification, request service from a server that needs a response (BC_TRANSACTION)

```
168  struct binder_write_read {
169      binder_size_t       write_size; /* bytes to write */
170      binder_size_t       write_consumed; /* bytes consumed by driver */
171      binder_uintptr_t    write_buffer;
172      binder_size_t       read_size;  /* bytes to read */
173      binder_size_t       read_consumed;  /* bytes consumed by driver */
174      binder_uintptr_t    read_buffer;
175  };
```

Binder transaction

- Server registers its capabilities with Binder driver and waits for request
- Each server gets a Binder token that identifies the service
- Client look up server, obtains the server's binder token
- Client sends ioctl command containing the request to binder driver
- Binder driver suspends the caller, copies request data to server's adders space, wakes up the waiting server and provides the request
- Server completes the requested action, sends the result via Binder driver
- Binder wakes up the suspended caller and provides the server's reply
- The process of sending a request till getting back a response via the Binder driver is called as a Binder Transaction

Binder transaction



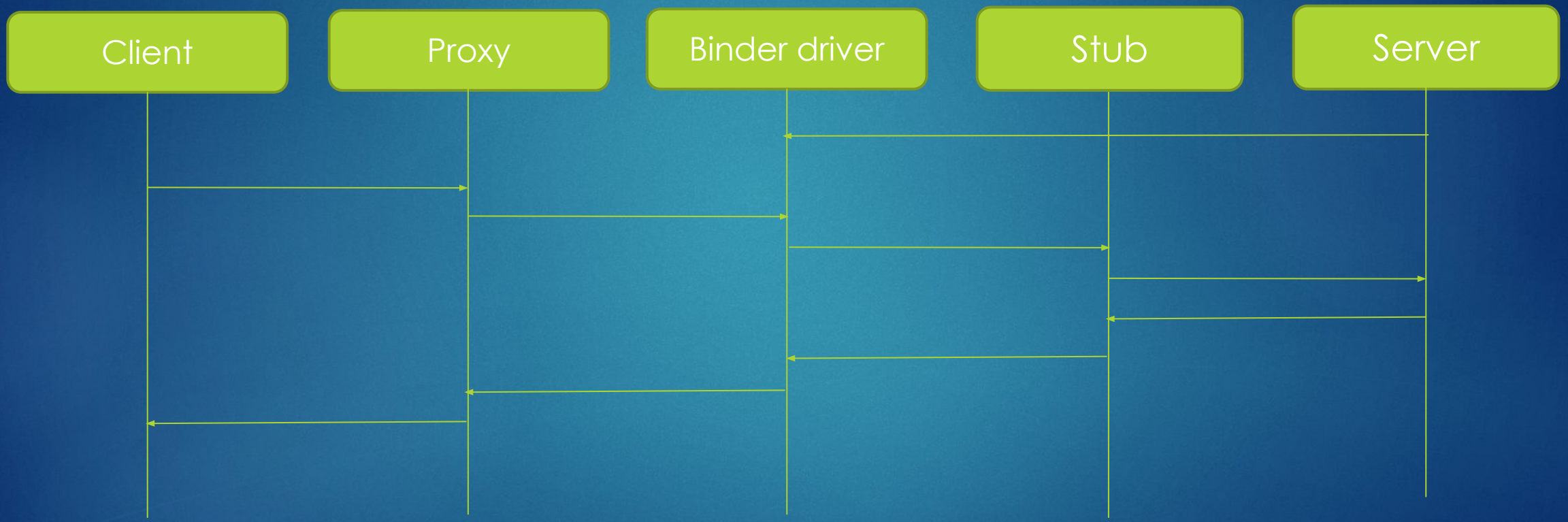
Parcel, libbinder, IBinder

- For client and server to understand the request / response a common data exchange format should be used between them
- The common data format is called as Parcel
- Storing data in a Parcel is called as marshalling, retrieving data from a Parcel is called as umarshalling
- Low level calls and providing Parcel to binder driver is abstracted by libbinder (C++ native library)
- IBinder is an interface that service must implement in order to use libbinder
- A convenience Java interface called IBinder exists that calls the corresponding native IBinder interface

Proxies, Stubs and AIDL

- Binder driver understands ioctl commands
- AIDL is a interface definition language that describes the functionality that a server implements
- Proxy implements the AIDL interface, marshals/unmarshals the data and makes IBinder calls, this is used in client side
- Stub is similar to Proxy, used in server side
- AIDL tool parses AIDL file and auto generates Proxy and Stub in Java language
- Native apps should implement Proxy and Stub

Binder in action



Service discovery

- Binder tokens are used to identify server
- Binder tokens are not fixed, changes on every boot
- Service discovery is required for finding out servers based on the functionality they provide
- Context manager is a process that stores binder token of servers registered with unique string acting as a key for the token
- Apps query the context manager with the unique string, obtains the binder token (using getSystemService API) and calls the service with that token
- Only one context manager can exists in a system so it has to be started before any other server starts

Service discovery

- ServiceManager is the context manager in Android
- ServiceManager registers first with binder and gets binder token 0
- It gets binder token 0 on every boot
- All other system services register their tokens with ServiceManager

Location Request

- To get location an app (client) calls LocationManger's (server) getBestProvider method

```
1   @Override
2   public void onCreate(Bundle savedInstanceState) {
3       super.onCreate(savedInstanceState);
4
5       locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
6       provider = locationManager.getBestProvider(new Criteria(), false);
7   }
8
9   @Override
10  public void onLocationChanged(Location location) {
11      int lat = (int) (location.getLatitude());
12      int lng = (int) (location.getLongitude());
13
14      /* Do something with lat and lang */
15  }
```

Thank you

Questions?

Backup Slides

Binder History

- Started at Be, Inc, as a part of "next generation BeOS" in 2001
- Binder's first implementation is used in Palm Cobalt (a micro kernel OS) after Be, Inc, was acquired by PalmSource
- Binder is ported to Linux in 2005 as Palm started using Linux and Binder's code is open sourced
- Open source implementation is called as OpenBinder
- Dianne Hackborn, a key member of OpenBinder team, joined Android team at Google in 2008
- Used OpenBinder for initial Android bring up and used in internal Android release
- User space parts of Binder were rewritten due to license incompatibility
- Kernel driver is rewritten to follow Linux kernel model and it is used in external release

Why not existing IPC mechanism?

- Most of the low level Android libraries use standard IPC and not Binder
- Input flinger uses pipes to send input data
- Setprop, getprop uses sockets

Linux mainline kernel and Binder

- Attempts were made to implement Binder features with existing IPC mechanisms but all failed eventually
- Except Binder no other IPC mechanism has ioctl based interface
- Global lock in Binder was removed due to push from mainline Linux developers leading to massive improvement in Binder's performance