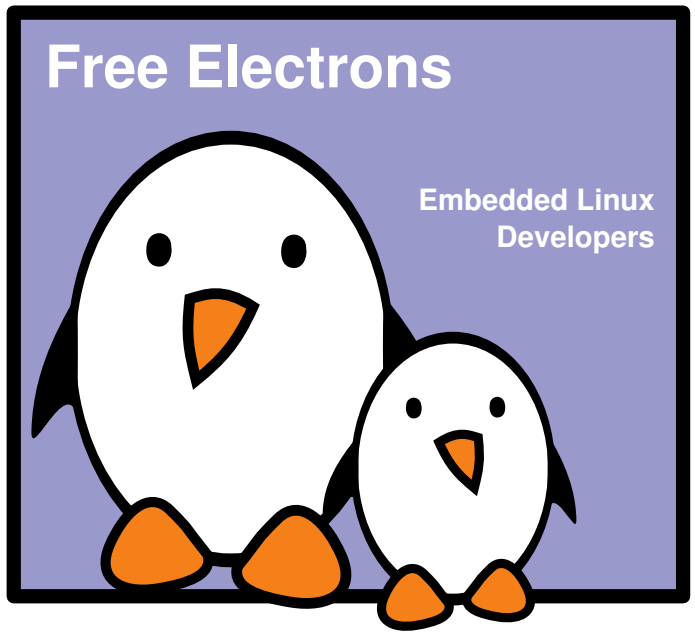
A Look through the Android Stack

Maxime Ripard
Free Electrons



Latest update: July 10, 2012.
Document updates and sources:
Corrections, suggestions, contributions and translations are welcome!

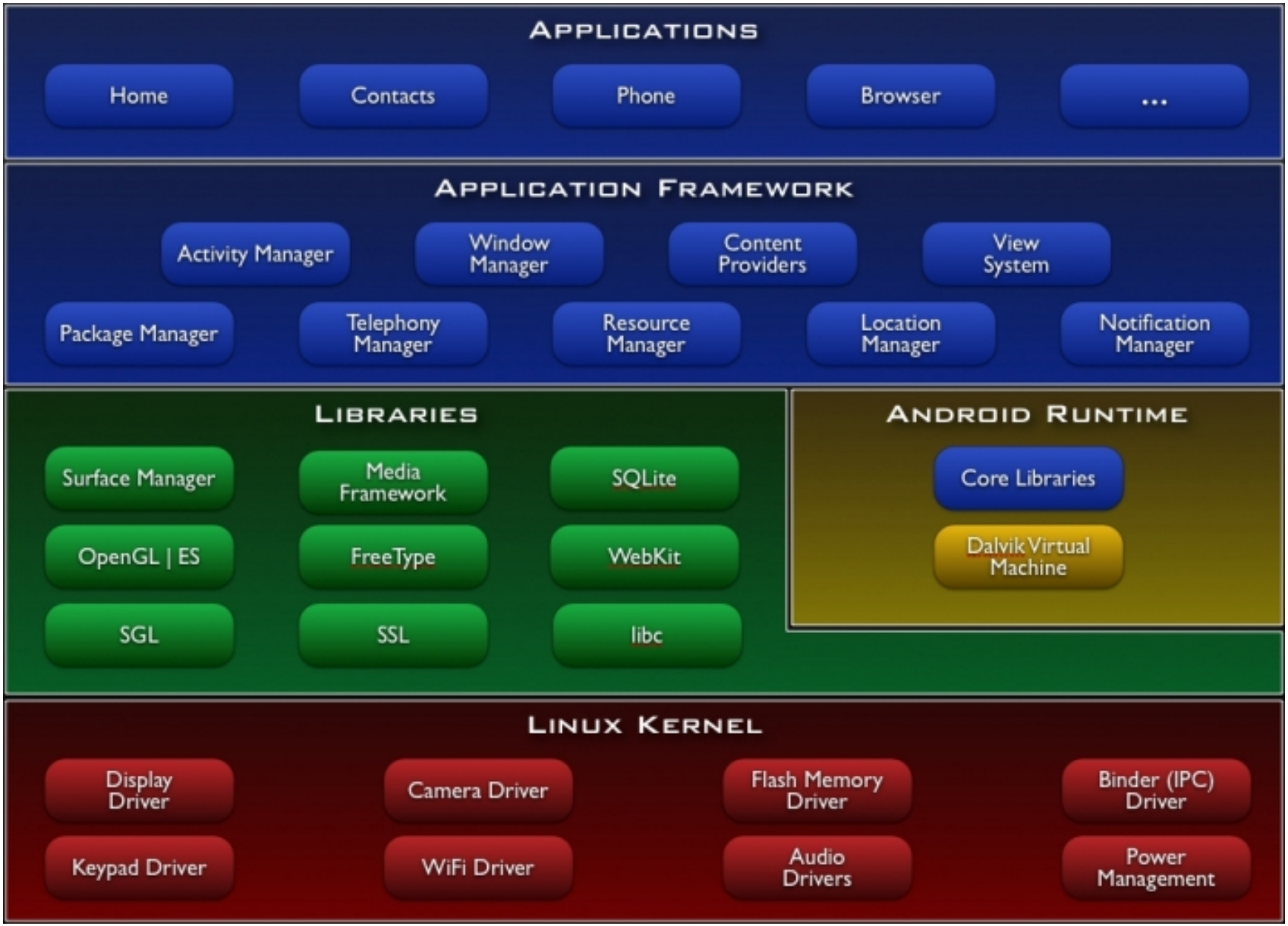
Maxime Ripard

- Embedded Linux engineer and trainer at Free Electrons since 2011
  - Embedded Linux development: kernel and driver development, system integration, boot time and power consumption optimization, etc.
  - Embedded Linux, Embedded Android and driver development training, with materials freely available under a Creative Commons license.
  - `http://free-electrons.com`
- Contributor to Buildroot, a simple open-source embedded build system

What is Android?

- Mobile OS bought by Google in 2005
- Opened up to the world through the Android Open Source Project the 21 October 2008 with the release of Android 1.0 (mostly) under the Apache 2.0 License
- Has deep foundation into the open-source/free software world, even if a huge part has been written from scratch: uses Linux, WebKit, SQLite, OpenSSL, etc.
- Is now one of the most successful mobile OS in the world

Architecture



Booting Up...

Booting Up...

The Linux Kernel

The Linux Kernel

- Used as the foundation of the Android system
- Numerous additions from the stock Linux, including new IPC (Inter-Process Communication) mechanisms, alternative power management mechanism, new drivers and various additions across the kernel
- These changes are beginning to go into the `staging/` area of the kernel, as of 3.4, after being a complete fork for a long time

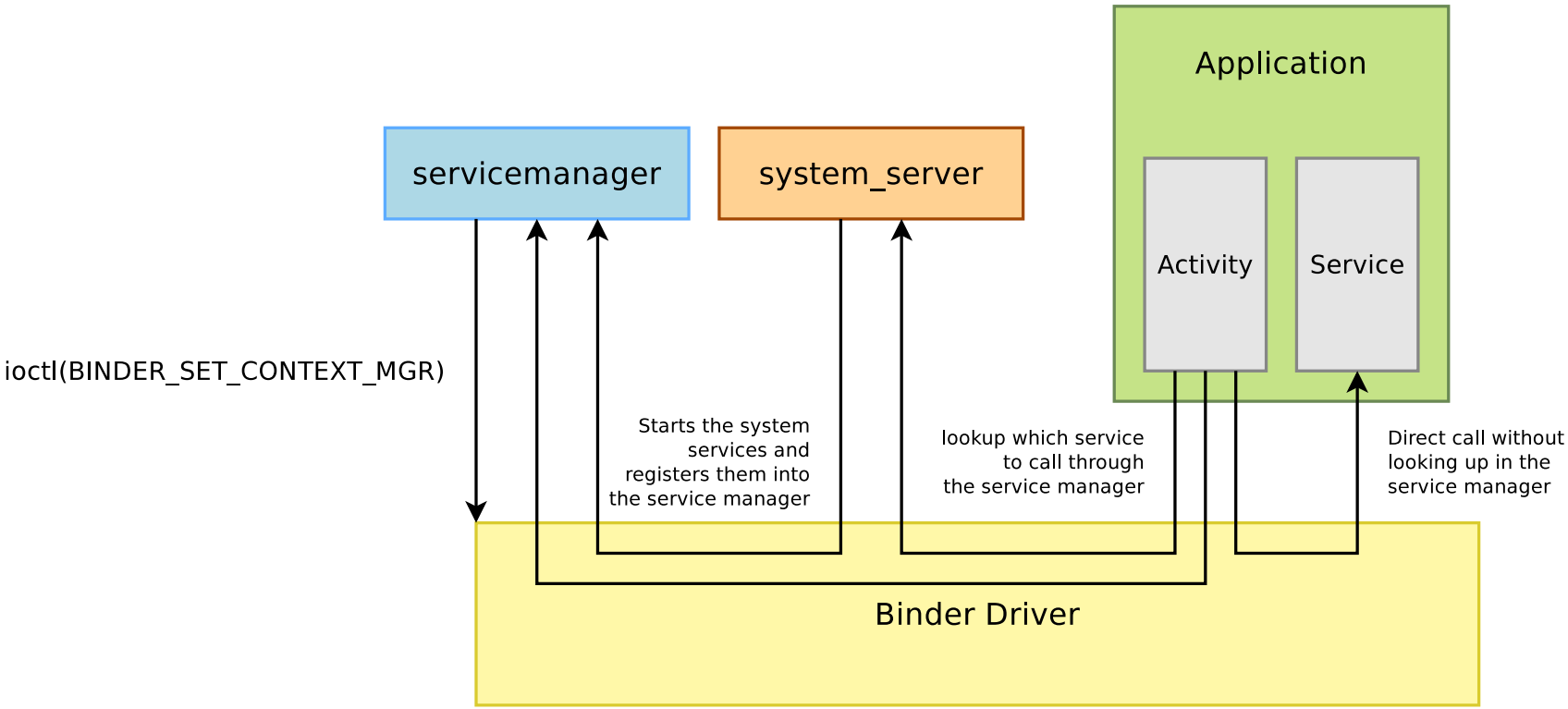
Wakelocks

- Android's answer to these power management constraints is wakelocks
- One of the most famous Android changes, because of the flame wars it spawned
- The main idea is instead of letting the user decide when the devices need to go to sleep, the kernel is set to suspend as soon and as often as possible.
- In the same time, Android allows applications and kernel drivers to voluntarily prevent the system from going to suspend, keeping it awake (thus the name wakelock)
- This implies to write the applications and drivers to use the wakelock API.
  - Applications do so through the abstraction provided by the API
  - Drivers must do it themselves, which prevents to directly submit them to the vanilla kernel

Binder

- RPC/IPC mechanism
- Takes its roots from BeOS and the OpenBinder project, which some of the current Android engineers worked on
- Adds remote object invocation capabilities to the Linux Kernel
- One of the very basic functionalities of Android. Without it, Android cannot work.
- Every call to the system servers go through Binder, just like every communication between applications, and even communication between the components of a single application.

Binder


Application
servicemanager
system_server
Activity
Service
ioctl(BINDER_SET_CONTEXT_MGR)
Starts the system services and registers them into the service manager
lookup which service to call through the service manager
Direct call without looking up in the service manager
Binder Driver

Ashmem

- `ndk/docs/system/libc/SYSV-IPC.html` illustrates all the love Android developers have for the shared memory mechanism in Linux
- Ashmem is the response to the flaws exposed
- Notable differences are:
  - Reference counting so that the kernel can reclaim resources which are no longer in use
  - There is also a mechanism in place to allow the kernel to shrink shared memory regions when the system is under memory pressure.
- The standard use of Ashmem in Android is that a process opens a shared memory region and share the obtained file descriptor through Binder.

Low Memory Killer

- When the system goes out of memory, Linux throws the OOM Killer to cleanup memory greedy processes
- However, this behaviour is not predictable at all, and can kill very important components of a phone (Telephony stack, Graphic subsystem, etc) instead of low priority processes (Angry Birds)
- The main idea is to have another process killer, that kicks in before the OOM Killer and takes into account the time since the application was last used and the priority of the component for the system
- It uses various thresholds, so that it first notifies applications so that they can save their state, then begins to kill non-critical background processes, and then the foreground applications
- As it is run to free memory before the OOM Killer, the latter will never be run, as the system will never run out of memory

Various additions

- Android also has a lot of minor features added to the Linux kernel:
  - RAM Console, a RAM-based console that survives a reboot to hold kernel logs
  - *pmem*, a physically contiguous memory allocator, written specifically for the HTC G1, to allocate heaps used for 2D hardware acceleration
  - Alarm Timers, which combines Real-Time Clock with the high-resolution timers to wake up a process even when the device is in suspend.
  - Paranoid Network, which restricts access to the network interfaces only to a given GID
  - ADB
  - YAFFS2
  - Timed GPIOs

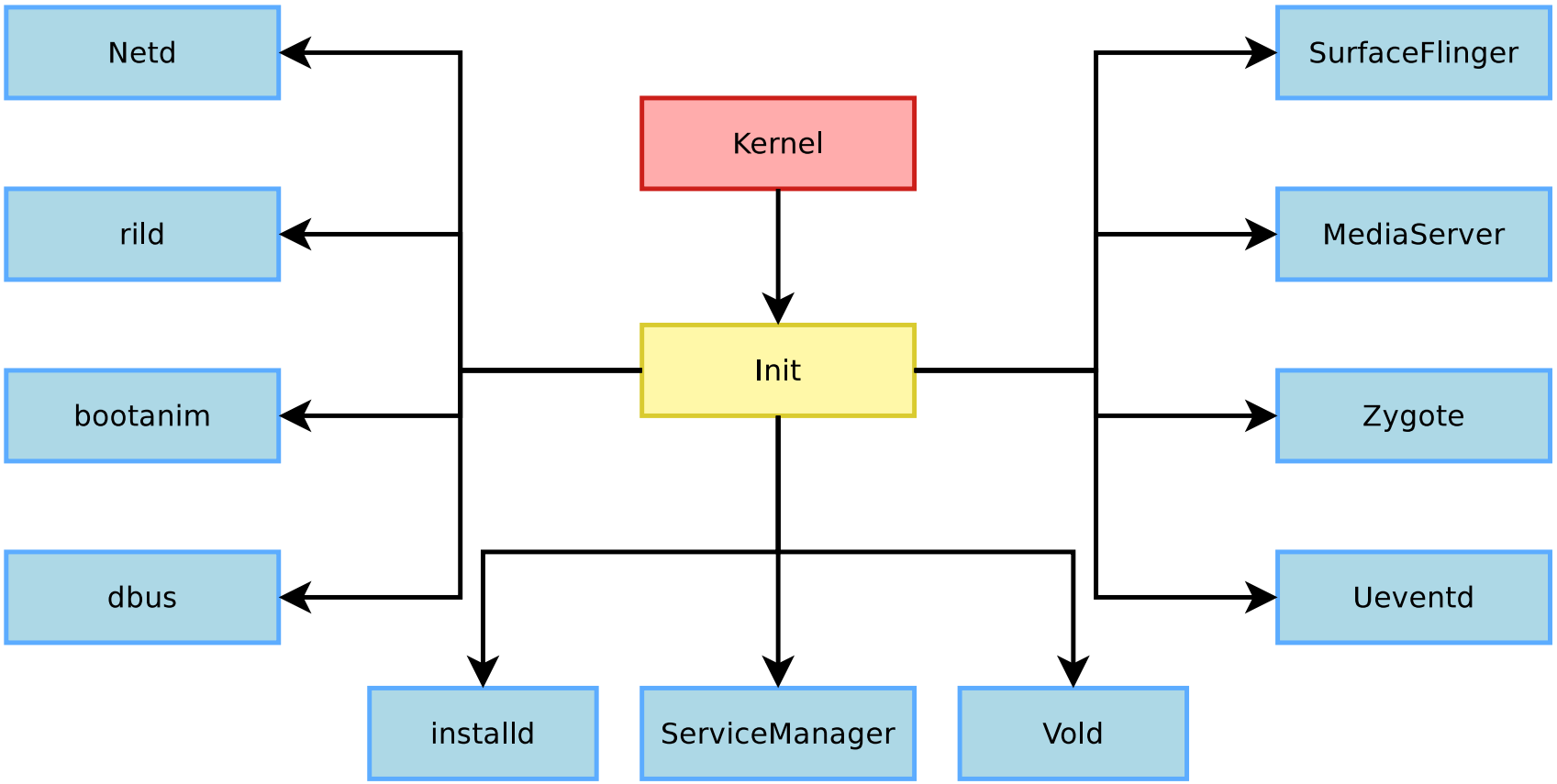
Booting Up...

Init

Android's init

- Once again, Google has developed his own instead of relying on an existing one.
- However, it has some interesting features, as it can also be seen as a daemon on the system
  - it manages device hotplugging, with basic permissions rules for device files, and actions at device plugging and unplugging: set particular permissions to the device files, mount the filesystems, start a daemon, etc.
  - it monitors the services it started, so that if they crash, it can restart them, reboot the device, etc.
  - it monitors system properties so that you can take actions when a particular one is modified
- Basically, it is quite comparable to sysvinit + udev

Android startup



Booting Up...

Dalvik and Zygote

Dalvik

- Dalvik is the virtual machine, executing Android applications
- It is an interpreter written in C/C++, and is designed to be portable, lightweight and run well on mobile devices
- It is also designed to allow several instances of it to be run at the same time while consuming as little memory as possible
- Two execution modes
  - `portable`: the interpreter is written in C, quite slow, but should work on all platforms
  - `fast`: Uses the *mterp* mechanism, to define routines either in assembly or in C optimized for a specific platform. Instruction dispatching is also done by computing the handler address from the opcode number
- It uses the *Apache Harmony* Java framework for its core libraries

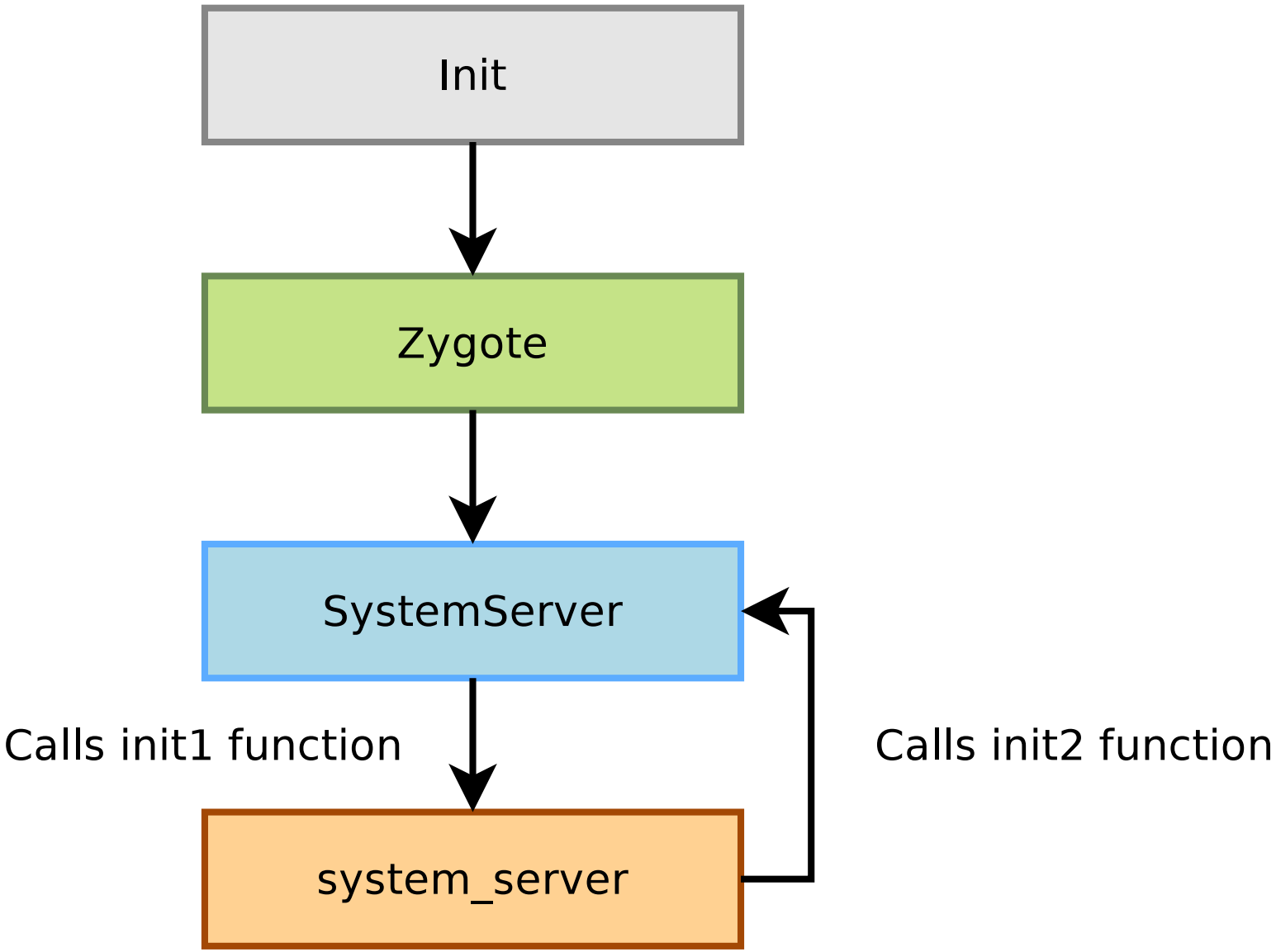
Zygote

- Dalvik is started by `Zygote`
- `frameworks/base/cmds/app_process`
- At boot, Zygote is started by init, it then
  - Initializes a virtual machine in its address space
  - Loads all the basic Java classes in memory
  - Starts the system server
  - Waits for connections on a UNIX socket
- When a new application should be started:
  - Android connects to Zygote through the socket to request the start of a new application
  - Zygote forks
  - The child process loads the new application and start executing it

Booting Up...

Service Manager and Various Services

System Server boot



The first step: system_server.c

- Located in frameworks/base/cmds/system_server
- Started by Zygote through the SystemServer
- Starts all the various native services:
  - SurfaceFlinger
  - SensorService
  - AudioFlinger
  - MediaPlayerService
  - CameraService
  - AudioPolicyService
- It then calls back the SystemServer object's init2 function to go on with the initialization

Java Services Initialization

- Located in `frameworks/base/services/java/com/android/server/SystemServer.java`
- Starts all the different Java services in a different thread by registering them into the Service Manager
- `PowerManager`, `ActivityManager` (also handles the ContentProviders), `PackageManager`, `BatteryService`, `LightsService`, `VibratorService`, `AlarmManager`, `WindowManager`, `BluetoothService`, `DevicePolicyManager`, `StatusBarManager`, `InputMethodManager`, `ConnectivityService`, `MountService`, `NotificationManager`, `LocationManager`, `AudioService`, `...`
- If you wish to add a new system service, you will need to add it to one of these two parts to register it at boot time

Communication within the system

IPCs

- On modern systems, each process has its own address space, allowing to isolate data
- This allows for better stability and security: only a given process can access its address space. If another process tries to access it, the kernel will detect it and kill this process.
- However, interactions between processes are sometimes needed, that's what IPCs are for.
- On classic Linux systems, several IPC mechanisms are used:
  - Signals
  - Semaphores
  - Sockets
  - Message queues
  - Pipes
  - Shared memory
- Android, however, uses mostly:
  - Binder
  - Ashmem and Sockets

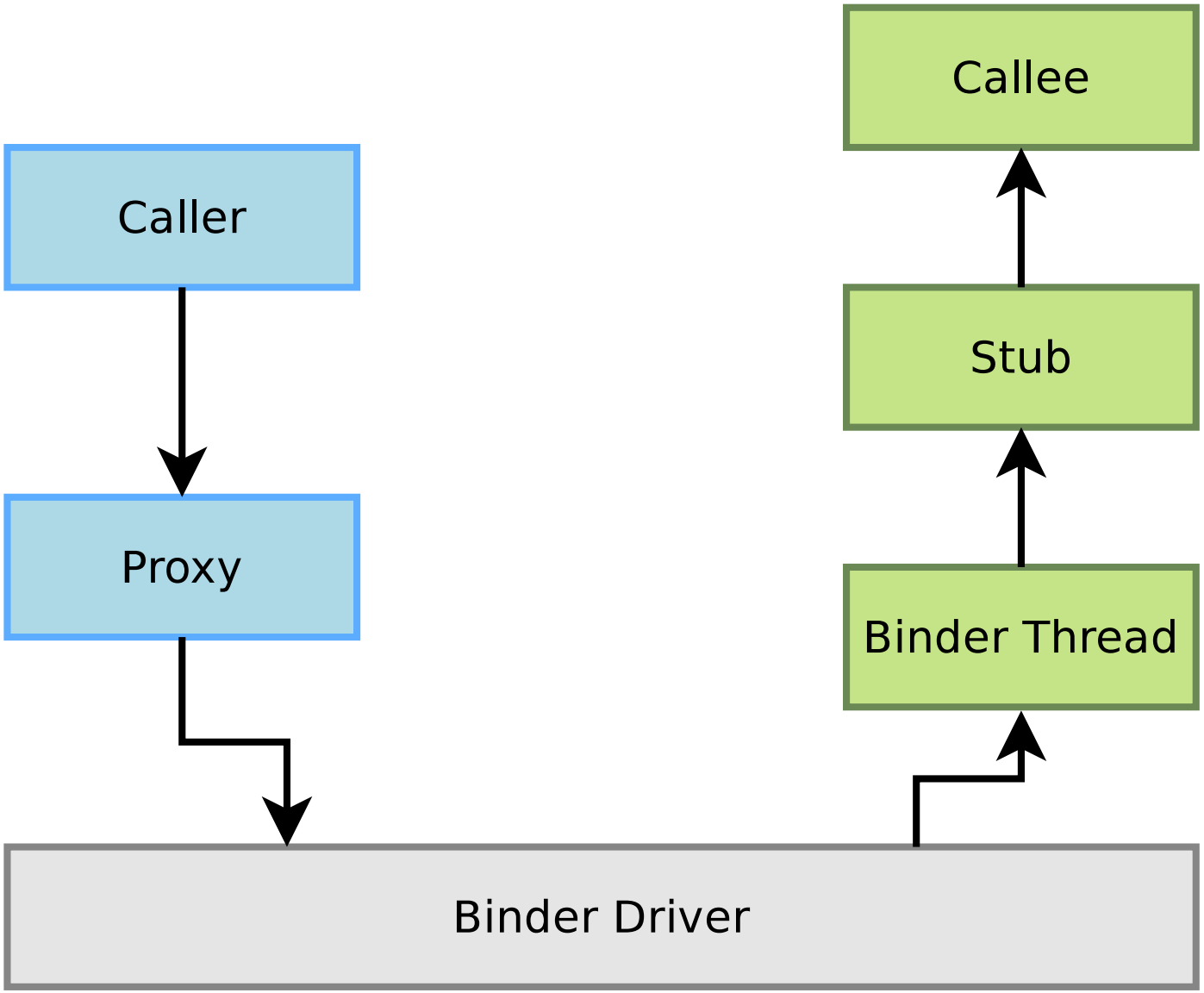
Binder 1/2

- Uses shared memory for high performance
- Uses reference counting to garbage collect objects no longer in use
- Data are sent through *parcels*, which is some kind of serialization
- Used across the whole system, e.g., clients connect to the window manager through Binder, which in turn connects to SurfaceFlinger using Binder
- Each object has an *identity*, which does not change, even if you pass it to other processes.

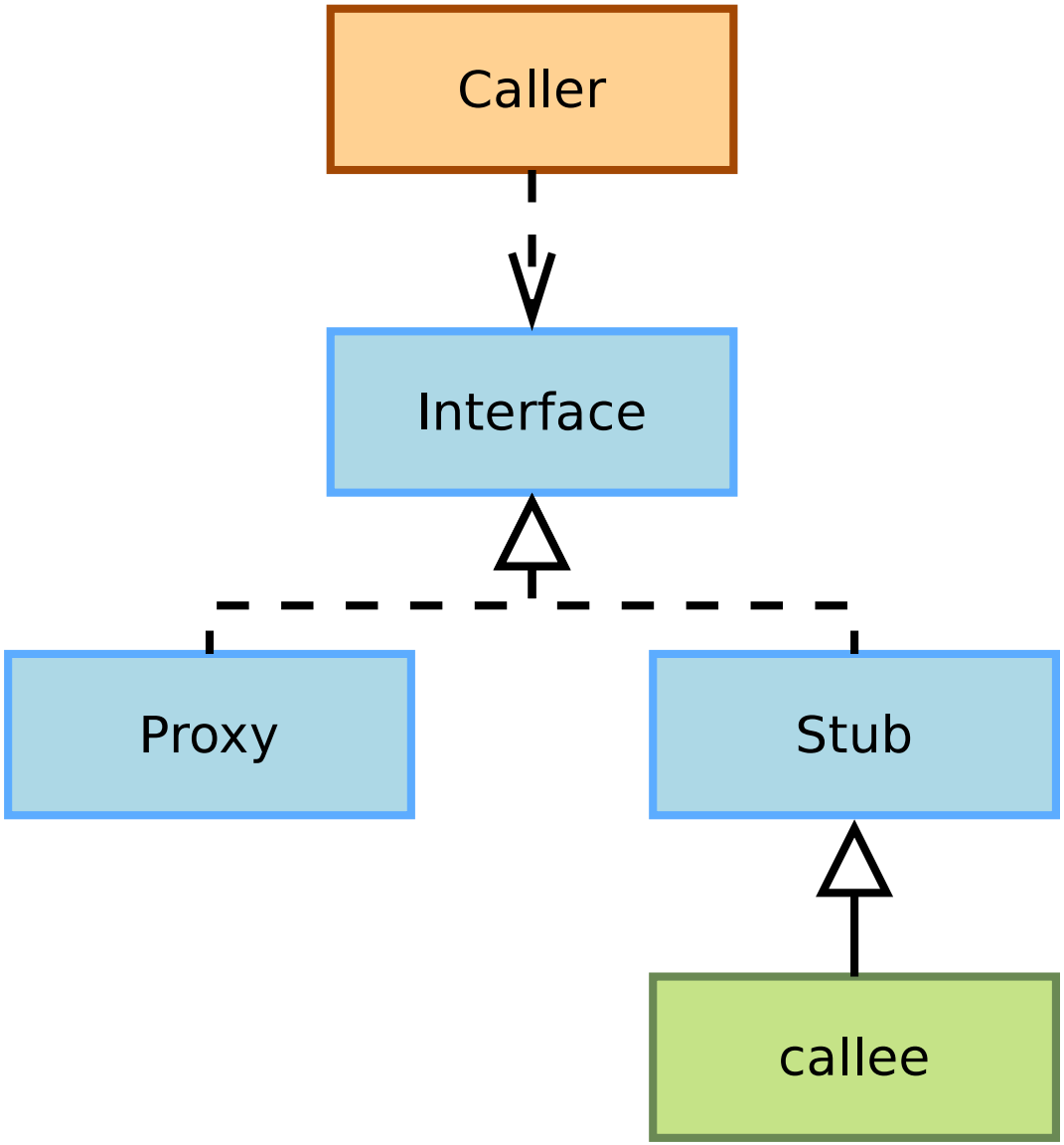
Binder 2/2

- This is useful if you want to separate components in distinct processes, or to manage several components of a single process (i.e. Activity's Windows).
- Object identity is also used for security. Some token passed correspond to specific permissions. Another security model to enforce permissions is for every transaction to check on the calling UID.
- Binder also supports one-way and two-way messages

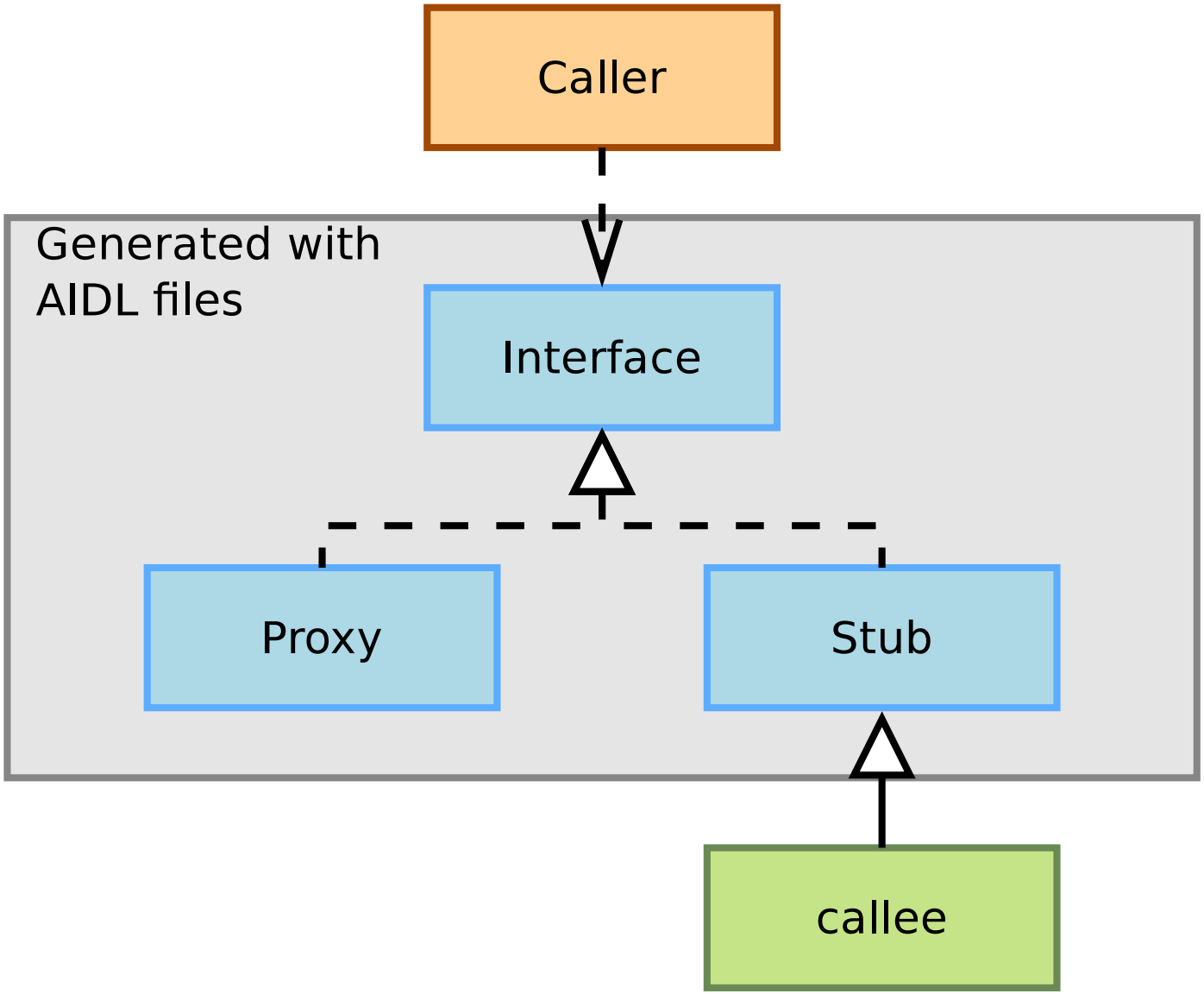
Binder Mechanism



Binder Implementation 1/2



Binder Implementation 2/2



Android Interface Definition Language (AIDL)

- Very similar to any other Interface Definition Language you might have encountered
- Describes a programming interface for the client and the server to communicate using IPCs
- Looks a lot like Java interfaces. Several types are already defined, however, and you can't extend this like what you can do in Java:
  - All Java primitive types (`int`, `long`, `boolean`, etc.)
  - `String`
  - `CharSequence`
  - `Parcelable`
  - `List` of one of the previous types
  - `Map`

Intents

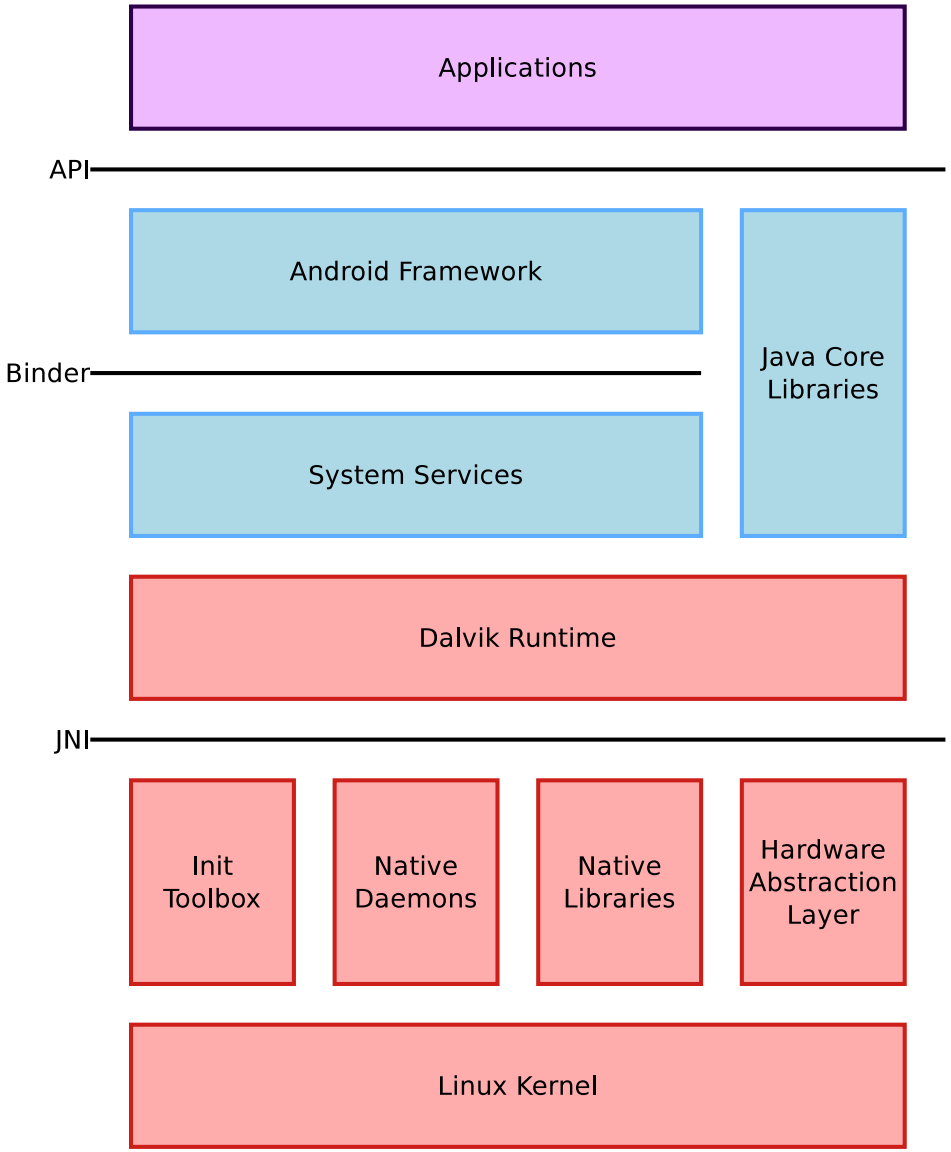
- Intents are a high-level use of Binder
- They describe the intention to do something
- They are used extensively across Android
  - Activities, Services and BroadcastReceivers are started using intents
- Two types of intents:

  explicit The developer designates the target by its name

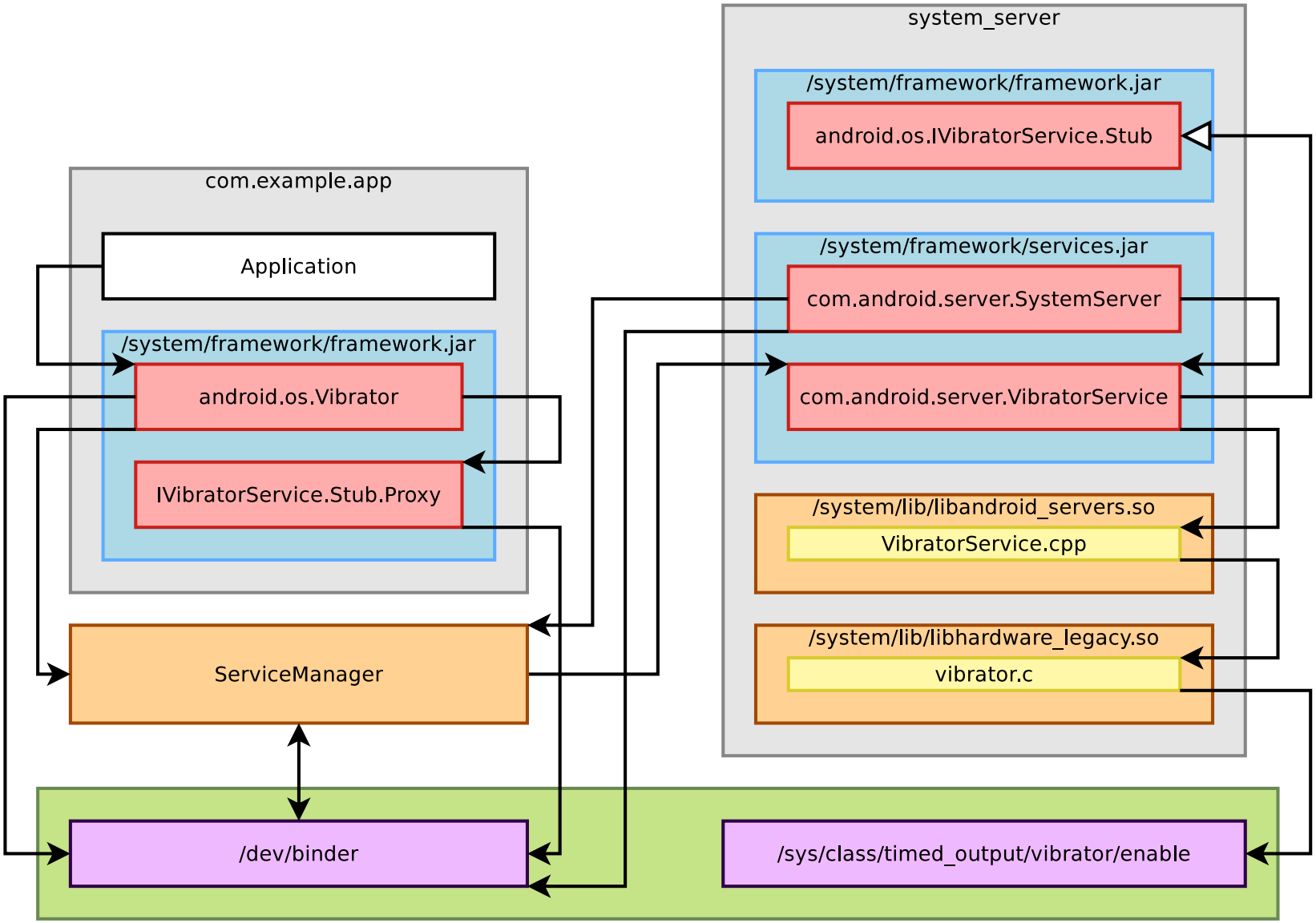
  implicit There is no explicit target for the Intent. The system will find the best target for the Intent by itself, possibly asking the user what to do if there are several matches

Wrap-up

Interactions in the System



Example: VibratorService



Questions?

Our training slides are available on
http://free-electrons.com/doc/training/android/

Maxime Ripard
maxime.ripard@free-electrons.com

PDF and sources of these slides will be available on http://free-electrons.com/pub/conferences/2012/lsm/