Leveraging Android's Linux Heritage

Android Builders Summit 2012

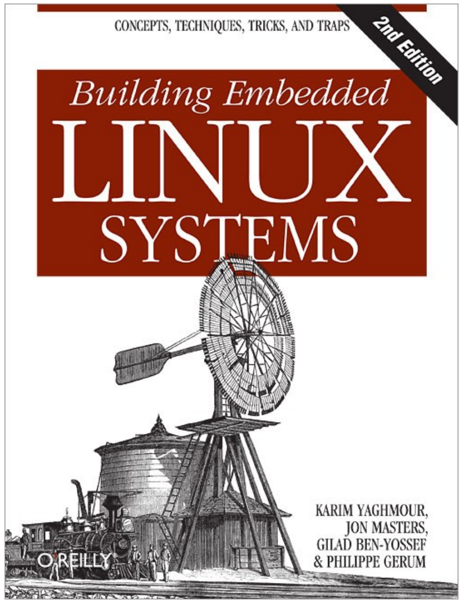
Karim Yaghmour
@karimyaghmour

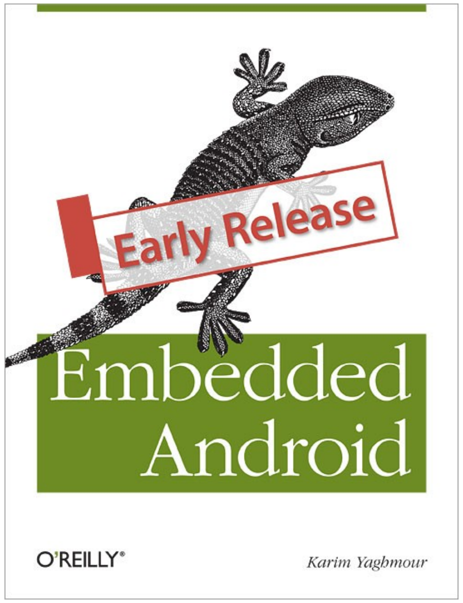
>OPERSYS™

Delivered and/or customized by

About

- Author of:


CONCEPTS, TECHNIQUES, TRICKS, AND TRAPS
2nd Edition
Building Embedded LINUX SYSTEMS
KARIM YAGHMOUR, JON MASTERS, GILAD BEN-YOSSEF & PHILIPPE GERUM
O'REILLY



- Introduced Linux Trace Toolkit in 1999
- Originated Adeos and relayfs (kernel/relay.c)
- Training, Custom Dev, Consulting, ...

“Android took GNU out the back door,
shot him in the head, and
ran away with the penguin”

-- Surely from Tarantino's next flick

Agenda

- Goal
- Rationale
- Stack Comparison
- Roadblocks
- Where do I start?
- Coexistence Approaches
- Unresolved / Uncharted
- Demo

1. Goal

- ~~Opening as many cans of worms as possible~~
- Can “Linux” and Android Coexist and Interact?

2. Rationale

- A ton of mature user-space packages available
  - Linux has been around for 20 years
  - Linux's user-space has been developed in the open
- A ton of “Linux”-centric stacks have been developed through the years
  - “Porting” to Android not always possible/desirable/realistic
- Android doesn't provide everything
  - Touch-based, consumer-oriented
  - Linux is very strong on backend/server side
- Android exhibits symptoms of “my way or the highway” design

3. Stack Comparison

- Android Concepts Recap
- Overall Architecture EL
- Overall Architecture Android

3.1. Android Concepts Recap

- Components
- Intents
- Component lifecycle
- Manifest file
- Processes and threads
- Remote procedure calls

3.1.1. Components

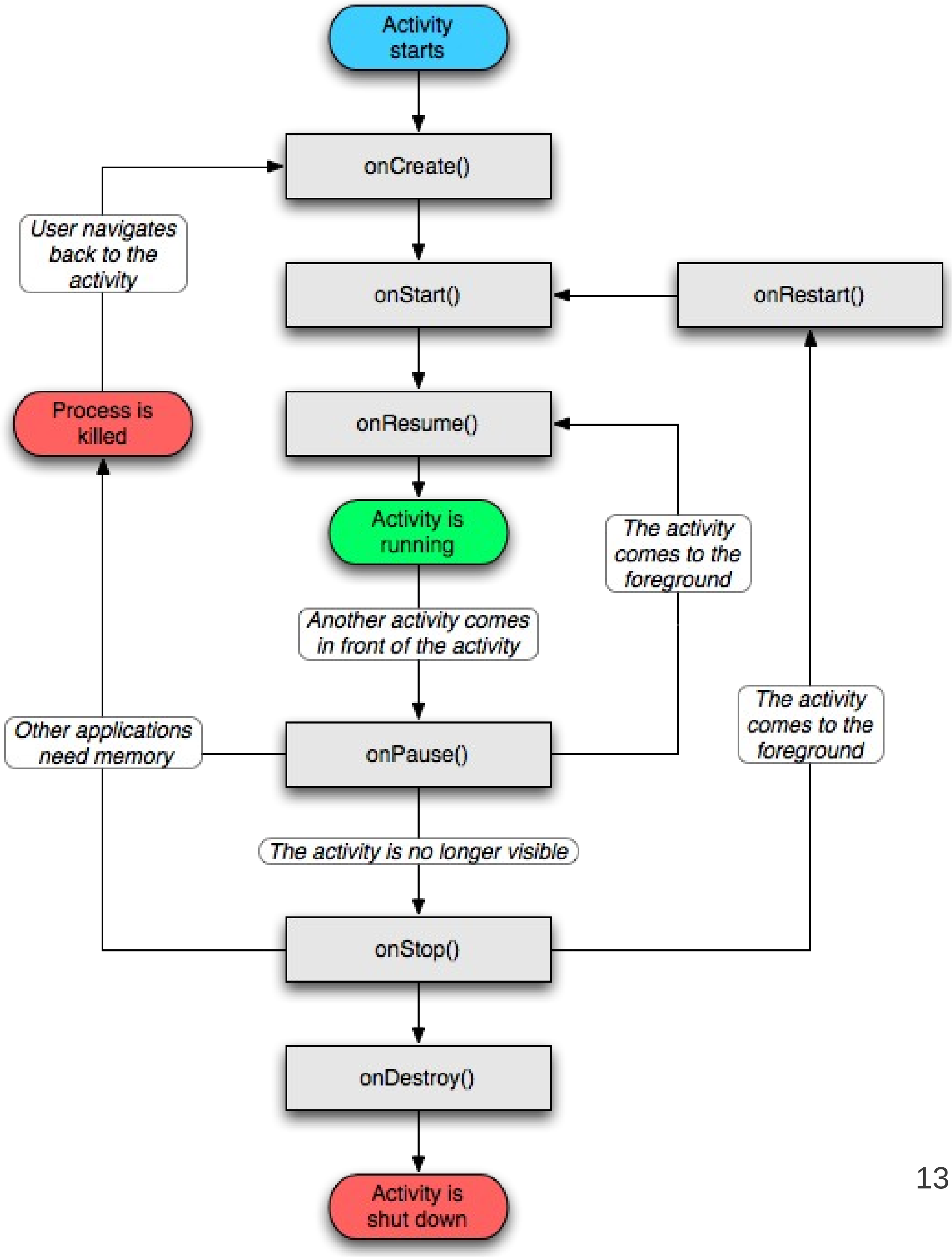
- 1 App = N Components
- Apps can use components of other applications
- App processes are automagically started whenever any part is needed
- Ergo: N entry points, !1, and !main()
- Components:
  - Activities
  - Services
  - Broadcast Receivers
  - Content Providers

3.1.2. Intents

- Intent = asynchronous message w/ or w/o designated target
- Like a polymorphic Unix signal, but w/o required target
- Intents “payload” held in Intent Object
- Intent Filters specified in Manifest file

3.1.3. Component lifecycle

- System automagically starts/stops/kills processes:
  - Entire system behaviour predicated on low memory
- System triggers Lifecycle callbacks when relevant
- Ergo: Must manage Component Lifecycle
- Some Components are more complex to manage than others


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

3.1.4. Manifest file

- Informs system about app’s components
- XML format
- Always called AndroidManifest.xml
- Activity = <activity> ... static
- Service = <service> ... static
- Broadcast Receiver:
  - Static = <receiver>
  - Dynamic = Context.registerReceiver()
- Content Provider = <provider> ... static

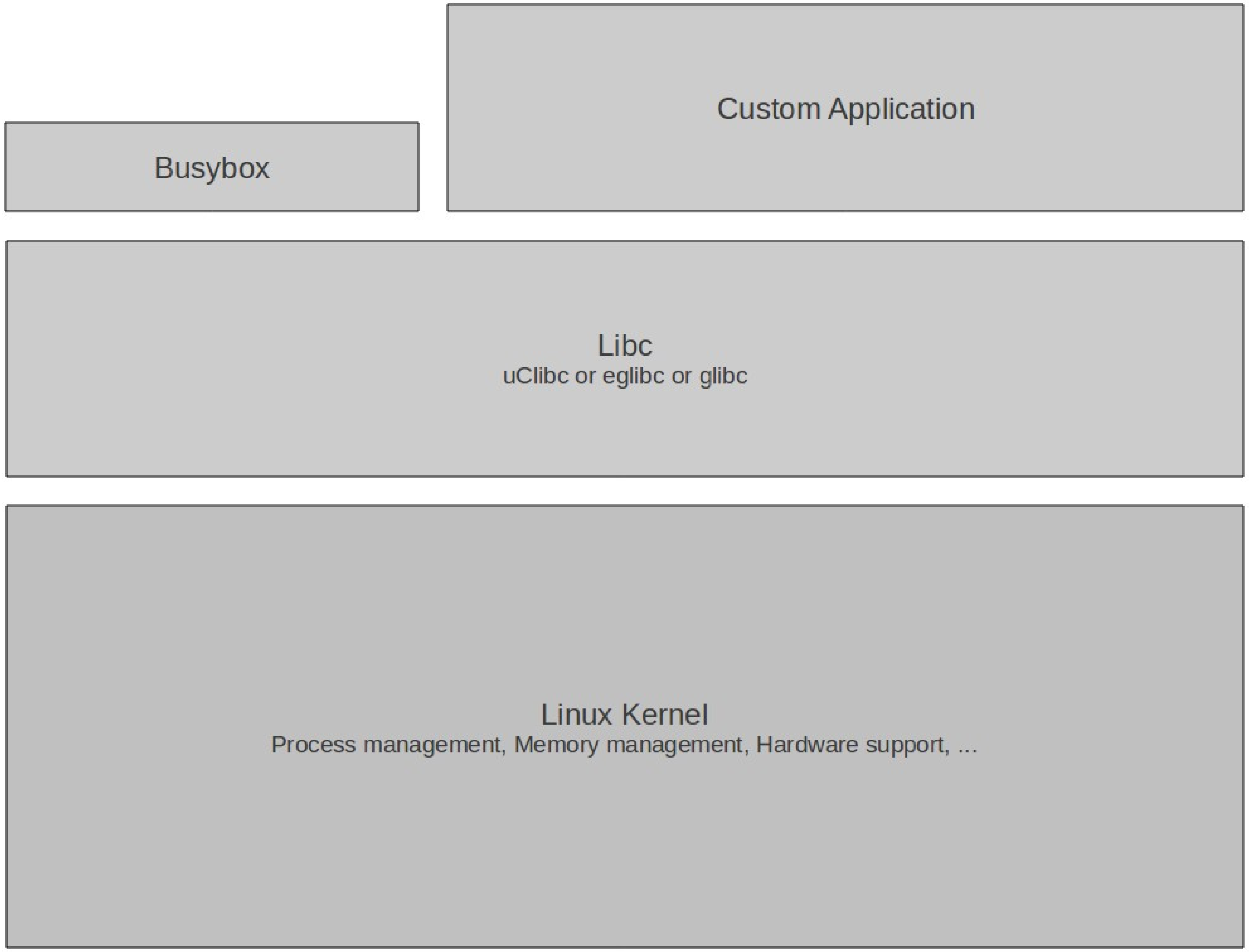
3.1.5. Processes and threads

- Processes
  - Default: all callbacks to any app Component are issued to the main process thread
  - <activity>—<service>—<recipient>—<provider> have process attribute to override default
  - Do NOT perform blocking/long operations in main process thread:
    - Spawn threads instead
  - Process termination/restart is at system’s discretion
  - Therefore:
    - Must manage Component Lifecycle
- Threads:
  - Create using the regular Java Thread Object
  - Android API provides thread helper classes:
    - Looper: for running a message loop with a thread
    - Handler: for processing messages
    - HandlerThread: for setting up a thread with a message loop

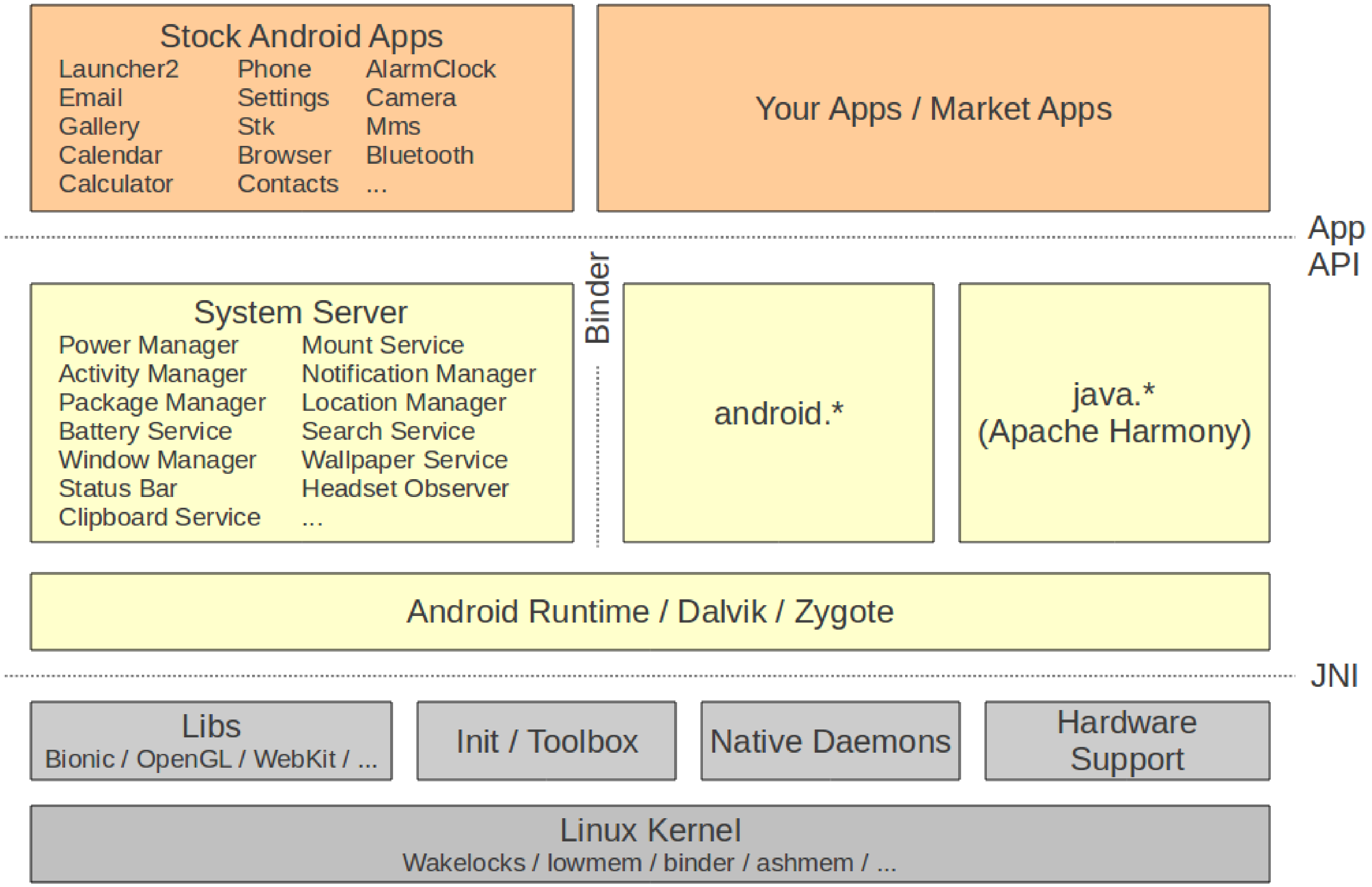
3.1.6. Remote procedure calls

- Android RPCs = Binder mechanism
- No Sys V IPC due to in-kernel resource leakage
- Binder is a low-level functionality, not used as-is
- Instead: must define interface using Interface Definition Language (IDL)
- IDL fed to aidl Tool to generate Java interface definitions

3.2. Overall Architecture - EL


Custom Application
Busybox
Libc
uClibc or eglibc or glibc
Linux Kernel
Process management, Memory management, Hardware support, ...

3.3. Overall Architecture – Android



4. Roadblocks

- Filesystem
  - Android is non-FHS-compliant
- C library
  - Bionic vs. glibc
- Interconnect fabric
  - Intents vs. DBUS
- IPC
  - Binder vs. Sockets and other std Unix IPC
- Display management
  - SurfaceFlinger vs. X
- I/O
  - Framebuffer, keyboard, mouse, disk, ...

5. Where do I start?

- Android-side:
  - AOSP
- “Linux”-side:
  - Traditional distro
    - Ubuntu, Fedora, Debian, Gentoo, ...
  - Embedded distro
    - Yocto, Buildroot, LTIB, ...
  - Build Your Own
  - Cherry-picking

6. Coexistence Approaches

- Single filesystem
  - Build system integration
  - Build-time aggregation
  - Image repackaging
- chroot jails
  - Have a look at AlwaysInnovating Gregoire Gentil's ELC presentation
  - Patching to lots of pieces of the OS
  - Use of one FB for each OS or chvt
- Virtualization / Paravirtualization
  - QEMU
  - XEN?

7. Been there done that

- BusyBox in CyanogenMod
- Gstreamer vs. Stagefright
- Don't know how they do it:
    - Alien Dalvik: Android on Meego
- ...

8. Unresolved / Uncharted

- Binder from glibc
- Intent <-> DBUS bridge
- Running Android apps in X
- Running X apps in Android

  “The easier thing to do, which would work on just about all Android phones without having to modify the system software at all, would be to port an X server to the NDK, using a SurfaceFlinger Surface as its root window.

  You could do a generic "X11WrapperApp" that has you XSurfaceFlinger bundled and launches whatever X based app you want, and have it all play nice together.

  A bit more work would be to just do an implementation of xlib that sits on top of a native Android window (opengl ES 2 if you like) without any server in the middle, and again bundle this and the X based app of your choice and you have something that is a first class app on the phone without any need for modifying the OS.”

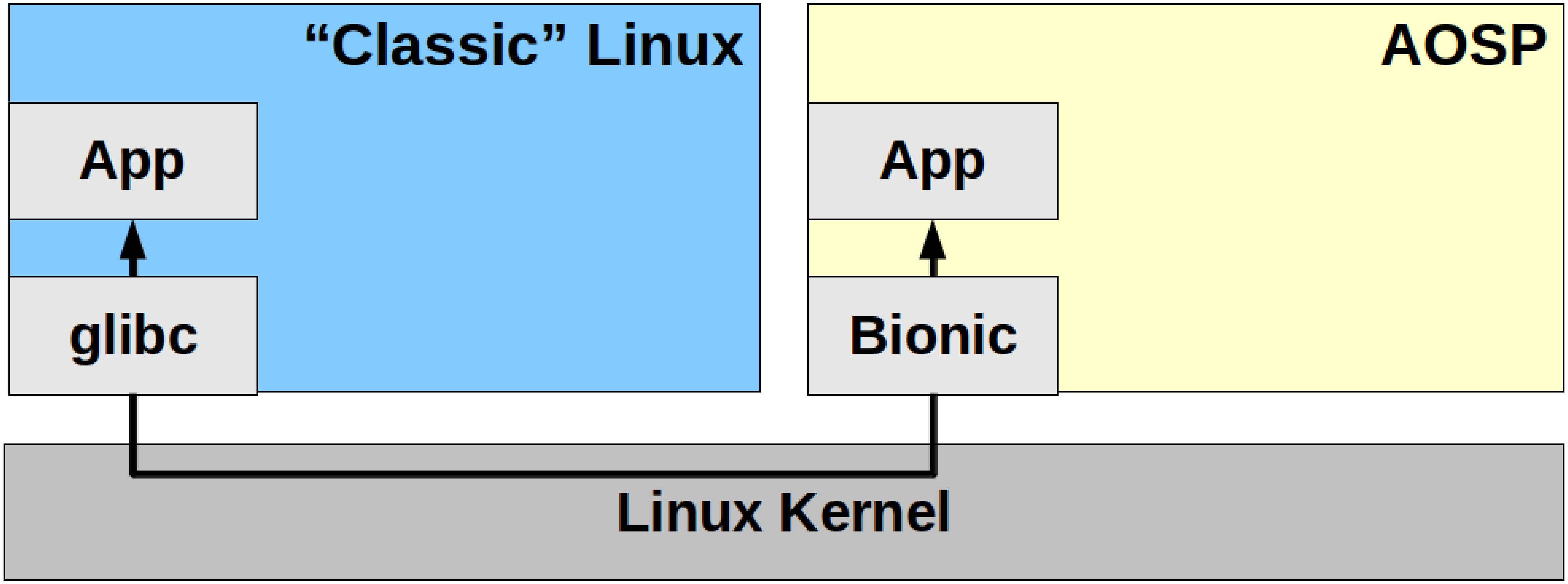
9. Demos

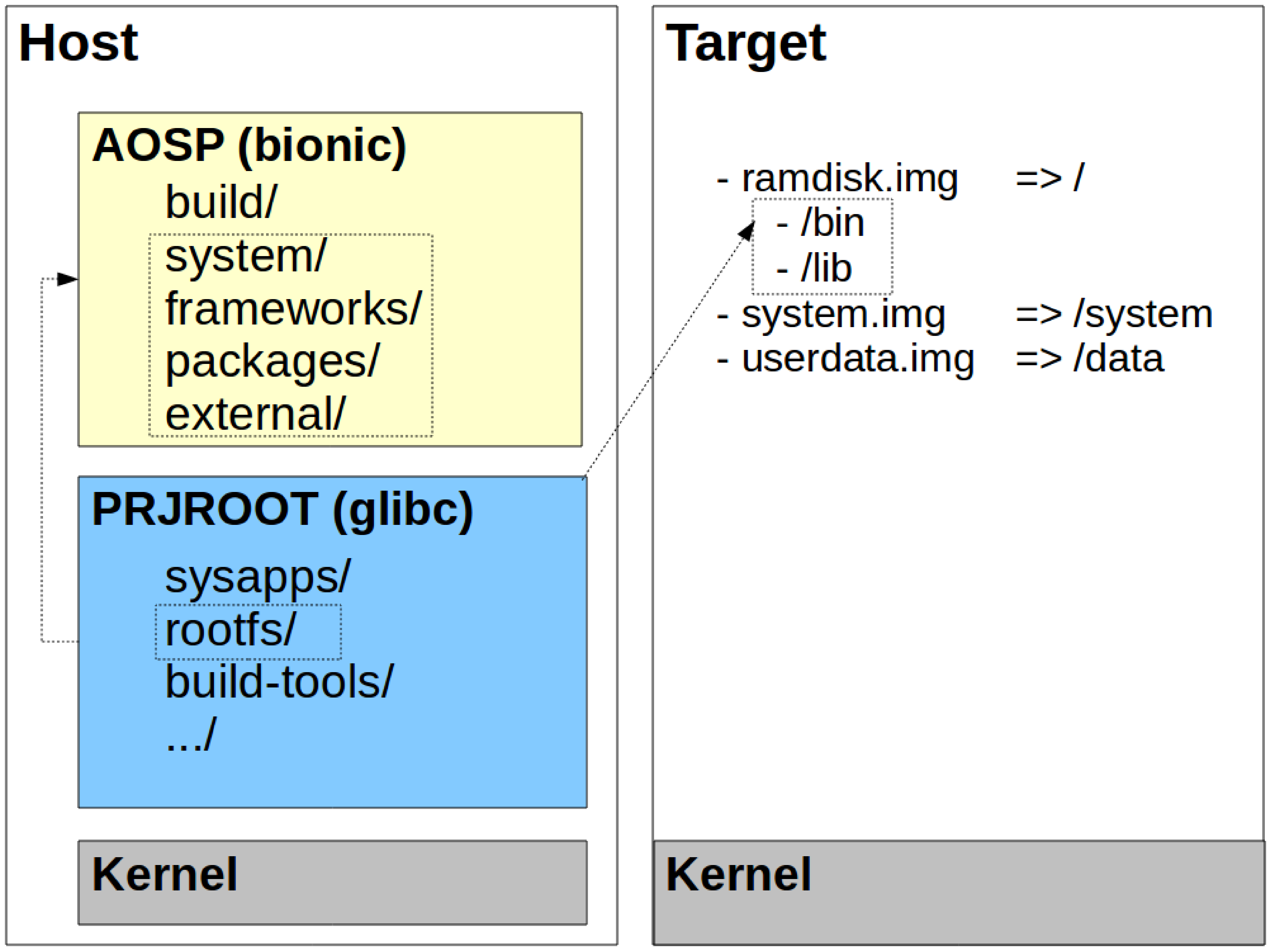
- All:
  - AOSP
  - BYO glibc-based rootfs
- Demo 1: BusyBox
- Demo 2: Client-Server app talking through socket
  - glibc client
  - bionic server
- Demo 3: Surprise!

9.1. Demo 1 - BusyBox

- Configure, build and “install” BusyBox
- Get it copied into final RAM disk image
- Modify AOSP to:
  - Make sure /lib/* is executable
  - Path start with “/bin”
  - adb shell is BusyBox, not Toolbox

9.2. Demo 2 - Architecture




Host
AOSP (bionic)
build/
system/
frameworks/
packages/
external/
PRJROOT (glibc)
sysapps/
rootfs/
build-tools/
.../
Kernel
Target
- ramdisk.img => /
- /bin
- /lib
- system.img => /system
- userdata.img => /data
Kernel

9.3. Demo 3 – LTT, the revenge

- Patch kernel
- Cross-build ltt-ctl
- Modify AOSP to log to LTT

Thank you ...

karim.yaghmour@opersys.com