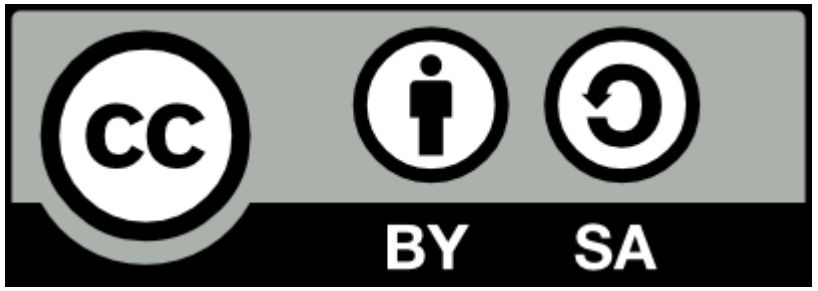
Leveraging Android's Linux Heritage

AnDevCon 2013

Karim Yaghmour
@karimyaghmour



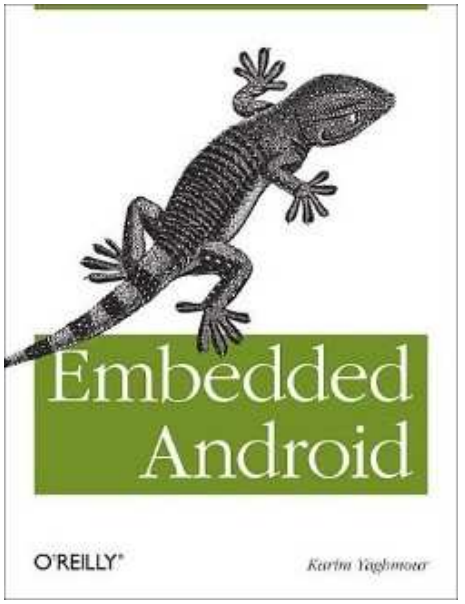
These slides created by: Karim Yaghmour

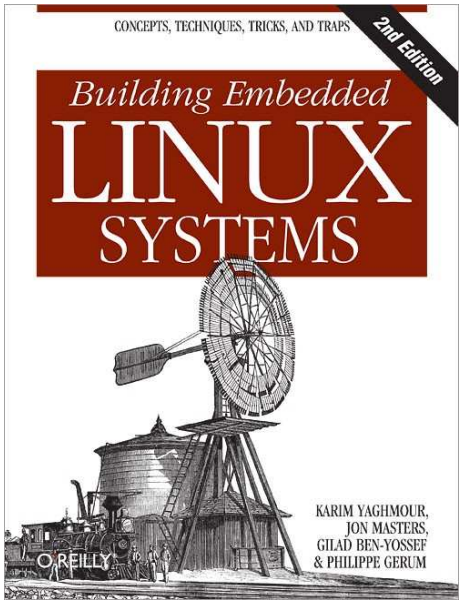
Originals at: www.opersys.com/community/docs

Delivered and/or customized by

About

- Author of:





- 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

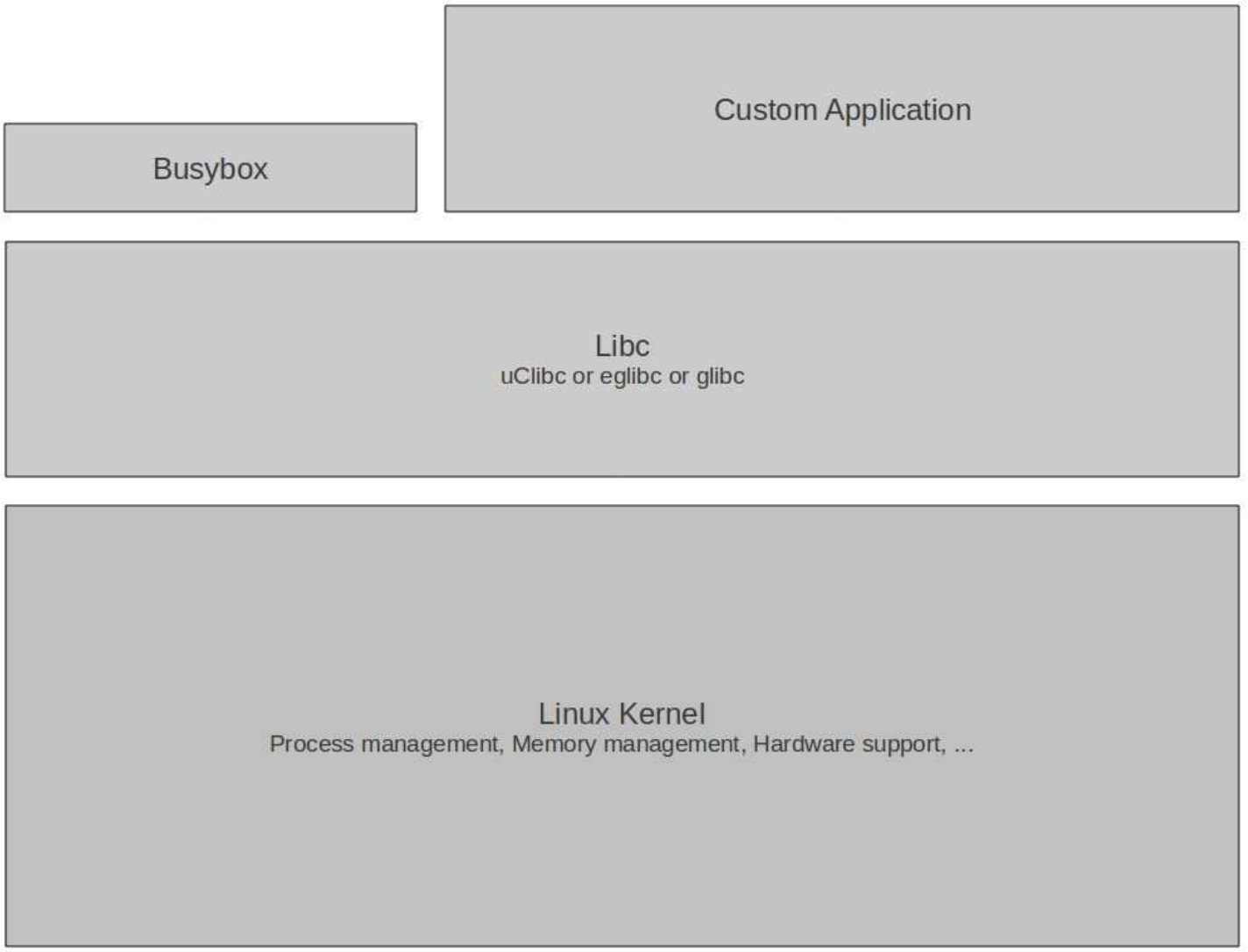
- ~~Opening as many cans of worms as possible~~
- Can “Linux” and Android Coexist and Interact?

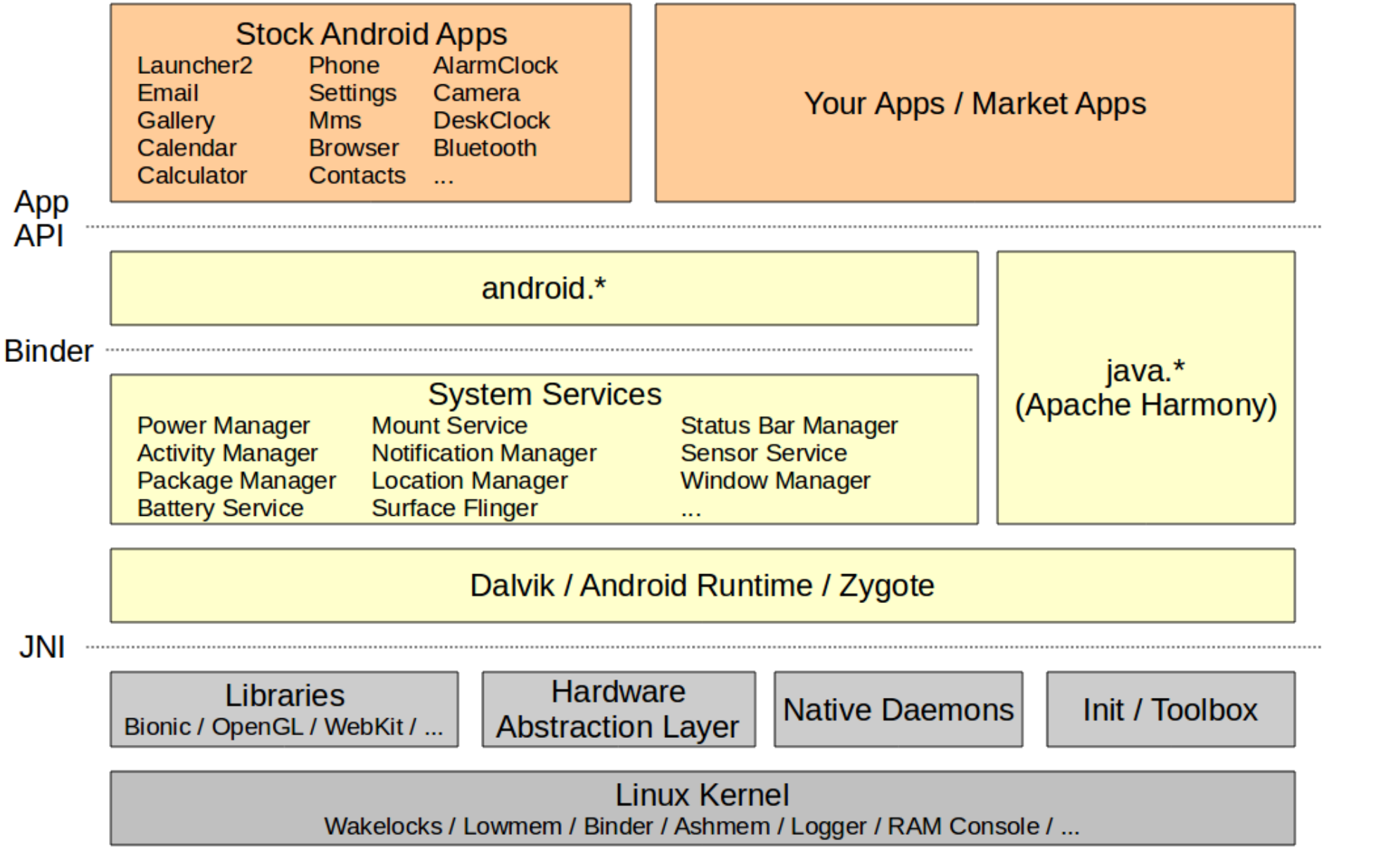
2. Rationale

- 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

- Overall Architecture EL
- Overall Architecture Android


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


Stock Android Apps
Launcher2 Phone AlarmClock
Email Settings Camera
Gallery Mms DeskClock
Calendar Browser Bluetooth
Calculator Contacts ...
Your Apps / Market Apps
App API
android.*
Binder
System Services
Power Manager Mount Service Status Bar Manager
Activity Manager Notification Manager Sensor Service
Package Manager Location Manager Window Manager
Battery Service Surface Flinger ...
java.*
(Apache Harmony)
Dalvik / Android Runtime / Zygote
JNI
Libraries
Bionic / OpenGL / WebKit / ...
Hardware Abstraction Layer
Native Daemons
Init / Toolbox
Linux Kernel
Wakelocks / Lowmem / Binder / Ashmem / Logger / RAM Console / ...

4. Roadblocks

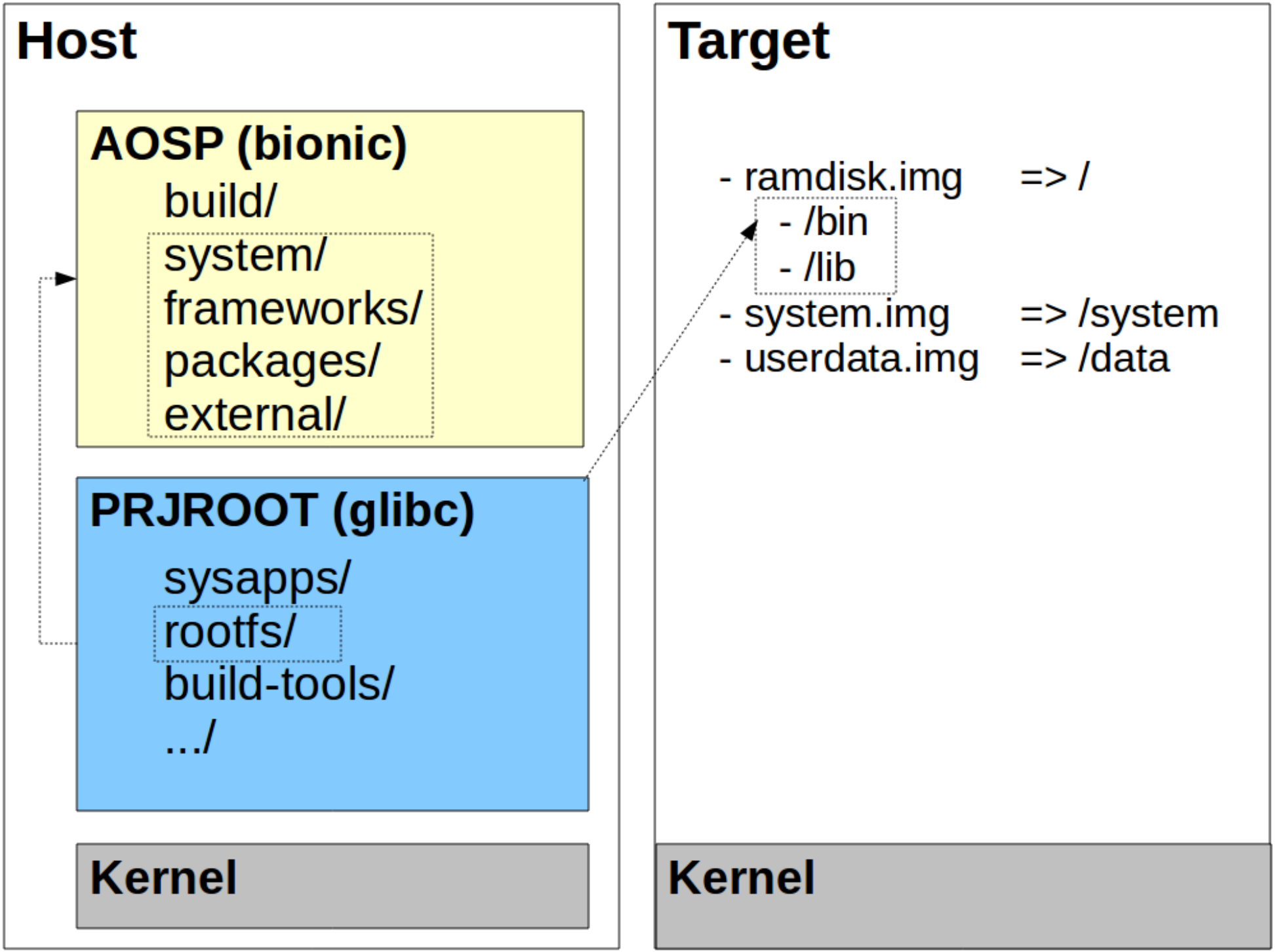
- Filesystem
  - Android is non-FHS-compliant
- C library
  - Bionic vs. glibc
- Interconnect fabric
  - Intents vs. DBUS
- IPC
  - Binder vs. Sockets and other std Unix IPC
- Display management
  - SurfaceFlinger vs. X
- I/O
  - Framebuffer, keyboard, mouse, disk, ...

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?


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

7. Been there done that

- BusyBox in CyanogenMod
- Gstreamer vs. Stagefright
- UML Android
- 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. Tools

- GNU cross-development toolchain:
  - gcc - compiler
  - as - assembler
  - ld - linker
  - gdb/gdbserver - debugger
  - etc.
- C library: uClibc, eglibc or glibc

10. Embedded Linux Workspace

- Need to organize the components used during cross-platform development. Workspace layout:

bootldr:	target bootloader (s)
build-tools:	toolchain build packages and sources
debug:	debugging tools
doc:	project documentation
images:	binary images ready to be used on target
kernel:	sources and build directories for target kernels
project:	your own custom code for the target
rootfs:	root filesystem as seen on the target
sysapps:	sources for target's system applications
tmp:	temporary data and experiments
tools:	toolchain and all other tools required to build software for the target.

10.1. Workspace env. vars. script

- Complete workspace script (devex)

```
export PROJECT=emblinux
export PRJROOT=/home/karim/${PROJECT}
export TARGET=arm-none-linux-gnueabi
export PATH=${PATH}:[CODESOURCERY_DIR]/bin
cd $PRJROOT
```

- To use this script:

```
$ . devex
```

- Possible values for $TARGET:
  - ARM: arm-linux, arm-unknown-linux-gnueabi
  - MIPS: mips-linux, mipsel-unknown-linux-gnu
  - I386: i386-linux, i586-geode-linux-uclibc

11. Basic root filesystem structure

- Unix FS structured for multi-user systems
- Some directories not necessary for embedded
- Filesystem Hierarchy Standard (FHS):
  - /bin => Essential user binaries
  - /boot => Bootloader and kernel images
  - /dev => Device files
  - /etc => System configuration
  - /home => User home directories
  - /lib => Essential shared libs and kernel modules
  - /mnt => Temporary mount point
  - /opt => Add-on software packages
  - /sbin => Essential system binaries
  - /tmp => Temporary files
  - /usr => Secondary hierarchy (mostly user apps)
  - /var => Variable data generated by daemons

- Non-essential multi-user dirs:
  - /home, /mnt, /opt, /root
- Depends on bootloader:
  - /boot
- ***Traditionally*** “essential”:
  - /bin, /dev, /etc, /lib, /proc, /sbin, /usr, /tmp, /var
- Careful with “/etc”, Android needs it to point to “/system/etc” for Dbus config ... Just hack it.
- Contain their own hierarchy:
  - /usr, /var

- What are all these binaries directories for?
  - /bin => Essential binaries for user and admin
  - /sbin => Essential binaries for admin
  - /usr/bin => Non-essential user and admin binaries
  - /usr/sbin => Non-essential admin binaries
- What are all those libraries directories for?
  - /lib => Essential system libraries
  - /usr/lib => Non-essential libraries
- The kernel does not force FS layout. Layout is “universally” agree upon (i.e. FHS.)

- To start working on rootfs:
  ```
  $ cd ${PRJROOT}/rootfs
  ```
- Create core rootfs directories:
  ```
  $ mkdir bin lib sbin usr var
  ```
- Create the /usr hierarchy:
  ```
  $ mkdir usr/{bin,lib,sbin}
  ```
- Create the /var hierarchy:
  ```
  $ mkdir var/{lib,lock,log,run,tmp}
  $ chmod 1777 var/tmp
  ```

12. Libraries

1. glibc
2. uClibc

12.1. glibc

- glibc components:
  - Actual shared libraries:
    - Format: libLIB_NAME-GLIBC_VER.so
    - Examples: libm-2.3.2.so, libc-2.3.2.so
  - Major revision version symbolic links:
    - Format: libLIB_NAME.so.MAJOR_REV_VER
    - Examples: libdl.so.2, libc.so.6
  - Version-independent symbolic links to the major revision version symbolic links:
    - Format: libLIB_NAME.so
    - Examples: libdl.so, libm.so
  - Static library archives:
    - Format: libLIB_NAME.a
    - Examples: libdl.a, libm.a

- For target, need:
  - The actual shared libs
  - The major revision version symbolic links
- Also need dynamic linker:
  - Actual linker: ld-GLIBC_VER.so
  - Symbolic link to linker:
    - x86, ARM, SH, m68k => ld-linux.so.MAJOR_REV_VER
    - MIPS, PPC => ld.so.MAJOR_REV_VER
- Must determine exact library components required.
- BELS table 6.2 contains complete list

- Most important components:
  - ld => the dynamic linker
  - libc => the C library
  - libm => the math library
  - libdl => the shared objects manipulation library
- Must determine exact dependencies of your applications.
- Native `ldd` is not cross-platform-capable
- Can use `readelf` or `uclibc-ldd`:

- Copying all libraries:

```
$ cp -d [CODESOURCERY_DIR]/arm-none-linux-gnueabi/libc/lib/* \
> ${PRJROOT}/rootfs/lib
```

- Stripping all target libraries for space efficiency:

```
$ arm-none-linux-gnueabi-strip ${PRJROOT}/rootfs/lib/*.so*
```

12.2. uClibc

- Same naming conventions as glibc
- Implements most of the glibc components:
  - ld, libc, libcrypt, libdl, libm, libpthread, libresolv, libutil.
- uClibc libraries can coexist with glibc libraries in target's /lib directory.
- Copying all uClibc components:

```
$ cd ${PREFIX}/uclibc/lib
$ cp *-*.so ${PRJROOT}/rootfs/lib
$ cp -d *.so.[*0-9] ${PRJROOT}/rootfs/lib
```

- No need to strip uClibc libraries, they are stripped by the uClibc build script

13. Main system applications

- Unix systems rely on a common set of commands
- Standard distros have one binary per command
- May compile each relevant command one-by-one or use packages that provide many commands in a single binary:
  1. Busybox
  2. Distro

13.1. BusyBox

- Main package used in embedded Linux to provide core set of Unix commands: busybox.net

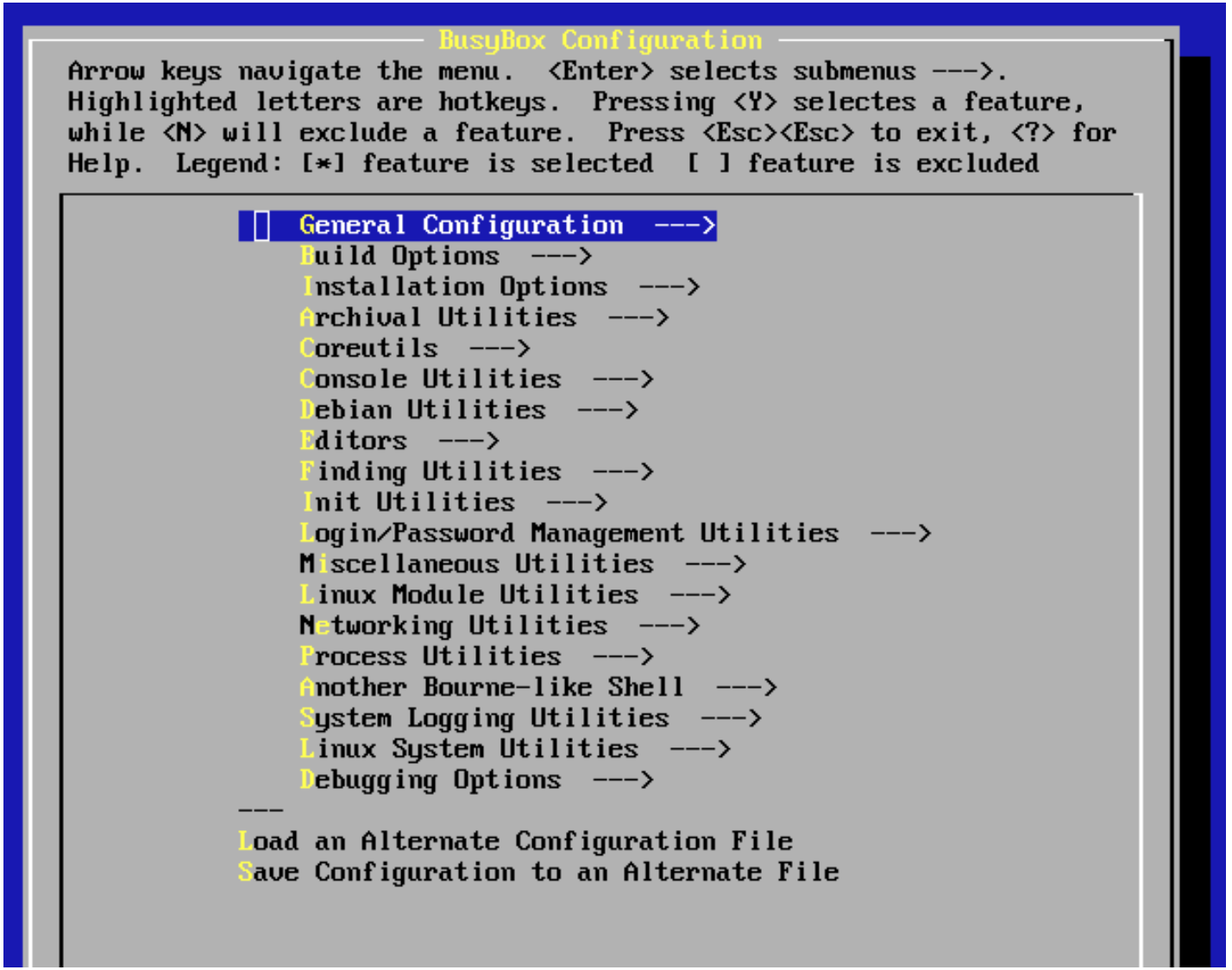
[, [[, acpid, add-shell, addgroup, adduser, adjtimex, arp, arping, ash, awk, base64, basename, beep, blkid, blockdev, bootchartd, brctl, bunzip2, bzcat, bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devmem, df, dhcprelay, diff, dirname, dmesg, dnsd, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, ed, egrep, eject, env, envdir, envuidgid, ether-wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fgconsole, fgrep, find, findfs, flock, fold, free, freeramdisk, fsck, fsck.minix, fsync, ftpd, ftpget, ftpput, fuser, getopt, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifplugd, ifup, inetd, init, insmod, install, ionice, iostat, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, last, length, less, linux32, linux64, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs, makemime, man, md5sum, mdev, mesg, microcom, mkdir, mkdosfs, mke2fs, mkfifo, mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, mkpasswd, mkswap, mktemp, modinfo, modprobe, more, mount, mountpoint, mpstat, mt, mv, nameif, nbd-client, nc, netstat, nice, nmeter, nohup, nslookup, ntpd, od, openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, pmap, popmaildir, poweroff, powertop, printenv, printf, ps, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, reformime, remove-shell, renice, reset, resize, rev, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script, scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont, setkeycodes, setlogcons, setsid, setuidgid, sh, sha1sum, sha256sum, sha512sum, showkey, slattach, sleep, smemcap, softlimit, sort, split, start-stop-daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, timeout, top, touch, tr, traceroute, traceroute6, true, tty, ttysize, tunctl, udhcpc, udhcpd, udpsvd, umount, uname, unexpand, uniq, unix2dos, unlzma, unlzop, unxz, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, volname, wall, watch, watchdog, wc, wget, which, who, whoami, xargs, xz, xzcat, yes, zcat, zcip

- Download BusyBox (1.18.3) to your ${PRJROOT}/sysapps directory and extract it there.
- Move to the directory for the rest of the setup:

```
$ cd ${PRJROOT}/sysapps/busybox-1.18.3
```

- Configuration of BusyBox's options:

```
$ make menuconfig
```


BusyBox Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help. Legend: [*] feature is selected [ ] feature is excluded
General Configuration --->
Build Options --->
Installation Options --->
Archival Utilities --->
Coreutils --->
Console Utilities --->
Debian Utilities --->
Editors --->
Finding Utilities --->
Init Utilities --->
Login/Password Management Utilities --->
Miscellaneous Utilities --->
Linux Module Utilities --->
Networking Utilities --->
Process Utilities --->
Another Bourne-like Shell --->
System Logging Utilities --->
Linux System Utilities --->
Debugging Options --->

Load an Alternate Configuration File
Save Configuration to an Alternate File

- “Busybox Settings”:
  - “Build Options” -> Cross-compiler prefix:

    `${TARGET}-`
  - “Installation Options” -> Installation prefix:

    `${PRJROOT}/rootfs`
- Build:

  `$ make`
- Install:

  `$ make install`

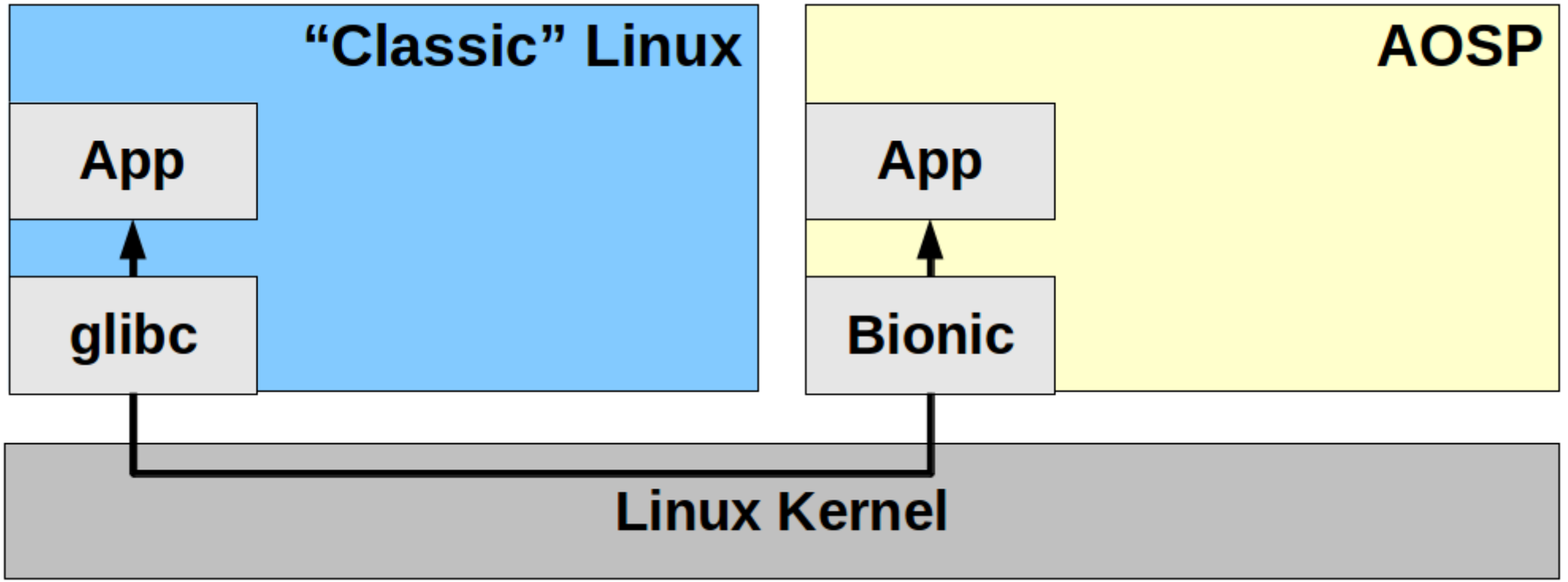
14. Demos

- Demo 1 and 2:
  - AOSP
  - BYO glibc-based rootfs
- Demo 1: BusyBox
- Demo 2: Client-Server app talking through socket
  - glibc client
  - bionic server
- Demo 3: Buildroot integration
- Demo 4: systrace/atrace/ftrace

14.1. 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

14.2. Demo 2 - Architecture



14.3. Demo 3 - Buildroot integration

14.4. Demo 4 – systrace / atrace / ftrace

- ftrace is built into the kernel
- Use systrace/atrace to collect data
- Use Chrome browser to view data

Thank you ...

karim.yaghmour@opersys.com