Facilitating Android Custom ROM Development with Kexec

Mike Kasick

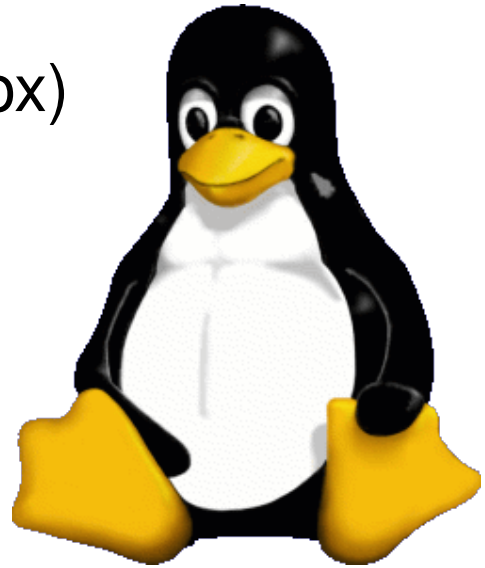
Outline

What is Android?

- “Google’s” open-source mobile operating system
- Developed by the Open Handset Alliance (OHA)
  - Google, HTC, Motorola, Qualcomm, TI, Samsung, 78 others
- Very popular: 68% smartphone OS market share (2012 Q2)
- Runs on phones (smart & feature), tablets, ereaders, etc.

Android is Linux (but not GNU)

- Uses a modified Linux kernel
- Features a highly-customized user space
- **Bionic** (libc): BSD-based with Linux support
- **Toolbox**: shell tools, omnibus binary (BusyBox)
- **Dalvik**: register-based VM, “runs apps”
- **Framework**: user interface layer, middleware



Android is “Open Source”

- Components released under various licenses
  - Kernel (Linux) and associated utilities are GPLv2
  - Certain libraries (e.g., WebKit) are GPL or LGPL
  - Bionic & Toolbox are (three-clause) BSD
  - Everything else (Dalvik, framework) is Apache v2
- Full platform sources are available (only) for Nexus devices
- Third-party manufacturers:
  - Use proprietary kernel modules, HAL libraries, custom UIs
  - Obligated to release kernel source code
  - Generally do not release non-GPL platform sources

Android Community Development

- Manufacturer-based custom ROMs
  - Kernels built from manufacturer sources
  - ROM code patches (Dalvik assembly), themed GUIs
  - Generally aim to support a single device

Android Community Development

- Manufacturer-based custom ROMs
  - Kernels built from manufacturer sources
  - ROM code patches (Dalvik assembly), themed GUIs
  - Generally aim to support a single device

- Source-based: The Android Open Source Project (AOSP)
  - OHA's portion of Android they're willing to open source
  - Contains full support for Nexus devices
  - Follow source.android.com directions, build a bootable ROM

Android Community Development

- Manufacturer-based custom ROMs
  - Kernels built from manufacturer sources
  - ROM code patches (Dalvik assembly), themed GUIs
  - Generally aim to support a single device
- Source-based: The Android Open Source Project (AOSP)
  - OHA's portion of Android they're willing to open source
  - Contains full support for Nexus devices
  - Follow source.android.com directions, build a bootable ROM
- CyanogenMod: Community-developed Android distribution
  - Supports Nexus and third-party devices (137 officially)
  - Significant reverse engineering and reimplementation effort
  - CM9 supports 69 devices from a single source tree
  - Basis for many source-based offshoot ROMs

Outline

Application Development Summary

- Developer site: developer.android.com
  - Android SDK available for Windows, Mac, and Linux
  - Lots of documentation: Guides, APIs, examples, etc.

Application Development Summary

- Developer site: developer.android.com
  - Android SDK available for Windows, Mac, and Linux
  - Lots of documentation: Guides, APIs, examples, etc.
- Primary programming language is “Java”
  - Java syntax and semantics, but API is mostly custom
  - JDK-produced .class files translated to Dalvik .dex format
  - C & C++ native code supported with Android NDK

Application Development Summary

- Developer site: developer.android.com
  - Android SDK available for Windows, Mac, and Linux
  - Lots of documentation: Guides, APIs, examples, etc.
- Primary programming language is “Java”
  - Java syntax and semantics, but API is mostly custom
  - JDK-produced .class files translated to Dalvik .dex format
  - C & C++ native code supported with Android NDK
- Unencumbered development platform
  - Most devices support testing/debugging out of the box
  - Distribute apps through Google Play for one-time $25 fee
  - Open ecosystem: third-party stores, publish independently

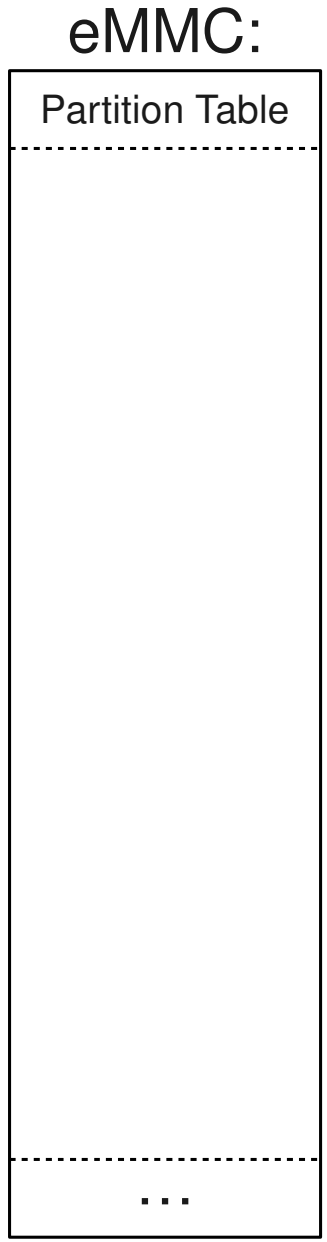
Android Debug Bridge (adb)

- Tool and protocol to communicate with devices over USB
  - No need for serial console, setting up Ethernet and ssh, etc.
- Supports a variety of useful commands
  - `adb backup` / `restore`: backup or restore device contents
  - `adb forward`: forward host network sockets
  - `adb install`: upload and install an app
  - `adb logcat`: dump application log messages
  - `adb push` / `pull`: send files to, or receive files from device
  - `adb shell`: open a shell, or invoke a shell command
- Powerful development tool, kudos for making available

Outline

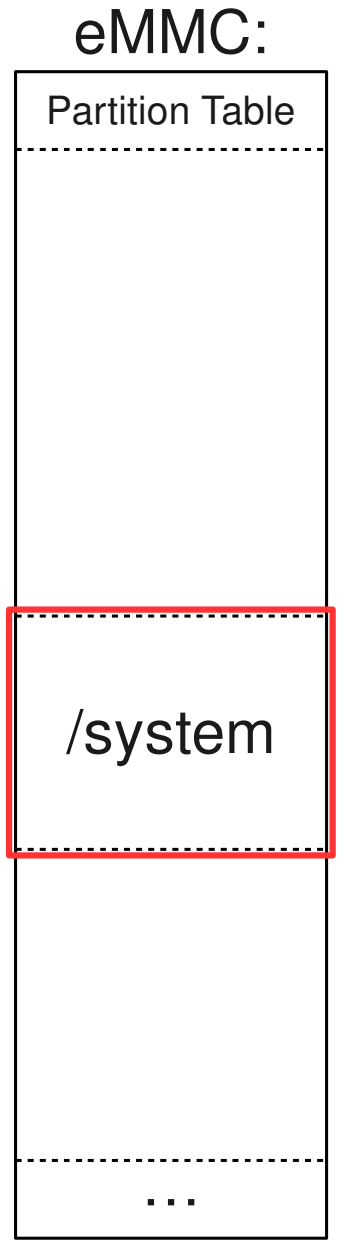
Typical Storage Setup (I)

- eMMC (flash) storage divided into partitions



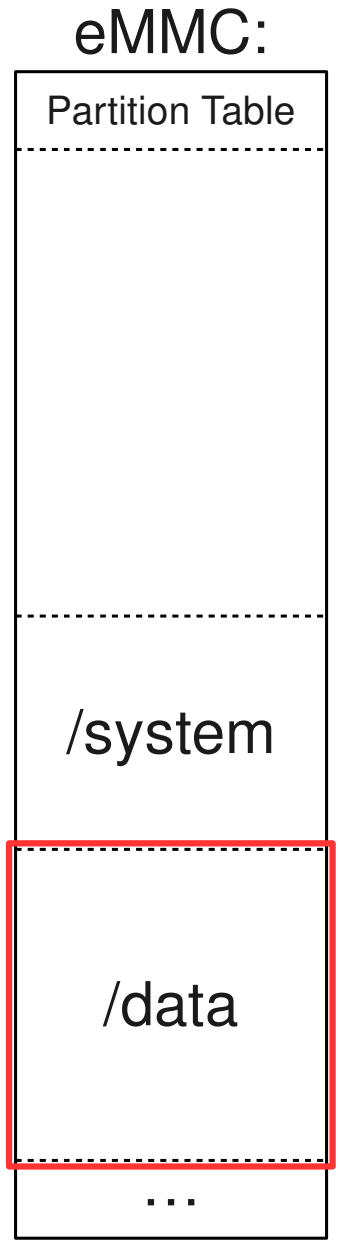
Typical Storage Setup (I)

- eMMC (flash) storage divided into partitions
- /**system** partition
  - Read-only, contains OS and preinstalled apps



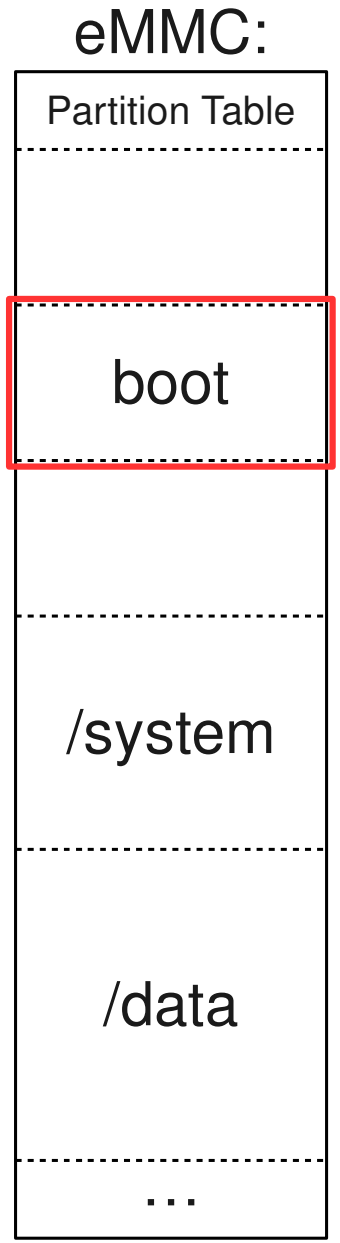
Typical Storage Setup (I)

- eMMC (flash) storage divided into partitions
- /**system** partition
  - Read-only, contains OS and preinstalled apps
- /**data** partition
  - Writable, contains user-installed apps, media
  - “Factory data reset” wipes entire partition, restores device to factory state



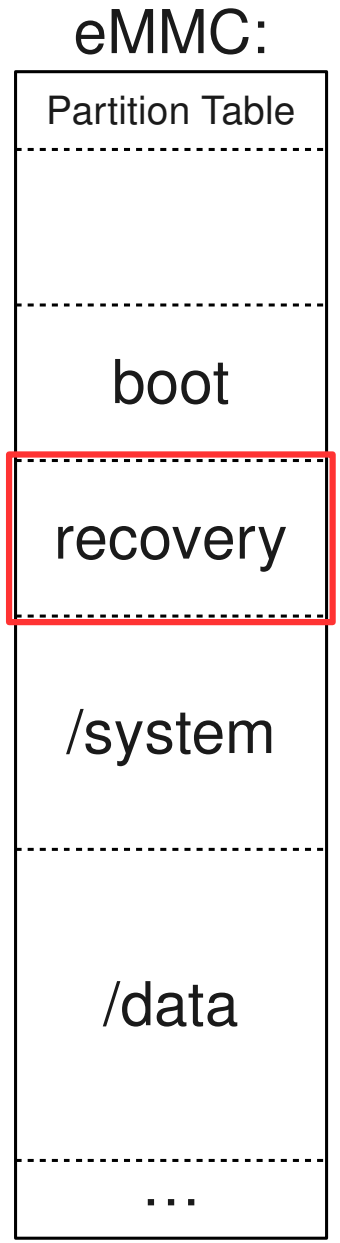
Typical Storage Setup (II)

- **boot** partition
  - Contains boot kernel and initrd
  - Android boot.img format
  - initrd: “/”, no `pivot_root`, mounts partitions



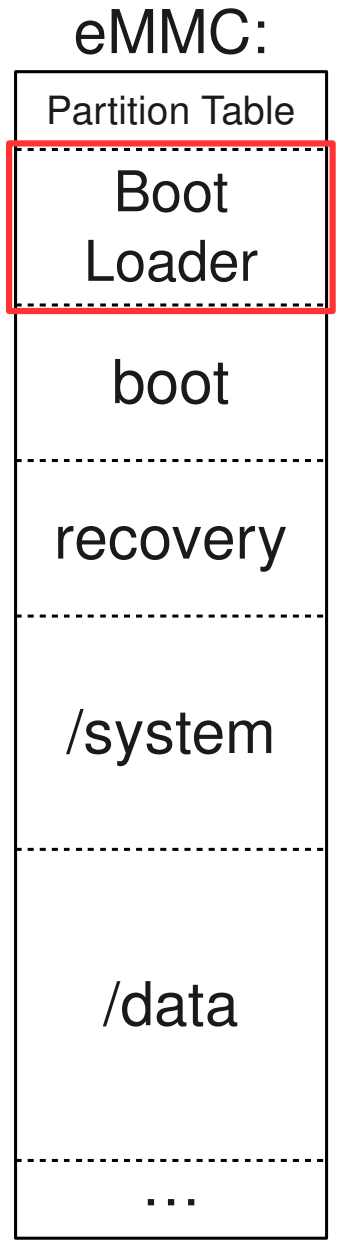
Typical Storage Setup (II)

- **boot** partition
  - Contains boot kernel and initrd
  - Android boot.img format
  - initrd: “/”, no `pivot_root`, mounts partitions
- **recovery** partition
  - Contains recovery kernel and initrd
  - Minimal boot environment (“Safe Mode”)
  - For applying system updates (update.zips)
  - Can wipe /data, `adb shell`, etc.



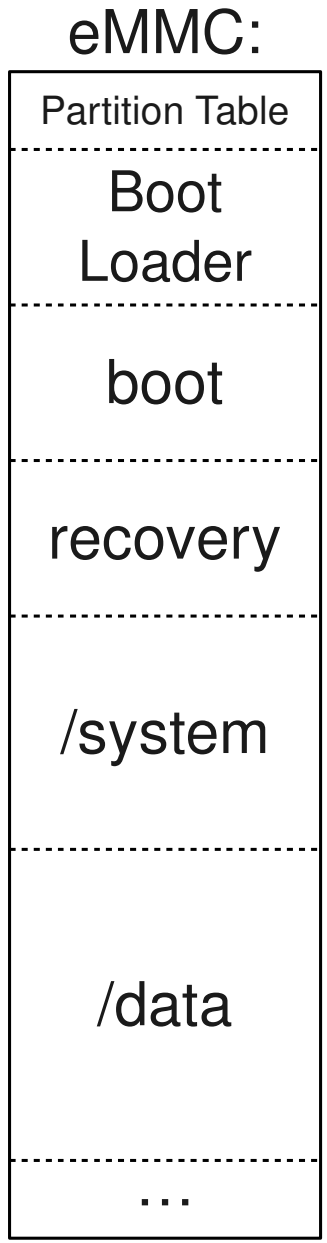
Typical Storage Setup (III)

- Boot loader partitions
  - Contain the device's boot loader



Typical Storage Setup (III)

- Boot loader partitions
  - Contain the device's boot loader
- Boot loader
  - Loads **boot** or **recovery** kernels, jumps to them
  - Special mode to flash partitions over USB
  - No single "Android boot loader", varies by OEM



Fastboot

- adb for boot loaders
  - Tool and protocol to communicate with boot loader over USB
  - `fastboot flash`: upload image and flash to partition
  - `fastboot boot`: upload kernel image to RAM, boot directly

Fastboot

- adb for boot loaders
  - Tool and protocol to communicate with boot loader over USB
  - `fastboot flash`: upload image and flash to partition
  - `fastboot boot`: upload kernel image to RAM, boot directly
- Another powerful tool, kudos for making available
  - But many OEM boot loaders don't support it

Fastboot

- adb for boot loaders
  - Tool and protocol to communicate with boot loader over USB
  - `fastboot flash`: upload image and flash to partition
  - `fastboot boot`: upload kernel image to RAM, boot directly
- Another powerful tool, kudos for making available
  - But many OEM boot loaders don't support it
- Alternative: proprietary flashing protocols
  - Implemented by most boot loaders, even non-fastboot ones
  - Used to field reflash devices with broken partitions
  - Ex: Odin (Samsung), RSD (Motorola), nvflash (Nvidia)

Storage & Boot Loader Problems

- Inflexible setup
  - Only one set of **boot**, /**system**, and /**data** partitions
  - Switching ROMs requires flashing many partitions
  - Appropriate for an appliance, not platform development

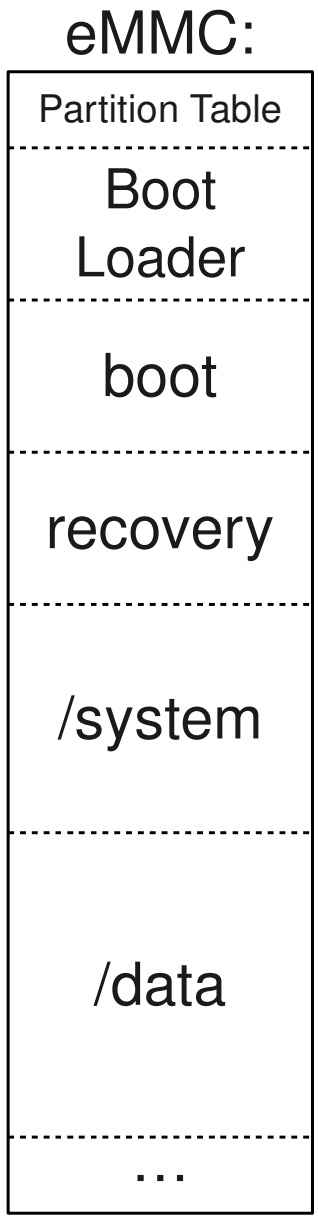
Storage & Boot Loader Problems

- Inflexible setup
  - Only one set of **boot**, /**system**, and /**data** partitions
  - Switching ROMs requires flashing many partitions
  - Appropriate for an appliance, not platform development
- "`fastboot boot`" great for kernel testing
  - Has limited device support
  - Subtle bugs on some devices (broken USB)

Storage & Boot Loader Problems

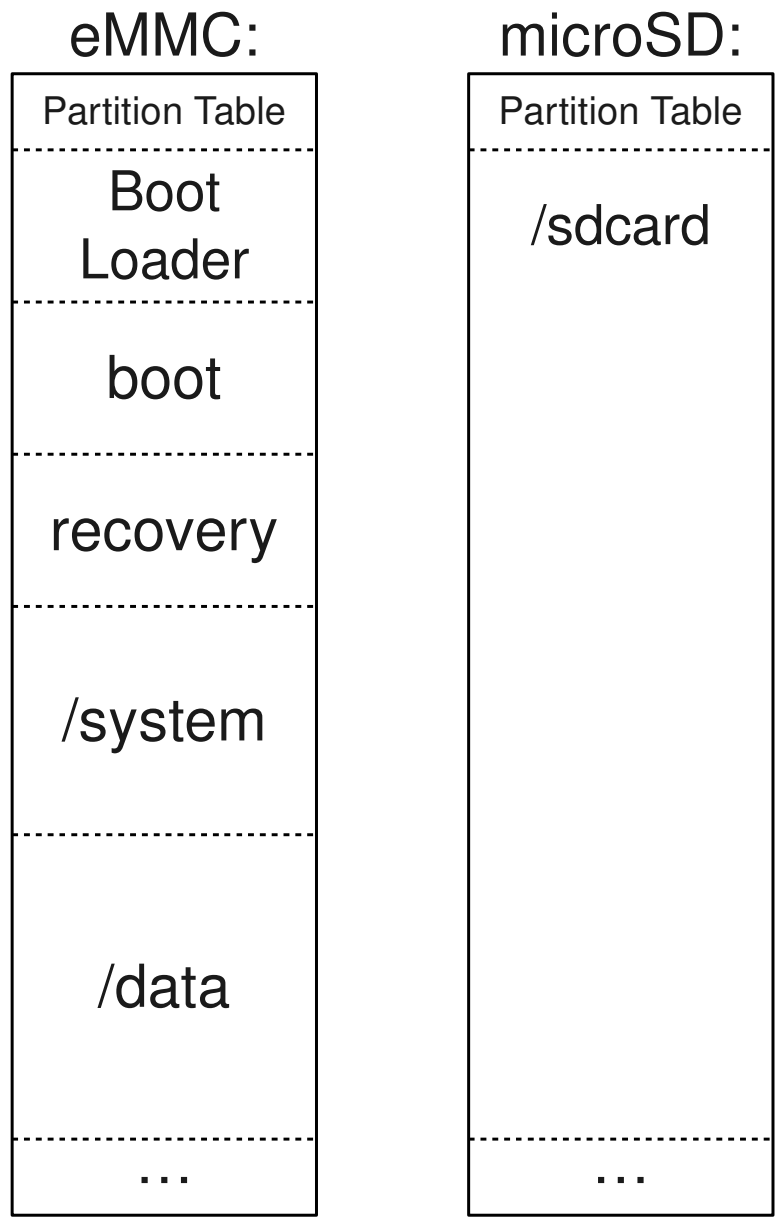
- Inflexible setup
  - Only one set of **boot**, /**system**, and /**data** partitions
  - Switching ROMs requires flashing many partitions
  - Appropriate for an appliance, not platform development
- "`fastboot boot`" great for kernel testing
  - Has limited device support
  - Subtle bugs on some devices (broken USB)
- Development & testing is inefficient
  - Lots of time spent reflashing device
  - Many erase cycles, wear on eMMC
  - **Dangerous!** If no way to recover from bad flash

Solution: Dual-boot



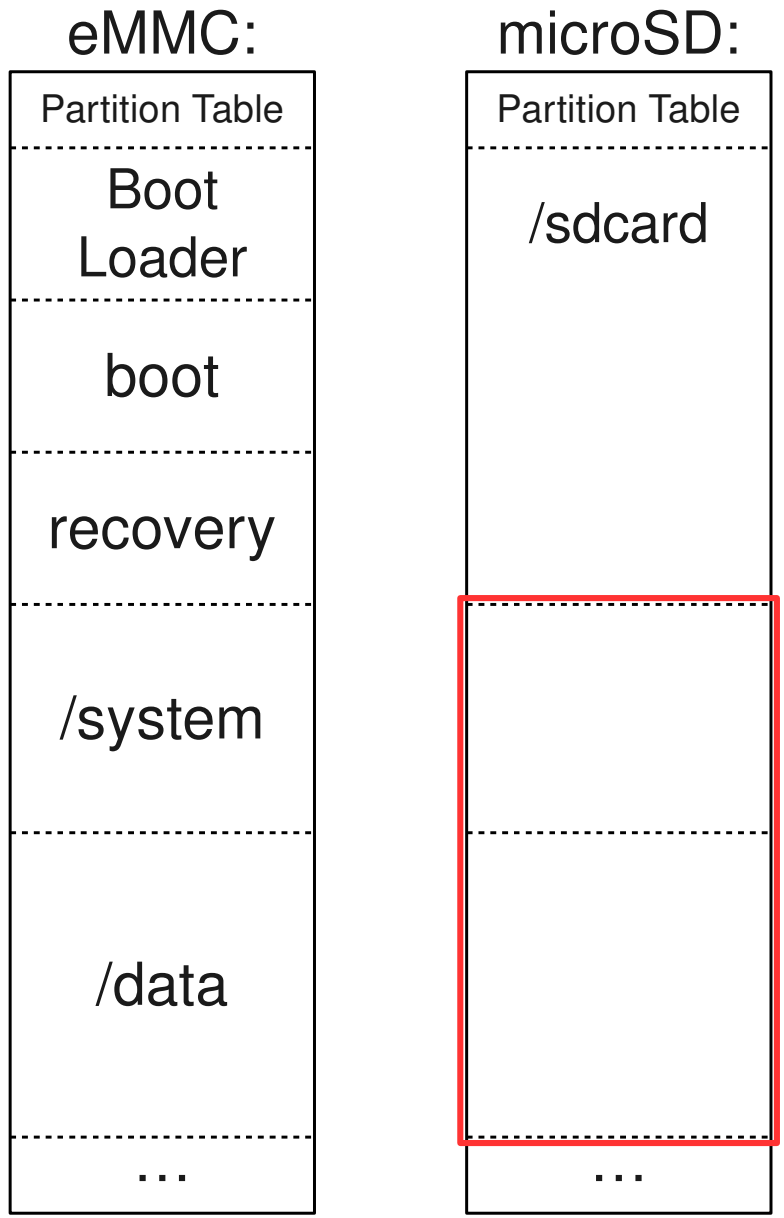
Solution: Dual-boot

- Place alternate ROM on microSD



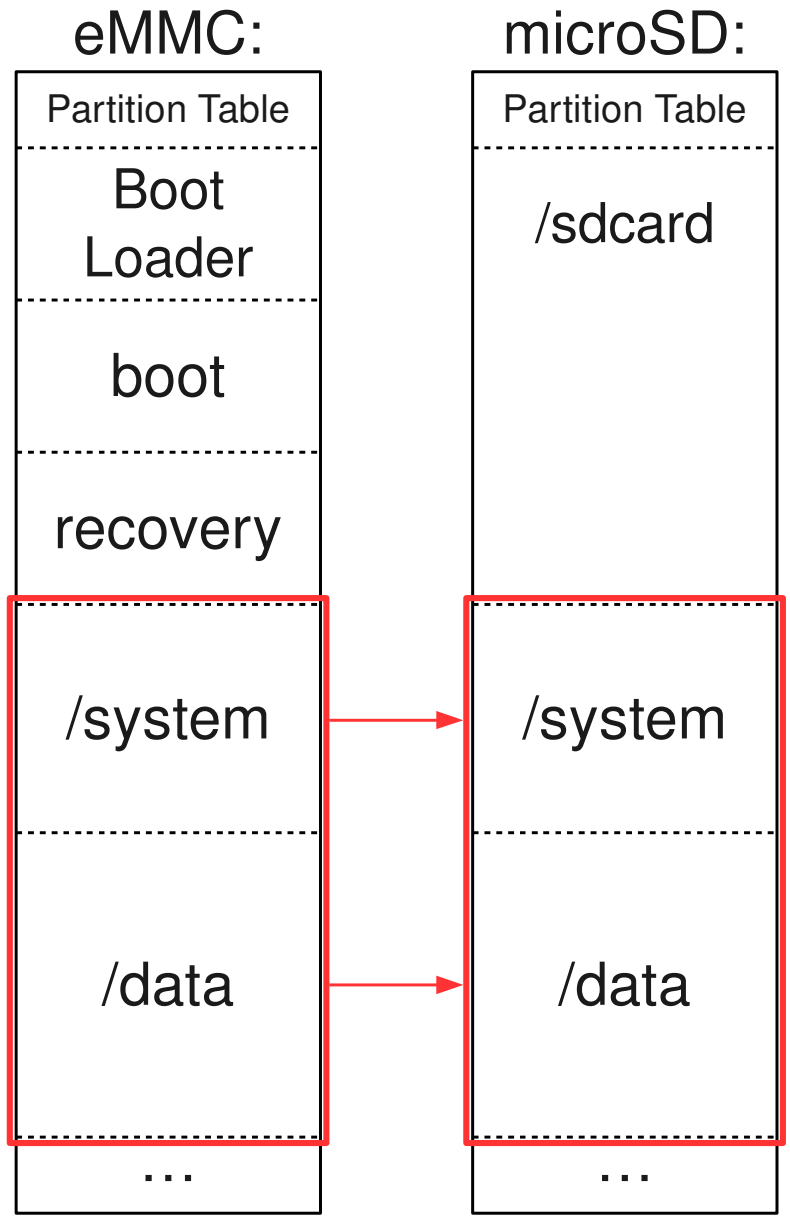
Solution: Dual-boot

- Place alternate ROM on microSD
- Carve out two new partitions



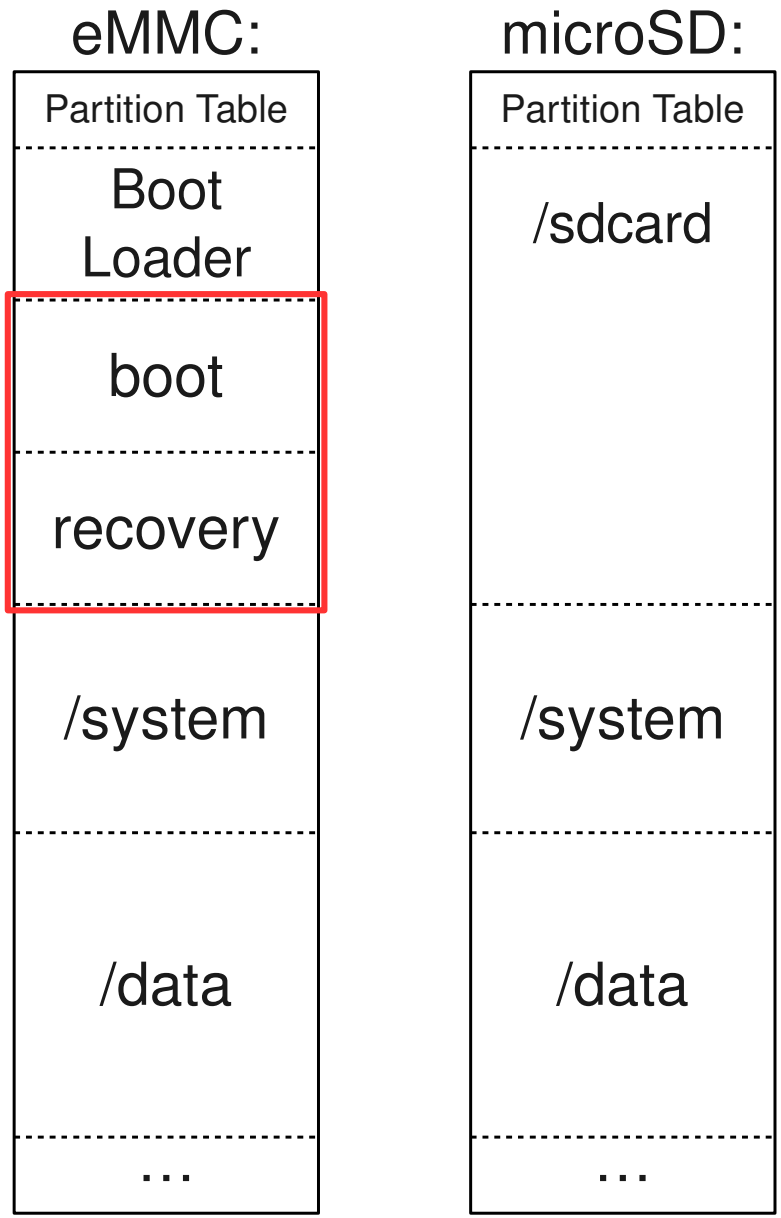
Solution: Dual-boot

- Place alternate ROM on microSD
- Carve out two new partitions
- Shadow /**system** and /**data**



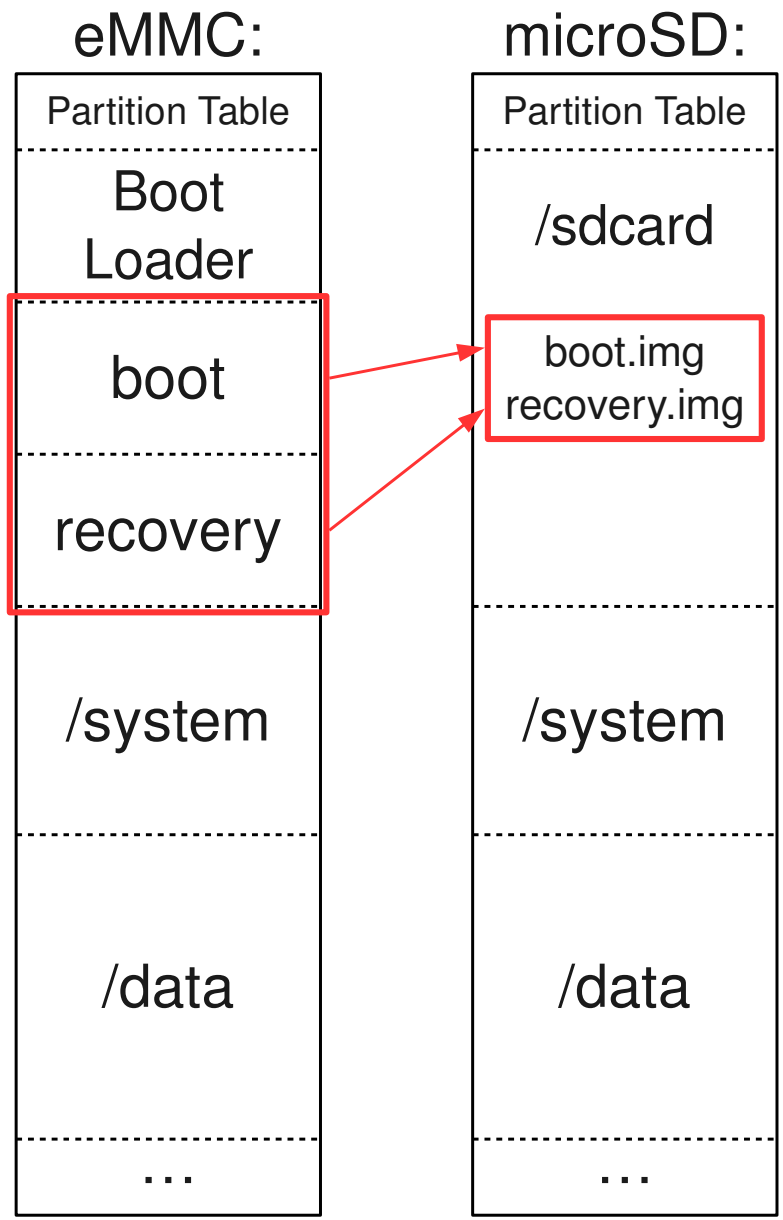
Solution: Dual-boot

- Place alternate ROM on microSD
- Carve out two new partitions
- Shadow /**system** and /**data**
- Change **boot** & **recovery** fstabs



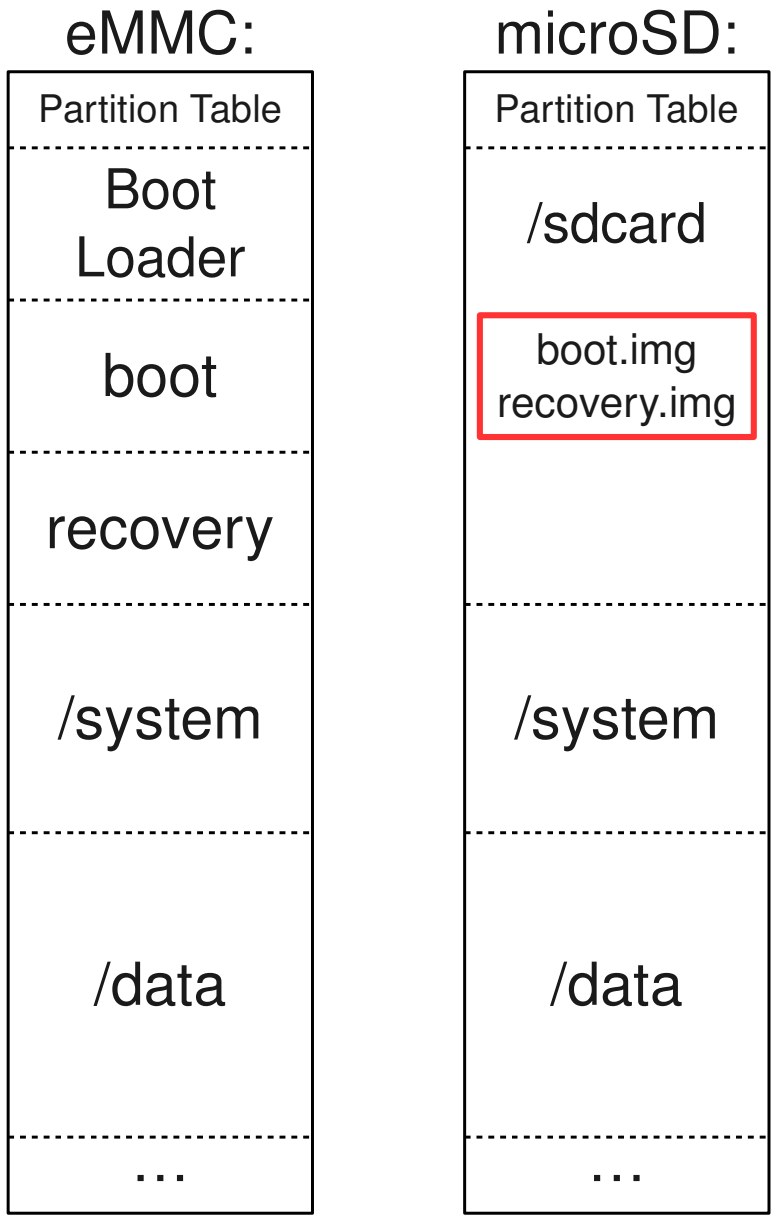
Solution: Dual-boot

- Place alternate ROM on microSD
- Carve out two new partitions
- Shadow /**system** and /**data**
- Change **boot** & **recovery** fstabs
- Place kernel **.img**s on /**sdcard**



Solution: Dual-boot

- Place alternate ROM on microSD
- Carve out two new partitions
- Shadow /**system** and /**data**
- Change **boot** & **recovery** fstabs
- Place kernel **.img**s on /**sdcard**
- How to boot /**sdcard/boot.img**?



Outline

Kexec

- Linux system call to load and boot another kernel
  - Kernel version of `exec` syscall
  - Enables Linux to serve as its own boot loader

Kexec

- Linux system call to load and boot another kernel
  - Kernel version of `exec` syscall
  - Enables Linux to serve as its own boot loader
- `kexec` user-space tool, uses a two-part invocation

Kexec

- Linux system call to load and boot another kernel
  - Kernel version of `exec` syscall
  - Enables Linux to serve as its own boot loader
- `kexec` user-space tool, uses a two-part invocation
- ```
  kexec -l --append="$(cat /proc/cmdline)" \
           --initrd=ramdisk.gz zImage
  ```
  - Loads kernel `zImage` and initrd `ramdisk.gz` in memory
  ```

Kexec

- Linux system call to load and boot another kernel
  - Kernel version of `exec` syscall
  - Enables Linux to serve as its own boot loader
- `kexec` user-space tool, uses a two-part invocation
- ```
  kexec -l --append="$(cat /proc/cmdline)" \
           --initrd=ramdisk.gz zImage
  ```
  - Loads kernel `zImage` and initrd `ramdisk.gz` in memory
- `kexec -e`
  - Boots into the loaded kernel
  ```

Kexec procedure

- Prepares for a typical reboot
  - Calls `.shutdown` method for each device
  - Stops all other CPUs
  - Disables IRQs, caches, MMU, etc.
- But, instead of performing a hardware reboot
  - Stages kernel at load address
  - Jumps to kernel entry point
  - New kernel begins to boot

The Kexec Problem

- Driver `.shutdown` methods must be `.probe` compatible
  - `.shutdown` is called when system halts (power-off or reboot)
  - `.probe` is called (on boot) when device is initialized
  - In kexec, `.probe` is called after `.shutdown`, never otherwise

The Kexec Problem

- Driver `.shutdown` methods must be `.probe` compatible
  - `.shutdown` is called when system halts (power-off or reboot)
  - `.probe` is called (on boot) when device is initialized
  - In kexec, `.probe` is called after `.shutdown`, never otherwise
- Platform-specific drivers often fail to reinitialize devices
  - Drivers often broken unless specifically tested with kexec
  - Drivers assume devices initialized by boot loader, power-on
  - Ex: Serial UART blocks on “Uncompressing Linux...”

The Kexec Problem

- Driver `.shutdown` methods must be `.probe` compatible
  - `.shutdown` is called when system halts (power-off or reboot)
  - `.probe` is called (on boot) when device is initialized
  - In kexec, `.probe` is called after `.shutdown`, never otherwise
- Platform-specific drivers often fail to reinitialize devices
  - Drivers often broken unless specifically tested with kexec
  - Drivers assume devices initialized by boot loader, power-on
  - Ex: Serial UART blocks on “Uncompressing Linux...”
- Secondary CPUs may not be recoverable

The Kexec Problem

- Driver `.shutdown` methods must be `.probe` compatible
  - `.shutdown` is called when system halts (power-off or reboot)
  - `.probe` is called (on boot) when device is initialized
  - In kexec, `.probe` is called after `.shutdown`, never otherwise
- Platform-specific drivers often fail to reinitialize devices
  - Drivers often broken unless specifically tested with kexec
  - Drivers assume devices initialized by boot loader, power-on
  - Ex: Serial UART blocks on “Uncompressing Linux...”
- Secondary CPUs may not be recoverable
- Problems can be fixed, requires deep hardware knowledge

Outline

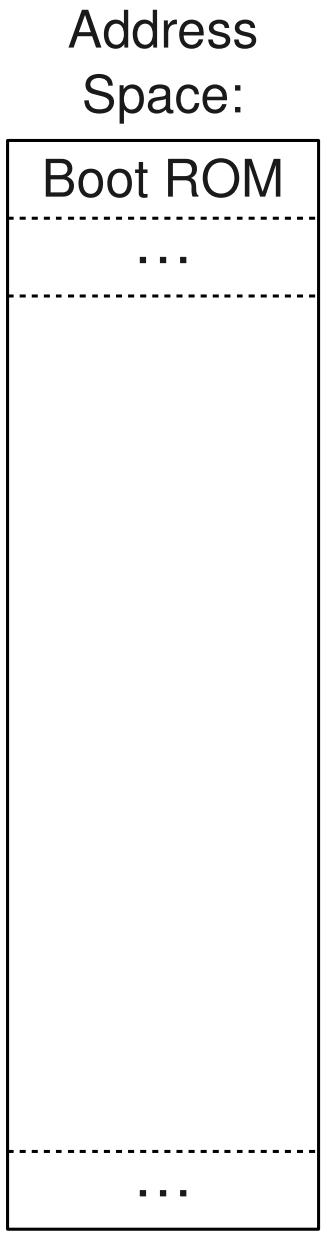
Kexec Hardboot

- Modification, uses hardware reboot to fix “kexec problem”
- Basic idea
  - Use existing kexec to stage new kernel in memory
  - Perform hardware reboot to reinitialize devices
  - Hook into Linux boot process, jump to new kernel
- Requirement: “Some” RAM must preserve across reboot
  - DRAM controller must provide refresh cycles
  - Bootloader must leave some part of memory untouched

RAM Console

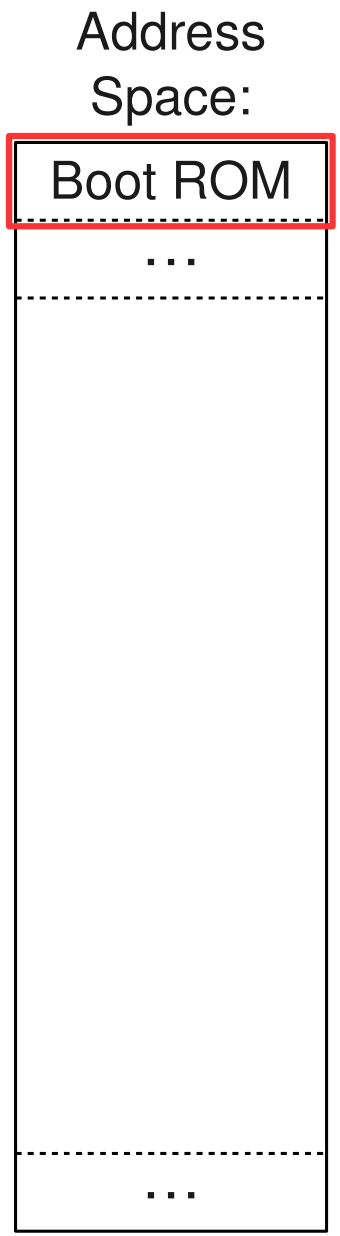
- Linux logs panic messages to kmsg buffer
  - Prints to VGA console on desktops and servers
  - Serial consoles used for embedded devices
  - No way to retrieve when phone panics in pocket
- Android RAM Console
  - Logs to circular buffer outside normal system RAM
  - Device automatically reboots on panic, preserves buffer
  - Old logs made available on reboot, `/proc/lastkmsg`
- Immensely useful for debugging, kudos for making available
- Serves as precedent for preserving RAM across reboot

Linux Boot Example



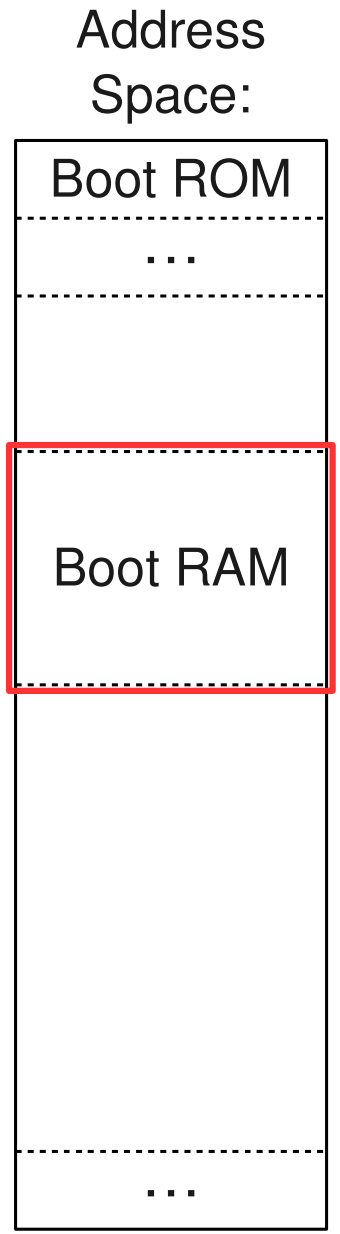
Linux Boot Example

- Power on: Starts execution in boot ROM



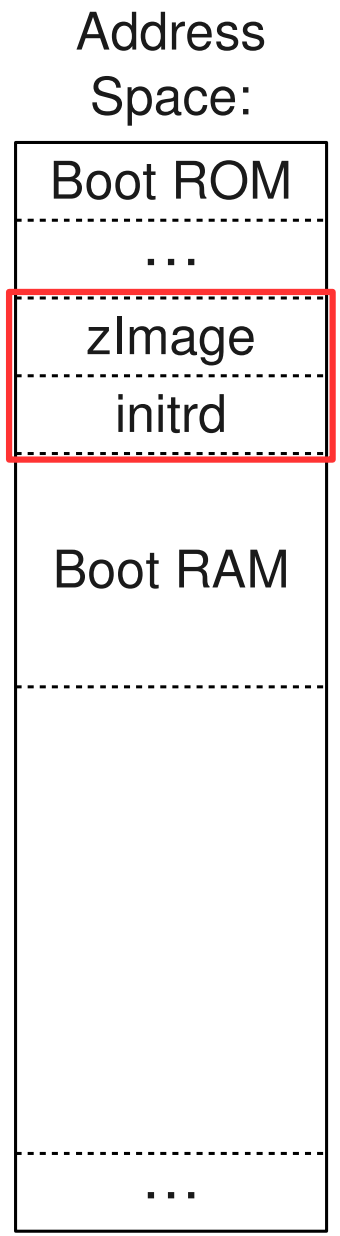
Linux Boot Example

- Power on: Starts execution in boot ROM
- Initializes memory, sets up boot loader in RAM



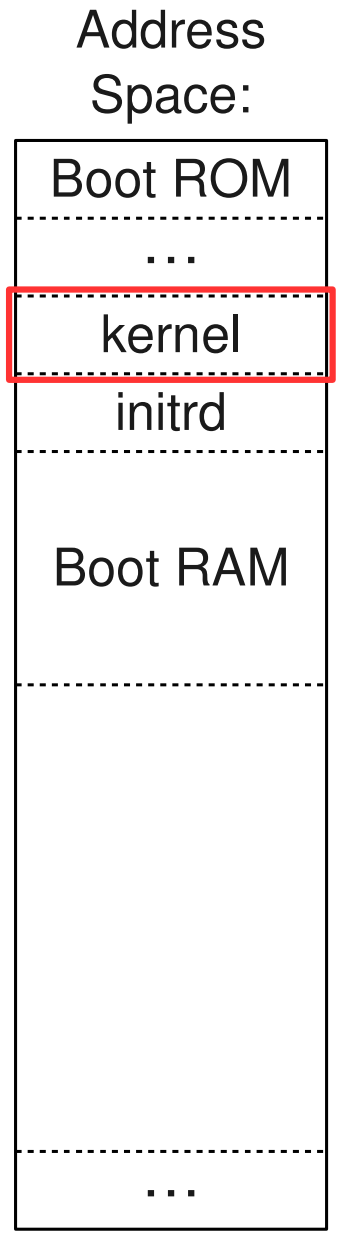
Linux Boot Example

- Power on: Starts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd



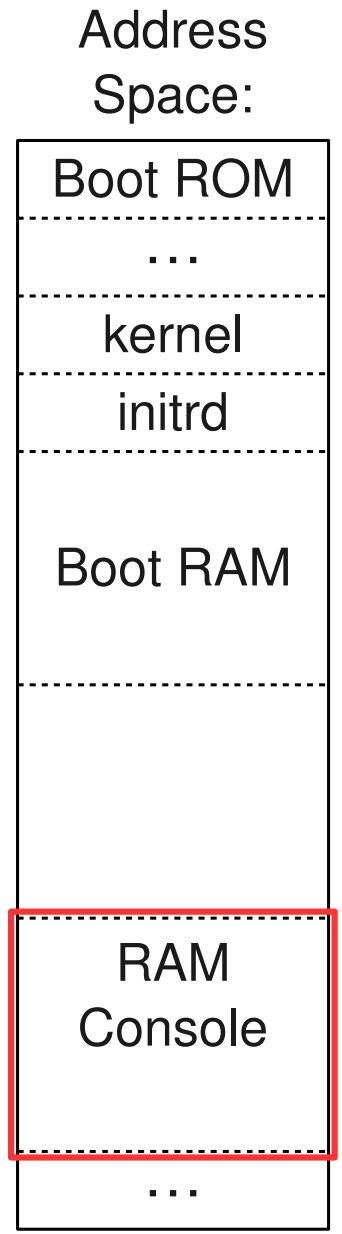
Linux Boot Example

- Power on: Starts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage, decompresses itself, boots



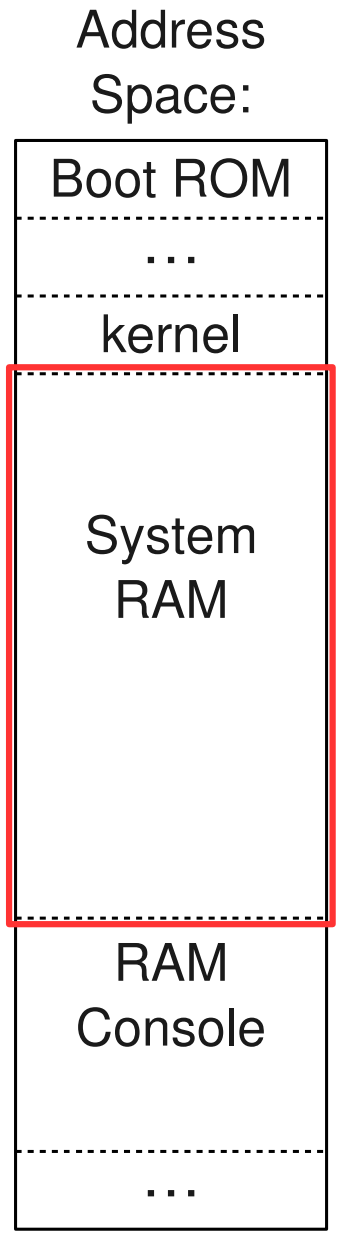
Linux Boot Example

- Power on: Starts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage, decompresses itself, boots
- Carves out RAM Console buffer

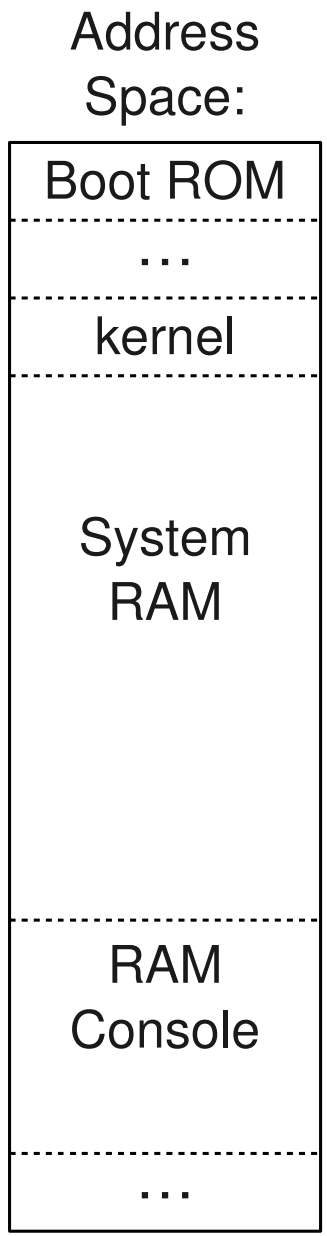


Linux Boot Example

- Power on: Starts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage, decompresses itself, boots
- Carves out RAM Console buffer
- Uses remaining memory as System RAM

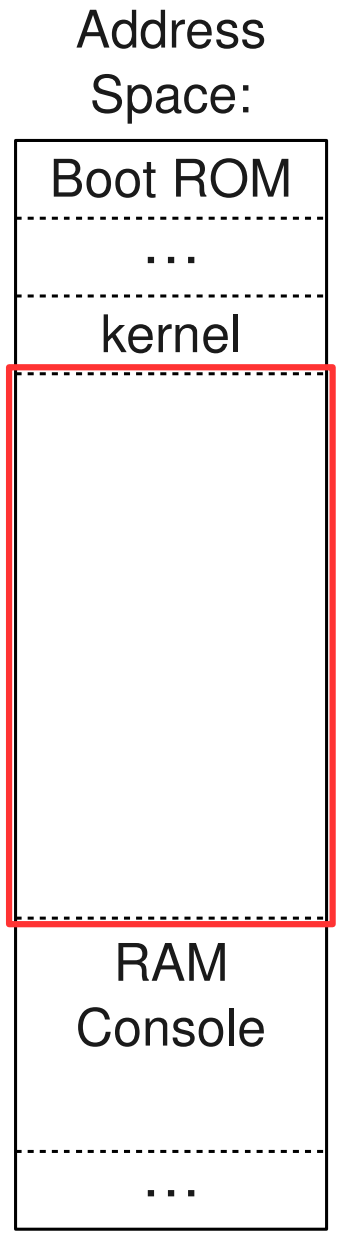


Kexec Hardboot Example (I)



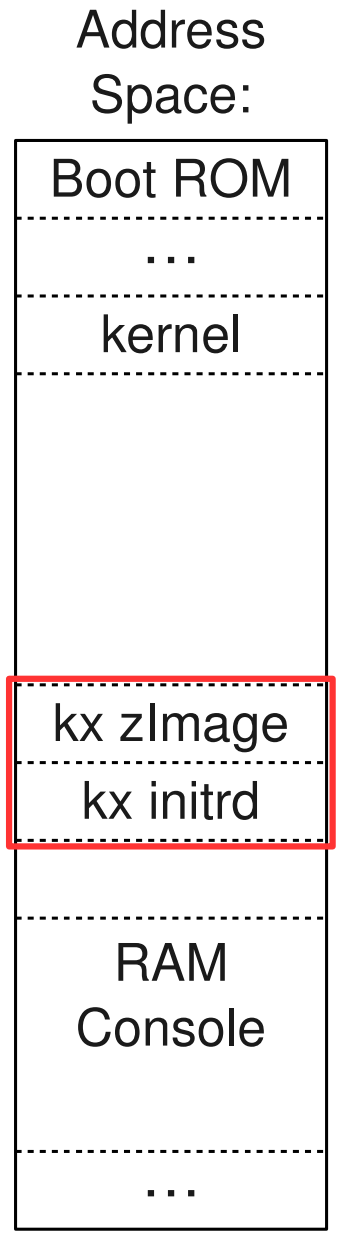
Kexec Hardboot Example (I)

- "`kexec -e`", prepares reboot, frees System RAM



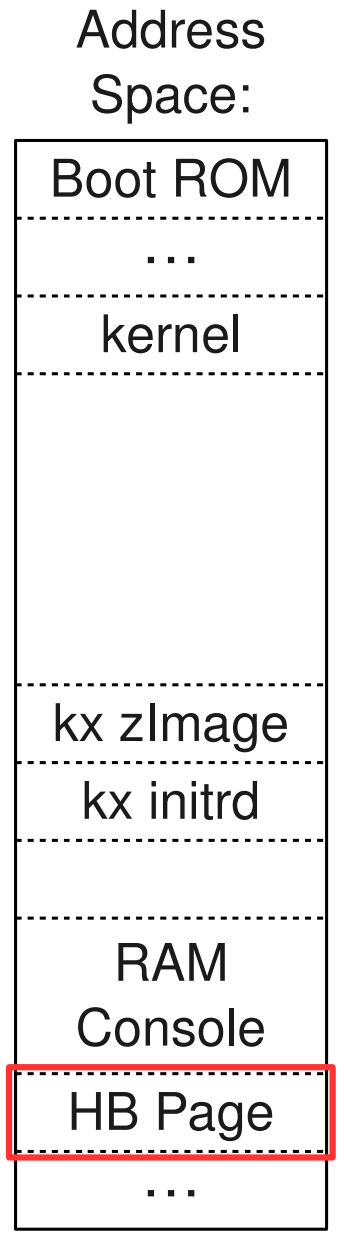
Kexec Hardboot Example (I)

- "`kexec -e`", prepares reboot, frees System RAM
- Uses "`--mem-min`" to stage kernel high in RAM



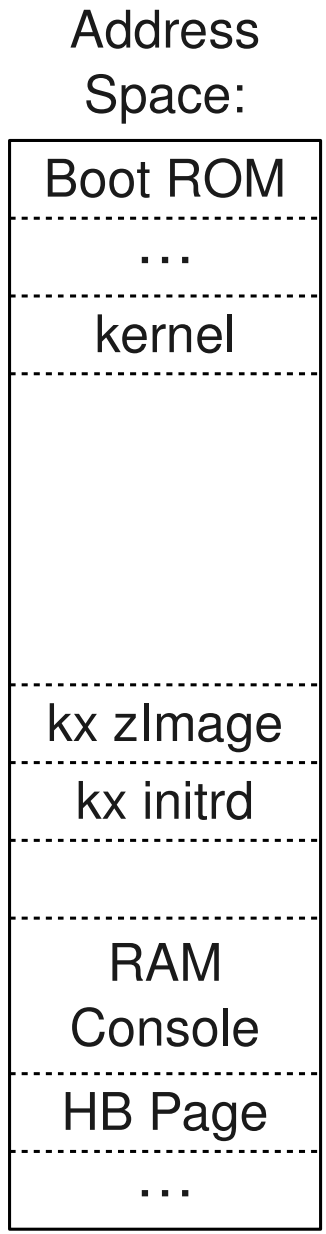
Kexec Hardboot Example (I)

- "`kexec -e`", prepares reboot, frees System RAM
- Uses "`--mem-min`" to stage kernel high in RAM
- Scribbles kernel location to hardboot page



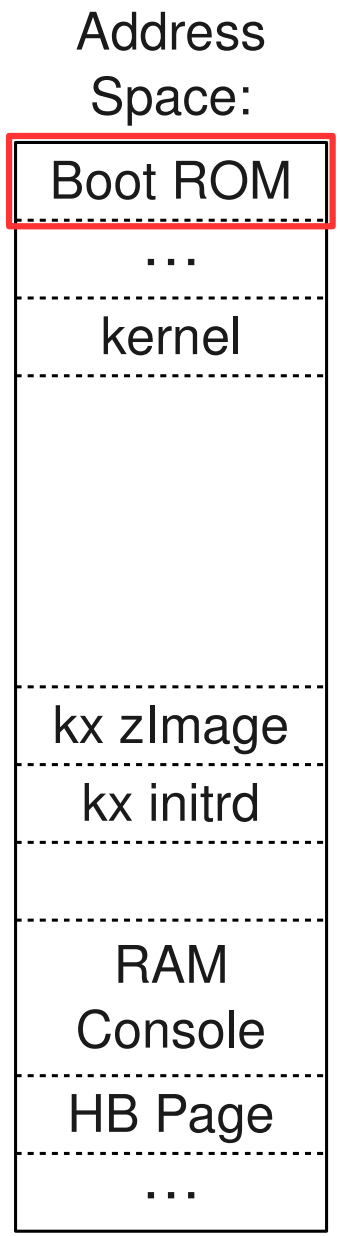
Kexec Hardboot Example (I)

- "`kexec -e`", prepares reboot, frees System RAM
- Uses "`--mem-min`" to stage kernel high in RAM
- Scribbles kernel location to hardboot page
- Performs hardware reboot, RAM preserved



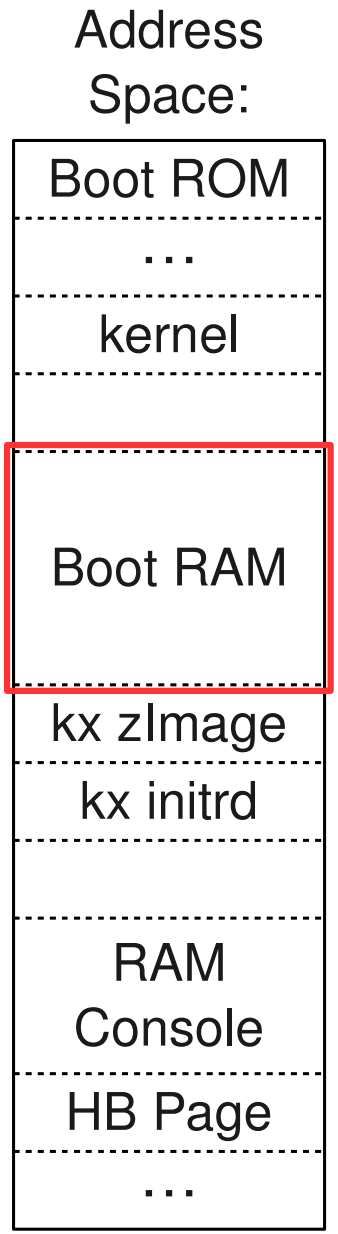
Kexec Hardboot Example (II)

- Restarts execution in boot ROM



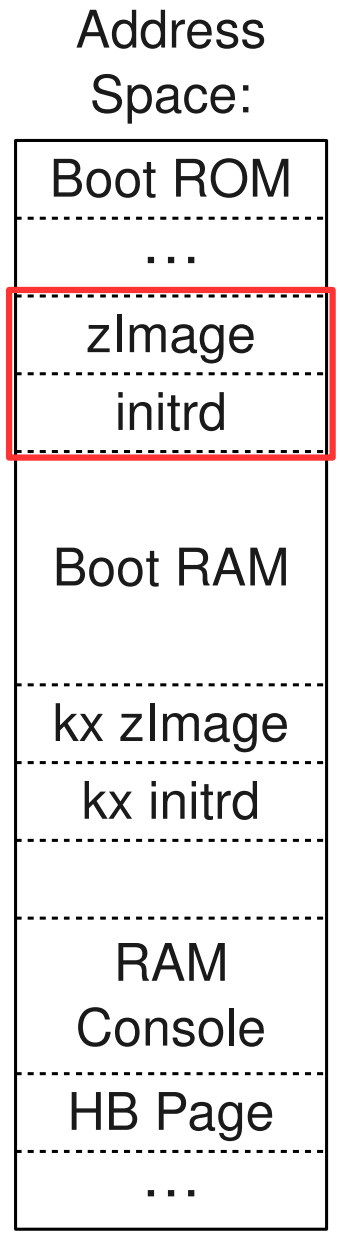
Kexec Hardboot Example (II)

- Restarts execution in boot ROM
- Initializes memory, sets up boot loader in RAM



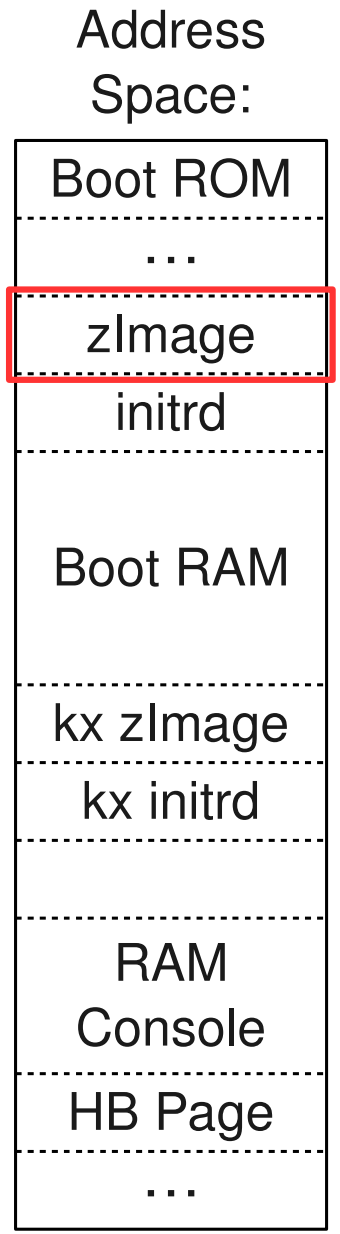
Kexec Hardboot Example (II)

- Restarts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd



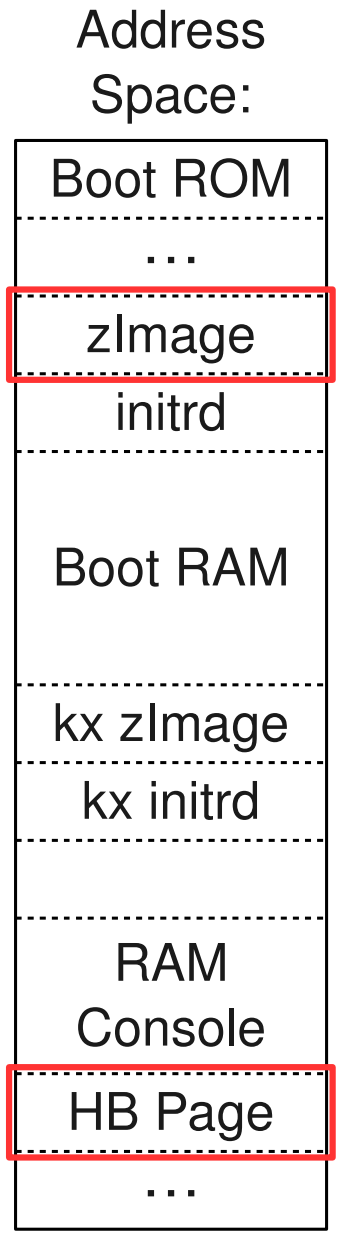
Kexec Hardboot Example (II)

- Restarts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage



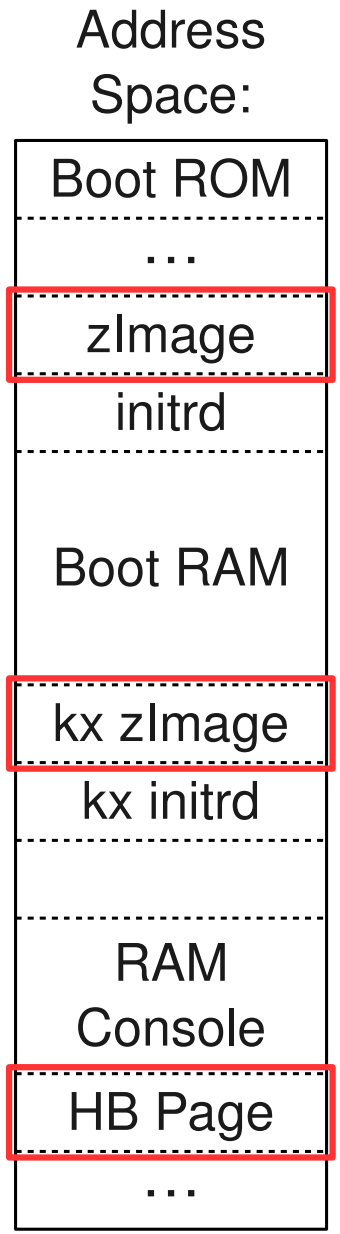
Kexec Hardboot Example (II)

- Restarts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage
- Decompressor checks for hardboot page



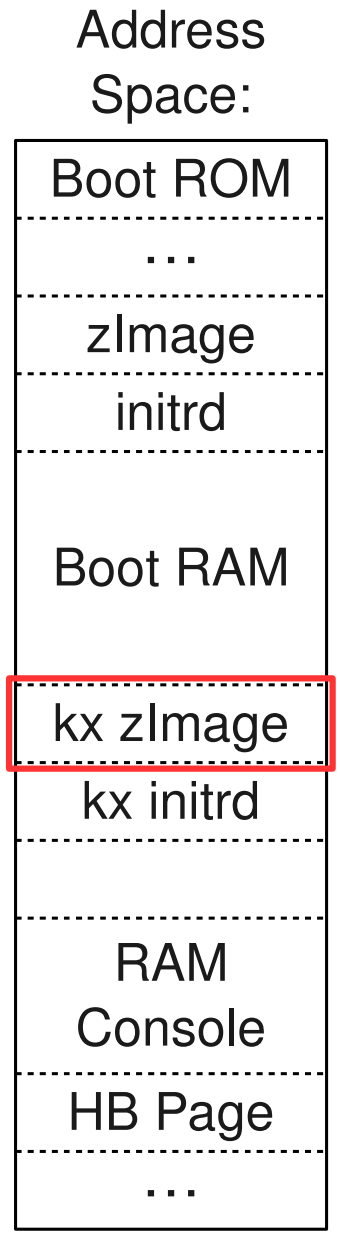
Kexec Hardboot Example (II)

- Restarts execution in boot ROM
- Initializes memory, sets up boot loader in RAM
- Loads compressed kernel (zImage) and initrd
- Jumps to zImage
- Decompressor checks for hardboot page
- Finds compressed kexec kernel in high RAM



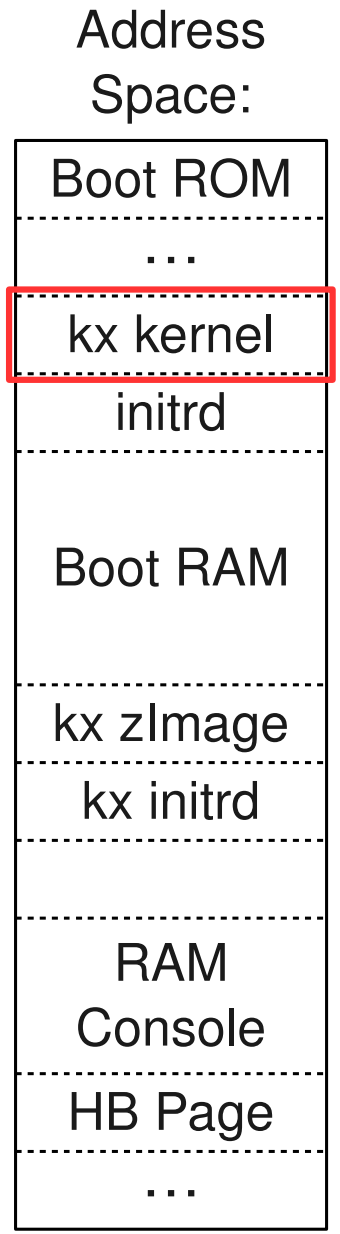
Kexec Hardboot Example (III)

- Jumps to kexec zImage



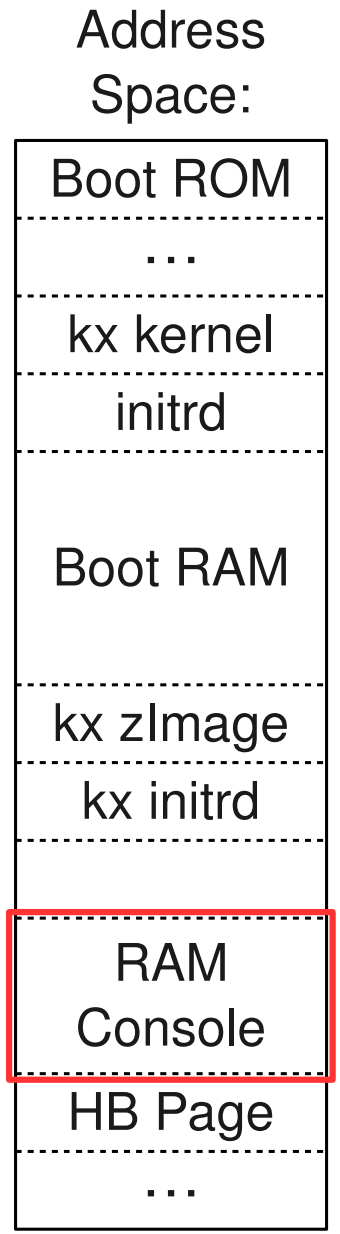
Kexec Hardboot Example (III)

- Jumps to kexec zImage
- Decompresses to regular load address, boots



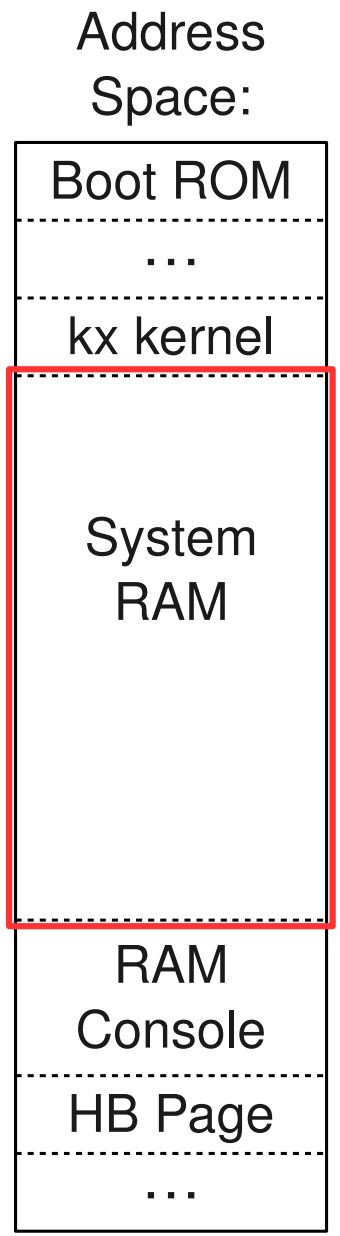
Kexec Hardboot Example (III)

- Jumps to kexec zImage
- Decompresses to regular load address, boots
- Saves old RAM Console buffer



Kexec Hardboot Example (III)

- Jumps to kexec zImage
- Decompresses to regular load address, boots
- Saves old RAM Console buffer
- Reuses remaining memory as System RAM



Code Components (I)

- Backports of recent ARM kexec patches to vendor kernels
  - Traditionally machines “soft reboot”, jump to reset vector
  - Soft reboot (MMU disable) broken as of ARMv6
  - Every vendor implements own hardware-based reboot code
  - Parts of soft reboot and kexec pathway left unimplemented
- Hardboot path for `relocate_new_kernel`
  - Scribble kernel location and boot params to hardboot page
  - Machine-specific code to force hardware reboot (port from C)
  - Tricky ARM assembly, no other kernel facilities available

Code Components (II)

- Aside: The zImage Decompressor
  - Very first Linux code to execute on boot
  - Self-contained program, essentially a third-stage boot loader
  - Flexible, can be located anywhere, relocates itself if needed
- Decompressor trampoline
  - Checks hardboot page for kexec kernel, jumps to its zImage
- Fixes to speed-up boot, relocate kernel parameters (atags)

Outline

Kexec Applications

- Kernel testing & debugging
  - `adb push` kernel to /**tmp**, run `kexec` from shell
- Repurpose Android Recovery as a boot menu
  - Usually accessed with a power-on key combination
  - Intended for applying system updates (update.zips)
  - Apply an “update.zip” that kexecs /**sdcard/boot.img**
- Dual-boot & multi-boot support
  - Keep stock ROM on device, custom ROM on microSD
  - SD-based “temp boot” stock ROMs for provisioning updates
  - NFS root: Boot ROMs from NFS shares, no flashing!

Device Support

- Currently supported devices
  - Samsung Epic 4G (epicmtd)
  - Samsung Epic 4G Touch
  - Samsung Galaxy S III (US variants: d2spr, d2vzw)
- Proof-of-concept ports
  - Samsung Nexus S (crespo, crespo4g)
  - Samsung Galaxy Tab (p1)
  - ASUS Transformer Prime TF201
- Working platforms: Exynos, Snapdragon (MSM), Tegra
- Little platform-specific code, relatively easy to port

Summary

- Android is a fun hacking platform, some neat tools
- Consumer-oriented devices are inflexible, difficult to hack
- Kexec makes devices far more flexible
- Hardboot approach makes kexec work on buggy devices