Taking the Plunge – The Marriage of X86 and Embedded Linux®

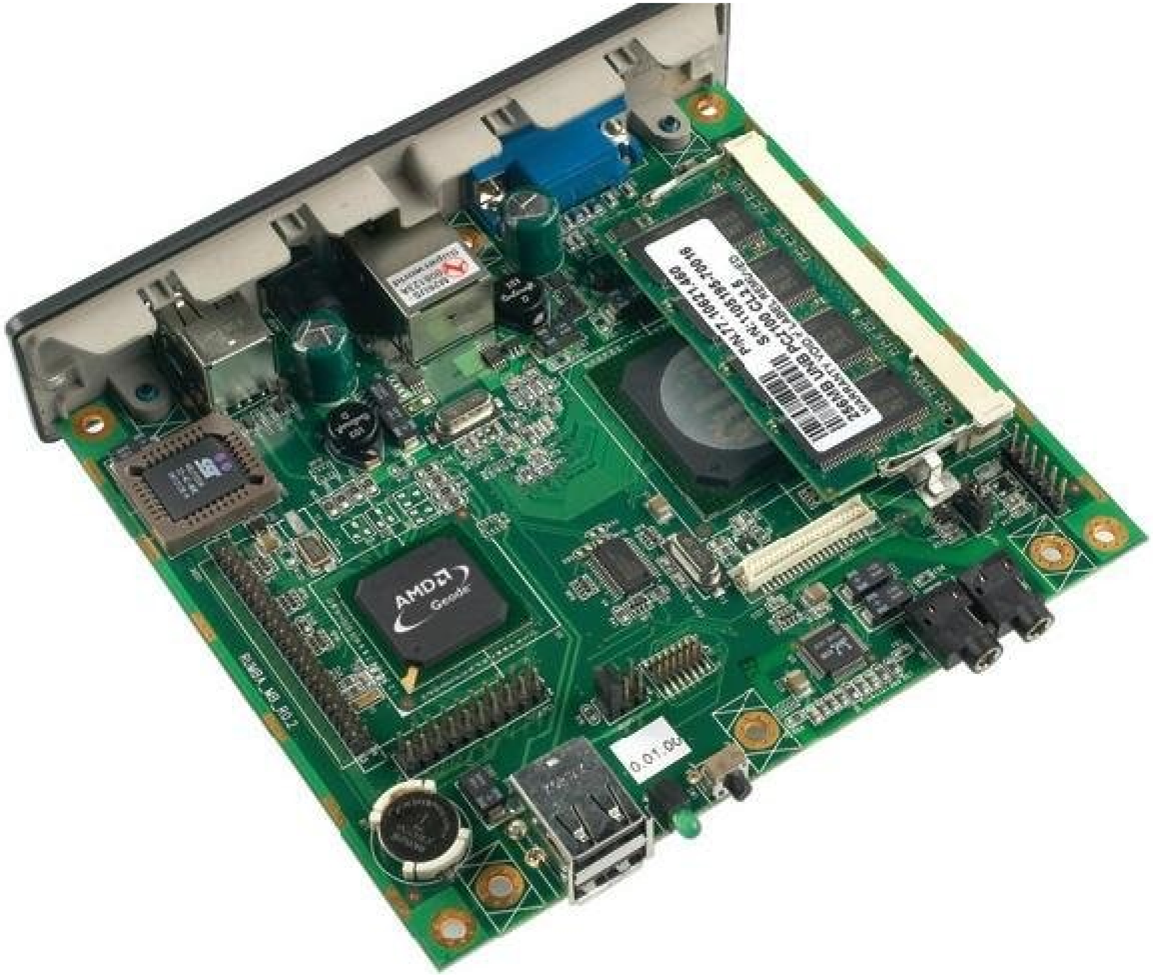
Jordan Crouse, Senior Software Engineer, AMD

April 11, 2006

Benefits of X86 and Embedded Images

- Benefits of X86
  - Familiar architecture
  - Large pre-existing code base
  - Versatile
- Benefits of Embedded Images
  - Small image sizes
  - Customized and targeted to the platform
  - Compact image binaries are easy to share and install

An Embedded X86 Platform



Media on the X86

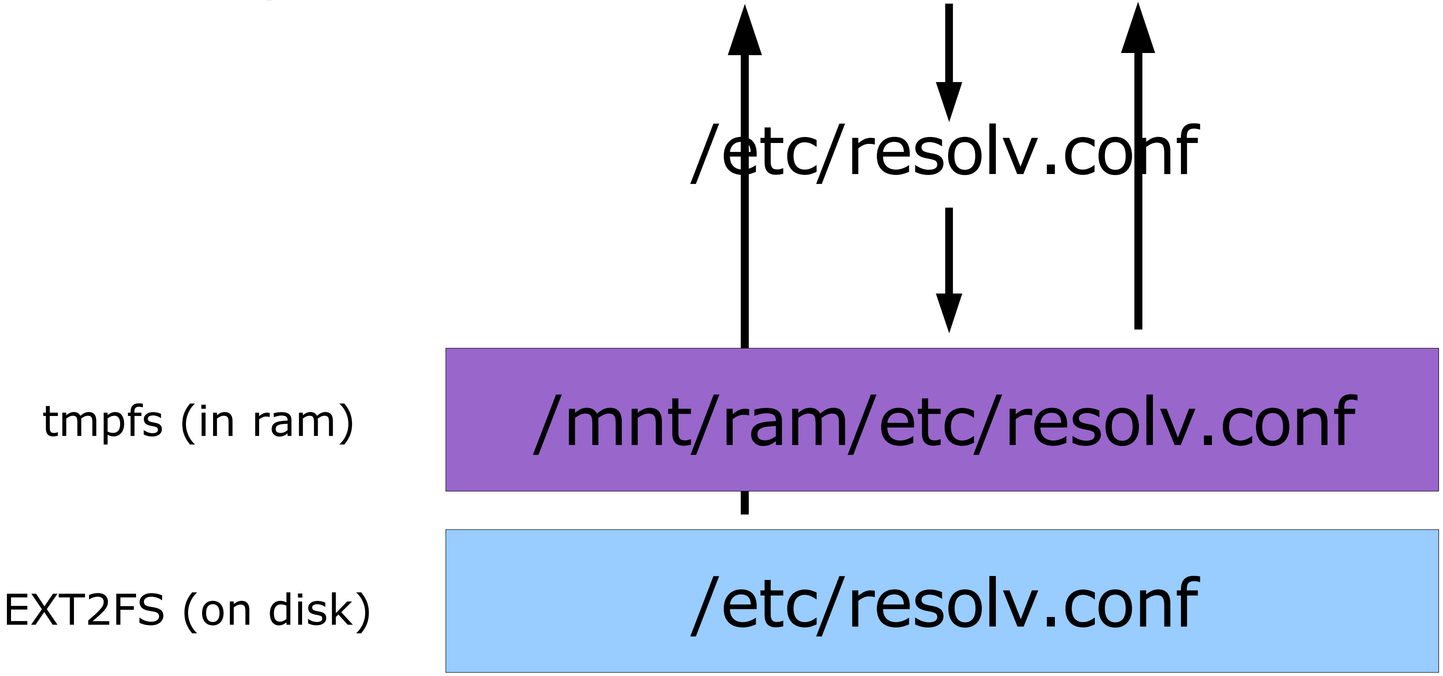
- Compact Flash
  - Always a popular alternative to IDE hard drives
  - Many boards provide CF slot options
  - Adapter available for most commonly used IDE connectors
  - CF writers for desktop devices cheap and plentiful
- USB Flash Keys
  - Widely available
  - Can be used with any device with a USB slot
  - Not very “user-proof” since users can pull key out at any time
- Others
  - NOR/NAND Flash
  - SD/MMC
  - DOC/DOM

Designing an Image for Compact Flash

- Be aware of the Flash Lifetime
  - Always a concern for production systems
  - Write to the device as little as possible
  - Use RAM file systems as much as possible
- Protect the Image from the User
  - All it takes is a clueless or malicious user to bring down a device
- Provide Separate Persistent Storage
  - Preserving user data across upgrades
  - Power loss or other catastrophe more likely than on a desktop
  - Preventing the user from destroying the binary image

Using UnionFS

- A stackable unification file system, which can appear to merge the contents of several directories (branches), while keeping their physical content separate.



/etc/resolv.conf

tmpfs (in ram)

/mnt/ram/etc/resolv.conf

EXT2FS (on disk)

/etc/resolv.conf

Advantages of UnionFS

- Allows a more “standard” root file system layout
- Files are shadowed as they are needed
  - Saves persistent storage and RAM by only storing the files that have been modified.
- Allows us to “user-proof” the system
  - Retains a read-only master of all system files
- Provides flexibility for different persistent storage requirements
- Easy to use
  -
    ```
    mount -t unionfs -o dirs=/tmp/ram/etc:/etc none /etc
    ```
- Simpler and safer than a bind mount or symbolic link solution

Populating the Image

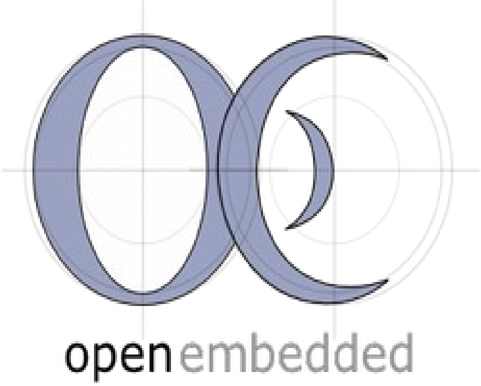
- Use Embedded Friendly Applications
  - uclibc, busybox and tinylogin
- Keep an eye on new innovations from the community
  - Embedded focus is bigger then ever
- Know your audience
  - Remove unneeded locales, themes and documentation
- Always keep size in mind!

Building an Embedded Image For X86

- The build process must take advantage of high speed processor resources
- The build process must be as automated as possible
- The image must be easy to share
- Must be installable on any media that can be booted by the BIOS
  - Hard drive, Compact Flash, or USB key
- Must be very easy to install with as few steps as possible

OpenEmbedded

- BitBake build tool
  - Python based
  - Metadata parser
  - Package graph to handle dependencies
  - Task graph to handle task dependencies
  - On the fly script generator
- OpenEmbedded metadata Repository
  - Defines tasks and packages for building embedded images
  - Machine configurations (over 50!)
  - Distribution policies
  - Package recipes (over 2500 unique recipes!)



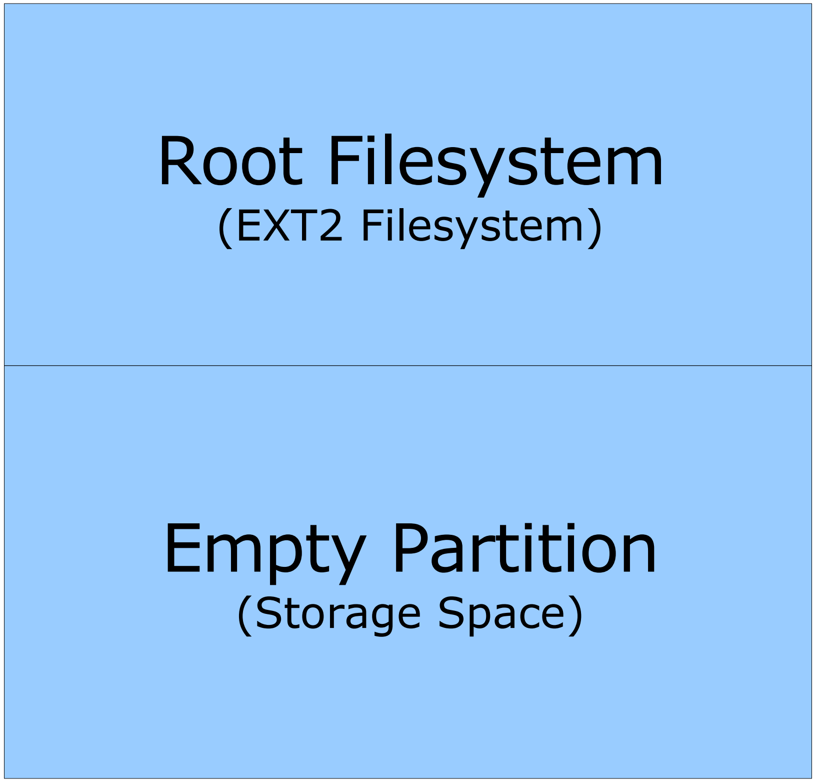
Why Did We Pick OpenEmbedded?

- Runs on AMD64 machines
  - Able to take full advantage of environment by cross compiling 32 bit applications with 64 bit native applications
- Easy to Extend
  - Easy to add new package recipes and configuration files
- Very Configurable
  - Machine type, optimization flags, package versions, and the image file system type among many others
- Minimal or no user interaction needed
  - One command builds everything from scratch all the way to the final image
- Very active development community

Building the Root Filesystem in OE

1) Build the toolchain and supporting packages
2) Build the kernel for the machine
3) Build the target applications and scripts
4) Construct the root filesystem image
   - Build the filesystem from the application packages
   - Build the image with a filesystem tool

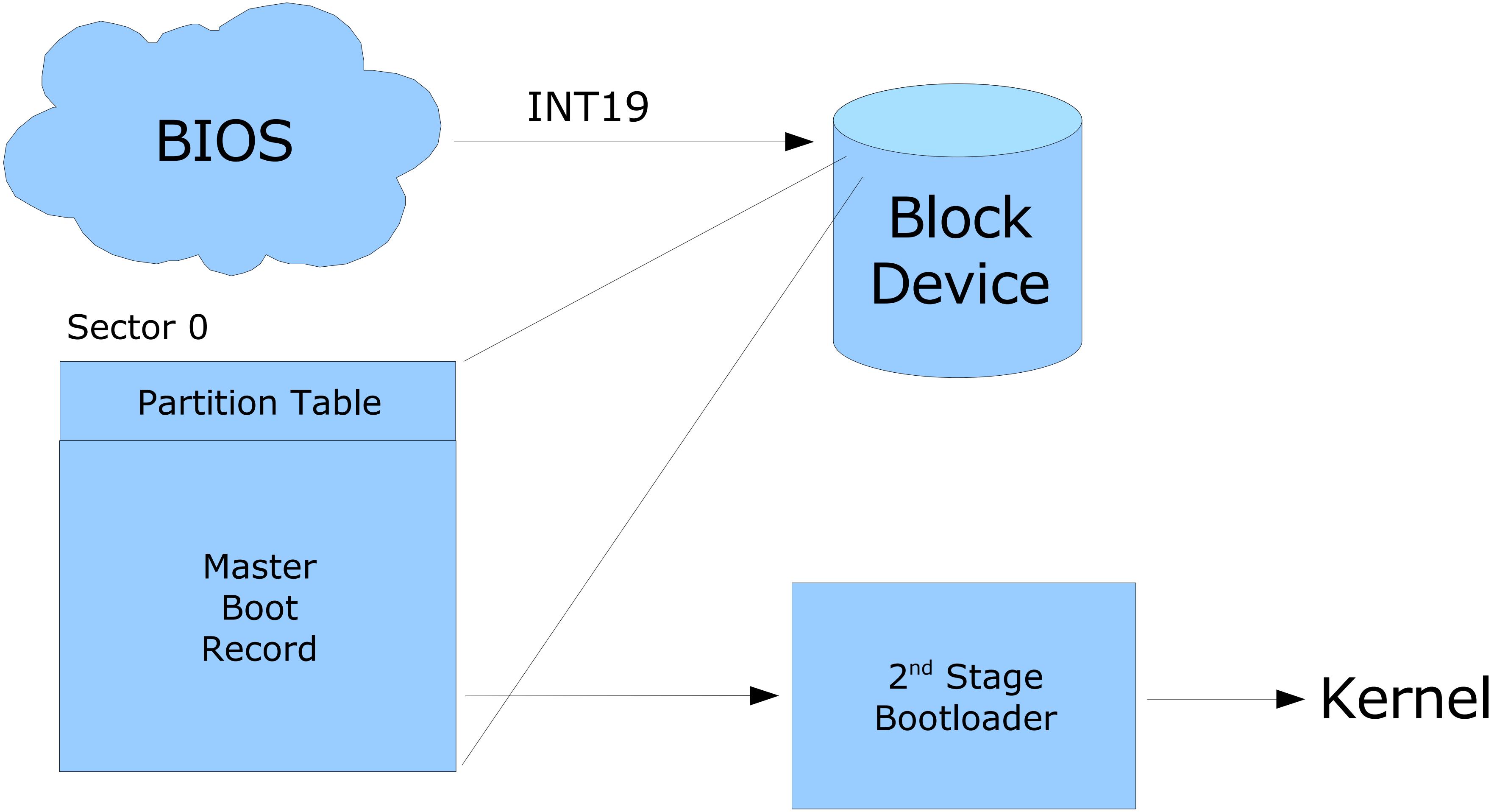
The Story So Far...



Root Filesystem
(EXT2 Filesystem)

Empty Partition
(Storage Space)

How To Boot an X86



Manually Installing an X86 Image

- Create the partition table on the device
  - Requires knowledge of the device geometry.
- Create or copy a root filesystem to a partition
  - In our case, the root filesystem is created by the build process
- Install the bootloader
  - Requires knowledge of the device geometry
  - The installation process writes the 1st stage loader to the MBR and the 2nd stage loader to the bootable partition

Syslinux

- Lightweight bootloader that runs from a FAT filesystem
- Simple interface – no advanced editing like GRUB
  - Not a huge concern for embedded platforms
- Does not require knowledge of the media geometry to be installed
- Does require an additional FAT partition on the image
  - Containing the boot loader, configuration file and kernel

Creating a FAT Partition

- Extension to mkdosfs utility
  - Add option '-d' to create a FAT filesystem from a directory of files
  - Usage: mkdosfs -F 12 -d boot/ -C boot.dosfs 512

Building the Boot Partition in OE

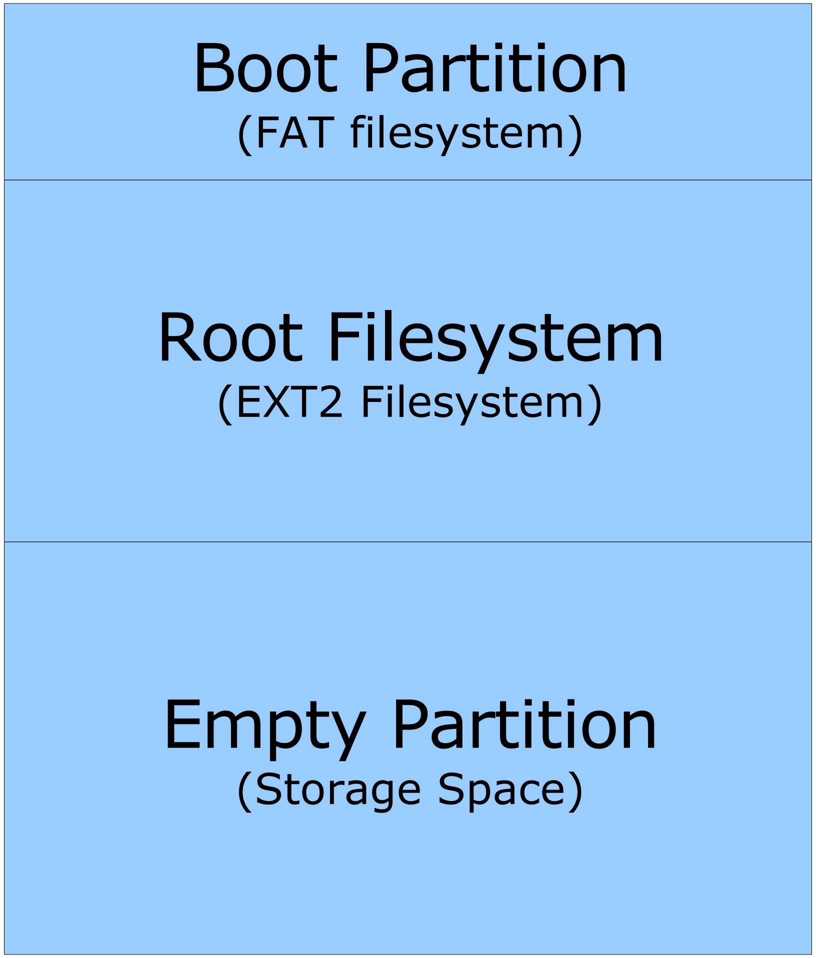
1) Copy kernel bzImage to a build directory
2) Auto generate the configuration file
3) Make the FAT filesystem (with mkdosfs)
4) Use the syslinux installer to install the bootloader into the filesystem

```
AUTO_SYSLINUXCFG=”1”
AUTO_SYSLINUXMENU=”1”

LABELS="default debug"
APPEND_default="root=/dev/hda2"
APPEND_debug="root=/dev/hda2 kgdbwait"

USAGE_default="Boot default configuration"
USAGE_debug="Boot default configuration, but wait for debugger"
```

Another Look at our Image



Boot Partition

(FAT filesystem)

Root Filesystem

(EXT2 Filesystem)

Empty Partition

(Storage Space)

Imgloader

- A system that allows us to create a single binary image file (.img) containing several filesystem images plus partition table information.
- Filesystems are gziped to reduce the size of the .img file.
- The imgloader system includes two important utilities:
  - mkimg – Creates a new .img file
  - imgloader – Interactive tool to write .img files onto physical media

Imgloader Advantages

- A single binary image is easier to share and maintain
- Images are media agnostic at build time
  - No problems with differing manufacturer or card specifications
- The quantity and type of partitions is determined at image creation time
- New filesystem types or partition configurations can be dropped in without changing anything at the image writing stage.

Mkimg Configuration File

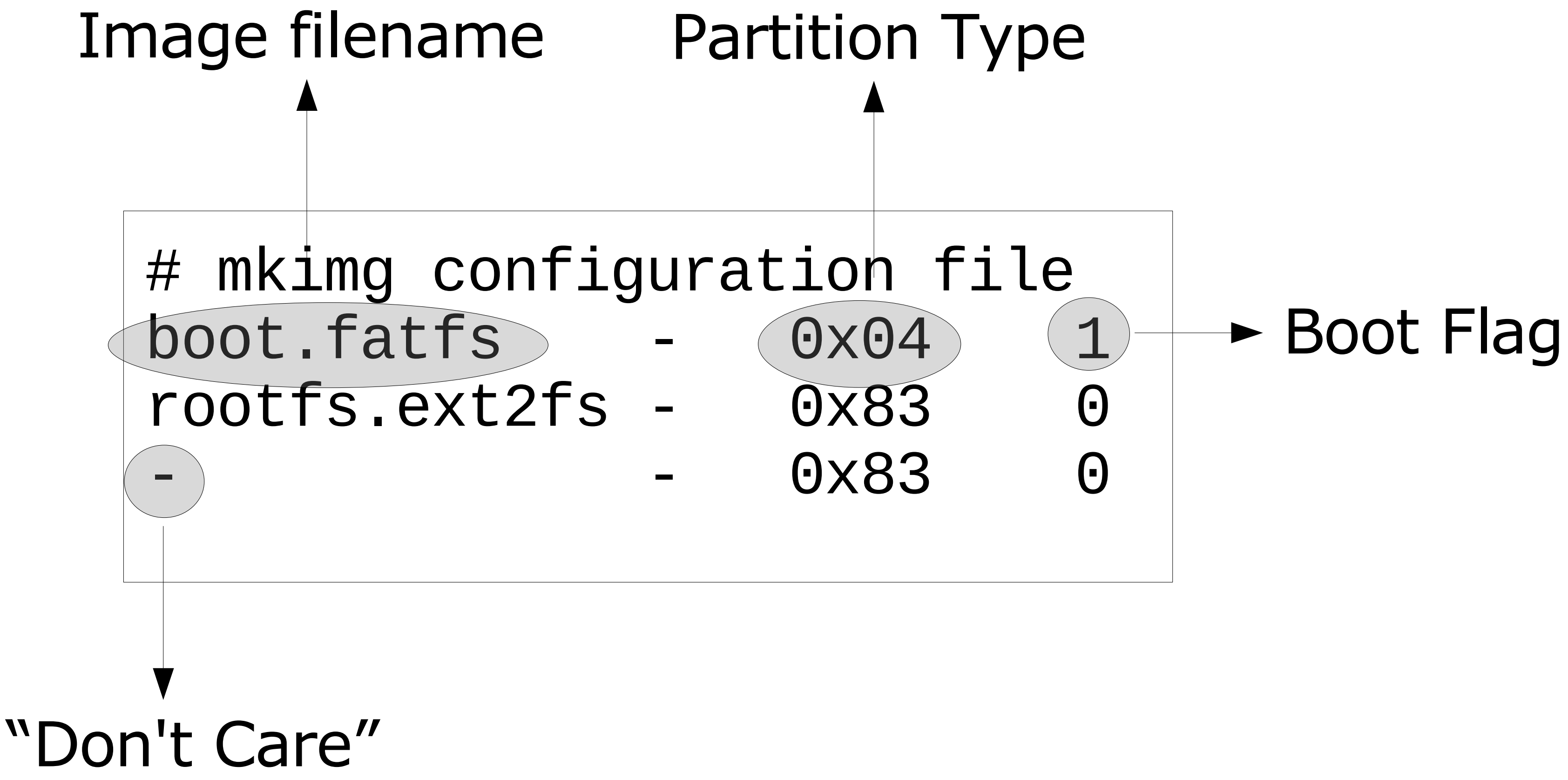


Image filename

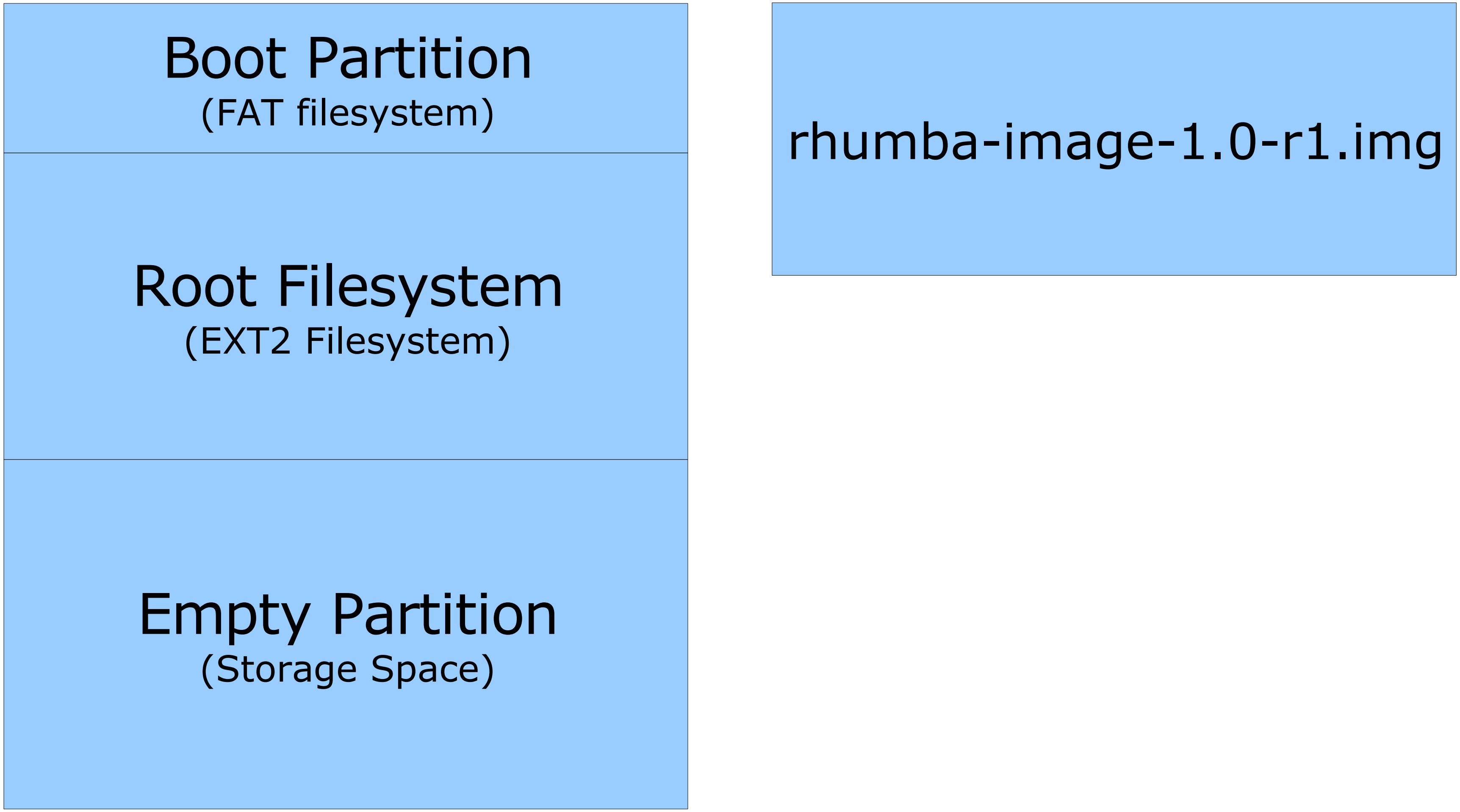
Partition Type

```
# mkimg configuration file
boot.fatfs     -    0x04    1
rootfs.ext2fs  -    0x83    0
-              -    0x83    0
```

Boot Flag

“Don't Care”

Converting to an .IMG



Boot Partition
(FAT filesystem)

Root Filesystem
(EXT2 Filesystem)

Empty Partition
(Storage Space)

rhumba-image-1.0-r1.img

Imgloader Utility

- An interactive console based tool that writes .img files to the actual media
- Designed to run from either the desktop or directly on the target device
  - Uses few libraries and is both glibc and uclibc friendly.
  - Works very well in a tiny initrd booted from a kernel on a device such as a USB key
- Optional support for CURL allows images to be downloaded via HTTP, FTP and TFTP.
- Coming soon:
  - Non-interactive script support
  - Curses menu system with “dialog”

How Imgloader Works

1) Get the geometry of the block device
2) Load the header from the .img file
   - Either from a file or optionally downloaded
3) Calculate the partition table entries and offsets
4) Write the partition table and MBR
   - The .img can specify a “custom” MBR, or it will use a default one from Syslinux
5) Uncompress and write each image to the correct offset

Imgloader Example

```
AMD Linux Image Loader Version 1.06.0001
Copyright 2004-2006, Advanced Micro Devices, Inc.
(loader) load /home/jcrouse/rumba-image-1.0-r1.img /dev/sda

Device [/dev/sda]: 8/62/1008
Units:  496 * 512 bytes = 253952 bytes

    Units   Sectors Blocks Type Payload
1   7       3906    1953   04   Binary
2   265     131936  65968  83   Binary
3   731     363072  181536 83   Empty

Done.
Writing master boot record...Done.
Writing partition table...Done.
Writing partition 1...Done.
Writing partition 2...Done.
Zapping partition 3...Done
(loader)
```

Wrap Up

- ✓ Advantages of X86 and embedded Images
- ✓ Options for storage media
- ✓ How to protect the image and the user from disaster
- ✓ Building images with OpenEmbedded
- ✓ Creating and installing embedded images with imgloader

Resources

- UnionFS
  - http://www.fsl.cs.sunysb.edu/project-unionfs.html
- OpenEmbedded
  - http://www.openembedded.org
  - #oe on irc.freenode.net
- Bitbake
  - http://developer.berlios.de/projects/bitbake/
- SysLinux
  - http://syslinux.zytor.com/
- Dosfstools patches and Imgloader
  - http://cosmicpenguin.net/pub/sources/

Trademark Attribution

Linux is a registered trademark of Linus Torvalds.
AMD, the AMD Arrow logo and combinations thereof, and Geode are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.