EASY ARM

Embedded Linux Workshop on Blueboard-AT91

B. Vasu Dev

vasu@easyarm.com

Workshop Overview

Day 1

- ❑ Embedded Linux Intro
- ❑ Linux Basics (Lab)
- ❑ Toolchain (Lect)
- ❑ Toolchain (Lab)
- ❑ Bootloader (Lect)
- ❑ Bootloader (Lab)
- ❑ Running Linux on Target
  - > Using SD Card for rootfs
  - > Configuring NFS
  - > Configuring TFTP

Day 2

- ❑ Linux Kernel Arch.
- ❑ Linux Kernel Source code browsing and porting changes (Lect)
- ❑ Kernel Configuration & Compiling (Lab)
- ❑ Kernel modifications
- ❑ Root File System (Lect)
- ❑ Building Root FS (Lab)
- ❑ App Development & Cross Compilation (Lab)

Embedded Linux

Embedded Linux

- Customisation of Linux to fit on a Embedded Board designed for one or a few dedicted functions.
- All hardware sits on a single board most of the time.
- Computer system used in common deviced in daily use.

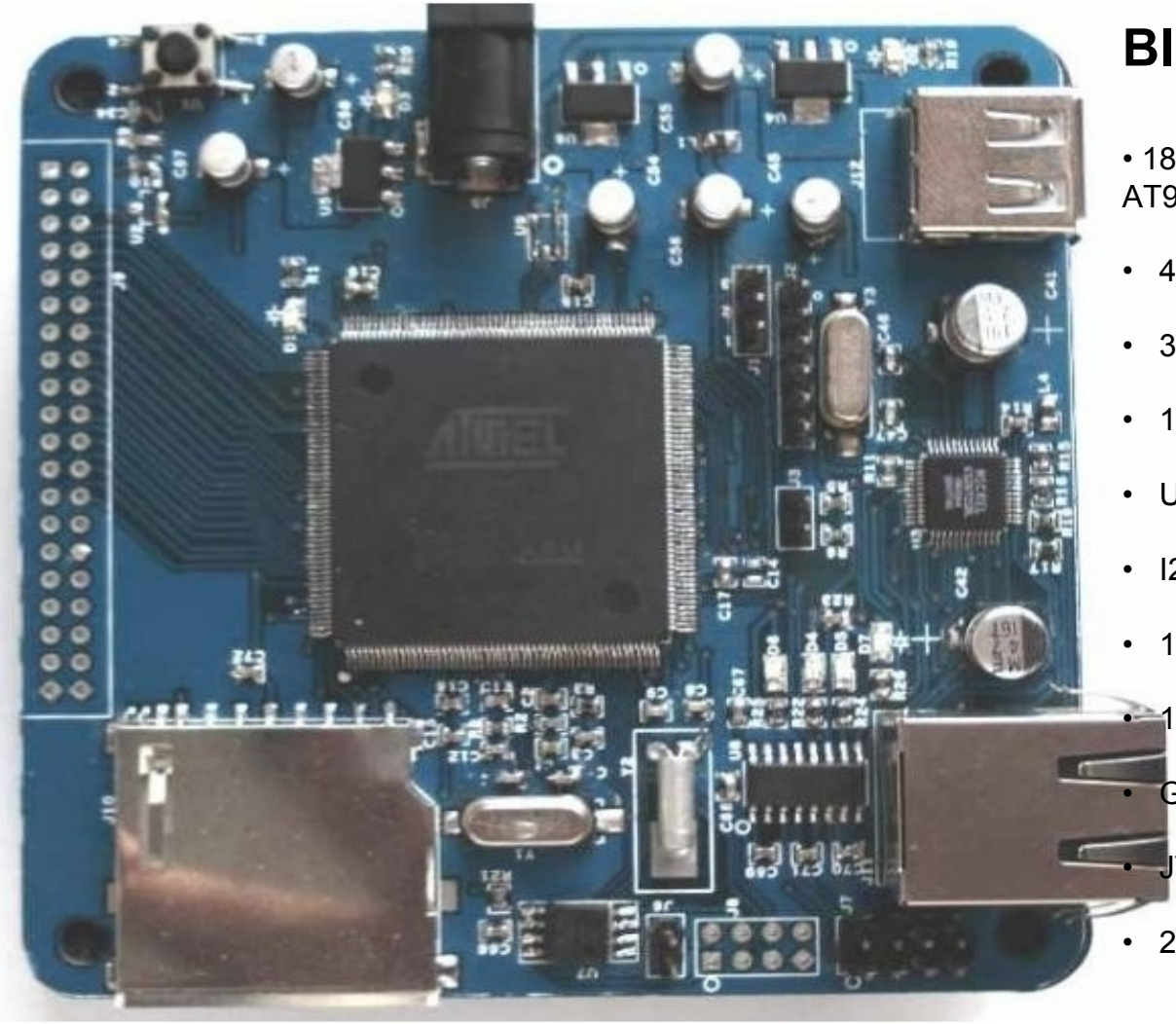
Applications

Hardware Requirement

- A CPU supported by gcc and the Linux kernel
  - 32 bit CPU
  - MMU-less CPUs are also supported, through the uClinux project.
- A few MB of RAM, from 4 MB.
  8 MB are needed to do really do something.
- A few MB of storage, from 2 MB.
  4 MB to really do something.
- Linux isn't designed for small microcontrollers that just have a few tens or hundreds of KB of flash and RAM.
  - Base metal, no OS
  - Reduced systems, such as FreeRTOS

EASY ARM

Our Hardware



BlueBoard AT91

• 180 MHz ARM9 processor (Atmel AT91RM9200)

- 4MB of serial flash
- 32MB SDRAM
- 1 SD/MMC slot
- USB 2.0 host
- I2C port
- 1 10/100 Ethernet interface
- 1 high speed USB 2.0 interface
- GP IO lines brought out on to a header
- JTAG support
- 2 serial ports

ARM Core Family

Application Cores	Embedded Cores	Secure Cores
ARM720T	ARM7EJ-S	SecureCore SC100
ARM920T	ARM7TDMI	SecureCore SC110
ARM922T	ARM7TDMI-S	SecurCore SC200
ARM926EJ-S	ARM946E-S	SecurCore SC210
ARM1020E	ARM966E-S	
ARM1022	ARM968E-S	
ARM1026EJ-S	ARM996HS	
ARM11 MPCore	ARM1026EJ-S	
ARM1136J(F)-S	ARM1156T2(F)-S	
ARM1176JZ(F)-S	ARM Cortex-M0	
ARM Cortex-A8	ARM Cortex-M1	
ARM Cortex-A9	ARM Cortex-M3	

ARM Core Family

T: Thumb
D: On-chip debug support
M: Enhanced multiplier
I: Embedded ICE hardware
T2: Thumb-2
S: Synthesizable code
E: Enhanced DSP instruction set
J: JAVA support, Jazelle
Z: Should be TrustZone?
F: Floating point unit
H: Handshake, clockless design for synchronous or asynchronous design

Development of the ARM Architecture

Early ARM architectures

1

2

3

Halfword and signed halfword / byte support

System mode

4

SA-110

SA-1110

Thumb instruction set

4T

ARM7TDMI **ARM9TDMI**

ARM720T **ARM940T**

Improved ARM/Thumb Interworking

CLZ

Saturated maths

DSP multiply-accumulate instructions

5TE

ARM1020E

XScale

ARM9E-S

ARM966E-S

Jazelle

Java bytecode execution

5TEJ

ARM9EJ-S **ARM926EJ-S**

ARM7EJ-S **ARM1026EJ-S**

SIMD Instructions

Multi-processing

V6 Memory architecture (VMSA)

Unaligned data support

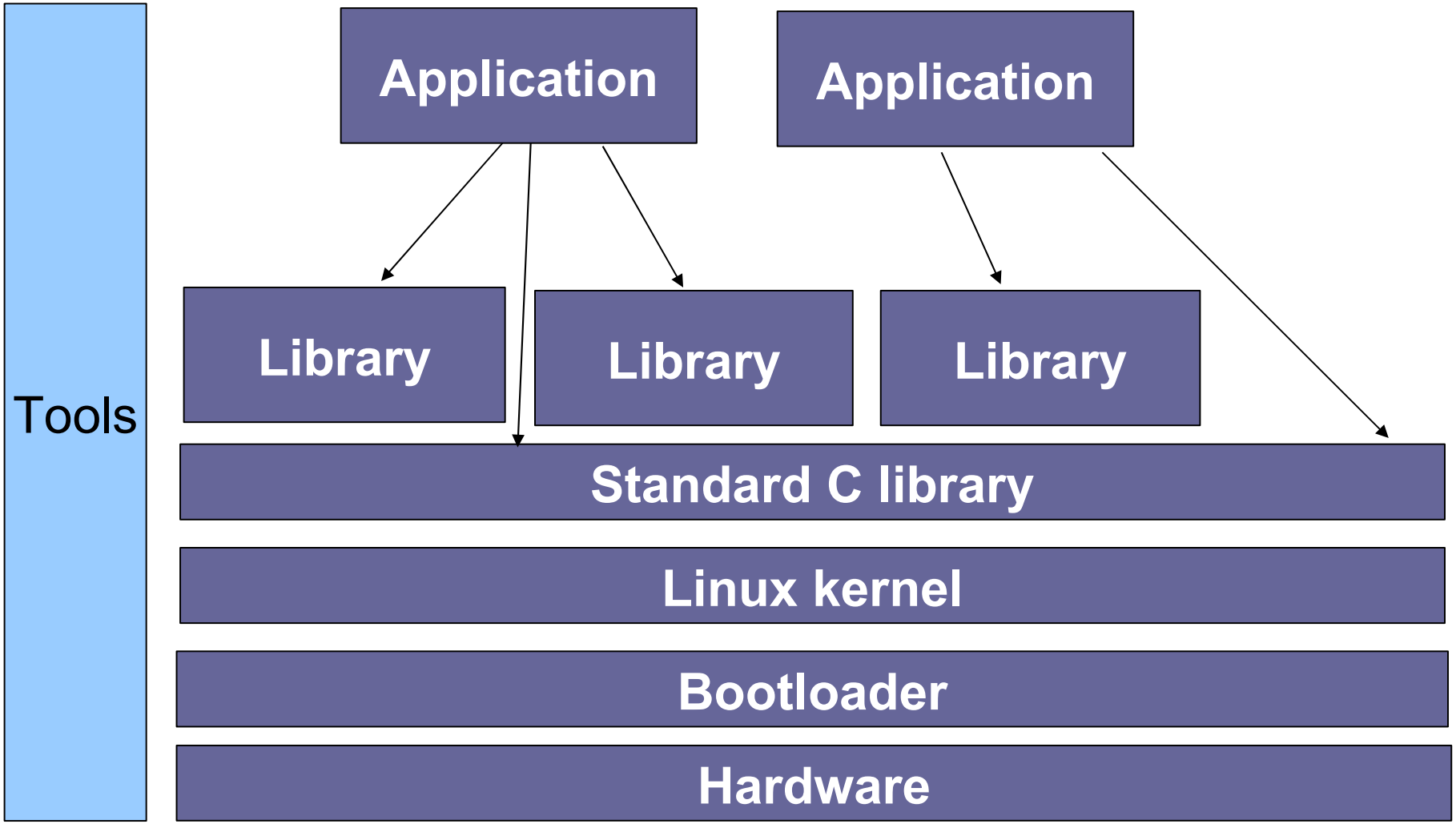
6

ARM1136EJ-S

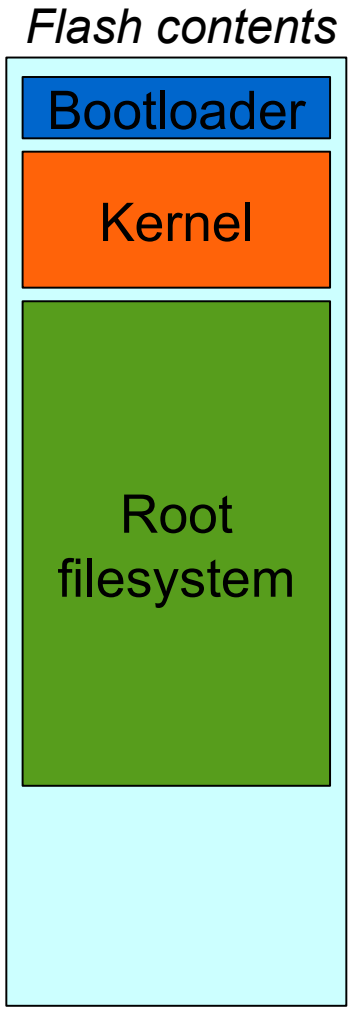
ARM Cores & Arch Ver

Core	Architecture
ARM1	v1
ARM2	v2
ARM2as, ARM3	v2a
ARM6, ARM600, ARM610	v3
ARM7, ARM700, ARM710	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
StrongARM, ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	V4T
ARM9E-S, ARM10TDMI, ARM1020E	v5TE
ARM10TDMI, ARM1020E	v5TE
ARM11 MPCore, ARM1136J(F)-S, ARM1176JZ(F)-S	v6
Cortex-A/R/M	v7

Software Components



Linux on Board



- Bootloader: First program executed by hardware and is responsible for basic initilization, loading and starting kernel.

- Kernel: Is the main program which provides platform to the application to execute. Kernel contains the process and memory management, network stack, device drivers and provides services to userspace applications.

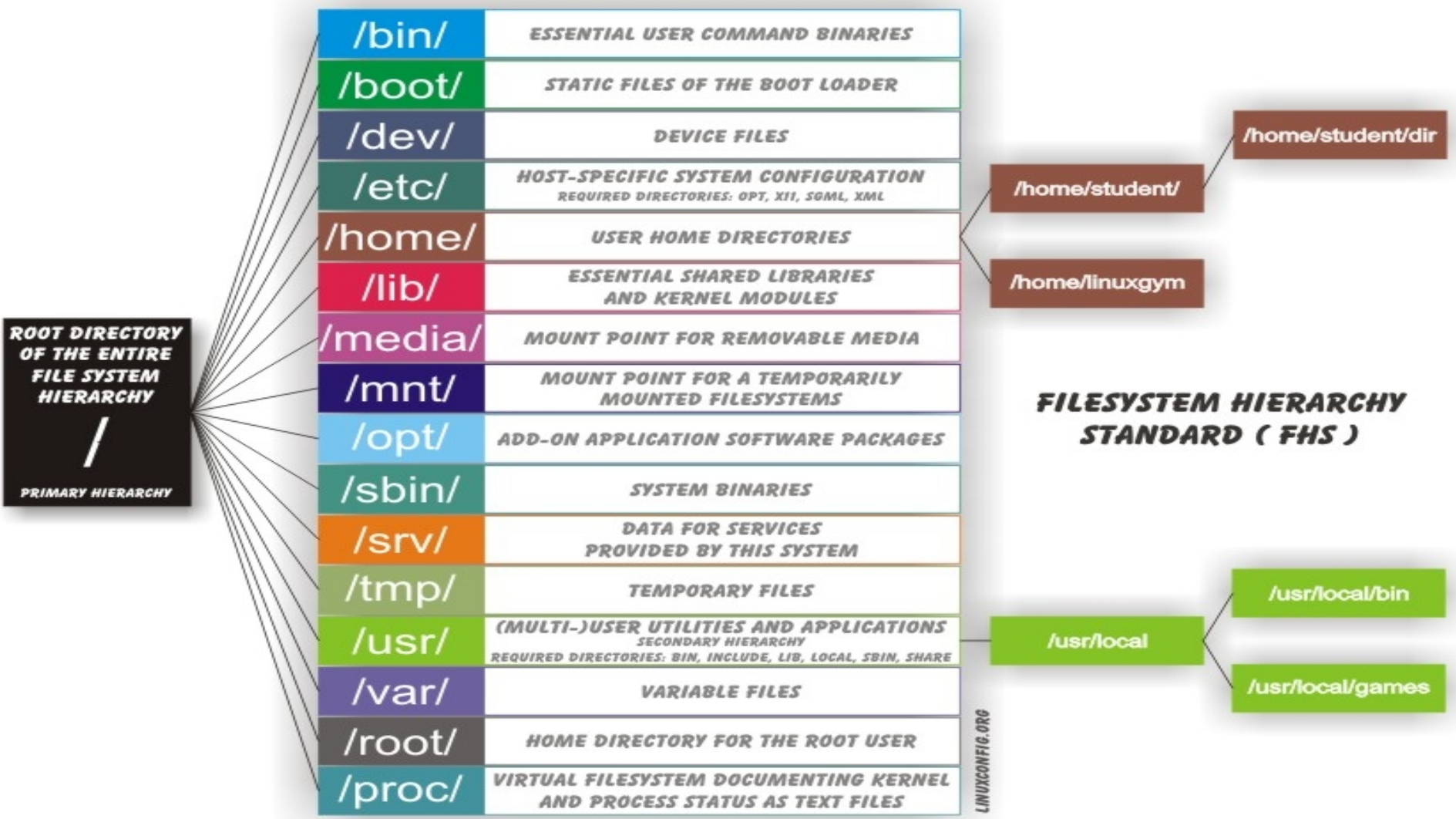
- File System: Linux is File system based OS. It supports various tyes of file system. The first node of linux file system is root **'/'** and the file system mounted to '/' is called root file system

Linux Basics

Linux Basics

- Linux Structure
- Basic Linux Commands
- Shell Scripting Basics

Linux Directory Structure



- **`ls`**, Give a listing of the current directory. Try also `ls -l`
- **`cp`**, Copy file from sorce to destination
- **`mv`**, Move file from source to destination. If both are the same directory, the file is renamed
- **`vi`**, Edit a file. `vi` is one of the most powerful text editors
- **`chmod`**, Change file permissions
- **`mkdir`**, **`rmdir`** Make/Remove a directory
- **`cd`**, Change directory
- **`rm`**, Remove a file. Can also remove directory tree
- **`man ls`**, Get help for `ls`. All commands have help

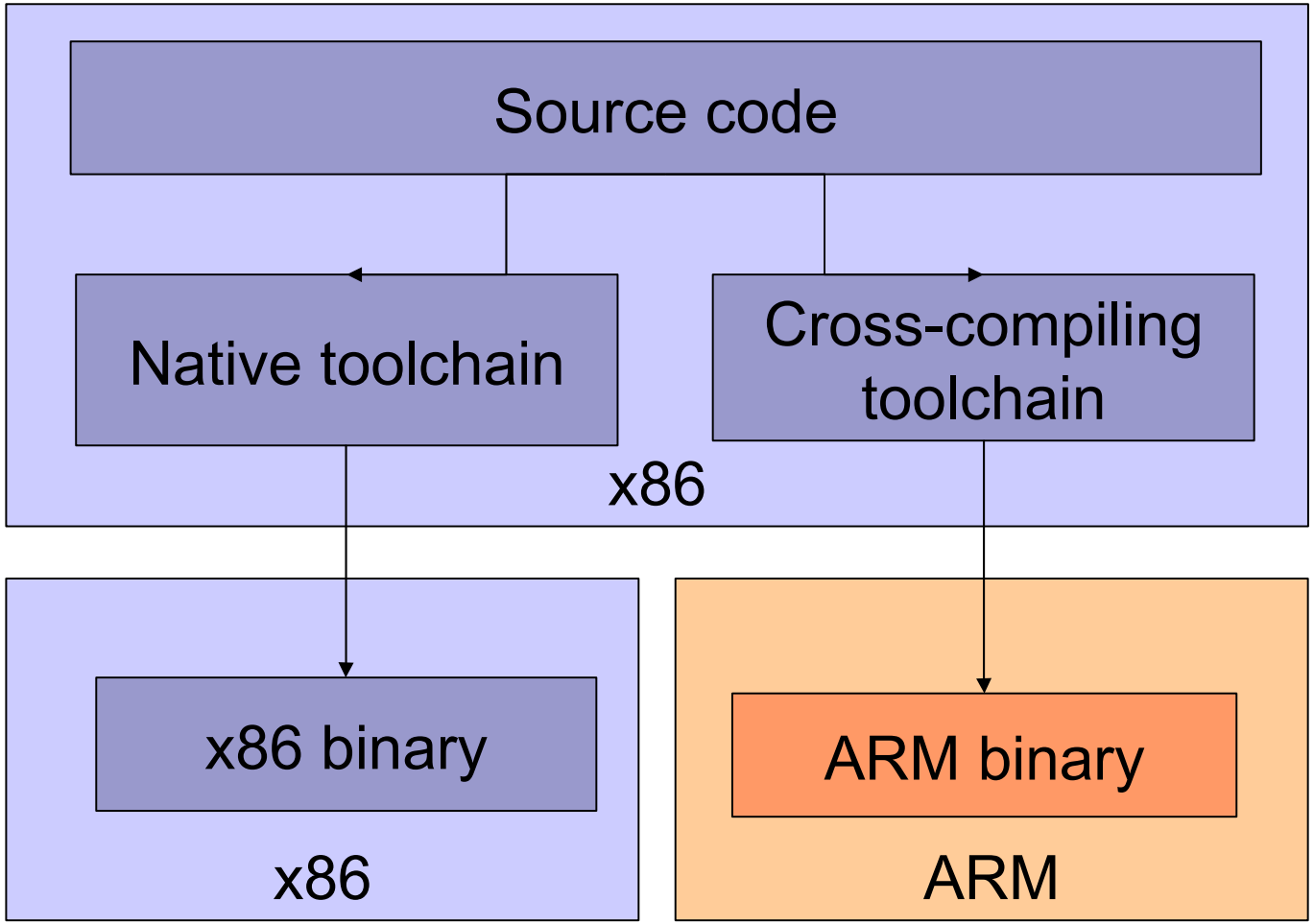
What is Shell Scripting

```
% cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo ‘Hello, world’
MY_PROGRAM
% chmod +x hello.sh
% ./hello.sh
Hello, world
```

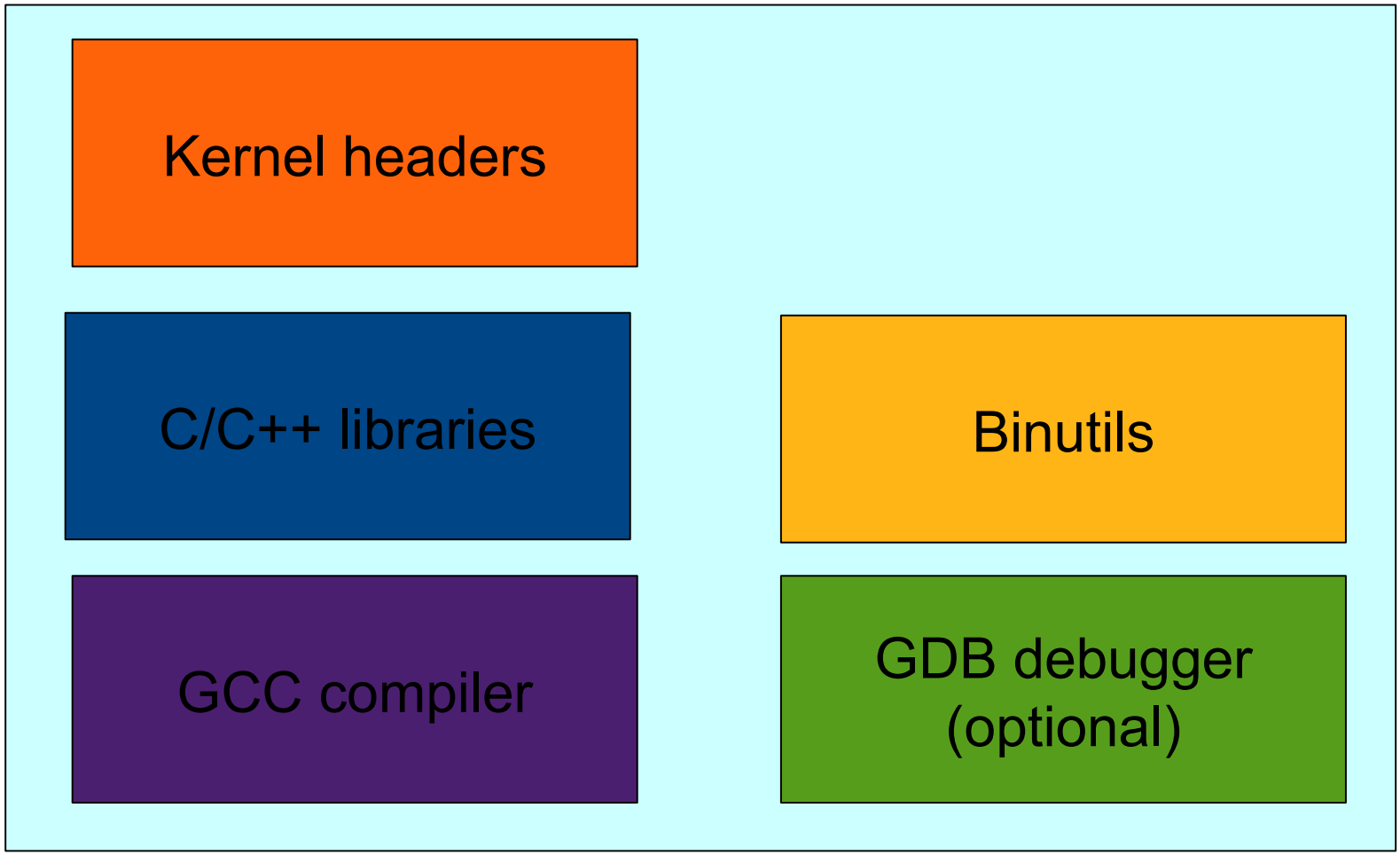
Tool Chain

- Native Compilers
- Cross Compilers

Toolchain

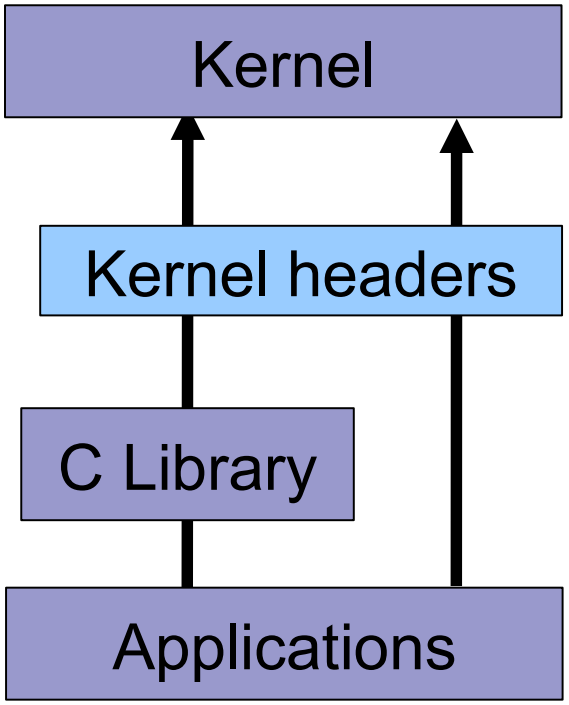

Source code
Native toolchain
Cross-compiling toolchain
x86
x86 binary
x86
ARM binary
ARM
Compilation machine
Execution machine

Toolchain components



Kernel Headers

- The C library and compiled programs needs to interact with the kernel
  - Available system calls and their numbers
  - Constant definitions
  - Data structures, etc.
- Therefore, compiling the C library requires kernel headers, and many applications also require them.



- Available in `<linux/...>` and `<asm/...>` and a few other directories corresponding to the ones visible in `include/` in the kernel sources

Binutils

- **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture
  - `as`, the assembler, that generates binary code from assembler source code
  - `ld`, the linker
  - `ar, ranlib`, to generate `.a` archives, used for libraries
  - `objdump, readelf, size, nm, strings`, to inspect binaries. Very useful analysis tools !
  - `strip`, to strip useless parts of binaries in order to reduce their size
- http://www.gnu.org/software/binutils/
- GPL license

GNU Compiler

- GNU C Compiler, the famous free software compiler
- Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc.
- http://gcc.gnu.org/
- Available under the GPL license, libraries under the LGPL.

C Library

- The C library is an essential component of a Linux system
  - Interface between the applications and the kernel
  - Provides the well-known standard C API to ease application development
- Several C libraries are available:
  glibc, uClibc, eglibc, dietlibc, newlib, etc.
- The choice of the C library must be made at the time of the cross-compiling toolchain generation, as the GCC compiler is compiled against a specific C library.

Toolchain Compiling

- **Steps to compile Toolchain Manually**
  1. Download and extract sources
  2. Patch the sources for ARM
  3. Install Kernel headers
  4. Build & Install GMP
  5. Build & Install MPFR
  6. Build & Install binutils
  7. Build & Install core GCC
  8. Build & Install glibc
  9. Build & Install full GCC

Crosstool GCC-3.4

Directory structure used:

/home/ubuntu/bb_at91/tools => top_dir

/home/ubuntu/bb_at91/tools/crosstool-0.42 => source

/home/ubuntu/bb_at91/tools/crosstool => output

/home/ubuntu/bb_at91/tools/downloads => download dir

Copy all the tars in downloads directory

Step1: Download the source in bb_at91/tools

ubuntu$ wget http://kegel.com/crosstool/crosstool-0.42.tar.gz

Step 2: Download the build.sh or create it

ubuntu$ wget http://masterarm.googlecode.com/files/build.sh

Crosstool GCC-3.4

build.sh

```
#!/bin/sh
set -ex
TOP_DIR=/home/ubuntu/tools
TARBALLS_DIR=$TOP_DIR/downloads
RESULT_TOP=$TOP_DIR/crosstool
GCC_LANGUAGES="c,c++"
PARALLELMFLAGS="-j2"
GDB_DIR=gdb-6.5
export GCC_LANGUAGES PARALLELMFLAGS TARBALLS_DIR  RESULT_TOP GDB_DIR
mkdir -p $RESULT_TOP
# Create the toolchain. This will take a lot of time.
# Add --gdb to the next line if you are going to use GDB!
eval `cat arm-softfloat.dat gcc-3.4.5-glibc-2.3.6.dat`  sh all.sh --notest
echo Done.
```

Crosstool GCC-3.4

Step 3: sudo apt-get install patch bison flex gcc-3.4

Step 4: Change the gcc link to gcc-3.4

```
ubuntu$ sudo rm /usr/bin/gcc
ubuntu$ sudo ln -s /usr/bin/gcc-3.4 /usr/bin/gcc
```

Step 5: Change the default shell to bash from dash

```
ubuntu$  ls -l /bin/sh
lrwxrwxrwx 1 root root 4 2009-10-23 22:13 /bin/sh -> dash

ubuntu$ sudo rm /bin/sh
ubuntu$ sudo ln -s /bin/bash /bin/sh
ubuntu$  ls -l /bin/sh
lrwxrwxrwx 1 root root 4 2009-10-23 22:13 /bin/sh -> bash
```

Crosstool GCC-3.4

Step 6: Add execute permission to build.sh

```
ubuntu$ chmod +x build.sh
```

Step 7: Execute the build.sh to start compiling

```
ubuntu$ ./build.sh
```

If all things goes fine U will get print "**Done**" on console.

Credits

Most of the data for this presentation is taken from

1. http://free-electrons.com/
2. http://wiki.emqbit.com/