EEEN301 Embedded systems

Lecture 16 – Introduction to Embedded Linux

Embedded Linux Overview

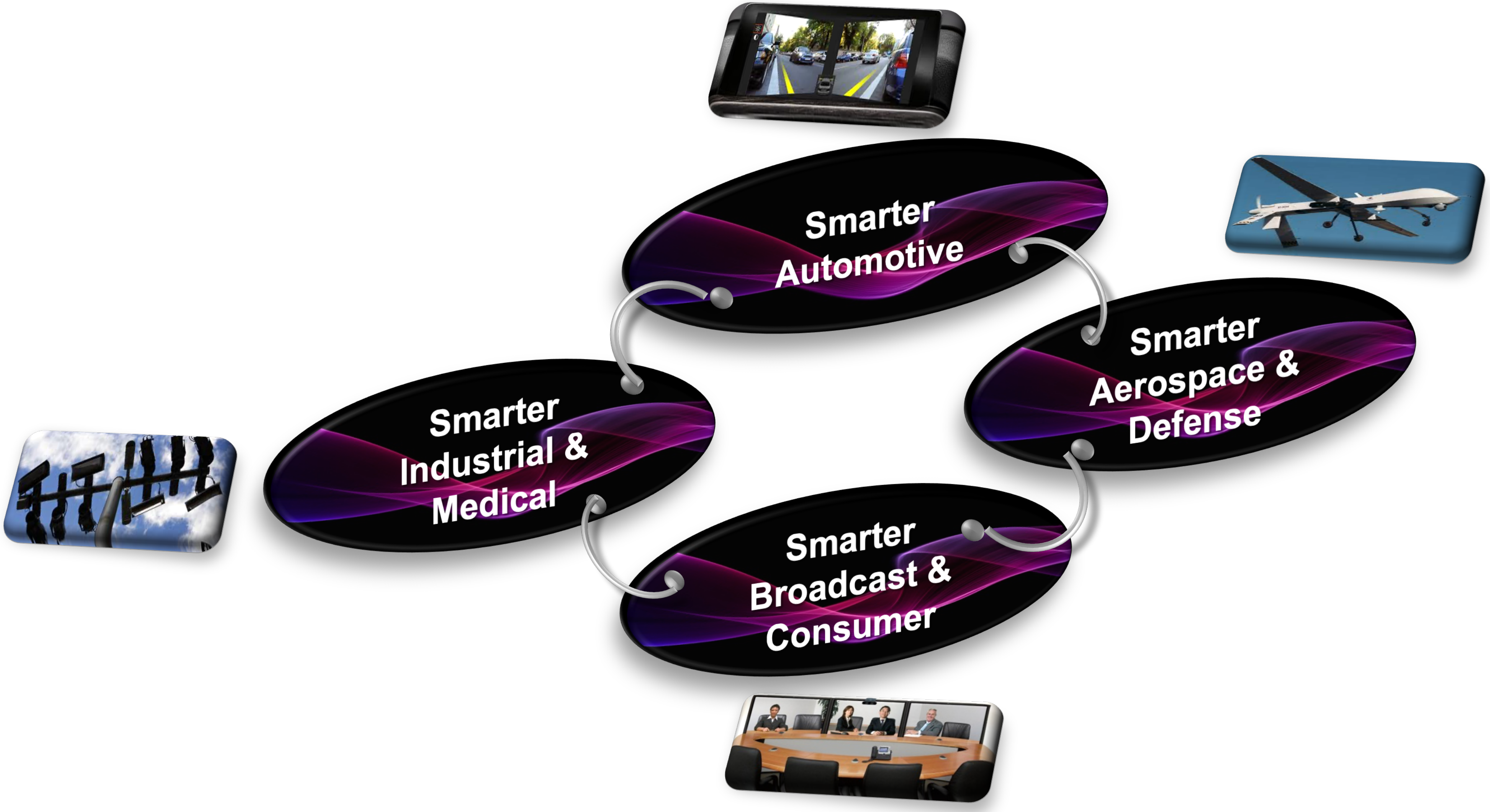
Objectives

- **After completing this module, you will be able to:**
  - Describe the kernel architecture and device driver model
  - Identify embedded development and tool chain issues
  - Describe the similarities and differences between embedded and desktop Linux

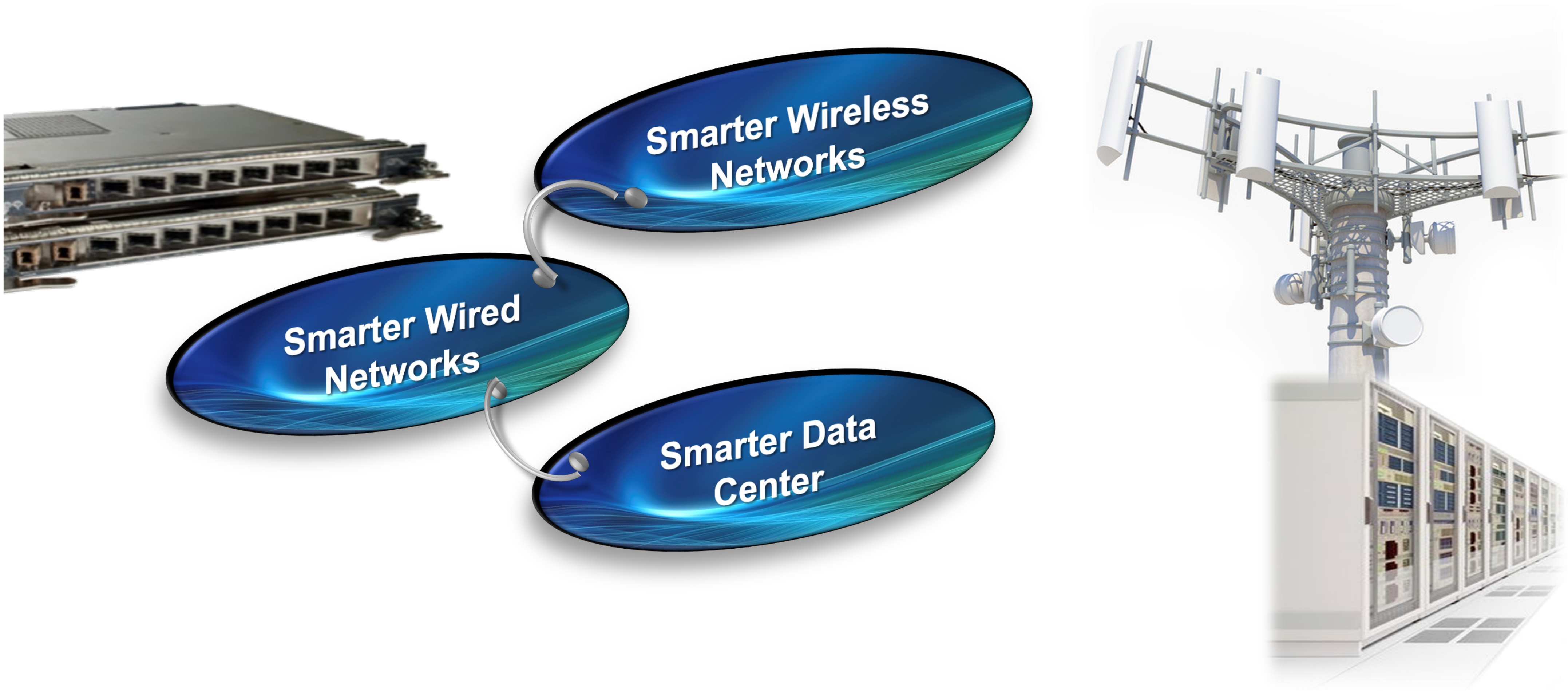
Outline

- ***Reasons for Using Embedded Linux***
- **History and Development**
- **Architecture Overview**
- **Embedded Linux Considerations**
- **Summary**

Where is Embedded Linux Used?



Where is Embedded Linux Used?



Why use Linux?

- Low cost of ownership
- Open-source, full control and customizability
- Rich feature set
- Quality of Linux components
- Re-use of Linux components
- Big active developer community & resources
- Pool of talented embedded Linux developers

What does Embedded Linux Offer? (1)

- **Processor Architecture Support**
  - ARM, MIPS, PowerPC, x86, MicroBlaze, SuperH, and others
  - 32 and 64-bits architecture support
  - Single and multi-processor support
- **Scalable Linux Image Size**
  - Linux Kernel image and file system could be as small as 4MB
- **Fast Boot time**
  - Basic Linux system can cold boot as fast as ~4 s

What does Embedded Linux Offer? (2)

- **Security**
  - SE Linux (Security-Enhanced)
- **Networking**
  - Support for Ethernet, Wifi, Bluetooth, USB, PCIe and other interfaces
  - Networking stack, libraries and application
- **Graphics Support**
  - X Server, DirectFB & DRM/KMS libraries
  - GTK & embedded Qt toolkits
- **Multimedia and Video framework**
  - Video4Linux, GStreamer frameworks
  - FFMPeg, OpenCV, alsa-lib, and other libraries

What does Embedded Linux Offer? (3)

- **Interoperability and OS infrastructure**
  - Networking, file systems, device support
  - Scheduling, processes/threads, interrupt handling, interprocess communication, symmetric multiprocessing (SMP)
- **Portability**
  - X86, Alpha, MIPS, PowerPC® processor, MicroBlaze™ processor, ARM, or ColdFire, for example

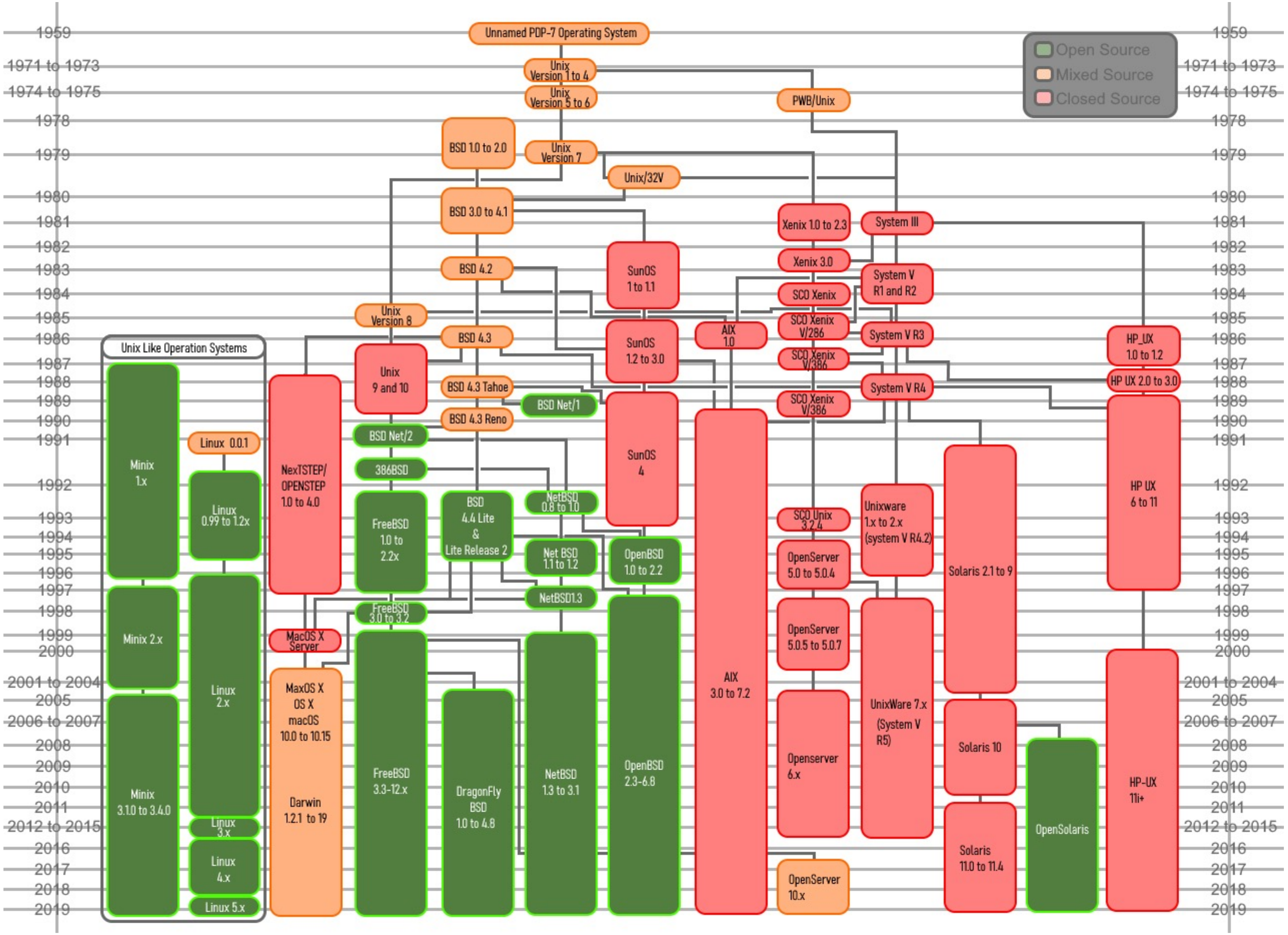
What does Embedded Linux Offer? (4)

- **Developer familiarity**
  - Standard tools
  - Standard run-time environment
  - Application code can be prototyped on the desktop
- **Scalability**
  - Deeply embedded → single board computers → desktop → server → cluster
- **Freedom and openness**
  - Accessibility to modify source code

Outline

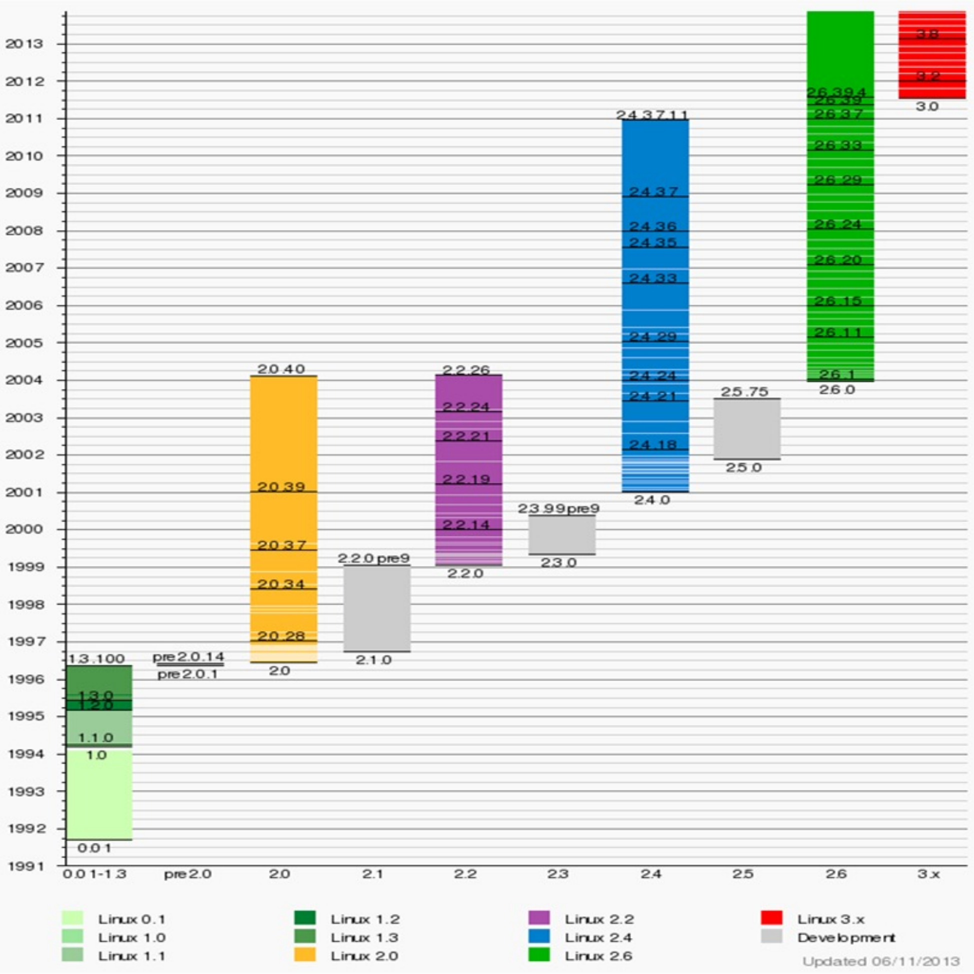
- Reasons for Using Embedded Linux
- *History and Development*
- Architecture Overview
- Embedded Linux Considerations
- Summary

Before Linux there was UNIX



Linux History and Development

- **25th August, 1991**
  - “I’m doing this hobby project called Linux”
    - Linus Torvalds, comp.os.minix
- **1992 Linux 0.95**
- **1994 Linux 1.0**
- **1996 Linux 2.0**
- **1999 Linux 2.2**
- **2001 Linux 2.4**
- **2003 Linux 2.6**
- **June 2010 Linux 2.6.35**
- **March 2012 Linux 3.3**
- **Jan 2013 Linux 3.13**



XILINX ALL PROGRAMMABLE.

Linux Porting to Various Processors

- **Some milestones**
  - 1995 Linux 1.3.94
    - First non-x86 architecture merged (m68k)
  - 1998 - Linux 2.0 era
    - IBM S/390
    - First uClinux port
    - Linus Torvalds on cover of Forbes magazine
  - 2009 - Linux 2.6.31
    - > 20 processor architectures
    - Thousands of device drivers
    - MicroBlaze processor in mainline kernel distribution
  - 2011 – Linux 2.6.37

Linux Kernel Tree

- **There is only ONE Linux Kernel tree**
  - Mainline or linus or kernel.org
  - Includes Linux kernel and drivers
  - Hosted in a Git repository
  - Everyone derives (or clone/merge) their Linux kernel from mainline; including Xilinx
  - Contributors, including Xilinx, push (or upstream) their source code to mainline
  - Thousands of options to customize the kernel

Outline

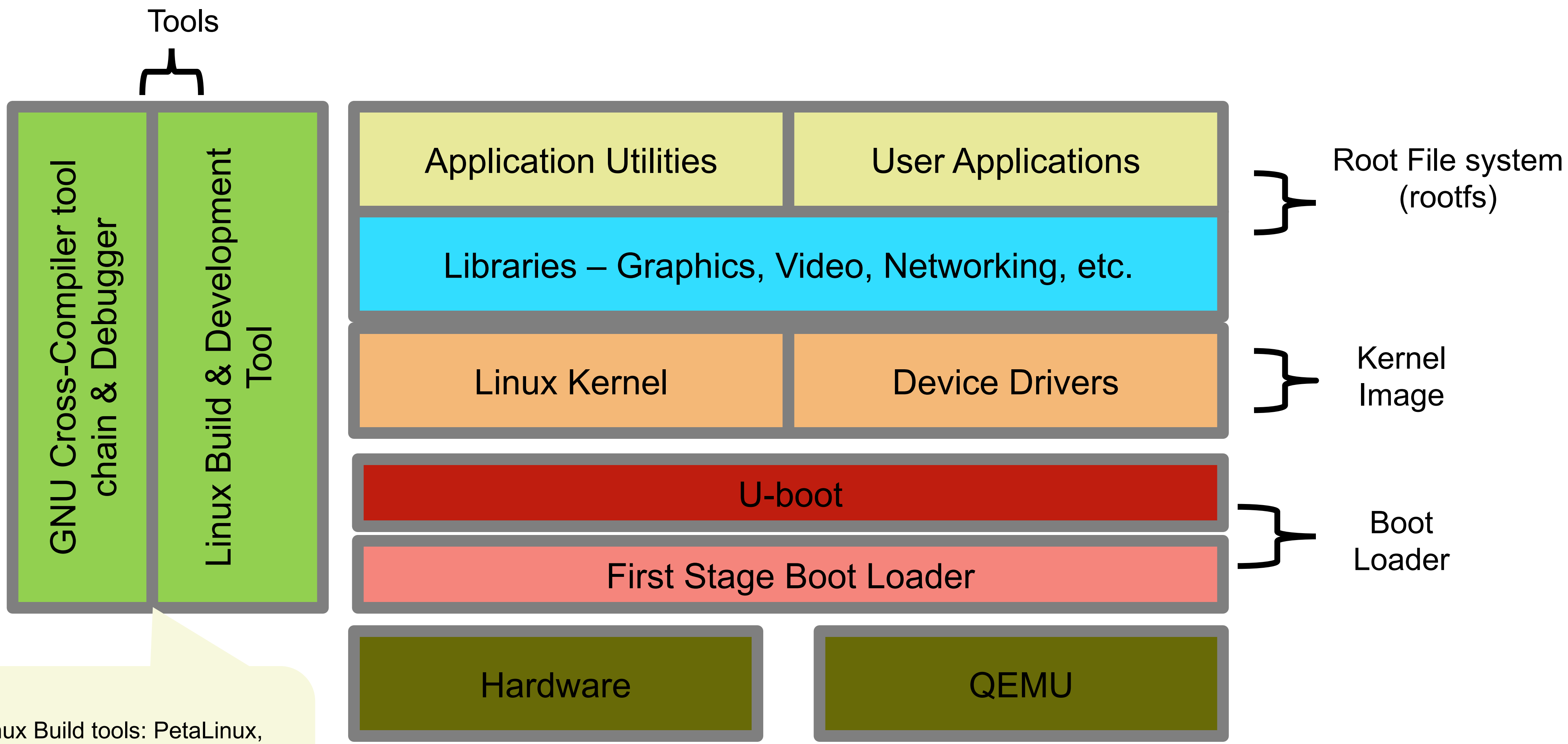
- **Reasons for Using Embedded Linux**
- **History and Development**
- ***Architecture Overview***
- **Embedded Linux Considerations**
- **Summary**

Major components of a Linux OS

Every embedded Linux system needs four major components to work:

- Linux Kernel, with all the device drivers
- Root filesystem, which contains all the applications and libraries
- Bootloader, which is responsible for the initial boot process of the system, and for loading the kernel into memory
- Toolchain, which doesn’t usually run on the target platform, but allows to generate code for the target from a development machine.

Linux Components

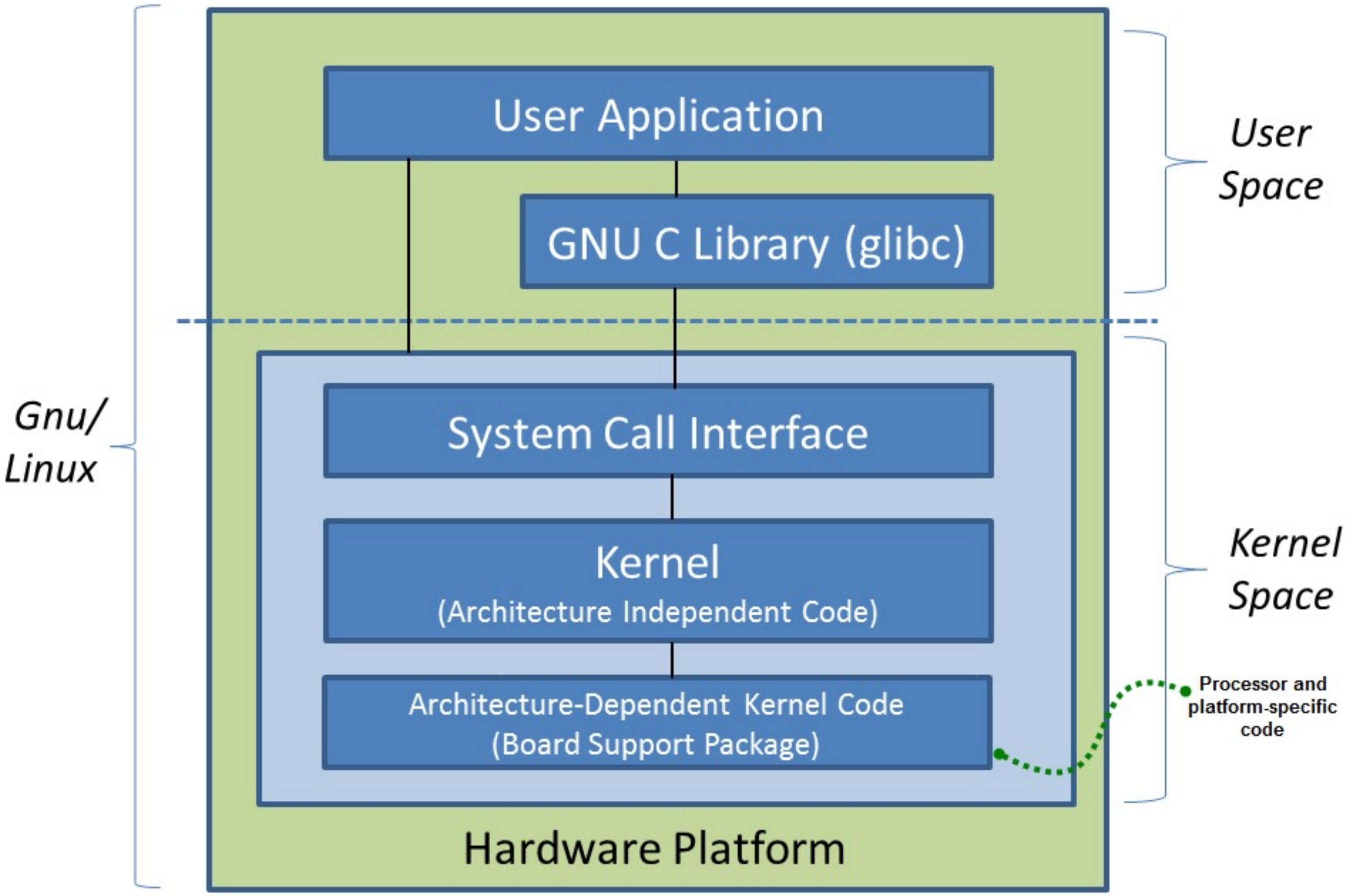


Linux Build tools: PetaLinux, Yocto, OpenEmbedded, buildroot
Other tool chains: Linaro, GCC mainline

Board Support Package (BSP): Collection of SoC/board specific Linux components that include Linux kernel & device drivers, boot loader and tools

Anatomy of a Linux System

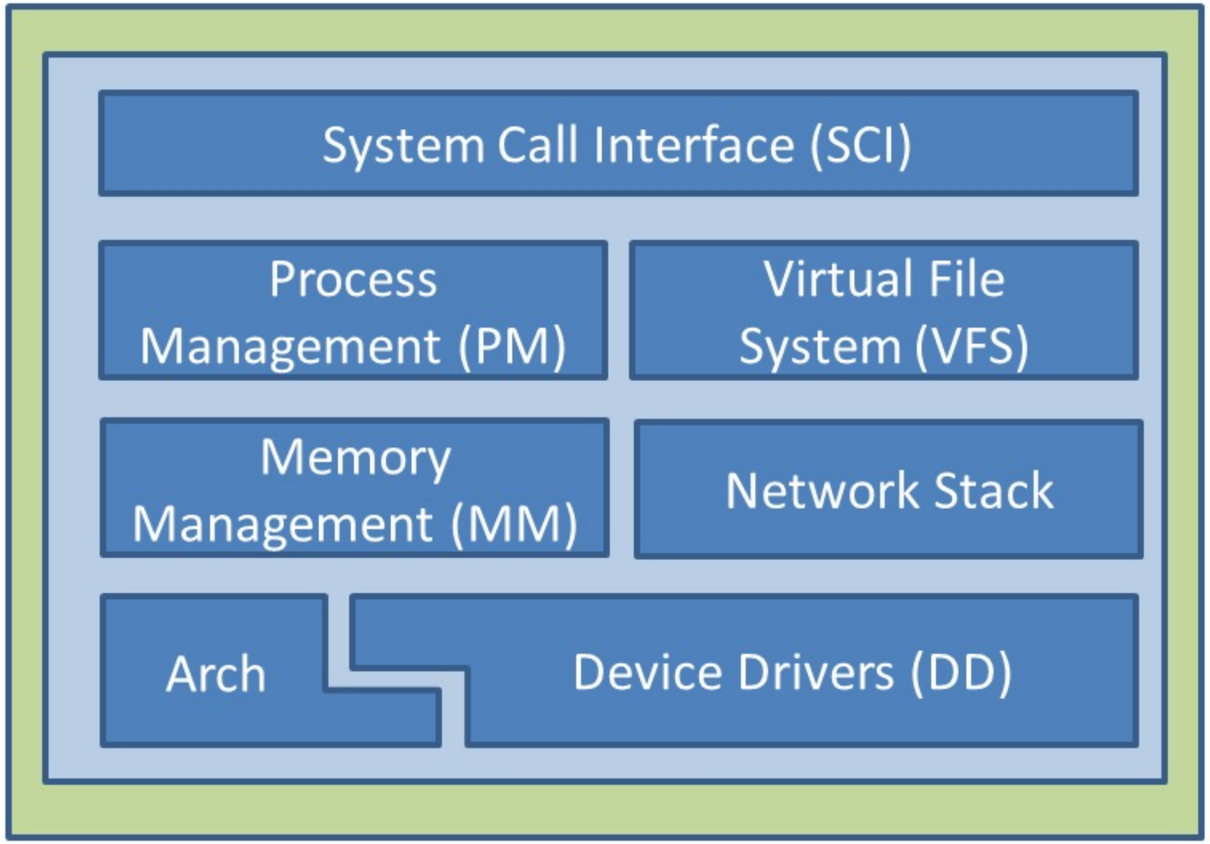
- **User applications**
  - Task specific
- **Libraries**
  - C run time, networking, graphics, for example
- **Kernel**
  - More on this soon
- **Hardware**
  - CPU, memory
  - Timer, interrupt controller, I/O



Source : IBM DeveloperWorks

Linux Kernel Architecture Overview (1)

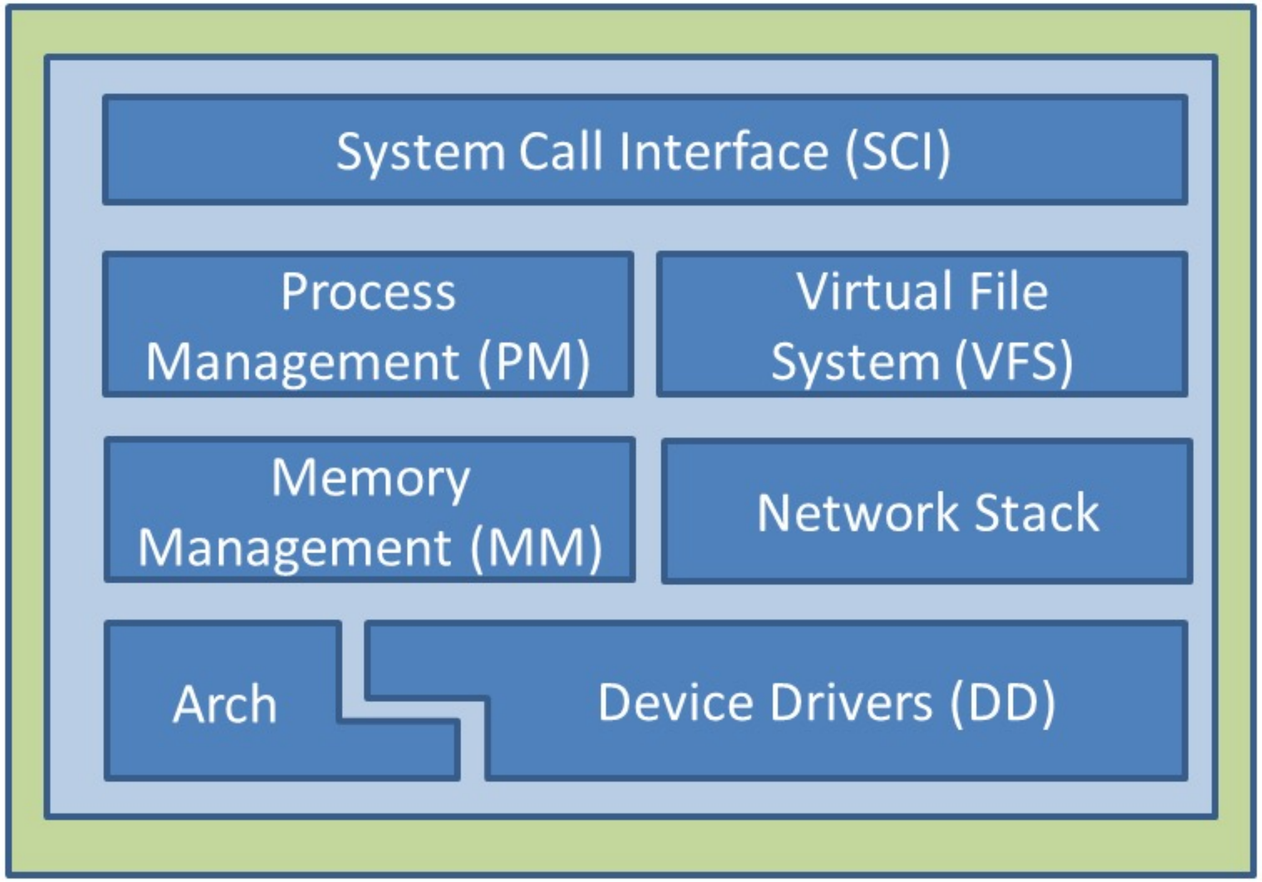
- **System call interface (SCI)**
  - Function calls from user space to kernel
- **Process management**
  - Processes or threads
  - Share the CPU between the active threads
- **Memory management**
  - Manages the virtual and physical memory
- **Virtual file system**
  - Switching layer between the SCI and the file systems supported by Linux



Source : IBM DeveloperWorks

Linux Kernel Architecture Overview (2)

- **Network stack**
  - Above TCP, socket layer
  - Invoked through the SCI
- **Device drivers**
  - Supports most of the common devices
- **Architecture-dependent code (BSP)**
  - Processor and platform-specific code



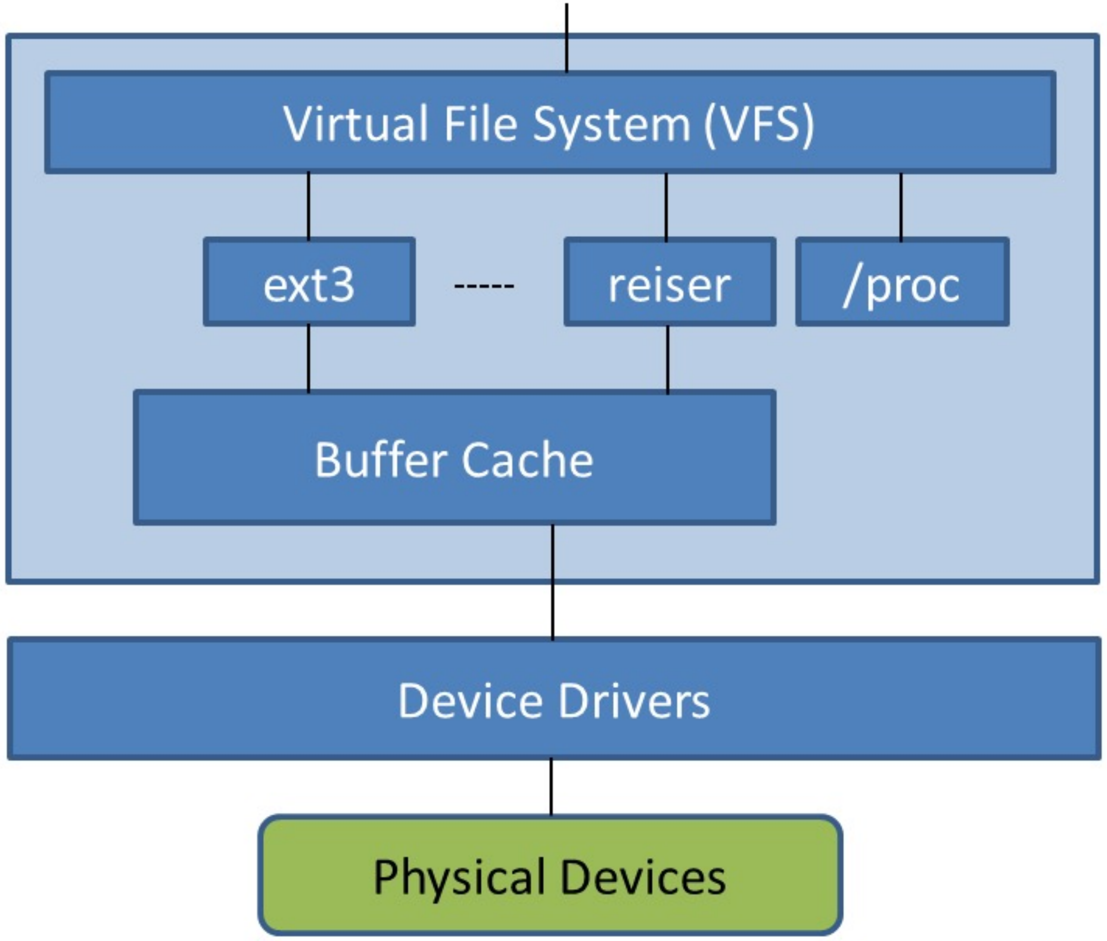
Source : IBM DeveloperWorks

System Call Interface

- **Contract between user space and kernel space**
  - Request services
    - Memory, I/O
    - Process management
- **The standard C library is a wrapper around the system call interface**
  - For example, `malloc()` is a C library call
    - Maintains a pool of memory
    - Calls kernel memory allocator only when that pool is exhausted

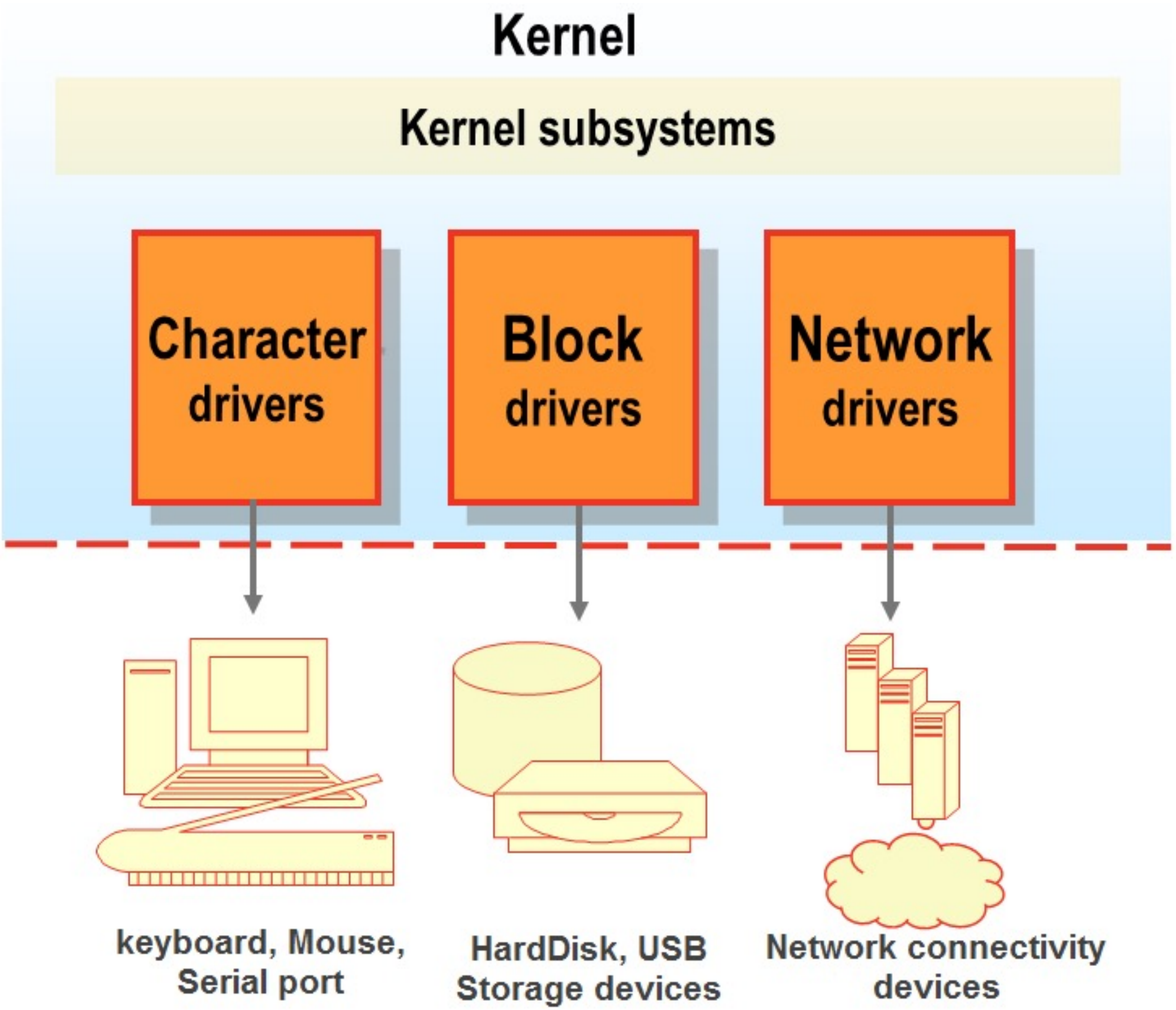
Virtual File System

- **Abstraction of standard file system API**
- **Each specific file system type implements required functionality**
- **Decouples file system format from storage media**
  - ext3/4 on SATA disk
  - FAT32 on USB drive
  - XFS on software RAID



Source : IBM DeveloperWorks

Device Drivers Types



Device Model

- **Three fundamental types**
  - char
    - Stream of bytes
    - `open(), close(), read(), write(), seek()...`
  - block
    - Byte-accessible interface to block-based devices
  - net
    - Packet oriented
    - Asynchronous

Device Model

- **Accessing devices**
  - char and block exported through the file system as device nodes
    - `/dev/ttyS0` (serial port zero)
    - `/dev/ttyUSB0` (USB-based serial port zero)
    - `/dev/hda` (first IDE disk)
    - `/dev/sda` (first SATA disk)
  - Network devices accessed by name
    - For example, `eth0`, `wifi0`, ...
    - No direct file system representation

Process Management

- **Tasks or processes are the fundamental schedulable work unit**
  - Have independent
    - memory space
    - File system context
    - Security context
    - Network context
  - Created by `fork()`
- **Threads share the same address space**
  - Low-overhead context switching
  - Better scalability for multi-CPU systems
  - Threads can corrupt each other's memory spaces

Outline

- Reasons for Using Embedded Linux
- History and Development
- Architecture Overview
- *Embedded Linux Considerations*
- Summary

Desktop vs. Embedded Linux Development

- **Cross-compilation**
  - Compiler runs on the host, not the target
  - Different run-time environment from host to target
    - Using Linux eases this greatly
  - Cross-platform file system creation
  - A good environment tries to makes this easy
- **Cross-platform compile/test/debug cycle**
- **Bootstrap process**
  - From Flash memory
  - Via network

Toolchains

- **The toolchain is comprised of**
  - Compilers (C, C++, ...)
  - Assembler, linker (binutils)
  - Debugger
  - Other supporting tools
- **GCC is the defacto standard for Linux**
  - GNU Compiler Collection
  - `gcc, as, ld, gdb`

Differences With the Toolchains

- **Linux-specific toolchain**
  - Create application binaries targeting embedded Linux
  - Link against the Linux C library, not the standalone version
- **ARM Cortex-A9 processor Linux toolchain**
  - `arm-xilinx-linux-gnueabi-gcc`
  - As well as, `-as`, `-ld`, `-gdb`, ...

Outline

- Reasons for Using Embedded Linux
- History and Development
- Architecture Overview
- Embedded Linux Considerations
- *Summary*

Knowledge Check

- What are some reasons for using Linux as an embedded OS?
- What are the considerations you must consider when targeting an embedded design with Linux?
- Why is a cross-compiler required for embedded Linux development?

Summary

- Designers choose to use an embedded Linux because of its interoperability, OS infrastructure, portability, operator familiarity, scalability, and freedom
- A Linux system includes user applications, libraries, a kernel, and embedded system hardware
- Three device types
  - char
  - block
  - net

Where Can I Learn More?

- **Linux history**
  - www.linuxjournal.com/article/6000
- **Linux kernel anatomy**
  - www.ibm.com/developerworks/linux/library/l-linux-kernel
- **Publications**
  - Linux Weekly News (www.lwn.net)
  - Kernel Trap (www.kerneltrap.org)
  - Linux Journal(www.linuxjournal.com)
  - Linux Devices (www.linuxdevices.com)