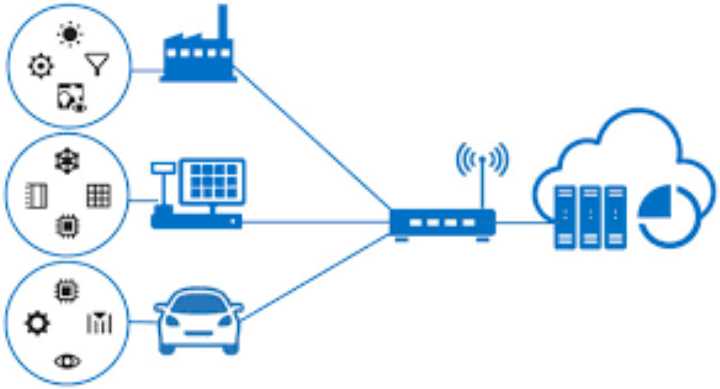
IoT & Embedded Software

조 진 성
경희대학교 컴퓨터공학과
Mobile & Embedded System Lab.

Internet of Things



Internet of Things

IoT (Internet of Things)

유사 정의

ITU-T

A global **infrastructure** for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies.
In a broad perspective, the IoT can be perceived as a vision with technological and societal implications.

Gartner

The Internet of Things is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment.

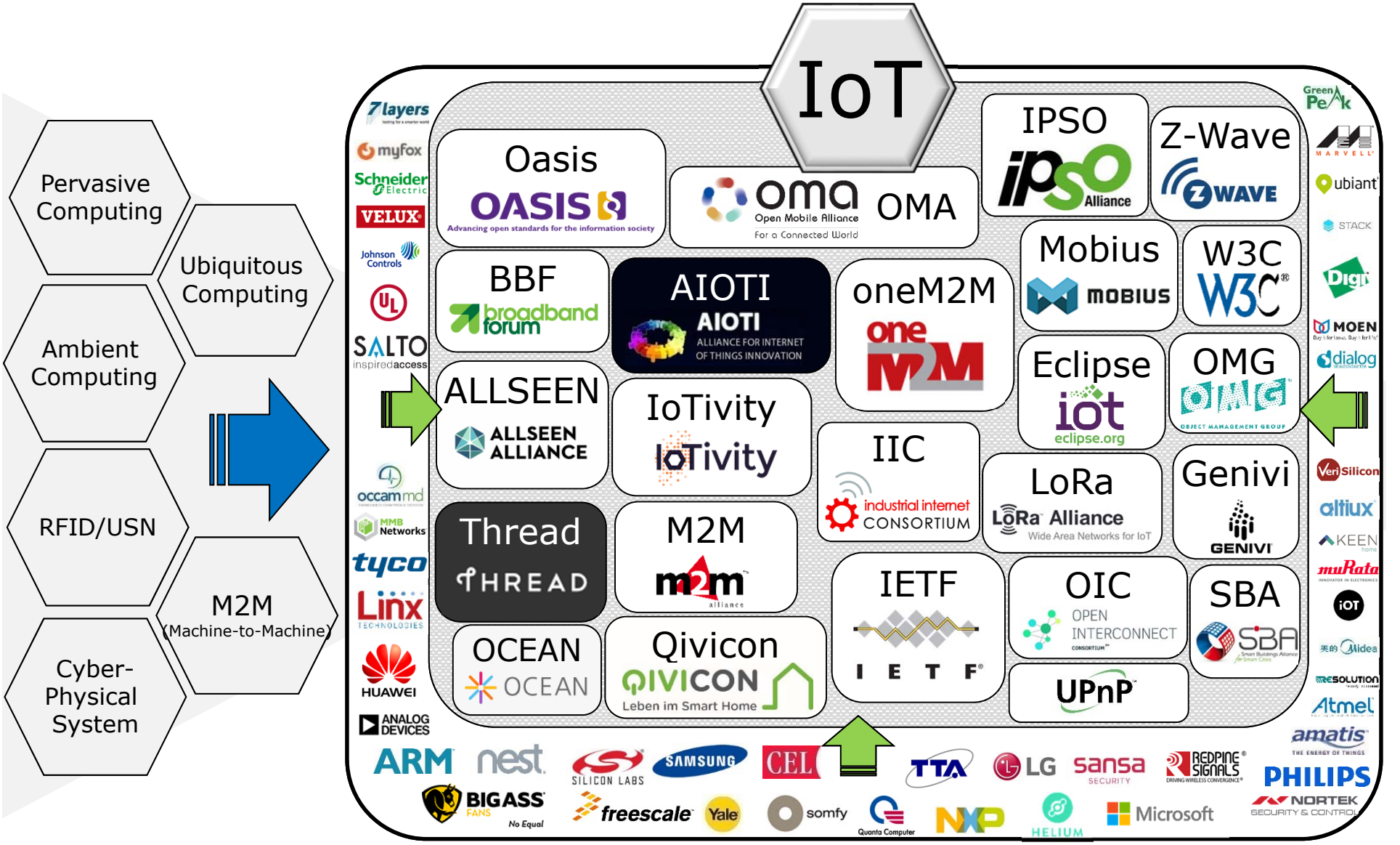
IBM

Internet-of-Things (IoT) provides the foundational infrastructure for a smarter planet, and offers significant growth opportunities in IT, infrastructures and services.

Internet of Things 변천사

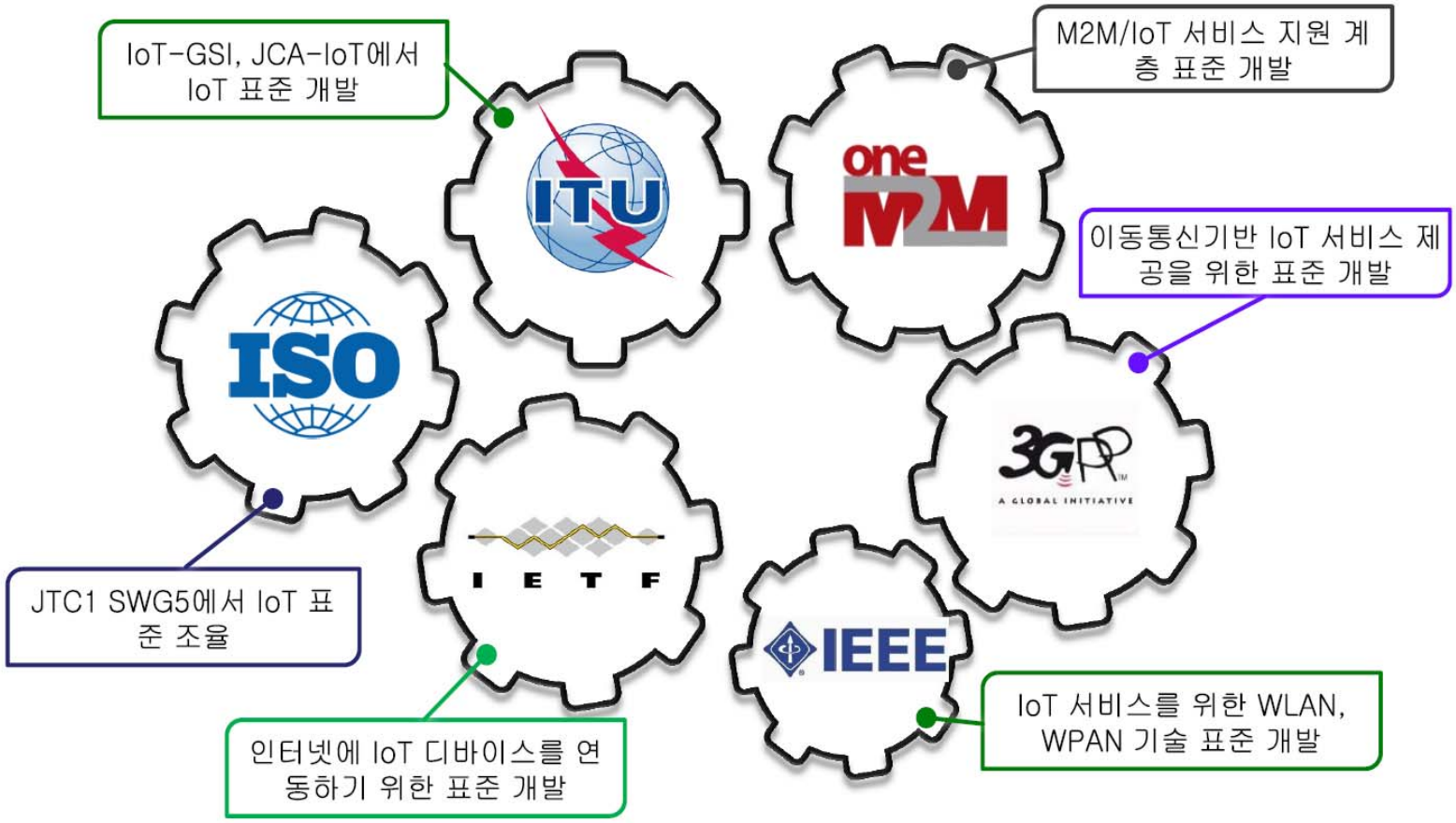
- Ubiquitous Computing (1988)
  - Pervasive Computing / Ambient Computing
  - Cyber Physical System
- Wireless Sensor Network (2000)
  - RFID/USN
  - M2M (Machine-to-Machine)
- Internet of Things (2010)
  - Internet of Everything
  - Intelligence of Things
  - ???
- ???

IoT 세상


Pervasive Computing
Ubiquitous Computing
Ambient Computing
RFID/USN
M2M (Machine-to-Machine)
Cyber-Physical System
IoT
Oasis
OMA
IPSO
Z-Wave
BBF
AIOTI
oneM2M
Mobius
W3C
Eclipse
OMG
ALLSEEN
IoTivity
IIC
LoRa
Genivi
Thread
M2M
IETF
OIC
SBA
OCEAN
Qivicon
UPnP
7layers
myfox
Schneider Electric
VELUX
Johnson Controls
SALTO
occamd
MMB Networks
tyco
Linx
HUAWEI
ANALOG DEVICES
GreenPeak
MARVELL
ubiant
STACK
Digi
MOEN
dialog
VeriSilicon
altiux
KEEN
muRata
iOT
Midea
RESOLUTION
Atmel
amatis
PHILIPS
NORTEK
ARM
nest
SILICON LABS
SAMSUNG
CEL
TTA
LG
sansa
REDPINE SIGNALS
BIGASS FANS
freescale
Yale
somfy
Quanta Computer
NXP
HELIUM
Microsoft

IoT Standard

- For Connectivity



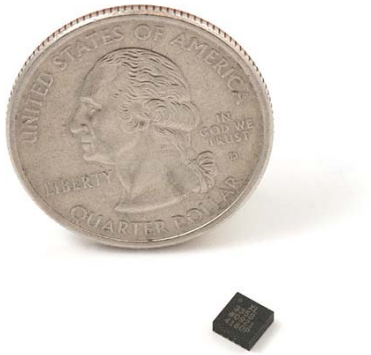
IoT 요소 기술

- Sensor Technology – Tiny, Cheap, Variety
- Cheap Miniature Computers
- Low Power Connectivity
- Capable Mobile Devices
- Cloud Service

출처: Shahriar Nirjon

IoT 요소 기술

- Sensor Technology – Tiny, Cheap, Variety



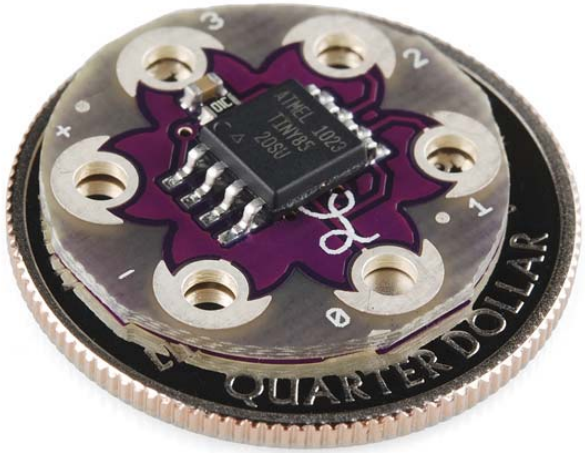
Accelerometer
(4mm diameter)

Force Sensor
(0.1N – 10N)

Pulse Sensor
$25

IoT 요소 기술

- Cheap Miniature Computers



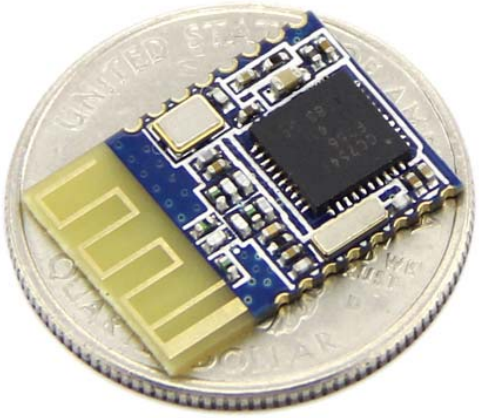
Lily Tiny
$4.95

Key Parameters

Flash: 8 Kbytes
Pin Count: 8
Max. Operating Freq: 20 MHz
CPU: 8-bit AVR
Max I/O Pins: 6
Ext Interrupts: 6
SPI: 1
I2C: 1

IoT 요소 기술

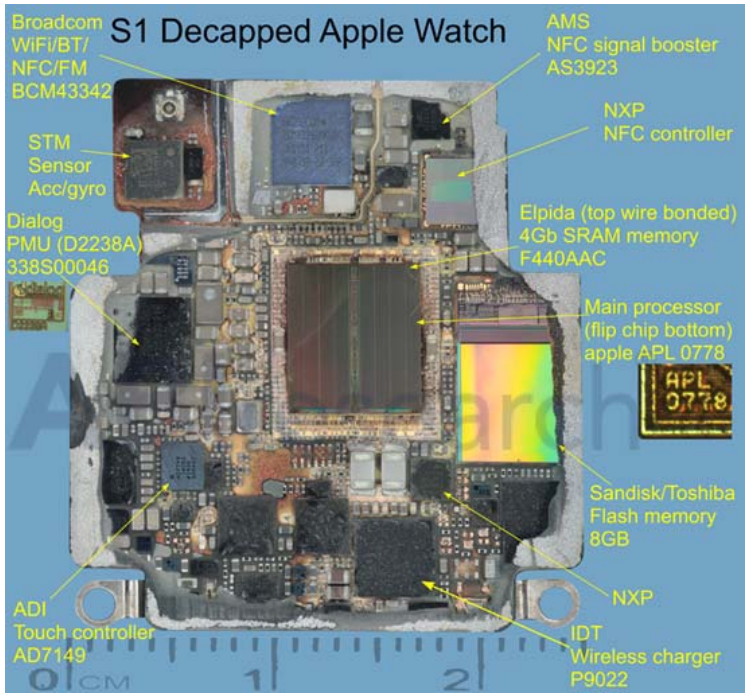
- Low Power Connectivity



Bluetooth Smart (4.0)
(Up to 2 years with a single
Coin-cell battery)

IoT 요소 기술

- Capable Mobile Devices



Quad Core 1.5 GHz
128 GB Internal Memory
3 GB RAM
16 MP Camera
2160p@30fps video
WiFi, GPS, BLE

IoT 요소 기술

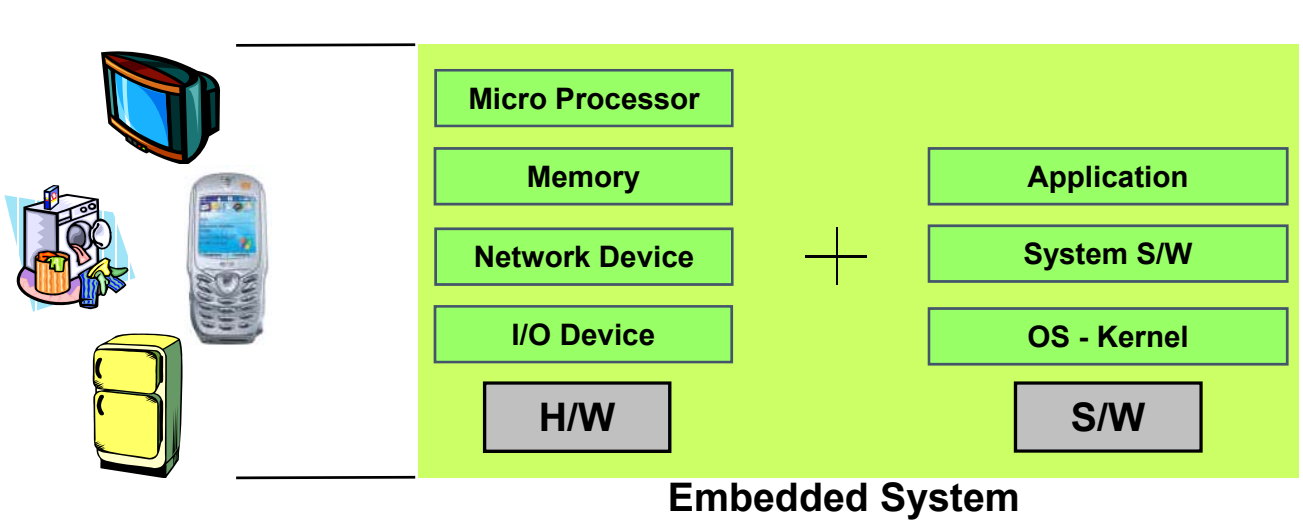
- Cloud Service
  - IaaS / PaaS / SaaS

IoT 요소 기술 @ KHU

- Embedded System Hardware
  - IoT **디지털 시스템**
- Embedded System Software
  - IoT **소프트웨어**
- Cloud Computing Software
  - **클라우드 컴퓨팅**

What is Embedded System ?

- Embedded system
  - Computer inside a product (or a system)
  - Device that includes a programmable computer but is not itself a general-purpose computer
  - Computer H/W and S/W system that perform a specific-purpose



Embedded System Cases

- FA (Factory Automation)
  - Unmanned system consisting of sensors, control systems, and robots
  - Driving forces behind real-time systems and embedded systems
  - Increased productivity
  - Robot, conveyor belt

Embedded System Cases

- Aircraft
  - Hundreds of processors are embedded
- Space shuttle
  - Pathfinder runs on VxWorks (famous real-time embedded OS)
  - Typical application of real-time systems
  - Various functions including image processing, communications, …

NASA Pathfinder
(mission to MAR 1997)

Embedded System Cases

- Networking devices
  - Voice service devices such as digital switch, PABX (Private Automatic Branch Exchange)
  - Data service devices such as router, gateway, AP, …
  - Set-Top Box

Embedded System Cases

- Logistics
  - POS (Point Of Sales) terminal
- Finance
  - ATM machine
- Office machine
  - Combination printer, scanner, fax, copy machine, …

Embedded System Cases

- Mobile terminal devices
  - Perform complex functions including information retrieval, entertainment, messaging, game
  - Consist of microprocessor, memory, OS, and applications optimized for given functions
  - Are expected various terminals may be integrated into one device
  - Cellular phone, PDA, Smart phone, MP3 player, PMP, …

Embedded System Cases

- Consumer Electronics
  - Multiple functions in a product
  - Application of embedded system technology for the multiple functions
  - Remote control and information gathering in Home network
  - Home automation, home networking, …
  - Internet Refrigerator, IPTV, Internet Microwave, Internet Washer, …

Smart TV
Internet surfing

Boiler
Internet remote control

Washer
Internet remote control

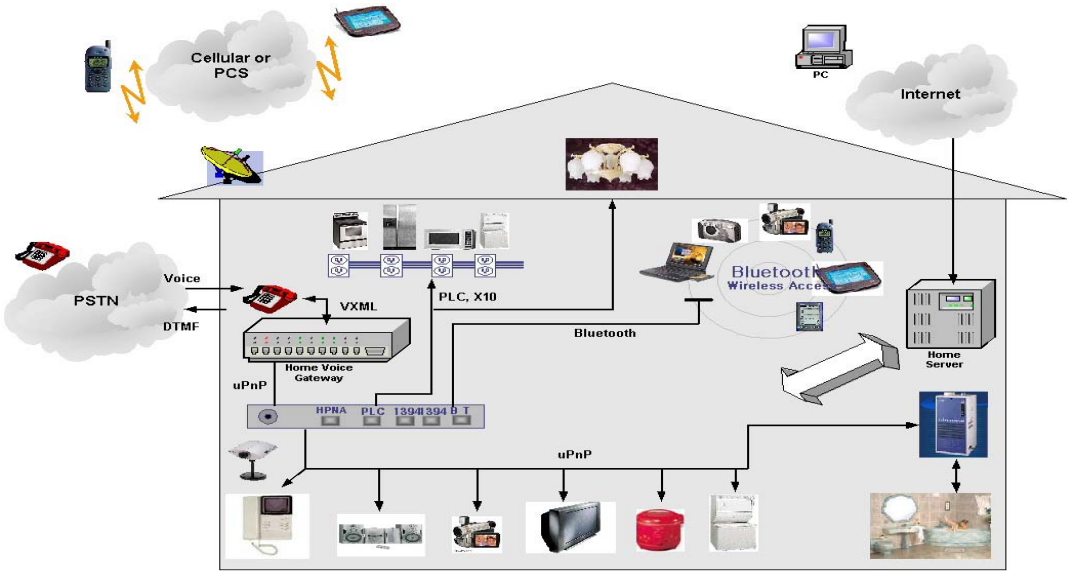
Camcoder
Digital camera, networking, multiple functions

Microwave
Internet download of food information

Refrigerator
Internet remote control & download

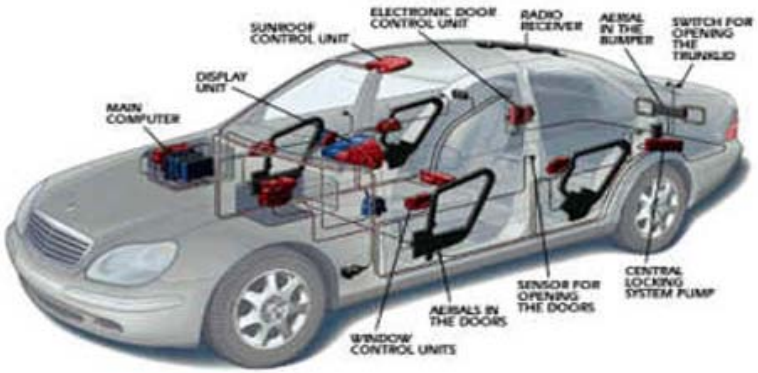
Embedded System Cases

- HA (Home Automation)
  - Remote control to everything at home
  - Web pad, voice recognition, …
  - All the devices are smart embedded systems and interconnected through home network



Embedded System Cases

- Transport
  - Smart car
  - ITS : Intelligent Transport Systems
- Intelligent toy
  - Evolution from simple toy to intelligent one



Embedded System Cases

- Game machine
  - MS Xbox
  - Sony Playstation 2
  - Nintendo Gameboy
  - Nintendo Wii
- Medical system
  - ABI PRISM 3700 DNA Analyzer

Embedded System Hardware

- PC Hardware vs. Embedded System Hardware

Hardware	PC	Embedded System
Processor	High performance	Minimum performance
Main Memory	Large size	Minimum size
Secondary Storage	Variety: HDD,ODD,DVD	Flash memory
I/O Devices	Variety: KBD,MOUSE,SPKR	Only devices needed

Embedded System Software

- PC Software vs. Embedded System Software

Software	PC	Embedded System
OS	Windows, Linux	Real-Time OS, Embedded Linux, Widows CE, …
System call	Windows API	Real-Time OS & Embedded Linux API
Application Software	Stored in HDD	Stored in flash memory
Development Environment	Native-Development	Cross-Development

Embedded System Industry

- **고부가가치 산업**
  - **임베디드 소프트웨어가 제품의 가치를 결정하는 기술집약적 선업**
    - 시스코 라우터의 경우, 하드웨어 원가는 수십 만원에 불과하나 각종 통신 및 제어 소프트웨어가 탑재되면 최종가격이 수백만원으로 상승함

- **대한민국 제조업의 장점 활용**
  - **제품에 지능을 불어 넣어 고부가가치화**
  - Digital converged consumer electronics, mobile phones, smart cars, …

Embedded System Industry

▣ 대규모의 임베디드 시스템 소프트웨어 개발 인력 필요

⊙ 임베디드 시스템 하드웨어에 대한 이해

- Microprocessor, Memory, various devices
- Hardware & Software co-design

⊙ 임베디드 시스템 소프트웨어 개발 능력

- **기본적인 프로그래밍 능력**
- **교차개발환경**
- RTOS / Embedded Linux
- Embedded Middleware
- Embedded Application

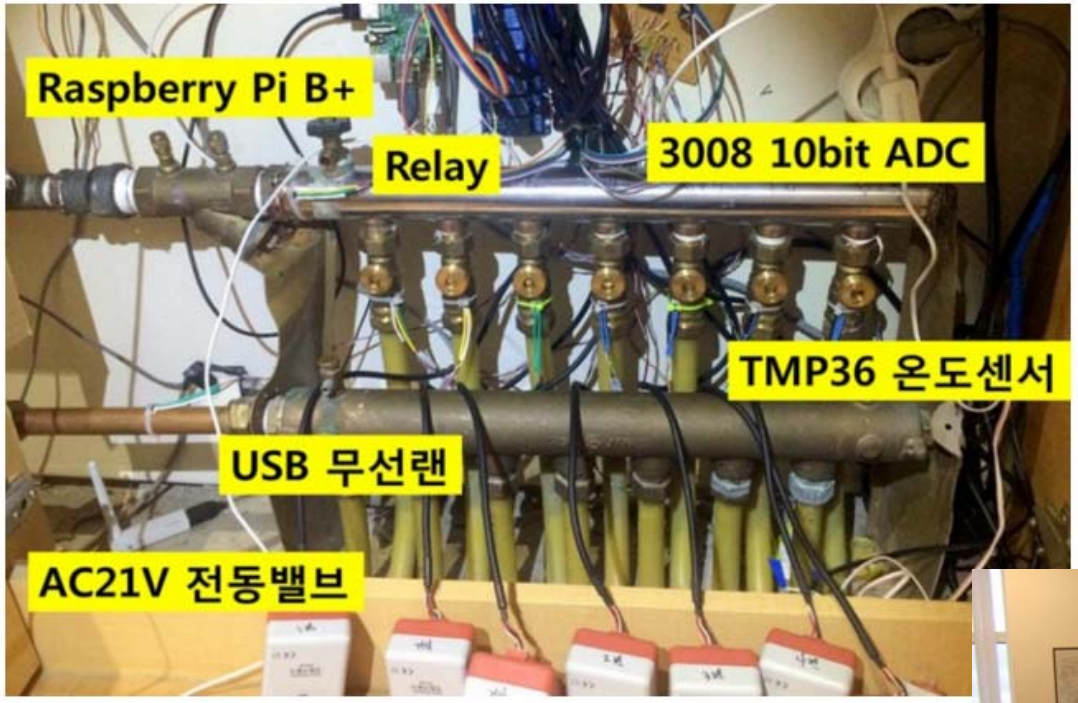
교과목 변천사

- 2004 **임베디드 시스템**
- 2005 **임베디드 시스템**1 & **임베디드 시스템**2 (**내용 및 실습장비 변경**)
- 2007 **임베디드 시스템**1 & **임베디드 시스템**2 (**내용 및 실습장비 변경**)
- 2010 **임베디드 시스템**1 & **임베디드 시스템**2 (RTOS)
- 2012 **임베디드 소프트웨어**
- 2018 IoT **소프트웨어**

Embedded Software 개발 환경

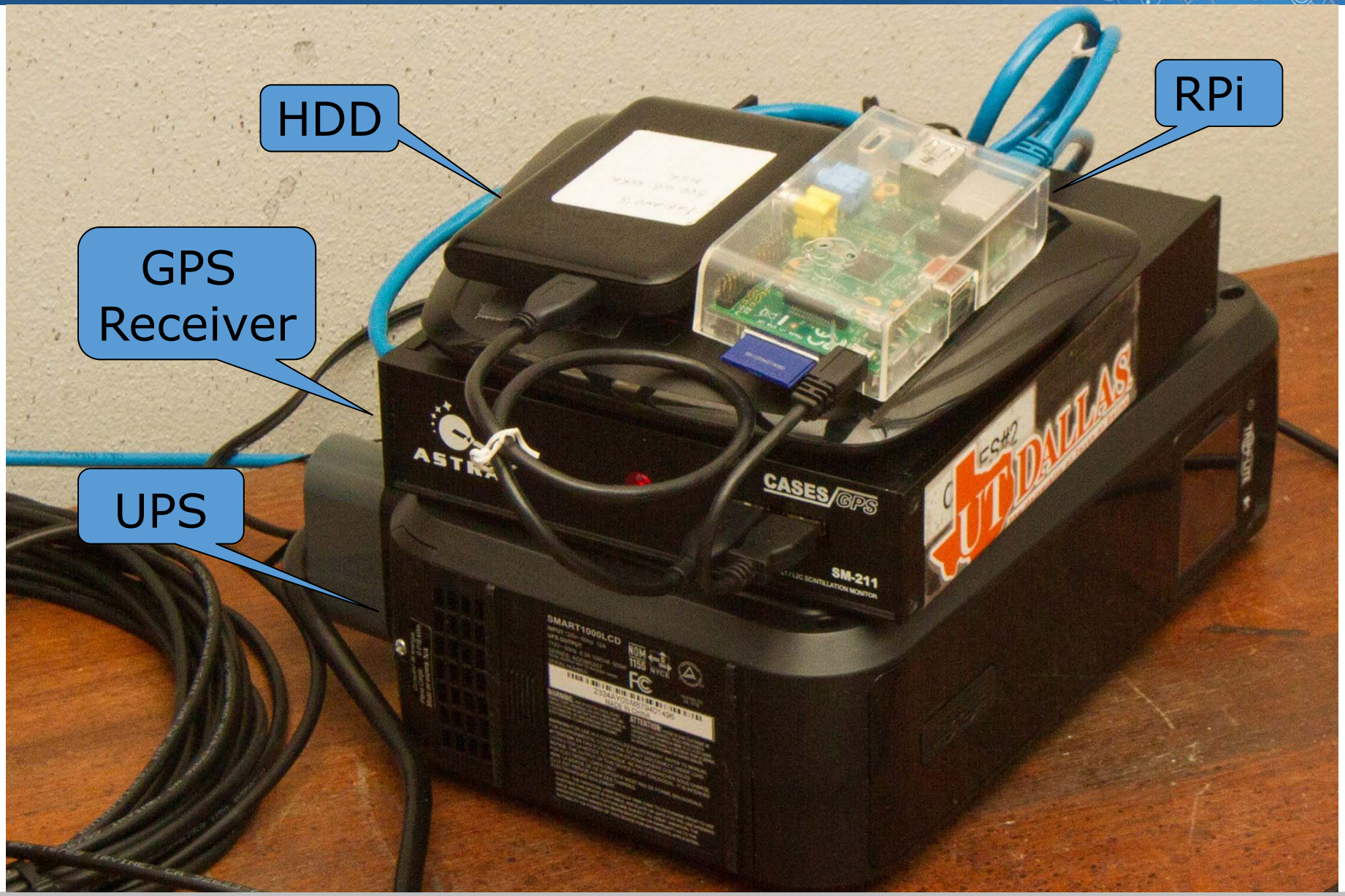
- Single task [Lab. 1 / Lab. 2]
  - Non-multitasking
  - AVR studio
- Real-Time OS (Embedded OS) [Lab. 3]
  - Preemptive multitasking
  - Micro-kernel approach
  - VxWorks, pSOS, QNX, NEOS
  - uC/OS-II, FreeRTOS
- Full-function OS [Lab. 4]
  - Preemptive multitasking
  - Embedded Linux
  - WinCE, Android, iOS

IoT Device ?


Raspberry Pi B+
Relay
3008 10bit ADC
TMP36 온도센서
USB 무선랜
AC21V 전동밸브


21°C
98%

IoT Device ?


HDD
RPi
GPS Receiver
UPS
ASTRA
CASES GPS
UT DALLAS
SM-211
SMART1000LCD

Embedded Software 개발 환경

- Single task
  - Non-multitasking
  - Arduino IDE, AVR Studio, Atmel Studio, …
- Real-time OS (Embedded OS)
  - Preemptive multitasking
  - Micro-kernel approach
  - VxWorks, pSOS, QNX, NEOS
  - uC/OS-II, FreeRTOS
- Full-function OS
  - Preemptive multitasking
  - Embedded Linux
  - WinCE, Android, iOS

Single Task

- Non-multitasking
- I/O control
  - Polling

```
void main (void)
{
  Initialization;
  while (1) {
    Read analog inputs;
    Read digital inputs;
    Monitoring function;
    Control function;
    Update analog outputs;
    Update digital outputs;
    Scan keyboards;
    Process user interfaces;
    Update displays;
    Process communications;
    etc...
  }
}
```

Single Task

- Non-multitasking
- I/O control
  - Interrupt

```
void main (void)
{
  Initialization;
  while (1) {
    Read analog inputs;
    Read digital inputs;
    Monitoring function;
    Control function;
    Update analog outputs;
    Update digital outputs;
    Scan keyboards;
    Process user interfaces;
    Update displays;
    Process communications;
    etc...
  }
}
```

```
ISR1 (void)
{
   Process
   asynchronous
   events;
}

ISR2 (void)
{
   Process
   asynchronous
   events;
}
```

Embedded Software 개발 환경

- Single task
  - Non-multitasking
  - AVR studio
- Real-Time OS (Embedded OS)
  - Preemptive multitasking
  - Micro-kernel approach
  - VxWorks, pSOS, QNX, NEOS
  - uC/OS-II, FreeRTOS
- Full-function OS
  - Preemptive multitasking
  - Embedded Linux
  - WinCE, Android, iOS

RTOS의 필요성

- **제품의 개발 주기 및 라이프 사이클이 빨라짐**
  - 개발 기간을 단축시키기 위해
  - 검증된 코드를 재사용하기 위해
  - 동시에 여러명이 제품의 개발을 진행하기 위해
- **제품 기능의 복잡도 증가**
  - Real-Time control 필요
  - Multi-Tasking의 기능이 필요한 제품 증가
- **Microcontroller의 성능은 좋아지고 가격은 하락**
  - RTOS의 성능 오버헤드가 문제되지 않음
- **파일 시스템과 네트워크를 필요로 하는 제품 증가**

출처: 조원경 교수님

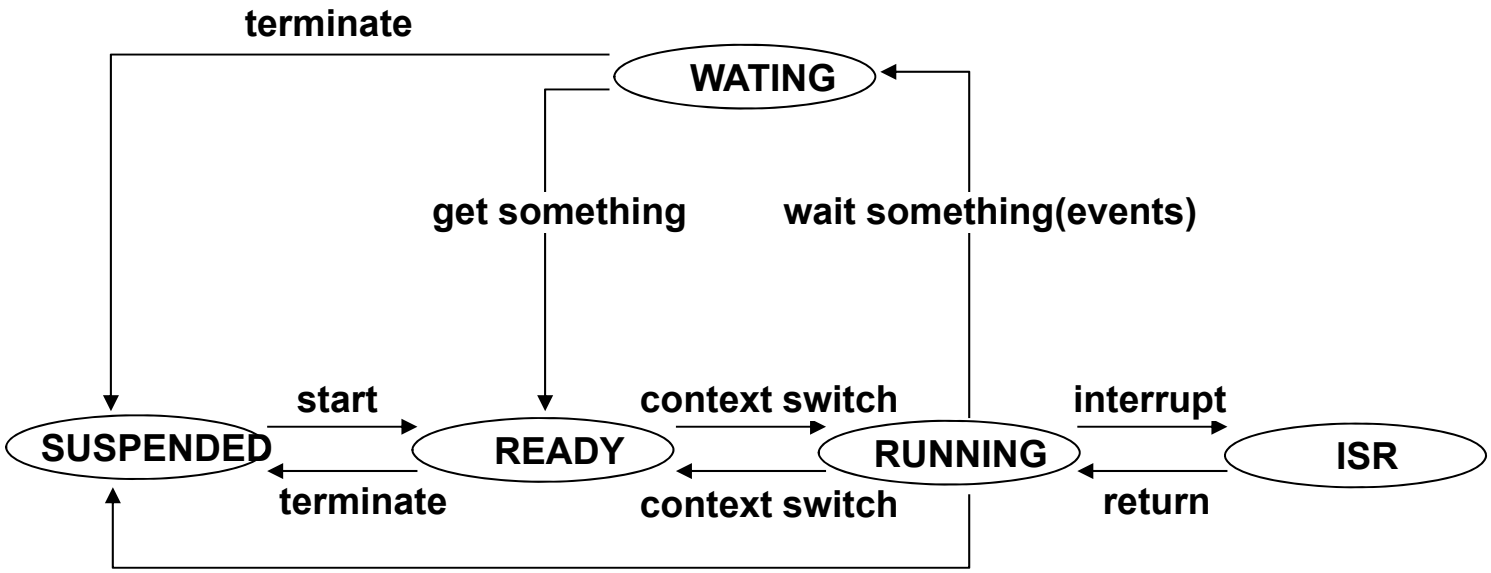
Multitasking

- Multi-Tasking (Multi-Processing)
  - An application is sub-divided into logical tasks in a system.
  - Each task processes its own function and can be implemented with ease.
  - OS handles multitasking with scheduling and context-switching.
  - Synchronization issues may occur and should be handled with semaphores and mutexes.
  - Ex) ADSL Router
    - PPP(point-to-point) task
    - IP(Internet Protocol) task
    - UDP(User Datagram Protocol) task
    - TCP(Transmission Control Protocol) task
    - RIP(Routing Information Protocol) task
    - ATM(Asynchronous Transfer Mode) task

Multitasking

- Task
  - Smallest execution unit
  - Similar to `Thread'
  - Has its priority, state, stack, context (CPU register sets)

- Task state diagram

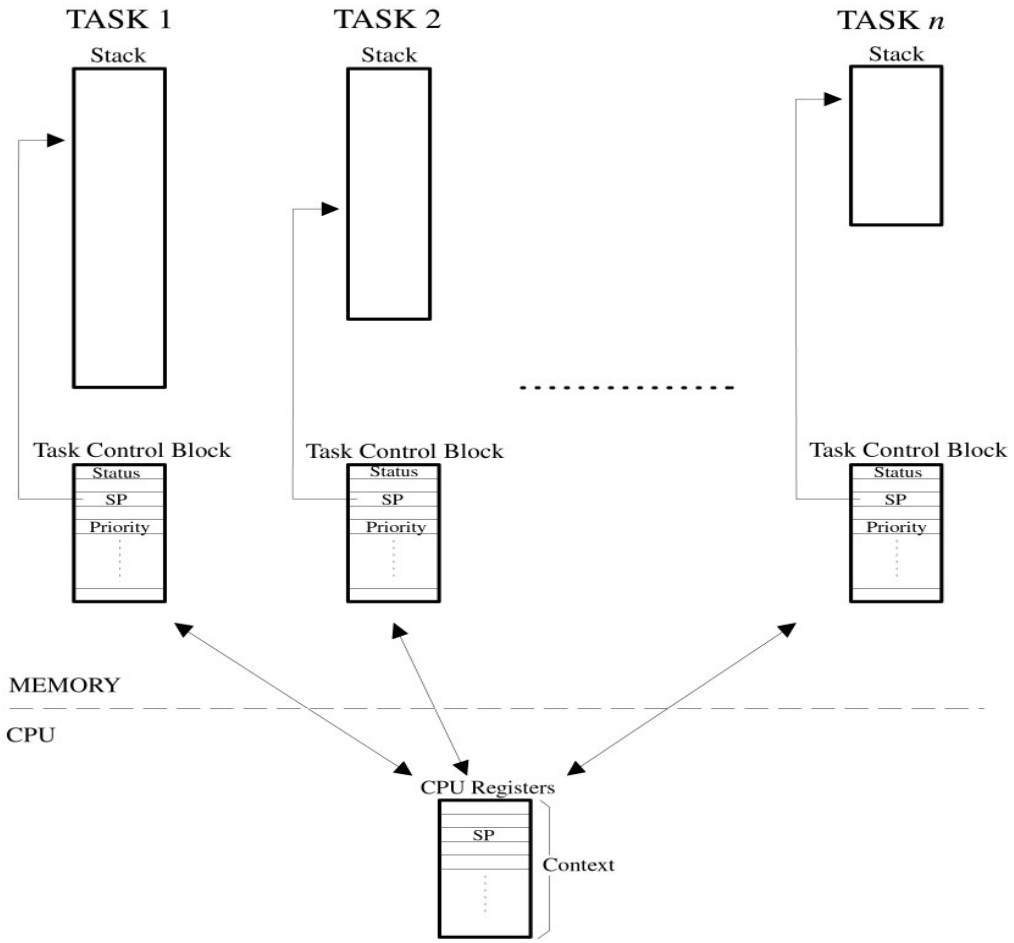


Multitasking

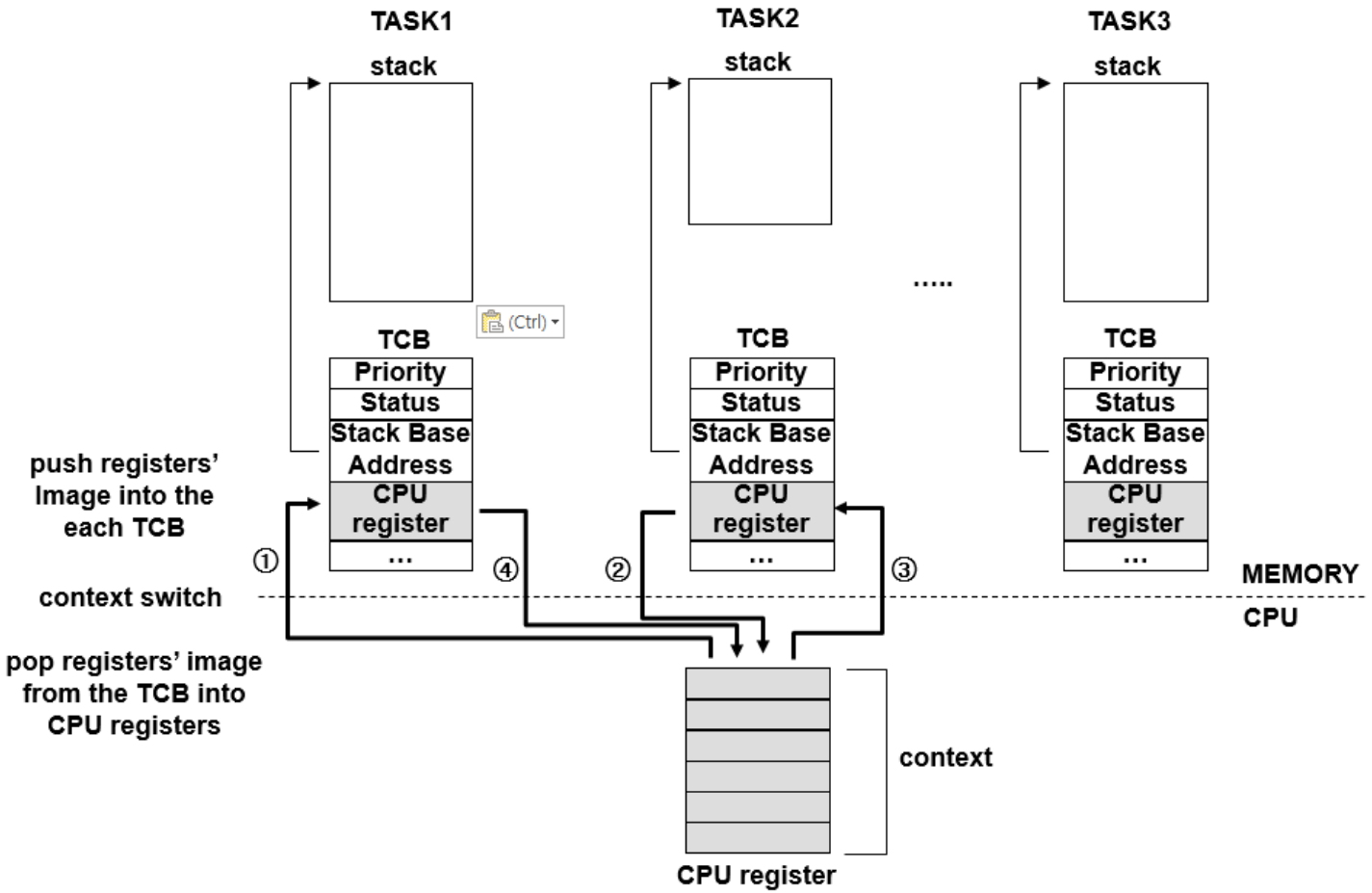
- Task priority
  - Task-specific parameter which indicates its importance
  - The more importance, the higher value
  - (0 is the highest or the lowest)
- Static priority
  - Not changed while running
  - Given in compile-time
  - Rate-Monotonic Scheduling (RMS)
- Dynamic priority
  - Can be changed and given in run-time
  - Earliest Deadline First (EDF)

Multitasking

- Multitasking



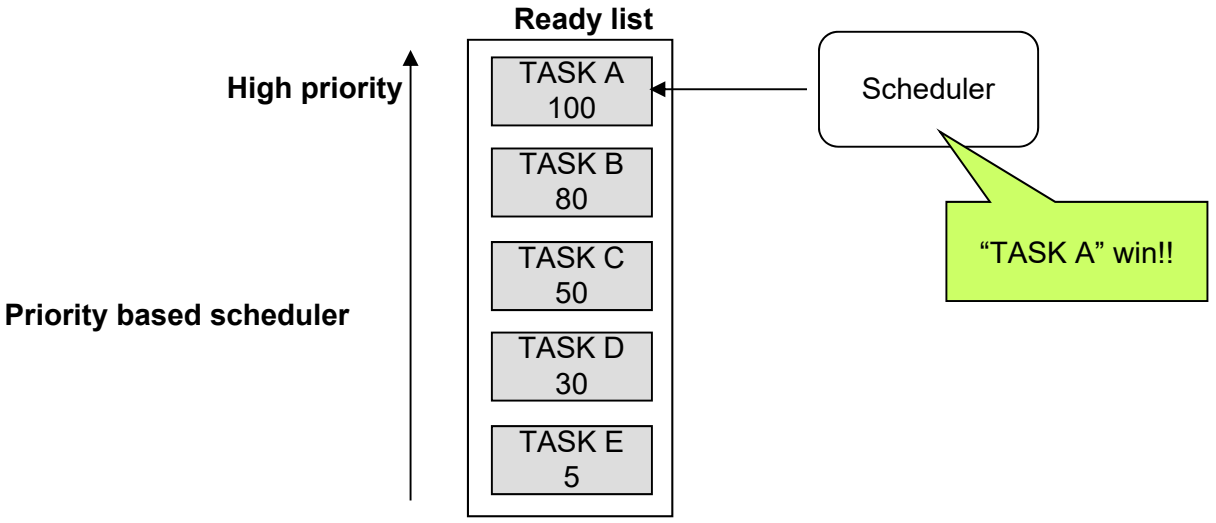
- Context switching



Multitasking

- Scheduler
  - Selects which task should be executed among tasks in ready list
  - In general, Priority & RR scheduling in embedded systems

- Dispatcher
  - Allocates CPU to the scheduled task

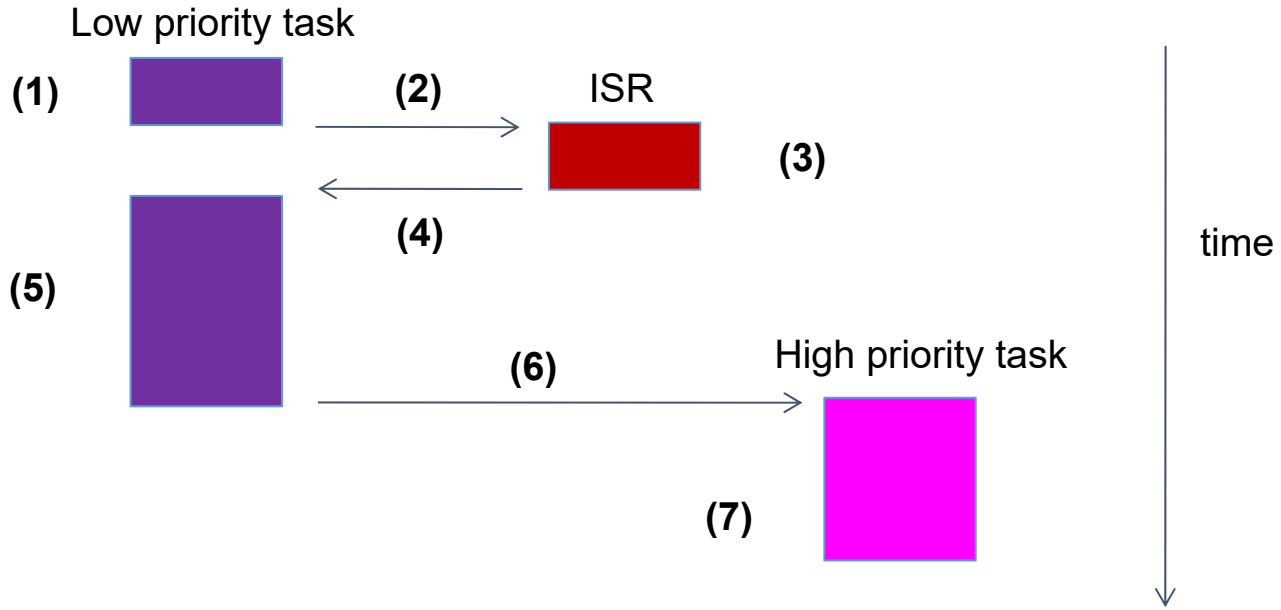


Cooperative Multitasking

- Multitasking (multiple tasks) but non-preemptive kernel
  - Context switch only on explicit system call, e.g. yield()
  - So, programmers should control the flow of tasks, i.e. context switching of tasks.
  - After an interrupt, CPU returns to the same task before the interrupt
  - Example of non-preemptive kernel
    - Original Windows 3.X, Macintosh
  - Pros
    - Low interrupt latency
    - Relatively small synchronization problem
  - Cons
    - Large latency of high-priority tasks
    - System fault due to a malfunctioning task (e.g. indefinite loop)

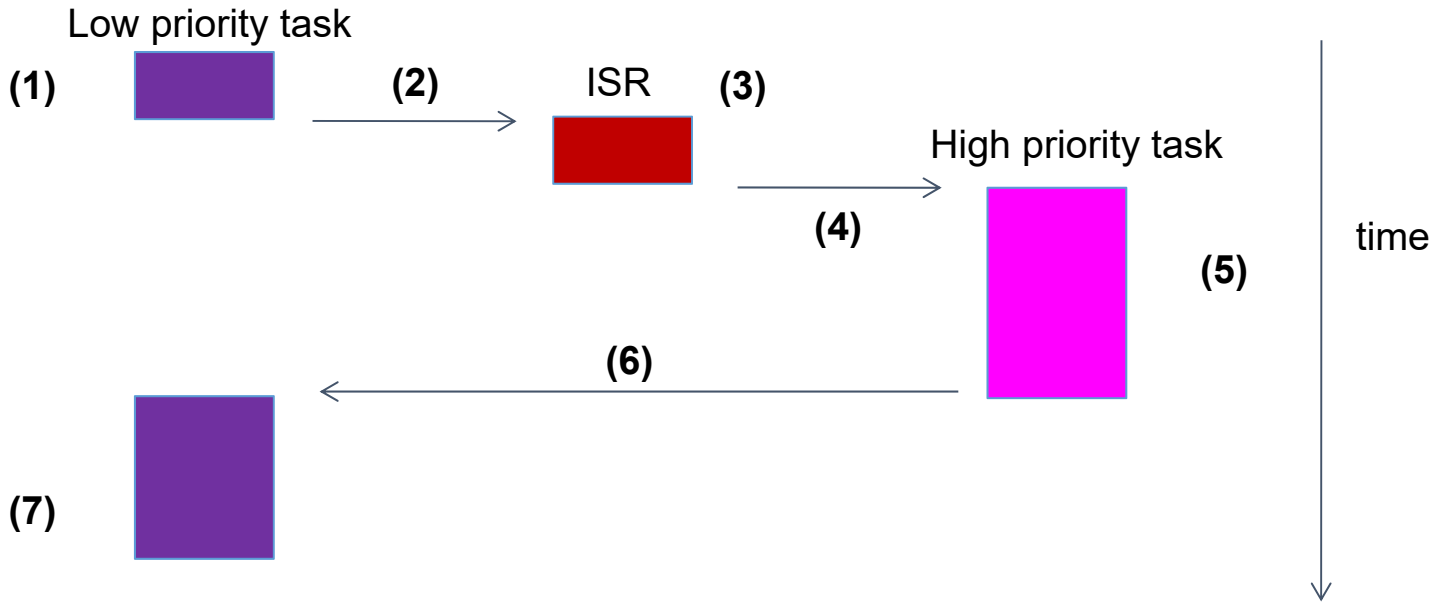
Cooperative Multitasking

- Non-preemptive kernel (non-preemptive scheduling)



Preemptive Multitasking

- Preemptive kernel
  - Fast response of higher priority tasks
  - The highest priority ready task takes control of CPU
  - On ISR return, OS schedules the highest priority ready task

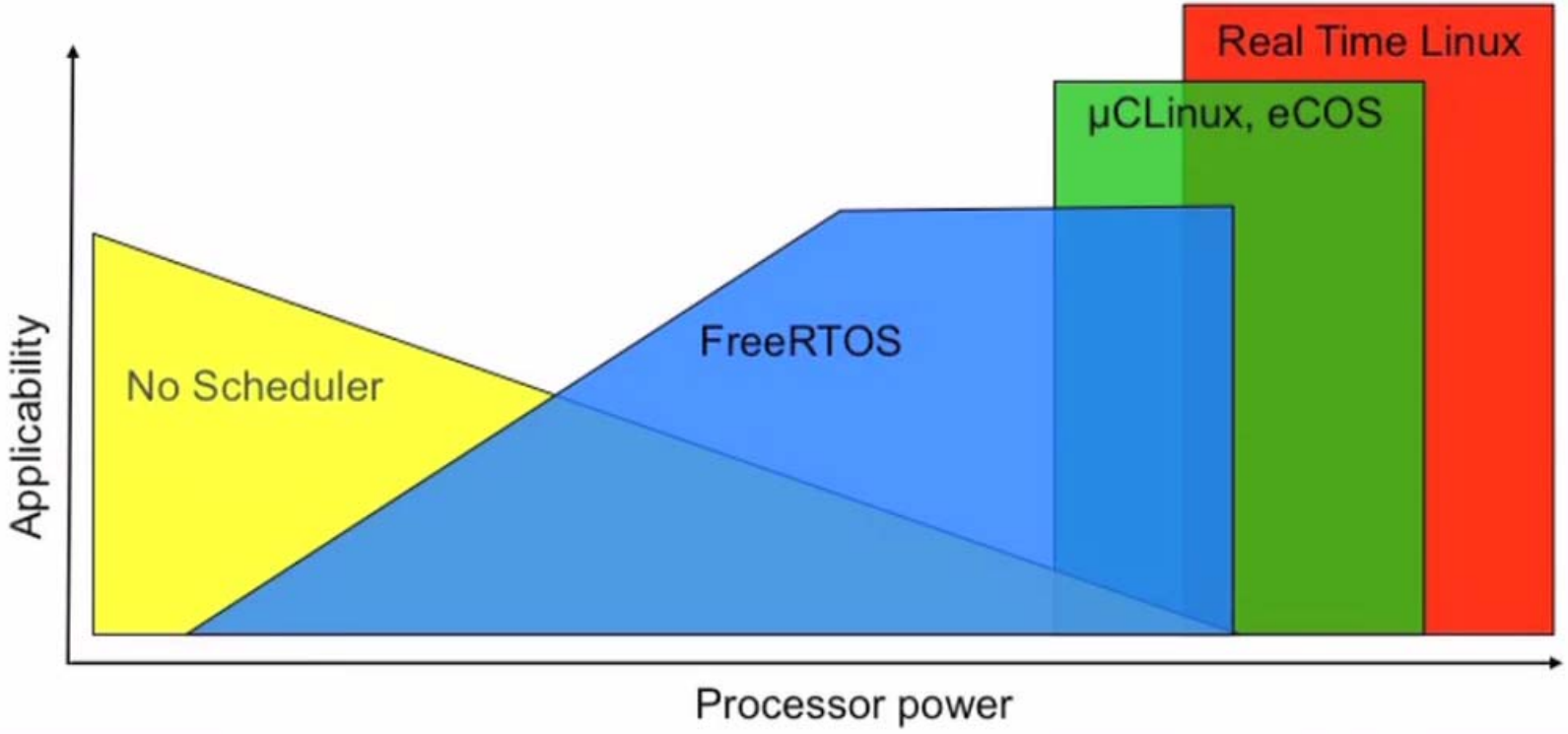


FreeRTOS

- A Real Time Operating System
- Written by Richard Barry & FreeRTOS Team
- Owned by Real Time Engineers Ltd. but free to use
- Huge number of users all over the world
- 6000 download per month
- Simple but very powerful

FreeRTOS

- When to use FreeRTOS



FreeRTOS Features

- High Quality Source code
- Portable
- Scalable
- Preemptive and co-operative scheduling
- Multitasking
- Services
- Interrupt management
- Advanced features

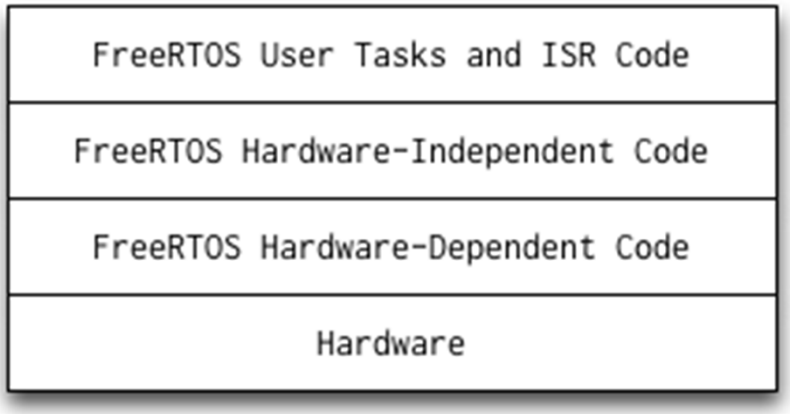
출처: Amr Ali Abdel-Naby@2010

Source Code

- High quality
- Neat
- Consistent
- Organized
- Commented

```
signed portBASE_TYPE xTaskRemoveFromEventList( const xList * const pxEventList )
{
tskTCB *pxUnblockedTCB;
portBASE_TYPE xReturn;

    /* THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED OR THE
    SCHEDULER SUSPENDED.  It can also be called from within an ISR. */

    /* The event list is sorted in priority order, so we can remove the
    first in the list, remove the TCB from the delayed list, and add
    it to the ready list.

    If an event is for a queue that is locked then this function will never
    get called - the lock count on the queue will get modified instead.  This
    means we can always expect exclusive access to the event list here.

    This function assumes that a check has already been made to ensure that
    pxEventList is not empty. */
    pxUnblockedTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxEventList );
    configASSERT( pxUnblockedTCB );
    vListRemove( &( pxUnblockedTCB->xEventListItem ) );

    if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
    {
        vListRemove( &( pxUnblockedTCB->xGenericListItem ) );
        prvAddTaskToReadyQueue( pxUnblockedTCB );
```

- Software layers




```
ccess the delayed or ready lists, so will hold this
ntil the scheduler is resumed. */
( xList * ) &( xPendingReadyList ), &( pxUnblockedTCB->xEventListItem ) );
```

Portable

- Highly portable C
- 24 architectures supported
- Assembly is kept minimum
- Ports are freely available in source code
- Other contributions do exist

Scalable

- Only use the services you only need
  - FreeRTOSConfig.h
- Minimum footprint = 4 KB

Scheduling

- Preemptive scheduling
  - Fully preemptive
  - Always runs the highest priority task that is ready to run
  - Comparable with other preemptive kernels
  - Used in conjunction with tasks
- Cooperative scheduling
  - Context switch occurs if:
    - A task/co-routine blocks
    - Or a task/co-routine yields the CPU
  - Used in conjunction with tasks/co-routines

Multitasking

- No software restriction on
  - # of tasks that can be created
  - # of priorities that can be used
- Priority assignment
  - More than one task can be assigned the same priority
  - RR with time slice = 1 RTOS tick

Services

- Queues
- Semaphores
  - Binary and counting
- Mutexes
  - With priority inheritance
  - Support recursion

Advanced Features

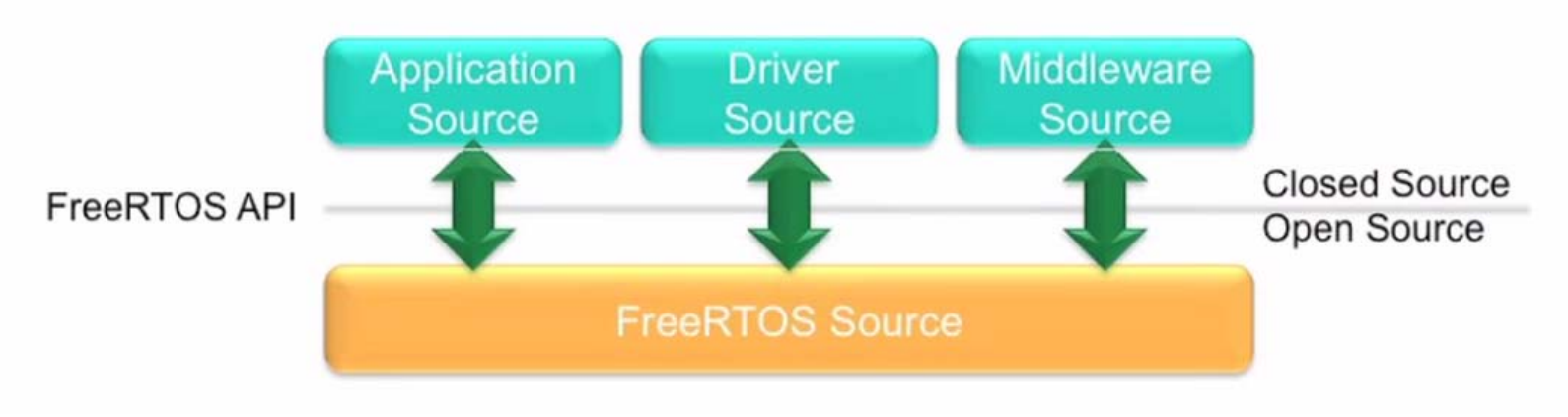
- Execution tracing
- Run time statistics collection
- Memory management
- Memory protection support
- Stack overflow protection

Device Support

- Connect Suite from High Integrity Systems
  - TCP/IP stack
  - USB stack
    - Host and device
  - File systems
    - DOS compatible FAT

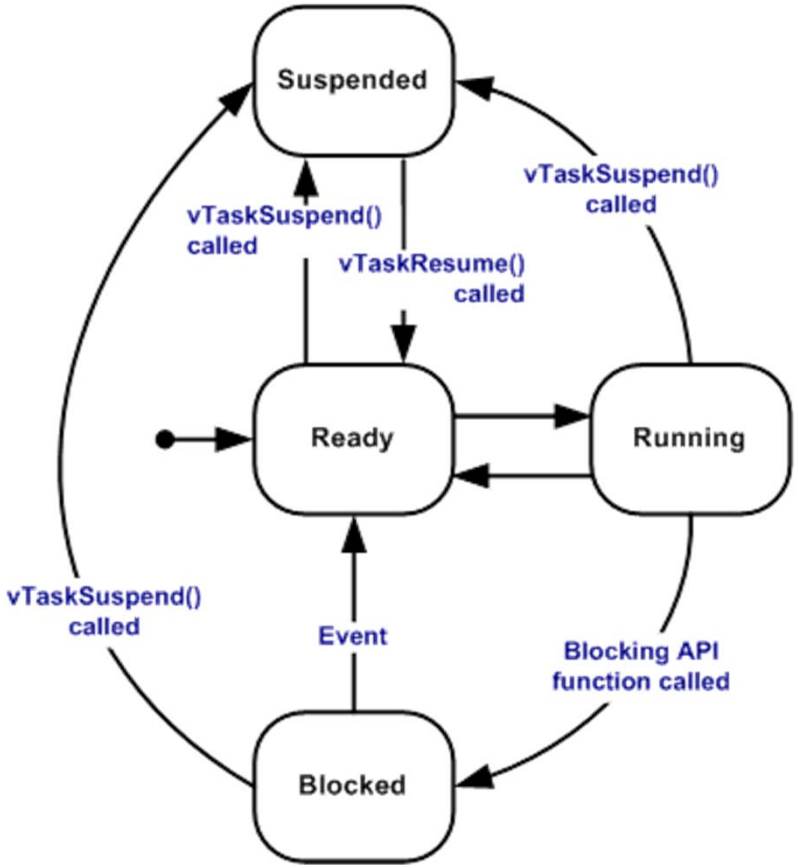
Licensing

- Modified GPL
  - Only FreeRTOS is GPL
  - Independent modules that communicate with FreeRTOS through APIs can be anything else
  - FreeRTOS can’t be used in any comparisons without the authors’ permission



Task in FreeRTOS

- Task States
  - Running
    - Task is actually executing
  - Ready
    - Task is ready to execute but a task of equal or higher priority is Running.
  - Blocked
    - Task is waiting for some event.
      - Time: if a task calls vTaskDelay() it will block until the delay period has expired.
      - Resource: Tasks can also block waiting for queue and semaphore events.
  - Suspended
    - Much like blocked, but not waiting for anything.
    - Tasks will only enter or exit the suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.



Task in FreeRTOS

- Task Priority
  - Each task is assigned a priority from 0 to (configMAX_PRIORITIES - 1)
  - Low priority numbers denote low priority tasks
  - The idle task has priority zero (tskIDLE_PRIORITY)
  - The task placed into the Running state is always be the highest priority task that is able to run
  - Ready state tasks of equal priority share the available processing time using a time sliced round robin (RR) scheduling scheme

Task in FreeRTOS

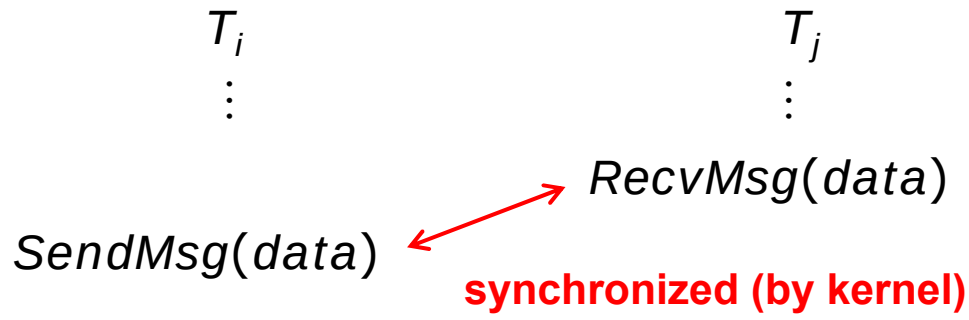
- Idle Task
  - Created automatically when the RTOS scheduler is started to ensure there is always at least one task that is able to run
  - Responsible for freeing memory allocated by the RTOS to tasks that have since been deleted
  - An idle task hook is a function that is called during each cycle of the idle task

Task in FreeRTOS

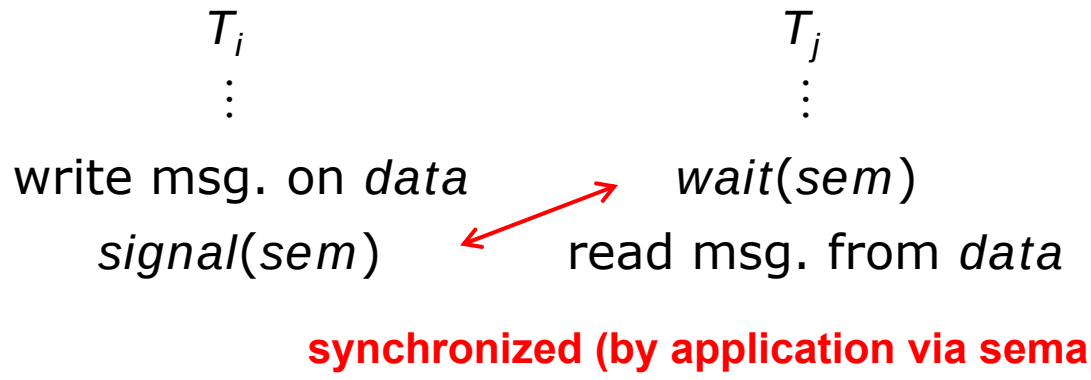
- API
  - BaseType_t xTaskCreate(TaskFunction_t pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    UBaseType_t uxPriority,
    TaskHandle_t *pxCreatedTask);
  - void vTaskDelay(const TickType_t xTicksToDelay);
  - void vTaskDelayUntil(portTickType *pxPreviousWakeTime,
    portTickType xTimeIncrement);
  - void vTaskStartScheduler(void);

Inter-Task Communication

- Two types of ITC (Inter-Task Communication)
  - Message passing



  - Shared memory



ITC in FreeRTOS

- API
  - QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength,
    UBaseType_t uxItemSize);
  - BaseType_t xQueueSend(QueueHandle_t xQueue,
    const void * pvItemToQueue,
    TickType_t xTicksToWait);
  - BaseType_t xQueueReceive(QueueHandle_t xQueue,
    void *pvBuffer,
    TickType_t xTicksToWait);

Synchronization

- Tasks cooperate in multitasking systems
  - To coordinate their execution
  - To share resources, and to access shared data structures
- Definitions
  - Resource
    - Entities which process/thread/task may utilize
    - Physical entities such as CPU, memory, I/O devices (display, keyboard, printer, etc.)
    - Logical entities such as variables, arrays, data structures (queue, stack, tree, etc.)
  - Shared Resource
    - Resource which one or more process/thread/task share
    - Process/thread/task needs exclusive access on shared resources to avoid data corruption through synchronization mechanisms.

Critical Section Example

- Withdraw money from a bank account
  - Suppose you and your girl(boy) friend share a bank account with a balance of 1,000,000won
  - What happens if both go to separate ATM machines, and simultaneously withdraw 100,000won from the account?

```
int withdraw (account, amount)
{
   balance = get_balance (account);
   balance = balance - amount;
   put_balance (account, balance);
   return balance;
}
```

Synchronization

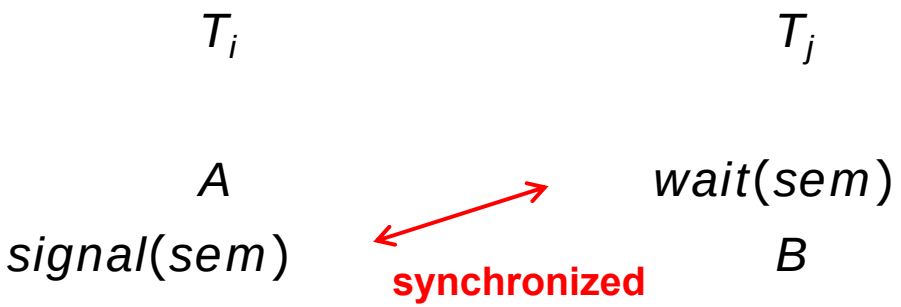
- Definitions
  - Race condition
    - A situation where several tasks access and manipulate shared resources concurrently
    - The result is non-deterministic and depends on timing
  - Critical section
    - Code segment in which shared resources are accessed
    - Mutually exclusive access should be guaranteed in critical sections (mutual exclusion, mutex)
  - Critical section problem or Synchronization problem
    - Shared resources may be corrupted in multitasking systems

Synchronization Mechanisms

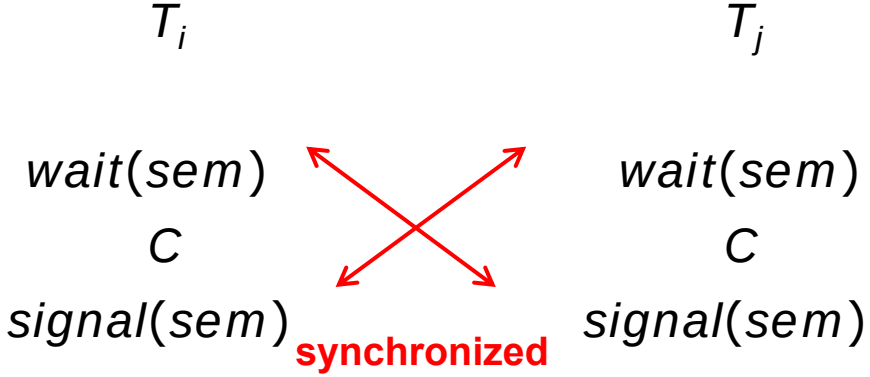
- Primitive mechanisms for OS
  - Software algorithms
  - Disabling interrupts
  - Spinlocks
    - Hardware atomic instruction
    - Busy waiting
- High-level mechanisms for application task/process/thread
  - Semaphores
    - Basic, easy to get the hang of, hard to program with
    - Binary semaphore = mutex (≅ lock)
    - Counting semaphore
  - Monitors
    - High-level, requires language support, implicit operations
    - Easy to program with: Java "synchronized"

Synchronization Examples

- To coordinate executions
  - Execute *B* in T_j after *A* executed in T_i



- To share resources
  - Protect critical section *C* between T_i and T_j

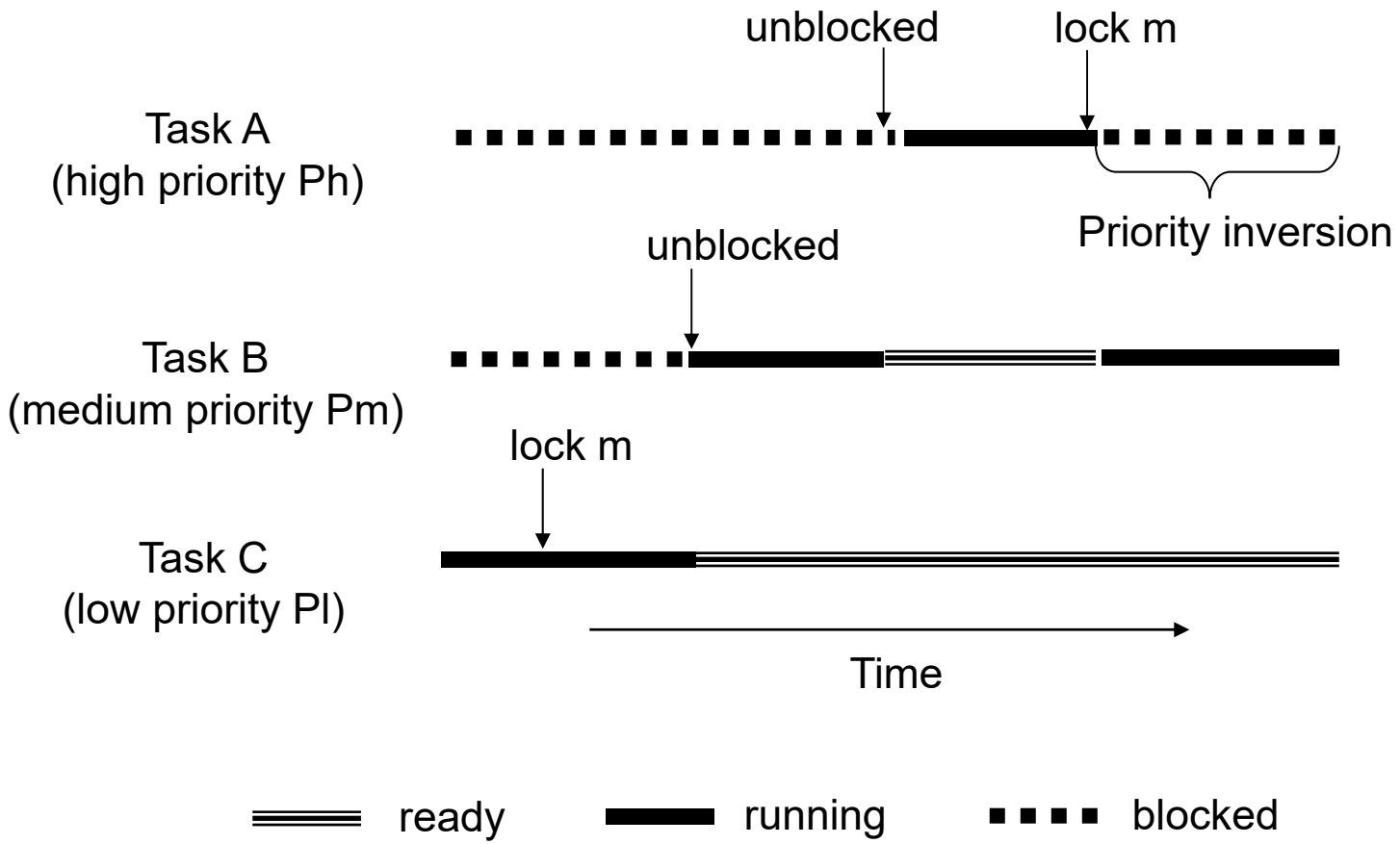


Synchronization in FreeRTOS

- API
  - SemaphoreHandle_t xSemaphoreCreateBinary(void);
  - SemaphoreHandle_t xSemaphoreCreateCounting(
    UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);
  - SemaphoreHandle_t xSemaphoreCreateMutex(void);
    - Support PIP(Priority Inheritance Protocol)
  - int xSemaphoreTake(SemaphoreHandle_t xSemaphore,
    TickType_t xTicksToWait);
  - int xSemaphoreGive(SemaphoreHandle_t xSemaphore);

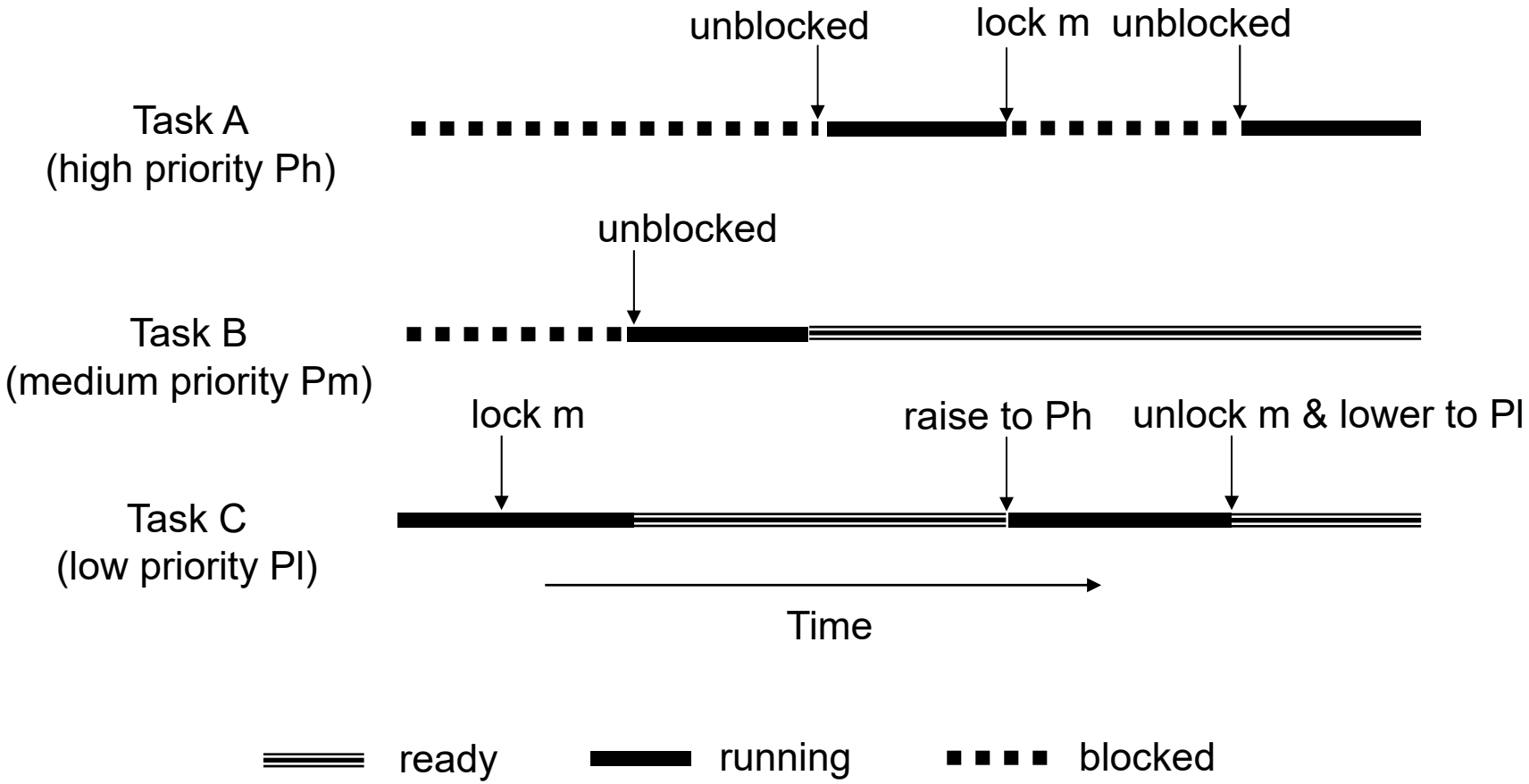
In Real-Time Systems

- Priority inversion problem



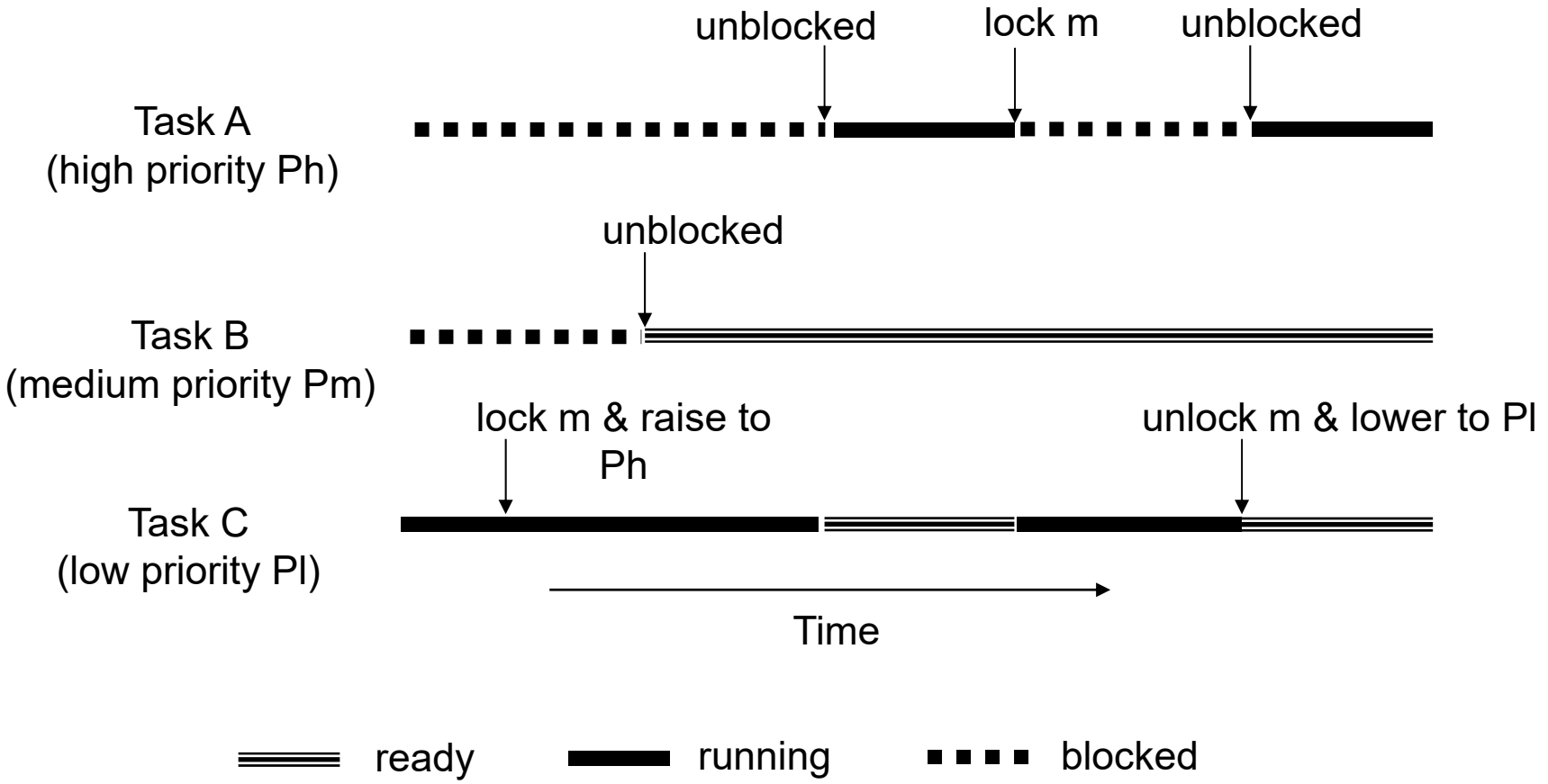
In Real-Time Systems

- Priority Inheritance Protocol



In Real-Time Systems

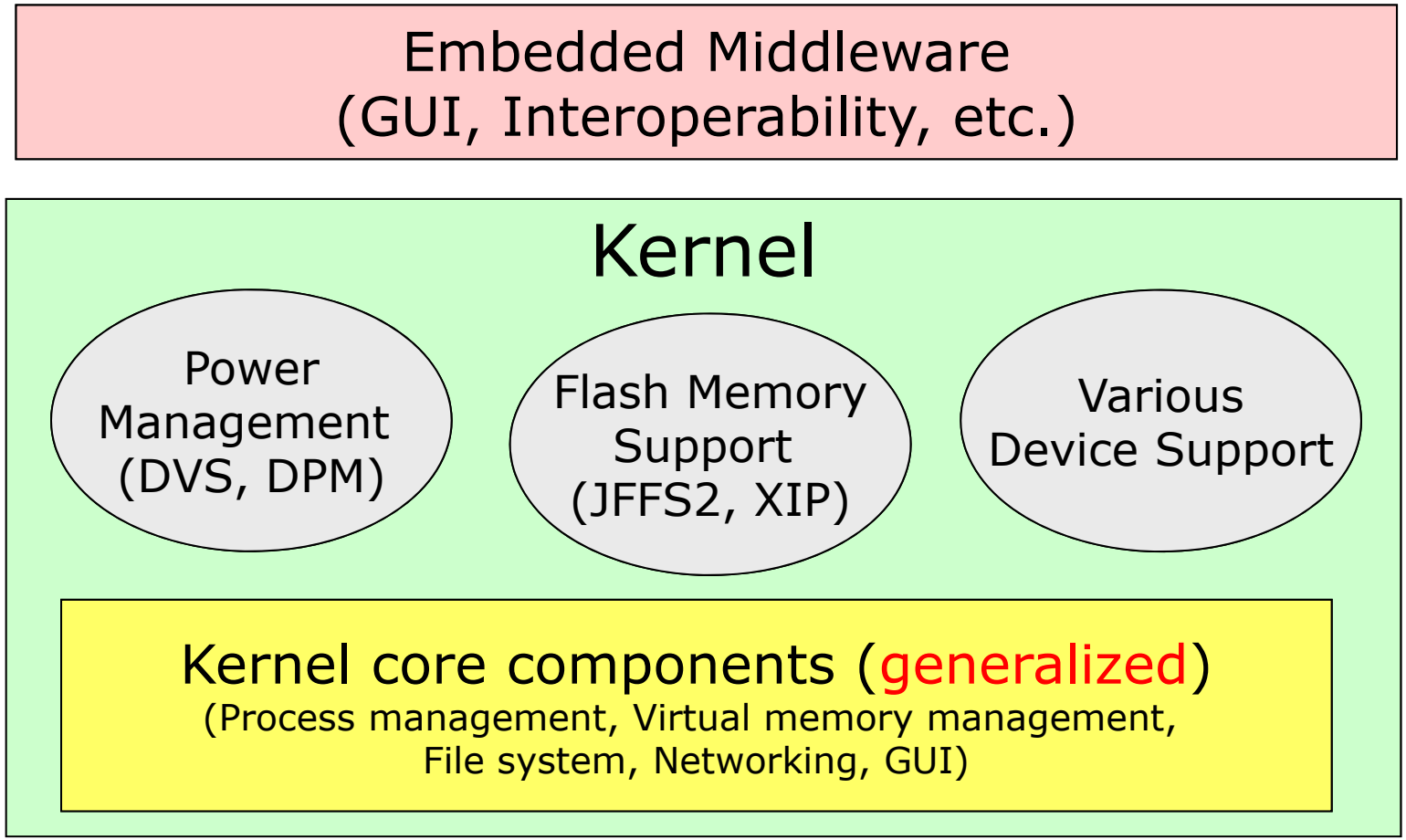
- Priority Ceiling Protocol



Embedded Software 개발 환경

- Single task
  - Non-multitasking
  - AVR studio
- Real-Time OS (Embedded OS)
  - Preemptive multitasking
  - Micro-kernel approach
  - VxWorks, pSOS, QNX, NEOS
  - uC/OS-II, FreeRTOS
- Full-function OS
  - Preemptive multitasking
  - Embedded Linux
  - WinCE, Android, iOS

Appearance of Full-function OS



Comparison

- Kernel

	Library	Real-Time OS	Full-function OS
Kernel Structure	N/A	Micro-kernel	Monolithic or Micro-kernel
Kernel Size	N/A	Small	Large
Kernel Overhead	N/A	Low	High
Real-Time Features	N/A	Yes	No, but …
Product	N/A	VxWorks, pSOS, QNX, uC/OS-II, uClinux, Symbian, bada, etc.	Embedded Linux, Windows, Android, iOS, bada, Tizen, etc.
System Call	N/A	Proprietary	POSIX, Win32, S/W Platform-dependent

Comparison

- H/W & Application

	Library	Real-Time OS	Full-function OS
Application S/W	Simple I/O Control	Medium Real-Time System	Complicated / Large GUI
CPU Requirement	4 / 8 bits	8 /16 /32 bits	32 / 64 bits (MMU & Dual-mode)
Memory Requirement	Small Lab.1-2: 4KB	Small (KBs) Lab.3-2: 20KB	Large (MBs)
I/O Devices	Application dependent	Application dependent	Many in general

Comparison

- Task (Process) Management

	Library	Real-Time OS	Full-function OS
Task Model	Single	Multi-task	Multi-process & Multi-thread
Task Scheduling	N/A	Static Priority (& RR)	(Static or Dynamic) Priority & RR
IPC	N/A	Yes	Yes
Synchronization	N/A	Yes	Yes

Comparison

- Memory Management

	Library	Real-Time OS	Full-function OS
Address Space	Single	Single	Kernel vs. Application Address Space
Virtual Memory Management	No	No	Yes
Memory Protection	No	No	Yes

Comparison

- I/O Management

	Library	Real-Time OS	Full-function OS
I/O Control	Application	Application	OS
I/O Protection	No	No	Yes
File System	No	Optional	Support
GUI	No	Optional	Support
Networking	No	Optional	Support

Embedded Linux

- What is Embedded Linux?
  - Embedded Linux is the usage of the Linux kernel and various open-source components in embedded systems
- Why Embedded Linux?
  - Completely free operating system: Linux/GNU
  - Advantages of Linux and open-source for embedded systems
    - Re-using components
    - Low cost
    - Full control
    - Quality
    - Easy testing of new features
    - Community support
    - Taking part into the community

Birth of Free Software

- 1983
  - Richard Stallman, GNU project and free software concept
  - gcc, gdb, glibc, and other tools
- 1991
  - Linus Tovalds, Linux kernel project
  - Completely free operating system: Linux/GNU
- 1995
  - Linux is more and more popular on server systems
- 2000
  - Linux is more and more popular on embedded systems
- 2008
  - Linux is more and more popular on mobile devices
- 2010
  - Linux is more and more popular on phones

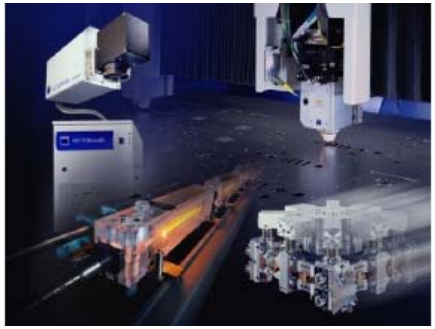
Examples Running Embedded Linux

- Television

- Personal router

- PoS (Point of Sales) terminal

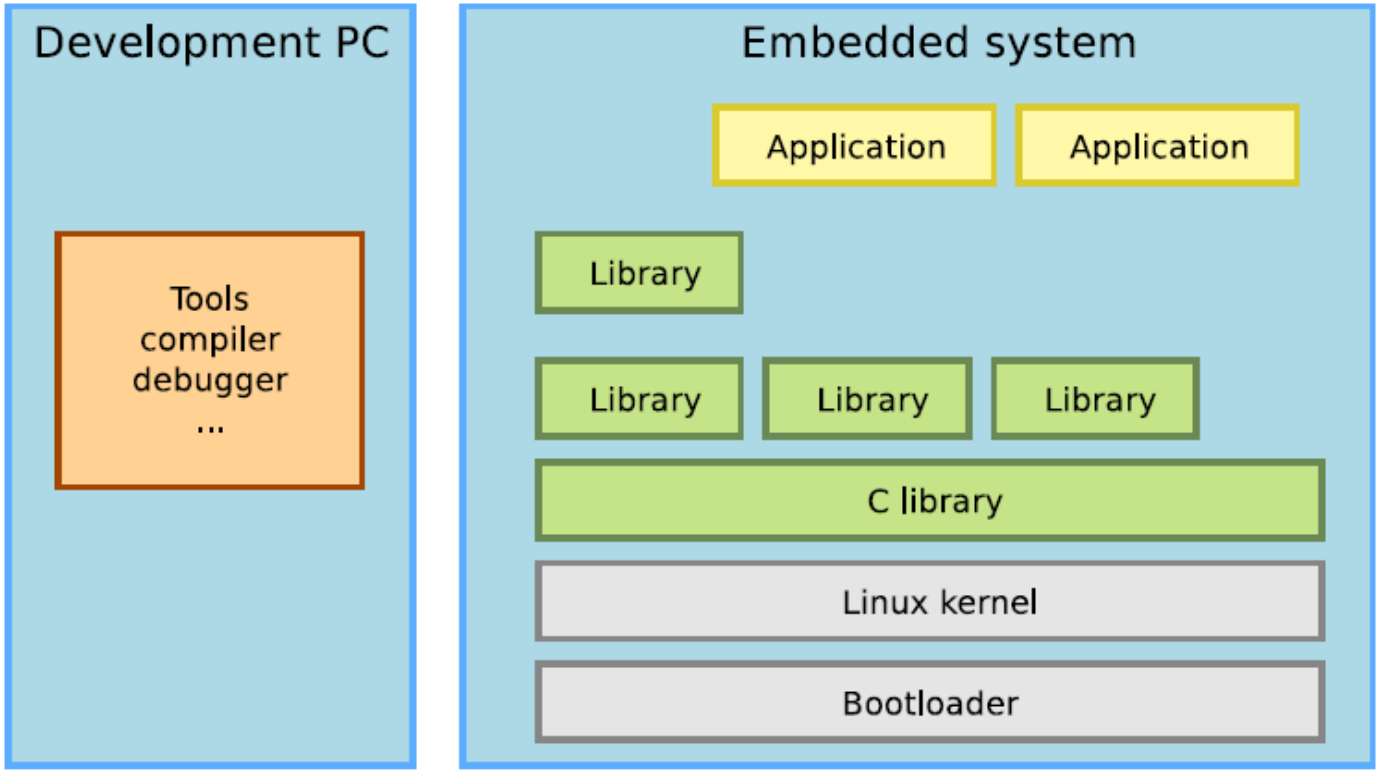
- Laser cutting machine



- Viticulture machine

Embedded Linux System Architecture

- Overall architecture



출처: Free Electrons

Embedded Linux System Architecture

- Software components
  - Cross-compilation toolchain
    - Compiler that runs on the development machine, but generates code for the target
  - Bootloader
    - Started by the hardware, responsible for basic initialization, loading, and executing the kernel
  - Linux Kernel
    - Contains the process and memory management, network stack, device drivers and provides services to user space applications
  - C library
    - The interface between the kernel and the user space applications
  - Libraries and applications
    - Third-party or in-house

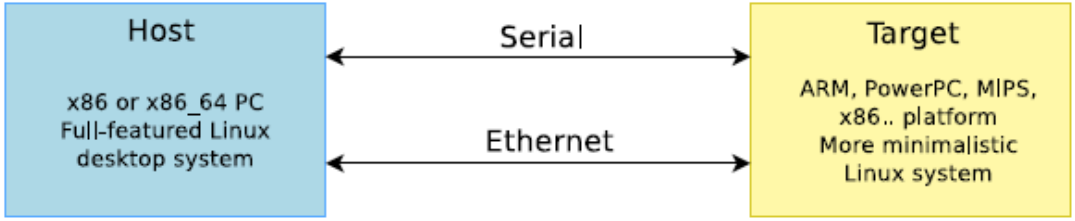
Embedded Linux System Architecture

- Embedded Linux work
  - Board Support Package development
    - A BSP contains a bootloader and kernel with the suitable device drivers for the targeted hardware
  - System integration
    - Integrate all the components, bootloader, kernel, third-party libraries and applications and in-house applications into a working system
  - Development of applications
    - Normal Linux applications, but using specifically chosen libraries

Embedded Linux Development Environment

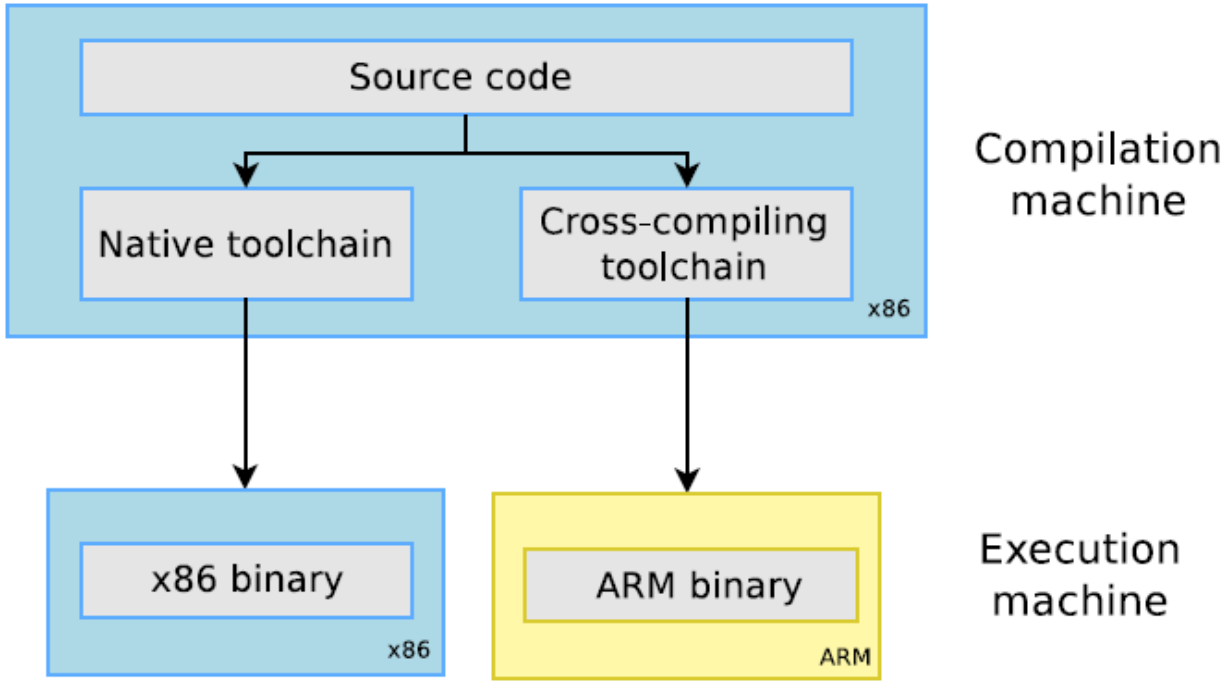
- Host OS
  - Linux is recommended (Ubuntu)

- Connection between host and target
  - Serial line
    - Minicom, Picocom, Gtkterm, Putty, etc.
  - Ethernet
    - Tftp, NFS, etc.

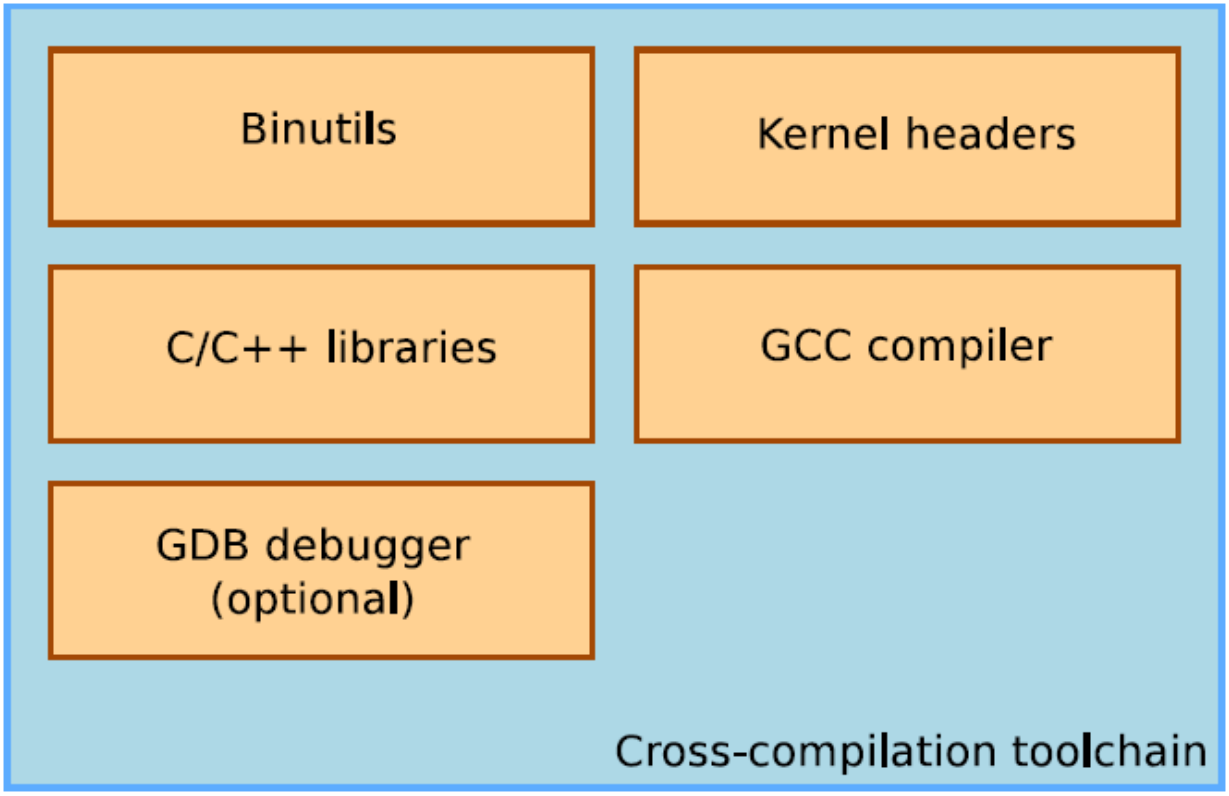


  - JTAG for bootloader fusing or low-level debugging (sometimes)

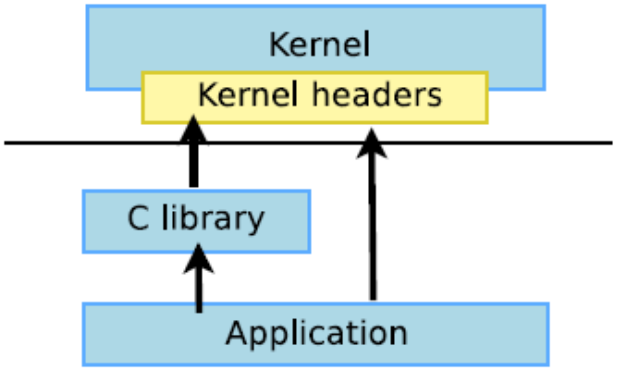
- Cross-compiling toolchains



- Cross-compiling toolchains
  - Components



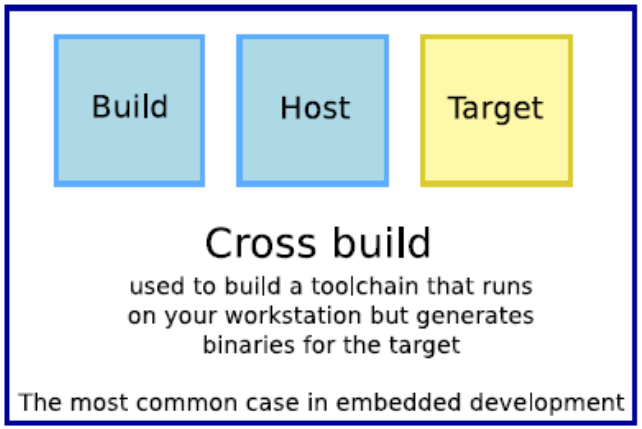
- Cross-compiling toolchains
  - Components
    - **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture (e.g., as, ld, ar, ranlib, strip, etc.)
    - **GCC** is a GNU C Compiler, the famous free software compiler
    - **C library** is an essential component of a Linux system and interface between the applications and the kernel (e.g. glibc, uClibc, etc.)
    - Compiling the C library requires **kernel headers**, and many applications also require them



- Cross-compiling toolchains
  - Different toolchain build procedures
    - Building a toolchain manually (too difficult)
    - Get a pre-compiled toolchain

Native build

used to build the normal gcc of a workstation



Cross-native build

used to build a toolchain that runs on your target and generates binaries for the target

Canadian build

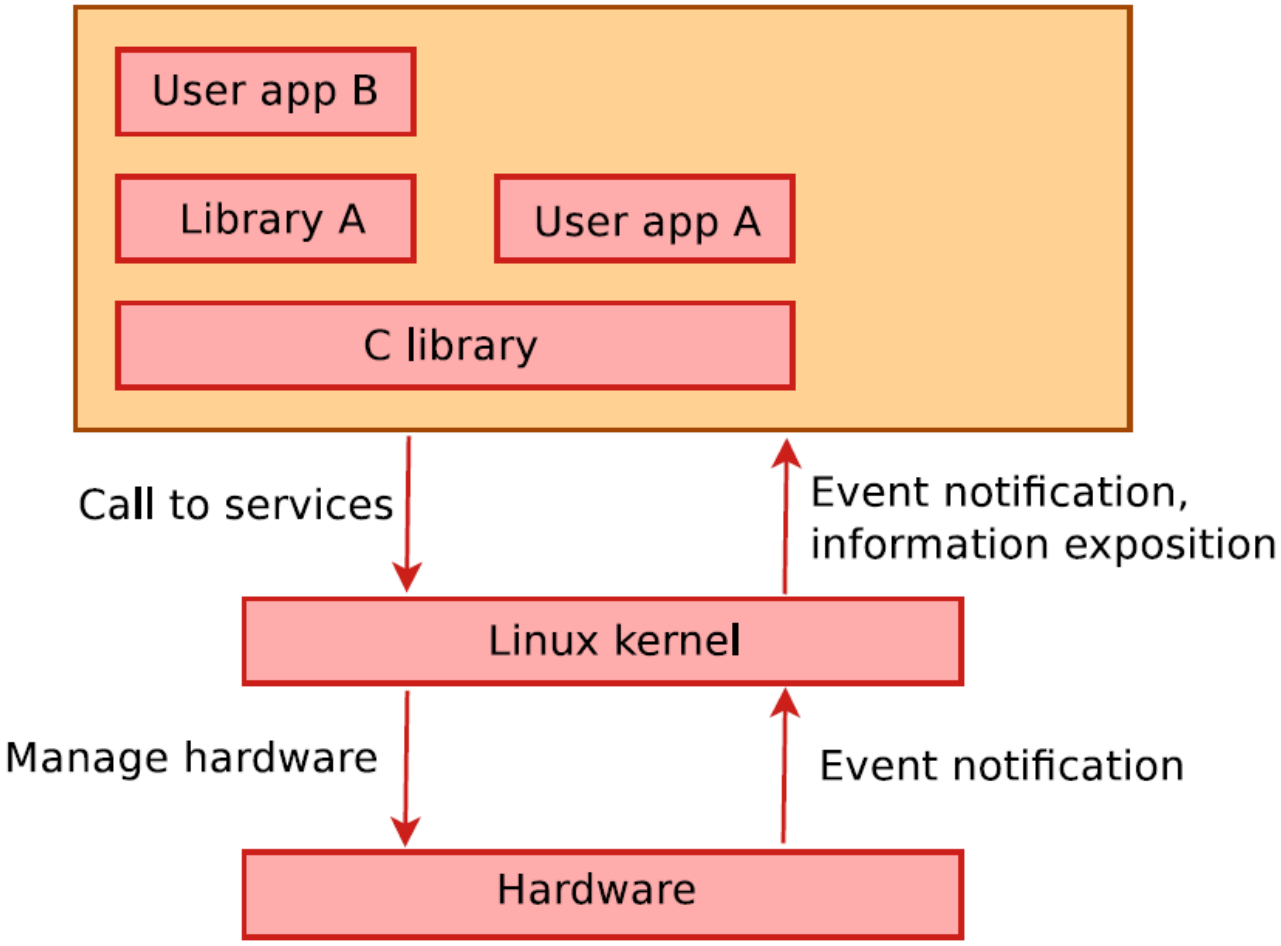
used to build on architecture A a toolchain that runs on architecture B and generates binaries for architecture C

Bootloader

- The bootloader is a piece of code responsible for
  - Basic hardware initialization
  - Loading of an application binary, usually an operating system kernel, from flash storage, from the network, or from another type of non-volatile storage
  - Possibly decompression of the application binary
  - Execution of the application binary
- Besides these basic functions, most bootloaders provide a shell with various commands implementing different operations
  - Loading of data from storage or network, memory inspection, hardware diagnostics and testing, etc.
- Generic bootloaders for embedded CPUs
  - U-Boot
  - Barebox, RedBoot, Yaboot, PMON, etc.

Embedded Linux Kernel

- Linux kernel in the system



Embedded Linux Kernel

- Location of kernel sources
  - The official version of the Linux kernel, as released by Linus Torvalds is available at http://www.kernel.org
    - This version follows the well-defined development model of the kernel
  - Many kernel sub-communities maintain their own kernel, with usually newer but less stable features
    - Architecture communities (ARM, MIPS, PowerPC, etc.)
    - Device drivers communities (I2C, SPI, USB, PCI, network, etc.)
    - Other communities (real-time, etc.)

Embedded Linux Kernel

- Linux kernel size
  - Linux 3.1 sources:
    - Raw size: 434 MB (39,400 files, approx. 14,800,000 lines)
    - gzip compressed tar archive: 93 MB
    - bzip2 compressed tar archive: 74 MB (better)
    - xz compressed tar archive: 62 MB (best)
  - Minimum Linux 2.6.29 compiled kernel size with CONFIG_EMBEDDED, for a kernel that boots a QEMU PC (IDE hard drive, ext2 filesystem, ELF executable support):
    - 532 KB (compressed), 1325 KB (raw)
  - Why are these sources so big?
    - Because they include thousands of device drivers, many network protocols, support many architectures and file systems...
    - The Linux core (scheduler, memory management...) is pretty small!

- Linux kernel size (Cont'd)
  - As of kernel version 3.2

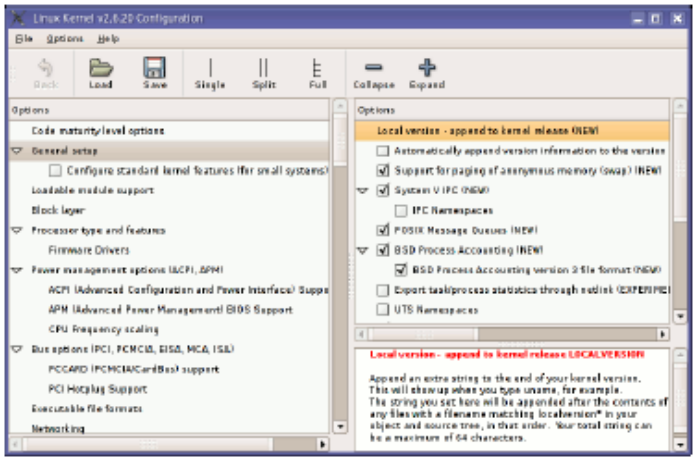
drivers/	: 53.65%	scripts/	: 0.44%
arch/	: 20.78%	security/	: 0.40%
fs/	: 6.88%	crypto/	: 0.38%
sound/	: 5.04%	lib/	: 0.30%
net/	: 4.33%	block/	: 0.13%
include/	: 3.80%	ipc/	: 0.04%
firmware/	: 1.46%	virt/	: 0.03%
kernel/	: 1.10%	init/	: 0.03%
tools/	: 0.56%	samples/	: 0.02%
mm/	: 0.53%	usr/	: 0%

Embedded Linux Kernel

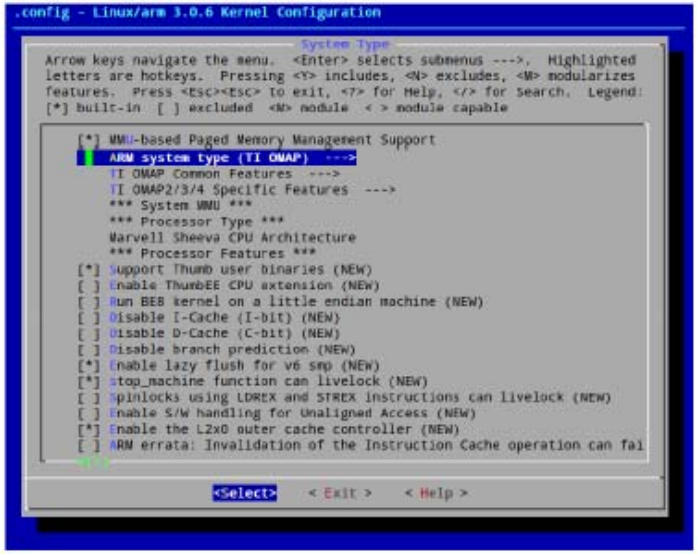
- Kernel vs. module
  - The **kernel image** is a single file, resulting from the linking of all object files that correspond to features enabled in the configuration
    - This is the file that gets loaded in memory by the bootloader
    - All included features are therefore available as soon as the kernel starts, at a time where no file system exists
  - Some features (device drivers, file systems, etc.) can however be compiled as **modules**
    - Those are plugins that can be loaded/unloaded dynamically to add/remove features to the kernel
    - Each module is stored as a separate file in the file system, and therefore access to a file system is mandatory to use modules
    - This is not possible in the early boot procedure of the kernel, because no file system is available

Embedded Linux Kernel

- Kernel configuration
  - GUI interface
    - make xconfig
    - make gconfig (→)



  - Text interface
    - make menuconfig (→)
    - make nconfig
    - make oldconfig
    - make allnoconfig



Embedded Linux Kernel

- Kernel compilation
  - "make" in the main kernel source directory
  - Generates
    - vmlinux
      - Raw uncompressed kernel image
      - Cannot be booted
    - bzImage for x86, zImage for ARM, vmImage.gz for Blackfin, etc.
      - arch/<arch>/*Image
      - Can be booted
    - all kernel modules
      - Spread over the kernel source tree, as ".ko" files
  - "make zImage" & "make modules"
  - "make clean"

Embedded Linux Kernel

- Kernel installation
  - “make install”
  - “make modules_install”
  - for host system only

Root File System

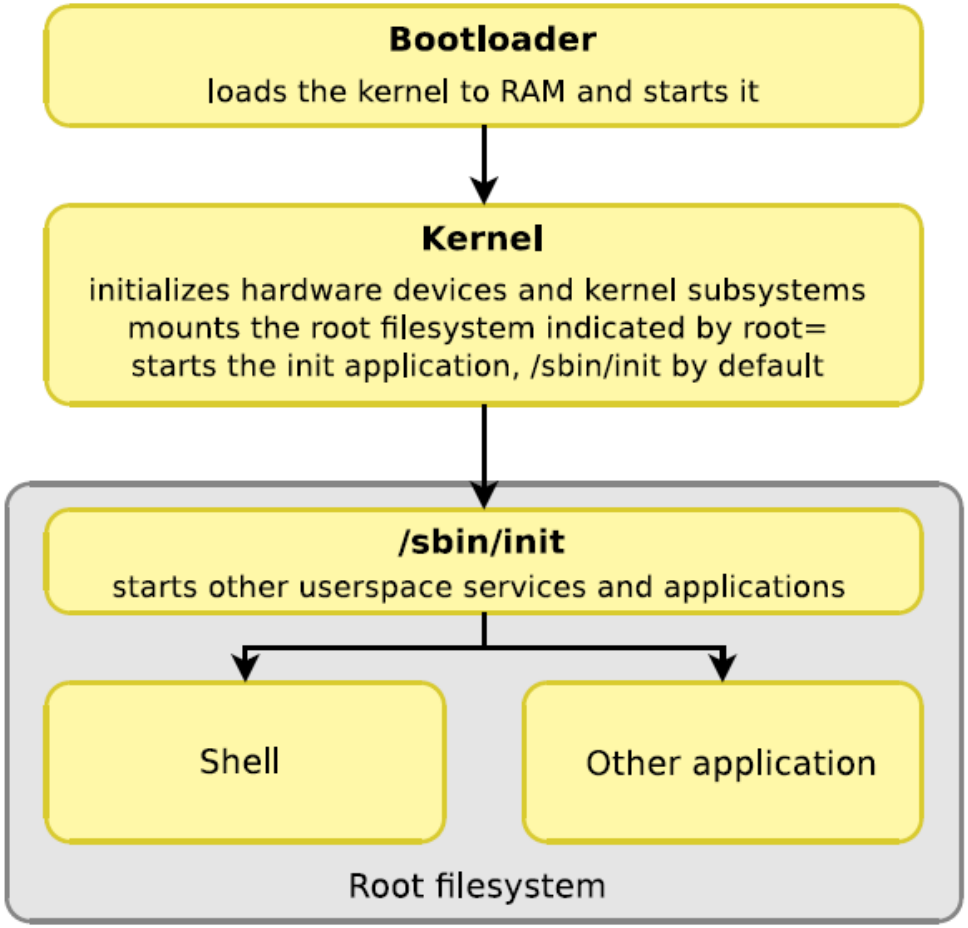
- Organization
  - Well-defined by the Filesystem Hierarchy Standard
    - http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs
- Important directories
  - /bin Basic programs
  - /boot Kernel image (only when the kernel is loaded from a file system, not common on non-x86 architectures)
  - /dev Device files (covered later)
  - /etc System-wide configuration
  - /home Directory for the users home directories
  - /lib Basic libraries
  - /media Mount points for removable media
  - /mnt Mount points for static media
  - /proc Mount point for the proc virtual file system

Root File System

- Important directories
  - /root Home directory of the root user
  - /sbin Basic system programs
  - /sys Mount point of the sysfs virtual filesystem
  - /tmp Temporary files
  - /usr/bin Non-basic programs
  - /usr/lib Non-basic libraries
  - /usr/sbin Non-basic system programs
  - /var Variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files

Root File System

- Overall booting process

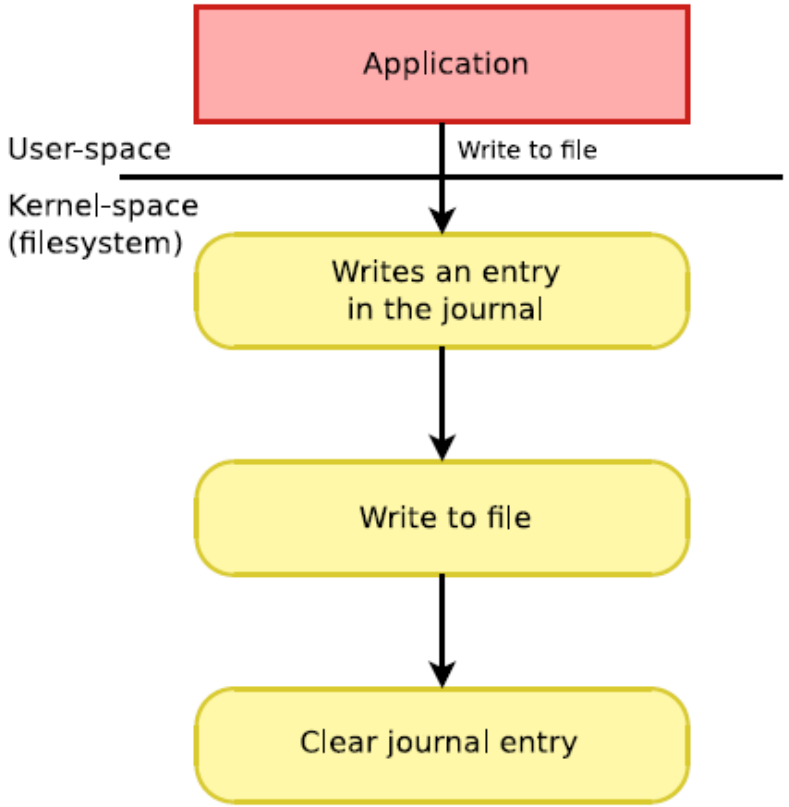


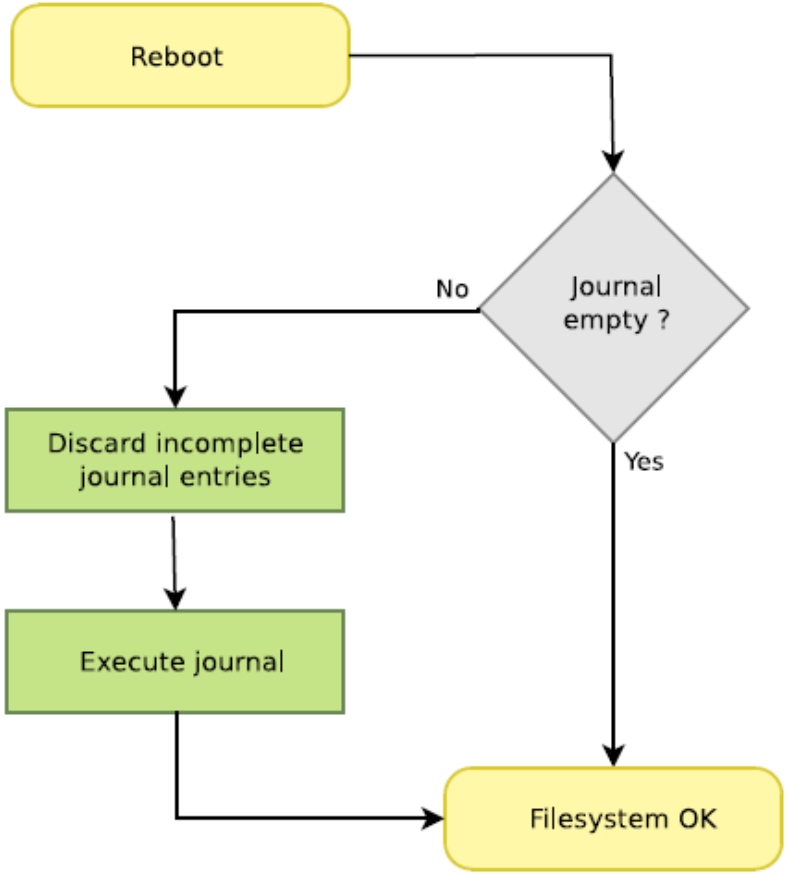
Root File System

- Traditional block file systems
  - ext2
    - Traditional Linux file system
  - vfat
    - Traditional Windows file system
- Journaling file systems
  - ext3
    - ext2 with journal extension
  - ext4
    - New generation with many improvements
- Recommendation
  - ext2 for very small partitions (< 5MB)
  - ext3 and ext4 need about 1MB metadata for a 4MB partition

Root File System

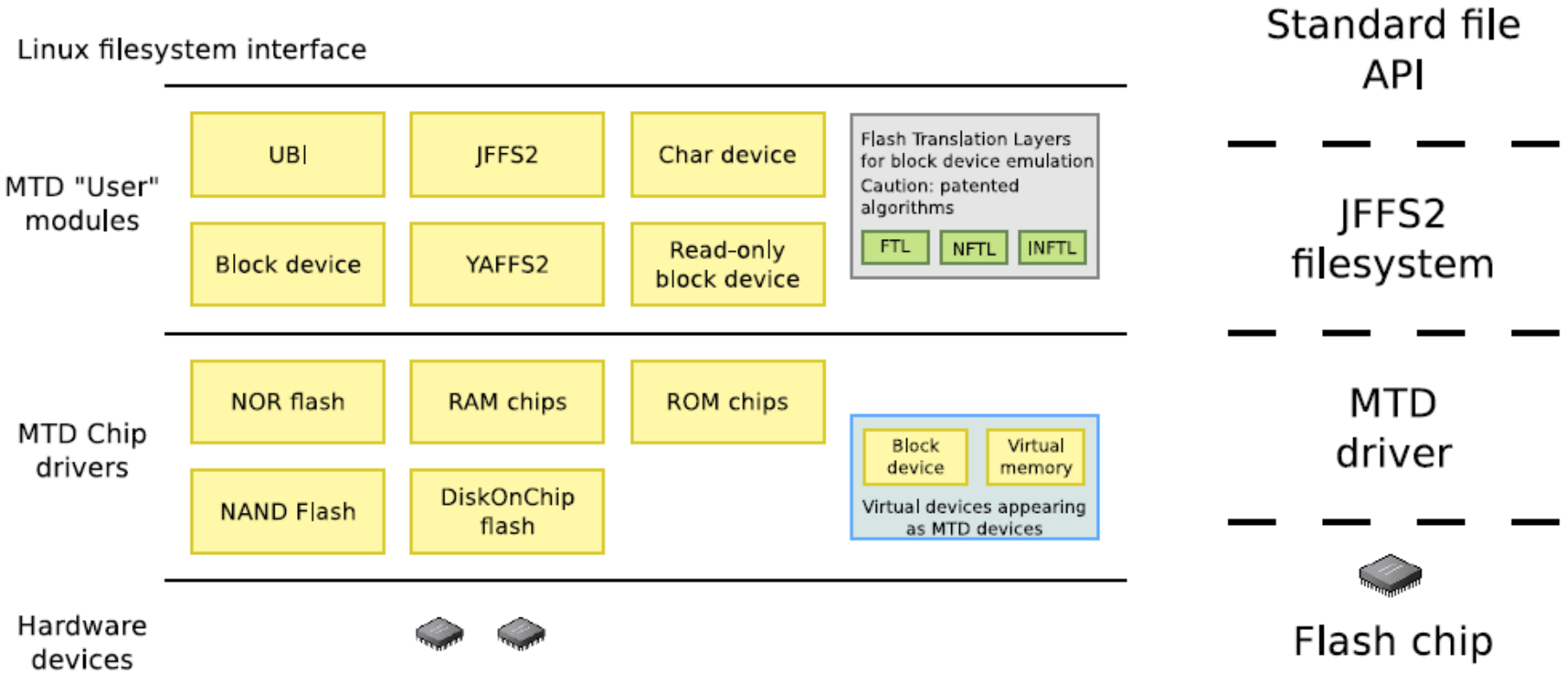
- Journaling file systems





Root File System

- Flash file systems
  - MTD subsystem (Memory Technology Devices)
  - JFFS2 is today's standard file system for MTD flash



Embedded Linux System Development

- Using open-source components
  - Third party libraries and applications
- Tools for the target device
  - Networking
  - System utilities
  - Language interpreters
  - Audio, video and multimedia
  - Graphical toolkits
  - Databases
  - Web browsers

Embedded Linux Application Development

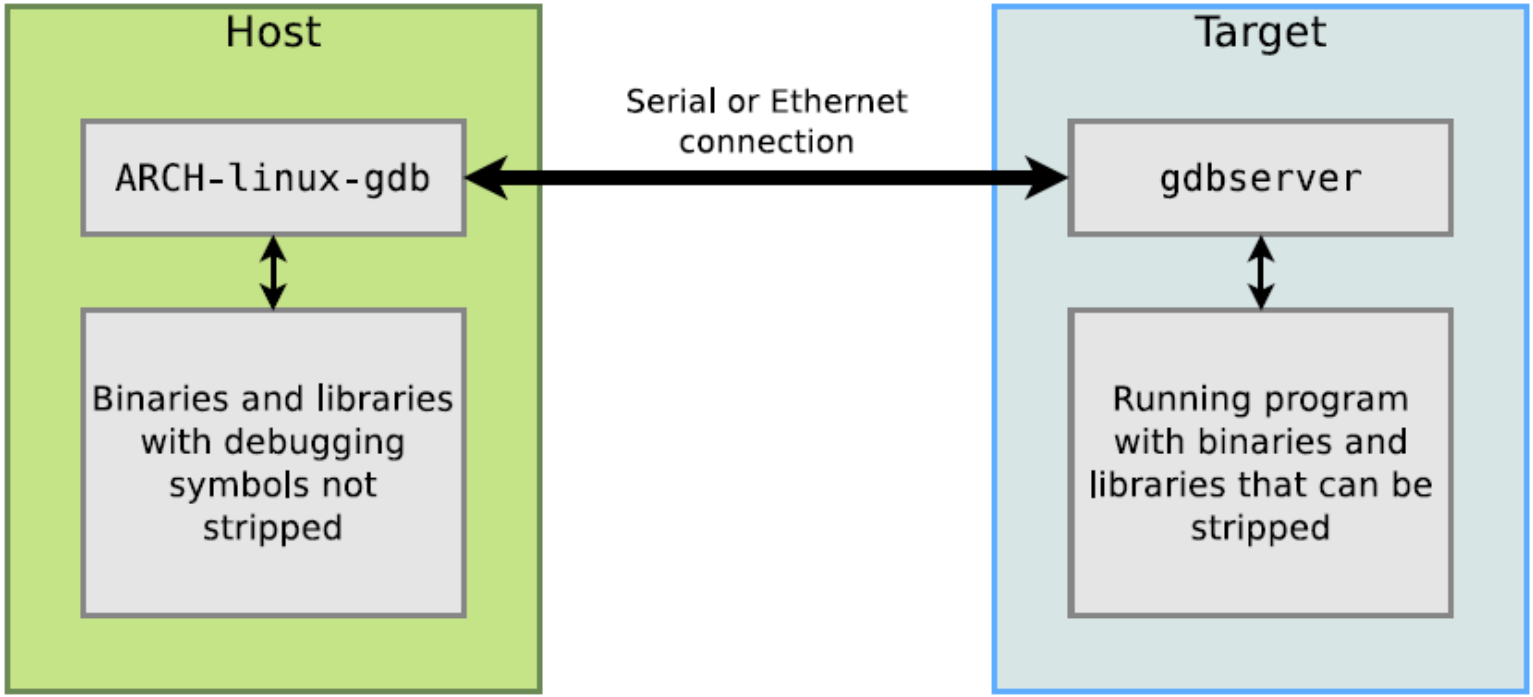
- Application development
  - An embedded Linux system is just a normal Linux system, with usually a smaller selection of components
  - In terms of application development, developing on embedded Linux is exactly the same as developing on a desktop Linux system
  - All existing skills can be re-used, without any particular adaptation
  - All existing libraries, either third-party or in-house, can be integrated into the embedded Linux system
    - Taking into account, of course, the limitation of the embedded systems in terms of performance, storage and memory

- Source browsers
  - LXR (Linux Cross Reference)
    - Web interface
  - cscope
    - Console mode
- Integrated Development Environments (IDE)
  - KDevelop 

  - Eclipse 

- Version control systems
  - Traditional version control systems
    - CVS (Concurrent Versions System)
    - Subversion
  - Distributed source control systems
    - Git
    - Mercurial

- Debugging and analysis tools
  - GDB
  - Remote debugging



- Developing on Windows
  - Cygwin 
  - VMware 
  - VirtualBox 

Q & A

http://mesl.khu.ac.kr