Embedded System

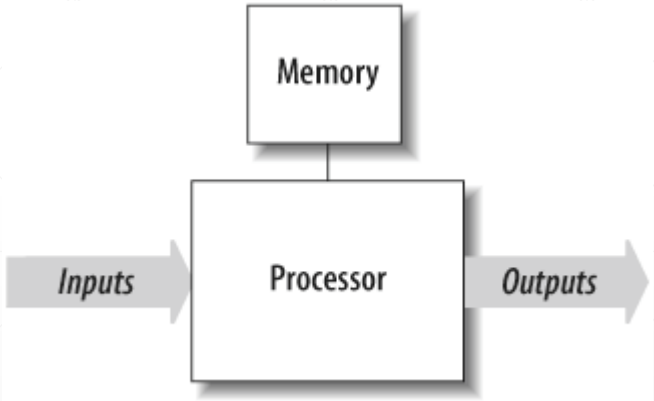
- A general-purpose definition of embedded systems:

  Devices used to **control**, **monitor** or **assist** the operation of equipment

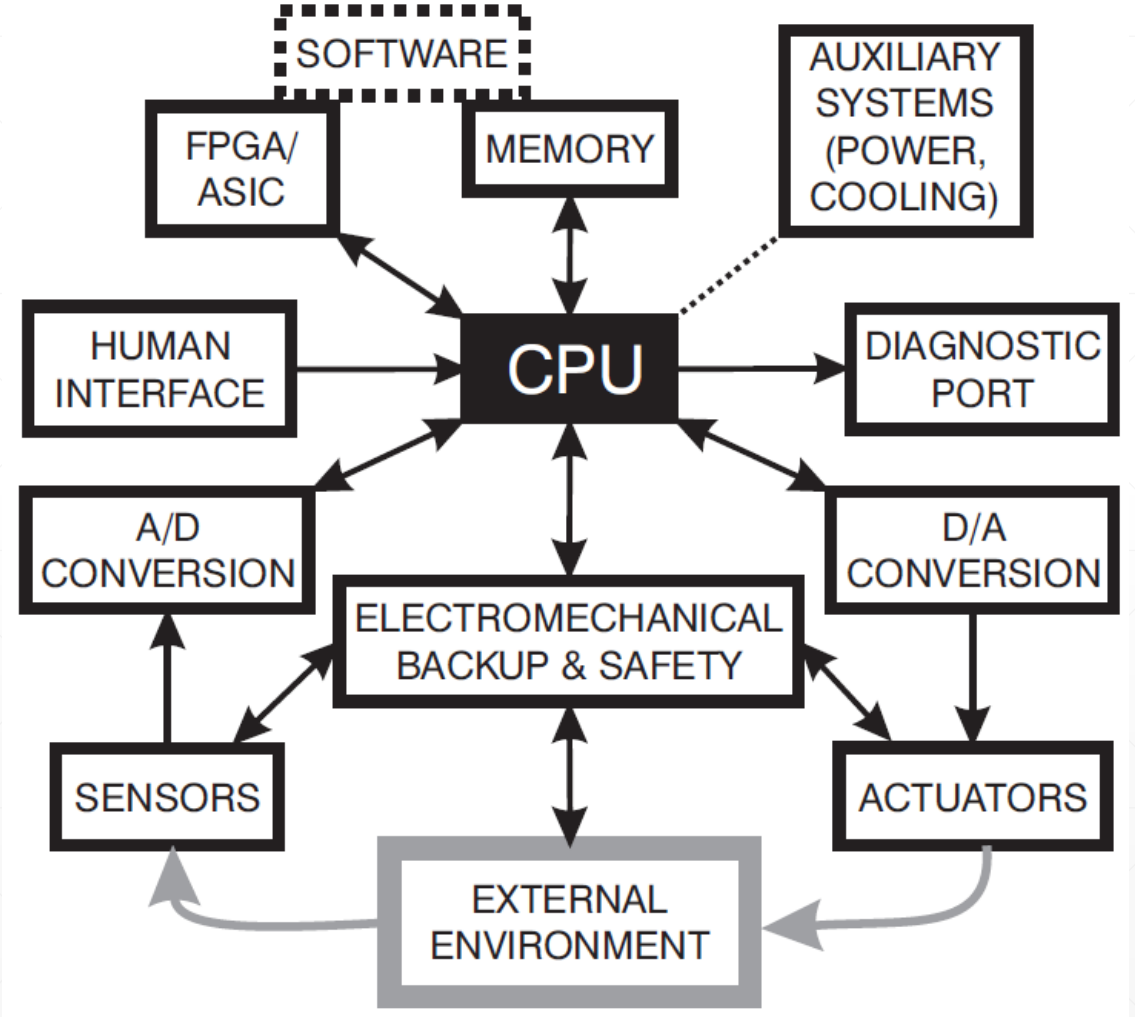
- An **embedded system** is a combination of computer **hardware** and **software** — and perhaps additional parts, either mechanical or electronic — designed to perform a **dedicated** function.

Embedded System

Embedded System



A generic embedded system

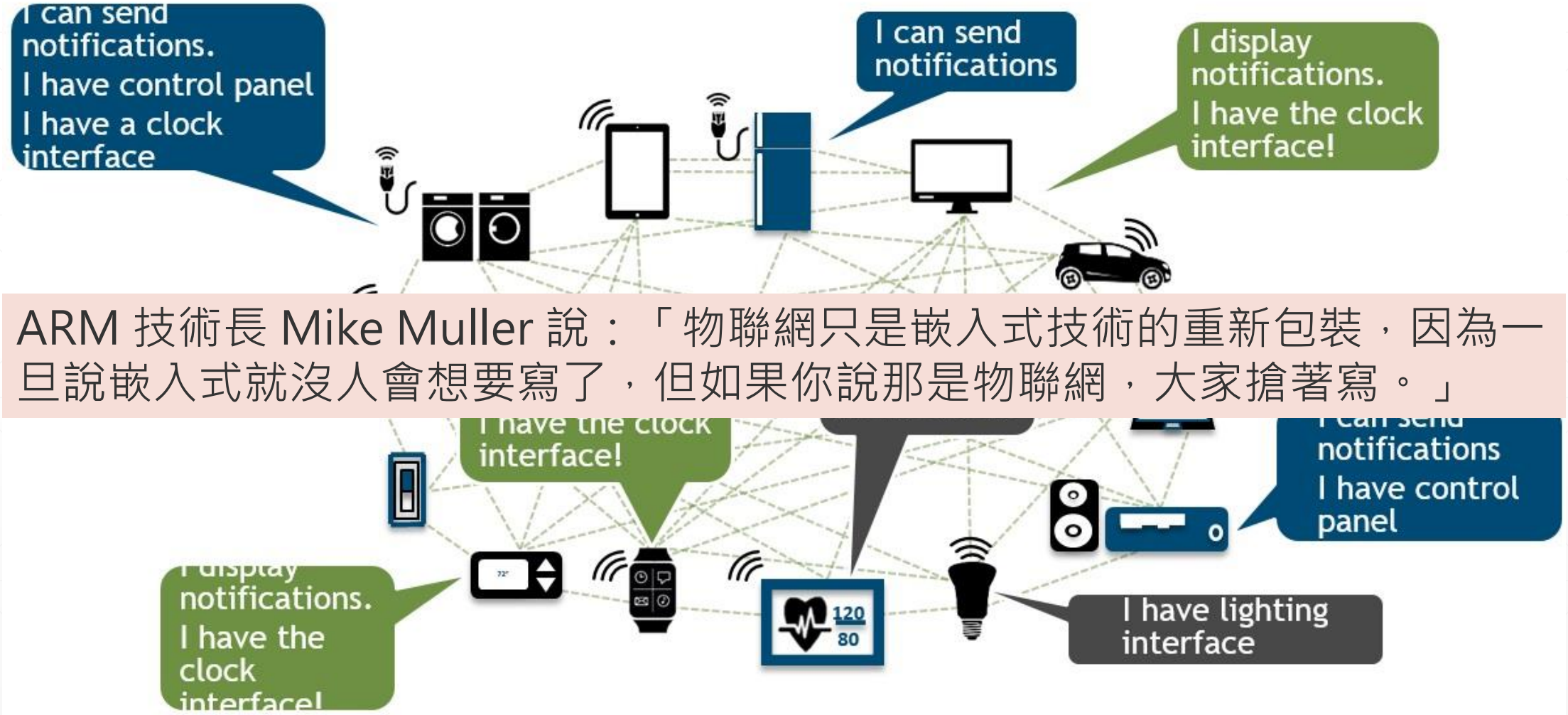


Possible Organization of an Embedded System

Embedded System

- **專用性**：執行特定任務。
- **實時性**：必須在一個可預測和有保證的時間段內對外部事件作出正確反應。
- **高可靠性**：嵌入式系統大多面向控制應用，任何誤動作都可能帶來嚴重後果。
- **軟體固化**：嵌入式系統是軟硬體高度結合的系統，使用者無法變動嵌入式系統的程序功能。
- **資源受限**：嵌入式系統常因為功能設計時的軟硬體裁切、低功耗和低成本的考量因素，所以其軟硬體資源會受到嚴格限制。

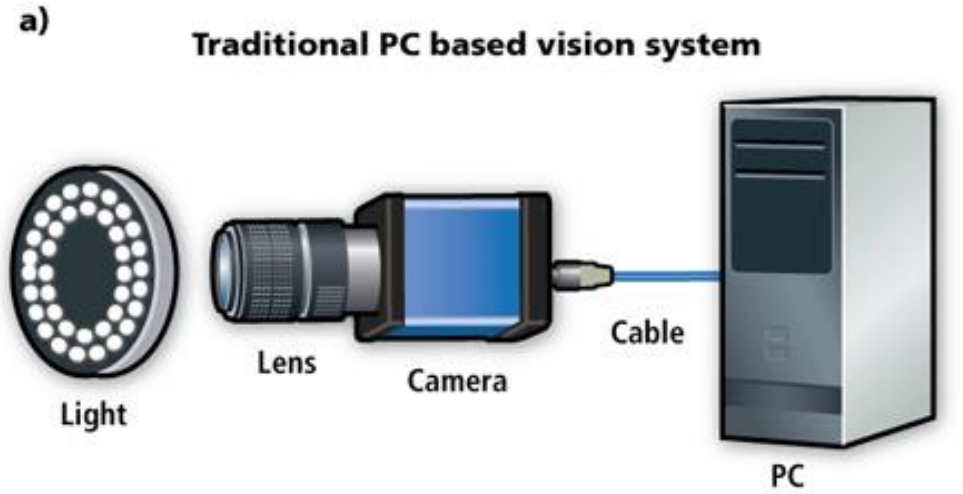
Embedded System – IoT、AIoT(AI+IoT)

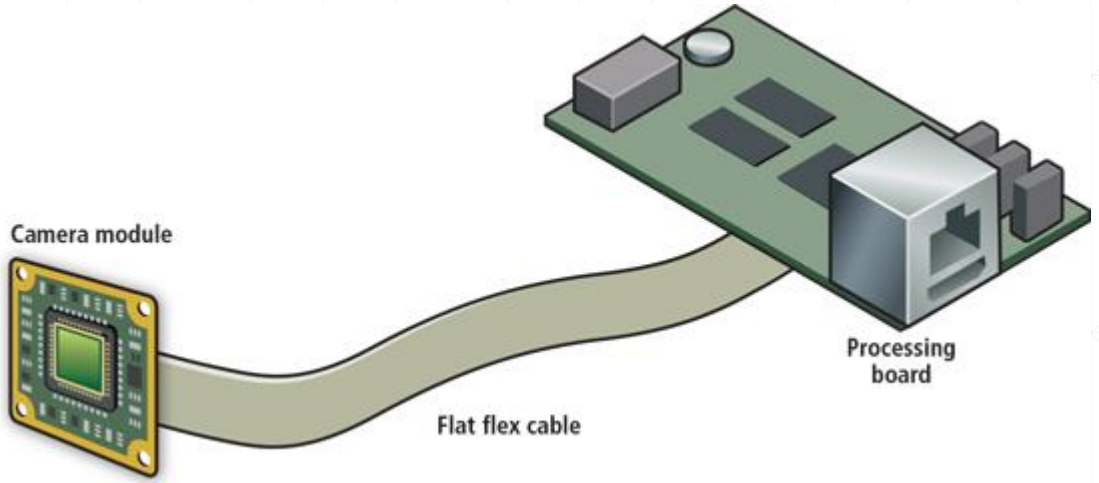


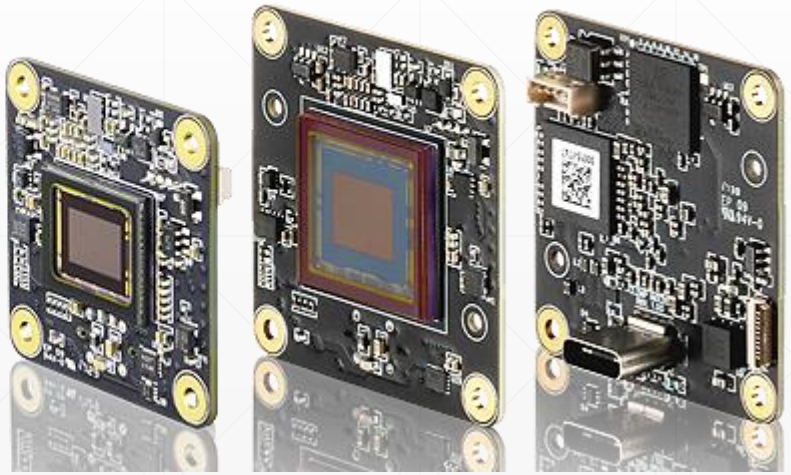
ARM 技術長 Mike Muller 說：「物聯網只是嵌入式技術的重新包裝，因為一旦說嵌入式就沒人會想要寫了，但如果你說那是物聯網，大家搶著寫。」

Image from : https://www.einfochips.com/blog/ongoing-trends-in-iot-device-lifecycle-management/

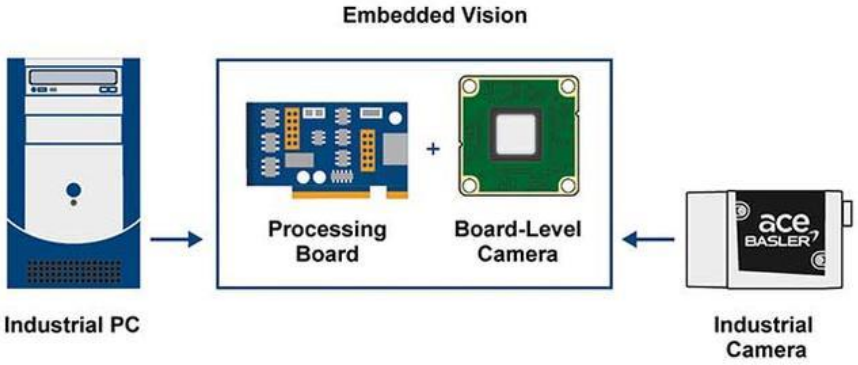
Embedded System

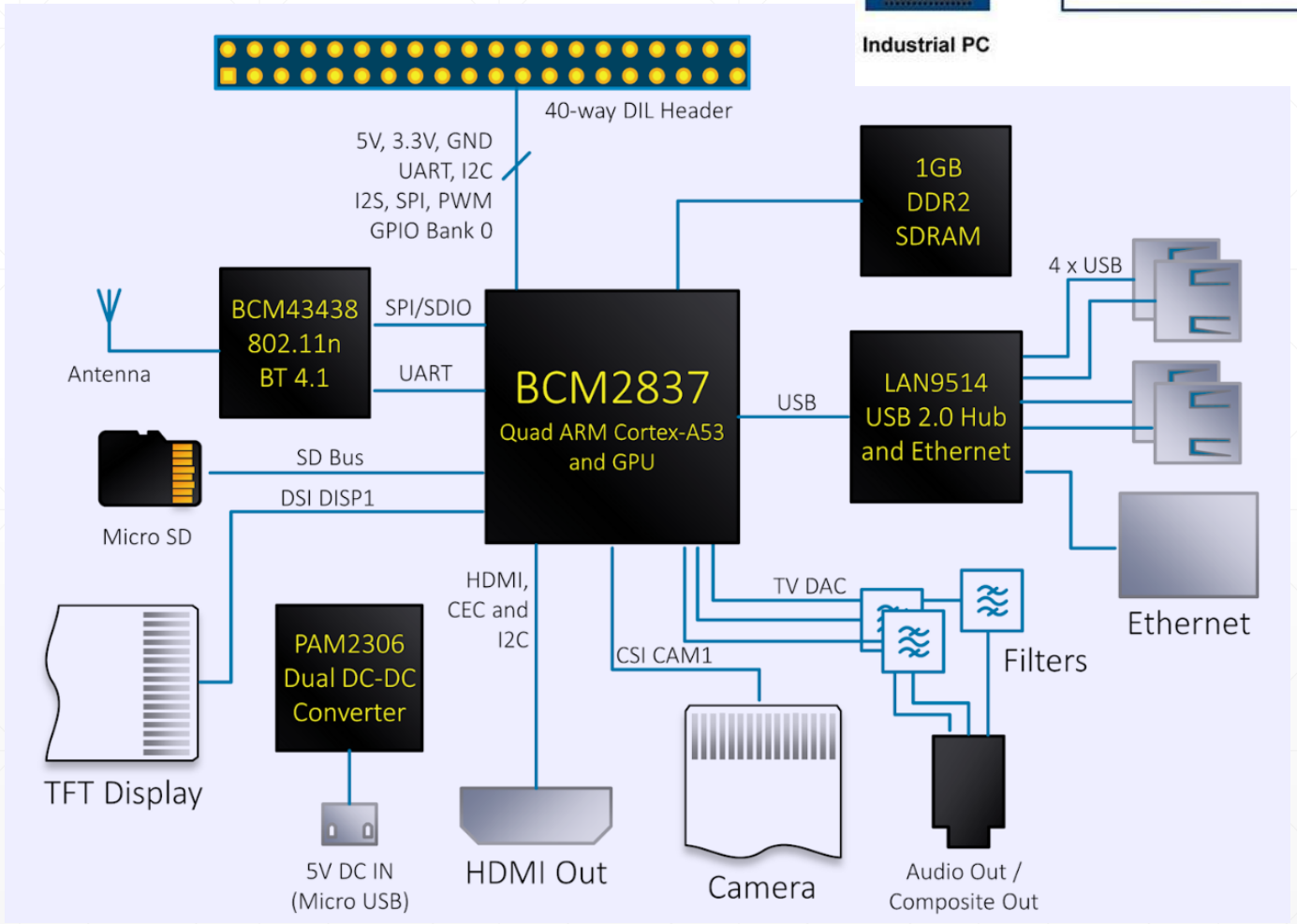






Embedded System



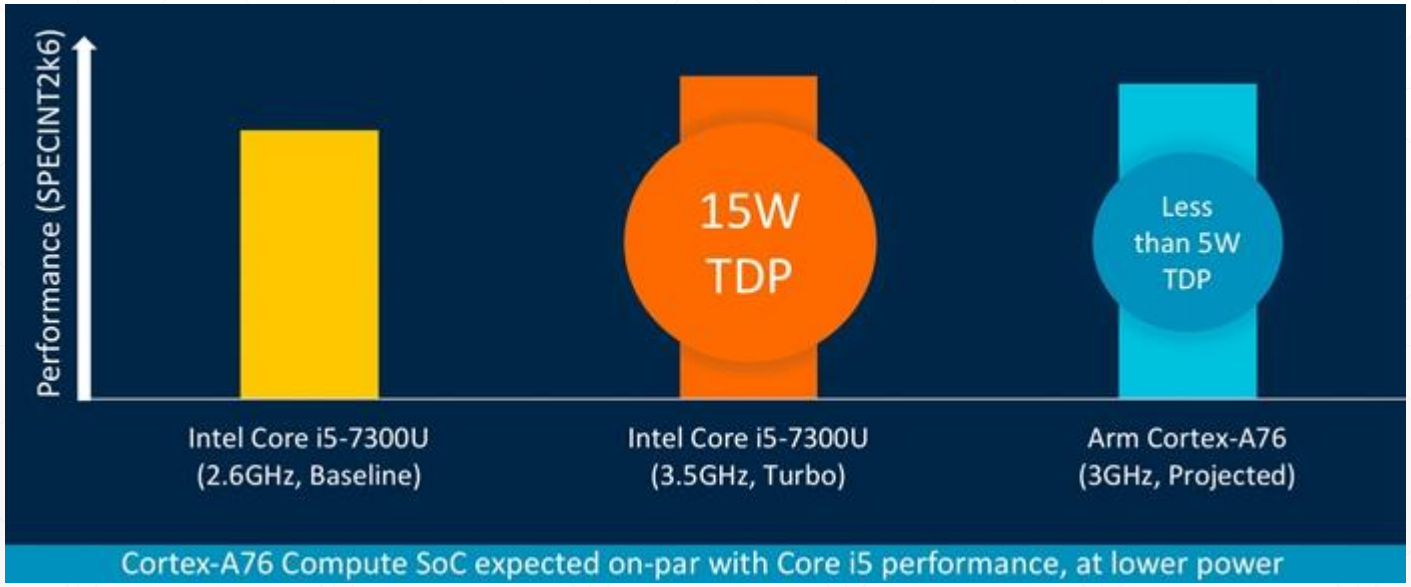


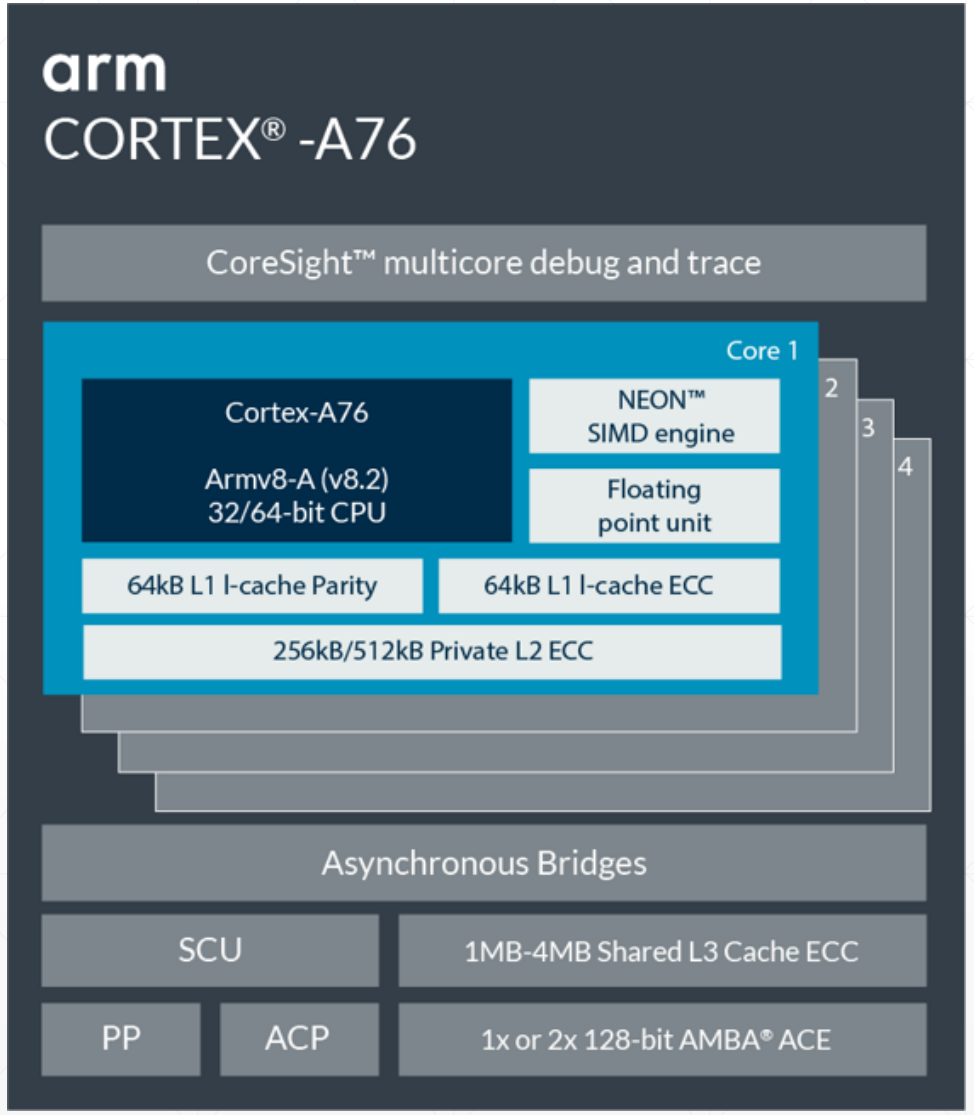
Raspberry pi 3b+ with Camera

Embedded System

- ARM (Advanced RISC Machine)
  - RISC 處理器架構
- 主要設計目標為低成本、高效能、低耗電的特性。
- 幾乎壟斷了所有的行動通訊晶片，

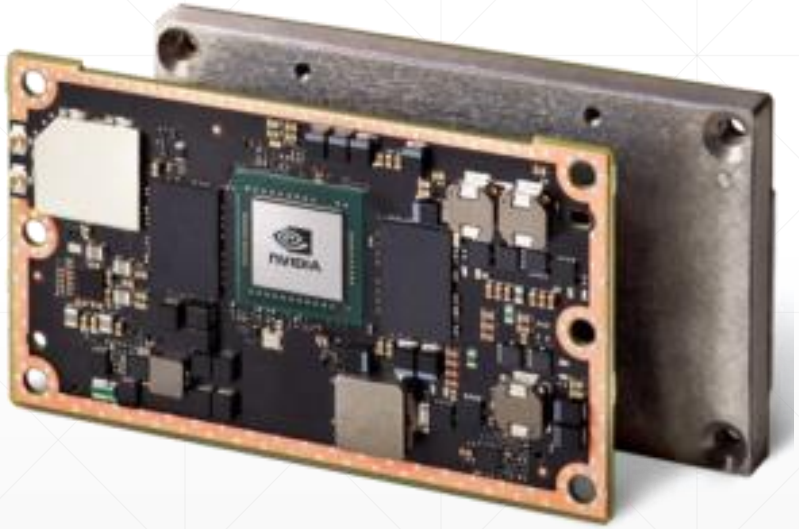
  市佔率高達 95% (智慧型手機)





Embedded System

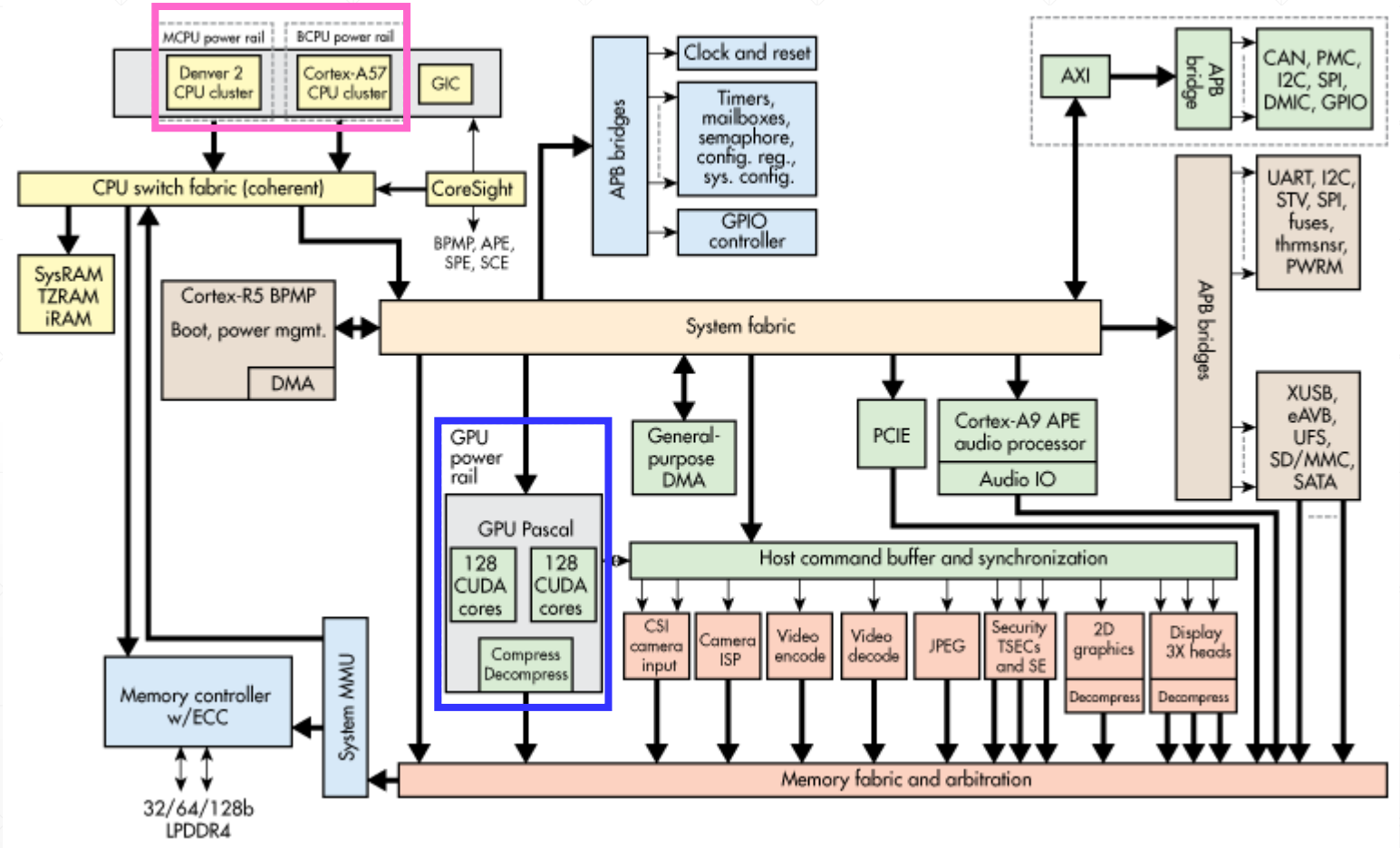
- ARM+GPU



	Jetson TX2
GPU	NVIDIA Pascal™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
VIDEO	4K x 2K 60 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (12-Bit Support)
MEMORY	8 GB 128 bit LPDDR4 59.7 GB/s
DISPLAY	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)
PCIE	Gen 2 | 1x4 + 1x1 OR 2x1 + 1x2
DATA STORAGE	32 GB eMMC, SDIO, SATA
OTHER	CAN, UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0
CONNECTIVITY	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
MECHANICAL	50 mm x 87 mm (400-Pin Compatible Board-to-Board Connector)

Embedded System

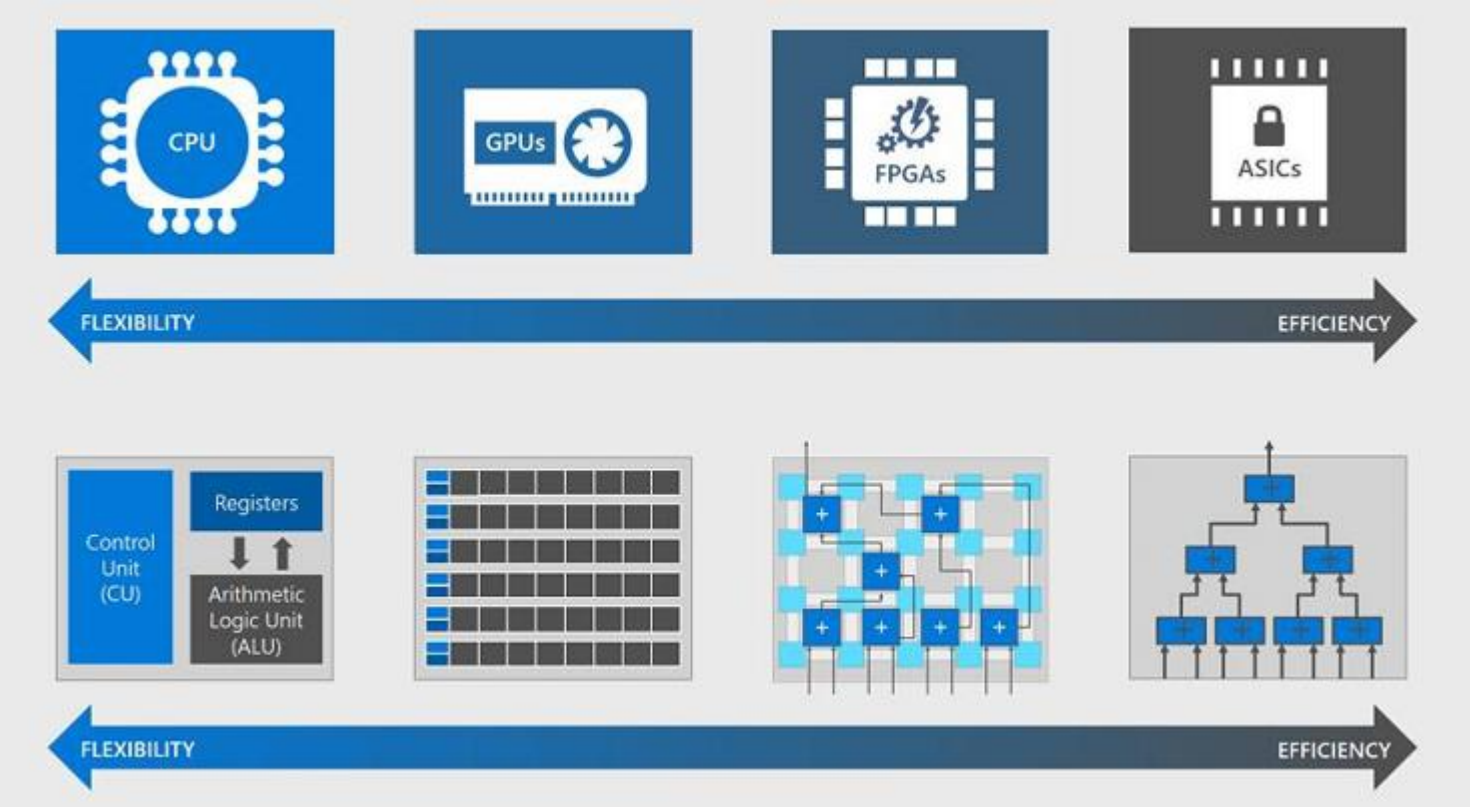
- ARM+GPU



The Jetson TX2 has a 256 CUDA core Pascal-based GPU plus multiple ARM-based cores.

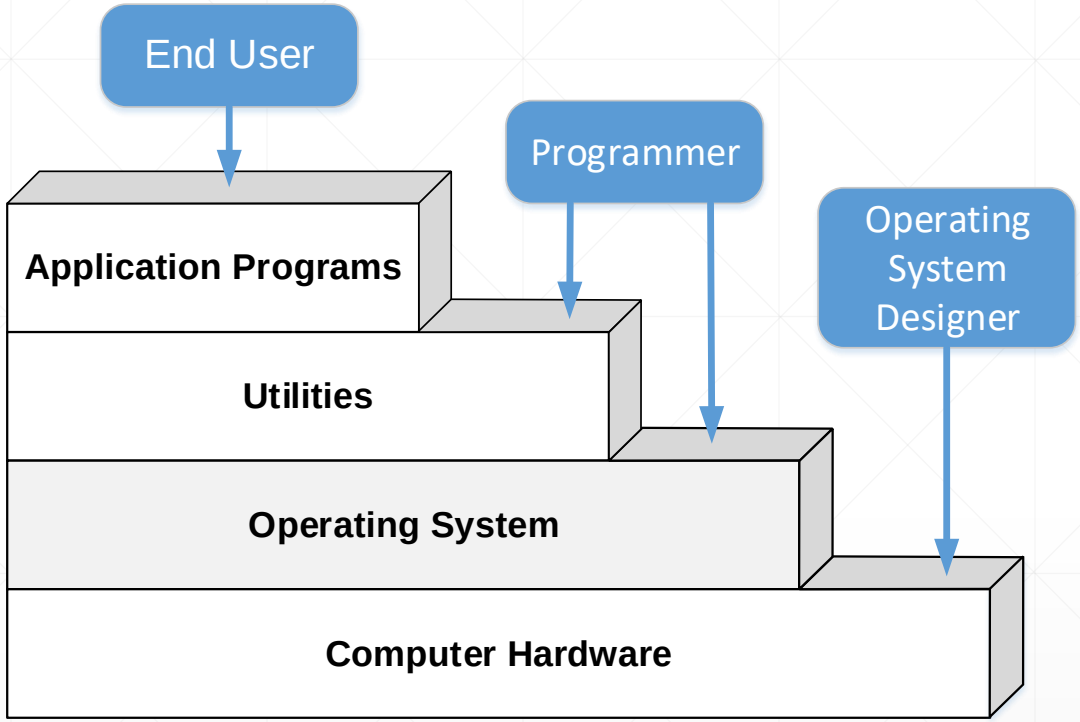
https://www.electronicdesign.com/blog/exploring-jetson-tx2

Embedded System



Embedded Operating System

- Components of an Operating System :
  - Processor Management
  - Memory Management
  - File Management
  - I/O (device) Management



- Layers of Operating System :
  - A program that controls the execution of application programs
  - An interface between applications and hardware

Embedded Operating System

- Using a general-purpose OS for an embedded system may not be possible
  - constraint of memory space
  - constraint of power consumption
  - real-time requirements

- Special-purpose OS designed for the embedded system environment is commonly used.

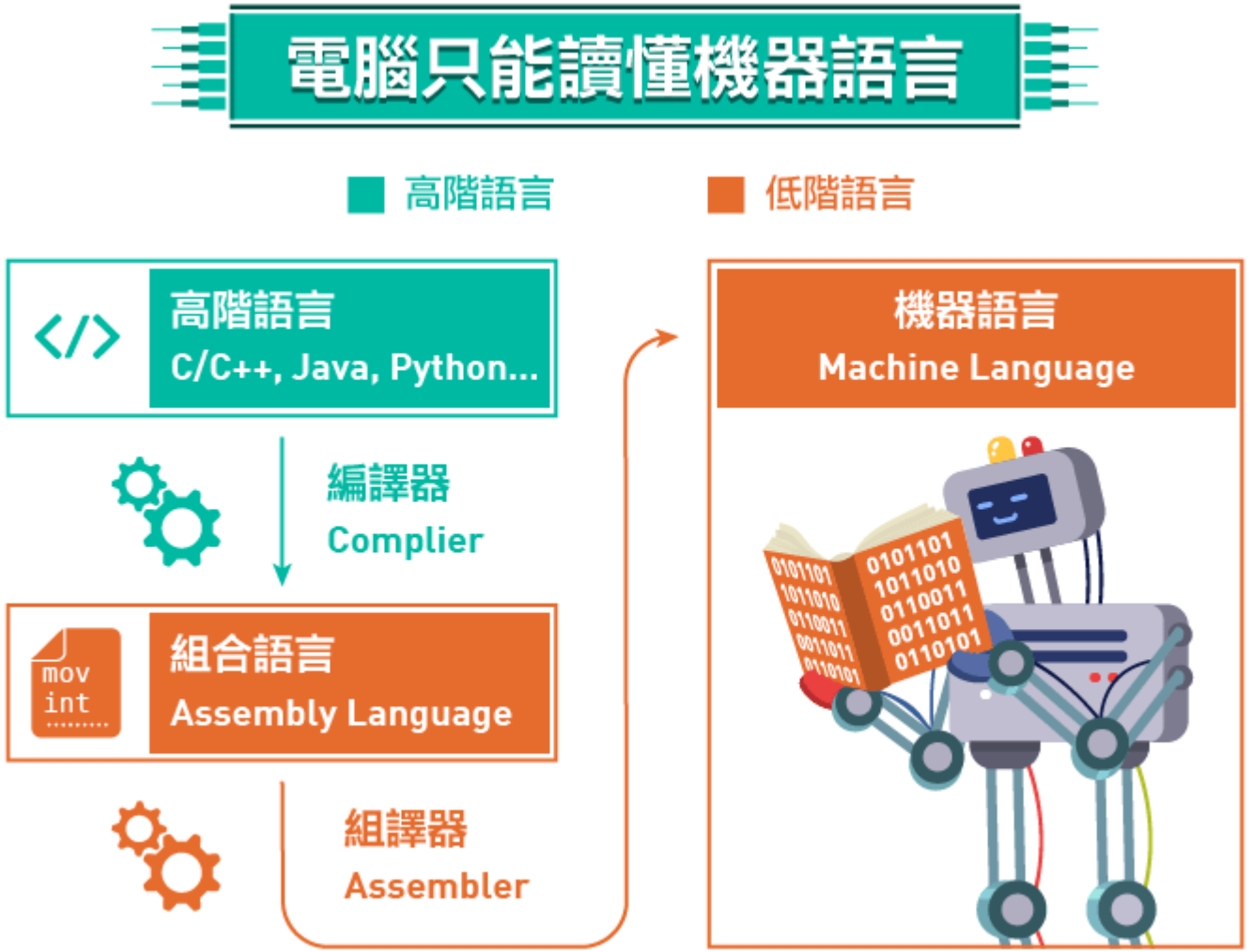
General Purpose OS
- Windows CE
- Windows Mobile
- Embedded Linux

Real Time OS
- VxWorks
- RTLinux
- e-Cos
- uc/OS-II

Embedded Software Porting

- 解開程式編譯的黑幕

Embedded Software Porting



Embedded Software Porting

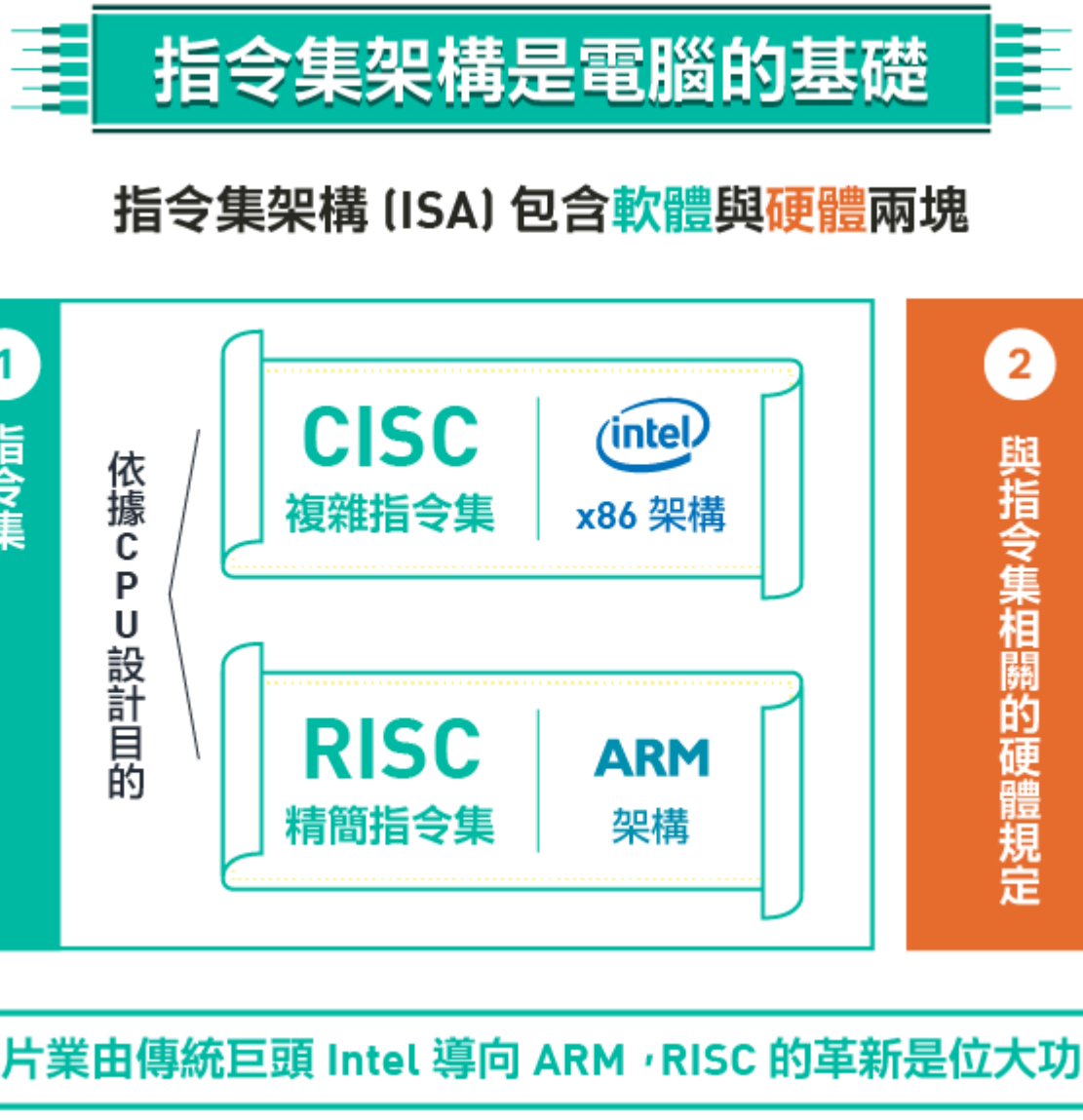
C 語言

```c
# include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
```

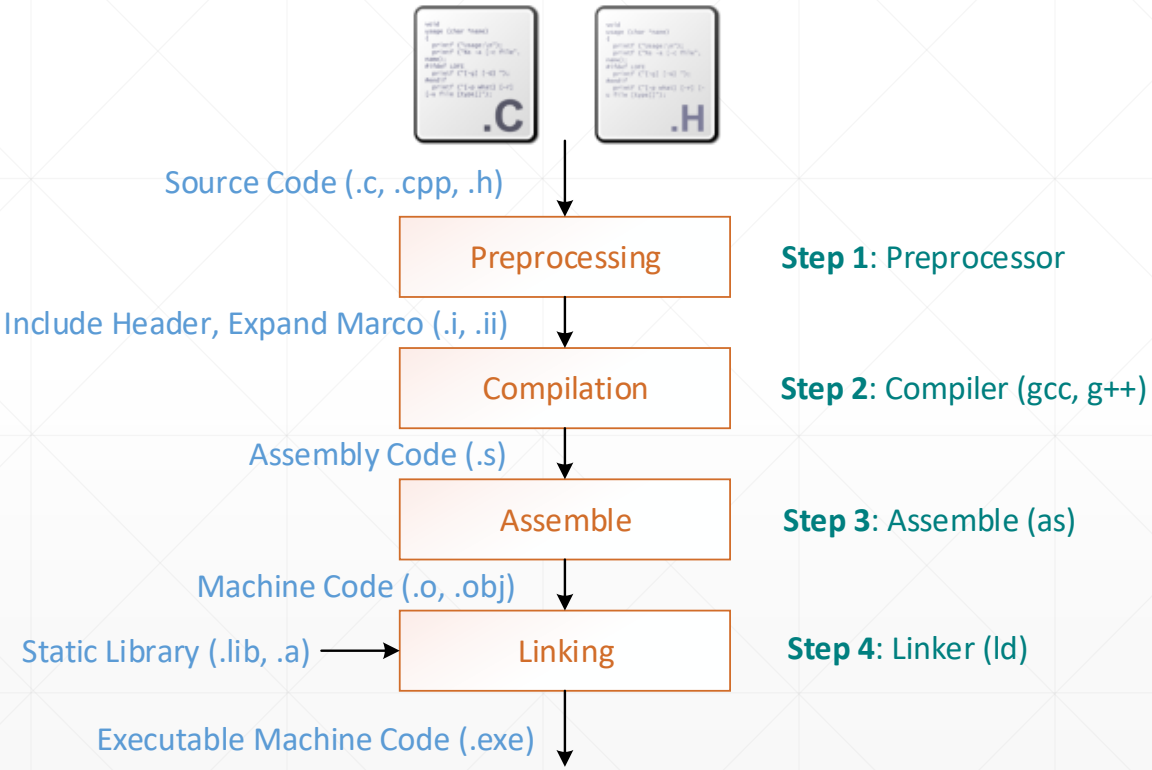
機器語言

```
BGN GRPLOT BIN
DRAWPLOT


0000000000000000000000000000000000000000000000000000
0011100111001111111001110000000011100000000011111000000
0001000010000100001000100000000001000000000010000100000
0001000010000100000000100000000001000000000010000100000
0001111111000011110000100000000001000000000010000100000
0001000010000100000000100000000001000000000010000100000
0001000010000100001000100000000001000000000010000100000
0011100111001111111001111111100011111111000011111000000
0000000000000000000000000000000000000000000000000000


ENDDRAW
END
```

Embedded Software Porting



Embedded Software Porting



Preprocessing :
主要處理 “#” 開頭的前編譯指令。
例如：#include、#define、#pragma

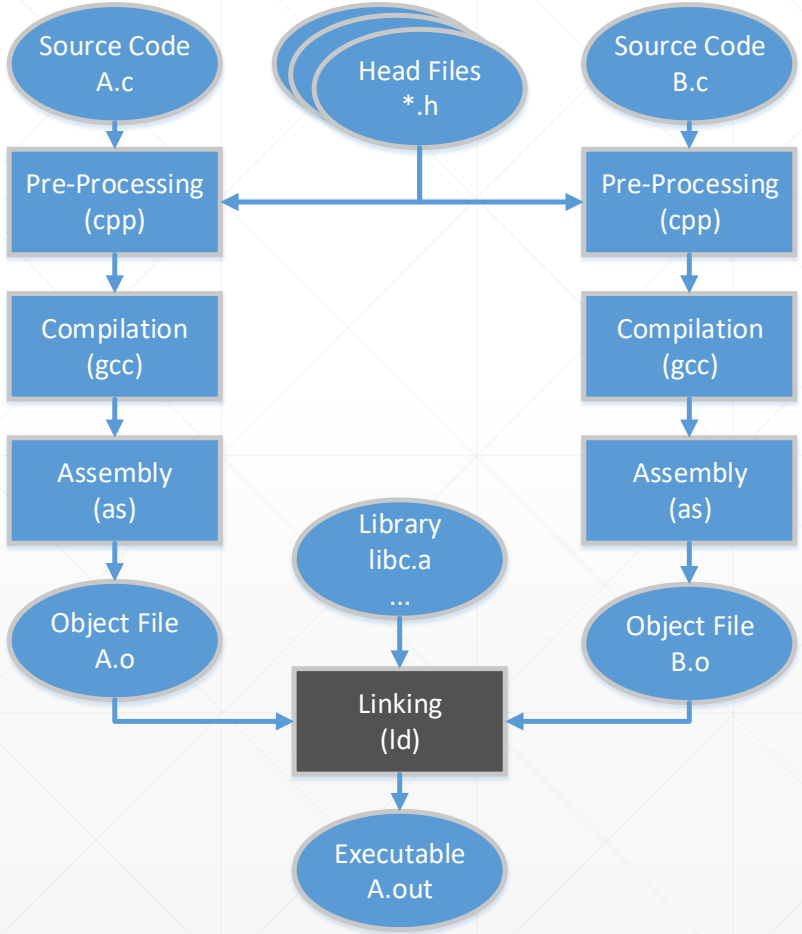
Compilation :
將前處理後的檔案進行詞法分析、語法分析、語意分析、最佳化後，產生對應的組合語言程式碼檔案。

Assemble :
將組合語言程式碼轉換成機器可執行的命令，每個組語語法幾乎都對應一條機器指令。

Linking :
將各模組“組裝”起來。主要流程：位置與記憶體配置、符號解析、重新定位。

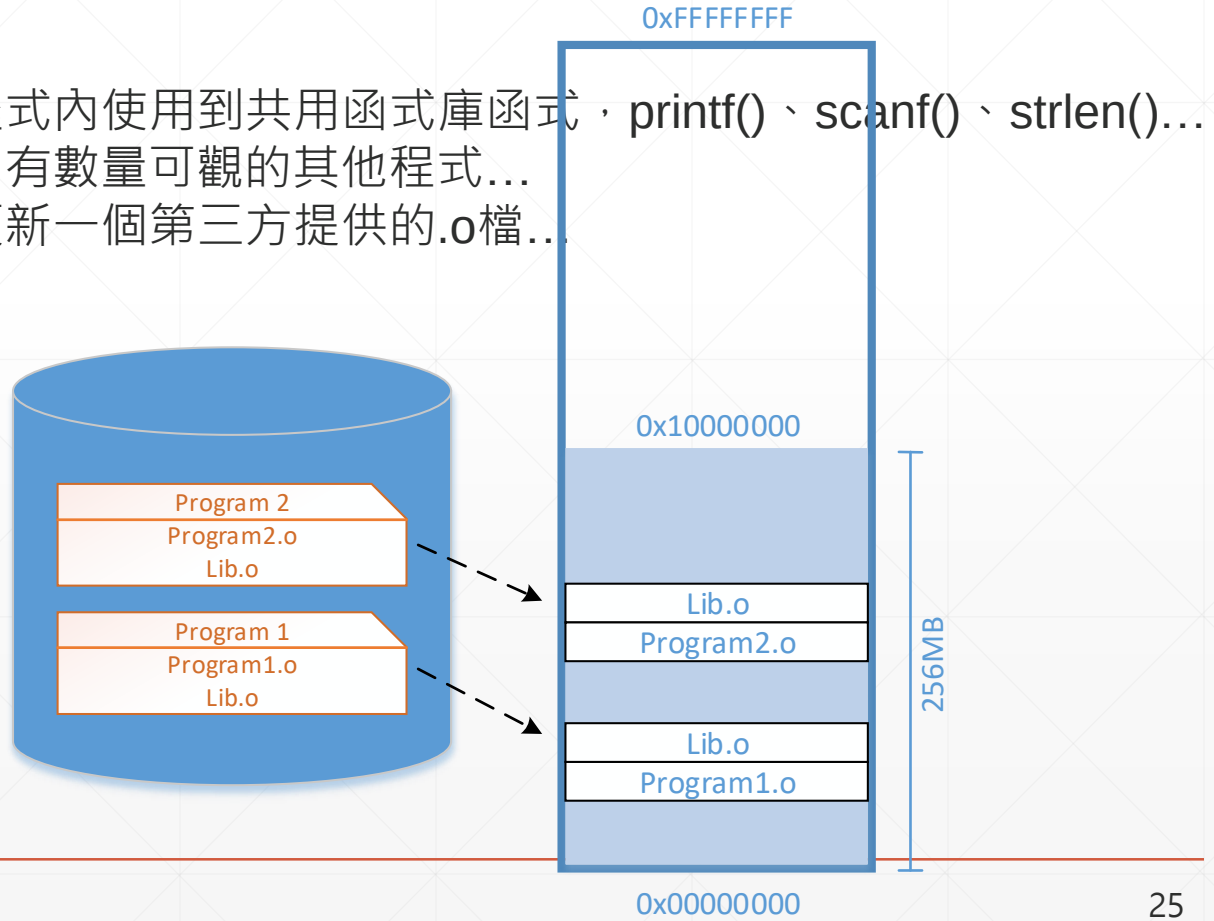
Embedded Software Porting

- 靜態連結 Static Linking



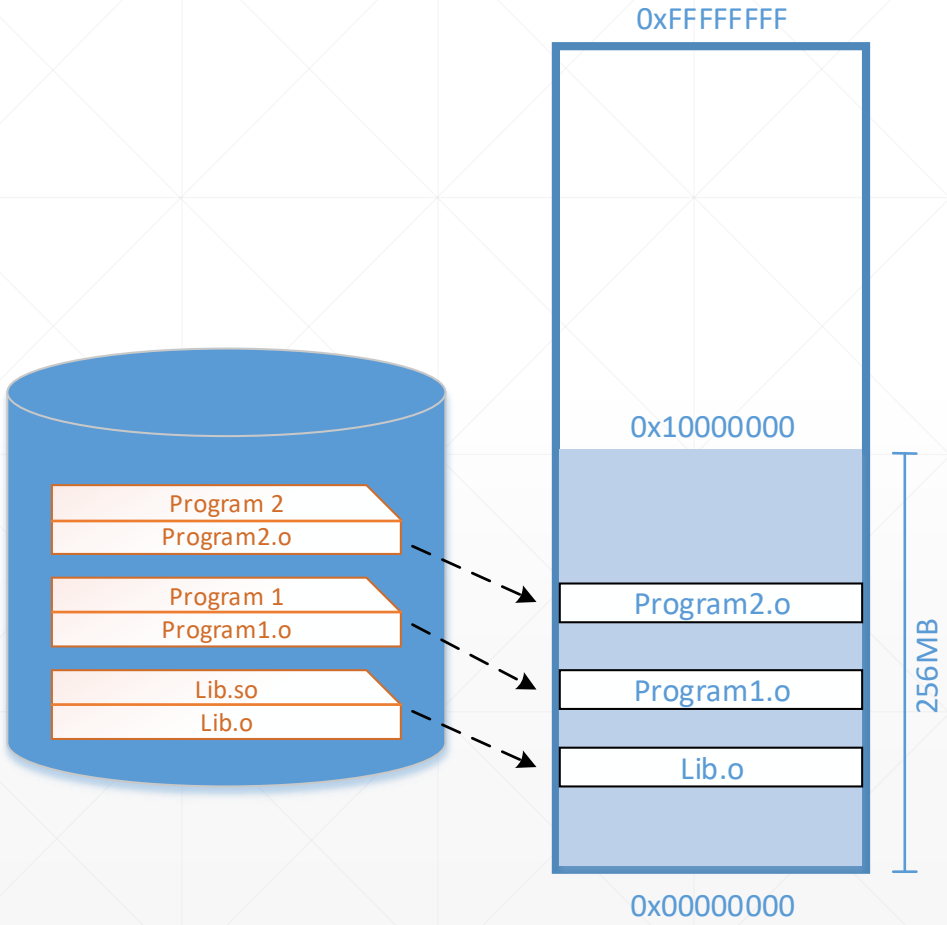
靜態連結問題：
記憶體、磁碟空間的浪費
模組更新困難

Ex:
每個程式內使用到共用函式庫函式，printf()、scanf()、strlen()…
系統中有數量可觀的其他程式…
系統更新一個第三方提供的.o檔…



Embedded Software Porting

- 動態連結 Dynamic Linking



靜態連結問題：
記憶體、磁碟空間的浪費
模組更新困難

動態連結：
- 透過減少儲存副本資料，減少記憶體的浪費，

並減少分頁記憶體的換入、換出，增加CPU快取的效能
- 更換目的檔，即可更新模組 (任意更新、保持最新狀態)

動態連結問題：
新、舊模組需相容。 (DLL HELL)

Embedded Software Porting

- 目的檔：編譯後，未連結的可執行檔格式，其中可能有符號或位置尚未調整，等待連結時調整。

- 可執行檔格式：Windows：PE-COFF; LINUX: ELF

  都屬於COFF(common object file format)的格式變化

PC	PE (Potable Executable)
Linux	ELF (Executable and linkable format)

- Linux 靜態函式庫(.a)，動態函式庫(.so)
  - 照著ELF的格式儲存。
  - 靜態函式庫是將很多的目的檔封裝為一個。
  - 動態函式庫是分割成很多模組，執行時進行連結。

Embedded Software Porting

ELF檔案類型	說明	案例
可重定位檔 (Relocatable file)	包含資料，程式碼可用來連結成執行檔或共用的目的檔以及靜態庫	Linux .o Windows .obj
執行檔 (executable file)	可直接執行的程式	Linux /bin/bash Windows .exe
共用目的檔 (shared object file)	包含資料，程式碼。 1.可用連接器與其他**可重定位檔**、**共用目的檔**鏈結，產生新的目的檔。 2.可用**動態**連接器與**共用目的檔**與**可執行檔**做為映像檔的一部分來執行。	Linux .so Windows .DLL
核心傾印檔 (core dump file)	當程式意外終止時，將程式位置與其內容及終止時的其他資料印出來	Linux Core dump

Embedded Software Porting

- Linux file 指令：查看檔案格式

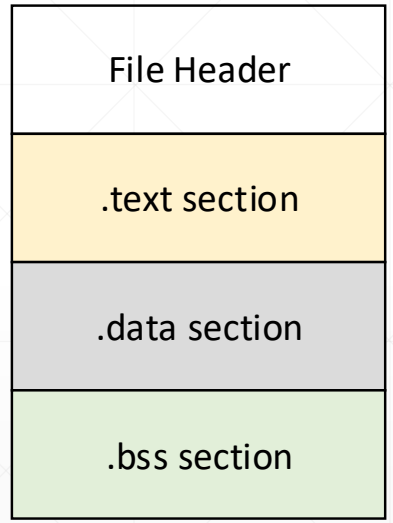
```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello Embedded System !" << endl;
    return 0;
}
```

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# g++ -o Hello.exe Hello.cpp
root@hank-X302LJ:/home/hank#
```

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# g++ -o Hello.exe Hello.cpp
root@hank-X302LJ:/home/hank# file Hello.exe
Hello.exe: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically link
ed, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=dc812f6c47d6f931a5ecbc6e0a2b8699b7dcb26c, not stripped
root@hank-X302LJ:/home/hank#
```

Embedded Software Porting

- 目的檔格式為ELF格式，其中包含：資料，程式碼，其實還有連接時所需的資訊：符號表、除錯資訊、字串等等。根據不同屬性以“**節**”(**Section**)的格式來做儲存。 **節**也稱為**段**或**區段**(**Segment**)。



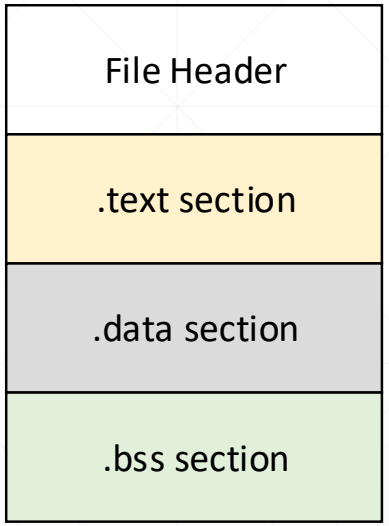
- ELF檔頭：
  - 描述檔案屬性：是否可執行、動態連結還靜態連結、入口位置(如果是可執行檔)、目標硬體、目標作業系統、段表(Section table)...等資訊。
- 程式碼編譯後的**機械指令**放置於：**程式碼區段 Code Section**，常見名稱為：**.code**、**.text**
- **全域變數**與**區域靜態變數**資料放置於：**資料區段 Data Section**，名稱為：**.data**

Embedded Software Porting

```
int g_init_var = 38;
int g_uinit_var =;

void func1(int i)
{
        printf("%d\n",i);
}

int main()
{
        int static_var1 = 58;
        int static_var2 ;

        int a = 1;
        int b;

        func1(var1 + var2 + a + b);
        return 0;
}
```

```
int g_init_var = 38;
int g_uinit_var ;
void func1(int i)
{
        printf("%d\n",i);
}

int main()
{
        int static_var1 = 58;
        int static_var2 ;
        int a = 1;
        int b;

        func1(var1 + var2 + a + b);
        return 0;
}
```


Executable File /
Object File



Code Section

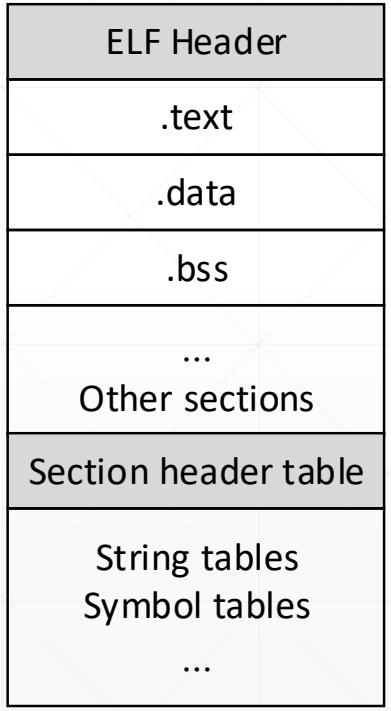
Data Section

.bss ：只為了未初始化的全域變數和區域靜態變數預留位置

程式原始碼編譯後主要分成：**程式指令**、**程式資料**
程式指令：程式碼區段
程式資料：資料區段、bss區段

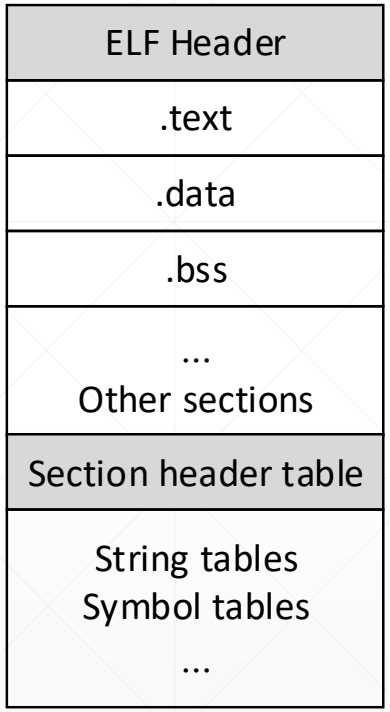
Embedded Software Porting

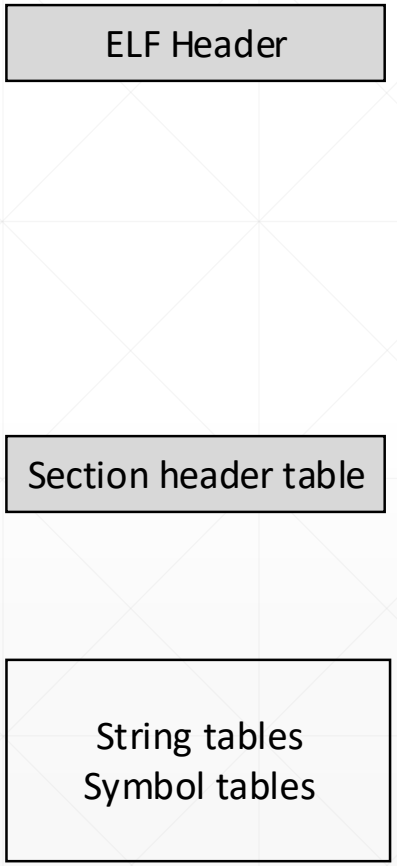
- 還有**唯獨資料區段(.rodata)**、**注釋資訊區段(.comment)**...



常用的區段名稱	說明
.rodata1	唯獨資料，比如字串常數、全域const變數，與.rodata一樣
.comment	存放編輯器版本資訊
.debug	除錯資訊
.dynamic	動態連結資訊
.hash	符號雜湊表
.line	除錯時的行號表(原始碼行號與編輯氣候指令對應表)
.note	額外的編譯資訊。例如：發佈版號...等等
.strtab	字串符號表，儲存ELF檔中用到的字串
.symtab	符號表
.shstrtab	區段名稱表
.plt .got	動態連結的跳轉表和全域入口表
.init .fini	程式初始化與終結程式碼區段

Embedded Software Porting





描述檔案屬性：是否可執行、動態連結或靜態連結、入口位置(如果是可執行檔)、目標硬體、目標作業系統、段表(Section table)...等資訊。

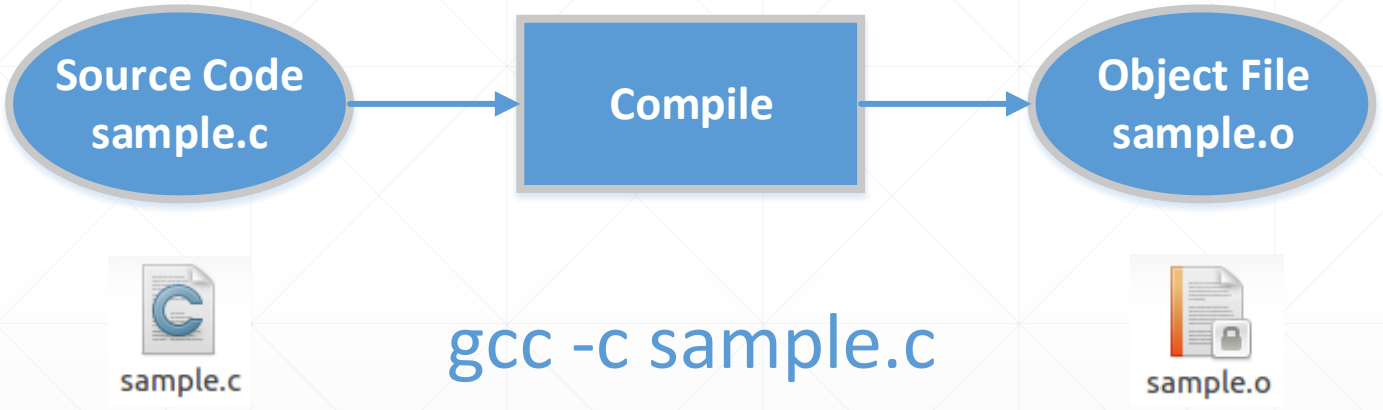
描述此檔案所有區段的訊息，例如：每個區段名稱、區段長度、偏移與讀寫權限等等屬性。

輔助性的結構：字串表、符號表等

Embedded Software Porting

- 讀取ELF檔範例

```c
/* sample.c */

int printf(const char* format, ...);

int global_init_var = 84;
int global_uninit_var;

void func1(int i)
{
  printf("%d\n", i);
}

int main(void)
{
    static int static_var = 85;
    static int static_var2;

    int a = 1;
    int b;

    func1(static_var + static_var2 + a + b);

    return a;
}
```



參數 –c 表示僅編譯不做連結

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# gcc -c sample.c
root@hank-X302LJ:/home/hank#
```

Linux 讀取ELF工具：readelf

讀取Header指令： # **readelf -h**

讀取Section：# **readelf -S**

讀取symbol：# **readelf -s**

root@hank-X302LJ: /home/hank

```
root@hank-X302LJ:/home/hank# readelf
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
  --dyn-syms             Display the dynamic symbol table
  -n --notes             Display the core notes (if present)
  -r --relocs            Display the relocations (if present)
  -u --unwind            Display the unwind info (if present)
  -d --dynamic           Display the dynamic section (if present)
  -V --version-info      Display the version sections (if present)
  -A --arch-specific     Display architecture specific information (if any)
  -c --archive-index     Display the symbol/file index in an archive
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  -x --hex-dump=<number|name>
                         Dump the contents of section <number|name> as bytes
  -p --string-dump=<number|name>
                         Dump the contents of section <number|name> as strings
  -R --relocated-dump=<number|name>
                         Dump the contents of section <number|name> as relocated bytes
  -z --decompress        Decompress section before dumping it
  -w[lLiaprmfFsoRt] or
  --debug-dump[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
               =frames-interp,=str,=loc,=Ranges,=pubtypes,
               =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
               =addr,=cu_index]
                         Display the contents of DWARF2 debug sections
  --dwarf-depth=N        Do not display DIEs at depth N or greater
  --dwarf-start=N        Display DIEs starting with N, at the same depth
                         or deeper
  -I --histogram         Display histogram of bucket list lengths
  -W --wide              Allow output width to exceed 80 characters
  @<file>                Read options from <file>
  -H --help              Display this information
  -v --version           Display the version number of readelf
```

Embedded Software Porting

- Read ELF file head

 # readelf –h sample.o

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# gcc -c sample.c
root@hank-X302LJ:/home/hank# readelf -h sample.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1064 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
root@hank-X302LJ:/home/hank#
```

- Read ELF Section Header Table

 # readelf –S sample.o

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# readelf -S sample.o
There are 13 section headers, starting at offset 0x428:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000055  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000318
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  00000098
       0000000000000008  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000a0
       0000000000000004  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000a0
       0000000000000004  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000a4
       0000000000000036  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000da
       0000000000000000  0000000000000000           0     0     1
  [ 8] .eh_frame         PROGBITS         0000000000000000  000000e0
       0000000000000058  0000000000000000   A       0     0     8
  [ 9] .rela.eh_frame    RELA             0000000000000000  00000390
       0000000000000030  0000000000000018   I      11     8     8
  [10] .shstrtab         STRTAB           0000000000000000  000003c0
       0000000000000061  0000000000000000           0     0     1
  [11] .symtab           SYMTAB           0000000000000000  00000138
       0000000000000180  0000000000000018          12    11     8
  [12] .strtab           STRTAB           0000000000000000  000002b8
       000000000000005f  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
root@hank-X302LJ:/home/hank#
```

Embedded Software Porting

符號繫結資訊		
定義名稱	值	說明
LOCAL	0	區域符號，外部不可見
GLOBAL	1	全域符號，外部可見
WEAK	2	弱引用

符號類型		
定義名稱	值	說明
NOTYPE	0	未知型態符號
OBJECT	1	資料物件; 如變數、陣列..等等
FUNC	2	函式或其他可執行程式碼
SECTION	3	表示為一個區段，必須為LOCAL
FILE	4	表示為檔案名稱，一般都是物件檔所對應的原始檔案名稱

Embedded Software Porting

符號所在區段特殊常數		
定義名稱	值	說明
ABS	0xfff1	該符號包含了一個絕對的值，例如檔案名稱的符號
COMMON	0xfff2	該符號為COMMON區塊類型的符號，一般來說未初始化的全域符號定義為此類型
UNDEF	0	該符號未定義，該符號被此物件檔引用，但定義在其他物件檔中。

Embedded Software Porting

- Read ELF symbol

 # readelf –s sample.o

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# readelf -s sample.o

Symbol table '.symtab' contains 16 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS sample.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 static_var.1840
     7: 0000000000000000     4 OBJECT  LOCAL  DEFAULT    4 static_var2.1841
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
    11: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 global_init_var
    12: 0000000000000004     4 OBJECT  GLOBAL DEFAULT  COM global_uninit_var
    13: 0000000000000000    34 FUNC    GLOBAL DEFAULT    1 func1
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    15: 0000000000000022    51 FUNC    GLOBAL DEFAULT    1 main
root@hank-X302LJ:/home/hank#
```

Embedded Software Porting

- 靜態連結

```
/* a.c */
extern int shared;

int main()
{
        int a = 100;
        func(&a, &shared);
}
```

```
/* b.c */
int shared = 1;

void func(int *a ,int *b)
{
        *a ^= *b ^= *a ^= *b;
}
```

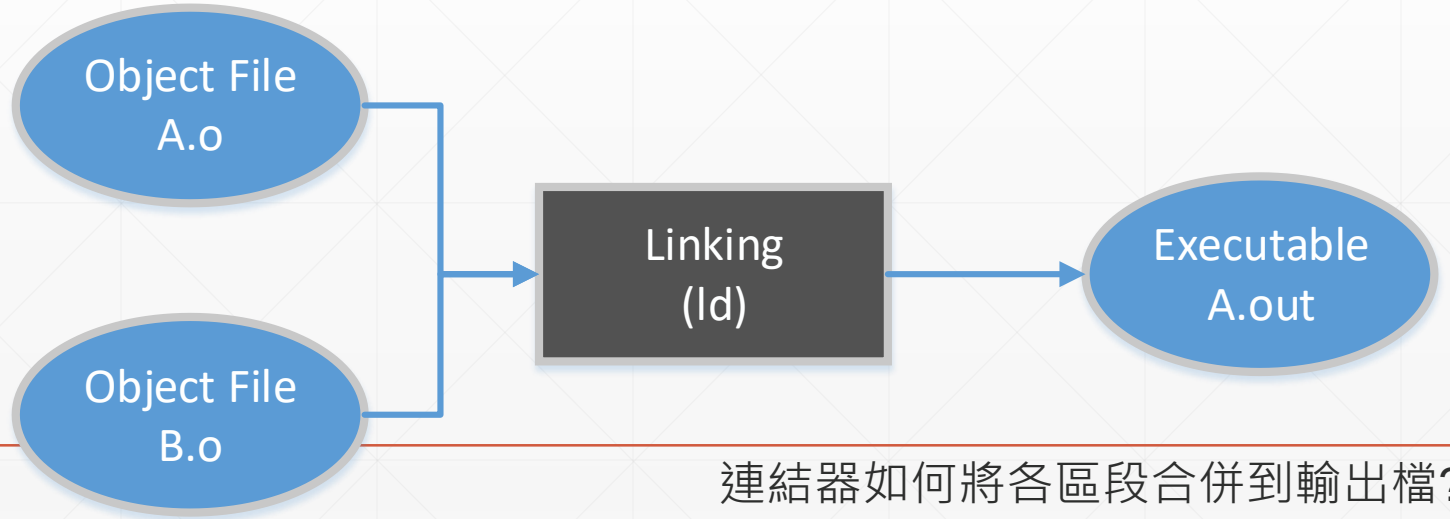
a.c 定義：

一個全域符號：main()

引用：b.c 的shared、func()

b.c 定義：

兩個全域符號：shared、func()



連結器如何將各區段合併到輸出檔??

Embedded Software Porting

- 靜態連結

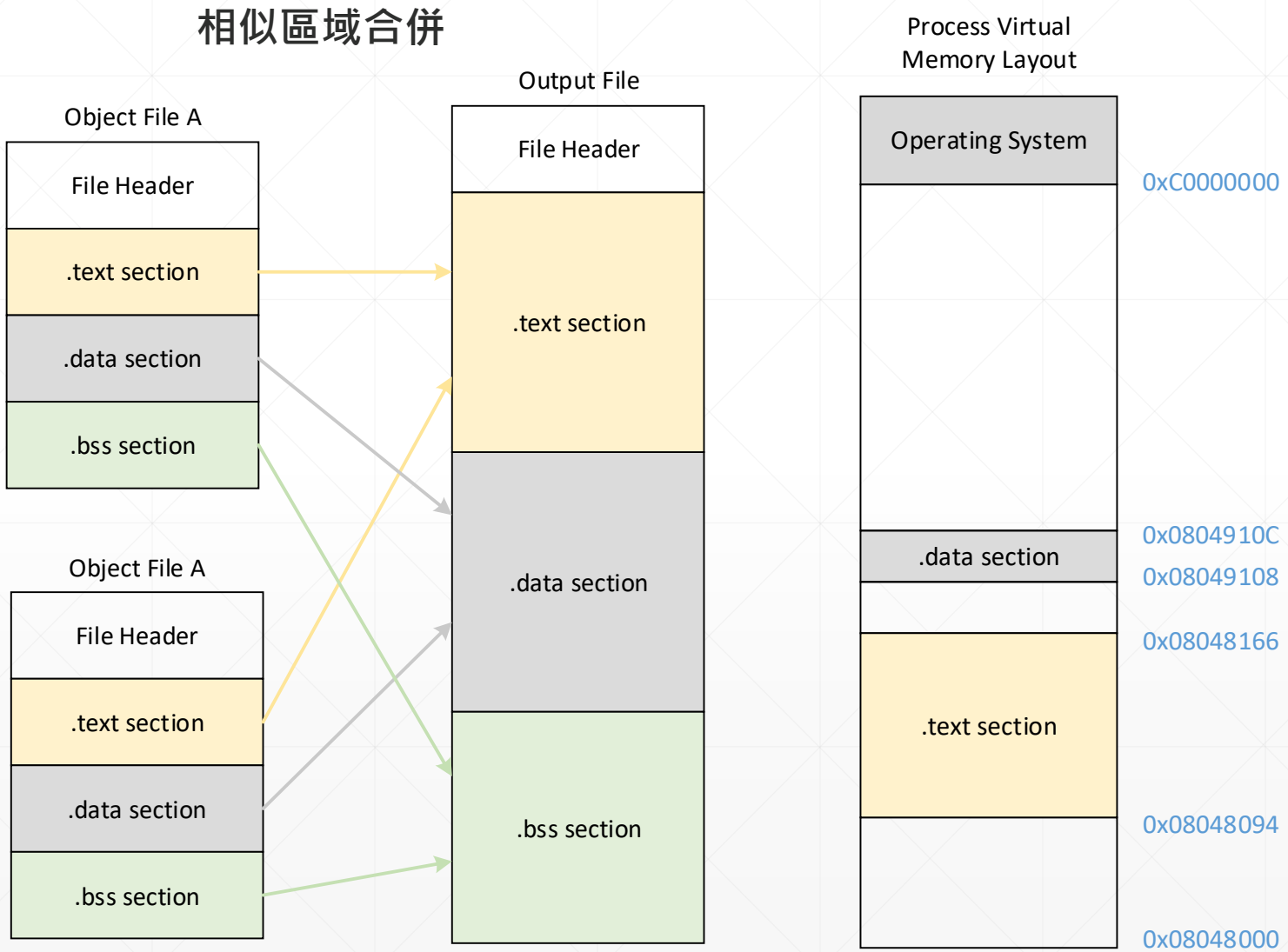
兩步連結(Two Step Linking)：

第一步：**空間與位置分配**

合併與統計輸入目標檔各區段長度與位置，並建立對應關係。

第二步：**符號解析與重定**

進行符號解析與重定，調整程式碼中的記憶體位置。



Embedded Software Porting

- **靜態函式庫** (一組物件檔的集合)

- Linux C 靜態函式庫 (libc.a) :

  輸出入、檔案操作、時間日期、記憶體管理...等等

  例如：

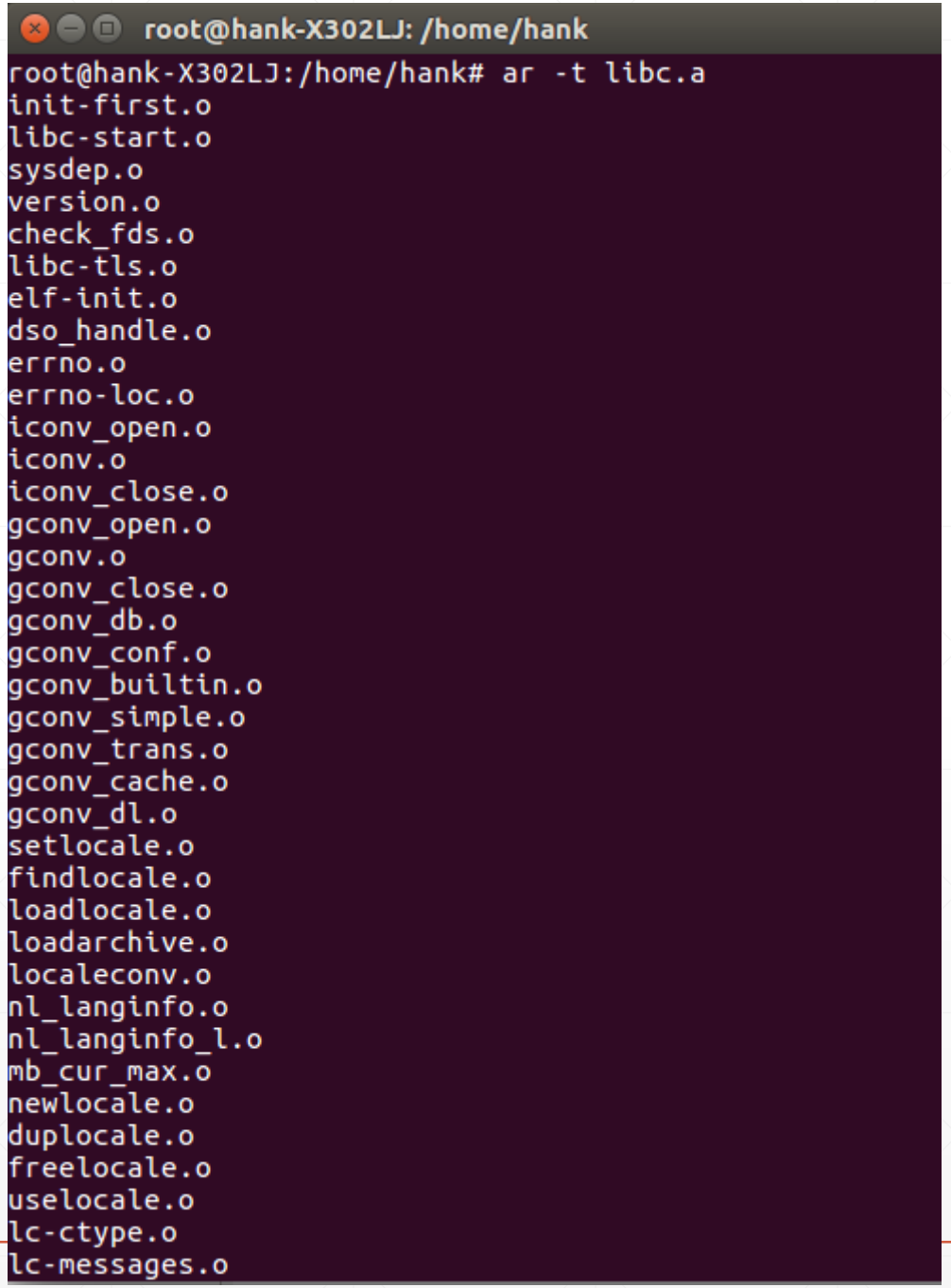
  輸出入：printf.o、scanf.o

  檔案操作：fread.o、fwrite.o

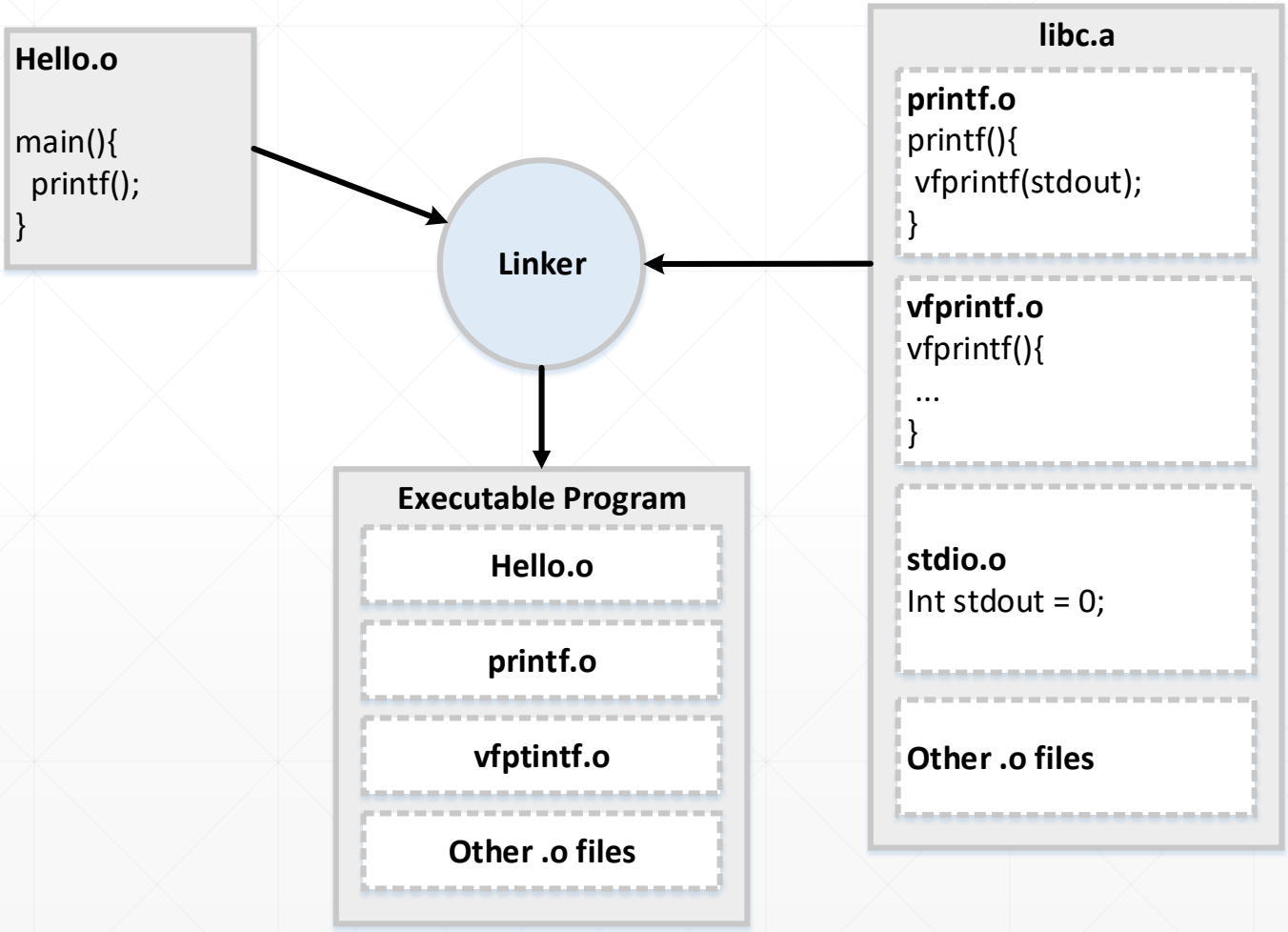
  時間日期：date.o、time.o

  記憶體管理：malloc.o

- /usr/lib32/libc.a
- # ar –t libc.a



Embedded Software Porting



Embedded Software Porting

- 動態連結
  - 將程式按造模組分割成數個獨立的部分，當程式要執行的時候，才連結成一個完整的程式。
  - 工具為**動態連結器**

- Linux：ELF動態連接檔稱為動態共用物件(Dynamic Shared Objects)，副檔名：".so"
- Windows：動態連接程式庫(Dynamic Linking Library)，附檔名：".dll"

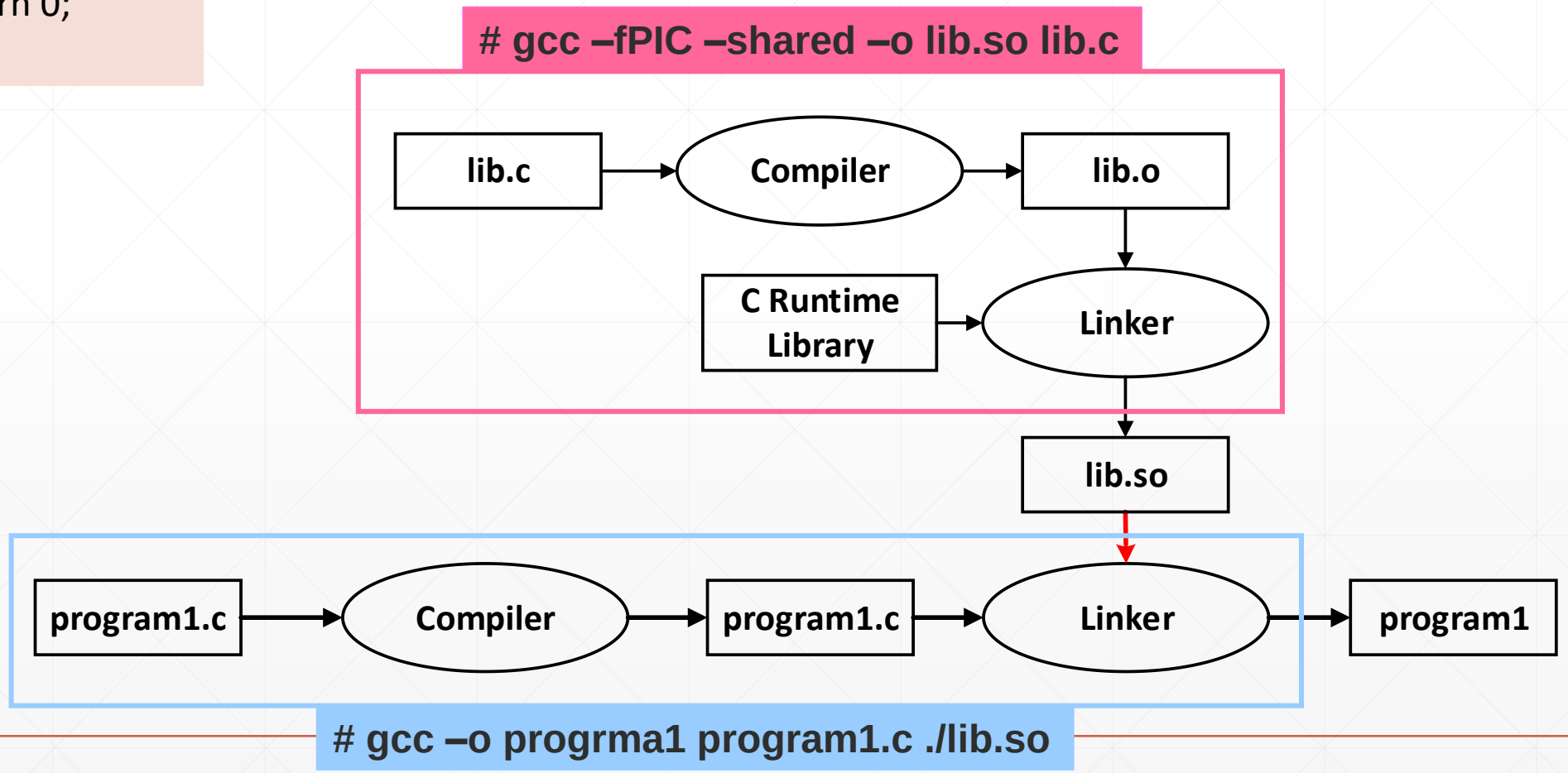
從作業系統看可執行檔的載入：
1. 建立一個獨立的虛擬位置空間
2. 讀取可執行檔的檔頭，並建立虛擬空間與可執行檔的對映關係
3. 將CPU的指令暫存器設定成可執行檔的入口位置，啟動執行

老爺！等等！ 還有外部符號的位置不知道的，這樣下去會爆掉!

```c
/* program1.c */
#include "lib.h"

int main()
{
	func(1);
	return 0;
}
```

```c
/* lib.h */
#ifndef LIB_H
#define LIB_H
void func(int i);
#endif
```

```c
/* lib.c */
#include <stdio.h>

void func(int i)
{
	printf("Printing from lib.so %d\n",i);
}
```



Embedded Software Porting

TAIPEI TECH

```
root@hank-X302LJ:/home/hank# readelf -l lib.so

Elf file type is DYN (Shared object file)
Entry point 0x5a0
There are 7 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x000000000000076c 0x000000000000076c  R E    200000
  LOAD           0x0000000000000e00 0x0000000000200e00 0x0000000000200e00
                 0x0000000000000228 0x0000000000000230  RW     200000
  DYNAMIC        0x0000000000000e18 0x0000000000200e18 0x0000000000200e18
                 0x00000000000001c0 0x00000000000001c0  RW     8
  NOTE           0x00000000000001c8 0x00000000000001c8 0x00000000000001c8
                 0x0000000000000024 0x0000000000000024  R      4
  GNU_EH_FRAME   0x00000000000006e8 0x00000000000006e8 0x00000000000006e8
                 0x000000000000001c 0x000000000000001c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e00 0x0000000000200e00 0x0000000000200e00
                 0x0000000000000200 0x0000000000000200  R      1

 Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text
.fini .rodata .eh_frame_hdr .eh_frame
   01     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   02     .dynamic
   03     .note.gnu.build-id
   04     .eh_frame_hdr
   05
   06     .init_array .fini_array .jcr .dynamic .got
```

Embedded Software Porting

- 動態連結相關結構：
- .interp：存放可執行檔的動態連結器的路徑

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# readelf -l program1 | grep interpreter
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
root@hank-X302LJ:/home/hank#
```

```
root@hank-X302LJ:/home/hank# objdump -s program1

program1:     file format elf64-x86-64

Contents of section .interp:
 400238 2f6c6962 36342f6c 642d6c69 6e75782d  /lib64/ld-linux-
 400248 7838362d 36342e73 6f2e3200           x86-64.so.2.
```

- .dynamic：存放動態連結器所需要的資訊

```
root@hank-X302LJ:/home/hank# readelf -d lib.so

Dynamic section at offset 0xe18 contains 24 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x548
 0x000000000000000d (FINI)               0x6c4
 0x0000000000000019 (INIT_ARRAY)         0x200e00
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x200e08
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x1f0
 0x0000000000000005 (STRTAB)             0x380
 0x0000000000000006 (SYMTAB)             0x230
 0x000000000000000a (STRSZ)              175 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000003 (PLTGOT)             0x201000
 0x0000000000000002 (PLTRELSZ)           24 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x530
 0x0000000000000007 (RELA)               0x470
 0x0000000000000008 (RELASZ)             192 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffffe (VERNEED)            0x450
 0x000000006fffffff (VERNEEDNUM)         1
 0x000000006ffffff0 (VERSYM)             0x430
 0x000000006ffffff9 (RELACOUNT)          3
 0x0000000000000000 (NULL)               0x0
```

Embedded Software Porting

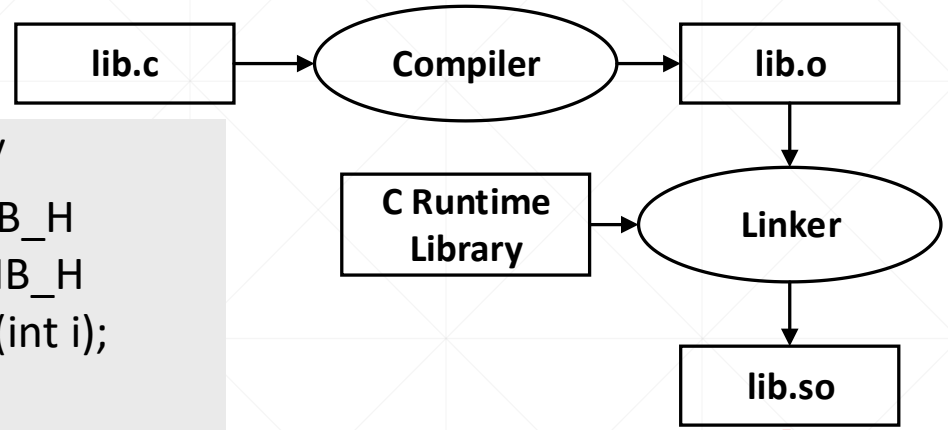
- .symtab (Symbol Table)

```
root@hank-X302LJ:/home/hank# readelf -s lib.so

Symbol table '.dynsym' contains 14 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000548     0 SECTION LOCAL  DEFAULT    9
     2: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.2.5 (2)
     4: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     5: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
     6: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
     7: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.2.5 (2)
     8: 0000000000201028     0 NOTYPE  GLOBAL DEFAULT   23 _edata
     9: 0000000000201030     0 NOTYPE  GLOBAL DEFAULT   24 _end
    10: 0000000000201028     0 NOTYPE  GLOBAL DEFAULT   24 __bss_start
    11: 0000000000000548     0 FUNC    GLOBAL DEFAULT    9 _init
    12: 00000000000006c4     0 FUNC    GLOBAL DEFAULT   13 _fini
    13: 00000000000006a0    36 FUNC    GLOBAL DEFAULT   12 func
```

Embedded Software Porting

```
/* lib.c */
#include <stdio.h>

void func(int i)
{
	printf("Printing from lib.so %d\n",i);
}
```

```
/* lib.h */
#ifndef LIB_H
#define LIB_H
void func(int i);
#endif
```




```
root@hank-X302LJ:/home/hank# ldd lib.so
	linux-vdso.so.1 =>  (0x00007ffe85598000)
	libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f856b24b000)
	/lib64/ld-linux-x86-64.so.2 (0x00007f856b817000)
```

```
/* program1.c */
#include "lib.h"

int main()
{
	func(1);
	return 0;
}
```


```
root@hank-X302LJ:/home/hank# ldd program1
	linux-vdso.so.1 =>  (0x00007ffffe678000)
	./lib.so (0x00007f3641f4a000)
	libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3641b80000)
	/lib64/ld-linux-x86-64.so.2 (0x00007f364214c000)
```

Embedded Software Porting

- **Tool chain**

  Set of applications used to perform a complex task or to create a product, which is typically another computer program or a system of programs

- The targets some times might have the following restrictions too!
  - Less memory footprints
  - Slow compared to host
  - You may not wish to have in a developed system finally

- So cross compiling toolchain are used

Embedded Software Porting

- **Tool chain**

It consists of a compiler, linker, assembler, and a debugger.

The GNU tool chain is a programming tools produced by the GNU Project. The GNU tool chain plays a vital role in development of software for embedded systems.

- GNU make : Build and compilation automation
- GNU Compilers: gcc, the well known C,C++ complier supported variable platforms
- GNU Binutils including: Assembler (as), linker (ld) and other binary file tools
- GNU Debugers : gdb, the command line interactive debugging, including remote debugging
- GNU C Libraries: glibc, uclibc

Embedded Software Porting

- GCC Compiler Command Line Options

C ➔ gcc compiler

```
#include<stdio.h>

int main(void)
{
   printf("Hello CC ARM\n");
   return 0;
}
```

gcc main.c → C → a.out

- Specify the Output Executable Name

Use option **-o** to specify the output file name for the executable.

gcc main.c -o main → C → main

Embedded Software Porting

- Enable all warnings set through **-Wall** option

```c
#include<stdio.h>

int main(void)
{
    int i;
    printf("Hello CC ARM Stuff [%d] \n", i);
    return 0;
}
```

gcc -Wall main.c -o main

C →

main.c: In function main:
main.c:6:10: warning: i is used uninitialized in this function [-Wuninitialized]

Embedded Software Porting

- Produce only the preprocessor output with -E option

- Produce only the assembly code using -S option

- Produce only the compiled code (without any linking) using the -C option

- Produce all the intermediate files using -save-temps function

- Pre-processing
- Compilation
- Assembly
- Linking

實際上gcc只是後台程式的包裝，根據不同參數要求呼叫前編譯器1、組譯器as、連結器ld

Embedded Software Porting

- Link with shared libraries using -I option

```
gcc  -Wall main.c -o main -IDynamic
```

main

linking with the shared library libDynamic.so

- Create position independent code using -fPIC option

```
gcc -c -Wall -fPIC myFunc.c
gcc -shared -o libmyFunc.so myFunc.o
```

libmyFunc.so

Embedded Software Porting

- Use compile time macros using -D option

```c
#include<stdio.h>

int main(void)
{
#ifdef MY_MACRO
  printf("\n Macro defined \n");
#endif
    char c = -10;
    // Print the string
    printf("\n The CC ARM Stuff [%d]\n", c);
    return 0;
}
```

gcc -Wall -DMY_MACRO main.c -o main

main

```
./main

Macro defined

The CC ARM Stuff [-10]
```

Embedded Software Porting

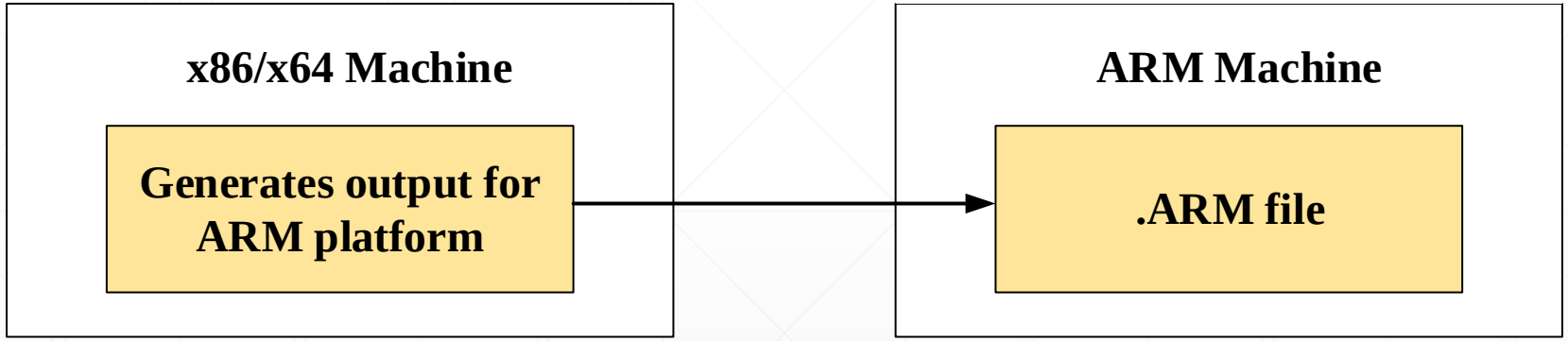
- This option adds a path to find headers files.

```
gcc -I/home/codeman/include main.c
```

- The options for C/C++ standard : c++11, c++14, c90, c89.

```
gcc –std=c90 main.c
g++ -std=c++11 main.cpp
```

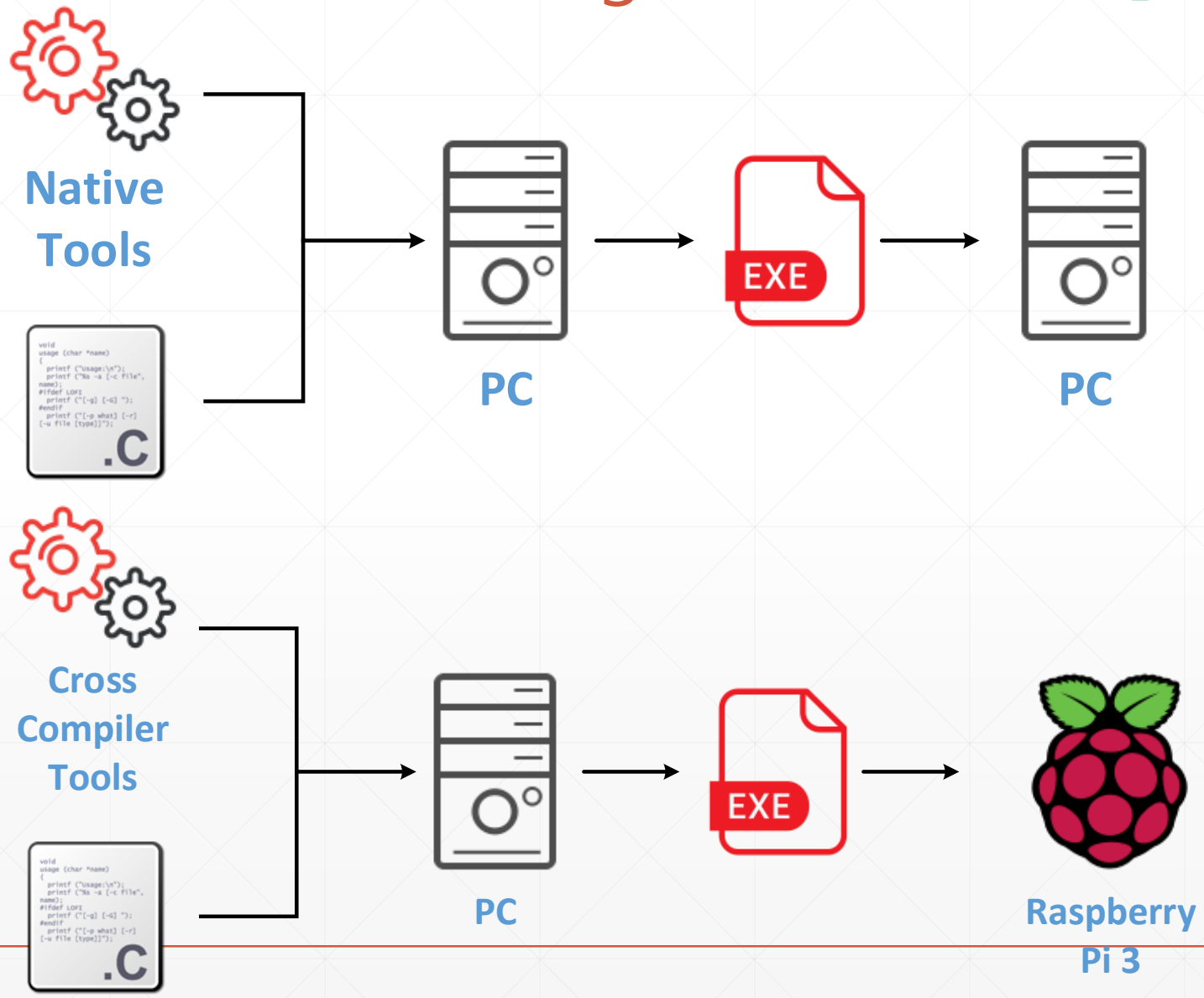
Embedded Software Porting



The host and target architectures are different, the toolchain is called a cross compiler

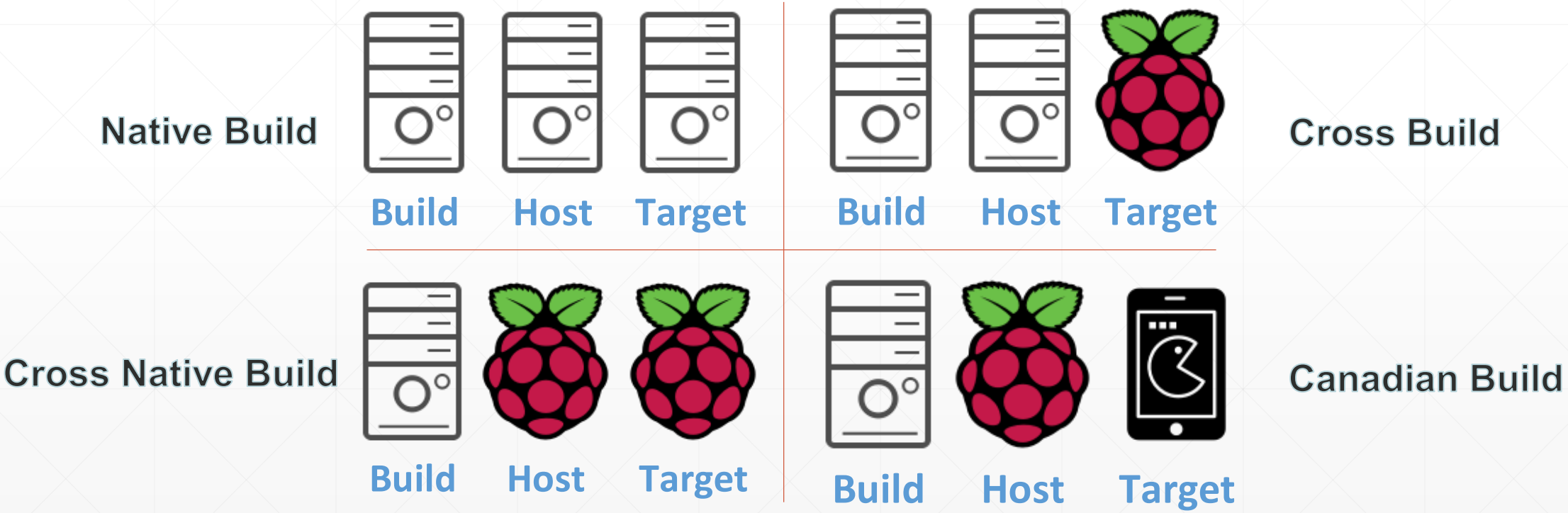
If we develop a code on a Linux machine based on the x64 architecture, but we're compiling for an ARM target, then we need Linux-based ARM-targeting cross compiler

Embedded Software Porting



Embedded Software Porting

- **Build** – which is used to create the toolchain
- **Host** – which will be used to execute the created toolchain
- **Target** – which will execute the binaries created by the toolchain



Embedded Software Porting

- Before building the toolchain following decisions have to be made

  Which library to be used?

  What version of the components to selected?

  Certain important configurations like

  - Architecture features like floating point
  - ABI selections
  - Networking features etc.,

- So you might have to put good amount of time in investigations

Linaro tool chain : https://www.linaro.org/

arm-linux-gnueabihf	*32-bit Armv7 Cortex-A, hard-float, little-endian*
armv8l-linux-gnueabihf	*32-bit Armv8 Cortex-A, hard-float, little-endian*
aarch64-linux-gnu	*64-bit Armv8 Cortex-A, little-endian*

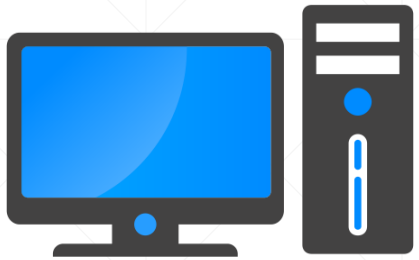
Embedded Software Porting

- Install Cross Compiler on PC
  - Linux server : https://launchpad.net/ubuntu/+archivemirrors

    sudo apt-get install gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf

    sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi

  - **Linaro tool chain : https://www.linaro.org**



x64/x86 Machine

- Terminal :

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# arm
arm2hpdl                        arm-linux-gnueabi-gprof          arm-linux-gnueabihf-gprof
arm-linux-gnueabi-addr2line     arm-linux-gnueabihf-addr2line    arm-linux-gnueabihf-ld
arm-linux-gnueabi-ar            arm-linux-gnueabihf-ar           arm-linux-gnueabihf-ld.bfd
arm-linux-gnueabi-as            arm-linux-gnueabihf-as           arm-linux-gnueabihf-ld.gold
arm-linux-gnueabi-c++filt       arm-linux-gnueabihf-c++filt      arm-linux-gnueabihf-nm
arm-linux-gnueabi-cpp           arm-linux-gnueabihf-cpp          arm-linux-gnueabihf-objcopy
arm-linux-gnueabi-cpp-5         arm-linux-gnueabihf-cpp-5        arm-linux-gnueabihf-objdump
arm-linux-gnueabi-dwp           arm-linux-gnueabihf-dwp          arm-linux-gnueabihf-ranlib
arm-linux-gnueabi-elfedit       arm-linux-gnueabihf-elfedit      arm-linux-gnueabihf-readelf
arm-linux-gnueabi-g++           arm-linux-gnueabihf-g++          arm-linux-gnueabihf-size
arm-linux-gnueabi-g++-5         arm-linux-gnueabihf-g++-5        arm-linux-gnueabihf-strings
arm-linux-gnueabi-gcc           arm-linux-gnueabihf-gcc          arm-linux-gnueabihf-strip
arm-linux-gnueabi-gcc-5         arm-linux-gnueabihf-gcc-5        arm-linux-gnueabi-ld
arm-linux-gnueabi-gcc-ar        arm-linux-gnueabihf-gcc-ar       arm-linux-gnueabi-ld.bfd
arm-linux-gnueabi-gcc-ar-5      arm-linux-gnueabihf-gcc-ar-5     arm-linux-gnueabi-ld.gold
arm-linux-gnueabi-gcc-nm        arm-linux-gnueabihf-gcc-nm       arm-linux-gnueabi-nm
arm-linux-gnueabi-gcc-nm-5      arm-linux-gnueabihf-gcc-nm-5     arm-linux-gnueabi-objcopy
arm-linux-gnueabi-gcc-ranlib    arm-linux-gnueabihf-gcc-ranlib   arm-linux-gnueabi-objdump
arm-linux-gnueabi-gcc-ranlib-5  arm-linux-gnueabihf-gcc-ranlib-5 arm-linux-gnueabi-ranlib
arm-linux-gnueabi-gcov          arm-linux-gnueabihf-gcov         arm-linux-gnueabi-readelf
arm-linux-gnueabi-gcov-5        arm-linux-gnueabihf-gcov-5       arm-linux-gnueabi-size
arm-linux-gnueabi-gcov-tool     arm-linux-gnueabihf-gcov-tool    arm-linux-gnueabi-strings
arm-linux-gnueabi-gcov-tool-5   arm-linux-gnueabihf-gcov-tool-5  arm-linux-gnueabi-strip
root@hank-X302LJ:/home/hank# arm
```

Embedded Software Porting

- **arch[-vendor][-os]-abiarch** - architecture
  arm, mips, x86, i686, etc.

  vendor - tool chain supplier

  os - operating system
    linux, none (bare metal)

  abi - application binary interface
    eabi, gnueabi, gnueabihf

eabi (*embedded-application binary interface*) :

standard conventions for file formats, data types, register usage, stack frame organization.

gnueabi / gnueabihf ➔ gcc “-mfloat-abi” (1). soft (2). softfp (3). hard

(armel) / (armhf)

Embedded Software Porting

- **arm-none-eabi**

  This tool chain targets the ARM architecture, has no vendor, does not target an operating system (i.e. targets a “bare metal” system), and complies with the ARM EABI.

- **arm-none-linux-gnueabi**

  This tool chain targets the ARM architecture, has no vendor, creates binaries that run on the Linux operating system, and uses the GNU EABI.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello Embedded System !" << endl;
    return 0;
}
```

PC / Linux

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# g++ -o Hello.exe Hello.cpp
root@hank-X302LJ:/home/hank#
```

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# g++ -o Hello.exe Hello.cpp
root@hank-X302LJ:/home/hank# file Hello.exe
Hello.exe: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically link
ed, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=dc812f6c47d6f931a5ecbc6e0a2b8699b7dcb26c, not stripped
root@hank-X302LJ:/home/hank#
```

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# g++ -o Hello.exe Hello.cpp
root@hank-X302LJ:/home/hank# file Hello.exe
Hello.exe: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically link
ed, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=dc812f6c47d6f931a5ecbc6e0a2b8699b7dcb26c, not stripped
root@hank-X302LJ:/home/hank# ./Hello.exe
Hello Embedded System !
root@hank-X302LJ:/home/hank#
```

Embedded / Raspberry pi

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
root@raspberrypi:/home/pi# arm-linux-gnueabihf-g++ -o Hello.exe Hello.cpp
root@raspberrypi:/home/pi#
```

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
root@raspberrypi:/home/pi# arm-linux-gnueabihf-g++ -o Hello.exe Hello.cpp
root@raspberrypi:/home/pi# file Hello.exe
Hello.exe: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically l
inked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=
214fc655d8fb87c22b3bac4130ed1cb571fafec2, not stripped
root@raspberrypi:/home/pi#
```

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
root@raspberrypi:/home/pi# arm-linux-gnueabihf-g++ -o Hello.exe Hello.cpp
root@raspberrypi:/home/pi# file Hello.exe
Hello.exe: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically l
inked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=
214fc655d8fb87c22b3bac4130ed1cb571fafec2, not stripped
root@raspberrypi:/home/pi# ./Hello.exe
Hello Embedded System !
root@raspberrypi:/home/pi#
```

Embedded Software Porting

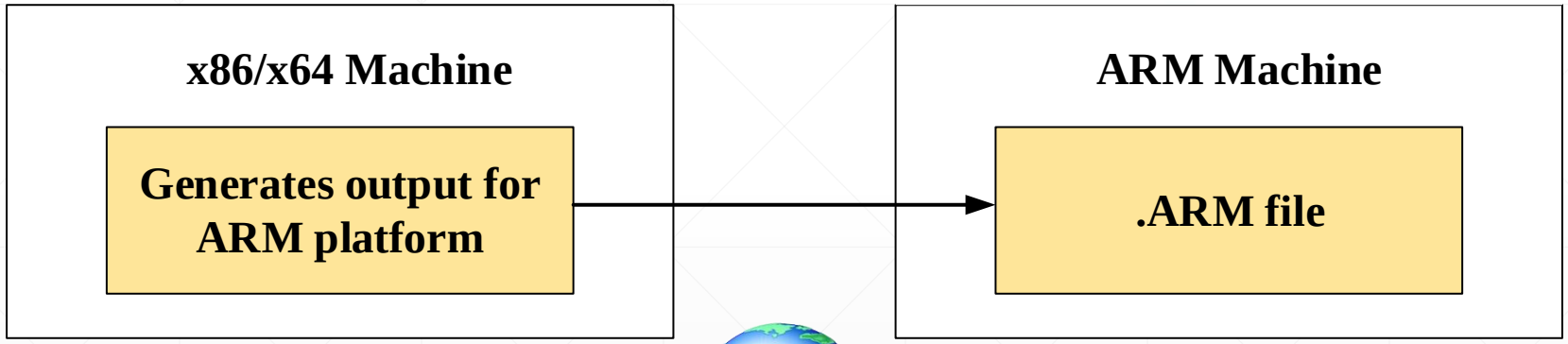
- Cross Compiler

x64/x86 Machine

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# arm-linux-gnueabi-g++ -o Hello_CC_ARM.exe Hello.cpp

root@hank-X302LJ:/home/hank#
```

```
root@hank-X302LJ: /home/hank
root@hank-X302LJ:/home/hank# arm-linux-gnueabi-g++ -o Hello_CC_ARM.exe Hello.cpp

root@hank-X302LJ:/home/hank# file Hello_CC_ARM.exe
Hello_CC_ARM.exe: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynami
cally linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 3.2.0, BuildID[sha1]
=6ce8b87c5c020bf04925a9b2ced55ced873e72d3, not stripped
root@hank-X302LJ:/home/hank#
```

Embedded Software Porting



scp Hello_CC_Arm.exe root@[your_board_ip]:/root

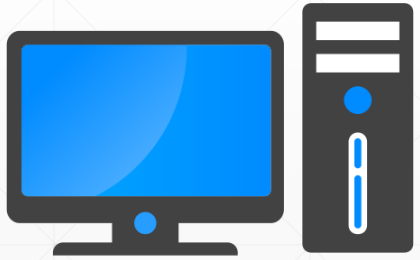
Embedded Software Porting

- Developing GUI for your Embedded target

On Target (Raspberry Pi)

O : Convenience

X : Limited hardware resources

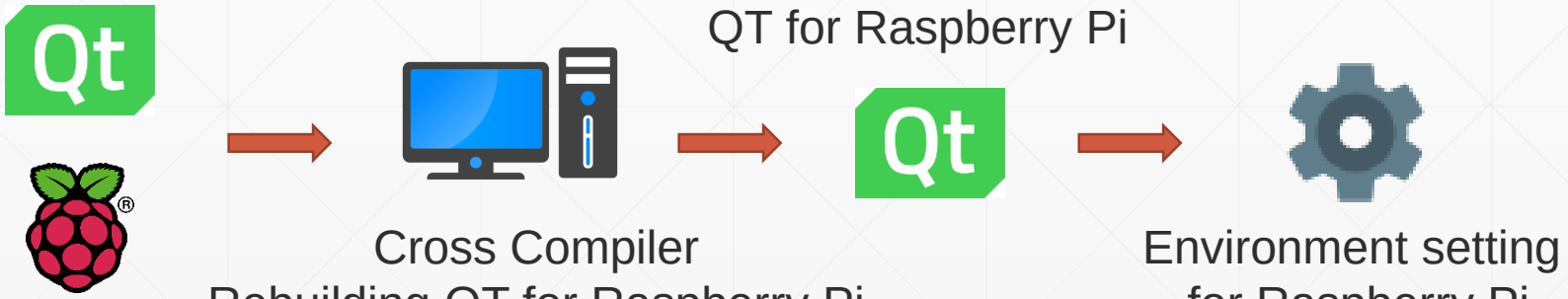


x64/x86 Machine

On PC Machine

O : Powerful hardware

X : Complex setup process for cross compiler



Embedded Software Porting

- Developing GUI for your Embedded target

https://www.raspberrypi.org/forums/viewtopic.php?f=75&t=204778

https://wiki.qt.io/RaspberryPi2EGLFS