Introduction to Computers and Programming

Lecture 13- Chap 22 File I/O Epilogue

Tien-Fu Chen

Dept. of Computer Science and Information Engineering

National Yang Ming Chiao Tung Univ.

File I/O

Stream and File Pointers

- A ***stream*** means any source of input or any destination for output.
- Accessing a stream is done through a ***file pointer,*** which has type `FILE *`.
- The `FILE` type is declared in `<stdio.h>`.
- Certain streams are represented by file pointers with standard names.
- Additional file pointers can be declared as needed:

```
FILE *fp1, *fp2;
```

Standard Streams and Redirection

- `<stdio.h>` provides three standard streams:

File Pointer	*Stream*	*Default Meaning*
`stdin`	Standard input	Keyboard
`stdout`	Standard output	Screen
`stderr`	Standard error	Screen

- ***input redirection:*** forcing a program to obtain its input from a file instead of from the keyboard:

```
demo <in.dat
```

- ***Output redirection*** :

```
demo >out.dat
```

All data written to `stdout` will go into the `out.dat` file instead of appearing on the screen.

Standard Streams and Redirection

- Input redirection and output redirection can be combined:

```
demo <in.dat >out.dat
```

- The < and > characters don’t have to be adjacent to file names, and the order in which the redirected files are listed doesn’t matter:

```
demo < in.dat > out.dat
demo >out.dat <in.dat
```

Text Files versus Binary Files

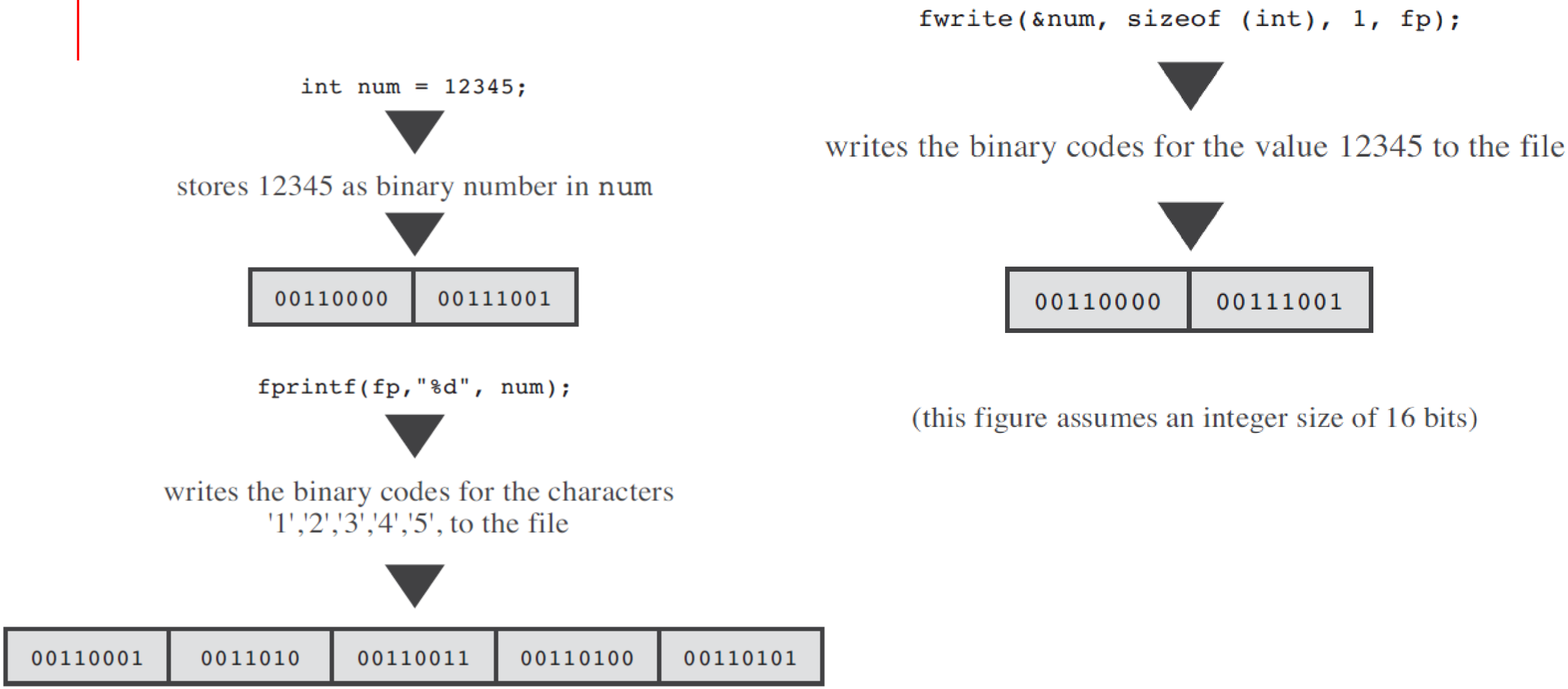
- `<stdio.h>` supports two kinds of files: text and binary.
- The bytes in a ***text file*** represent characters.
  - The source code for a C program is stored in a text file.

00110011	00110010	00110111	00110110	00110111
'3'	'2'	'7'	'6'	'7'

- A ***binary file,*** bytes don't necessarily represent characters.
  - Groups of bytes might represent other types of data, such as integers and floating-point numbers.
  - An executable C program is stored in a binary file.

01111111	11111111

Binary and text output



```
int num = 12345;
```

stores 12345 as binary number in num

00110000	00111001

```
fprintf(fp,"%d", num);
```

writes the binary codes for the characters
'1','2','3','4','5', to the file

00110001	0011010	00110011	00110100	00110101

```
fwrite(&num, sizeof (int), 1, fp);
```

writes the binary codes for the value 12345 to the file

00110000	00111001

(this figure assumes an integer size of 16 bits)

Opening a File

- Opening a file for use as a stream requires a call of the `fopen` function.
- Prototype for `fopen`:

```
FILE *fopen(const char * restrict filename,
            const char * restrict mode);
```

- `filename` is the name of the file to be opened.
  - Include information about the file's location, such as path.
- `mode` is a "mode string" that specifies what operations we intend to perform on the file.
- `fopen` returns a file pointer to be saved in a variable:

```
fp = fopen("in.dat", "r");
  /* opens in.dat for reading */
```

Modes

- Mode strings for text files:

String	***Meaning***
`"r"`	Open for reading
`"w"`	Open for writing (file need not exist)
`"a"`	Open for appending (file need not exist)
`"r+"`	Open for reading and writing, starting at beginning
`"w+"`	Open for reading and writing (truncate if file exists)
`"a+"`	Open for reading and writing (append if file exists)

Modes

- Mode strings for binary files:

String	*Meaning*
`"rb"`	Open for reading
`"wb"`	Open for writing (file need not exist)
`"ab“`	Open for appending (file need not exist)
`"r+b"` or `"rb+"`	Open for reading and writing, starting at beginning
`"w+b"` or `"wb+"`	Open for reading and writing (truncate if file exists)
`"a+b"` or `"ab+"`	Open for reading and writing (append if file exists)

Reading and Closing a File

- The outline of a program that opens a file for reading:

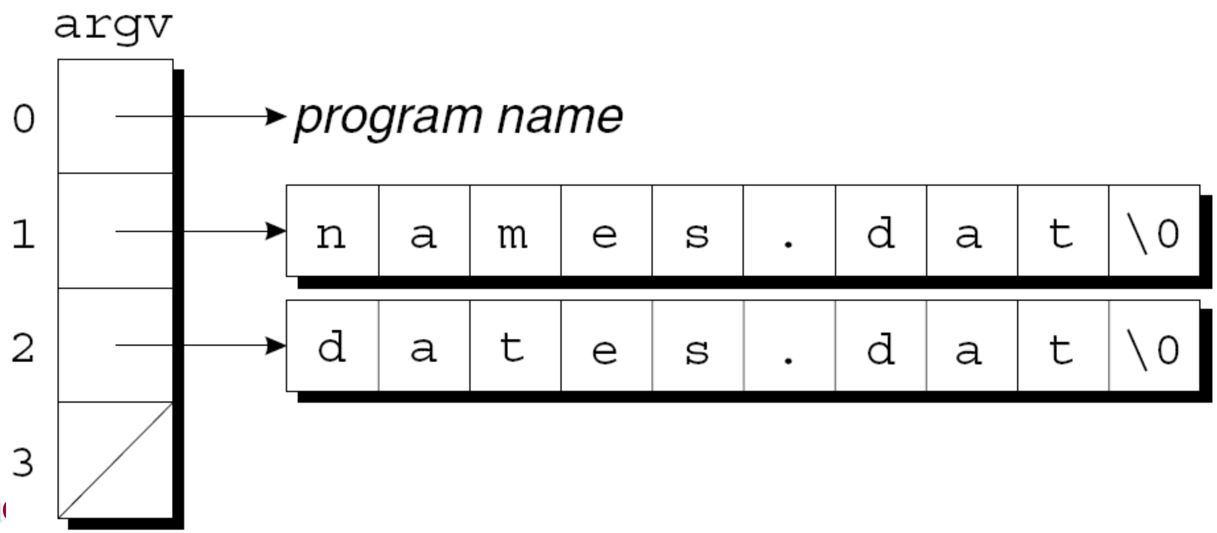
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  FILE *fp;

  fp = fopen("example.dat", "r");
  if (fp == NULL) {
    printf("Can't open %s\n", FILE_NAME);
    exit(EXIT_FAILURE);
  }
  …
   fread(ptr, size, cnt, fp);
   fclose(fp);
  return 0;
}
```

Obtaining File Names from the Command Line

- Access command-line arguments by defining `main` as a function with two parameters:

```
int main(int argc, char *argv[])
{
   …
}
```

- `argc` is the number of command-line arguments.
- `argv` is an array of pointers to the argument strings.



File Buffering

- Data written to a stream is stored in a **buffer** area in memory; when full (or is closed), the buffer is "flushed."
- A call that flushes the buffer for the file associated with `fp`:

  ```
  fflush(fp);    /* flushes buffer for fp */
  ```

- A call that flushes *all* output streams:

  ```
  fflush(NULL);  /* flushes all buffers */
  ```

- `fflush` returns zero if it's successful and `EOF` if an error occurs.

Formatted I/O

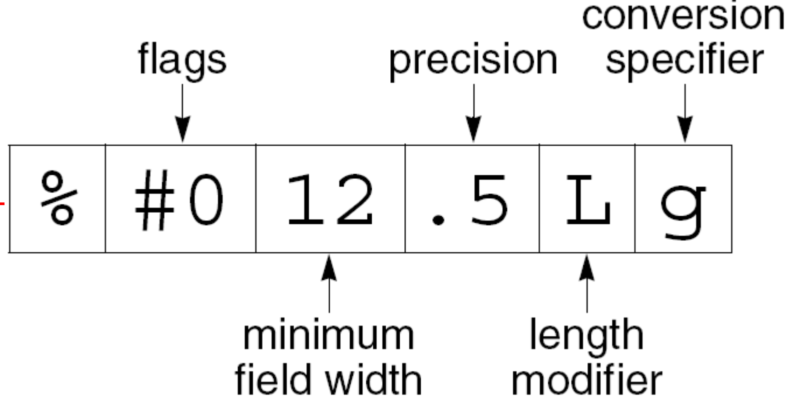
- `fprintf` and `printf` functions write variables to an output stream, using a format control.
- Both functions end with the `...` symbol, which indicates a variable number of additional arguments:

```
int fprintf(FILE * restrict stream,
            const char * restrict format, ...);
int printf(const char * restrict format, ...);
```

- `printf` always writes to `stdout`:

```
printf("Total: %d\n", total);
  /* writes to stdout */
fprintf(fp, "Total: %d\n", total);
  /* writes to fp */
```

...`printf`Specifications



flags

precision

conversion specifier

%

#0

12

.5

L

g

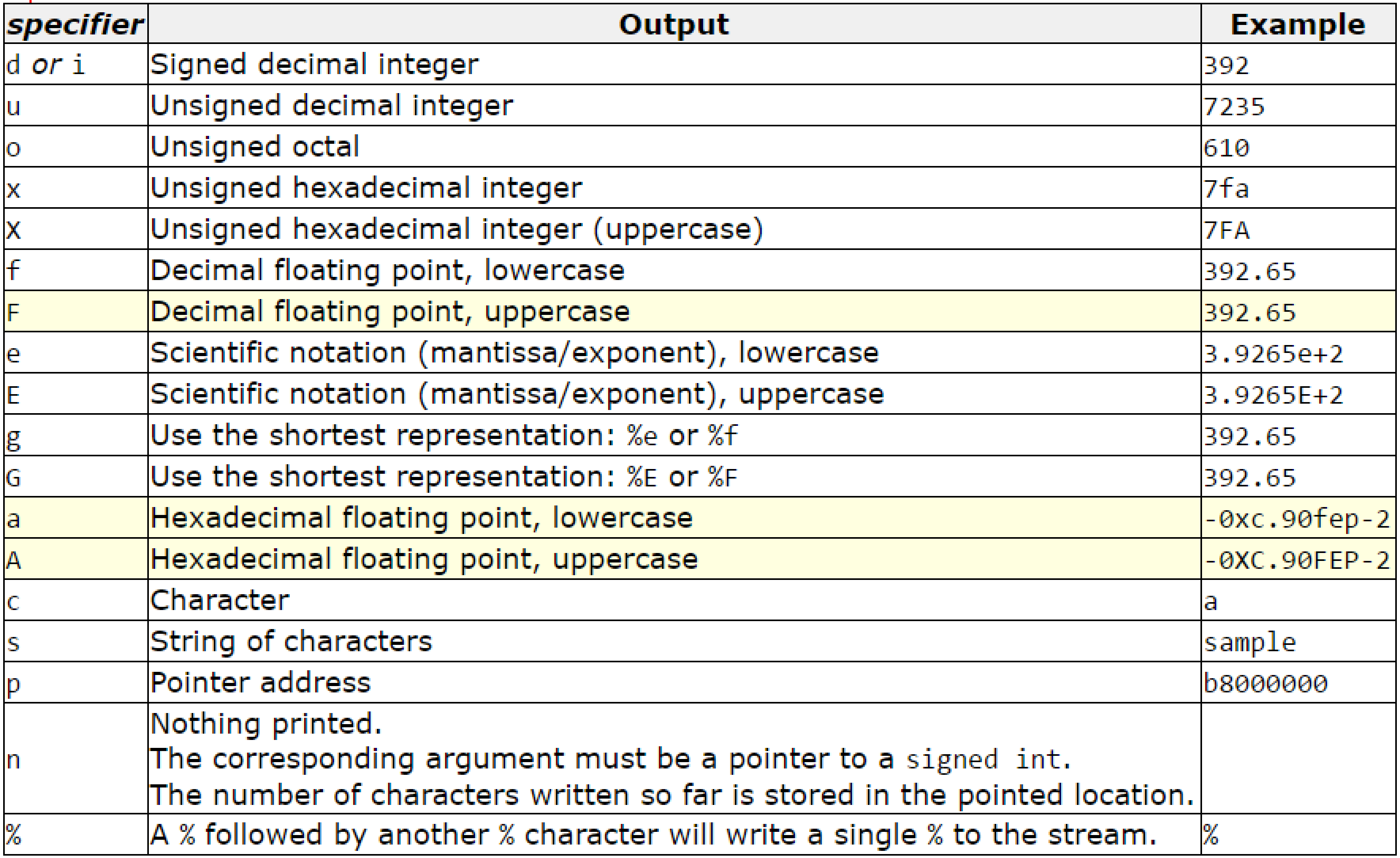
minimum field width

length modifier

General form: % [**flags**] [width] [.precision] [{ h | l | I64 | L }] type

flag	meaning	default
-	Left align the result within the given field *width*	Right align
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type.	Sign appears only for negative signed values (-)
0	If *width* is prefixed with 0, zeros are added until the minimum *width* is reached. If 0 and - appear, the 0 is ignored.	No padding (actually space padding)
blank	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears
#	When used with the o, x, or X format, the # flag prefixes any nonzero output with 0, 0x, or 0X, respectively. Ignored when used with c, d, i, u, or s.	No prefix
#	When used with the e, E, or f format, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.
#	When used with g or G format, forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros.	Decimal point appears only if digits follow it. Trailing zeros are truncated.

...`printf` type Specifications



specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

The …`scanf` Functions

- `scanf` always reads from `stdin`, whereas `fscanf` reads from the stream indicated by its first argument:

```
scanf("%d%d", &i, &j);
  /* reads from stdin */
fscanf(fp, "%d%d", &i, &j);
  /* reads from fp */
```

- A call of `scanf` is equivalent to a call of `fscanf` with `stdin` as the first argument.

Block I/O

- The `fread` and `fwrite` functions allow a program to read and write large blocks of data.
- Arguments in a call of `fwrite`:
  - Address of array
  - Size of each array element (in bytes)
  - Number of elements to write
  - File pointer
- A call of `fwrite` that writes the entire contents of the array `a`:

```
fwrite(a, sizeof(a[0]),
          sizeof(a) / sizeof(a[0]), fp);
```

Block I/O

- `fread` will read the elements of an array from a stream.
- A call of `fread` that reads the contents of a file into the array `a`:

```
n = fread(a, sizeof(a[0]),
          sizeof(a) / sizeof(a[0]), fp);
```

- `fread`'s return value indicates the actual number of elements read.

File Positioning

- The `fseek` function changes the file position associated with the first argument (a file pointer).
- int fseek(FILE *stream, long int offset, int whence)
- The third argument is one of three macros:

`SEEK_SET`	Beginning of file
`SEEK_CUR`	Current file position
`SEEK_END`	End of file

- The second argument, which has type `long int`, is a (possibly negative) byte count.

File Positioning

- Using `fseek` to move to the beginning of a file:

```
fseek(fp, 0L, SEEK_SET);
```

- Using `fseek` to move to the end of a file:

```
fseek(fp, 0L, SEEK_END);
```

- Using `fseek` to move back 10 bytes:

```
fseek(fp, -10L, SEEK_CUR);
```

- If an error occurs (the requested position doesn’t exist, for example), `fseek` returns a nonzero value.

Print and scan I/O with string

- `sprintf` and `snprintf` write characters into a string.
- `sprintf` function writes output into a character array (pointed to by its first argument) instead of a stream.
- A call that writes `"9/20/2010"` into `date`:

```
sprintf(date, "%d/%d/%d", 9, 20, 2010);
```

Input Functions from string

- `sscanf` is handy for extracting data from a string that was read by another input function.

  int sscanf (const char *str, const char *format, ...);

- An example that uses `fgets` to obtain a line of input, then passes the line to `sscanf` for further processing:

```
fgets(str, sizeof(str), stdin);
  /* reads a line of input */
sscanf(str, "%d%d", &i, &j);
  /* extracts two integers */
```

Find execution time of a C program

- ❑ <sys/time.h> header file

```
struct timeval {
    long tv_sec;    /* seconds */
    long tv_usec;   /* microsec */
};
struct rusage {
   struct timeval ru_utime
   struct timeval ru_stime
…
}
```

```
#include <sys/time.h>
#include <sys/resource.h>

int main (int argc, char *argv[])
{
   struct rusage start;
   struct rusage end;

   getrusage (RUSAGE_SELF, &start);    // get time at start
   some_function ();                   // Your Function to work
   getrusage (RUSAGE_SELF, &end);      // get time at end

   printf ("System: %d usecs, User: %d usecs\n",
         end.ru_stime.tv_usec - start.ru_stime.tv_usec,
         end.ru_utime.tv_usec - start.ru_utime.tv_usec);
…
```

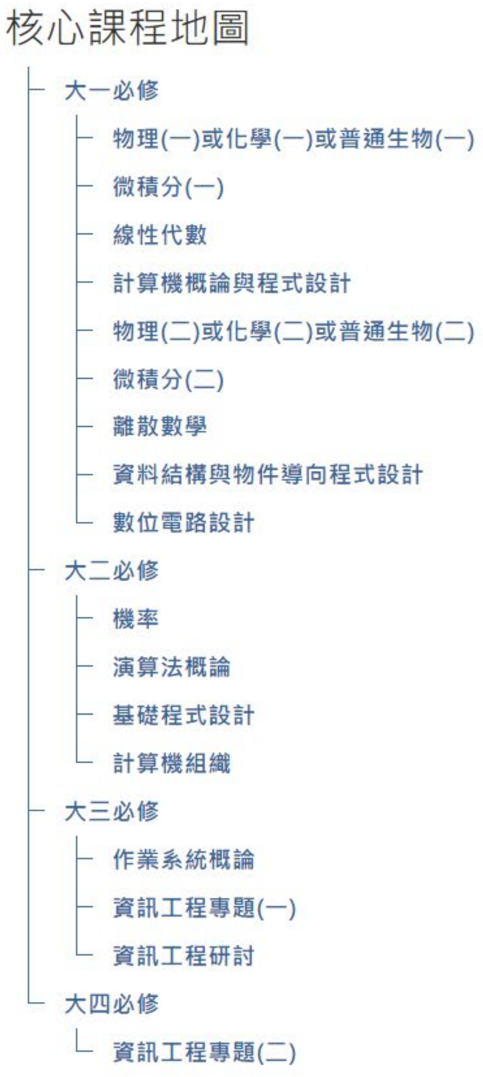
You have to consider seconds, too!

Epilogue

Your future life in 資工系

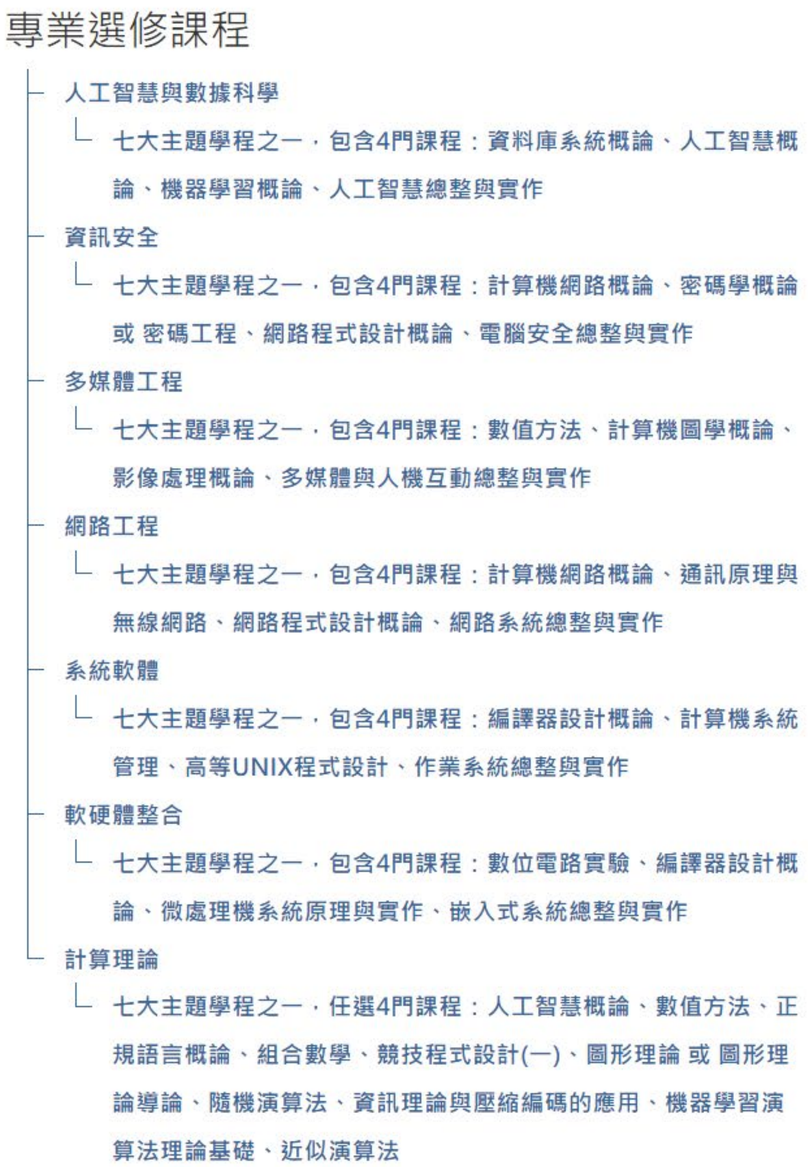
(copyright of figures from different course sources)

核心課程 + 專業選修課程



核心課程地圖

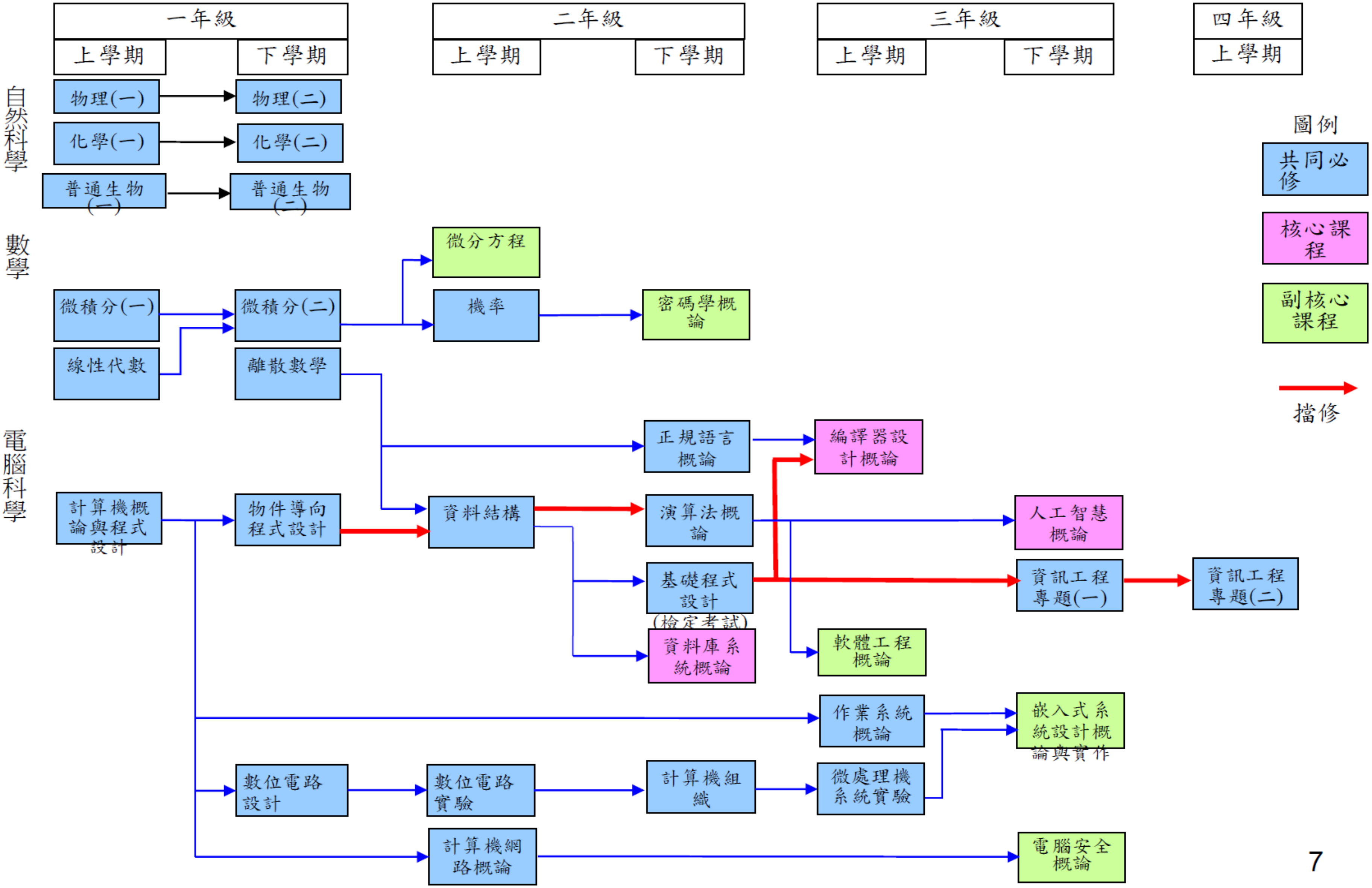
- 大一必修
  - 物理(一)或化學(一)或普通生物(一)
  - 微積分(一)
  - 線性代數
  - 計算機概論與程式設計
  - 物理(二)或化學(二)或普通生物(二)
  - 微積分(二)
  - 離散數學
  - 資料結構與物件導向程式設計
  - 數位電路設計
- 大二必修
  - 機率
  - 演算法概論
  - 基礎程式設計
  - 計算機組織
- 大三必修
  - 作業系統概論
  - 資訊工程專題(一)
  - 資訊工程研討
- 大四必修
  - 資訊工程專題(二)



專業選修課程

- 人工智慧與數據科學
  - 七大主題學程之一，包含4門課程：資料庫系統概論、人工智慧概論、機器學習概論、人工智慧總整與實作
- 資訊安全
  - 七大主題學程之一，包含4門課程：計算機網路概論、密碼學概論或 密碼工程、網路程式設計概論、電腦安全總整與實作
- 多媒體工程
  - 七大主題學程之一，包含4門課程：數值方法、計算機圖學概論、影像處理概論、多媒體與人機互動總整與實作
- 網路工程
  - 七大主題學程之一，包含4門課程：計算機網路概論、通訊原理與無線網路、網路程式設計概論、網路系統總整與實作
- 系統軟體
  - 七大主題學程之一，包含4門課程：編譯器設計概論、計算機系統管理、高等UNIX程式設計、作業系統總整與實作
- 軟硬體整合
  - 七大主題學程之一，包含4門課程：數位電路實驗、編譯器設計概論、微處理機系統原理與實作、嵌入式系統總整與實作
- 計算理論
  - 七大主題學程之一，任選4門課程：人工智慧概論、數值方法、正規語言概論、組合數學、競技程式設計(一)、圖形理論 或 圖形理論導論、隨機演算法、資訊理論與壓縮編碼的應用、機器學習演算法理論基礎、近似演算法

資工課程地圖



一年級

上學期

下學期

二年級

上學期

下學期

三年級

上學期

下學期

四年級

上學期

自然科學

物理(一)

物理(二)

化學(一)

化學(二)

普通生物(一)

普通生物(二)

數學

微分方程

微積分(一)

微積分(二)

機率

密碼學概論

線性代數

離散數學

電腦科學

正規語言概論

編譯器設計概論

計算機概論與程式設計

物件導向程式設計

資料結構

演算法概論

人工智慧概論

基礎程式設計(檢定考試)

資訊工程專題(一)

資訊工程專題(二)

資料庫系統概論

軟體工程概論

作業系統概論

嵌入式系統設計概論與實作

數位電路設計

數位電路實驗

計算機組織

微處理機系統實驗

計算機網路概論

電腦安全概論

圖例

共同必修

核心課程

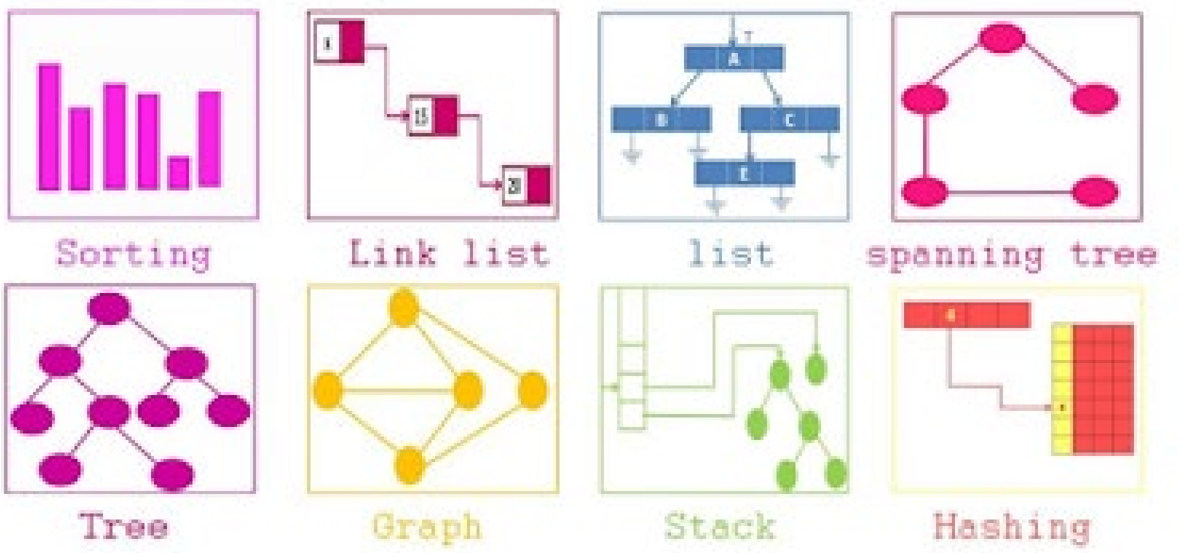
副核心課程

擋修

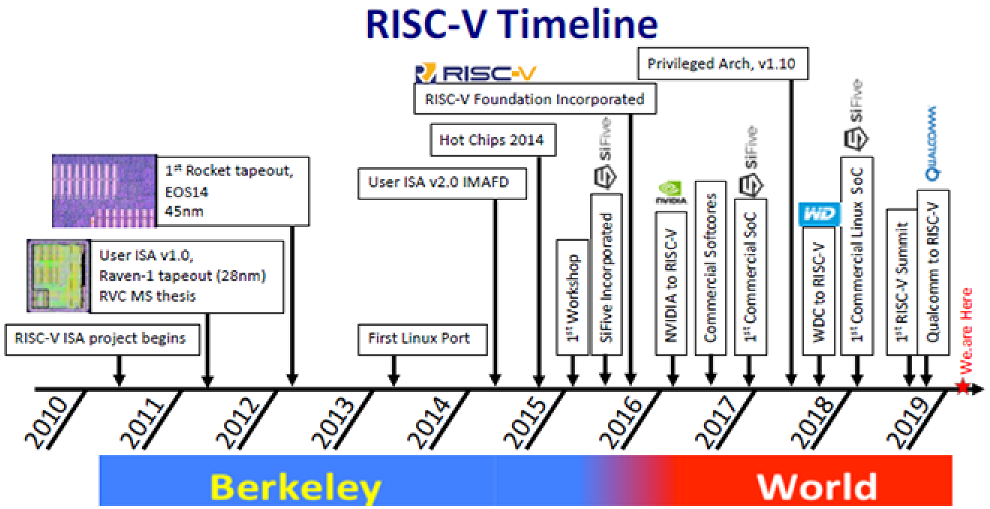
Data Structures

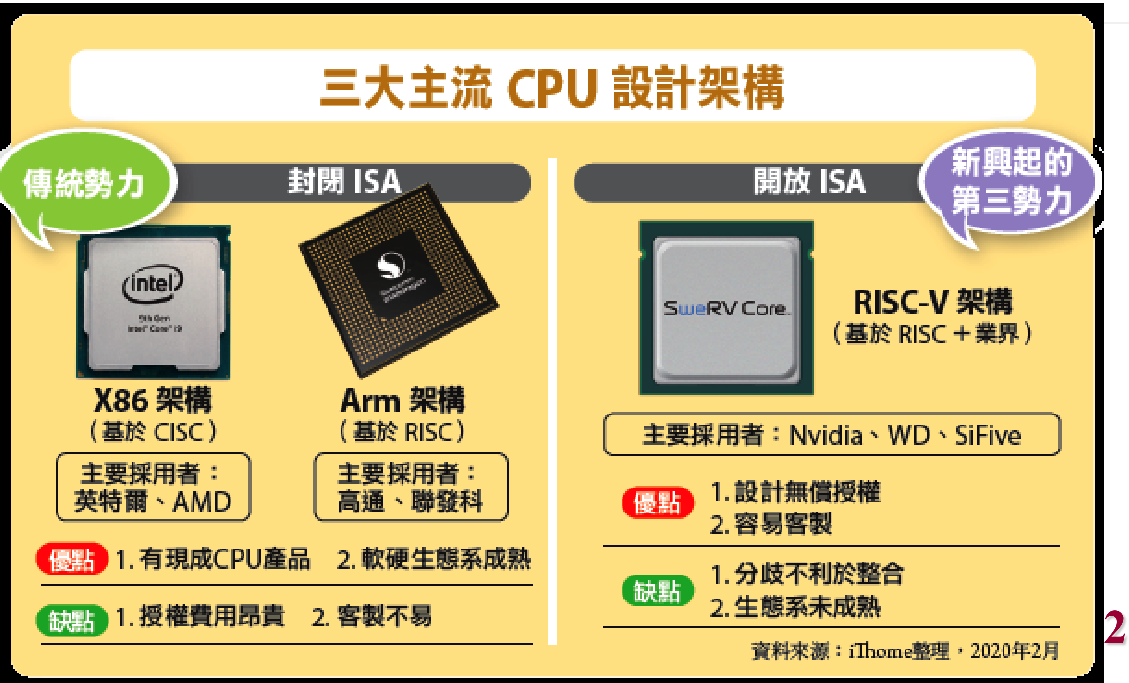
1. Arrays
2. Stacks and Queues
3. Linked Lists
4. Trees
5. Graphs
6. Sorting
7. Hashing

程式 =資料結構 + 演算法



計算機組織





三大主流 CPU 設計架構

傳統勢力

封閉 ISA

X86 架構
（基於 CISC）

主要採用者：
英特爾、AMD

Arm 架構
（基於 RISC）

主要採用者：
高通、聯發科

優點 1. 有現成CPU產品　2. 軟硬生態系成熟

缺點 1. 授權費用昂貴　2. 客製不易

開放 ISA

新興起的第三勢力

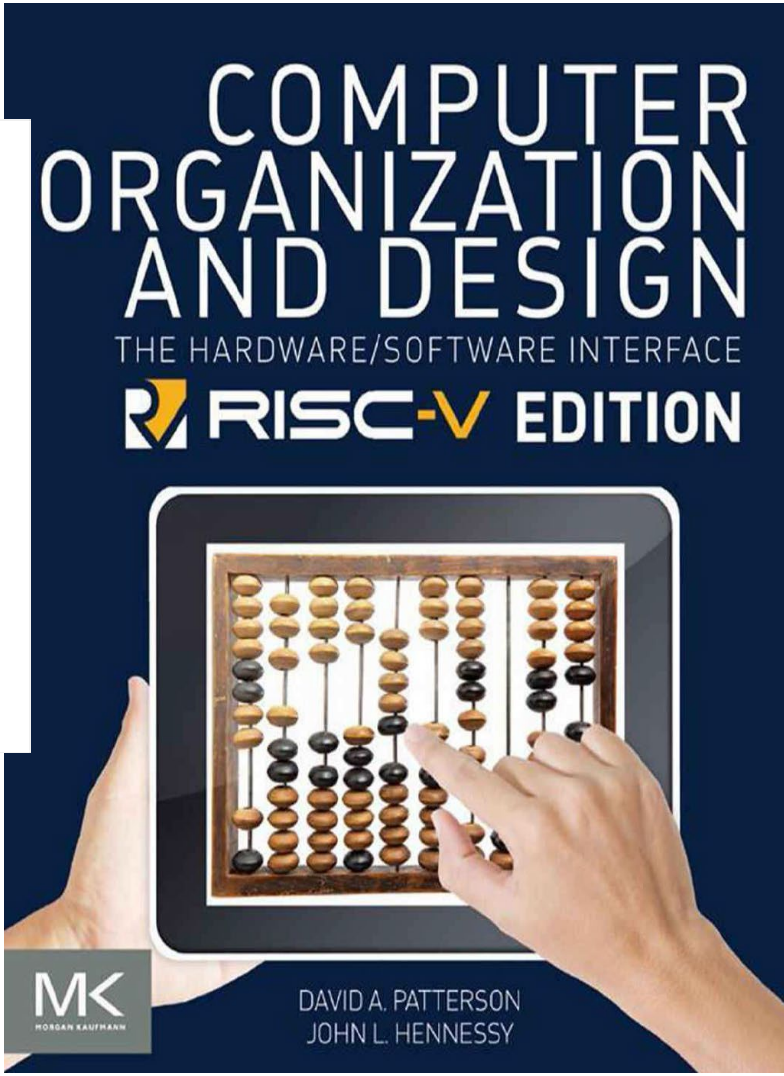
RISC-V 架構
（基於 RISC＋業界）

主要採用者：Nvidia、WD、SiFive

優點 1. 設計無償授權
2. 容易客製

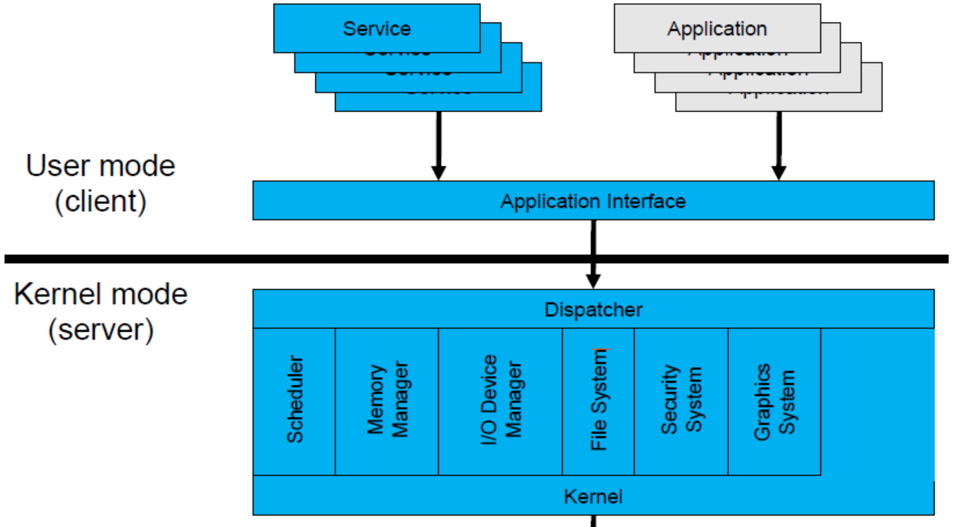
缺點 1. 分歧不利於整合
2. 生態系未成熟

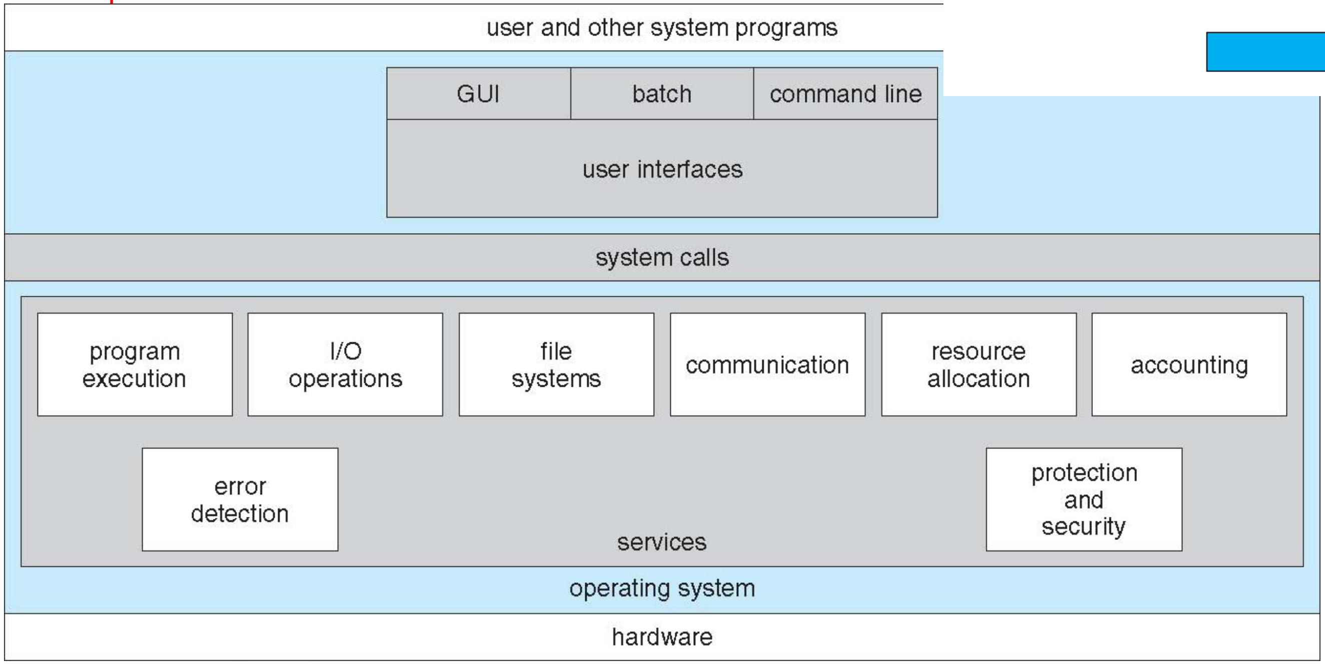
資料來源：iThome整理，2020年2月



作業系統 (Operating systems)

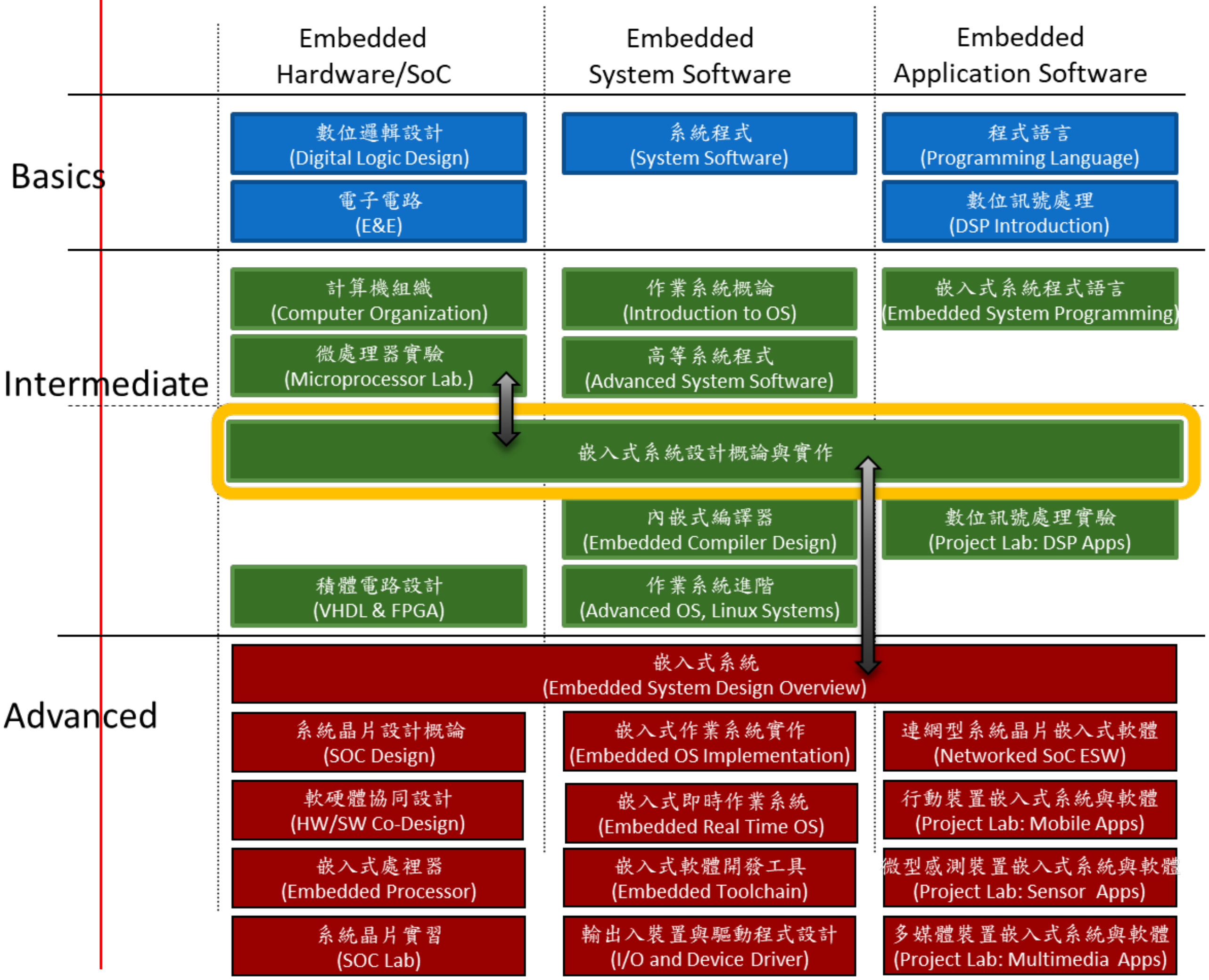
- ❑ Structure of Operating System
  - User mode
  - Kernel mode
- ❑ Linux



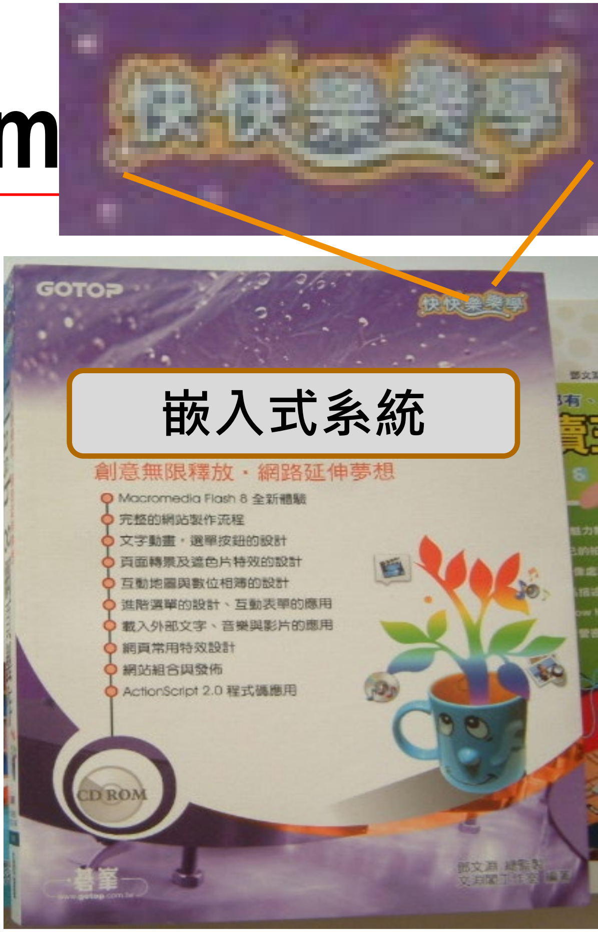


T.-F. Chen@NYCU CSIE

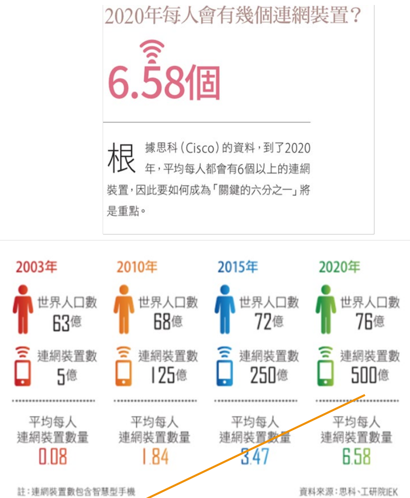
嵌入式系統(Embedded System



	Embedded Hardware/SoC	Embedded System Software	Embedded Application Software
Basics	數位邏輯設計 (Digital Logic Design) 電子電路 (E&E)	系統程式 (System Software)	程式語言 (Programming Language) 數位訊號處理 (DSP Introduction)
Intermediate	計算機組織 (Computer Organization) 微處理器實驗 (Microprocessor Lab.)	作業系統概論 (Introduction to OS) 高等系統程式 (Advanced System Software)	嵌入式系統程式語言 (Embedded System Programming)
	嵌入式系統設計概論與實作		
	積體電路設計 (VHDL & FPGA)	內嵌式編譯器 (Embedded Compiler Design) 作業系統進階 (Advanced OS, Linux Systems)	數位訊號處理實驗 (Project Lab: DSP Apps)
Advanced	嵌入式系統 (Embedded System Design Overview)		
	系統晶片設計概論 (SOC Design) 軟硬體協同設計 (HW/SW Co-Design) 嵌入式處裡器 (Embedded Processor) 系統晶片實習 (SOC Lab)	嵌入式作業系統實作 (Embedded OS Implementation) 嵌入式即時作業系統 (Embedded Real Time OS) 嵌入式軟體開發工具 (Embedded Toolchain) 輸出入裝置與驅動程式設計 (I/O and Device Driver)	連網型系統晶片嵌入式軟體 (Networked SoC ESW) 行動裝置嵌入式系統與軟體 (Project Lab: Mobile Apps) 微型感測裝置嵌入式系統與軟體 (Project Lab: Sensor Apps) 多媒體裝置嵌入式系統與軟體 (Project Lab: Multimedia Apps)


快快樂樂學
GOTOP
創意無限釋放・網路延伸夢想
Macromedia Flash 8 全新體驗
完整的網站製作流程
文字動畫，選單按鈕的設計
頁面轉景及遮色片特效的設計
互動地圖與數位相簿的設計
進階選單的設計、互動表單的應用
載入外部文字、音樂與影片的應用
網頁常用特效設計
網站組合與發佈
ActionScript 2.0 程式碼應用
CD ROM
碁峯

嵌入式系統



2020年每人會有幾個連網裝置？

6.58個

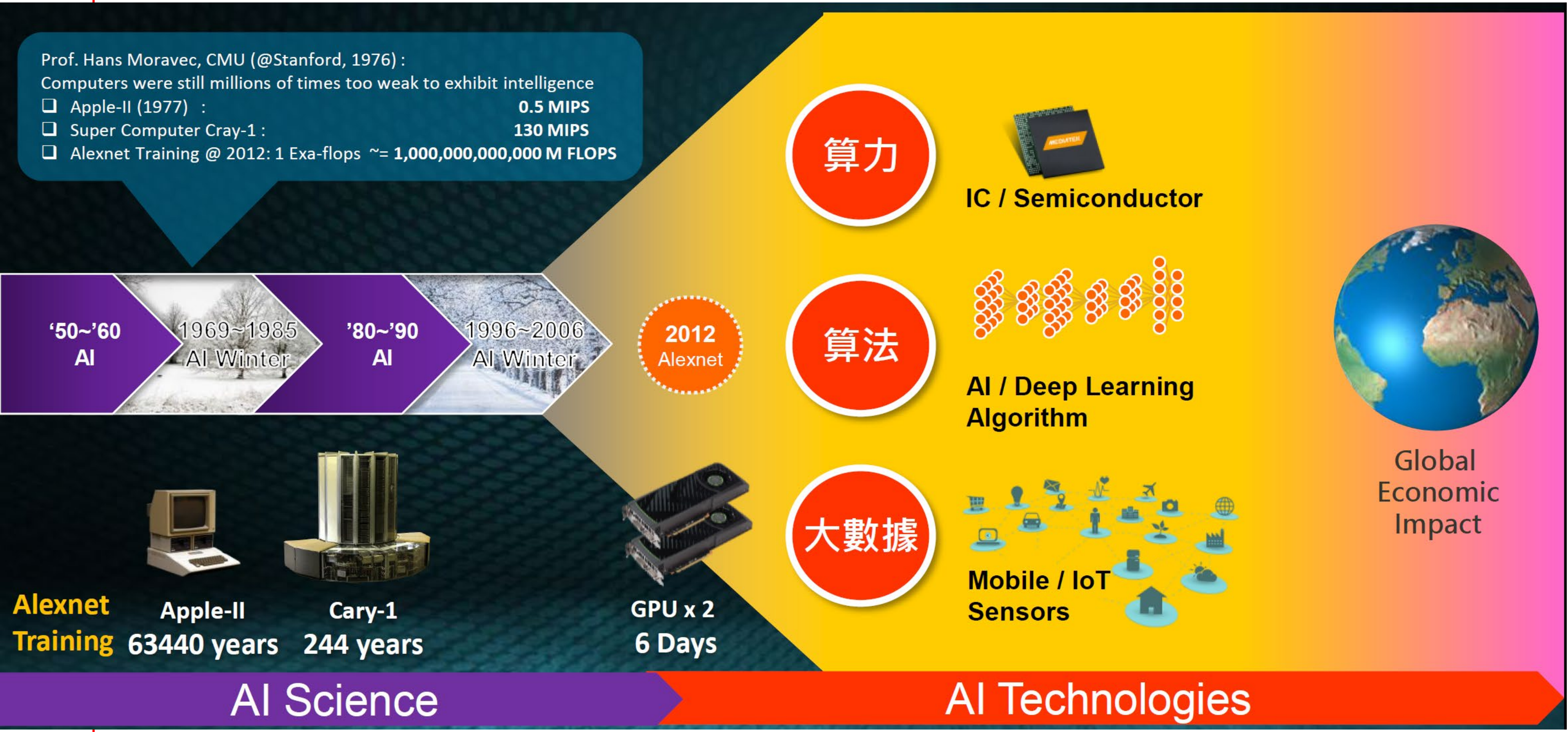
根據思科（Cisco）的資料，到了2020年，平均每人都會有6個以上的連網裝置，因此要如何成為「關鍵的六分之一」將是重點。

2003年	2010年	2015年	2020年
世界人口數 63億	世界人口數 68億	世界人口數 72億	世界人口數 76億
連網裝置數 5億	連網裝置數 125億	連網裝置數 250億	連網裝置數 500億
平均每人連網裝置數量 0.08	平均每人連網裝置數量 1.84	平均每人連網裝置數量 3.47	平均每人連網裝置數量 6.58

註：連網裝置數包含智慧型手機

資料來源：思科、工研院IEK

“Computing” is the key to enable AI



Prof. Hans Moravec, CMU (@Stanford, 1976) :
Computers were still millions of times too weak to exhibit intelligence

- ❑ Apple-II (1977) : 0.5 MIPS
- ❑ Super Computer Cray-1 : 130 MIPS
- ❑ Alexnet Training @ 2012: 1 Exa-flops ~= 1,000,000,000,000 M FLOPS

Source: (1) 聯發科 梁伯嵩資深處長
(2) History of artificial intelligence, Wikipedia
(3) Yann LeCun, Facebook AI Research, “Deep Learning Hardware: Past, Present, & Future”, ISSCC