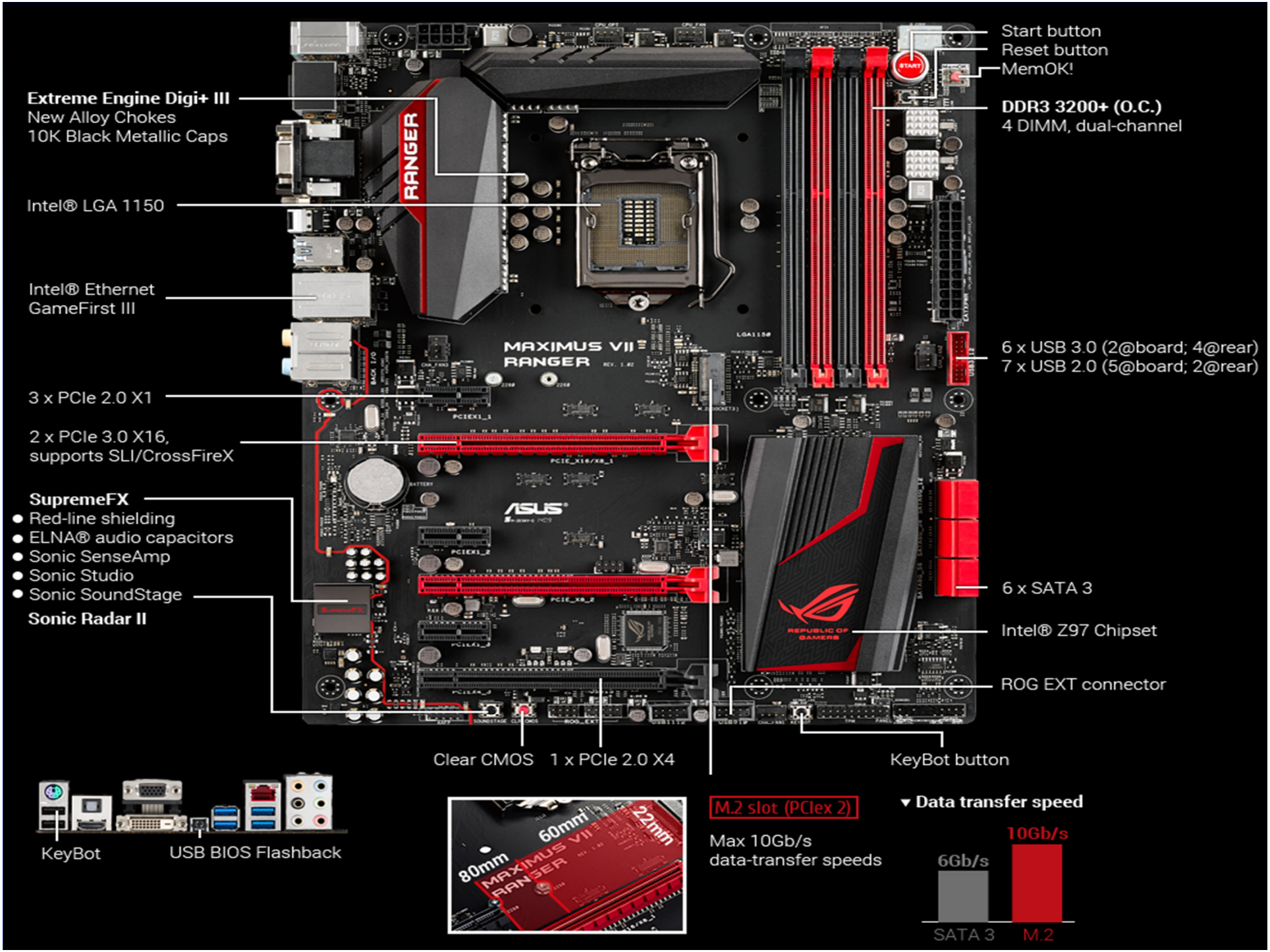
PC運作原理


Extreme Engine Digi+ III
New Alloy Chokes
10K Black Metallic Caps
Intel® LGA 1150
Intel® Ethernet
GameFirst III
3 x PCIe 2.0 X1
2 x PCIe 3.0 X16,
supports SLI/CrossFireX
SupremeFX
Red-line shielding
ELNA® audio capacitors
Sonic SenseAmp
Sonic Studio
Sonic SoundStage
Sonic Radar II
Start button
Reset button
MemOK!
DDR3 3200+ (O.C.)
4 DIMM, dual-channel
6 x USB 3.0 (2@board; 4@rear)
7 x USB 2.0 (5@board; 2@rear)
6 x SATA 3
Intel® Z97 Chipset
ROG EXT connector
KeyBot button
Clear CMOS
1 x PCIe 2.0 X4
KeyBot
USB BIOS Flashback
M.2 slot (PCIex 2)
Max 10Gb/s
data-transfer speeds
80mm
60mm
22mm
▼ Data transfer speed
6Gb/s
10Gb/s
SATA 3
M.2

PC開機程序(1)

- 源自於 Intel 8086 CPU 的規格，形成包袱至今
- 8086 有16位元資料BUS，20位元位址線。
- PC一開電，CPU就執行 CS:IP 所指位址之指令
- CS (Code Segment): FFFF
- IP (Instruction Pointer): 0000h
- CS:IP FFFF0h = 1048560
- 真實模式記憶體最高位址 1 Mega Bytes
  - 1MBytes = 1024 x 1024 = 1048576 Bytes
  - 1048576 = 2 的 20 次方，8086 只有 20 條位址線與RAM溝通。
- 1048576 – 1048560 = 16 Bytes
- jmp 0x????? => ?????乃ROM BIOS之初始化程序區，跳躍位址由各家BIOS廠商自訂，進入BIOS程序。

PC開機程序(2)

- BIOS
  - 一開始是真的唯獨晶片，但現代多由EEPROM（電氣可抹除暨可程式化唯讀記憶體）製作。
  - 內容通常分兩區，Boot block 與 Code block。前面提的 jump 先跳到 Boot block，Boot block 會先檢查 Code block 是否正確，若不正確，就用出廠內建的 Code block 來覆蓋，這是為了預防更新BIOS失敗的自救措施，如果 Code block 正確，就把 CS:IP 指向 Code block。
  - Code block 就是真正的 BIOS 程式碼，開始進行 POST 程序。
- POST（Power On Self Test 開機自我檢測）
  - 檢查電腦各項組件及其設定，如CPU、RAM、鍵盤、滑鼠等等狀態。
  - 尋找被內建在BIOS內部的顯示卡程序並執行(通常被放在記憶體C0000h的位置)，作用是顯示卡的初始化，而大部分的顯示卡都會在顯示器上顯示其相關訊息。
  - 掃瞄UMB（0xC0000～0xEFFFF）尋找其他裝置的ROM（唯讀記憶體），看看這些設備中哪些還有個別的BIOS。如果這時有找到任何其它裝置的BIOS，它們也會被執行。
  - 顯示啟動畫面，並開始更深入的檢測，包含我們平常可以在螢幕上看到的記憶體容量檢測，如果這時候遇到任何錯誤，就會在畫面上顯示錯誤訊息。
  - 按照CMOS內容開始偵測各項外部資源，例如USB、1394等隨插即用設備。
  - 按照CMOS設定內容決定是否將BIOS中斷函式遷移至主記憶體中，並修正BIOS的中斷向量表
  - 參照CMOS順序依次載入開機媒體的第一個磁區（Boot Sector）至記憶體 07C00h 位置，然後檢查07DFEh是否等於0xAA55(Bootable Flag)，如果不對，換下一個開機媒體，最後還不行就顯示開機失敗。
  - 找到 Bootable 媒體後，設定CS:IP至 07C00h，交出控制權，結束 BIOS 階段任務。

PC開機程序(3)

- Boot Sector：總長512 Bytes，分三段
  - MBR（Master Boot Record）：佔446 Bytes，此段程式碼先把自己複製到00600h 位置，然後找出Active的Partition，並載入該Partition的第一個Sector到記憶體的 07C00h 位置，接著交出控制權（CS:IP）給 07C00h
  - Partition Table：佔64 Bytes，記載本媒體的分割表，其中只能有一個分割是Active的狀態（Bootable）
  - 結束標記：2 Bytes（55 AA）(被Intel CPU讀進記憶體就會變成 0xAA55)
- Partition Table 硬碟分割表
  - 共64 Bytes，每個Partition需要16Bytes，故可分成四個Partition

Offset	Bytes	Description
01BE	1	Partition類型：00表示非活動Partition； 80表示活動Partition； 其他為無效Partition
01BF~01C1	3	Partition的起始地址（Head/Sector/Track），第一Partition通常起始於(1/1/0)，因此這三個Bytes應為010100 (Head: 8 bits、Sector: 6 bits、Track: 10 bits）
01C2	1	Partition的操作系統的類型（7：NTFS、83：Linux、82：Swap）
01C3~01C5	3	該Partition的結束地址（ Head/Sector/Track ）
01C6~01C9	4	該Partition起始Logical Sector，32位元整數
01CA~01CD	4	該Partition佔用的Total Sectors，32位元整數

PC開機程序(4)

- Boot Sector @ Active Partition
  - Windows NT：
    - 從根目錄載入NTLDR程式，控制權交給NTLDR
    - NTLDR將CPU從真實模式切換到保護模式，記憶體定址範圍提升到4GB
    - NTLDR程式讀取 Boot.ini 給 User 選擇 OS
    - 載入選定的OS之Kernel相關程式，交出控制權
  - Linux：
    - Boot Sector程式有多種Boot Loader可選擇，例如Lilo、Grub
    - Boot Sector僅有512Bytes，若程式複雜龐大放不下，Boot Sector內的 Loader 可加載其他磁區到記憶體來擴充功能
      - 例如Grub就將對應各種檔案系統的後續程式碼分開存放
    - 一樣都要先切換至保護模式
  - 以上OS在開機過程皆須安排好OS層級的中斷向量表

真實模式與保護模式

- 80386 32位元CPU的記憶體位址線有32條，所以可定址到2^32=4G的記憶體範圍，為了相容16位元時代產品，以A20位址線做開關，區分真實模式與保護模式 ，後來加入的AMD也不得不遵循。
- 真實模式 Real Mode
  - 8086 16位元CPU，有20條的位址線(A0-A19)，所能處理的最大容量為2的20次方，亦即 1048576 Bytes
  - 1048576 / 1024 = 1024 Kbytes = 1Mbytes
  - 定址方式：由兩個16位元暫存器的內容經處理後，變成20位元的絕對位址
  - 表達方式：0000:0000 => 0x00000～ 0xFFFFF

真實模式記憶體配置

00000H
中斷向量表
00400H
系統使用區
DOS
使用者程式區
A0000H
螢幕圖形資料區
B0000H
螢幕文字資料區
C0000H
其它介面卡控制區
F0000H
BIOS ROM
FFFFFH

保護模式定址(1)

- Protect Mode：由CPU硬體進行記憶體的存取定址與保護機制，有三種定址方式
  - 實體位址（Physical Address）
    - 32條位址線可實際定址4GB（2的32次方）
    - 0x00000000～0xFFFFFFFF
  - 線性位址（Linear Address）
    - Non-Paging 時，線性位址內容=實體位址內容
    - 有Paging時，某線性位址的內容可能在Swap中，不在實體位址中
    - Swap 由 OS 負責處理
  - 邏輯位址（Logical Address）：一般AP使用的定址法
    - 定址表示 Segment：offset
      - Segment => Segment Selector，16位元暫存器，指定Segment Descriptor號碼
      - Offset => 32位元暫存器，表示在Segment中的偏位移值
      - Segment：Offset等於Segment的基底位址加上Offset 就可以得到線性位址。
    - Segment Descriptor：GDT/LDT中的Record，描述一個 segment 的位置(Base Address)、大小、型態、存取權限等等資料
    - GDT/LDT：陣列的基底位址由GDTR/LDTR暫存器指定，藉由Segment Selector 指示的Index Number即可存取Segment Descriptor Record

保護模式定址(2)

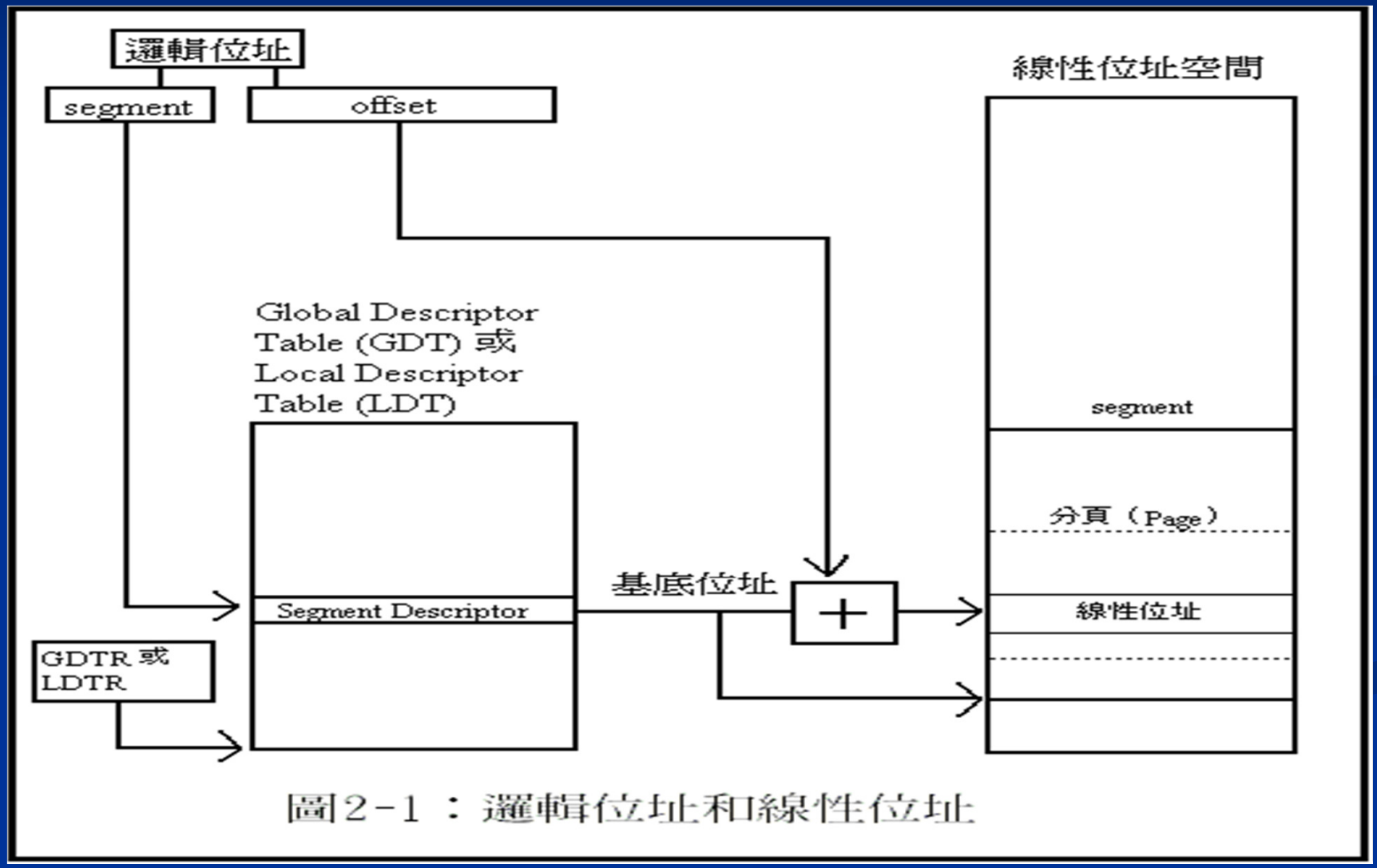


圖2-1：邏輯位址和線性位址

保護模式定址(3)

- 保護模式分頁
  - 將線性位址分割成固定大小的Page，有4KB與4MB兩種Page Size
  - CPU將Segment：Offset換算成線性位址，查對出此位址對應的Page，對應至實體位址中，如此頁並不在實體位址中，則發出 page-fault 的例外（Exception）中斷
  - OS必須提供一個 Exception handler 來處理這個例外中斷，從Swap取出（或交換）所需要的頁至實體位址中
  - 然後AP從中斷處繼續執行下去
- 分頁結構
  - 4KB Page：分 Page Directory、Page Table、Page Offset 三層
  - 4MB Page：分 Page Directory、Page Offset 兩層

EFI/UEFI BIOS

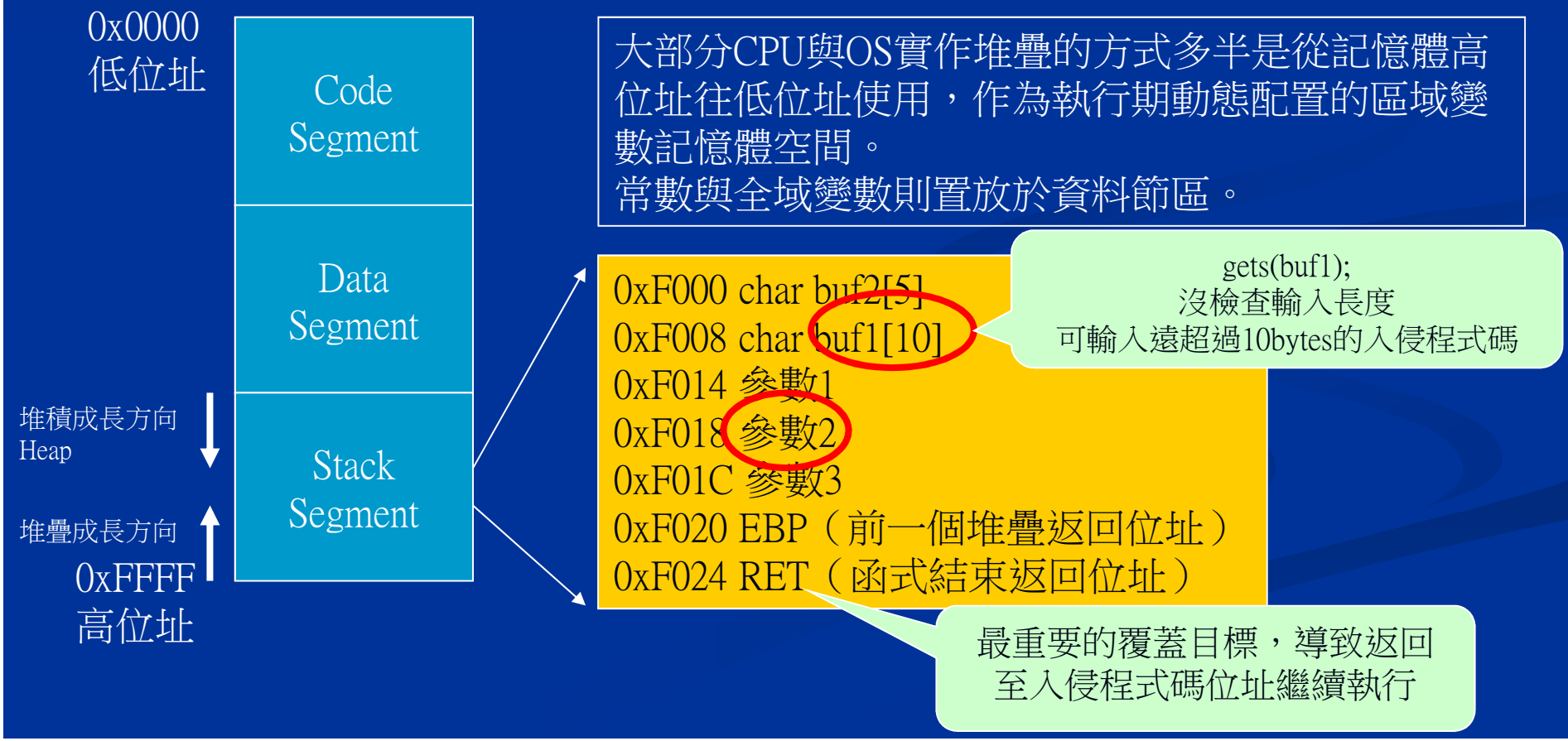
- SEC（安全性）階段：
  - 其主要的特色為「cache as RAM」，即處理器的快取當成記憶體。由於C語言需要使用堆疊，在這個階段的系統記憶體尚未被初始化，在沒有記憶體可用的情況下，便把處理器的快取當成記憶體來使用，在主記憶體被初始化之前來進行預先驗證CPU／晶片組及主機板。
  - 因為這時侯沒有快取，會導致處理器的效能變得較差，所以在記憶體初始化完畢之前，SEC和PEI階段的程式碼越簡短，越能減少這個副作用。
- PEI（EFI前初始化）階段：
  - 和傳統BIOS的初始化階段類似，用以喚醒CPU及記憶體初始化。這時候只起始了一小部分的記憶體。同時，晶片組和 主機板也開始初始化。接下來的服務程式會確定CPU晶片組被正確的初始化，在此時，EFI驅動程式派送器將載入EFI驅動程式記憶體，進入了起始所有記憶體的DXE階段（驅動程式執行環境）。
- DXE階段：
  - 主要功能在於溝通EFI驅動程式及硬體。也就是說此階段所有的記憶體、CPU（在此是指實體兩個或以上的非核心數目，也就是雙CPU插槽處理器甚至是四CPU插槽處理器）、PCI、USB、SATA和Shell都會被初始化。
- BDS（開機設備選擇）階段：
  - 使用者就可以自開機管理者程式頁面，選擇要從哪個偵測到的開機設備來啟動。
- TSL（短暫系統載入）階段：
  - 由作業系統接手開機。除此之外，也可以在BDS階段選擇UEFI Shell，讓系統進入簡單的命令列，進行基本診斷和維護。

Why UEFI?

- 避免傳統BIOS的缺點
  - 讓未來的CPU可以拋棄16位元真實模式
  - 組合語言BIOS維護人才難尋，而C語言BIOS太肥難以塞入真實模式定址中運行，必須拋棄真實模式
- 跨平台優勢
  - UEFI BIOS利用載入EFI driver的形式，來進行硬體的辨識／控制及系統資源掌控，這些driver並不是可以直接在CPU執行的代碼，而是用EBC（EFI Byte Code）這種專用於EFI driver的虛擬機器指令，該指令必須在UEFI的DXE階段被解壓縮後翻譯執行，所以driver開發可與實際CPU、晶片等脫節。
- EFI Shell 功能強大
  - EFI Shell 是個精簡的作業系統，可以讓使用者進行BIOS的更新、系統診斷、安裝特定軟體，甚至可以播放CD和DVD而不需完全載入OS，EFI driver可以被載入或卸載，連TCP/IP核心程式都可以使用。基於EFI的driver model可使UEFI系統接觸到所有的硬體功能，在進入作業系統之前瀏覽網站不再是天方夜譚，甚至實作起來也非常簡單。

緩衝區溢出攻擊技術(1)

- Runtime Process 結構分區與堆疊溢出原理



緩衝區溢出攻擊技術(2)

- 緩衝區溢出攻擊技術
  - 植入法
  - 利用Process現有的Function Code
  - 修改Function Point
  - 修改longjmp指標