“Simplitatea este complexitatea inteleasa”.

C. Brancusi

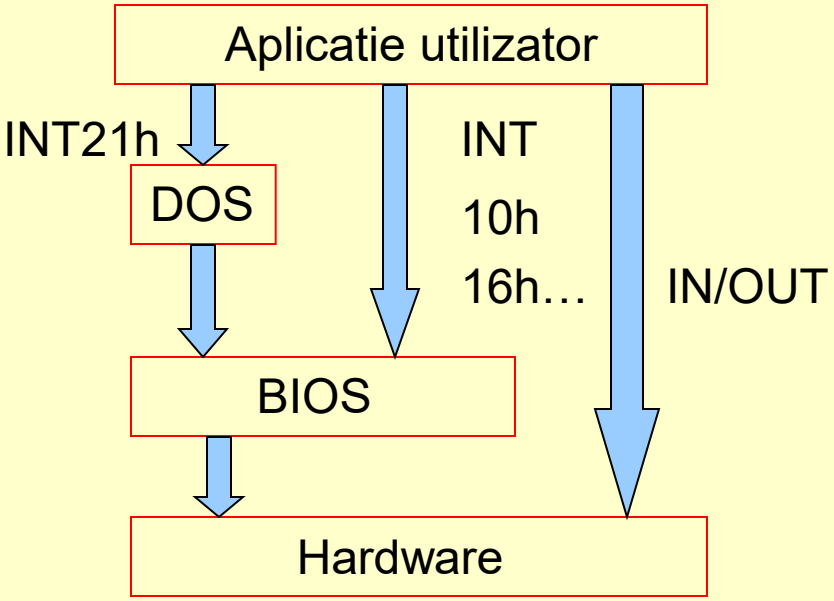
Curs 10

Servicii sistem
Interfata aplicatiilor ASM cu SO

Cuprins

- ✓ Servicii sistem - INT21h
- ✓ Interfata aplicatiilor ASM cu SO. Aplicatii .com si .exe
- ✓ **UEFI**

1. Servicii sistem



Ex. Sa se preia de la tastatura un sir de caractere si sa se afiseze pana la tastarea ‘ESC’.

Servicii DOS - INT 21h

INT 20H Terminate a program
INT 21H DOS Services
INT 22H Terminate address
INT 23H Control-Break address
INT 24H Critical Error Handler address
INT 25H/26H Absolute Disk Read/Write
INT 27H Terminate but Stay Resident
INT 28H DOS Idle (safe to pop up)
INT 29H DOS Internal Fast Screen Write
INT 2eH Perform DOS Command
INT 2fH Multiplex (DoubleSpace, spooler, TSR control, other APIs)
INT 31H DPMI DOS Protected Mode Interface Services
INT 33H Mouse Support
INT 67H EMS Expanded Memory Manager (HIMEM.SYS)

Servicii DOS - INT 21h

- INT 20h – Terminare program se foloseste ptr. a parasi aplicatia si a returna controlul procesului parinte (ex. COMMAND.COM)

  OBS. CS=PSP; JMP/RET => PSP:0

- INT 21h
  - peste 100 de functii
  - terminare, iesire program
  - citire KBD, display char, sir..
  - open/close/read/write/delete file
  - MD/RD/CD (directoare)
  - set/get vector (TVI)
  - ……..

Servicii DOS - INT 21h

Functii DOS
Grup de Servicii
Interuperi DOS

00H Terminate
01H Kybd Input
02H Display Char
05H Prn Output
06H Console I/O
07H NoEcho RawInp
08H NoEcho Inp
09H Display Text
0aH Bufrd Input
0bH Get InpStatus
0cH Clear & Input
0dH Reset Disk
0eH Set Dflt Disk
19H Get Dflt Disk
1aH Set DTA
25H Set INT Vector
2aH Get Sys Date
2bH Set Sys Date
2cH Get Sys Time
2dH Set Sys Time
2eH Set Verify
2fH Get DTA
30H Get Version
31H TSR
32H Get DPB
3300H Get BreakLvl
3301H Set BreakLvl
3305H Get Boot Drv
34H Get InDOS
35H Get INT Vector
36H Disk Size/Free
39H MkDir
3aH RmDir
3bH ChDir
3cH Create File
3dH Open File
3eH Close File
3fH Read File
40H Write File
41H Delete File
42H Move File Ptr
43H File Attrib
44H IOCTL
45H Dup Handle
46H Redir Handle
47H Get Dflt Dir
48H Mem Alloc
49H Mem Free
4aH Mem Resize
4bH Exec
4cH Terminate
4dH Get Exit Code
6cH Open/Create File
4eH Find File/GetInfo
4fH Find Next File
50H Set Cur PSP
51H Get Cur PSP
52H List Of Lists
54H Get Verify
56H Rename/Move File
57H Time/Date File
58H MemAlloc Strategy
59H Get Err Info
5aH Create Uniq File
5bH Create New File
5cH Lock File
5eH Network Misc
5fH Network Redirect
65H National Lang Fns
67H Set Handle Cnt
68H Commit File Data

Servicii DOS - INT 21h

AH=0 - terminare program CS=PSP

AH=4Ch - terminare program cu cod de sfarsit (elib.mem.)

AH=	terminate type
01h	normal termination (INT 20h, INT 21h Fct. 00h, or INT 21 Fct. 4Ch)
02h	Ctrl-C or Ctrl-Break abort
03h	termination by critical error handler
04h	terminate and stay resident (INT 21h Function or INT 27h)

AH=4Dh - preluare cod terminare

AH= 1h - citire caracter de la tastatura in AL

AH= 2h - display caracter din DL, la perifericul standard de iesire

AH= 9 h - display sir de caractere$, DX=offset sir

AH=25h - seteaza noua adresa a rutinei de intr. in TVI (AL=tip vector)

AH=35h - salveaza adresa rutinei din TVI (ES:DX)

INT 21h

- **functia 01h** : *input caracter cu ecou*

  AH=01h

- Asteapta pana cand este citit un caracter de la tastatura, apoi trimite ecoul pe display; caracterul citit va fi in AL

```
MOV AH,1
INT 21H       ; AL= cod ASCII al caracterul tastat
```

- **functia 02h** : *afisare caracter pe monitor*

  AH=02h, DL=caracter de afisat

```
MOV AH, 02 ;
MOV DL, ‘k’ ;
INT 21H
```

INT 21h

```
String    DB    'Hello world', 13, 10
                  ...
          LEA   DI,STRING
          MOV   AH,02h
          MOV   CX,13
LP1:      MOV   DL,[DI]
          INT   21h
          INC   DI
          LOOP  LP1
                  ...
```

INT 21h

- **Functia 09h** ; IESIRE TEXT LA DISPLAY

;Afisez pe ecran sir de caractere$, cu offset in DX

```
MOV    AH,9                    ; fct afisare sir caractere
MOV    DX, OFFSET MESAJ;       ; DX contine adresa primului caracter
INT    21H                     ; din sir
```

……

```
.data
MESAJ  DB " ACESTA ESTE UN TEXT $"
```

INT 21h

- **Functia 0Ah : INPUT STRING FROM KEYBOARD**

  ;set AH=0Ah, DX=offset- adresa la care memorez caracterele

  - I byte specifica dimensiunea buffer-ului
  - II byte specifica nr. caracterelor introduse (contor curent)
  - de la al III–lea byte => datele introduse

```
        ORG 100h
DATA1   DB      6,?,6DUP(FFh)       ;0100h=06, ??,0102h-07h = 0FFh
START:  MOV     AH,0Ah              ; fct de citire sir cu INT 21h
        MOV     DX,OFFSET DATA1     ; DX= adresa buffer
        INT     21h
```

INT 21h

- **Functia 35h** ; GET VECTOR INTR.

In: AH=35h
AL=tip INTR.

Out: ES= Adr. Segm. din TVI
BX=Adr. Offset din TVI

- **Functia 25h** ; SET VECTOR INTR.

In: AH=25h
AL=tip INTR.
DS=Noua Adr. Segm.
DX=Noua Adr. Offset

Out: DS= =>> 0: 4*tip+2 in TVI
DX= =>> 0:4*tip in TVI

INT 21h

- **Functia 3Ch** ; Crearea un nou fisier

In: AH=3Ch; DS:DX = Adr.Identif. Fisier; CX=atribut

Out: AX=error , if C=1// else AX = file handle (if C=0)

- Fie identificatorul de fisier: A:\PROGS\PROG1.ASM
- Erori posibile:
  - Calea nu exista
  - toate file handles sunt in uz
  - acces interzis --- director plin sau fisier read-only

Ex: Scrieti secventa de instructiuni ptr. a crea un nou fisier read-only numit "FILE1.txt"

```
FNAME    DB 'C:\file1.txt', 0
HANDLE   DW ?
.CODE
MOV AX,@DATA
MOV DS, AX          ; initializez DS
MOV AH, 3CH         ; open file
LEA DX, FNAME       ; copiez adresa FNAME in DX
MOV CL, 1           ; atribut read-only
 INT 21H            ; open the file
MOV HANDLE, AX      ; handle or err code

JC OPEN_ERROR       ; jump if error .....
```

- **Functia 3Dh** ; deschidere fisier

In: AH=3Dh, DS:DX = Adr.Identif. Fisier; CX=atribut
AL=mod deschidere (0-read/1-write/2-R/W)
Out: AX=error , if C=1//else AX=file handle if C=0

- **Functia 3Eh - "CLOSE" - CLOSE FILE**

In: BX = file handle
Out: CF =0, if successful, AX destroyed
CF=1 if error, AX = error code (06h)

- **Functia 3Fh - "READ" - READ FROM FILE OR DEVICE**

In: BX = file handle
CX = number of bytes to read
DS:DX -> buffer for data
Return: CF=0 if successful - AX = number of bytes actualy read (0 if an EOF before call)
CF =1 error AX = error code (05h,06h)

- **Functia 40h - "WRITE" - WRITE TO FILE OR DEVICE**

In: BX = file handle
CX = number of bytes to write
DS:DX -> data to write
Out: CF=0 if successful -AX = number of bytes actualy written
CF=1 if error - AX = error code (05h,06h)

File handle= un nr. pe care SO il asigneaza temporar unui fisier cand este deschis.
SO foloseste file handle intern cand acceseaza fisierul.

EX.1. Scrieti un program care curata ecranul, pozitioneaza cursorul in centrul ecranului si afiseaza un mesaj din segmentul de date

```
TITLE PROG 1 program display
.MODEL SMALL
.STACK 64
.DATA
   MESAJ DB ‘Rutina Test display’, ‘$’
.CODE
MAIN PROC FAR
      MOV AX, @DATA
      MOV DS, AX     ; initializez DS

      CALL CLEAR     ;clear screen
      CALL CURSOR   ;set pozitie cursor
      CALL DISPLAY   ;display mesaj
      MOV AH, 4CH
      INT 21H
MAIN ENDP
```

```
;subrutina de stergere ecran
CLEAR PROC
    MOV AX, 0600H      ;functia scroll up screen
    MOV BH, 7          ; atribut normal (A/N)
    MOV CX, 0          ;scroll  rind = 00, col = 00 (coltul stanga sus)
    MOV DX, 184FH      ;rind = 18H(24), col = 4FH(79)
    INT 10H            ;apel intr. ptr. clear screen
    RET
CLEAR ENDP
```

```
;subrutina setare pozitie cursor la centru ecran
CURSOR PROC
    MOV AH, 2           ;set cursor
    MOV BH, 0           ;pag  00
    MOV DH, 13          ;rand centru
    MOV DL, 39          ;col centru
    INT 10H             ; BIOS
    RET
CURSOR ENDP
;-------------------------------------------------------------------
; display sir pe ecran
DISPLAY PROC
    MOV AH, 9                   ;fct. display sir
    MOV DX, OFFSET MESAJ        ;DX pointeaza output buffer
    INT 21H ;
    RET
    DISPLAY ENDP
END MAIN
```

EX2. Genereaza sunet (bell) continuu pâna se tasteaza ‘Q’ sau ‘q’

```
.MODEL SMALL
.STACK 64
.DATA
    MESAJ DB ‘Pentru a opri  sunetul  BELL apasati  tasta Q (/ q)  $’
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    MOV AH, 9
    MOV DX, OFFSET MESSAGE      ;display  mesaj
    INT 21H

AGAIN:   MOV AH, 2              ; afisare caracter (DOS)
    MOV DL, 7                   ;generez sunet  bell
    INT 21H
    MOV AH, 1                   ;verific tastare (BIOS)
    INT 16H
    JZ AGAIN                    ; daca nu tastez, generez bell
```

```
        MOV AH, 0                ;citesc caracter
        INT 16H
        CMP AL, 'Q'              ;AL= 'Q'?
        JE EXIT                  ;daca DA, exit
        CMP AL, 'q'              ;AL= 'q'?
        JE EXIT                  ;daca DA, exit
        JMP AGAIN                ;daca nu, generez sunet  bell

EXIT:   MOV AH, 4CH
        INT 21H
        MAIN ENDP
        END
```

- **Ex3.** Sa se scrie un program in LA care sa creeze un fisier text file.txt in care sa scrie un sir de caractere definit in segmental de date

Redirectarea unei intreruperi – Exemplu

mod de realizare:

- se inlocuieste in TVI adresa rutinei vechi cu adresa noii rutine;
- la terminarea aplicatiei se reface vechea adresa in TVI
- noua rutina va contine noul “driver” al resursei (intreruperii)

```
. DATA
OLDVECT DW 2DUP(?)      ; aici salvez vechia adr. a rutinei de întrerupere

;initializare…….

MOV     AH, 35h         ; 35h-funcţie sistem pentru citirea
                        ;vectorului de întrerupere
MOV     AL, x           ; x – tipul întreruperii redirectate
INT     21h             ;apelul funcţiei sistem; funcţia returnează
                        ;în ES:BX adresa rutinei de tip x
MOV     OLDVECT, BX     ; salvare adresă offset (IP)
MOV     BX, ES
MOV     OLDVECT+2, BX   ; salvare adresă segment (CS)
MOV     AX, SEG RTI
MOV     DS, AX          ;DS <- adresa segment a noii rutine de
                        ; tratare a întreruperii (CS nou)
MOV     DX, offset RTI  ; DX <- adresa offset a noii  rutine de întrerupere (IP nou)
MOV     AH, 25h         ; 25h - funcţia de scriere vector in TVI;
                        ; în DS:DX se pune adresa rutinei de întrerupere
MOV     AL,x            ;x – tip  întrerupere
INT     21h             ;apelul funcţiei sistem
```

```
…….
; program
     ……
     INT x
……
; sfîrşit program
MOV     AX, OLDVECT+2    ; refacerea adresei vechi
MOV     DS, AX
MOV     DX, OLDVECT
MOV     AH, 25h          ; funcţia de scriere vector
MOV     AL,x                        ; n – nivel întrerupere
INT     21h                         ; înscrierea vechiului vector în
                                    ;tabela de intreruperi
…….
```

; rutina de tratare a întreruperii

```
RTI   PROC    FAR     ; noua rutina de tratare a intreruperii
      PUSH    r       ; salvarea registrelor utilizate în cadrul
      .....           ; rutinei (r = AX, BX, ….)
      STI             ; validare întrerupere, IF=1
      ……..
      ; corpul rutinei
      ……..
; sfîrşitul rutinei
      POP     r       ; refacere registre salvate
      IRET
RTI   ENDP
```

Recomandari de scriere a rutinelor

- parametrii de apel se transmit prin:
  - registrii procesorului
  - prin stiva (daca aplicatia se combina cu module scrise in HLL);
  - Reg. BP se foloseste pt. adresarea parametrilor de pe stiva
- se precizeaza modul de transmitere a parametrilor prin comentarii, ex.

```
;nume functie
; descriere tip operatie efectuata
; Intrari: ...
; Iesiri: ...
; registre afectate
```

Recomandari de scriere a rutinelor (cont.)

- continutul registrelor nu trebuie afectat de executia rutinei;
  - registrele folosite in procedura se vor salva pe stiva si se vor reface la sfarsitul rutinei
  - exceptie: registrele care contin parametrii de iesire
- majoritatea functiilor sistem nu sunt reentrante
  - nu pot fi apelate din nou inainte de terminare

- **Tema1:** Citeste numele de la tastatura si afiseaza lungimea lui:

```
    TITLE PROG_NUME
;Citeste nume si afisez lungimea
    .MODEL SMALL
    .STACK 64
    .DATA
    MSG1 DB ‘Cum va numiti?$’
    BUFFER1 DB 9, ?, 9 DUP(0)
    MSG2 DB CR, LF, ‘Numarul de litere al numelui este: $’
    RIND EQU 08
    COL EQU 05
    CR EQU 0DH ;
    LF EQU 0AH ;
    ……..
```

- **Tema2:** Analizati aplicatia de mai jos si stabiliti care este efectul ei.

```
.model small
.stack    256
.code
          mov ax,@data
          mov ds,ax
          mov ax,0B800h                    ; segment of video buffer
          mov es,ax                        ; put this into es
           mov ah,3                        ; attribute - cyan
          mov cx,19                         ; length of string to print
          mov si,offset Text                ; DX:SI points to string
          xor di,di                         ; DI=0
wr_Char: lodsb                             ; put next character into al
          mov es:[di],al                   ; output character to video memory
          inc di                           ; move along to next column
          mov es:[di],ah                   ; output attribute to video memory
          inc di
          loop wr_Char                     ; loop until done
          mov ax,4C00h
                                           ; return to DOS int 21h
.....
.data
          Text DB “Acesta este un text“
end
```

2. Interfatarea aplicatiilor ASM cu SO

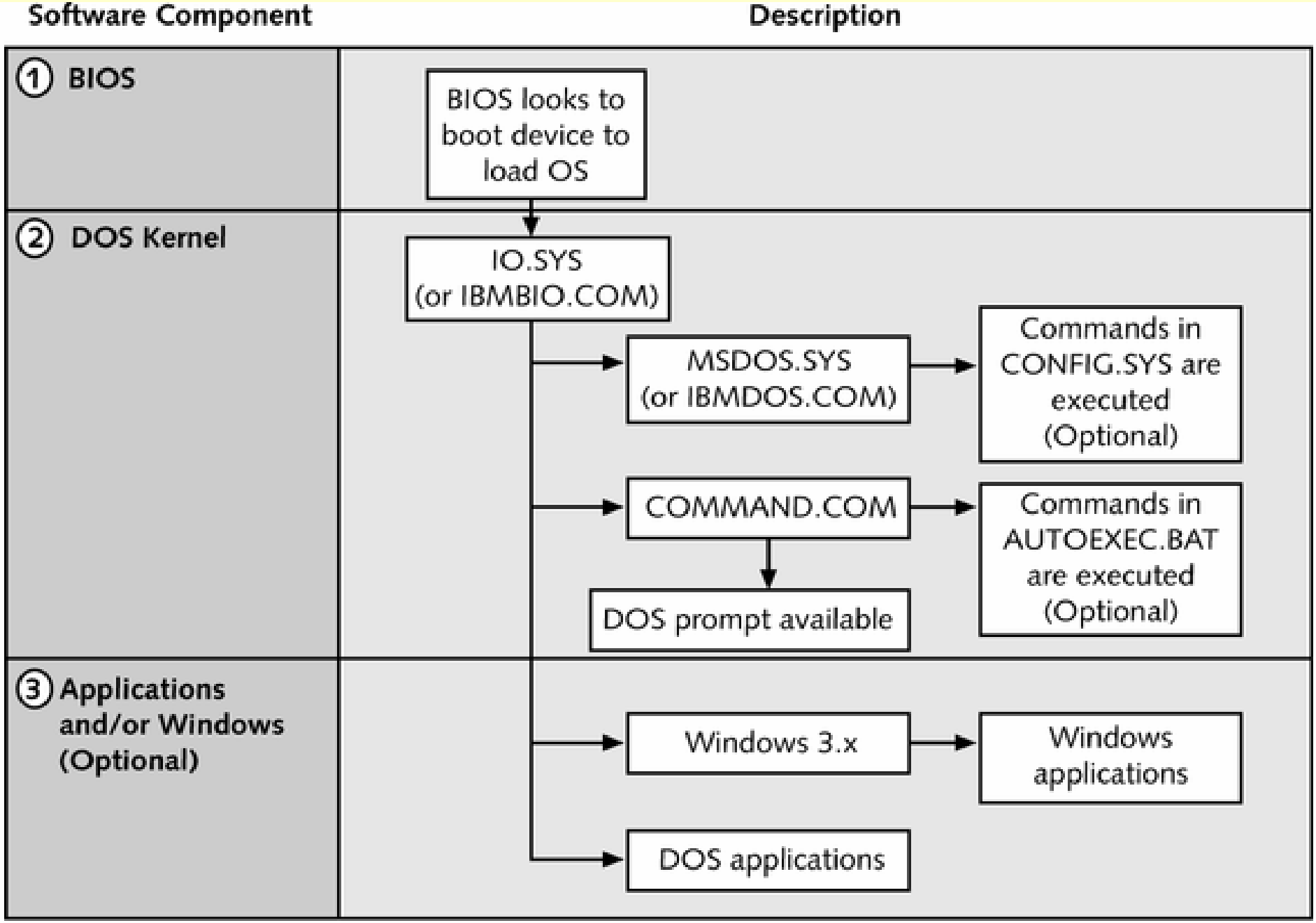
- Componentele SO DOS:

1. ROM BIOS (POST, bootstrap, I/O)
2. DOS BIOS (IBM.IO/IO.SYS, drivere CON, PRN, AUX, CLK, FD,HDD) - acces la drivere, implica apel rutine ROM BIOS
3. DOS kernel (MSDOS.SYS/IBMDOS.COM) - functii de acces la fisiere (I/O la nivel caracter, independent de hard, INT 21h)
4. Interpretorul de comenzi (shell - COMMAND.COM)

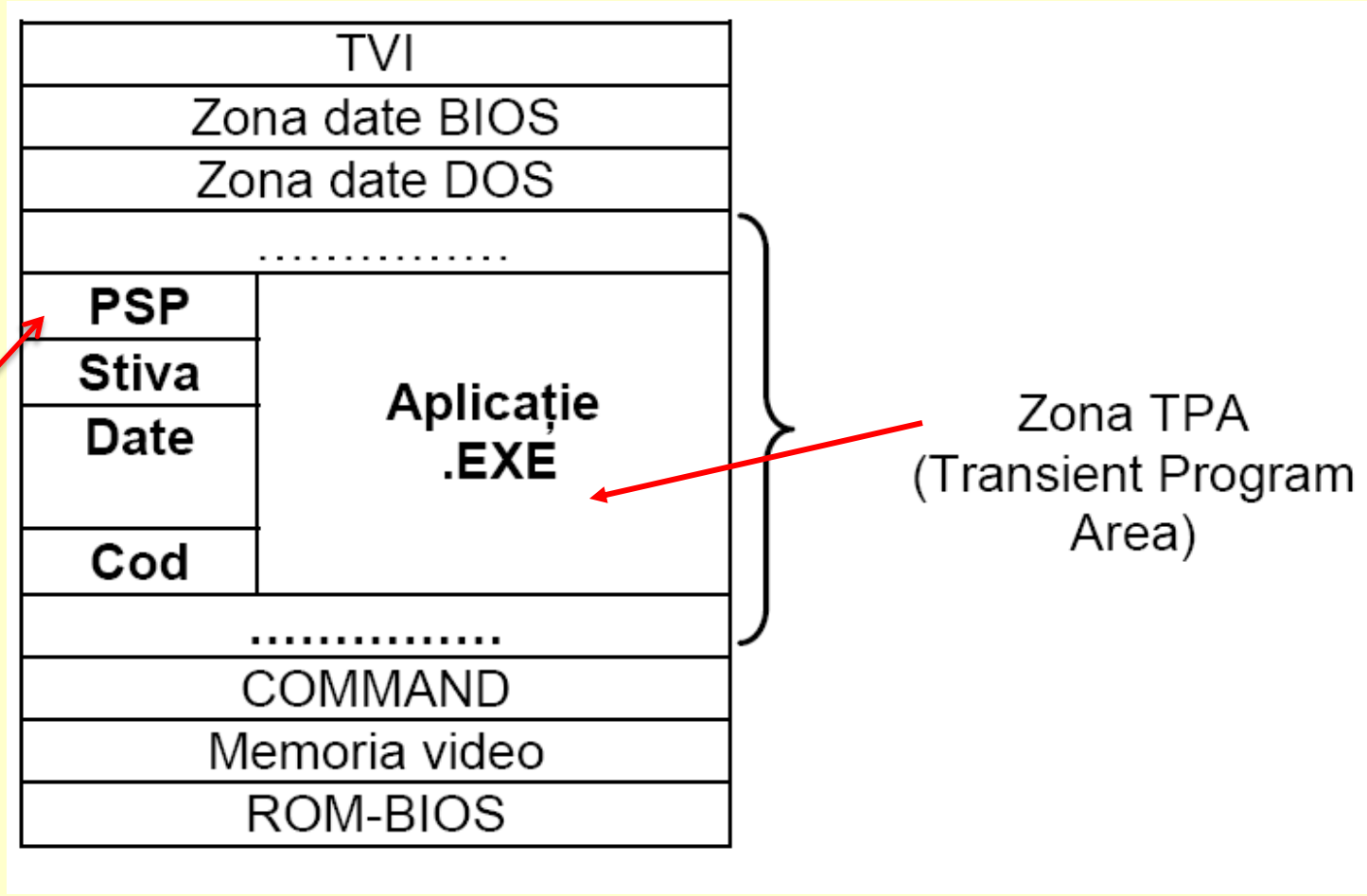
   - partea rezidenta (handlere=rutine tratare diferite actiuni)
   - partea tranzitorie (C:\>, citeste cda, com. interne, la terminare control => partea rezidenta
   - rutina de initializare

   COMENZI: Interne, Externe, Indirecte (.bat)

Procesul de Bootare



Interfatarea aplicatiilor ASM cu SO



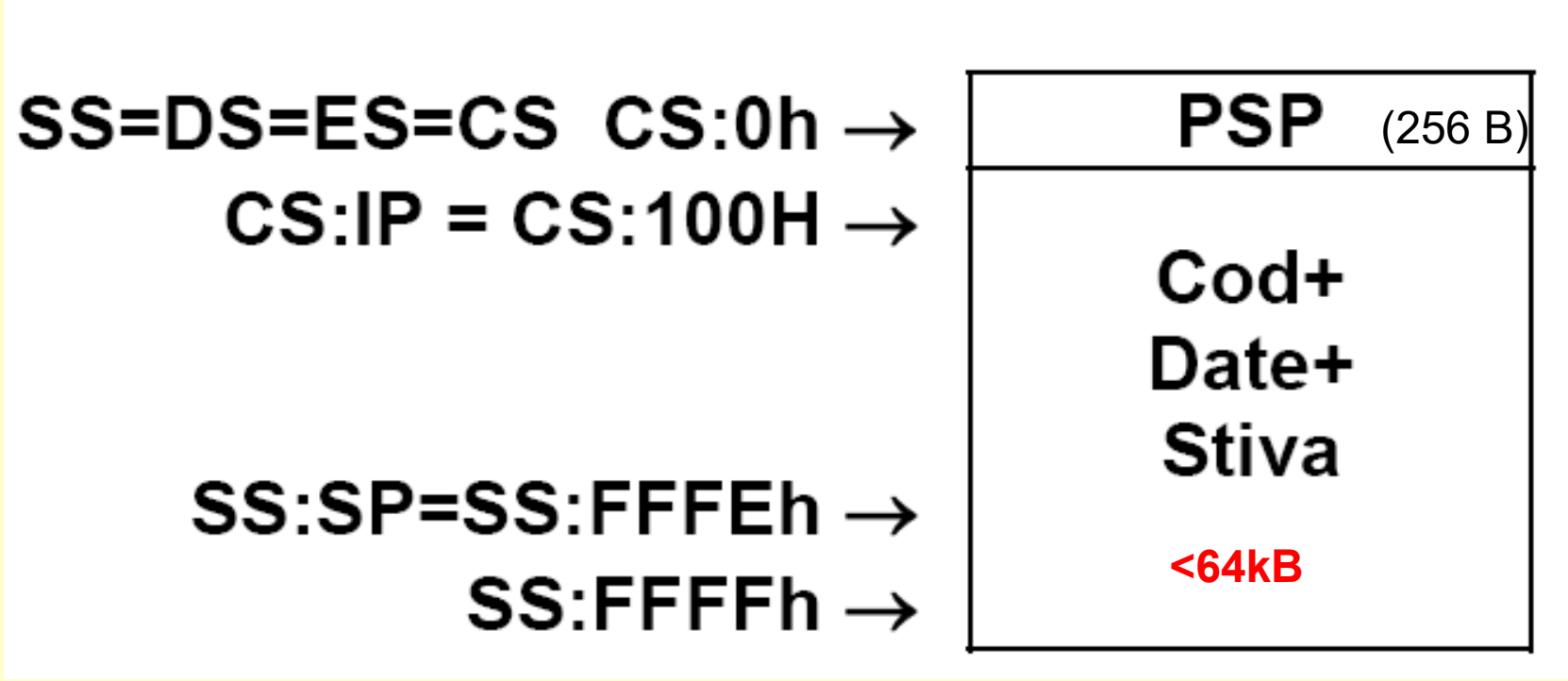
HARTA MEMORIEI

Interfatarea aplicatiilor ASM cu SO

STRUCTURA PSP

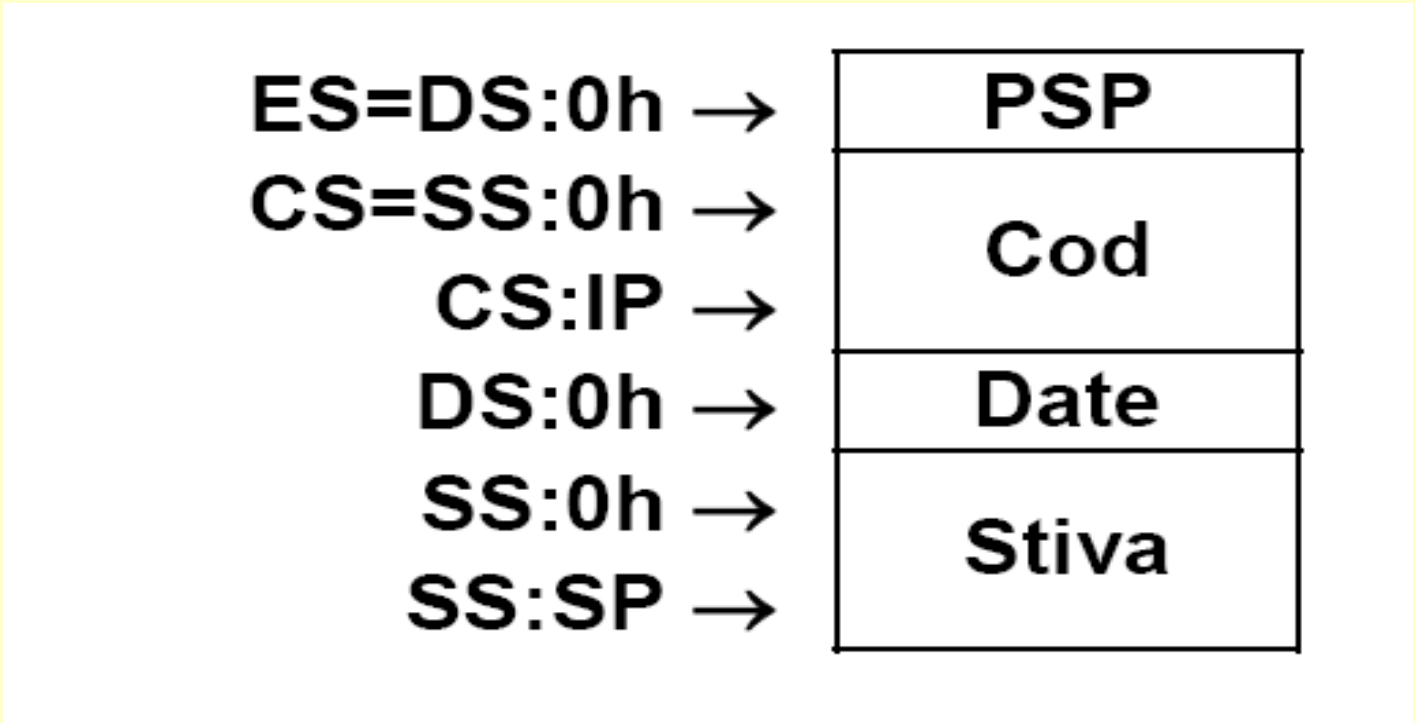
00h	**Codul instrucțiunii INT 20h**
02h	**Sfârşitul memoriei ocupate de program**
04h	Un octet rezervat
05h	Codul instrucțiunii INT 21h
0Ah	Adresa FAR a RTI 22h
0Eh	Adresa FAR a RTI 23h
12h	Adresa FAR a RTI 24h
16h	21 octeți rezervați
2Ch	Adresa segmentului de mediu
2Eh	46 octeți rezervați
5Ch	FCB1 şi FCB2, câte 16 octeți pentru fişierele standard de intrare şi ieşire (azi evitați)
80h	**Lungimea cozii liniei de comandă**
81h	**Coada liniei de comandă (<127 octeți)**

Interfatarea aplicatiilor ASM cu SO



.COM - un segment, apeluri near, terminare 4Ch, INT 20h, RET
- nu necesita relocare, toate simbolurile relative sunt deplasamente in cadrul unui sg. unic

Interfatarea aplicatiilor ASM cu SO



.EXE – mai multe segmente

- necesita relocare, ajustarea instr. care cont. adr. segment

- JMP/CALL FAR/MOV reg, reg. seg/

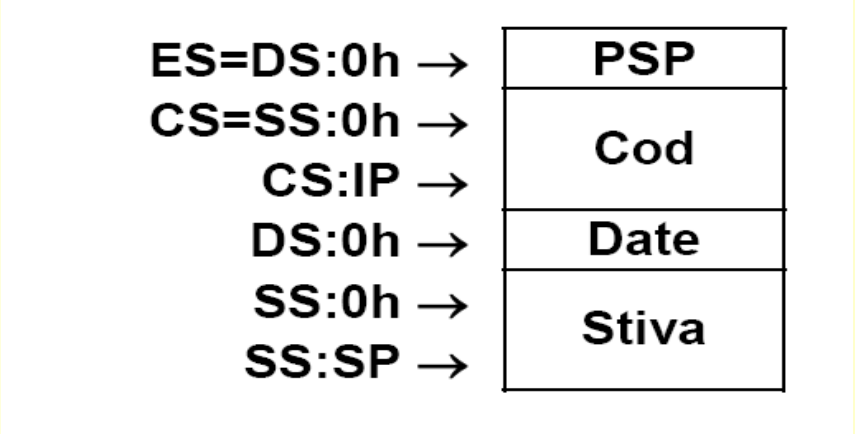
Interfatarea aplicatiilor ASM cu SO

```
    PR_COM SEGMENT PARA PUBLIC 'CODE'
    ORG 100h
    ASSUME CS:PR_COM,DS: PR_COM, SS: PR_COM, ES: PR_COM,

Start:    JMP inceput

          ; datele

inceput:
          ; proceduri
    ..............................
    ..............
    MOV AX,4C00H
    INT 21H
PR_COM ENDS
    END Start
```

Sablon program .COM

Interfatarea aplicatiilor ASM cu SO

```
STIVA       SEGMENT PARA STACK 'STACK'
    DW 512 DUP(?)
STIVA       ENDS
DATA        SEGMENT PARA      PUBLIC    'DATA'
    ;DATE
    ;       ..........................
DATA        ENDS

COD SEGMENT PARA PUBLIC 'CODE'
MAIN PROC FAR
    ASSUME CS:COD, DS:DATA, SS:STIVA,ES: NOTHING
    PUSH    DS      ; DS contine adr. Segment de la inceputul PSP
    XOR     AX,AX   ;AX=0
    PUSH    AX      ; salvez offsetul =0 ptr inceputul PSP
    MOV     AX,DATA ;
    MOV     DS,AX   ; DS poateaza inceputul seg. de date
.....................................
    RET
    ;proceduri
MAIN        ENDP
    COD     ENDS
    END     MAIN
```



Sablon program .EXE

De ce nu se mai folosesc serviciile BIOS/DOS în SO moderne?

- Sistemele de operare moderne (Windows, Linux, macOS pentru x86/x64) nu folosesc întreruperile BIOS/DOS din următoarele motive principale:
- **Modul de operare al procesorului:** SO-urile moderne rulează în **Protected Mode** (Mod Protejat) sau **Long Mode** (pe 64 de biți), care oferă acces la mai multă memorie (peste 4GB) și, esențial, **protecția memoriei**.
  Apelurile BIOS/DOS necesită ca procesorul să intre înapoi în **Real Mode**, un proces lent (16biti), ineficient și care compromite securitatea și stabilitatea sistemului.

- **Multi-Tasking:** Serviciile BIOS/DOS nu au fost concepute pentru medii multi-tasking. Ele sunt sincrone (blochează sistemul până la finalizare) și nu au mecanisme de gestionare a resurselor între mai multe programe.

- **Hardware Modern:** BIOS nu cunoaște sau *nu poate gestiona hardware-ul modern complex* (plăci video avansate, controlere SATA/NVMe, rețele).

Înlocuirea intreruperilor: mecanismele moderne

- Serviciile furnizate de întreruperile BIOS/DOS sunt înlocuite în SO moderne de următoarele mecanisme:

A. Apeluri de Sistem (System Calls)

- Acestea sunt interfața principală prin care o aplicație solicită servicii de la **nucleul (kernel-ul)** SO.
- **Cum funcționează:** Un apel de sistem este o solicitare de a executa o funcție privilegiată (cum ar fi accesul la un fișier sau rețea). În loc să folosească instrucțiunea INT, SO moderne folosesc mecanisme mult mai rapide și mai sigure:
  - Instrucțiuni specializate precum **SYSENTER/SYSCALL** (Intel/AMD) sau **INT 0x80** (în Linux mai vechi), care realizează o tranziție rapidă și controlată din **User Mode**(Mod Utilizator) în **Kernel Mode** (Mod Nucleu).
- **Ex:** În loc să folosești INT 21h (funcția DOS) pentru a citi un fișier, un program modern *apelează o funcție din API-ul SO* (de ex. ReadFile în Windows sau read în Linux) care, la rândul său, invocă apelul de sistem corespunzător.

B. Drivere de dispozitive (Device Drivers)

- **P**rograme specializate care rulează în **kernel mode** și asigură comunicarea directă între SO și o componentă hardware specifică.
- **Rolul lor:** Înlocuiesc complet rutinele generice de I/O din BIOS (INT13h, INT10h) Fiecare producător hardware oferă drivere proprii, permițând SO-ului să exploateze la maximum caracteristicile specifice ale dispozitivelor (ex: plăci grafice moderne, unități NVMe, SSD).

C. UEFI (Unified Extensible Firmware Interface)

- **Înlocuitorul BIOS-ului: UEFI** (sau, mai rar, **Legacy BIOS**) rămâne un strat software inițial care pornește sistemul. Însă, SO-urile moderne folosesc UEFI doar *pentru a inițializa* hardware-ul de bază și pentru a încărca *bootloader*-ul.
- **În timpul funcționării:** Odată ce nucleul SO-ului este încărcat, acesta preia controlul complet al hardware-ului și *nu mai folosește serviciile de rulare (runtime services) ale UEFI/BIOS* pentru I/O, bazându-se exclusiv pe propriile drivere și apeluri de sistem.

În concluzie, vechile întreruperi BIOS/DOS au fost înlocuite cu un set de interfețe stratificate, sigure și eficiente:

- ***UEFI/Firmware*** *pentru pornire,*
- ***Drivere*** *pentru gestionarea hardware-ului și*
- ***Apeluri de sistem*** *pentru interacțiunea aplicațiilor cu nucleul SO-ului.*

Un exemplu simplu de program în assembly x86 (pentru MASM) care creează o fereastră Windows folosind WinAPI.

- Programul afișează "Hello World!" într-un MessageBox și apoi închide aplicația. Este un GUI minimal, compilabil cu MASM32 sau Visual Studio.

Codul Sursă (hello_win.asm)

```
.386
.model flat, stdcall
option casemap:none

include windows.inc
include user32.inc
include kernel32.inc
includelib user32.lib
includelib kernel32.lib

.data
msgTitle  db "Assembly WinAPI", 0
msgText   db "Hello World din
Assembly!", 0

.code
start:
    invoke MessageBoxA, NULL, addr
msgText, addr msgTitle, MB_OK
    invoke ExitProcess, eax
end start
```

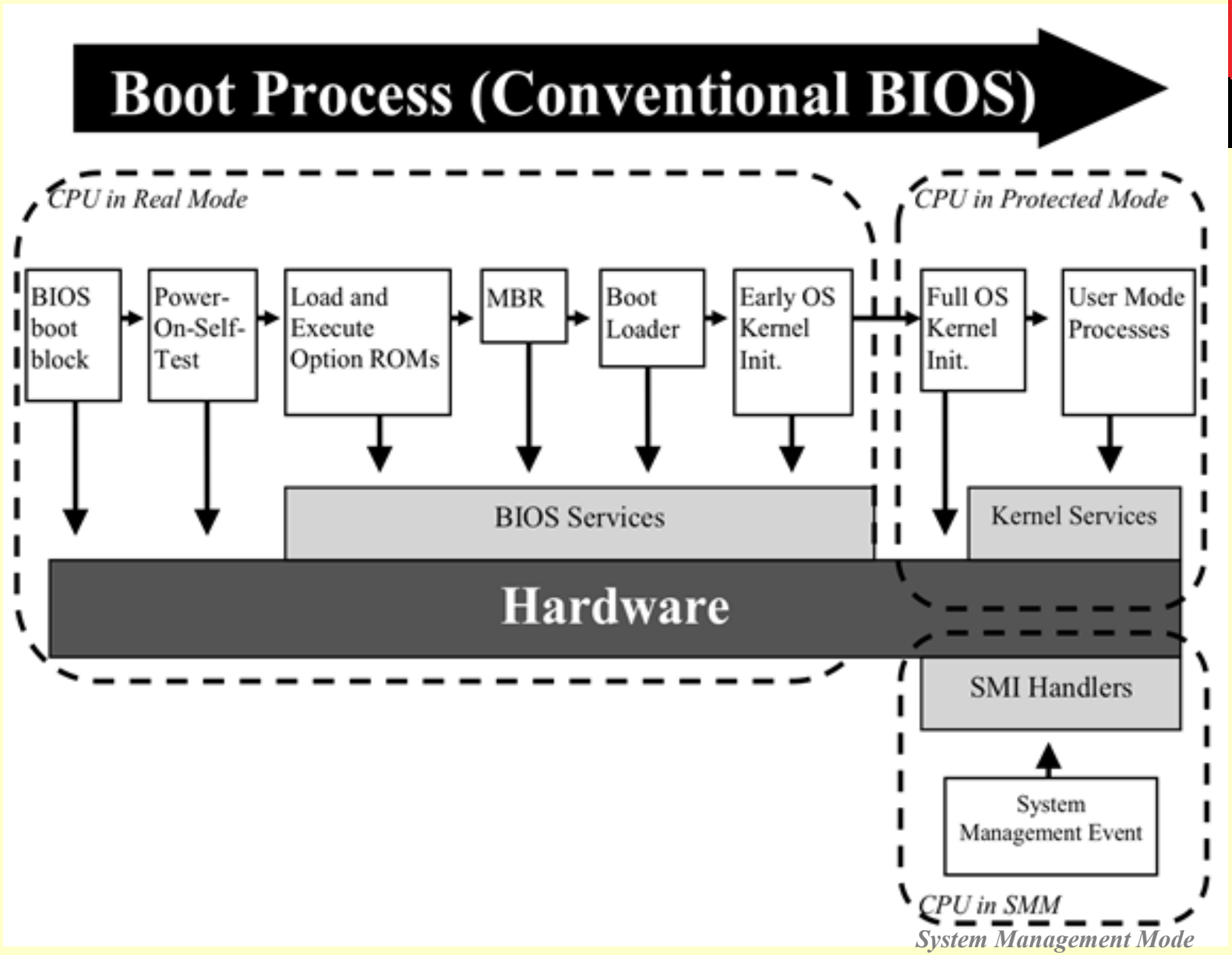
Compilare și Rulare
Compilează cu MASM: ml /c /coff hello_win.asm apoi link /subsystem:windows hello_win.obj user32.lib kernel32.lib.
Rulează hello_win.exe pe Windows x86 (32-bit).
Pentru x64, se adaptează cu MASM64 și registrele RCX/RDX etc. Acest cod înlocuiește complet BIOS/DOS cu apeluri native Win32.

Variante Ușoare
Pentru consolă: Adaugă include msvcrt.inc și includelib msvcrt.lib, apoi folosește printf în loc de MessageBox.

Pentru loop tastatură:
Integrează GetAsyncKeyState în buclă WM_PAINT pentru a citi taste și scrie text dinamic

3. UEFI – "Noul Bios"

Unified Extensible Firmware Interface



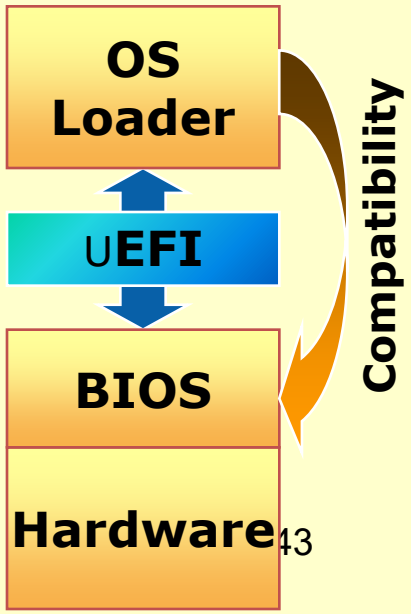
UEFI
Unified Extensible Firmware Interface

O noua tehnologie promite accelerarea pornirii PC-ului.

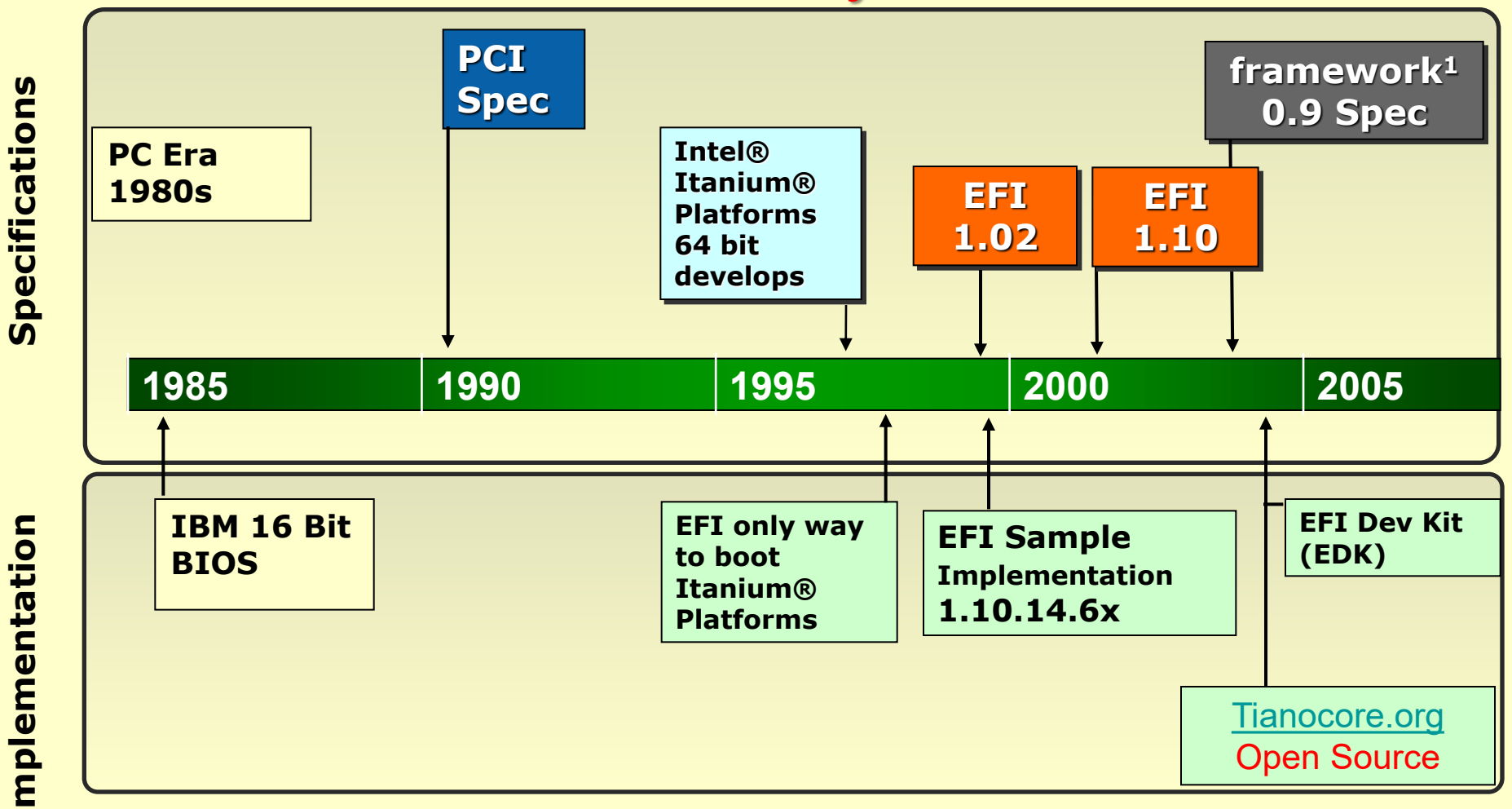
La DOS - inainte ca SO sa dea vreun “semn de viata” va apare tot timpul un ecran cu logo-ul producatorului placii de baza.
- In acest timp, BIOS-ul face numeroase verificari, detectand daca vreo componenta a PC-ului este defecta (POST).
- Problema BIOS-ului este ca :
BIOS-ul a fost conceput de mai bine de 40 de ani, iar in mare parte a ramas neschimbat (dezvoltat pe 16 biti)
- Din anul 2000 (1998) Intel a lucrat la o noua interfata firmware denumita EFI.

- Din 2005 a fost infiintat un organism independent, numit **Unified EFI Forum**, care gestioneaza specificatiile noului standard, denumit UEFI.
- Acest standard nu este sustinut doar de Intel, ci si de alte companii, cum ar fi AMD, Microsoft, Apple si Dell.

- UEFI poate rula deasupra traditionalului BIOS sau in locul lui.

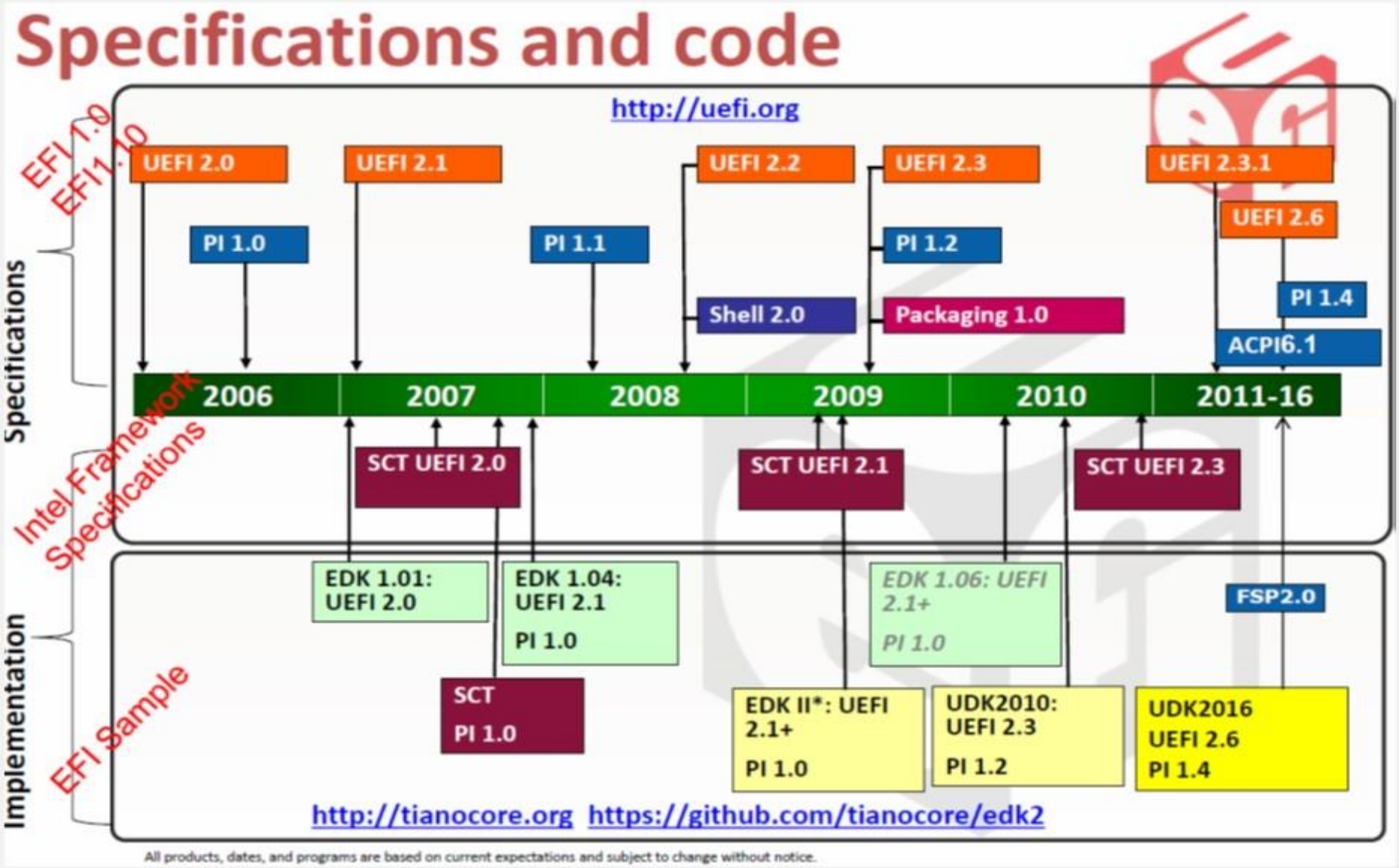


EFI / UEFI History Timeline

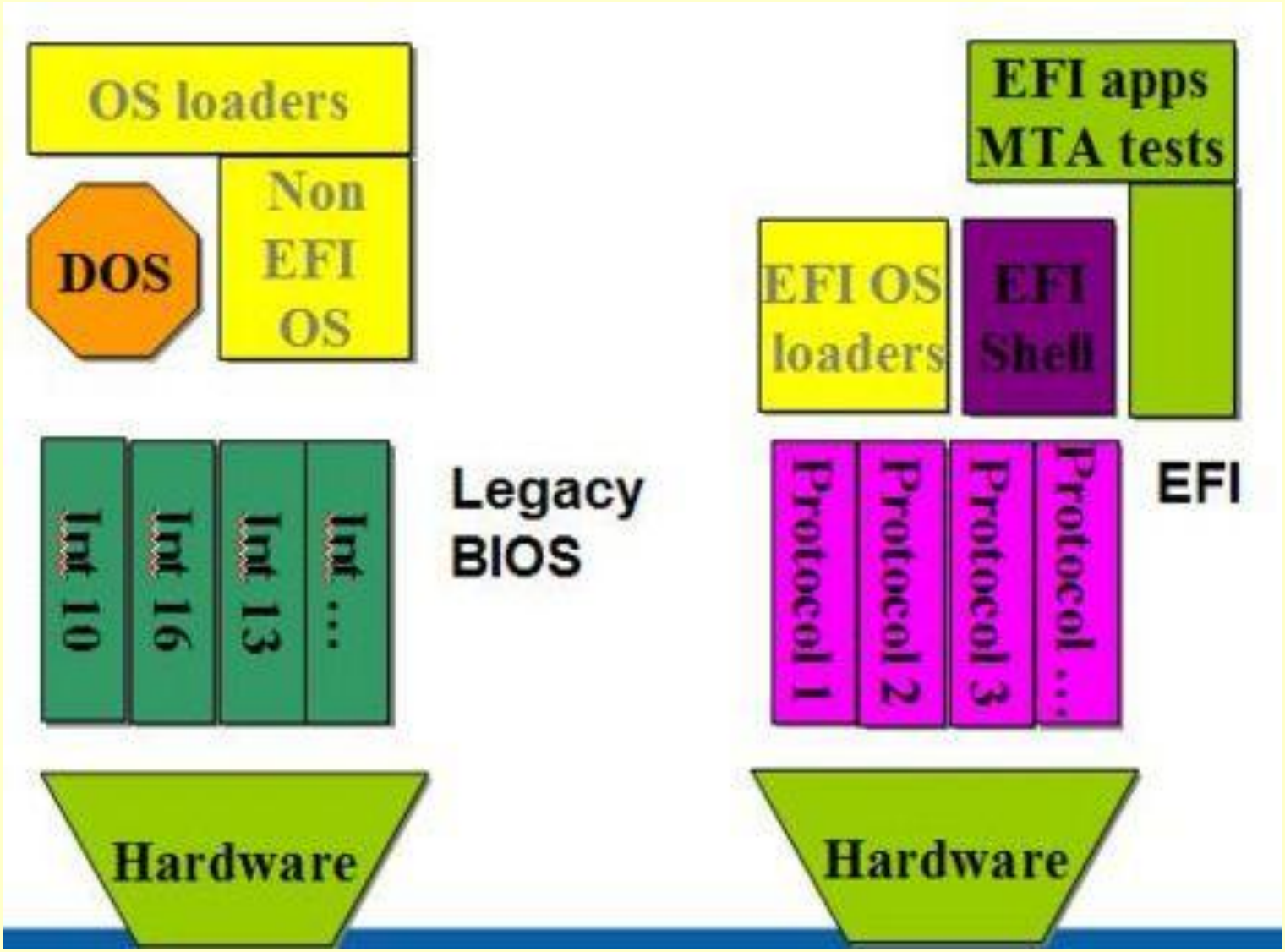


Open Source EFI Developer Kit (EDK) http://www.tianocore.Sourceforge.net
UEFI Specifications - http://www.uefi.org

Specifications and code

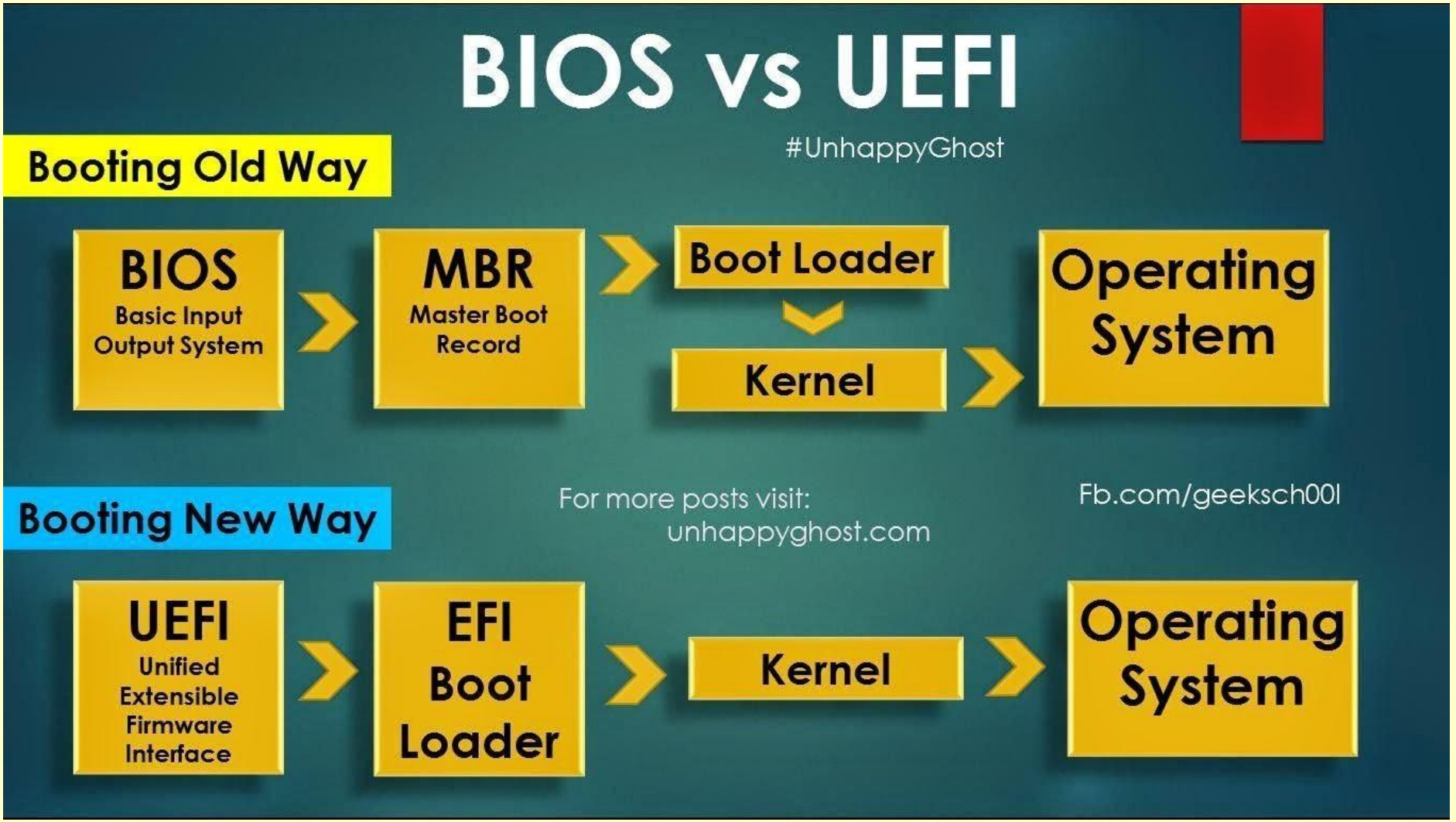


PI- platform initialization

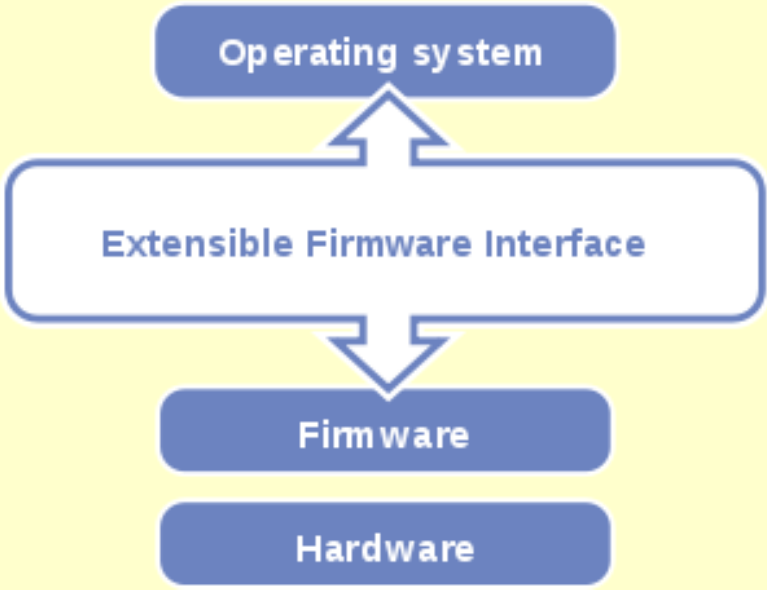


Analogie UEFI cu vechiul DOS: BIOS

http://www.uefi.org/learning_center/papers/


BIOS vs UEFI
#UnhappyGhost
Booting Old Way
BIOS
Basic Input Output System
MBR
Master Boot Record
Boot Loader
Kernel
Operating System
Booting New Way
For more posts visit:
unhappyghost.com
Fb.com/geeksch00l
UEFI
Unified Extensible Firmware Interface
EFI Boot Loader
Kernel
Operating System

Avantajele UEFI

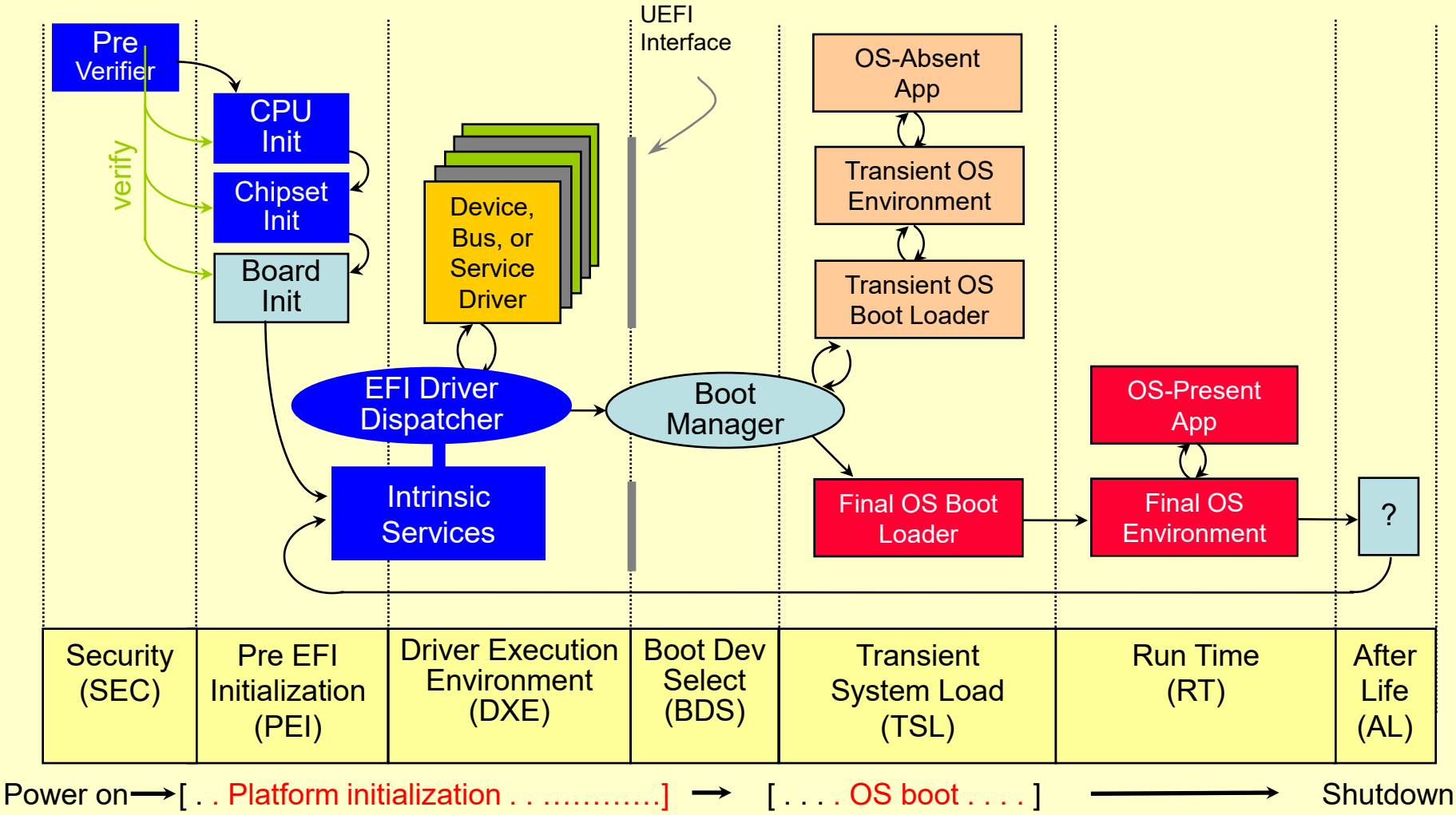


- Unul dintre marile avantaje cu care cocheteaza Intel este **viteza mai mare fata de cea a BIOS-ului**. BIOS-ul functioneaza ca un strat care face legatura dintre componentele hardware si SO.
- BIOS-ul cauta primii 512B dupa un boot-loader. In cazul HDD-urilor, acesti bytes se afla in MBR-ul (master boot record), care initializeaza SO

- Unul dintre *marile dezavantaje ale BIOS-ului este faptul ca se bazeaza pe instructiuni in limbaj de asamblare (ASM) pe 16 biti si nu poate accesa in mod direct componentele mai noi pe 64 de biti.*
- Cealalta problema este aceea ca *nu exista un standard fizic ptr. componentele mai noi pe 64 de biti*, astfel incat producatorii vin cu modificari proprii.
  Intel doreste rezolvarea acestor probleme cu ajutorul UEFI.

- Software (programe/date) care a fost scris în memoria nevolatila (ROM)
- Firmware-ul este o combinație de software și hardware. ROM-urile, PROM-urile, Flash-urile și EPROM-urile pe care sunt înregistrate date/programe sunt firmware.

Etapele bootarii prin UEFI



Boot Execution Flow

https://resources.infosecinstitute.com/uefi-and-tpm/

Etapele initializarii prin UEFI

- ***1. Etapa SEC*** este prima faza in arhitectura PI (platform initialization)
  - contine ***codul de inceput care este executat de CPU***, este în principiu acela moștenit
  - este cod minimal dezvoltat in asamblare fiind *dependent de arhitectura si nu este portabil*
  - se executa direct din flash, este necomprimat
- ***2. Pre-EFI Initialization (PEI),*** prin care este initializat ***procesorul, memoria si chipset-ul (placa).***
- ***3. Driver Execution Environment (DXE).***

In acest punct are loc ***pornirea in paralel a restului componentelor hardware***.

- Pe parcursul acestei etape apare si primul spor de viteza: UEFI ***integreaza in mod direct toate driverele,*** de aceea in momentul pornirii SO, acestea nu mai trebuie sa fie incarcate. Pe langa sporul evident de viteza, aceasta etapa poate aduce si un spor de fiabilitate, ***driver-ele functionand pe toate SO***, astfel ca problema drivere-lor pe distributii Linux va fi de domeniul trecutului.?!
- Alt avantaj al initializarii mai rapide al drivere-lor poate fi faptul ***ca interfetele pentru UEFI vor putea fi mai sofisticate,*** placa grafica putand fi folosita la intreg potentialul si se poate realiza chiar o conexiune la internet, datorita drivere-lor pentru placa de retea.

4. ***Boot Device Select*** - UEFI nu va cauta bootloader-ul pe toate discurile, deoarece ***discul de boot va fi setat in prealabil la instalarea UEFI***, prin aceasta metoda economisindu-se un anumit timp.

- Una dintre cele mai utile inovatii aduse de UEFI este ***crearea unei partitii EFI,*** pe care se vor putea stoca ***aplicatii de diagnosticare a PC-ului*** sau chiar ***aplicatii antivirus*** ce vor rula, chiar daca SO nu porneste.

- UEFI fiind o tehnologie noua va oferi suport pentru componentele hardware de ultima generatie. Pana acum, spatiului maxim de stocare ce putea fi accesat era de **2 TB** (2^{32} de sectoare de cate 512 B), asta din cauza limitarii BIOS-ului. UEFI lucreaza cu ajutorul **GPT** (GUID Partition Table), care are adrese pe 64 de biti, ceea ce inseamna ca poate adresa pana la **9 ZB** (1zettabyte ~ 1 miliard de terabytes). Suportul pentru **GPT** exista inca de la Windows Vista SP1 x64.

5. ***Etapa TSL (Transient System Load)*** - este punctul în care ***renunțăm la codul firmware***, ptr. codul stocat de obicei pe HDD. Dacă sistemul rulează cu SecureBoot activat, BDS va verifica semnătura înainte de a încărca codul în această fază și impiedică orice cod ne-semnat sa se incarce

6. ***Etapa RT (Run Time)*** **-** În mod obișnuit, atunci când ***boot loader-ul SO termină***, se va apela ExitBootService. Astfel, se va recupera majoritatea memoriei UEFI, astfel încât SO să o poată utiliza, chiar dacă este reținută o anumită memorie pentru a fi folosită pentru tabelul de servicii Runtime.

7. ***Etapa AL (After Life) -*** este pusa la sfarsit, formal astfel încât, din punct de vedere arhitectural, a avea opțiunea de a face “diverse operatii" inainte de oprire

Advantages of UEFI vs. BIOS

Interface	Legacy BIOS	UEFI
Architecture	x86 / X64 only	Agnostic
Mode	16 bit (real mode)	32/64 bit
Boot Partition	MBR (2.2 TB limit)	GPT (9.4 ZB* limit)
Runtime Services	No	Yes
Driver model	No	Yes
POST Graphics	VGA	Graphical Output Protocol (GOP)

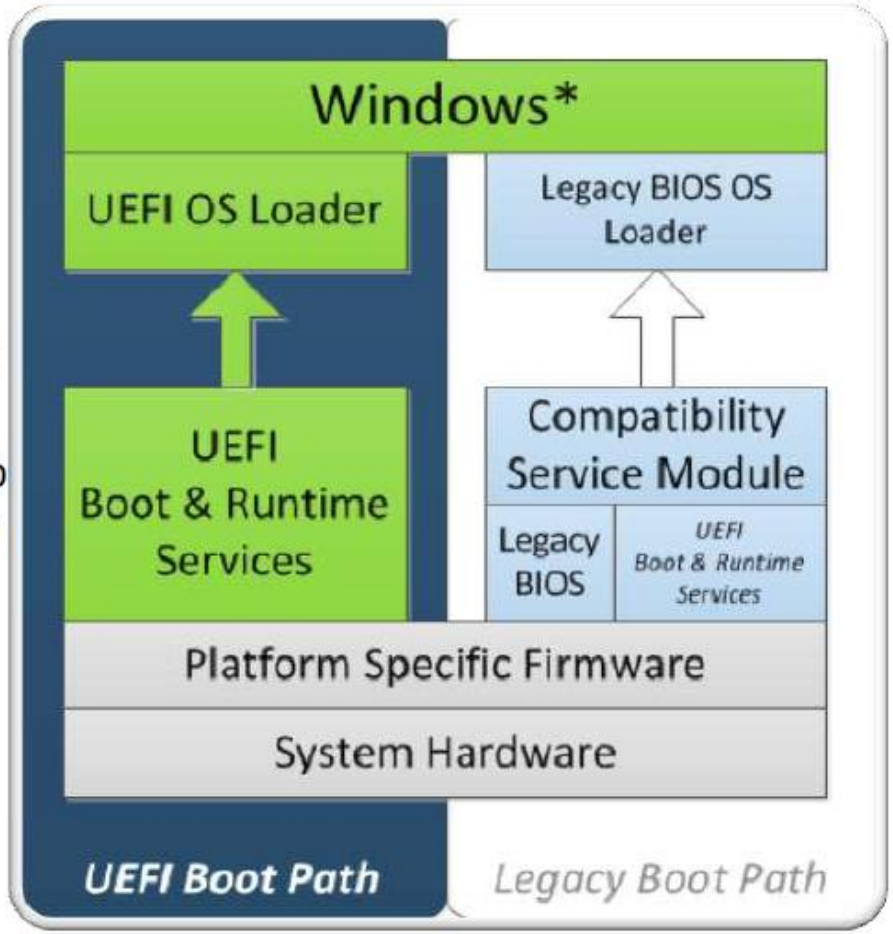
** A zettabyte is equal to 1B terabytes. The total amount of global data was expected to pass 1.2 ZB sometime during 2010.*

https://software.intel.com/en-us/articles/uefi-introduction

- Forumul UEFI defineste 4 tipuri de platforme (“clase”) bazate pe prezenta CSM (compatibility service module)

Windows 8 Boot Flow

- Windows 8 installs UEFI OS Loader if UEFI is detected
- Most PCs today boot through CSM path
- For compatibility the CSM boot path available
- Windows 8 logo requirement to boot UEFI only (cannot run csm in client builds)
- Client must boot with UEFI secure boot enabled
- For server if implemented



Windows 8 Certification Requirements – UEFI Boot

Boot Performance Requirements