# Compilation du Noyau et Séquence de Boot (bootstrap)

# Chargement et Exécution du Noyau

- Les contraintes initiales :
  - Le BIOS sait trouver et charger un secteur de boot
    - Un programme de 512 octets seulement (en assembleur)
      - Pas suffisant pour tout faire
  - Le BIOS offre aussi quelques primitives **basiques**
    - Chargement de secteurs dans le mémoire
  - Le BIOS ne fonctionne qu'en mode d'adressage réel
    - Pas suffisant : il faudra s'en passer ...
      - Pour utiliser la mémoire en mode protégé
- Comment démarrer avec ça ?
  - Un unique fichier contient tout ce qu'il faut
    - bzImage (ou vmlinuz)
    - En fait, un assemblage de **plusieurs** programmes !

# Contenu du Container du Noyau

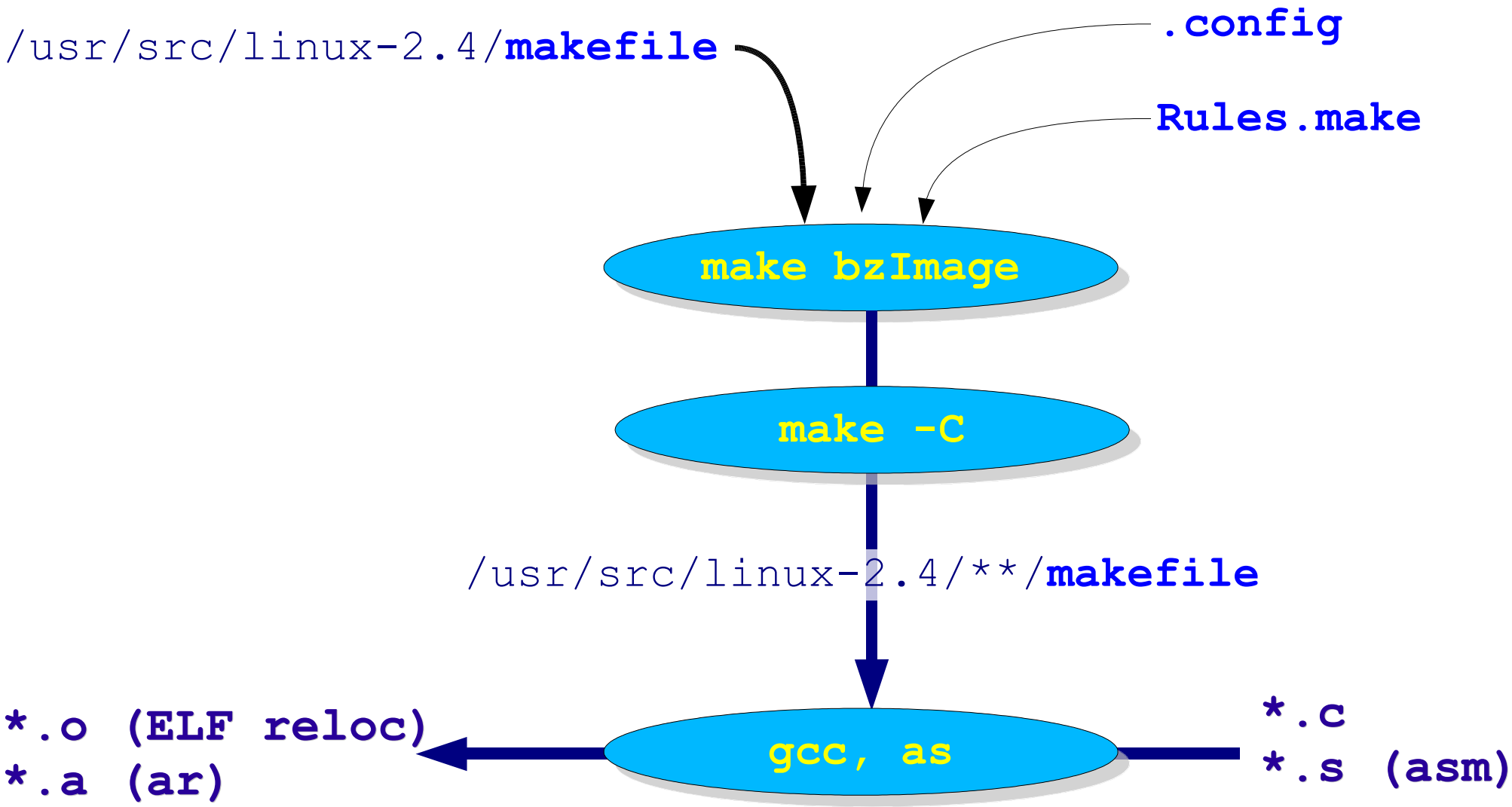
- bzImage contient :
  - Un secteur de boot (amorce)
  - Un programme de préparation : setup
  - Le noyau sous forme comprimée
    - En fait double programme qui s'auto-décomprime
      - D'abord un petit programme avec beaucoup de données
        - Programme : Routine de décompression
        - Données : Le code du noyau comprimé
    - L'image décomprimée est un **gros** programme
      - **bz**Image :
        - b= big : > 1 Mo
        - Pb : Pas assez de place en espace d'adressage reel
          - transit obligatoire par la mémoire haute ...

# Principe du Chargement

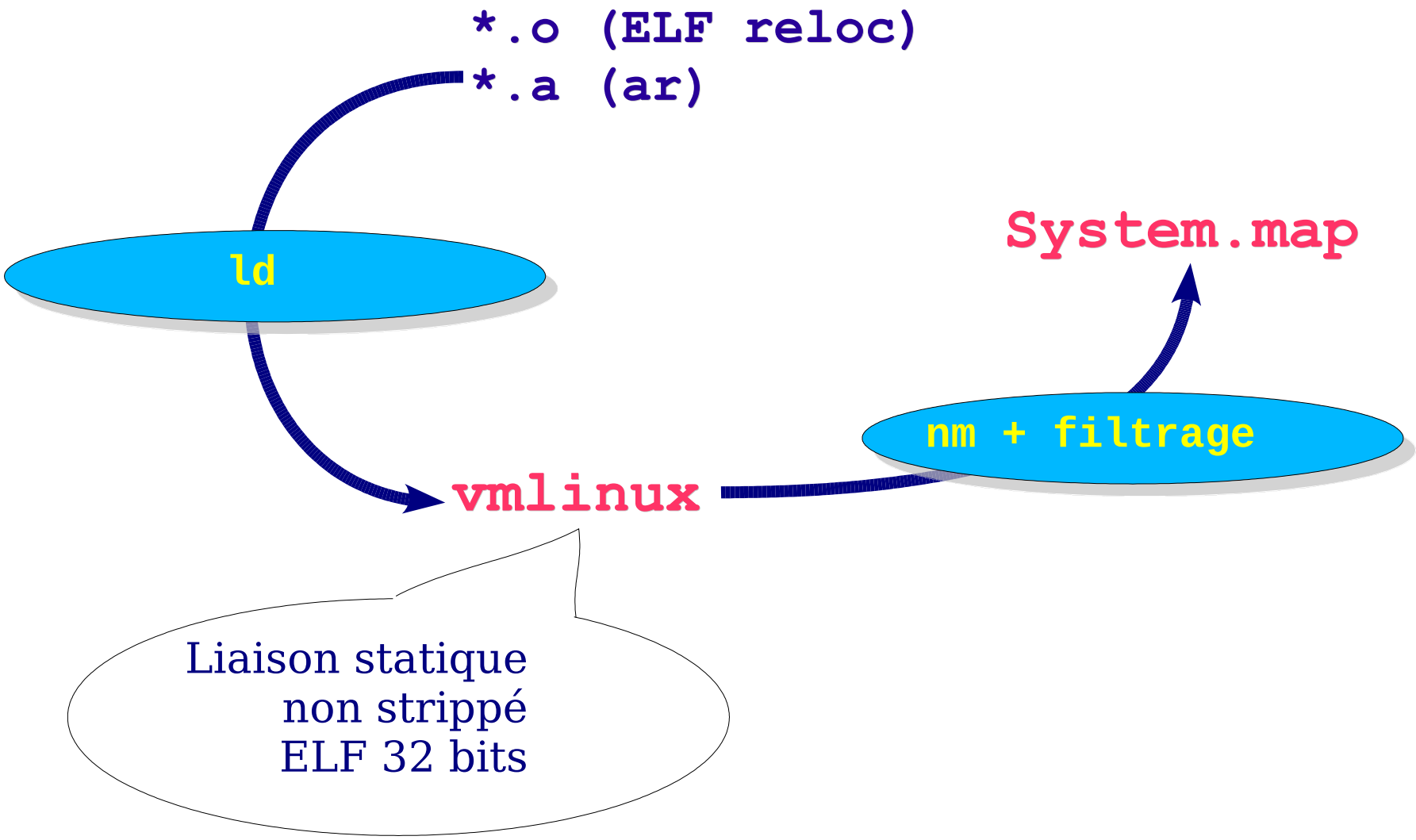
- Secteur de boot charge un (petit) programme
  - Le "setup"
    - Prépare le terrain pour l'exécution du noyau
- Mission du programme du secteur de boot :
  - Programmer appels au BIOS
    - chargement setup
    - chargement du noyau comprimé
- Mission du programme setup
  - Initialisations
    - Se préparer à "couper le cordon" avec le BIOS
  - Passage en mode d'adressage protégé (plus d'espace !)
    - Config mémoire pour que le noyau s'y retrouve une fois lancé ...

# I - Compilation du Noyau

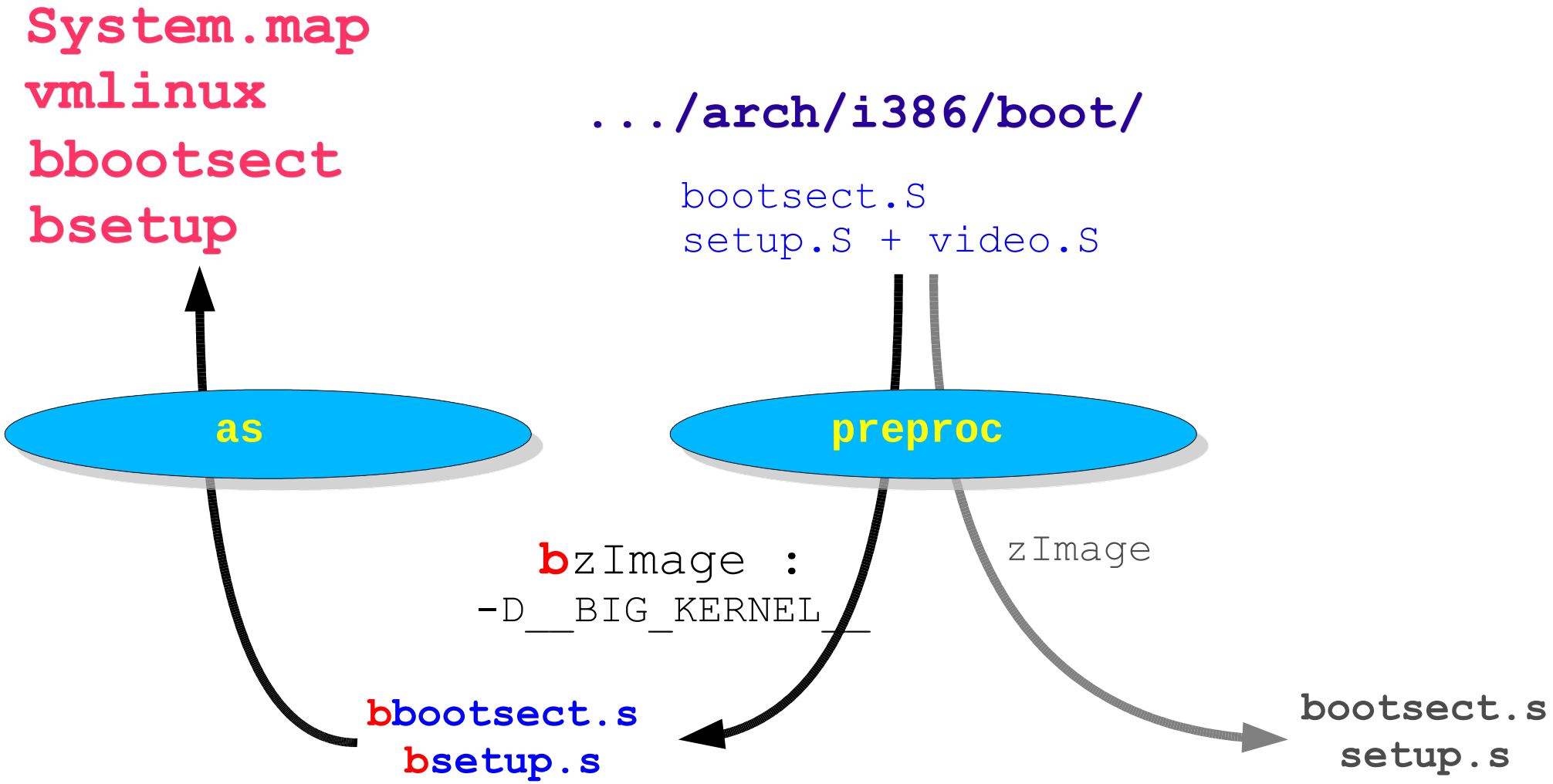
# Makefiles


/usr/src/linux-2.4/makefile
.config
Rules.make
make bzImage
make -C
/usr/src/linux-2.4/**/makefile
*.o (ELF reloc)
*.a (ar)
gcc, as
*.c
*.s (asm)

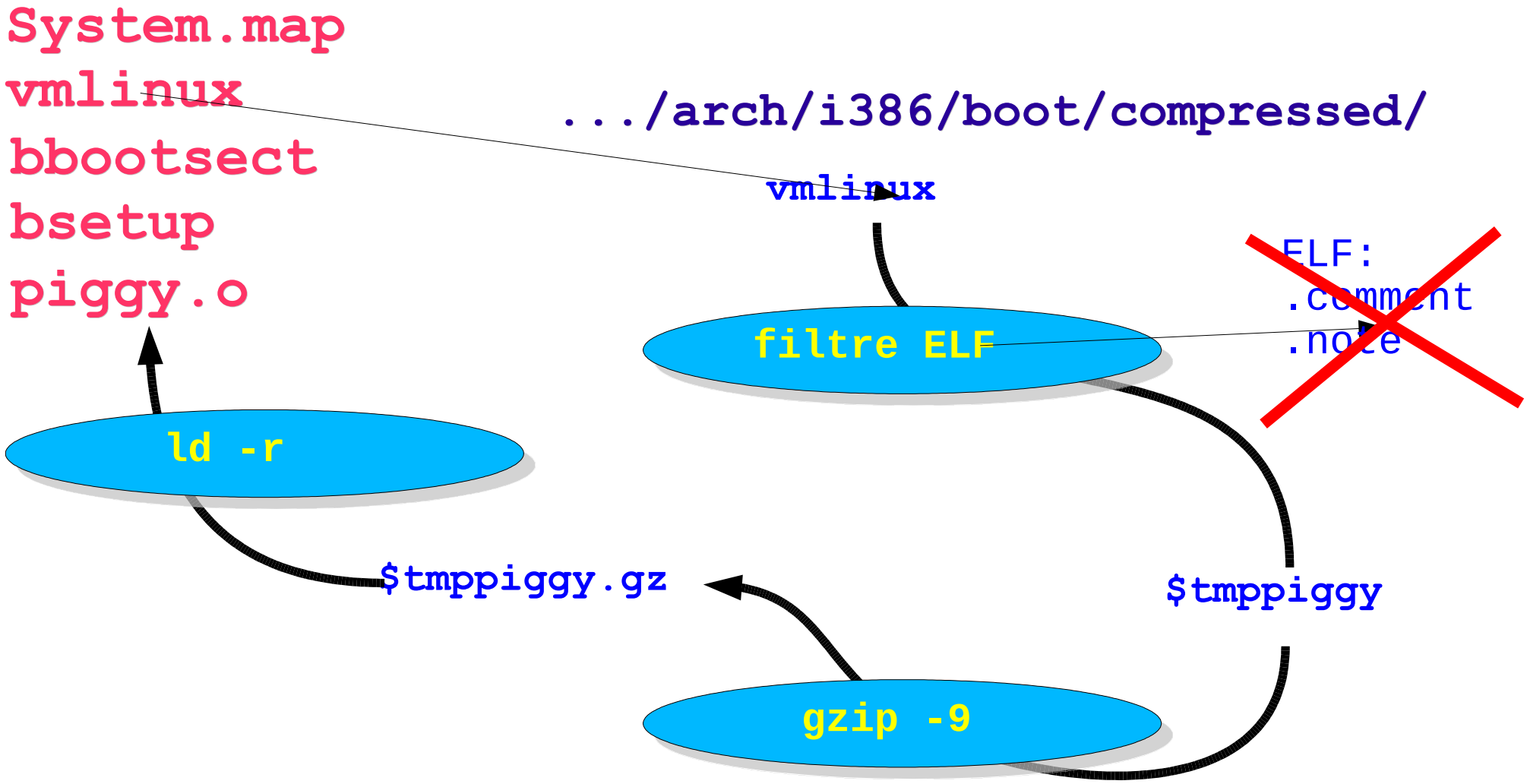
# Fichier Objet et Map Symboles



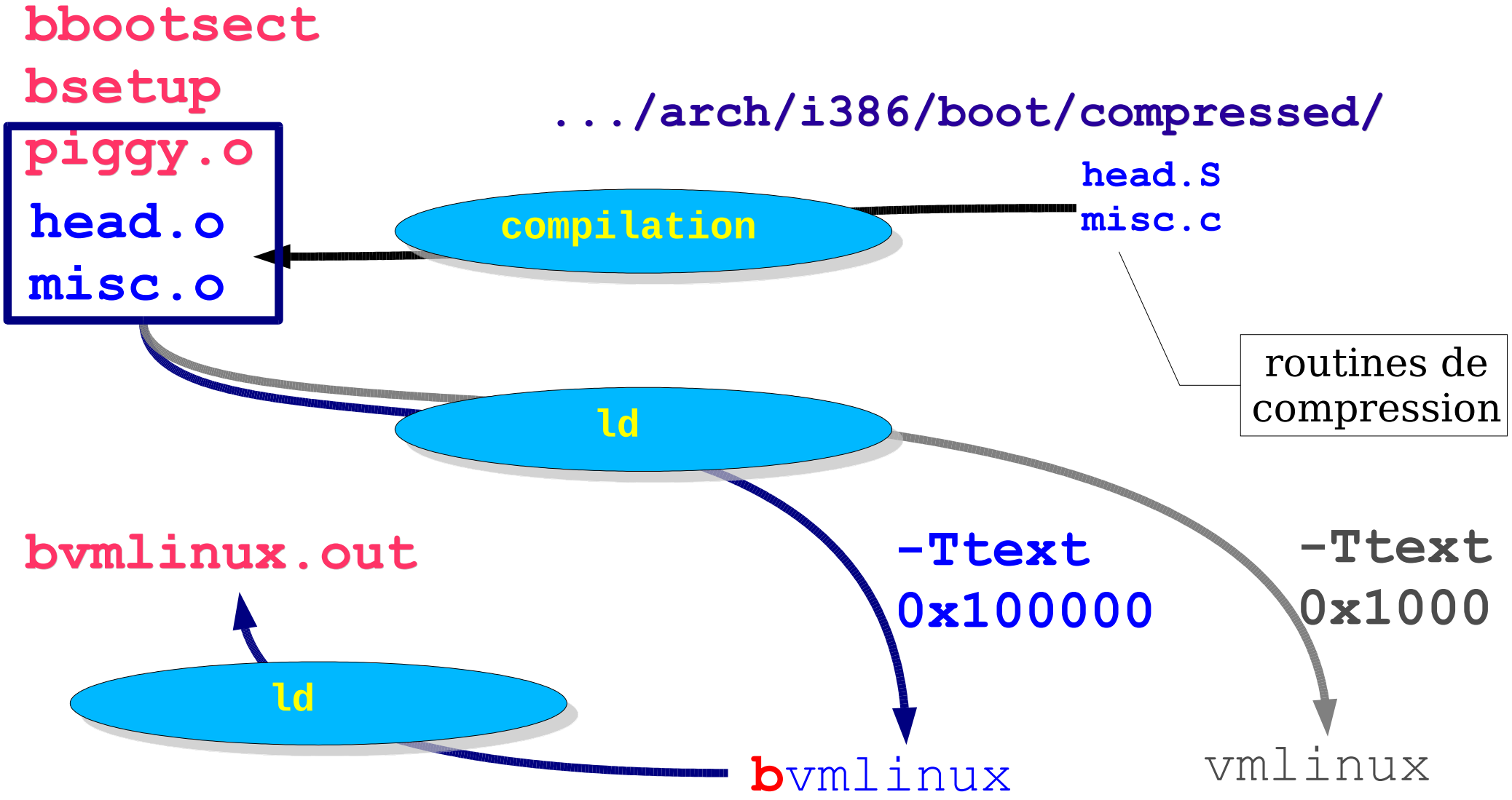
# Compilation du Chargeur



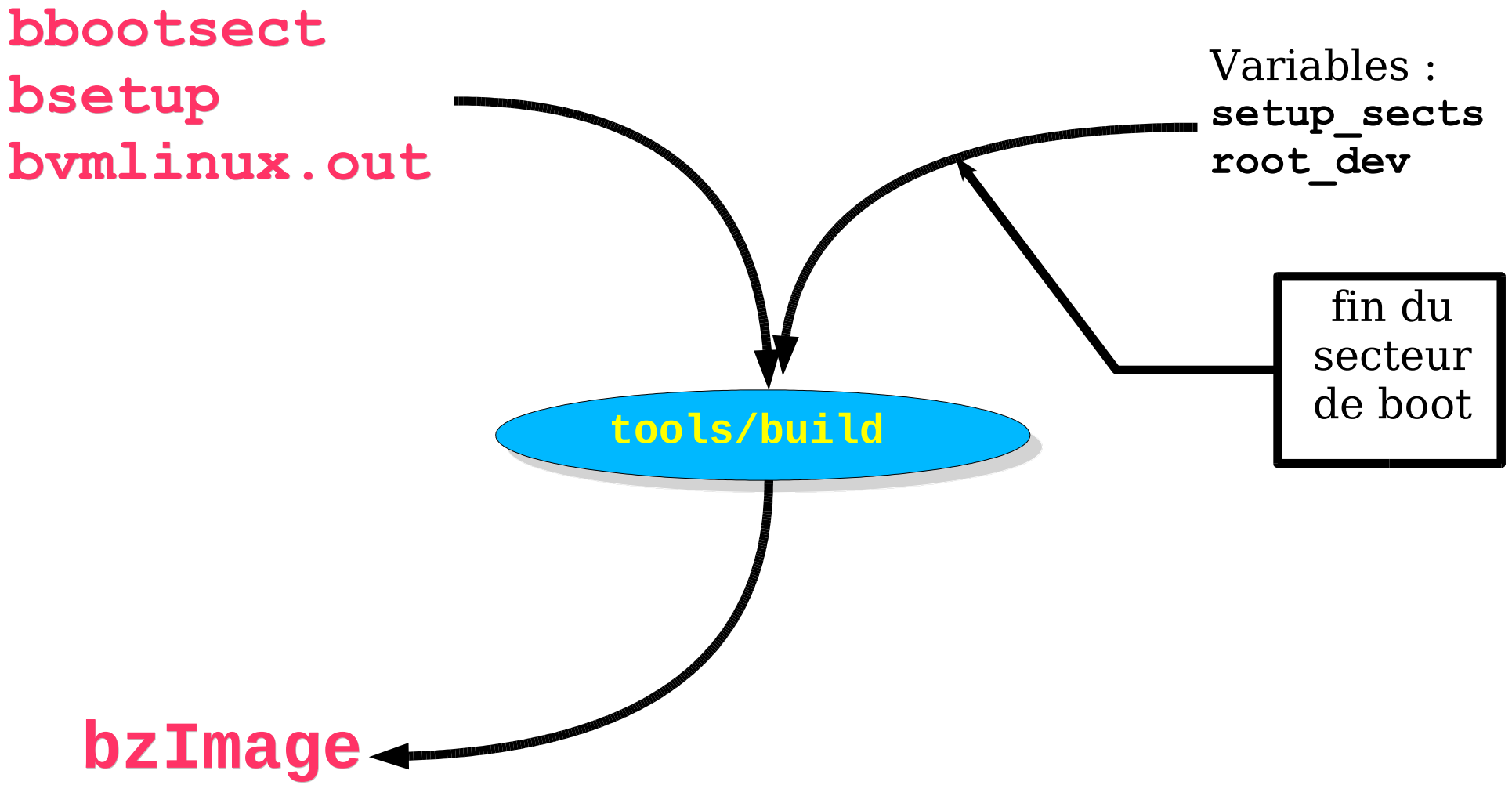
# Liaison et Compression



# Liaison du Noyau et du Chargeur



# Assemblage Final



# A propos de Compilation : le Makefile Principal

- Utile : la déclaration **`EXTRAVERSION = truc`**
  - Permet de faire cohabiter (compiler/installer) plusieurs fois la même version du noyau
  - Installation des modules **`/lib/modules/2.X.Ytruc`**
- Principales cibles :
  - **`oldconfig`**
  - **`config/menuconfig/xconfig`**
  - **`dep`**
  - **`clean/mrproper`**
  - **`bzImage/modules`**
  - **`install/modules_install`**
  - **`psdocs/pdfdocs/htmldocs`**
  - **`mandocs (=> /usr/share/man/man9 + makewhatis)`**

# II – La Séquence de Boot (bootstrap)

# Résumé

1) Le BIOS séléctionne le périphérique de boot
2) Le BIOS charge le secteur de boot
3) Le secteur de boot charge
   - setup,
   - routines de decompression
   - l'image du noyau
4) Le noyau est décomprimé en mode protégé
5) Initialisation de bas niveau (code asm)
6) Initialisation haut niveau en C

# La séquence de boot : POST du BIOS

Le BIOS déclenche les Power On Self Test :

1) Alim -> lance horloge : attente signal **#POWERGOOD** sur le bus.
2) CPU **#RESET** (CPU en mode réel 8086).
3) **%ds=%es=%fs=%gs=%ss=0, %cs=0xFFFF0000,%eip = 0x0000FFF0** (ROM BIOS POST).
4) Tous les POST sont lancés avec les interruptions désarmées
5) IVT (Interrupt Vector Table) initialisée avec adresse 0.
6) Appel fonction 'Bootstrap Loader' du BIOS :
   Fonction invoquée via int 0x19 (%dl= num périph. boot)
   ➔chargement piste 0, secteur à l'adr phys. **0x7C00** (**0x07C0:0000**).

# La séquence de boot : le secteur de boot

Le secteur de boot utilisé peut venir :

- de Linux (arch/i386/boot/bootsect.S),
- de LILO (ou un autre chargeur)
- pas de secteur de boot (loadlin etc)

Fichier bootsect.S (début) :

```
29 SETUPSECS = 4                /* default nr of setup-sectors */
30 BOOTSEG   = 0x07C0           /* original address of boot-sector */
31 INITSEG   = DEF_INITSEG      /* we move boot here - out of the way */
32 SETUPSEG  = DEF_SETUPSEG     /* setup starts here */
33 SYSSEG    = DEF_SYSSEG       /* system loaded at 0x10000 (65536) */
34 SYSSIZE   = DEF_SYSSIZE      /* system size: # of 16-byte clicks */
```

boot.h

0x9000

0x9020

0x1000

# La séquence de boot : le secteur de boot

## Fichier bootsect.S (suite) :

```
54              movw    $BOOTSEG, %ax
55              movw    %ax, %ds
56              movw    $INITSEG, %ax
57              movw    %ax, %es
58              movw    $256, %cx
59              subw    %si, %si
60              subw    %di, %di
61              cld
62              rep
63              movsw
64              ljmp    $INITSEG, $go

70      go:     movw    $0x4000-12, %di         # 0x4000 is an arbitrary value >=
71                                              # length of bootsect + length of
72                                              # setup + room for stack;
73                                              # 12 is disk parm size.
74              movw    %ax, %ds                # ax and es already contain INITSEG
75              movw    %ax, %ss
76              movw    %di, %sp                # put stack at INITSEG:0x4000-12.
```

déplace le code du secteur de boot de 0X7C00 à 0X90000

256 mots x 2o = 512o

Saute dans la copie (0x90000) pour y préparer la pile

# La séquence de boot : le secteur de boot

## Fichier bootsect.S (suite) :

```
107  load_setup:
108          xorb    %ah, %ah                # reset FDC (service 0)
109          xorb    %dl, %dl
110          int     $0x13
111          xorw    %dx, %dx                # drive 0, head 0
112          movb    $0x02, %cl              # sector 2, track 0
113          movw    $0x0200, %bx            # address = 512, in INITSEG
114          movb    $0x02, %ah              # service 2, "read sector(s)"
115          movb    setup_sects, %al        # (assume all on head 0, track 0)
116          int     $0x13                   # read it
117          jnc     ok_load_setup           # ok - continue

118          pushw   %ax                     # dump error code
119          call    print_nl
120          movw    %sp, %bp
121          call    print_hex
122          popw    %ax
123          jmp     load_setup

124  ok_load_setup:
```

Chargement des secteurs de setup à la suite du secteur de boot (0x90200)

# La séquence de boot : chargement noyau

- Après chargement des **setup_sects** secteurs de setup :
  - Chargement image noyau (compr) en **0x10000** (**zImage**) ou **0x100000** (**bzImage**)
    - ➔Préserve données firmware entre 0 et 64k (pour les appels au BIOS)
  - Saut en **$SETUPSEG:0** (code de **setup.S**)
    - Init RAM, video, géométrie disque, …
    - zImage : Copie image (compr) de **0x10000** en **0x1000**
      - ➔ car BIOS n'est plus utilisé
    - Init Interruptions, FPU, …
    - config mode protégé
    - saut en **0x1000** ou **0x100000** (début noyau compr.)

# La séquence de boot : décompression

- Début noyau compressé :
  - **startup_32()**
    - en **arch/i386/boot/compressed/head.S**
  - Début exécution en mode protégé (32 bits)
    - Config Pile
    - Appel **decompress_kernel()**
      - Décomprime le noyau en **0X100000**
      - Saut en **0x100000** (début noyau décomp)
      - **startup_32()** en **arch/i386/kernel/head.S**
- Cas des gros noyaux (bzImage)
  - lecture et copie en mem haute par paquets de 64k
    - routine **bootsect_helper (.../setup.S)**
    - label **bootsect_kludge (.../bootsect.S)**

# La séquence de boot : init haut niveau

- Début noyau décompressé (head.S, en `0x100000`)
  1) Initialise segments
  2) Initialise les tables de pages
  3) Active pagination (PG dans reg `%cr0`)
  4) Efface BSS (zéros)
  5) Copie 1ers 2ko des paramètres de boot
     - kernel command-line
  6) Vérif type CPU (EFLAGS + cpuid)
  7) 1er CPU appelle `start_kernel()`

     (et les autres `smpboot.c:initialize_secondary()` )
     - Initialise tous les composants du noyau
     - fork init