The ERESI Reverse Engineering Software Interface

ERESI :
une plate-forme d'analyse binaire au niveau noyau

The ERESI team

http://www.eresi-project.org

- **ERESI: quelques rappels**
  - 7 années de développement
  - Plusieurs contributeurs
  - 5 logiciels, 14 bibliothèques
  - GPL : Intégré dans différentes distributions

Le reverse engineering: Différents besoins

- Audit de vulnérabilités
- Compatibilité
- Propriété intellectuelle

Analyse de programmes:

- Statique:
  - Structure du binaire
    - Symboles, sections, point d'entrée ...
  - Représentation logique
    - Graphe d'exécution, dépendances ...
- Dynamique:
  - Debug, Traces
  - Analyse des données E/S.

The ERESI Reverse Engineering Software Interface

La Plateforme Eresi:

Un ensemble d'outils:

- Logiciels scriptables.
- Bibliothèques et API.
- Plusieurs architectures: Ia32, Sparc, Mips
- Plusieurs plateformes : Linux, Bsd, Solaris

Évolution du projet Eresi.

Les premiers composants : le projet Elfsh

- Libelfsh : extraction ELF
- Libasm : Désassemblage et typage multiarchitecture
- Libmjollnir : Graphe d'exécution, empreintes de fonctions

Évolution du projet Eresi

Vers une interface de scripting.

- Librevm : interpreteur de commandes, types (annotation), session
- Libaspect : vecteurs (modularité, portabilité), configuration
- Libstderesi : abstraction et accès au framework Eresi

Évolution du projet Eresi.

De l'analyse statique à l'analyse dynamique

- Libedfmt : extraction des informations de debug
- Liballocproxy : allocation mémoire autonome
- Libetrace/Libe2dbg : abstraction de ptrace

Le scripting haut niveau

Plusieurs logiciels scriptables.

Langage riche.

Accès aux structures internes.

Un exemple de scripting

```
set $num 1.sht.num
foreach $elem of 0 until $num
  print 1.sht[$elem].size
forend
```

De nombreux exemples sur le site Eresi ;)

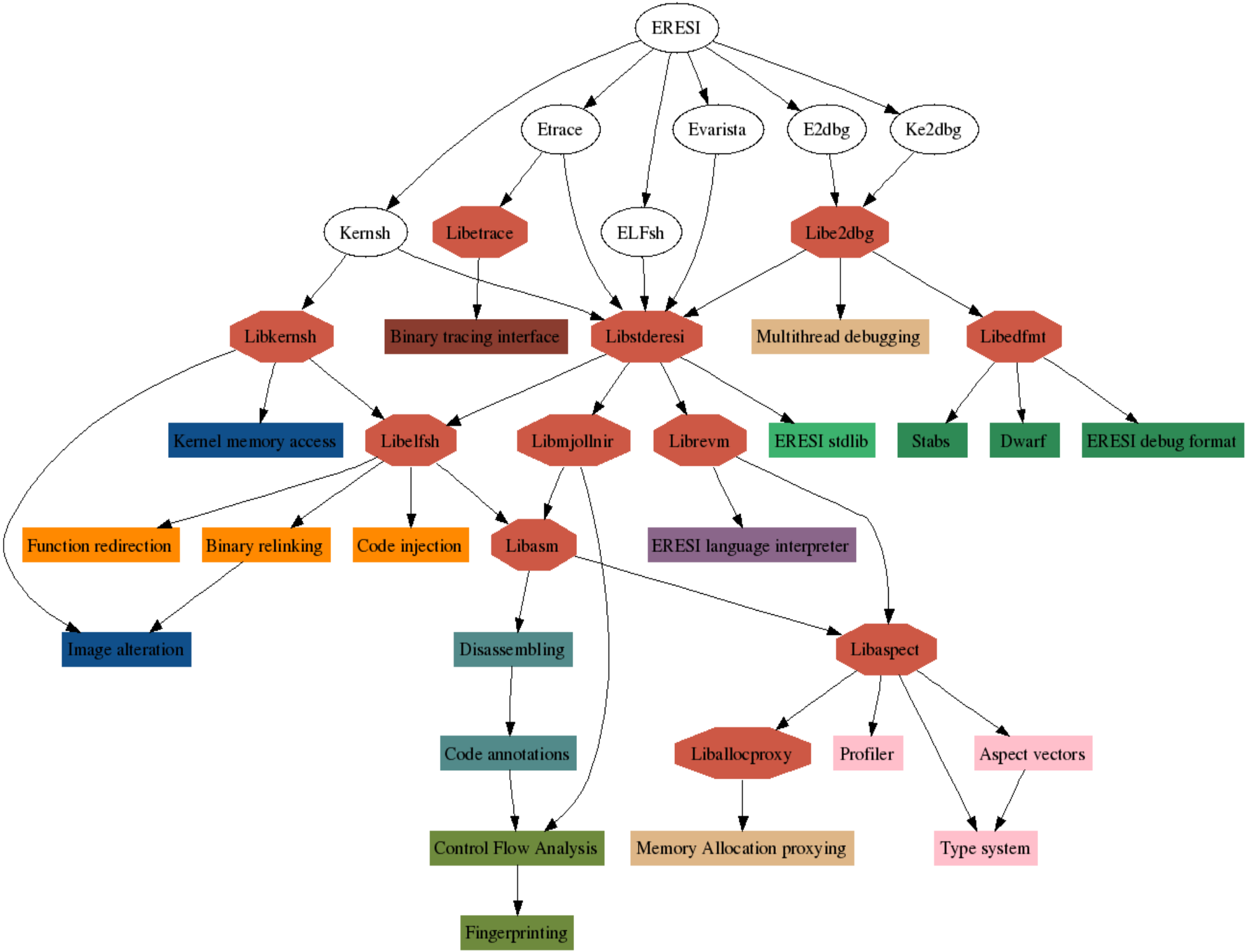
Les outils du framework Eresi

Elfsh : Un logiciel de manipulation de binaires Elf

Etrace/E2dbg : Des outils d'analyse dynamique.

Pas d'utilisation de ptrace.

Des performances accrues : pas de changement de contexte.

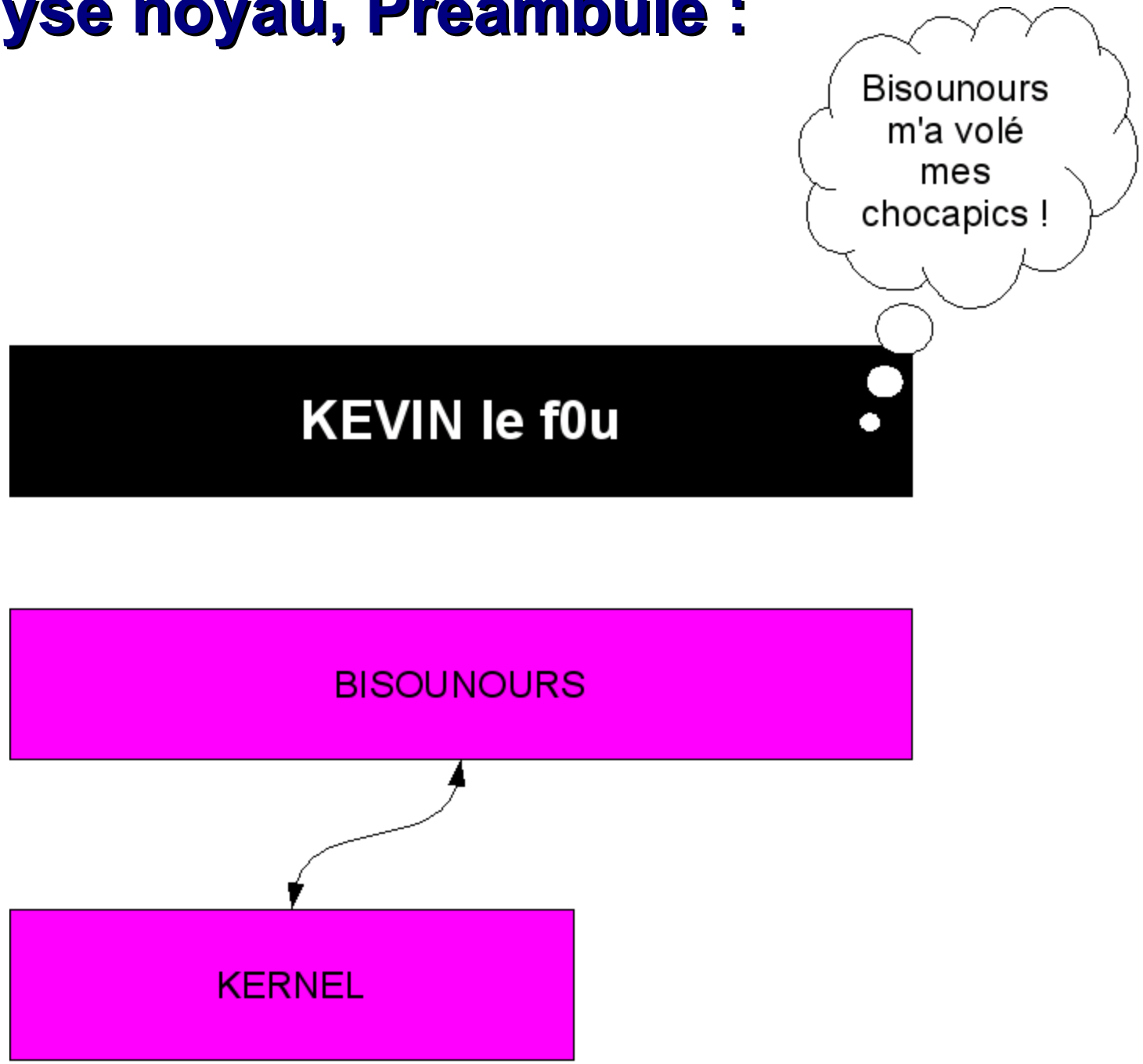

ERESI
Etrace
Evarista
E2dbg
Ke2dbg
Kernsh
Libetrace
ELFsh
Libe2dbg
Libkernsh
Binary tracing interface
Libstderesi
Multithread debugging
Libedfmt
Kernel memory access
Libelfsh
Libmjollnir
Librevm
ERESI stdlib
Stabs
Dwarf
ERESI debug format
Function redirection
Binary relinking
Code injection
Libasm
ERESI language interpreter
Image alteration
Disassembling
Libaspect
Code annotations
Liballocproxy
Profiler
Aspect vectors
Control Flow Analysis
Memory Allocation proxying
Type system
Fingerprinting

Eresi et le noyau :

De nouveaux composants:

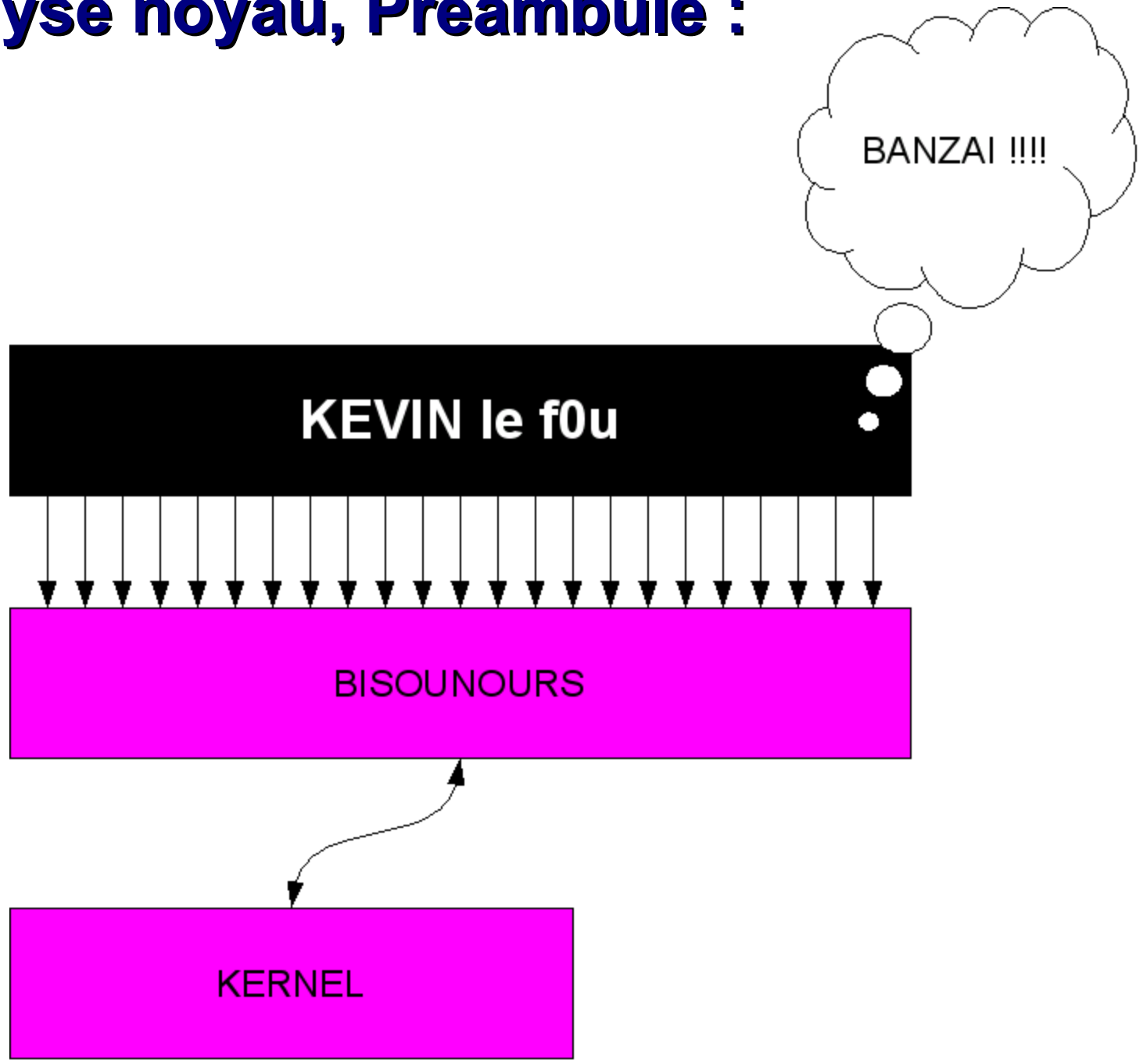
- Libkernsh
- Libke2dbg
- Kernsh
- Ke2dbg

Analyse noyau, Préambule :



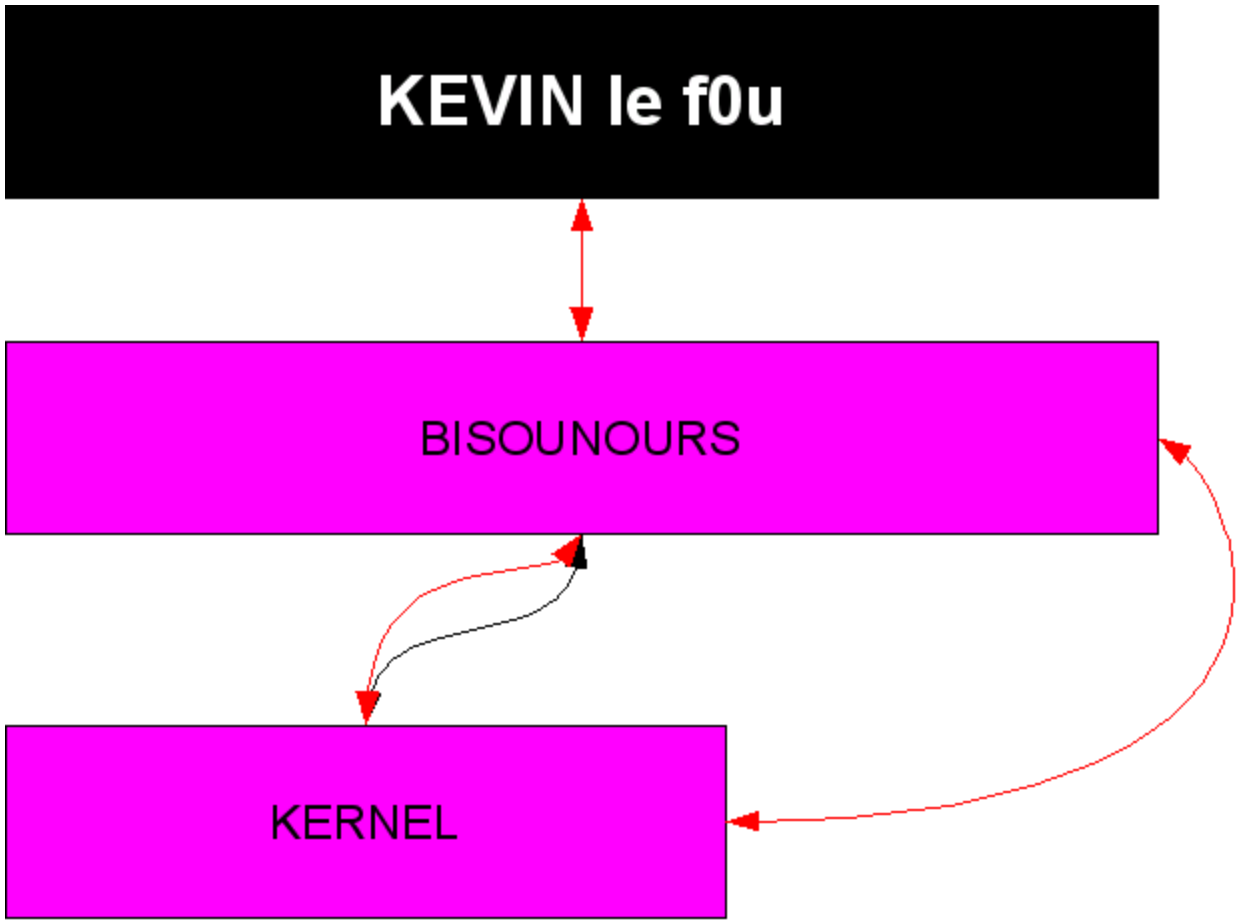
Analyse noyau, Préambule :

The ERESI Reverse Engineering Software Interface



Analyse noyau, Préambule :

The ERESI Reverse Engineering Software Interface



Analyse noyau :

- Le saint Graal !
- Plaguez (Weakening The Linux Kernel)
- On the fly :
  - sd et devik (Linux on th fly kernel patching without LKM)
  - Silvio Cesare (Runtime Kernel Patching)
- Exploiting :
  - sgrakkyu and twiz (Attacking the Core : Kernel Exploiting Notes)
  - Duverger (Exploitation Noyau)

ERESI : deux nouveaux projets :

- Analyse dynamique : Kernsh
- Debug du kernel : Ke2dbg

→ Uniquement sous des OS libres

The ERESI Reverse Engineering Software Interface

KERNSH :

- Shell Userland ⟷ Kernelland
- Premier développement en 2001 :
  - Samuel Dralet, Nicolas Brito et Kstat (et tous les contributeurs anonymes !!)(merci)
  - Linux 2.4.X, pas modulaire, mais bien fun !
- 2007 : Renaissance du projet (comme le phoenix !)
- Intégration dans ERESI
- Design pour les rootkits ?!?

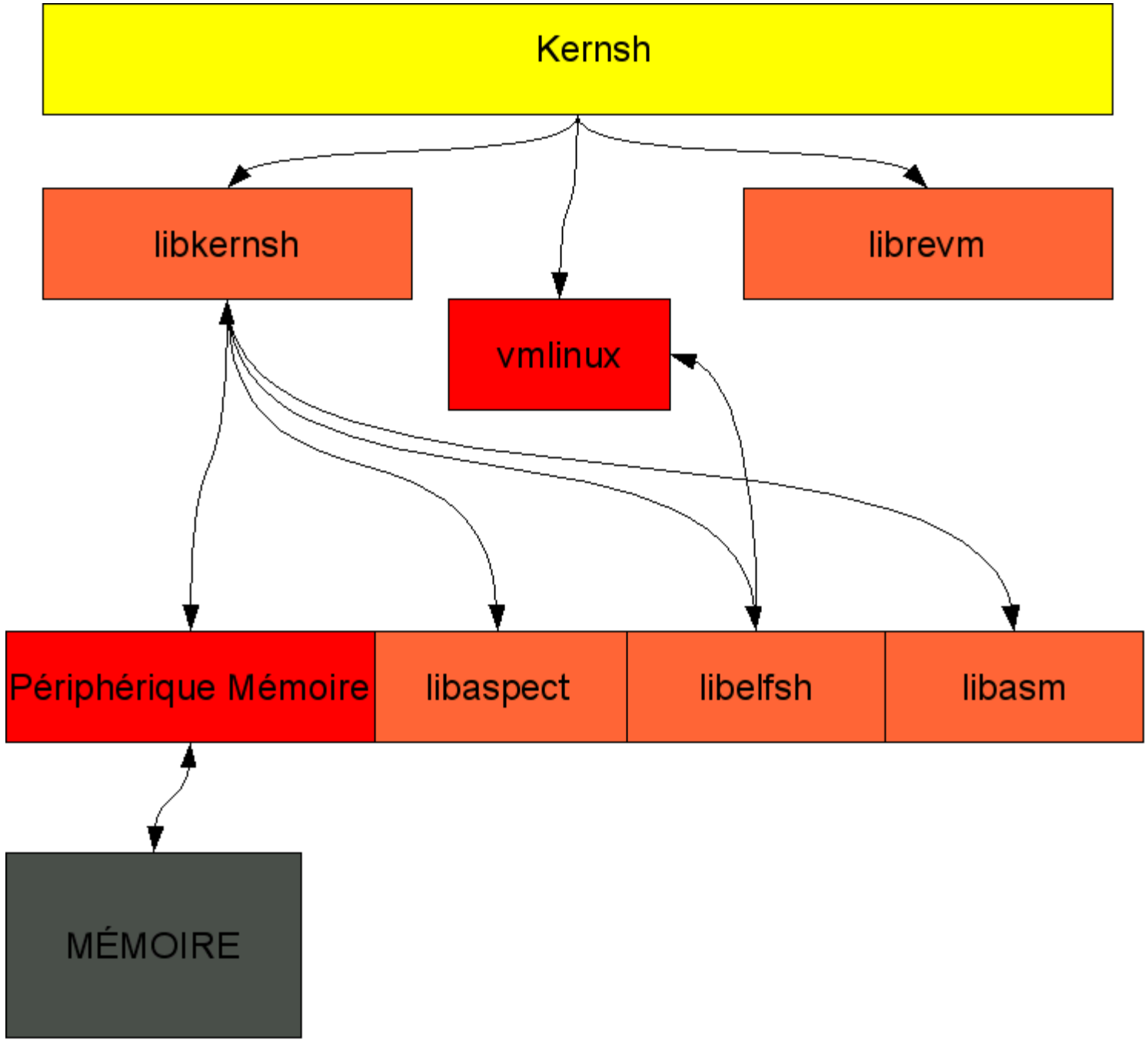
The ERESI Reverse Engineering Software Interface

KERNSH :

- Shell Userland
- Bibliothèque de communication Kernelland :
  - Périphérique mémoire :
    - /dev/(k)mem
    - /proc/kcore (lecture seule)
    - LKM (appel système, /proc)
  - Kernel static (binaire elf)
  - Réutilisation du framework ERESI (librevm, libelfsh, libasm, libaspect)

Kernsh :

The ERESI Reverse Engineering Software Interface



KERNSH :

- Lecture/Écriture dans la mémoire kernel ou kernel static
  - En langage ERESI
  - Basculement mode static <-> dynamic
- Ne pas réinventer la roue :
  - get_raw
- Surchage des fonctions existantes pour utiliser libkernsh
  - elfsh_get_raw
  - revm_get_raw

[KERNSH] Allocation, Libération mémoire :

- Mémoire kernel contiguë/non contiguë
  - kmalloc/vmalloc
- Utilisation d'un syscall libre

[KERNSH] Allocation, Libération mémoire :

- Mémoire kernel contigüe/non contigüe
  - ↙ kmalloc/vmalloc
- Utilisation d'un syscall libre

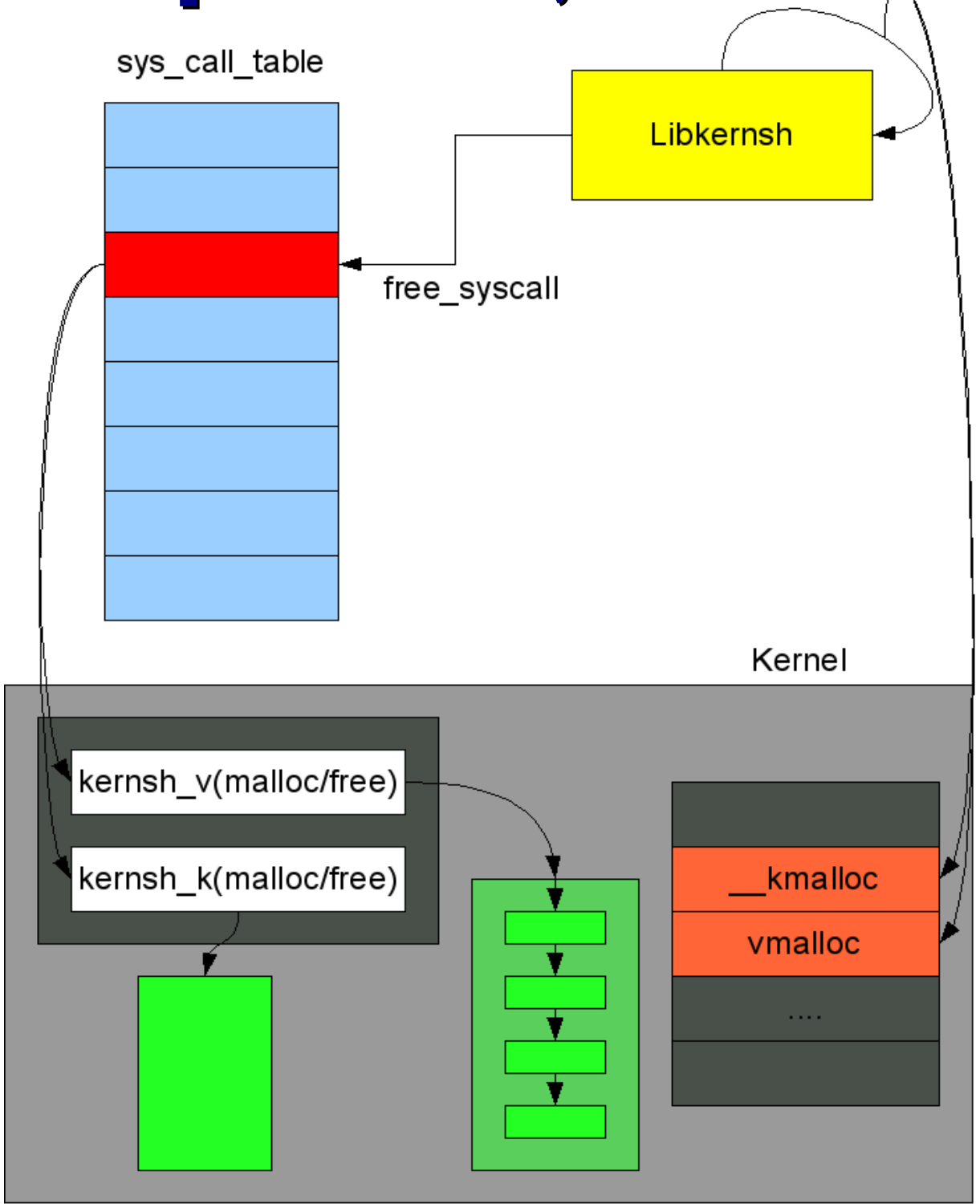
```
eresi@zion ~ $ grep sys_ni_syscall /usr/src/linux-2.6.24.2/arch/x86/kernel/syscall_table_32.S |wc -l
21
eresi@zion ~ $ grep sys_ni_syscall /usr/src/linux-2.6.25/arch/x86/kernel/syscall_table_32.S|wc -l
21
eresi@zion ~ $ grep sys_ni_syscall /usr/src/linux-2.6.25/arch/x86/kernel/syscall_table_32.S
        .long sys_ni_syscall    /* old break syscall holder */
        .long sys_ni_syscall    /* old stty syscall holder */
        .long sys_ni_syscall    /* old gtty syscall holder */
        .long sys_ni_syscall    /* 35 - old ftime syscall holder */
        .long sys_ni_syscall    /* old prof syscall holder */
        .long sys_ni_syscall    /* old lock syscall holder */
        .long sys_ni_syscall    /* old mpx syscall holder */
        .long sys_ni_syscall    /* old ulimit syscall holder */
        .long sys_ni_syscall    /* old profil syscall holder */
        .long sys_ni_syscall    /* old "idle" system call */
        .long sys_ni_syscall    /* old "create_module" */
        .long sys_ni_syscall    /* 130: old "get_kernel_syms" */
        .long sys_ni_syscall    /* reserved for afs_syscall */
        .long sys_ni_syscall    /* Old sys_query_module */
        .long sys_ni_syscall    /* reserved for streams1 */
        .long sys_ni_syscall    /* reserved for streams2 */
        .long sys_ni_syscall    /* reserved for TUX */
        .long sys_ni_syscall
        .long sys_ni_syscall
        .long sys_ni_syscall    /* sys_vserver */
        .long sys_ni_syscall            /* 285 */ /* available */
eresi@zion ~ $
```

[KERNSH] Allocation, Libération mémoire :

- Mémoire kernel contigüe/non contigüe
  - kmalloc/vmalloc
- Utilisation d'un syscall libre
- Utlisation d'un syscall existant
  - sys_sethostname
  - sys_setdomainname
  - sys_reboot
  - old_*

[KERNSH] Allocation, Libération mémoire :

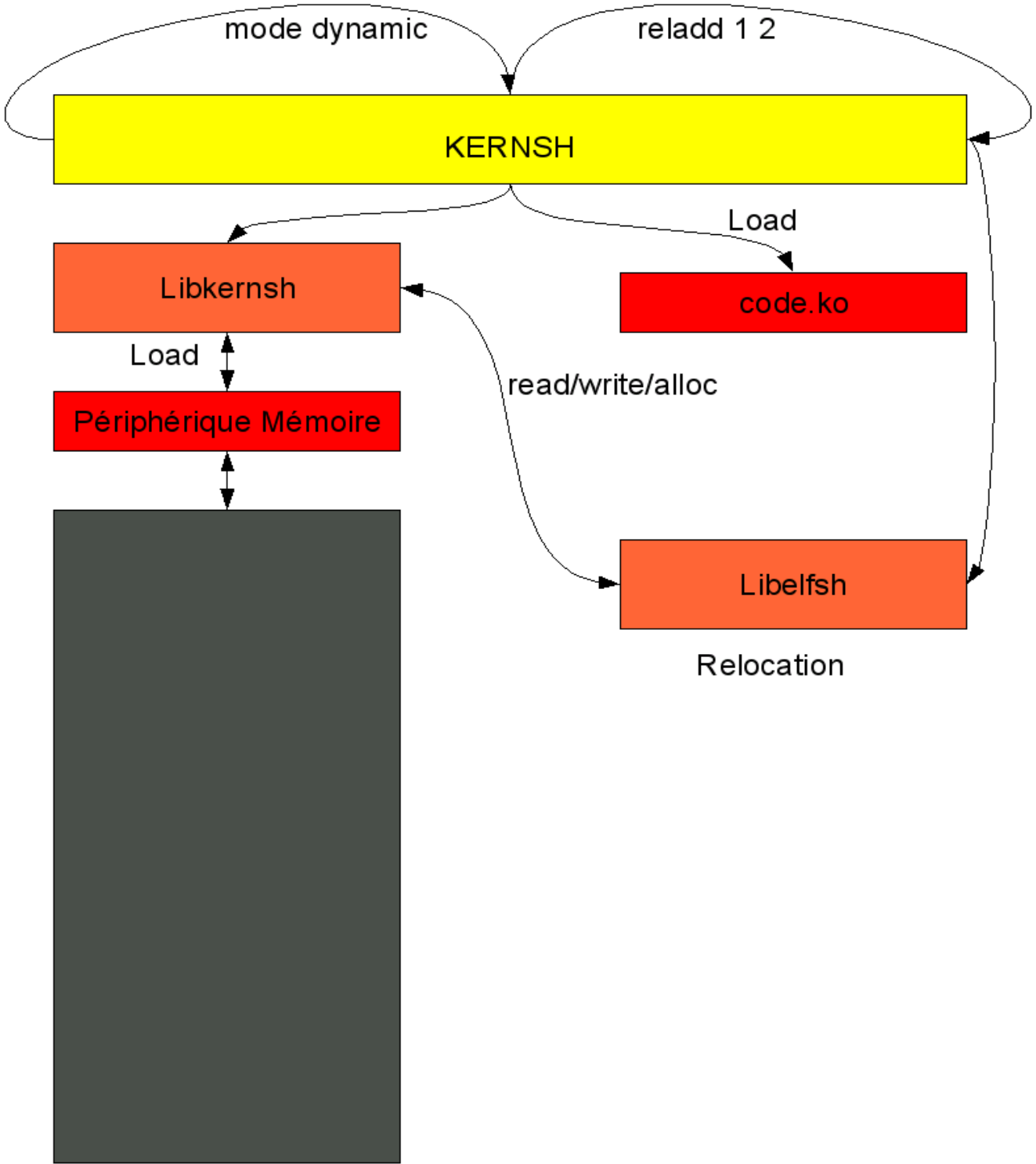
The ERESI Reverse Engineering Software Interface



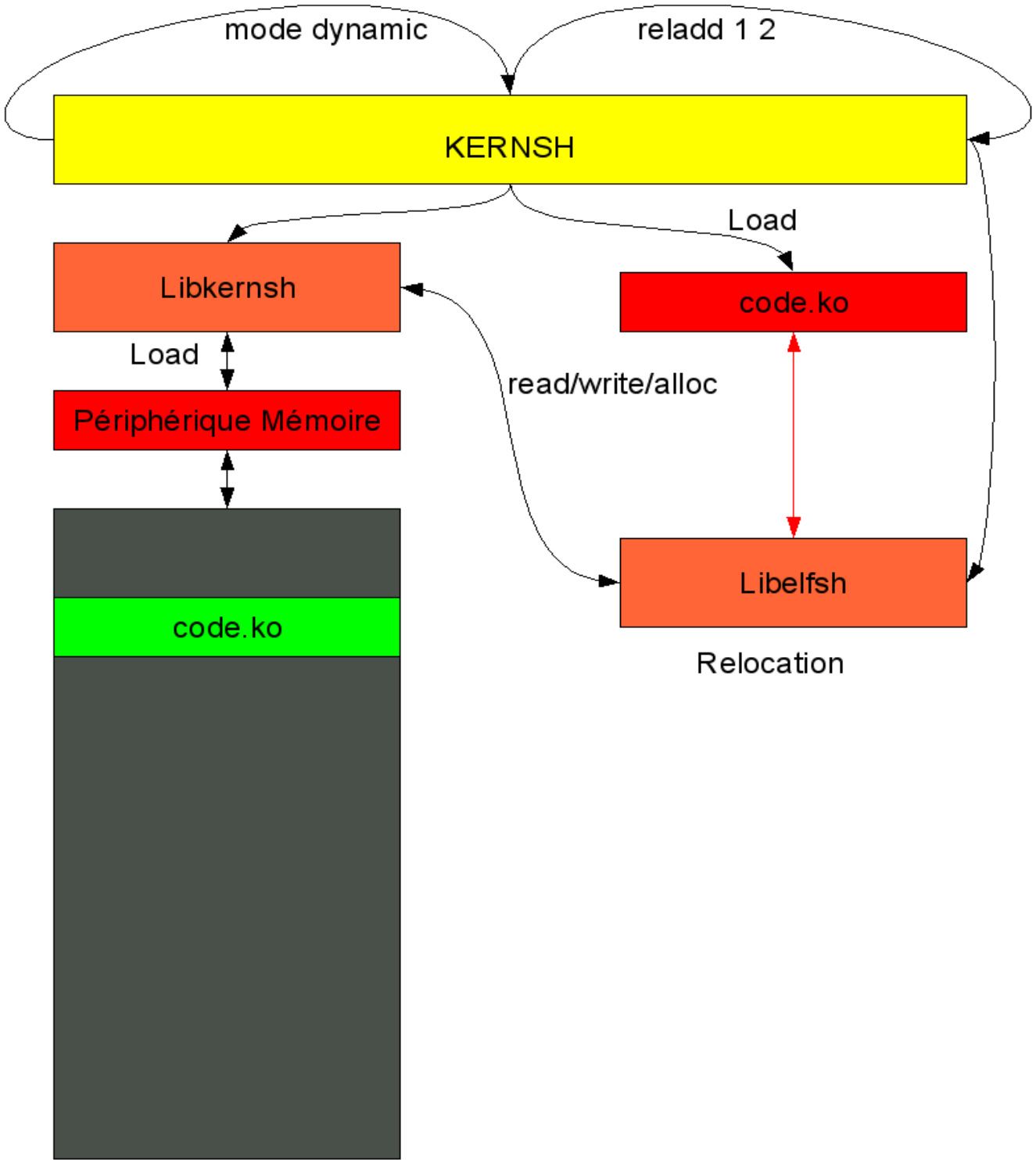
[KERNSH] Injection de code :

- D'objets ET_REL :
  - .o
  - .ko !
- Repose sur libelfsh :
  - Modification de la fonction de relocation :
    - Allocation de mémoire kernel
    - Lecture/Écriture dans la mémoire kernel
      - → Utilisation de libkernsh

[KERNSH] Injection de code :



[KERNSH] Injection de code :

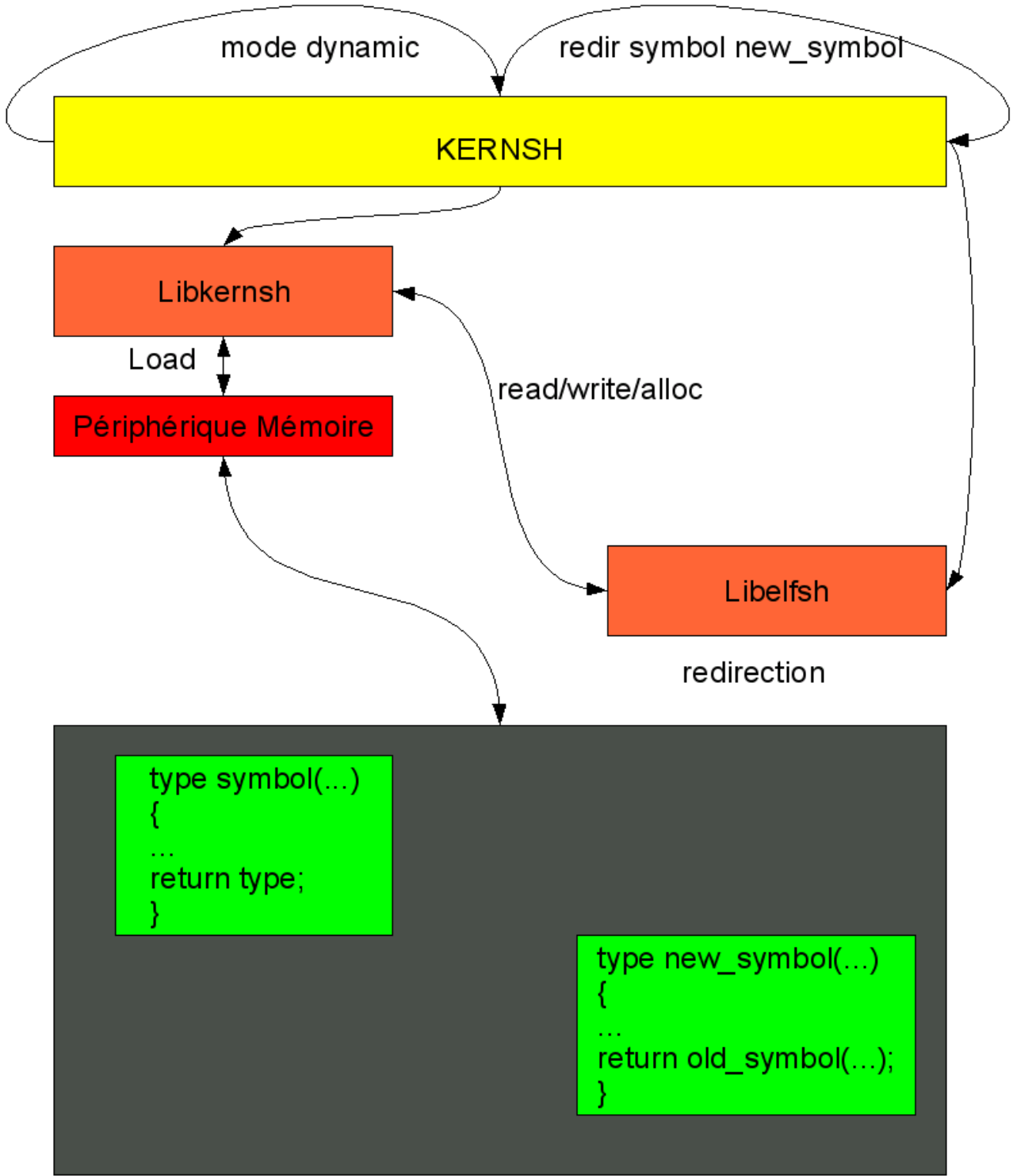


The ERESI Reverse Engineering Software Interface

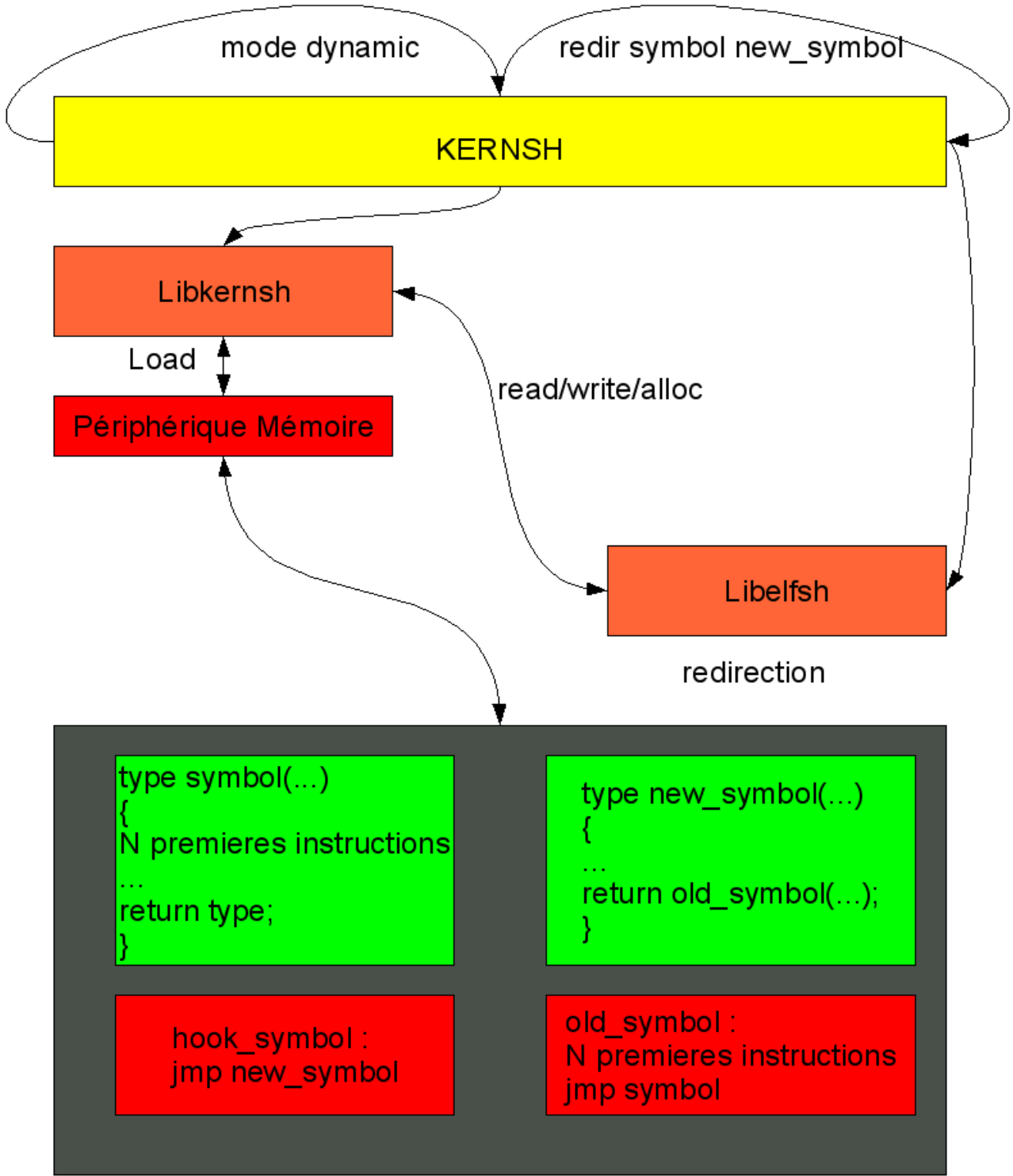
[KERNSH] Redirection de fonction :

- Redirection de fonction en mémoire kernel
  - rediriger la fonction X vers la fonction Y
- Technique CFLOW (jump jump jump !)
  - X : jmp vers la fonction Y
  - Y : jmp vers la fonction X
- Repose sur libelfsh

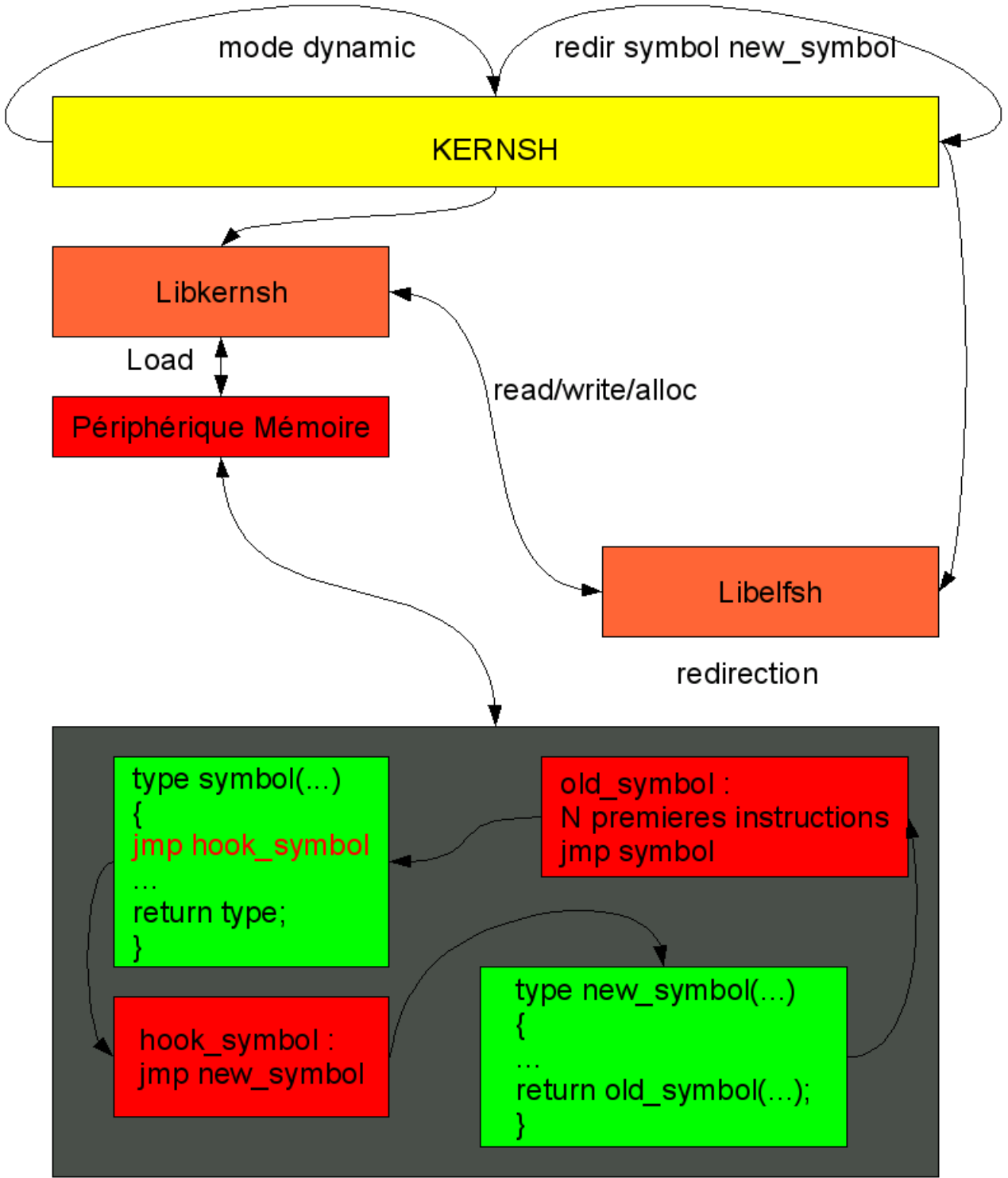
[KERNSH] Redirection de fonction :



[KERNSH] Redirection de fonction :



[KERNSH] Redirection de fonction :



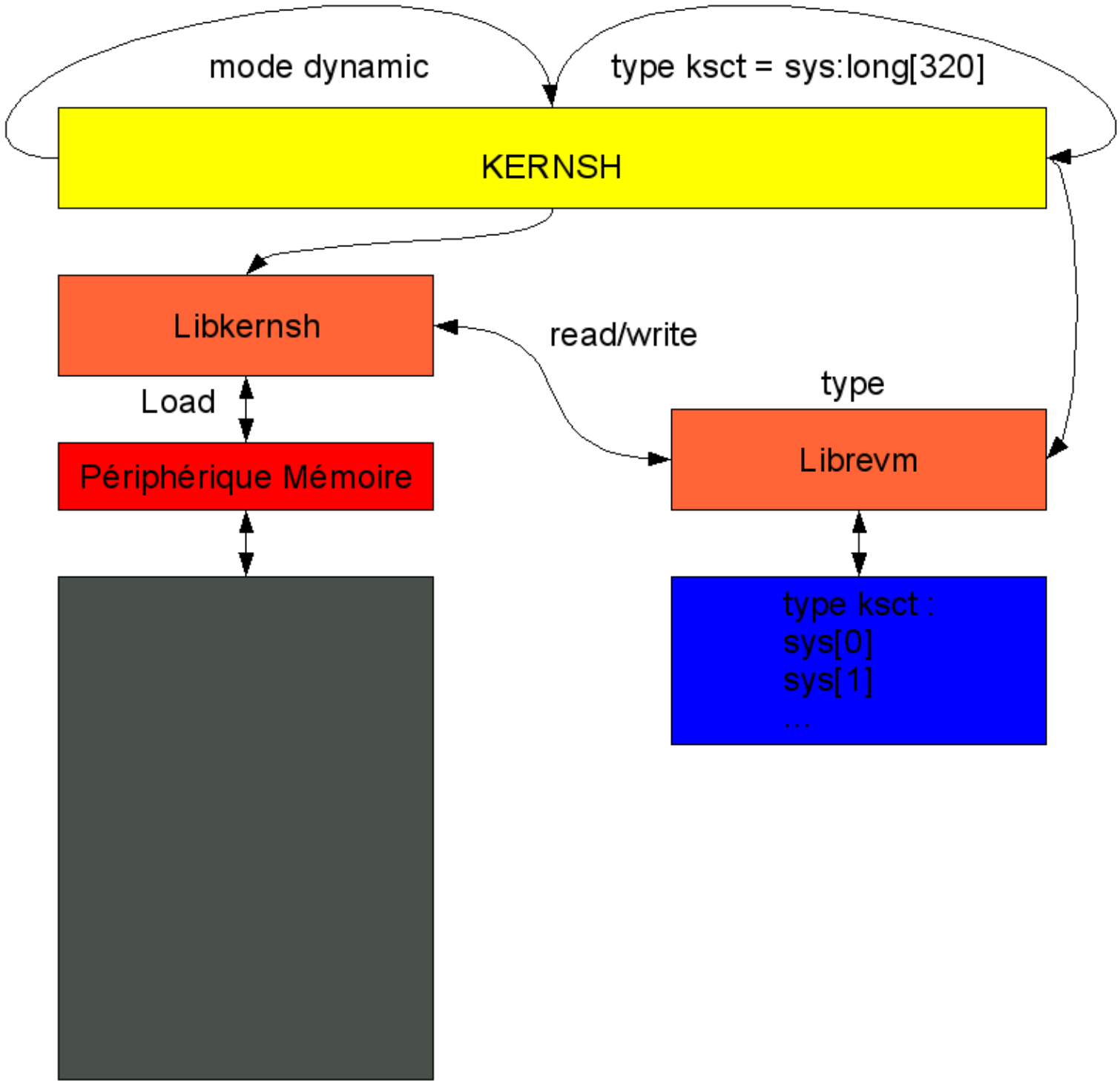
[KERNSH] Structures Kernel :

- Définition de nos propres types via librevm :
  - ↙ créer nos propres structures dynamiquement
  - ↙ Recréer les structures existantes du kernel !
- Correspondance d'une structure vers une adresse (mémoire ou dans le binaire)

➔ Tous les accès à la mémoire noyau peuvent être en script ERESI!!

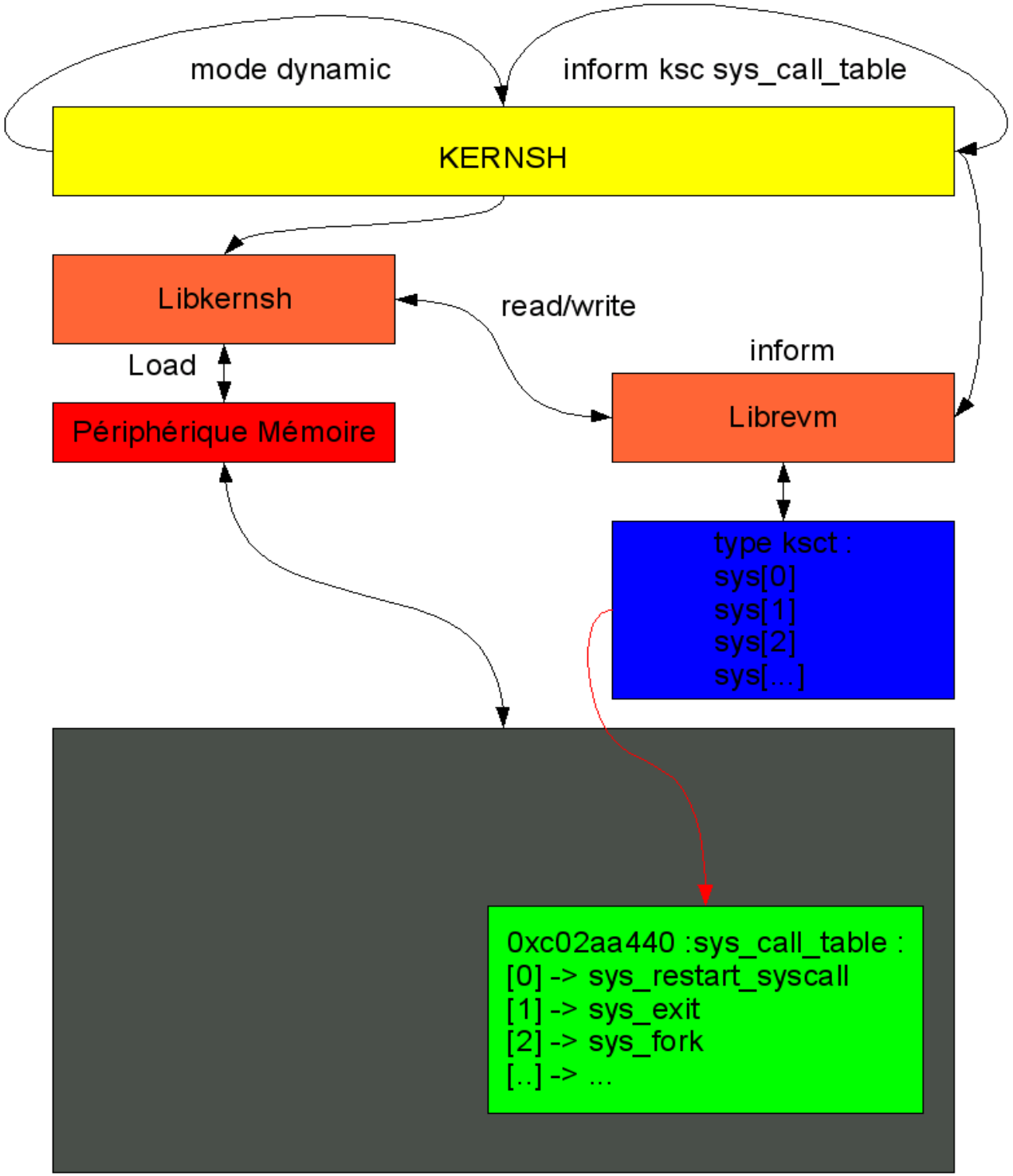
[KERNSH] Structures Kernel :

The ERESI Reverse Engineering Software Interface

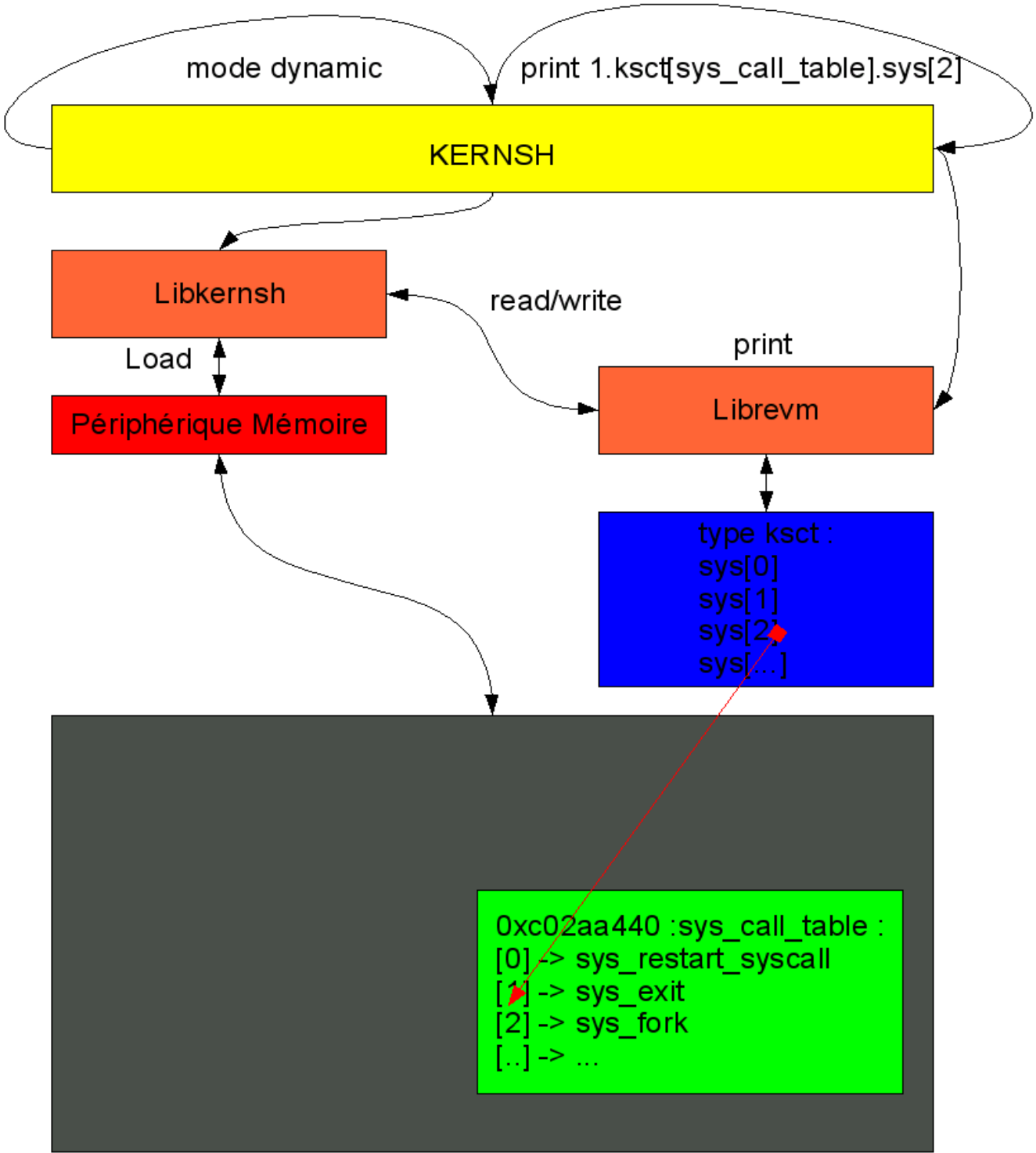


[KERNSH] Structures Kernel :

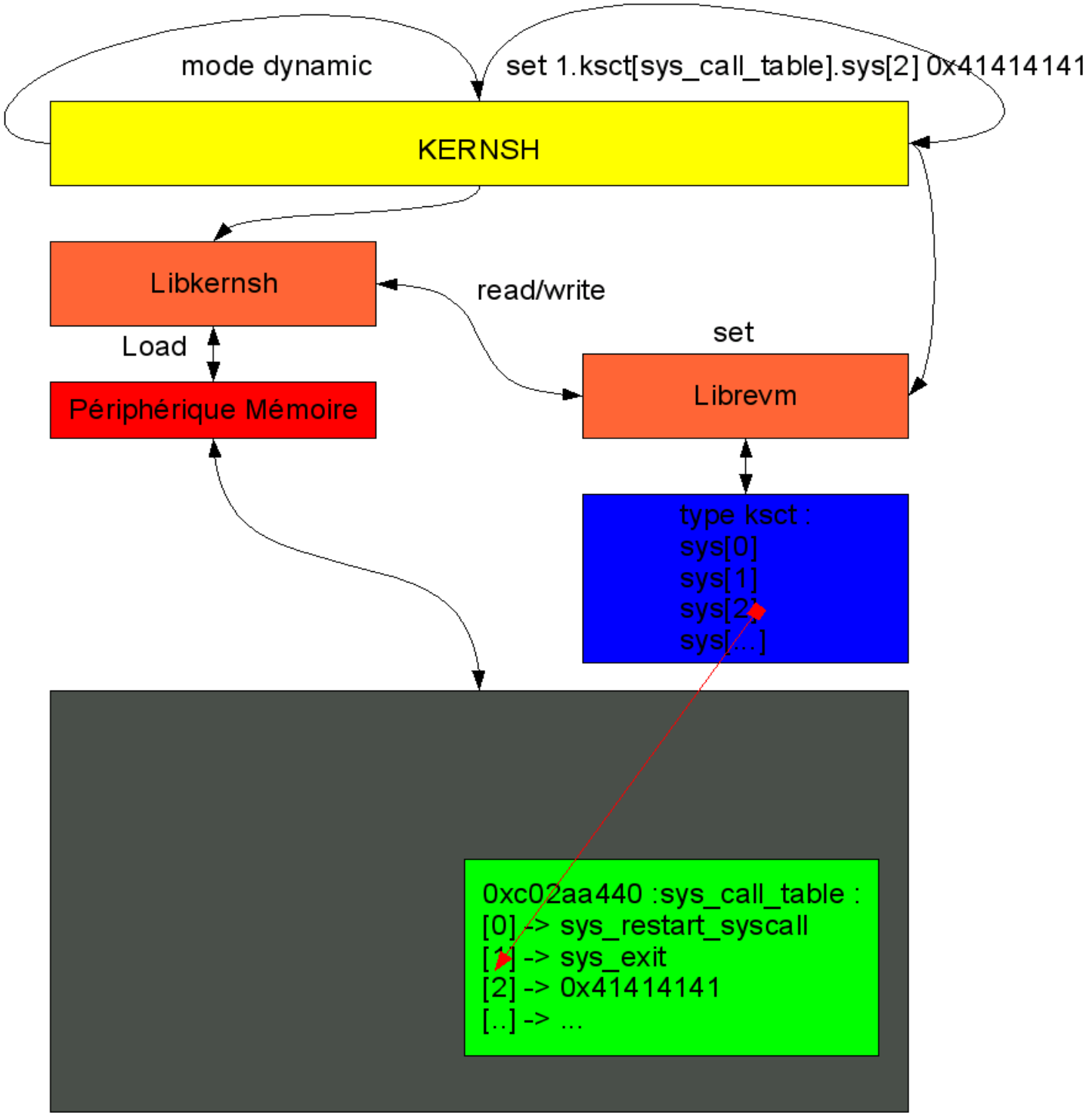
The ERESI Reverse Engineering Software Interface



[KERNSH] Structures Kernel :



[KERNSH] Structures Kernel :

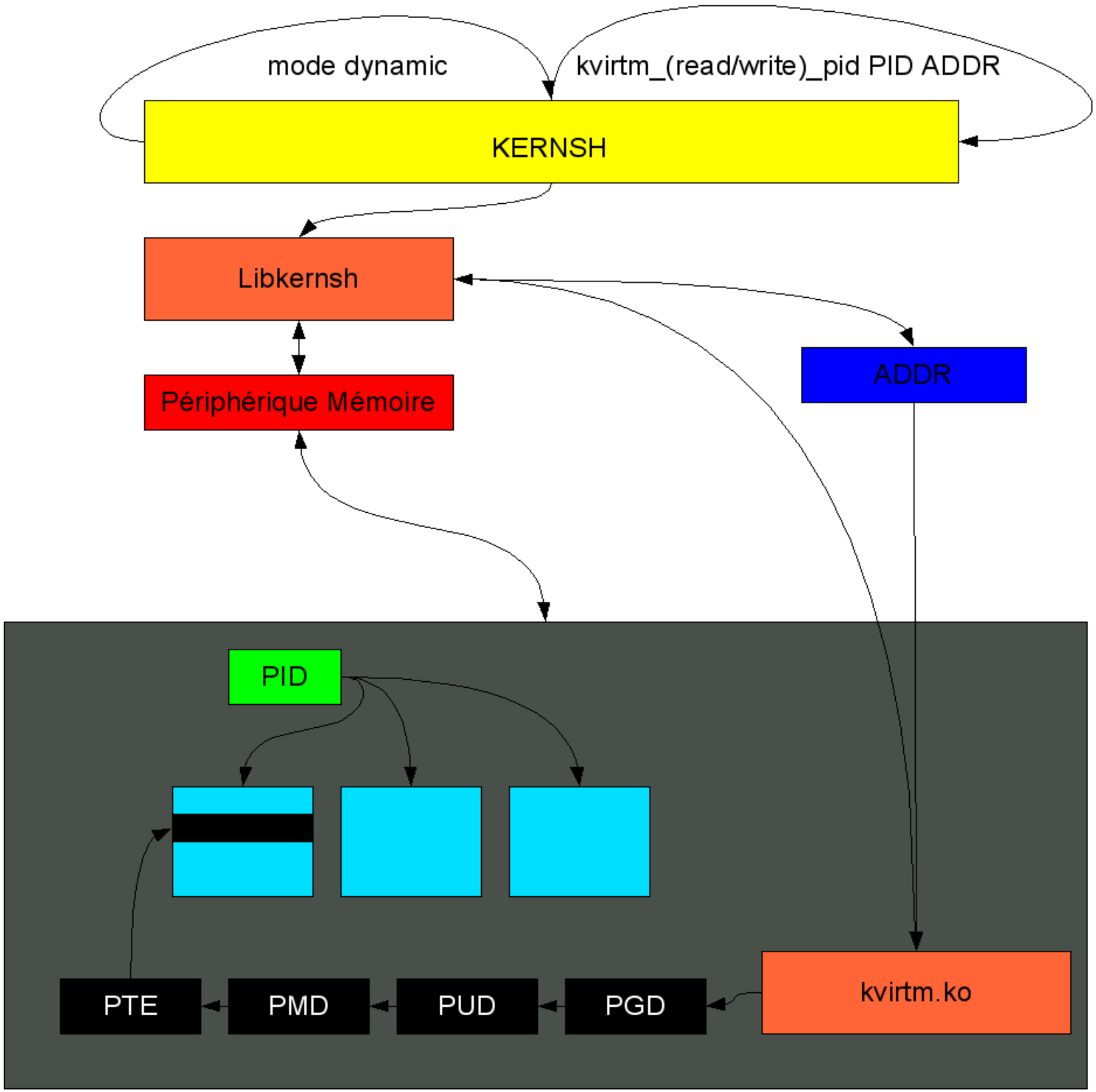


[KERNSH] Processus :

- Lecture/Écriture dans la mémoire virtuelle des processus
  - LKM effectuant la liaison
- Récupération du bloc dans une page pour une adresse virtuelle X et une taille Y
- Dump automatique des VMA d'un processus (intéressant pour le forensic !)

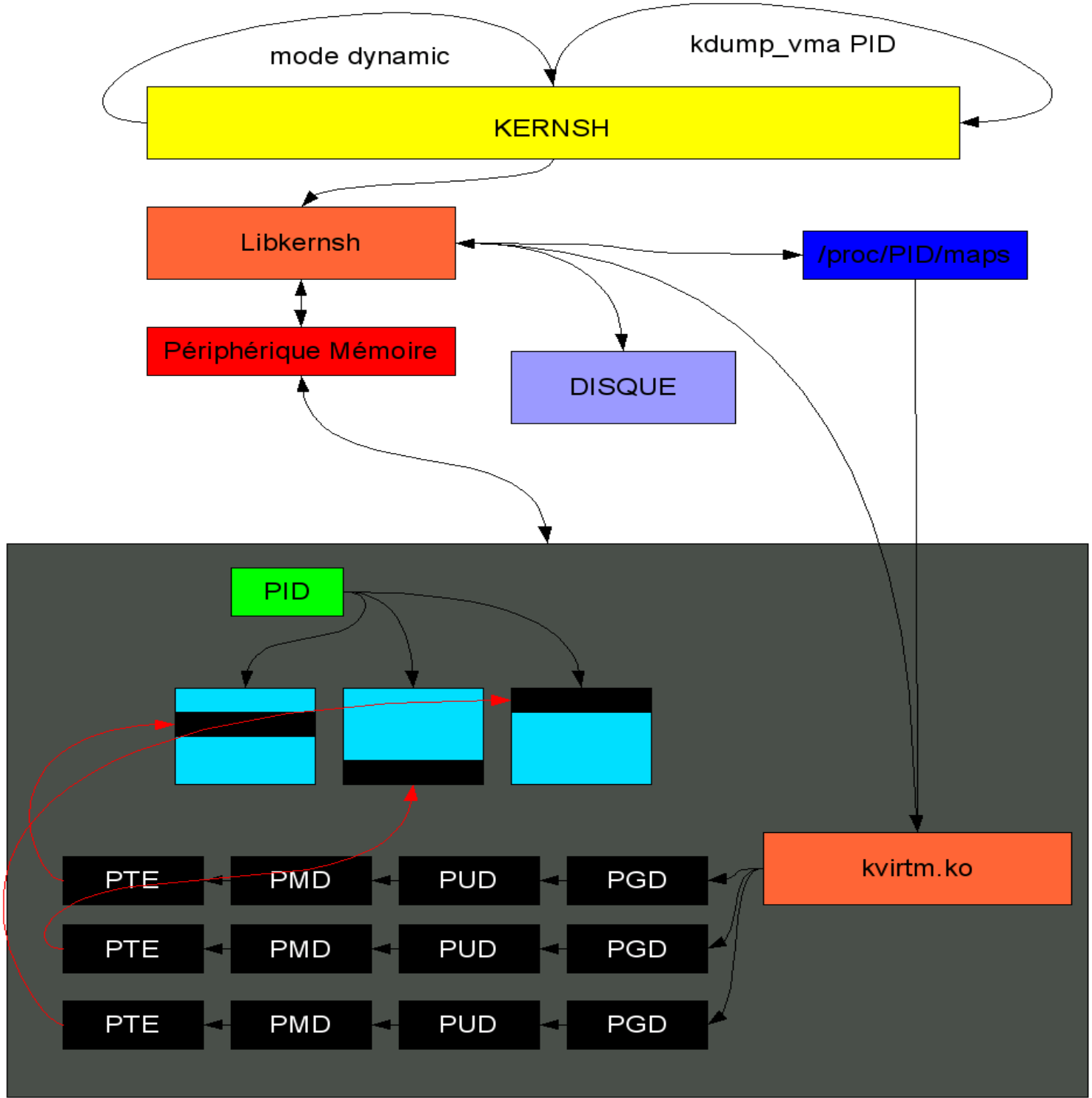
[KERNSH] Processus :

The ERESI Reverse Engineering Software Interface



[KERNSH] Processus :

The ERESI Reverse Engineering Software Interface



[KERNSH] Autres :

- Affichage de la sys_call_table, idt, gdt, symbols
- Hash de fonctions (ou portions)
- Désassemblage de la mémoire kernel (fonctions, etc)
- Redirection d'init des LKM
- ....

[KERNSH] Exemple redirection :

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

int asmlinkage new_sys_kill(int pid, int sig)
{
        printk("NEW SYS KILL, PID %d SIG %d\n", pid, sig);

        return old_sys_kill(pid, sig);
}

static int testreladd_init(void)
{
        printk(KERN_ALERT "TEST REL ADD INIT\n");
        return 0;
}

static void testreladd_exit(void)
{
        printk(KERN_ALERT "TEST REL ADD EXIT\n");
}


module_init(testreladd_init);
module_exit(testreladd_exit);
```

```
#!./kernsh32

#openmem

load examples/modules/linux2_6/lkm-sys_kill.ko

D sys_kill

mode dynamic

reladd 1 2

redir sys_kill new_sys_kill

D sys_kill
D old_sys_kill

redir

closemem
quit
```

The ERESI Reverse Engineering Software Interface

[KERNSH] Exemple redirection :

```
(kernsh-0.81-a8-dev@local) D sys_kill

 0xC012CEB8 [foff: 188088] sys_kill + 0          push        %ebp                          55
 0xC012CEB9 [foff: 188089] sys_kill + 1          push        %edi                          57
 0xC012CEBA [foff: 188090] sys_kill + 2          push        %esi                          56
 0xC012CEBB [foff: 188091] sys_kill + 3          push        %ebx                          53
 0xC012CEBC [foff: 188092] sys_kill + 4          add         $FFFFFF80,%esp                83 C4 80
 0xC012CEBF [foff: 188095] sys_kill + 7          mov         00000098(%esp,1),%ebp         8B AC 24 98 00 00 00
 0xC012CEC6 [foff: 188102] sys_kill + 14         mov         <per_cpu__current_task>,%fs:%ecx                    64 8B 0D 00 30 38 C0
 0xC012CECD [foff: 188109] sys_kill + 21         mov         $00000000,00000004(%esp,1)    C7 44 24 04 00 00 00 00
 0xC012CED5 [foff: 188117] sys_kill + 29         mov         00000094(%esp,1),%edx         8B 94 24 94 00 00 00
 0xC012CEDC [foff: 188124] sys_kill + 36         mov         $00000000,00000008(%esp,1)    C7 44 24 08 00 00 00 00
 0xC012CEE4 [foff: 188132] sys_kill + 44         mov         %ebp,(%esp,1)                 89 2C 24
 0xC012CEE7 [foff: 188135] sys_kill + 47         mov         000000A8(%ecx),%eax           8B 81 A8 00 00 00
 0xC012CEED [foff: 188141] sys_kill + 53         test        %edx,%edx                     85 D2
 0xC012CEEF [foff: 188143] sys_kill + 55         mov         %eax,0000000C(%esp,1)         89 44 24 0C
(kernsh-0.81-a8-dev@local) load examples/modules/linux2_6/lkm-sys_kill.ko

 [*] Sun Jun  1 22:27:08 2008 - New object loaded : examples/modules/linux2_6/lkm-sys_kill.ko

(kernsh-0.81-a8-dev@local) mode dynamic

[*] kernsh is now in DYNAMIC mode

(kernsh-0.81-a8-dev@local) reladd 1 2

 [*] ET_REL examples/modules/linux2_6/lkm-sys_kill.ko injected succesfully in ET_EXEC /tmp/vmlinux

(kernsh-0.81-a8-dev@local) redir sys_kill new_sys_kill

 [*] Function sys_kill redirected to addr 0xCFFB2B64 <new_sys_kill>

(kernsh-0.81-a8-dev@local)
```

The ERESI Reverse Engineering Software Interface

[KERNSH] Exemple redirection :

```
0xC012CEB8 [foff: 188088] sys_kill + 0          jmp     <hook_sys_kill@vmlinux>          E9 43 F1 63 0D
0xC012CEBD [foff: 188093] sys_kill + 5          nop                                      90
0xC012CEBE [foff: 188094] sys_kill + 6          nop                                      90
0xC012CEBF [foff: 188095] sys_kill + 7          mov     00000098(%esp,1),%ebp            8B AC 24 98 00 00 00
0xC012CEC6 [foff: 188102] sys_kill + 14         mov     <per_cpu__current_task@vmlinux>,%fs:%ecx          64 8B 0D 00 30 38 C0
0xC012CECD [foff: 188109] sys_kill + 21         mov     $00000000,00000004(%esp,1)       C7 44 24 04 00 00 00 00
```

```
0xCD76C000 [foff: 0] hook_sys_kill + 0          jmp     <new_sys_kill@vmlinux>           E9 5F 6B 84 02
0xCD76C005 [foff: 5] hook_sys_kill + 5          push    %ebp                             55
0xCD76C006 [foff: 6] hook_sys_kill + 6          push    %edi                             57
0xCD76C007 [foff: 7] hook_sys_kill + 7          push    %esi                             56
0xCD76C008 [foff: 8] hook_sys_kill + 8          push    %ebx                             53
0xCD76C009 [foff: 9] hook_sys_kill + 9          add     $FFFFFF80,%esp                   83 C4 80
0xCD76C00C [foff: 12] hook_sys_kill + 12        jmp     <sys_kill@vmlinux + 7> E9 AE 0E 9C F2

0xCD76C005 [foff: 0] old_sys_kill + 0                   push    %ebp                             55
0xCD76C006 [foff: 1] old_sys_kill + 1                   push    %edi                             57
0xCD76C007 [foff: 2] old_sys_kill + 2                   push    %esi                             56
0xCD76C008 [foff: 3] old_sys_kill + 3                   push    %ebx                             53
0xCD76C009 [foff: 4] old_sys_kill + 4                   add     $FFFFFF80,%esp                   83 C4 80
0xCD76C00C [foff: 7] old_sys_kill + 7                   jmp     <sys_kill@vmlinux + 7> E9 AE 0E 9C F2
```

The ERESI Reverse Engineering Software Interface

[KERNSH] Exemple redirection :

```
0xCFFB2B64 [foff: 36] new_sys_kill + 0          push    %esi                                    56
0xCFFB2B65 [foff: 37] new_sys_kill + 1          push    %ebx                                    53
0xCFFB2B66 [foff: 38] new_sys_kill + 2          sub     $0000000C,%esp                          83 EC 0C
0xCFFB2B69 [foff: 41] new_sys_kill + 5          mov     00000018(%esp,1),%esi                   8B 74 24 18
0xCFFB2B6D [foff: 45] new_sys_kill + 9          mov     0000001C(%esp,1),%ebx                   8B 5C 24 1C
0xCFFB2B71 [foff: 49] new_sys_kill + 13         mov     $<examples/modules/linux2_6/lkm-sys_kill.ko,rodata,str1.1@vmlinux + 44>,(%esp,1) C7 04 24 EC 2A FB CF
0xCFFB2B78 [foff: 56] new_sys_kill + 20         mov     %esi,00000004(%esp,1)                   89 74 24 04
0xCFFB2B7C [foff: 60] new_sys_kill + 24         mov     %ebx,00000008(%esp,1)                   89 5C 24 08
0xCFFB2B80 [foff: 64] new_sys_kill + 28         call    <printk@vmlinux>                        E8 24 0B 17 F0
0xCFFB2B85 [foff: 69] new_sys_kill + 33         mov     %ebx,%edx                               89 DA
0xCFFB2B87 [foff: 71] new_sys_kill + 35         mov     %esi,%eax                               89 F0
0xCFFB2B89 [foff: 73] new_sys_kill + 37         add     $0000000C,%esp                          83 C4 0C
0xCFFB2B8C [foff: 76] new_sys_kill + 40         pop     %ebx                                    5B
0xCFFB2B8D [foff: 77] new_sys_kill + 41         pop     %esi                                    5E
0xCFFB2B8E [foff: 78] new_sys_kill + 42         jmp     <old_sys_kill@vmlinux>                  E9 72 94 7B FD
```

[KERNSH] Exemple redirection :

```
(kernsh-0.81-a8-dev@local) redir

        .:.: ELFsh redirection list

        [00] TYPE:CFLOW  [C012CEB8] <sys_kill> redirected on [CFFB2B64] <new_sys_kill>


(kernsh-0.81-a8-dev@local) exec tail -5 /var/log/kern.log

Jun  1 22:26:09 eresi kernel: loop: module loaded
Jun  1 22:26:09 eresi kernel: eth0: link up
Jun  1 22:26:09 eresi kernel: NET: Registered protocol family 10
Jun  1 22:26:09 eresi kernel: lo: Disabled Privacy Extensions
Jun  1 22:26:18 eresi kernel: eth0: no IPv6 routers present

 [*] Command executed successfully

(kernsh-0.81-a8-dev@local) exec kill -69 1


 [*] Command executed successfully

(kernsh-0.81-a8-dev@local) exec tail -5 /var/log/kern.log

Jun  1 22:26:09 eresi kernel: eth0: link up
Jun  1 22:26:09 eresi kernel: NET: Registered protocol family 10
Jun  1 22:26:09 eresi kernel: lo: Disabled Privacy Extensions
Jun  1 22:26:18 eresi kernel: eth0: no IPv6 routers present
Jun  1 22:30:35 eresi kernel: NEW SYS KILL, PID 1 SIG 69

 [*] Command executed successfully
```

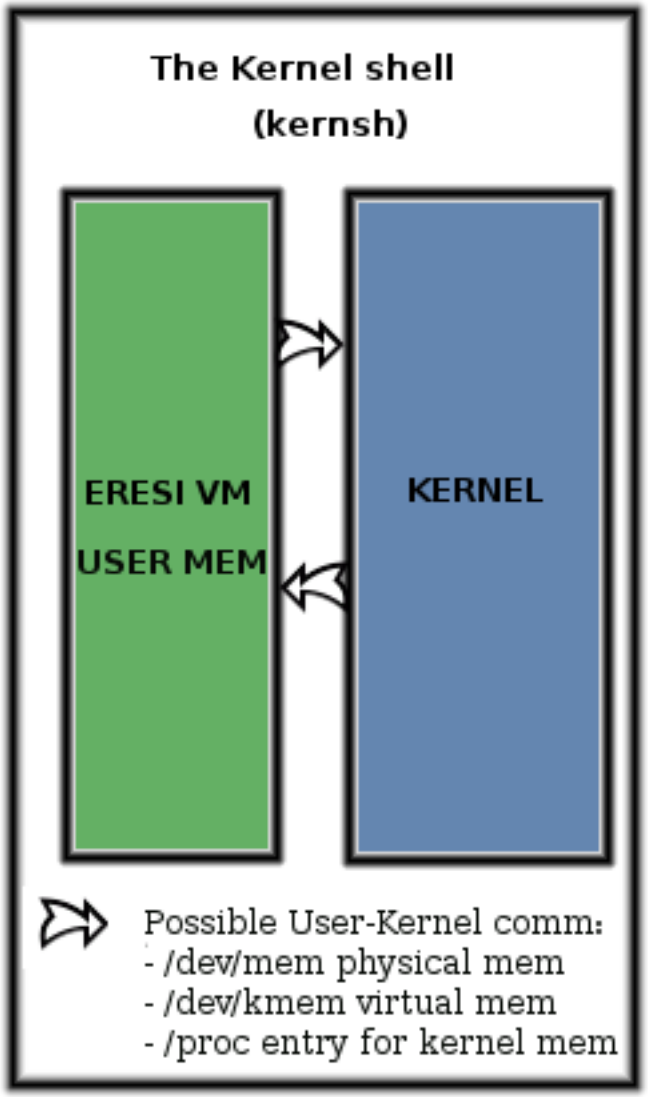
The ERESI Reverse Engineering Software Interface

[KERNSH] Exemple Intégrité :

```
type ksct = sys:long[320]
print "Sys_call_table " $sct
inform ksct sys_call_table $sct

mode dynamic

/* Fingerprint */
kmem_hash $sct%1280
set $md5save $$_

/* ROOTKIT */
set 1.ksct[sys_call_table].sys[17] 0x44444444


kmem_chash $md5save

print "CMP MD5"
cmp $_ 0
je good
print "MD5 MISMATCH !!!"
jmp end

good:
print "MD5 MATCH!!!"
end:
```

[KERNSH] Inconvénients :

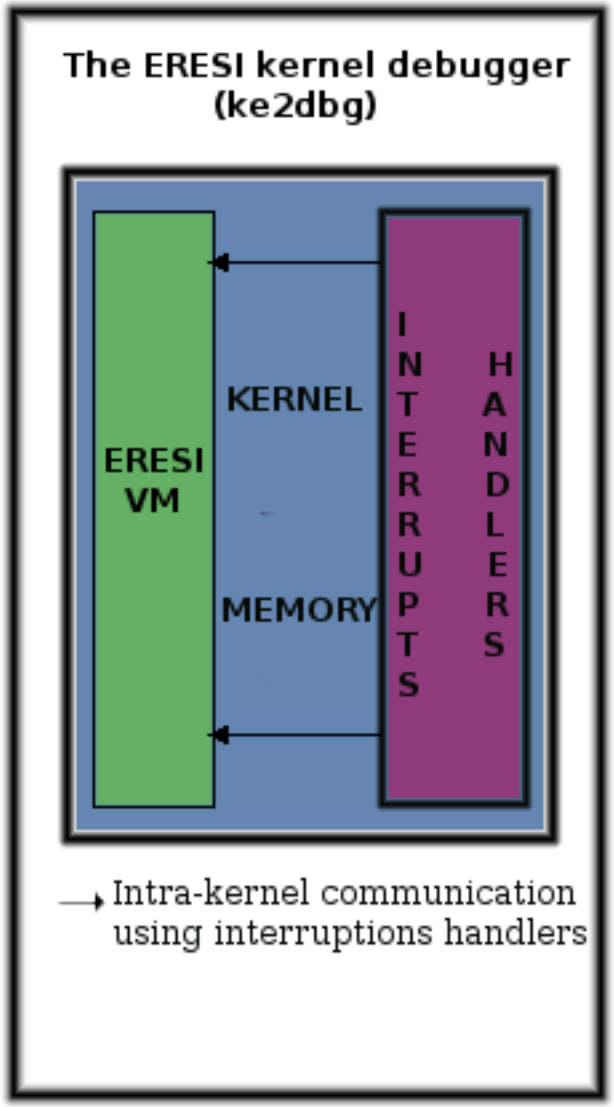
- Changement de contexte
  - Problèmes de performance
  - Fiabilité des informations récupérées
- Bug en SMP



Kernel unintrusiveness

vs

Kernel embedding



KE2DBG :

- Kernel Debugger
- Basé sur rr0d
- Chargement en LKM
- Intégration de tout le framework
  - → Code userland à migrer en kernelland

Prochaines étapes :

- Élargir le spectre des architectures
  - ARM, PPC, AMD64
- Enrichir les fonctionnalités du debugger
- Libmjollnir/Liballocproxy pour le kernel
- Vos contributions !!!

The ERESI Reverse Engineering Software Interface

team@eresi-project.org

The ERESI team vous souhaite un bon :

HAPPY HACKING !

Questions ?