Backdoors furtives et autres fourberies dans le noyau

Arnaud Ebalard <troglocan@droids-corp.org>
Pierre Lalet <pierre@droids-corp.org>
Olivier Matz <zer0@droids-corp.org>

Les 45 prochaines minutes ...

- Injection de code
- Récupération de données
- Emission de trames
- Interaction
- Furtivité

Les intervenants

- Un administrateur
  - augmenter le niveau de sécurité
    - HIDS avancé
- Un “pirate”, pour agir avec ...
  - plus de discrétion
  - plus de pouvoir

Le noyau

- Interface entre le matériel et les processus
- Coordination des processus
- Gestion de l'accès aux ressources
- Passage obligé des E/S

Injection de code

Patch des sources

- Nécessité :
  - sources
  - configuration
  - **redémarrage**
- Public visé : l'administrateur

LKM

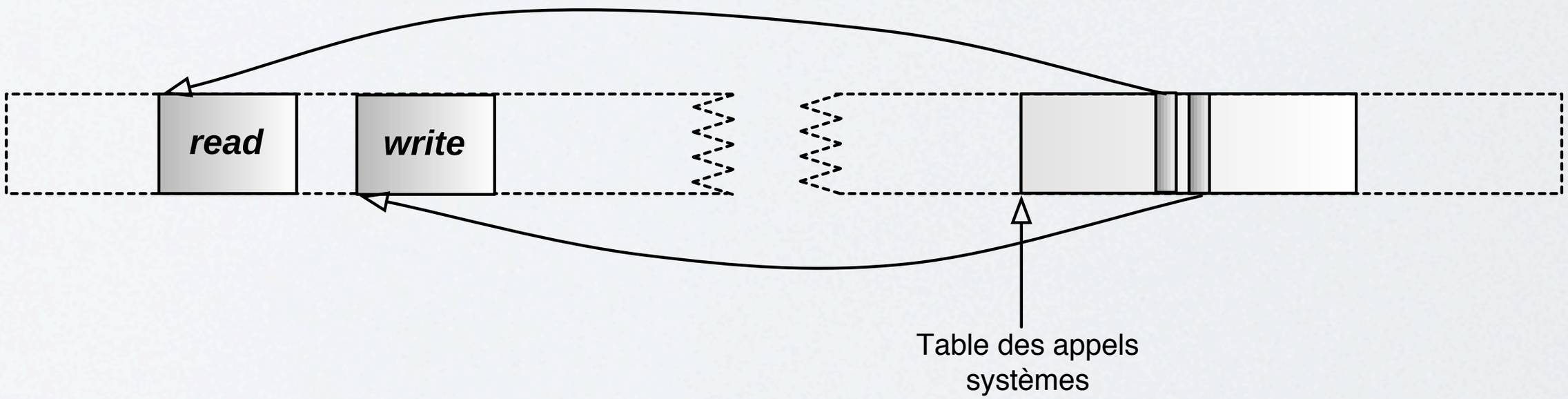
- Code chargé dynamiquement pour étendre les fonctionnalités du noyau
- Développement simple
- Relative furtivité
- Pas forcément autorisé

/dev/kmem (1/2)

- Accès userland à l'espace noyau
- Données binaires difficiles à manipuler, symboles non résolus, allocation mémoire.
- Sensibilité au reboot
- Pas forcément accessible

/dev/kmem (2/2)

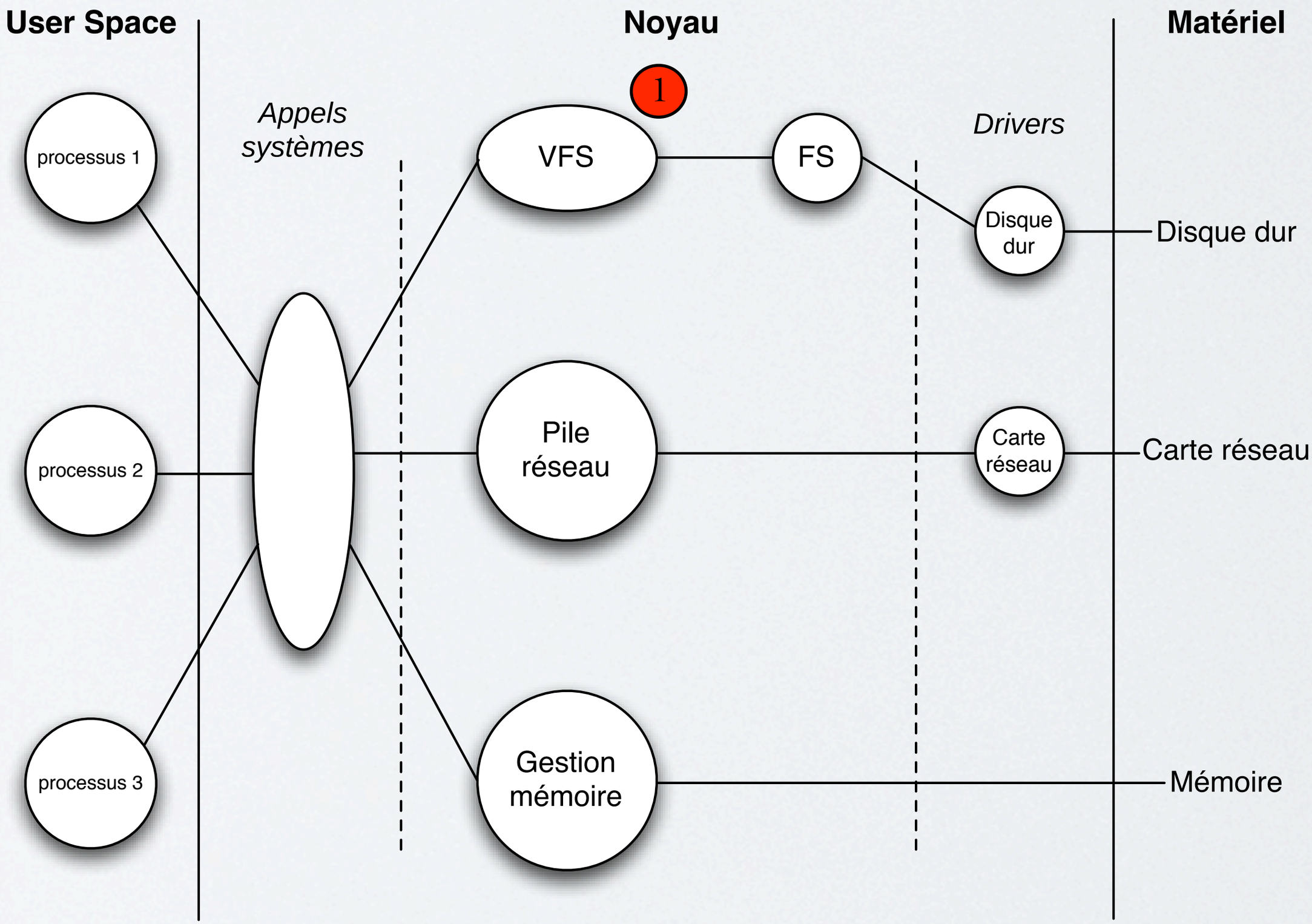
- Recherche des fonctions **read** et **write** (par le début de leur code)
- Recherche de la structure composée de leurs adresses

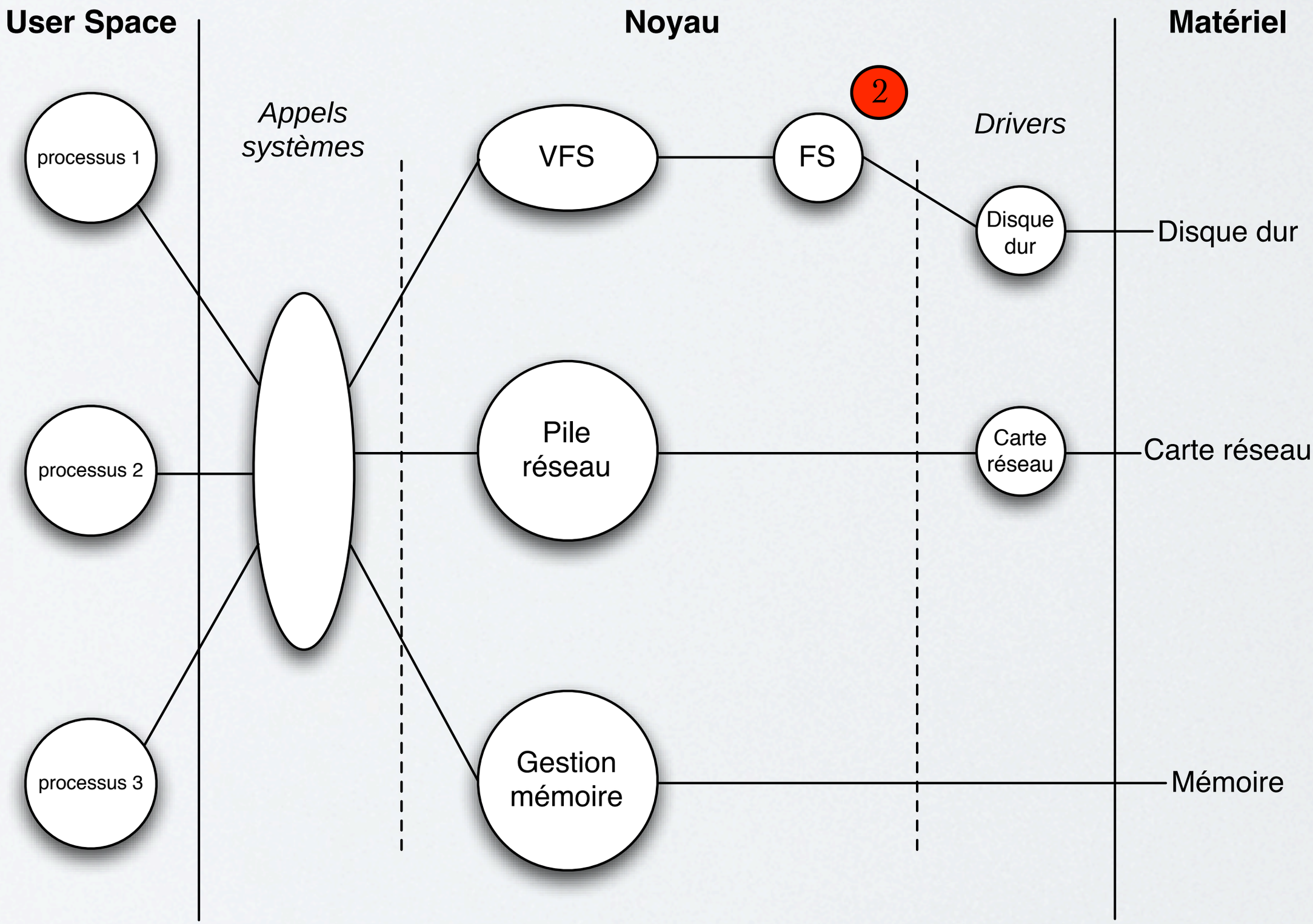


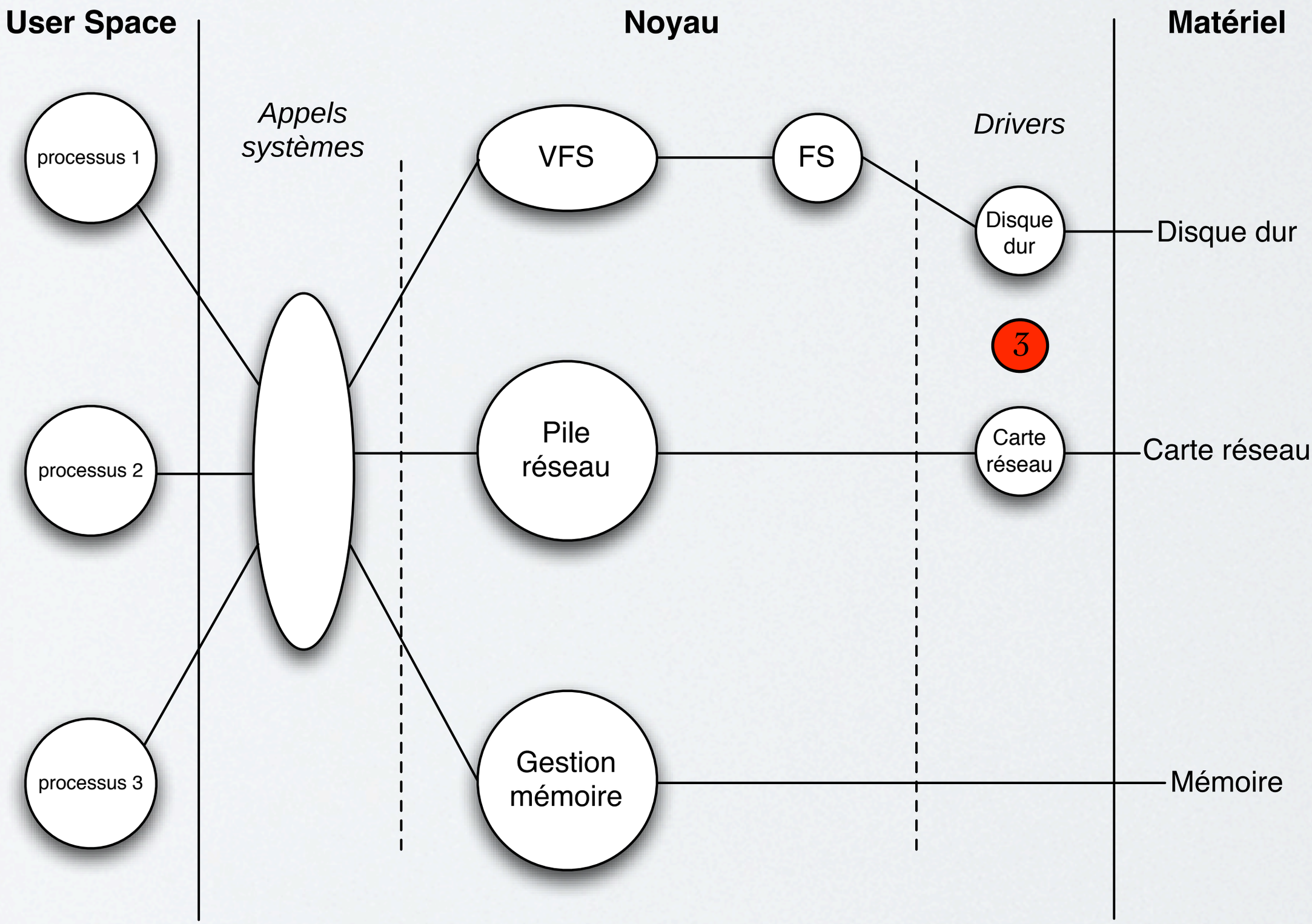
Conclusion

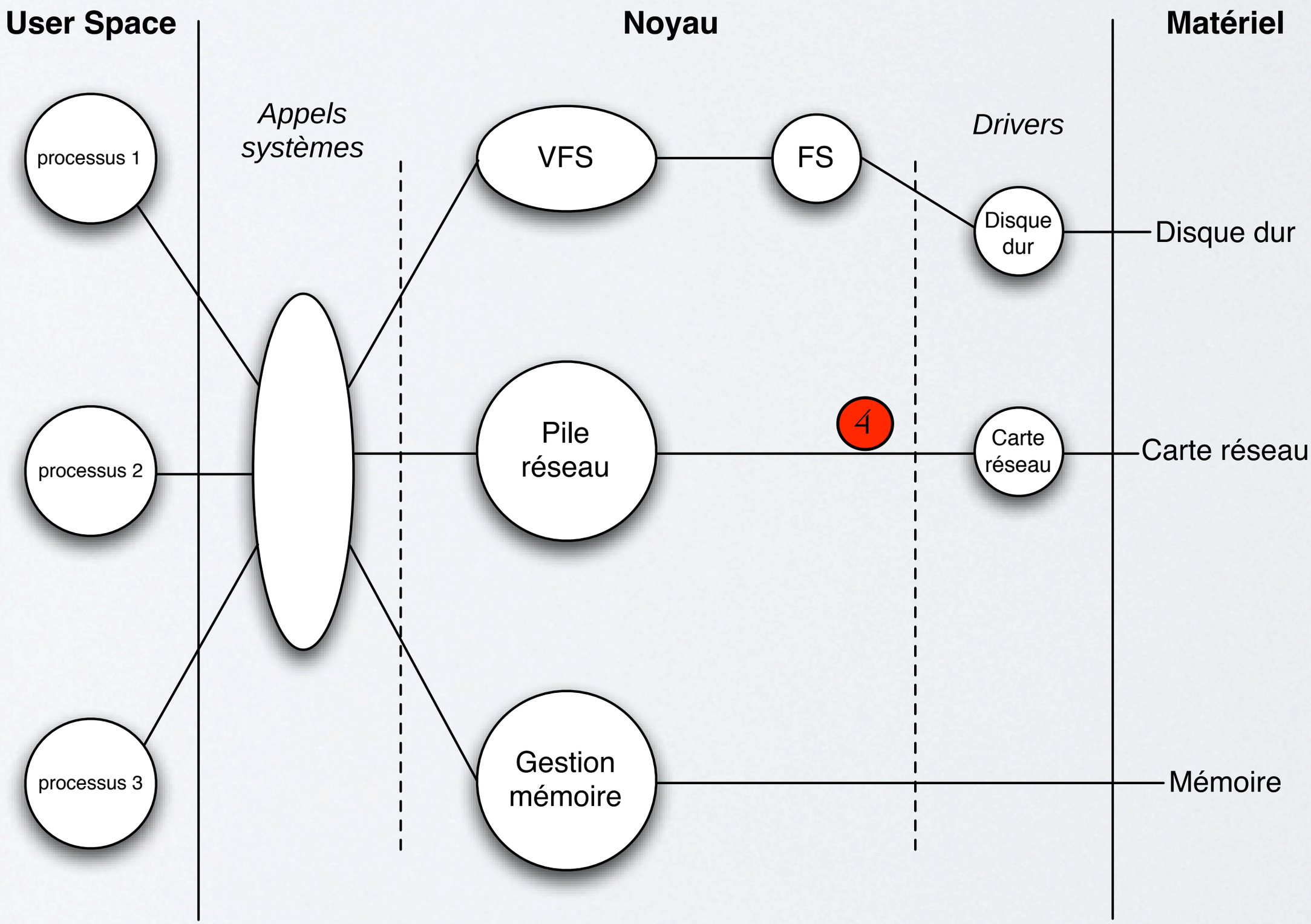
- Patch : simple, furtif, réservé aux admin.
- LKM : simple, moins furtif, parfois impossible.
- /dev/kmem : très technique, accessibilité aléatoire.
- Patch du binaire (complément des attaques dynamiques)

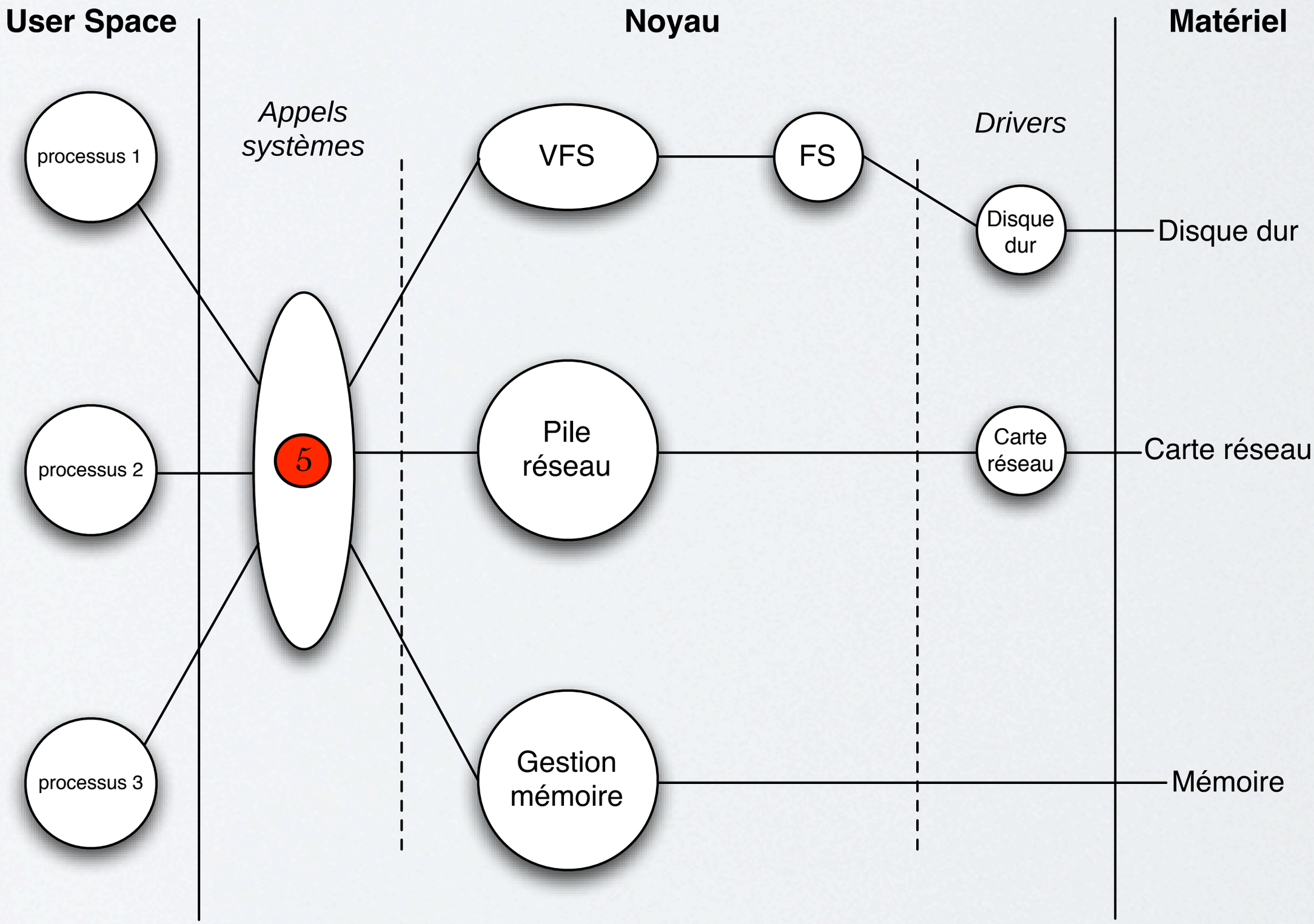
Récupération de données


User Space
Noyau
Matériel
1
Appels systèmes
Drivers
processus 1
processus 2
processus 3
VFS
FS
Disque dur
Disque dur
Pile réseau
Carte réseau
Carte réseau
Gestion mémoire
Mémoire


User Space
Noyau
Matériel
processus 1
processus 2
processus 3
Appels systèmes
VFS
FS
2
Drivers
Disque dur
Disque dur
Pile réseau
Carte réseau
Carte réseau
Gestion mémoire
Mémoire


User Space
Noyau
Matériel
processus 1
processus 2
processus 3
Appels systèmes
VFS
FS
Drivers
Disque dur
Disque dur
3
Pile réseau
Carte réseau
Carte réseau
Gestion mémoire
Mémoire


User Space
Noyau
Matériel
processus 1
processus 2
processus 3
Appels systèmes
VFS
FS
Drivers
Disque dur
Disque dur
Pile réseau
4
Carte réseau
Carte réseau
Gestion mémoire
Mémoire


User Space
Noyau
Matériel
processus 1
processus 2
processus 3
Appels systèmes
5
VFS
FS
Drivers
Disque dur
Disque dur
Pile réseau
Carte réseau
Carte réseau
Gestion mémoire
Mémoire

Hook d'appel système Version 1 : simple

- Hook intéressant : **read**, **write** ...
- Simplicité
- Furtivité limitée
- Nécessite l'accès à la table

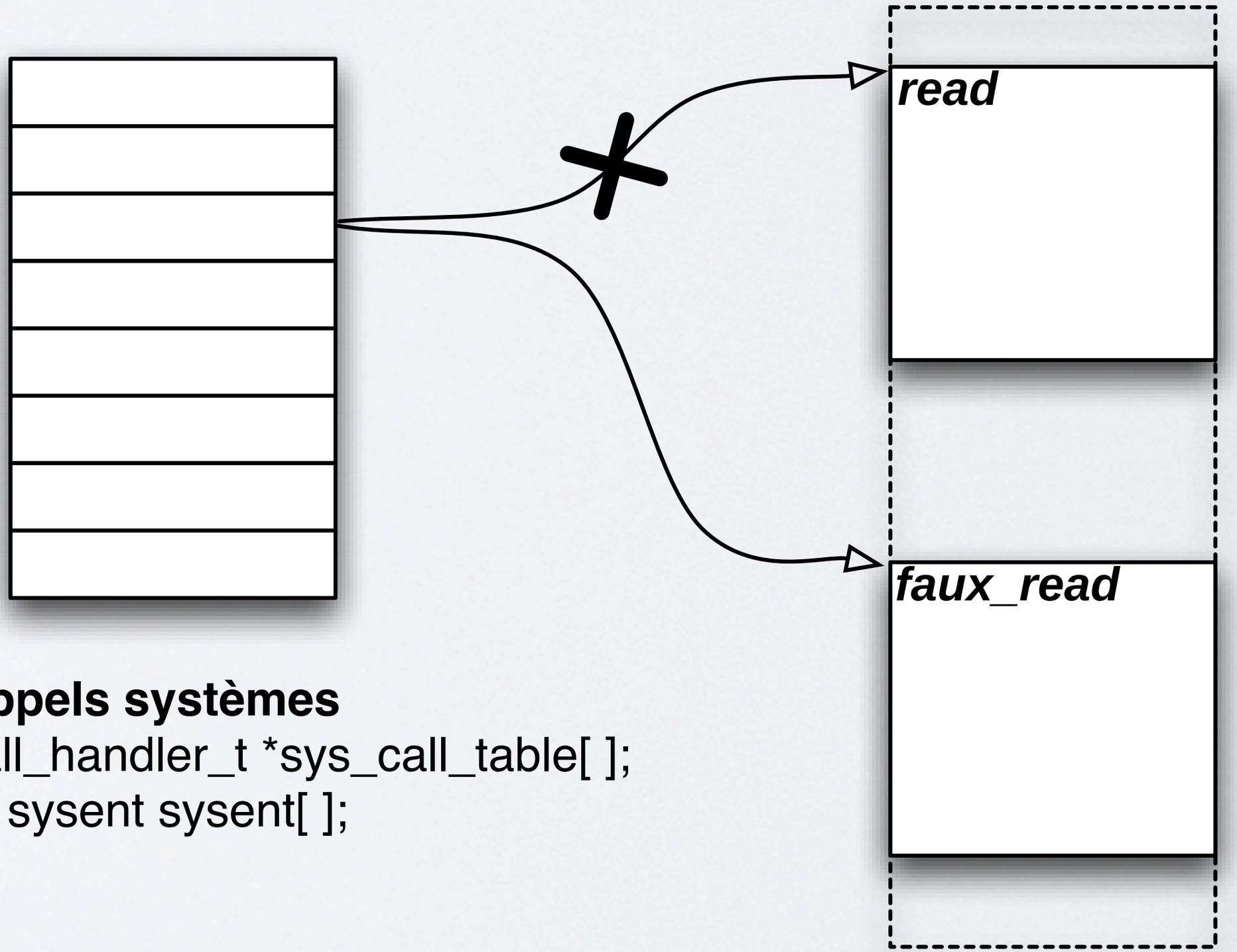


Table des appels systèmes

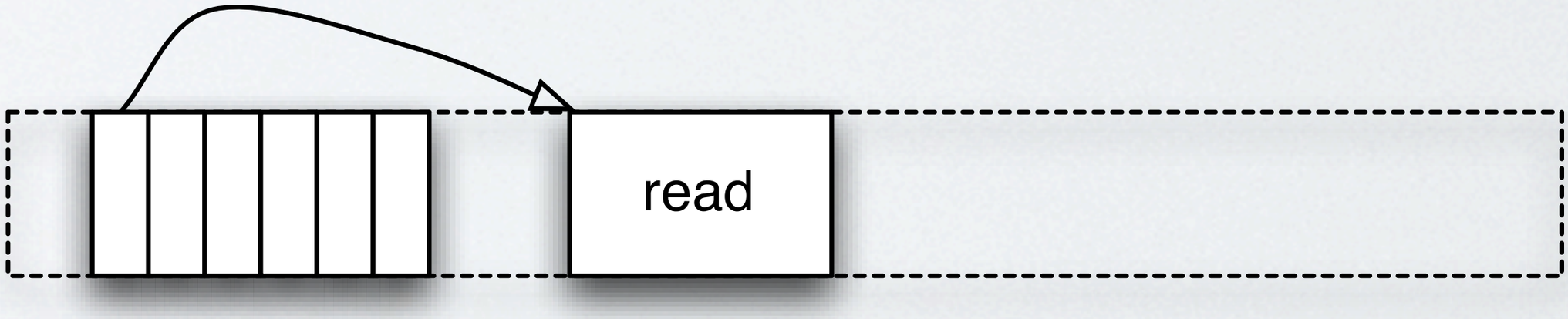
Linux : syscall_handler_t *sys_call_table[ ];

BSD : struct sysent sysent[ ];

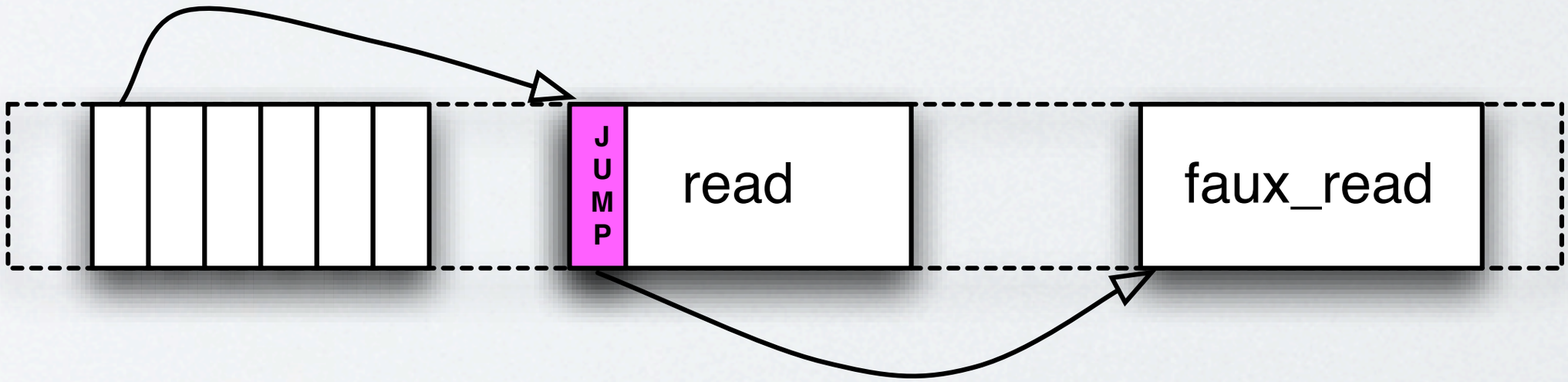
Hook d'appel système version 2 : JUMP

- Aucune modification de la table
- Recodage obligatoire de la routine
- Dépendance vis-à-vis de l'architecture
- Furtivité accrue

Avant



Après



Interruptions

- Retrouver l'adresse de l'IDT
- Fournir l'adresse de notre routine
- Réalisable par /dev/kmem ou LKM
- Très bas niveau, extrêmement technique, non portable

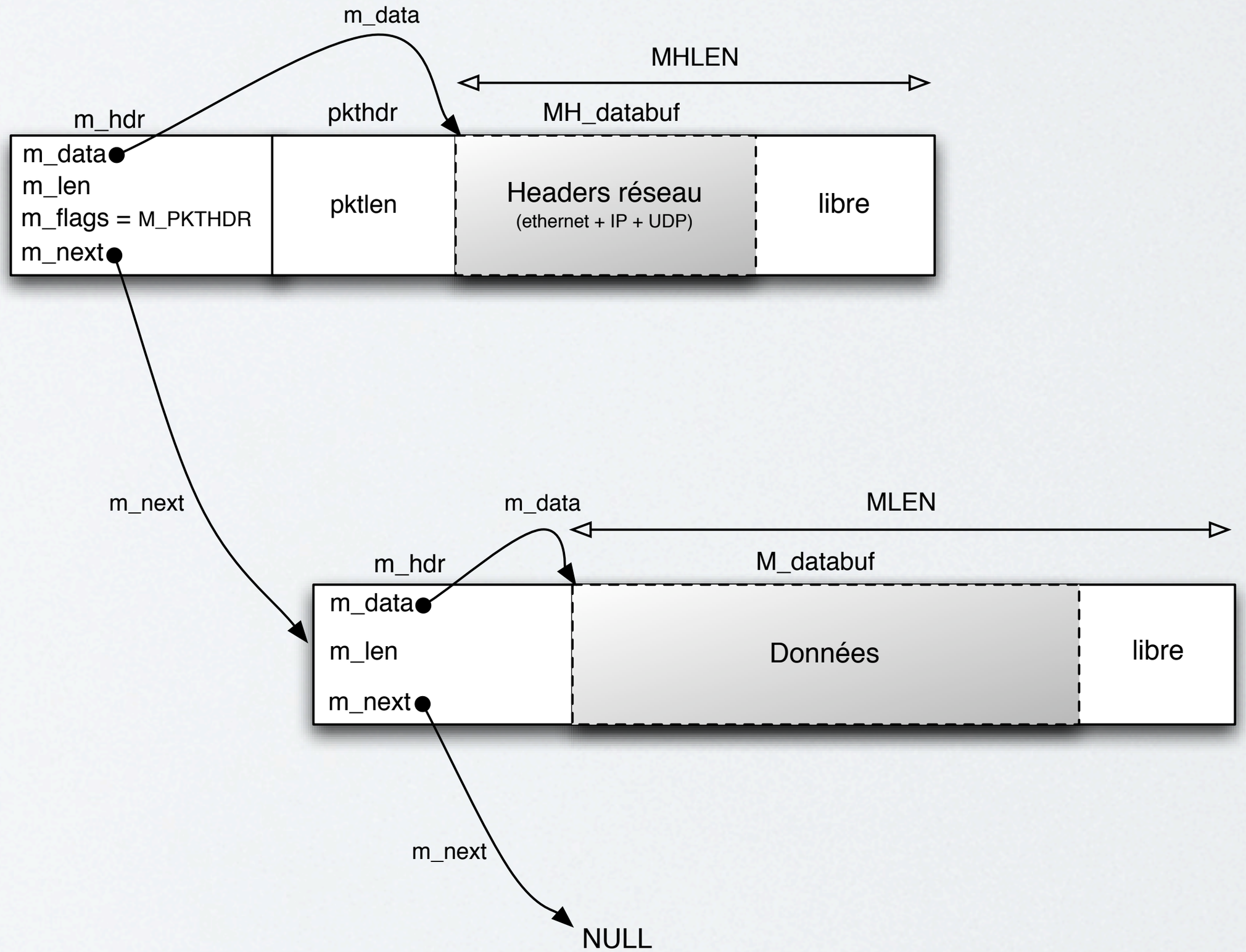
Conclusion

- Larges possibilités de hooks
- Nombreuses techniques
- Furtivité variable

Emission de trames

Juste avant le driver

- Allocation mémoire
- Remplissage de la structure représentant la trame
- Checksum
- Mise en queue et émission


m_data
MHLEN
m_hdr
pkthdr
MH_databuf
m_data
m_len
m_flags = M_PKTHDR
m_next
pktlen
Headers réseau
(ethernet + IP + UDP)
libre
m_next
m_data
MLEN
m_hdr
M_databuf
m_data
m_len
m_next
Données
libre
m_next
NULL

```
s = splnet();

/* If output queue is already full, we drop the packet */
if (IF_QFULL(ifq)) {
  m_freem(m);
  return -1;
}

/* Else we can enqueue our mbuf */
IF_ENQUEUE(ifq, m);

/* If interface is active, we can start sending */
if (!(ifp->if_flags & IFF_OACTIVE))
    (*ifp->if_start)(ifp);

splx(s);
```

Conclusion

- Pas de difficulté technique majeure
- Bas niveau, i.e. au plus proche du driver
- Furtivité (considérée plus tard)

Interactions

Interaction

- Masquer
  - flux réseau
  - fichier
  - processus
- Récupération de fichiers
- Emission de trafic réseau

Conclusion

- Diminution de la furtivité
- Valeur ajoutée
- Attaques “génériques”

Furtivité

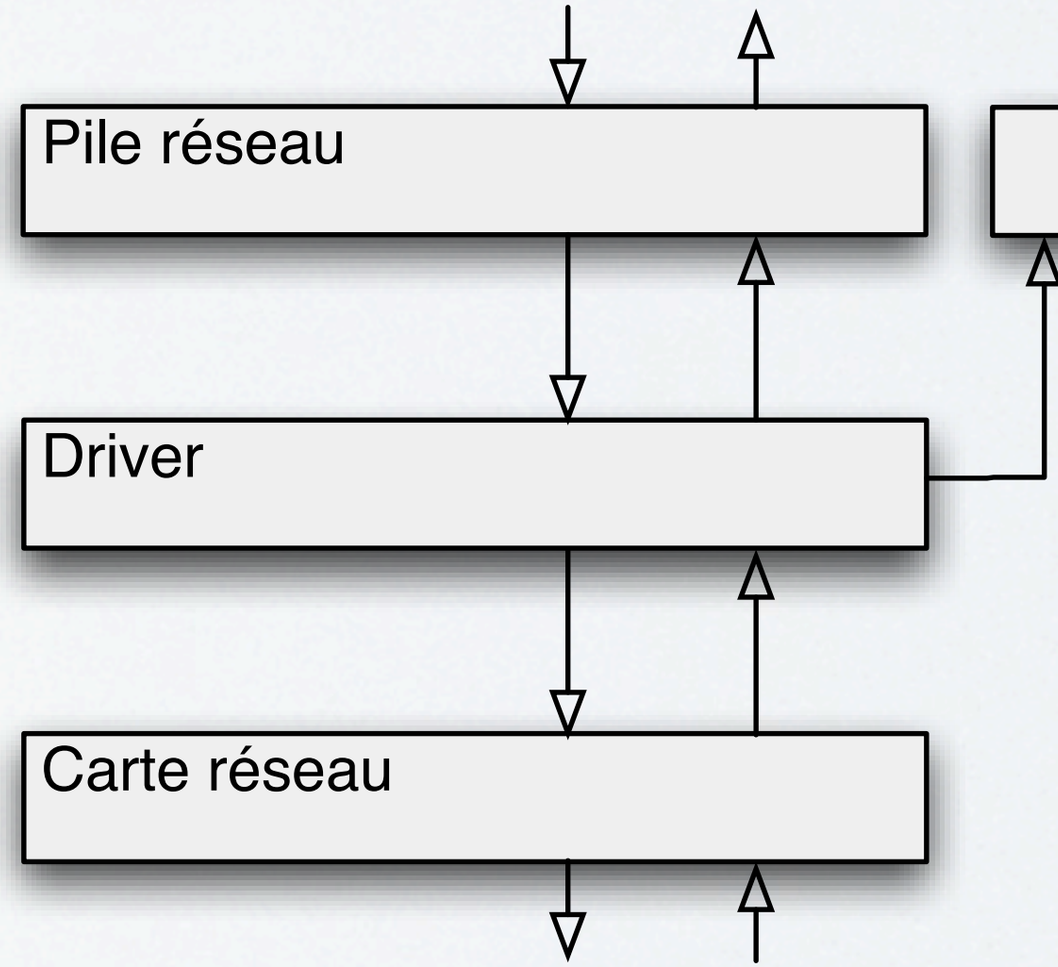
Locale

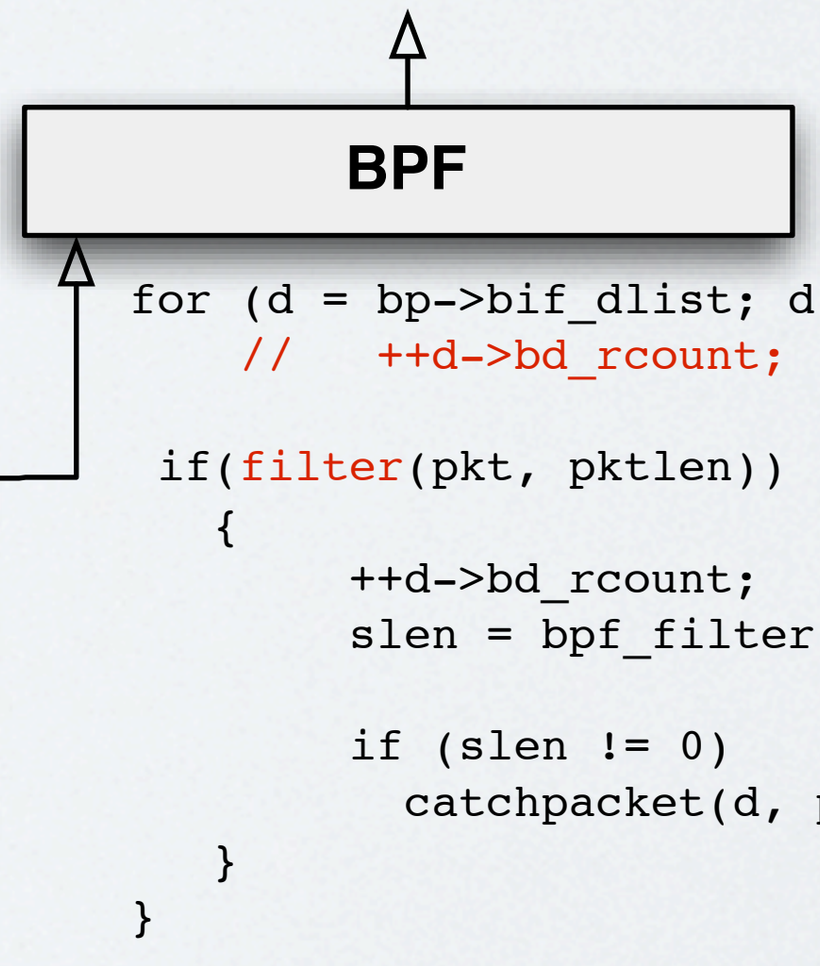
- Masquer la présence d'un module
- Réseau : masquer le trafic entrant/sortant
  - **bpf**
  - pile réseau

Distante

- NIDS
  - **p0f**
  - logs firewall
  - **snort**
- Penser au retour
- Trafic légitime

Masquage de trafic






```
for (d = bp->bif_dlist; d != 0; d = d->bd_next) {
    //   ++d->bd_rcount;

 if(filter(pkt, pktlen))
   {
        ++d->bd_rcount;
        slen = bpf_filter(d->bd_filter,
                          pkt, pktlen, pktlen);
        if (slen != 0)
          catchpacket(d, pkt, pktlen, slen, memcpy);
   }
}
```

Conclusion

- Capacité technique de l'attaquant
- Paranoïa et capacités de l'adversaire
- Architecture réseau

Prévention

- Limiter l'accès aux symboles
  - **strip /bsd**
  - System.map
- Interdire les modules
- Intégrité de l'image du noyau
- Limiter les accès à /dev/kmem
- Surveillance du traffic
- Surveiller les reboot

Des questions ?

Remerciements

- Laurent Oudot (recette de cuisine)
- Fred, Phil et Serpillière pour la relecture
- Team RSTACK
- You guys rule !