Hyper-V

Analyse, exploitation et défense

Jordan Rabet, Microsoft OSR

ANALYSE DYNAMIQUE DU CODE

On entend souvent parler des équipes d'analyse offensive « Red Team » chez Microsoft

- Elles faisaient quoi pendant ce temps?

Ce n'est pas pour minimiser leur travail – ces équipes bossent fort sur des nouvelles technologies d'exploitation et la protection des conteneurs Hyper-V et de l'isolation des hôtes

- Mais ou sont passe les principes de base?

Je trouve que l'industrie est passée dans un mode réactionnel face aux attaques, et non stratégique

- « On a découvert avec Eternal* que les RCE noyaux sont des vraies choses qui existent, on va donc passer 5 ans la dessus, au détriment des autres environnements non-sécurisés »
  - C'est comme enlever ses chaussures au détecteurs de métal aux aéroports américains…

Je suis d’accord avec Alex sur le fond

- La sécurité ne prend pas une place suffisamment importante dans l’éducation de nos ingénieurs software
  - Ce qu’on veut c’est pas ajouter un cours « sécurité » au programme, c’est intégrer la sécurité comme principe fondateur du génie logiciel
- L’économie des travaux offensifs vs défensifs dans l’industrie a un impact qu’on ne comprend pas encore forcément et qui est certainement négatif
- Même si on réécrit tout en Rust, il sera toujours possible d’introduire des bugs de logique comme celui qu’il a présenté sur NtDxgkEnumProcessList – l’éducation en sécurité restera importante

Je pense aussi qu'il faut nuancer..

- On enseigne déjà les principes de corruption de mémoire, juste sans dire que c'est de la sécurité
  - À l'Ensimag, je crois bien qu'on nous a appris à vérifier les index et à éviter les race condition – mais c'était plutôt dans une optique de fiabilité que de sécurité
  - Donc est-ce que juste dire qu'il faut vérifier les index parce que c'est pour la sécurité ça va suffire?
- Les ingénieurs en sécurité introduisent également des bugs
  - Personne n'est parfait
- La complexité des systèmes qu'on demande à nos ingénieurs d'implémenter joue un rôle au moins aussi important que leur éducation
  - Ok, les bugs d'Alex étaient très « low hanging fruit » qui ne devraient pas arriver, mais allez voir certains bugs qu'on a dans les JIT engines par exemple…
  - À moins que du jour au lendemain plus personne ne veuille de JavaScript, de drivers graphiques à haute performance, ou de cloud, il y a peu de chances que ça change

Il y aura toujours des bugs

- Est-ce que ça veut dire qu'on ne peut pas réduire le nombre de bugs via une meilleure éducation et de meilleurs processus de validation?
  - Non, au contraire
- Est-ce que ça veut dire qu'il faut également être pragmatique et présupposer que tout logiciel est potentiellement exploitable?
  - Je pense que oui
- Dans l'idéal, on veut investir dans des solutions telles que:
  - Il est très difficile pour un développeur d'introduire un bug exploitable
    - Langages memory-safe, élimination de classes de vulnerabilités…
  - Même si un développeur introduit un bug, il est extrêmement dur à exploiter
    - Tout ce qui est mitigations contre la corruption de mémoire

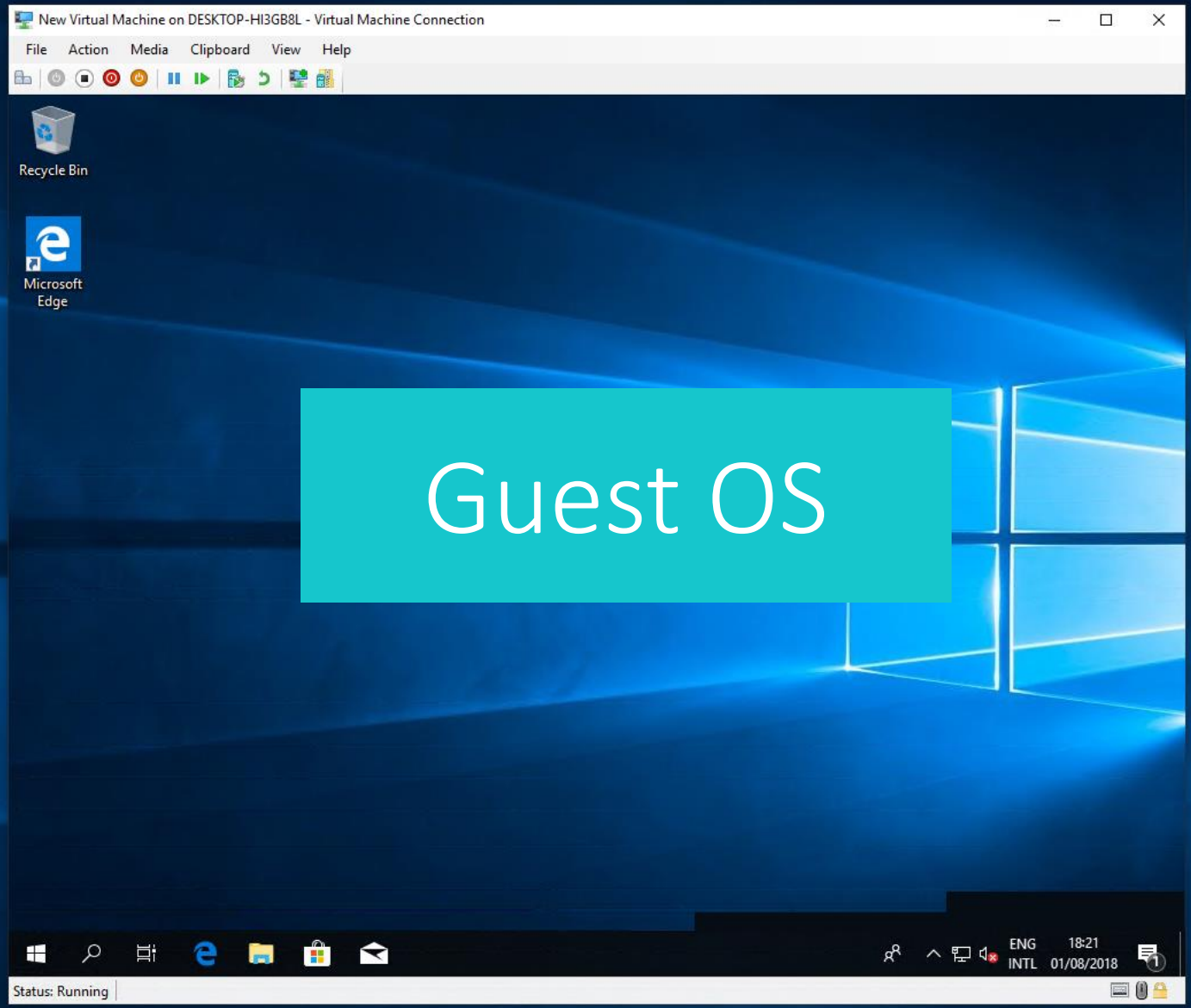
Tout ça pour expliquer… **OSR**

- On écrit des exploits contre des produits Microsoft
  - Pour nous donner une idée en interne de la difficulté que ça implique
  - Pour voir l'impact en terme de dégâts qu'un exploit peu encourir
  - Le plus important: pour être proactif et mitiger de nouvelles techniques d'exploitation avant même qu'elles ne soient utilisées contre nous
    - Dans ce sens, notre rôle est mixte: on fait de l'offensif purement au service du défensif – aucune opération OSR n'est complète sans un ensemble d'idées/prototypes de mitigations
- Le but de cette présentation
  - Montrer qu'une red team peut faire plus que juste écrire des exploits en contribuant directement au hardening des produits qu'elle cible
  - Partager les détails techniques d'un exploit assez complexe qu'on a écrit
  - Avec un peu de chance, donner le gout de la recherche offensive contre Hyper-V ☺
    - On va avoir une démo en live vers la fin ce sera cool

Intro à Hyper-V

Recycle Bin

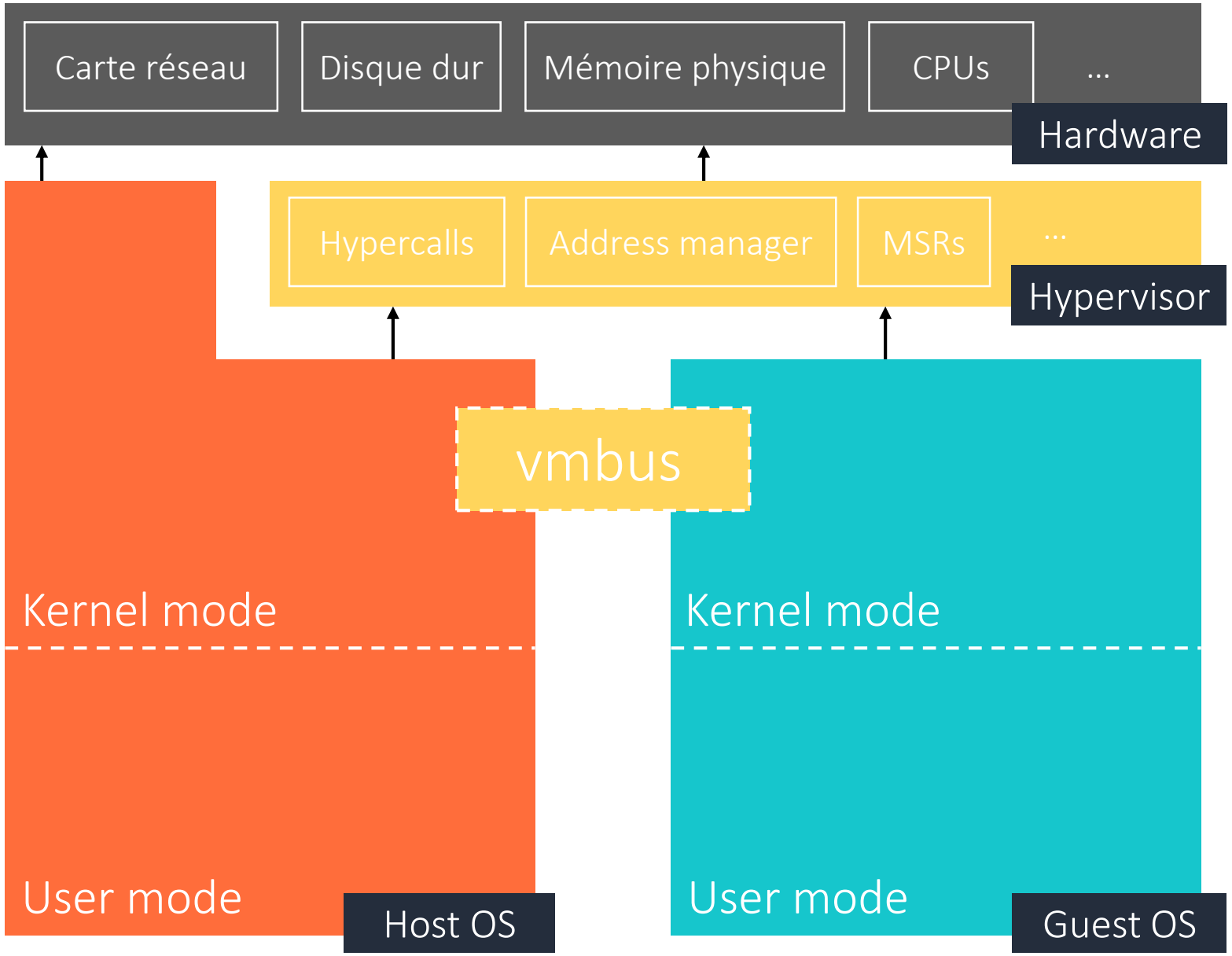
Host OS



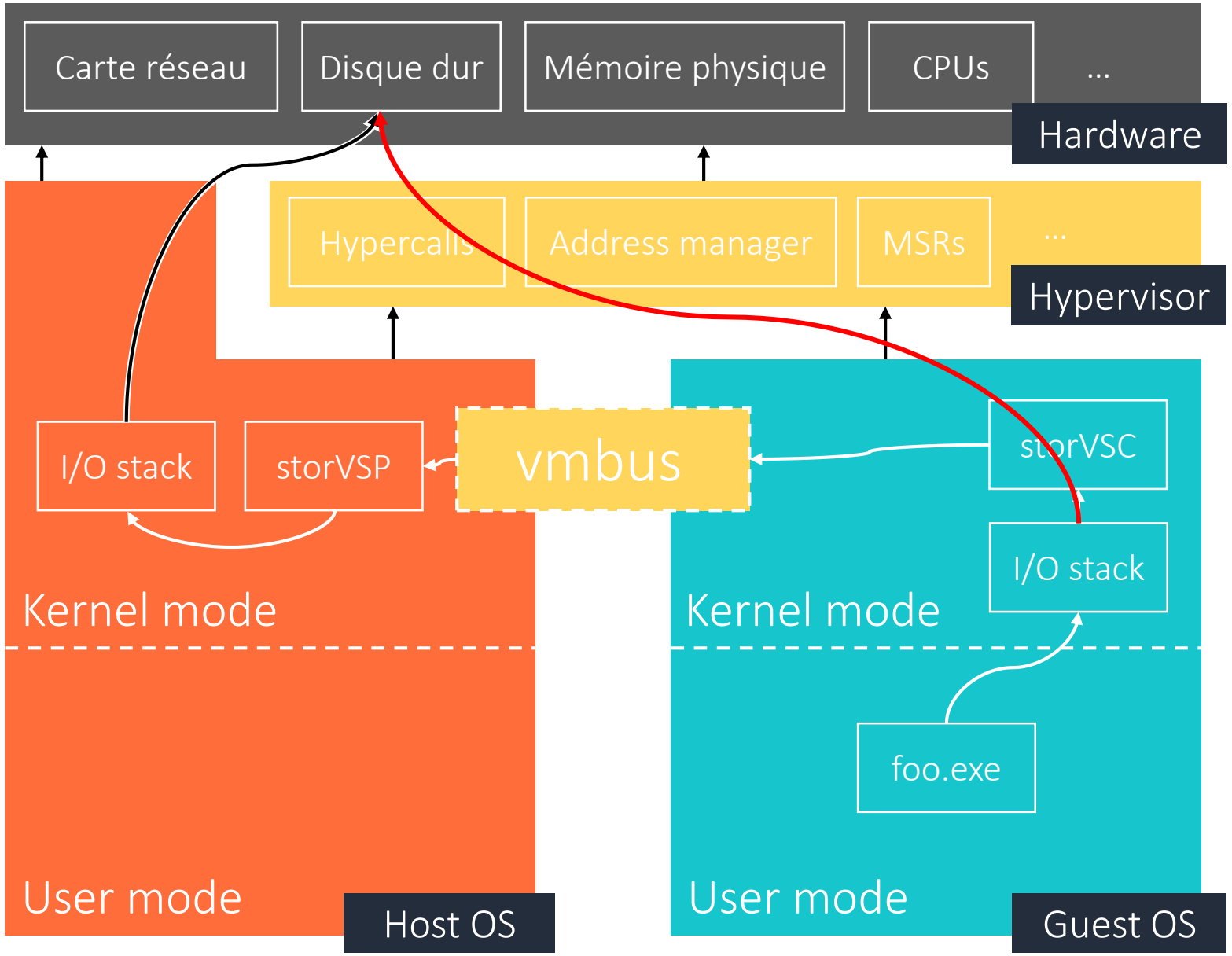
Guest OS

Type here to search

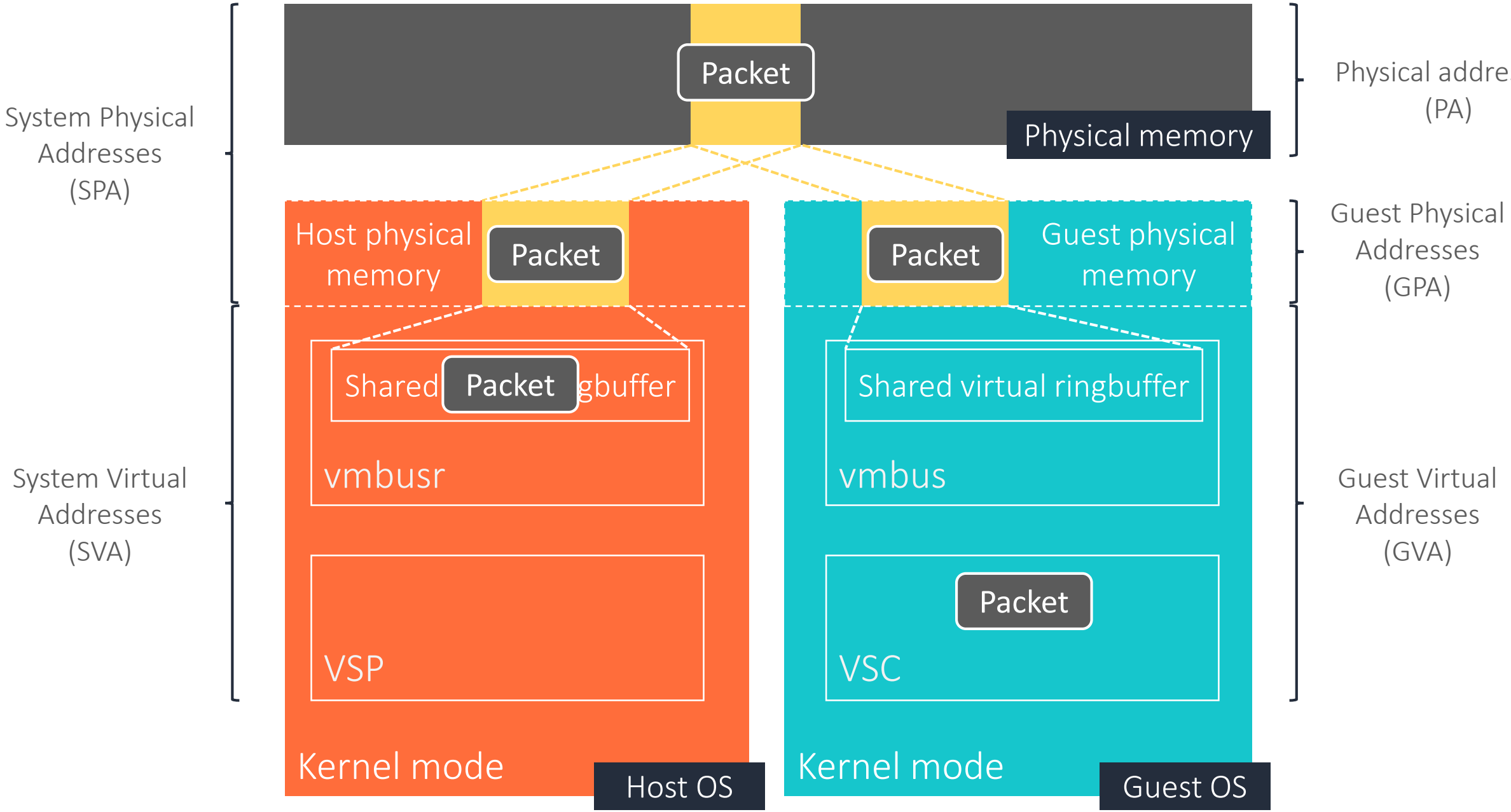
6:21 PM 8/1/2018



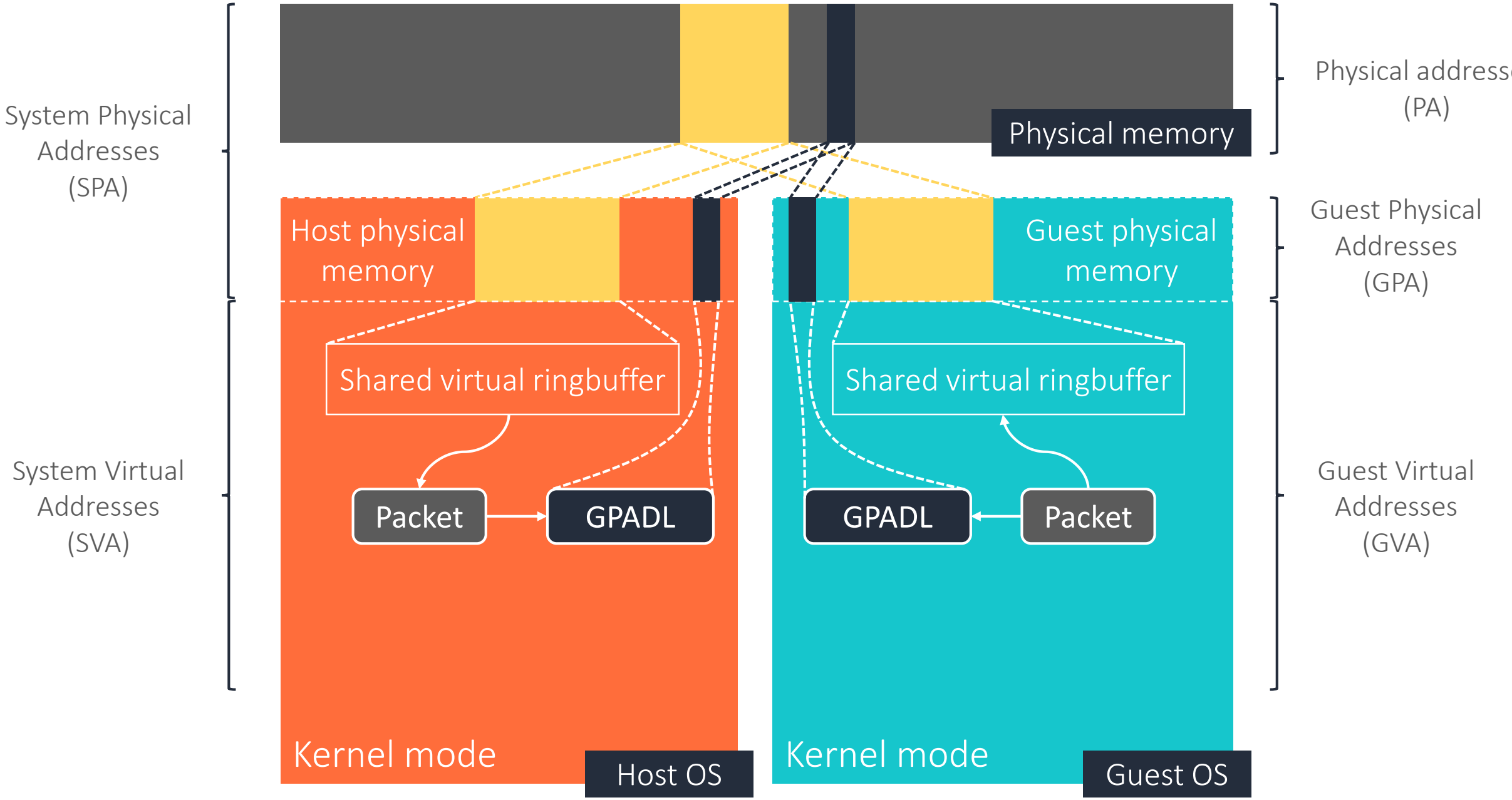
Architecture de Hyper-V: organisation



Architecture de Hyper-V: accès au hardware depuis le Guest OS

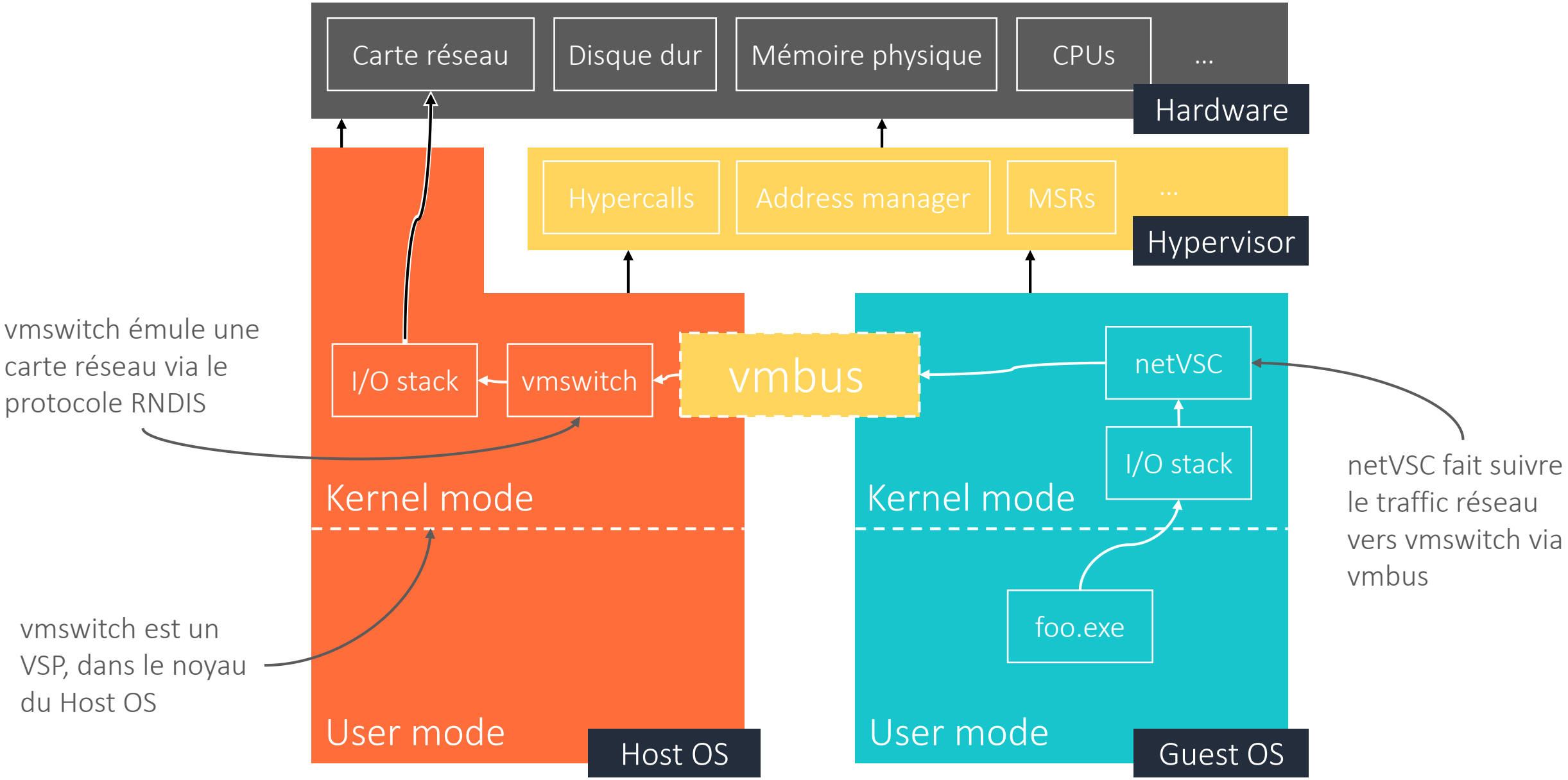


vmbus: paquets simples

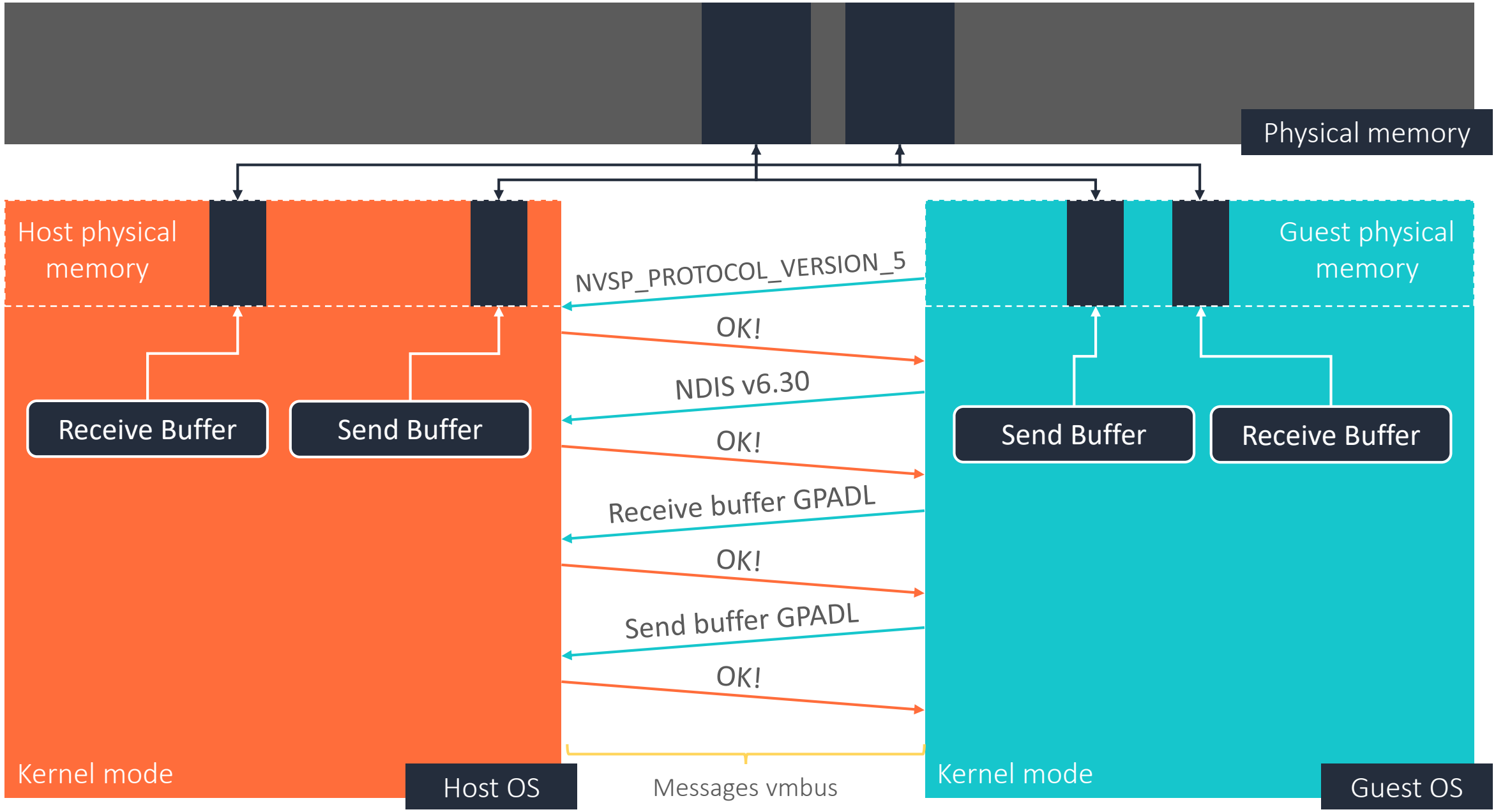


vmbus: paquet simple accompagné d'un "GPADL" (GPA Descriptor List)

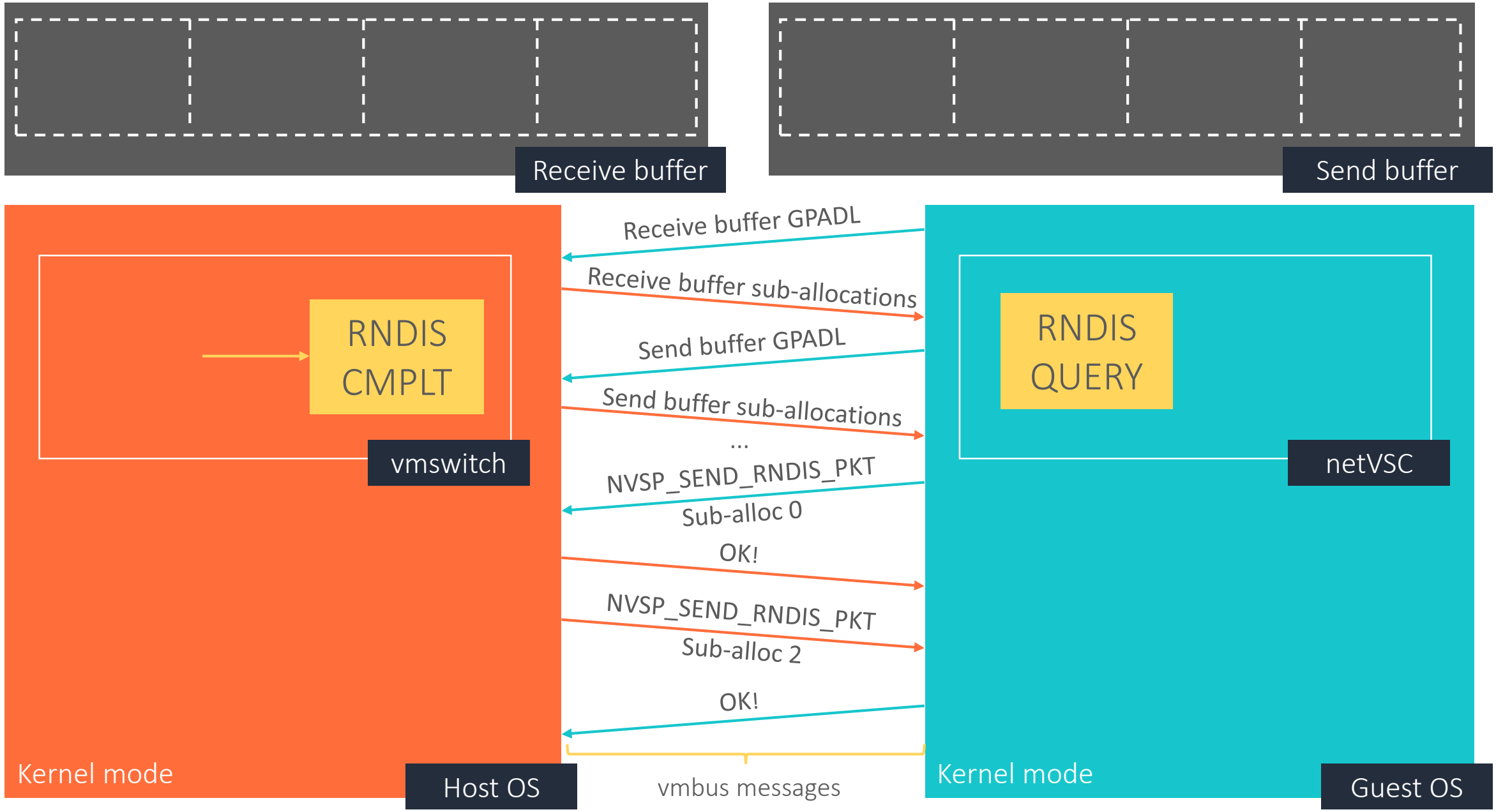
Analyse d’un VSP: vmswitch



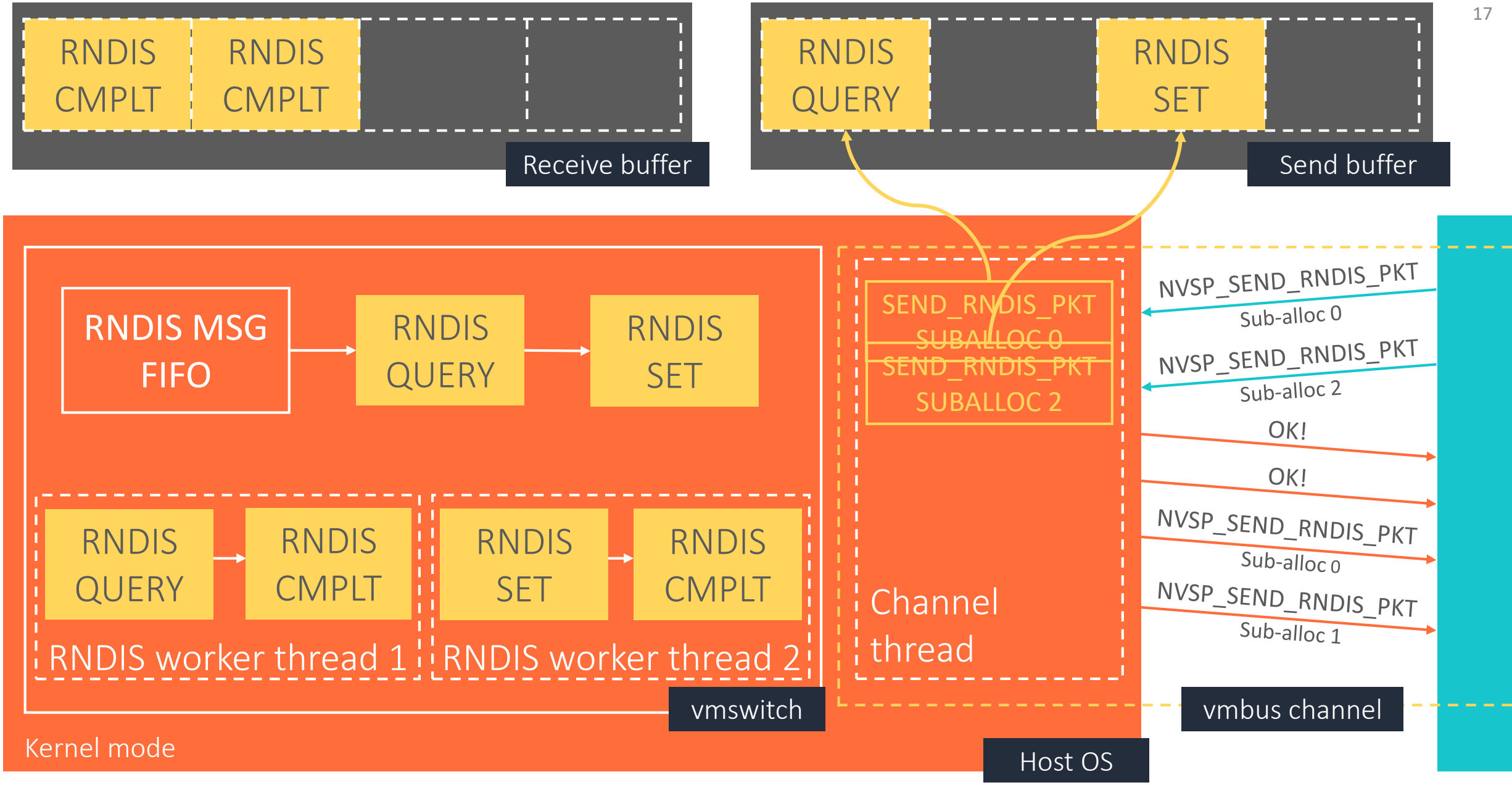
vmswitch: composant de virtualisation réseau



vmswitch: séquence d'initialisation

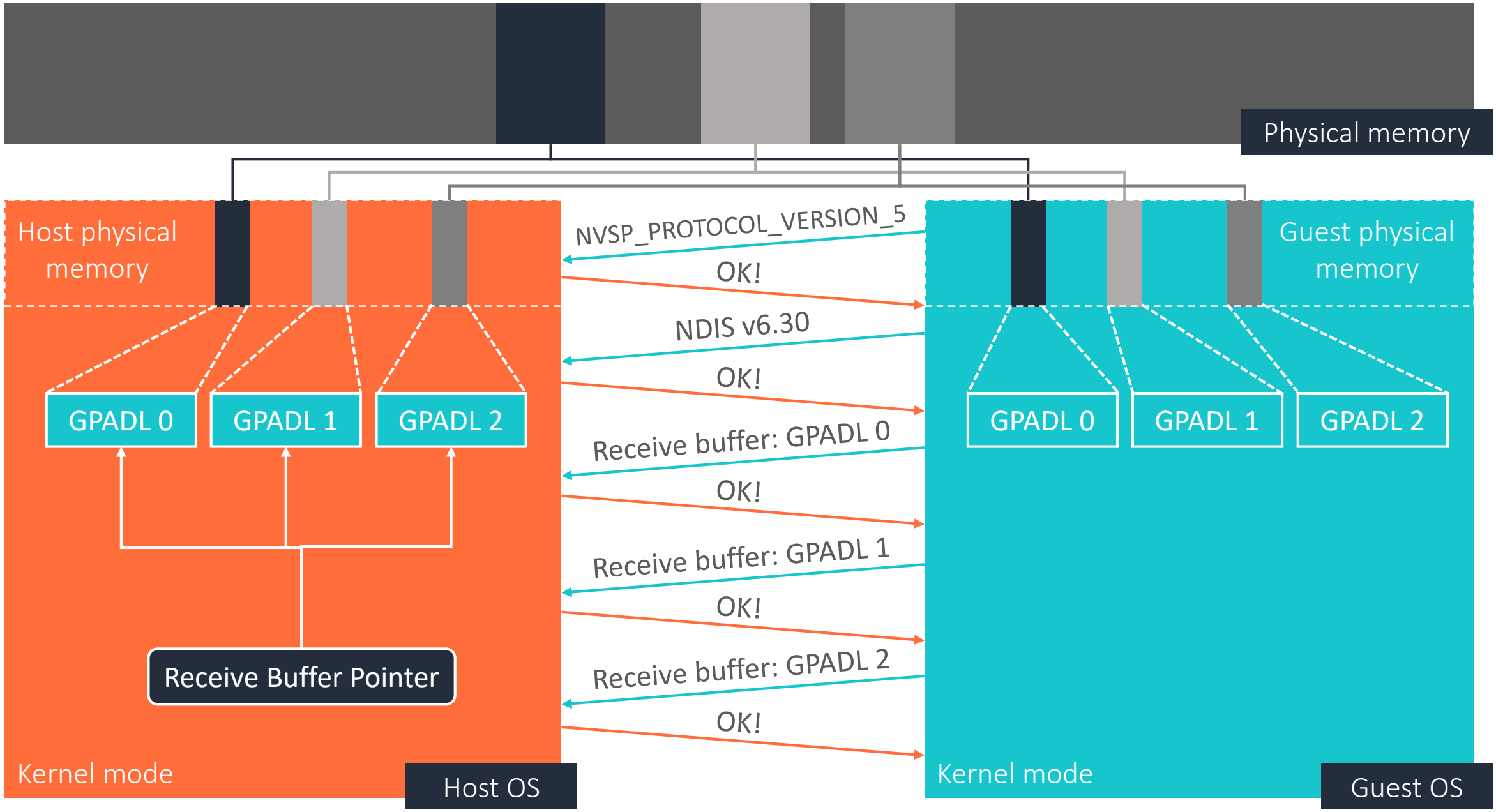


vmswitch: envoyer un paquet RNDIS



vmswitch: comment les messages RNDIS sont-ils traités?

La vulnérabilité



Problème dans la sequence d'initialisation…

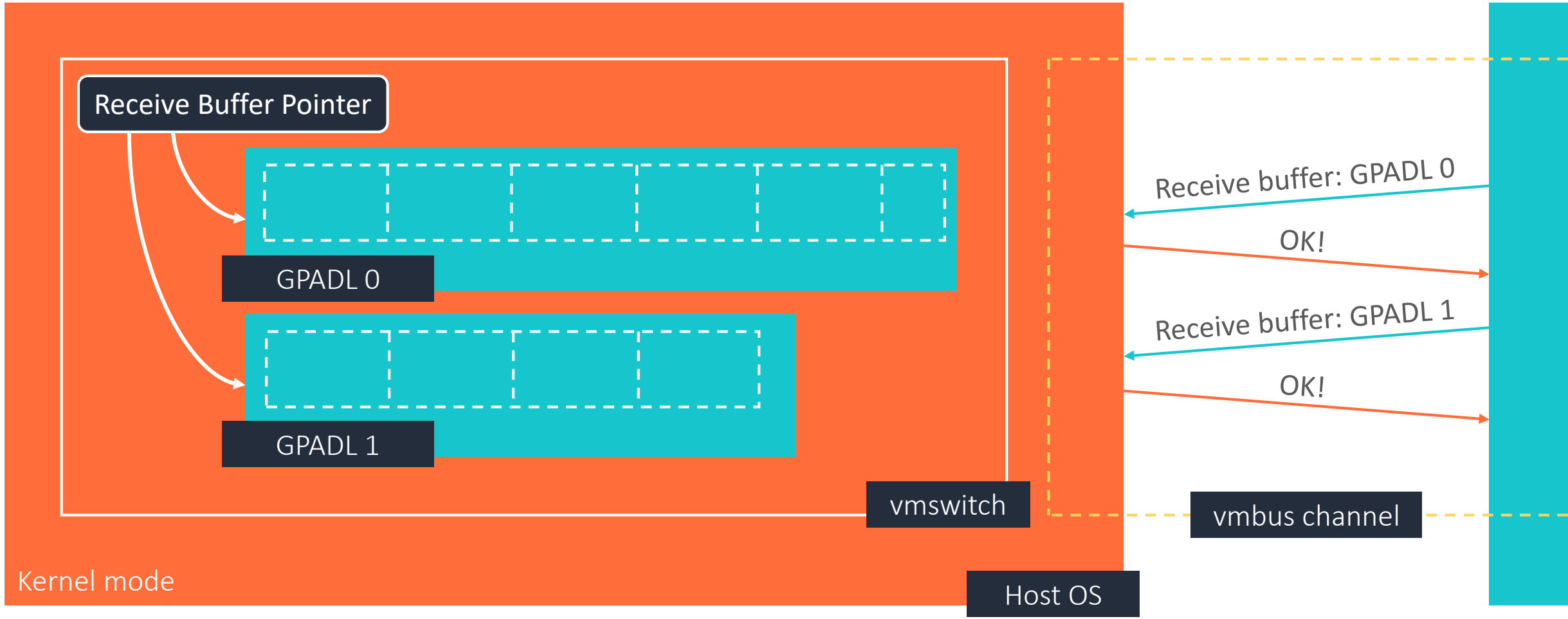
Mise à jour du receive buffer

La mise à jour du receive buffer n’est pas atomique

1. Le pointeur vers le buffer est mis à jour
2. Les sous-allocations sont **ensuite** calculées et mises à jour

Aucune synchronisation sur le receive buffer

- Un autre thread pourrait l’utiliser en parallèle



3

Mise à jour des sous-allocations

Mise à jour du receive buffer

Race condition sur le receive buffer

- Pendant une courte période, on peut induire des sous-allocations hors-limites
- Il est résulte une primitive très utile si:
  1. On peut contrôler ce qui est écrit dans le receive buffer
  2. On peut le faire pendant cette période
  3. On peut placer quelque chose d'utile à corrompre aux alentours du buffer

1. Mise à jour du pointeur vers le buffer
2. Calcul des limites des sous-allocations
3. Mise à jour des sous-allocations

Mise à jour du receive buffer

Comment exploiter le bug?

- Comment contrôler ce qui est écrit?
- Comment écrire exactement au bon moment?
- Que peut-on corrompre d’utile?

Comment exploiter le bug?

Comment contrôler ce qui est écrit?

Comment écrire exactement au bon moment?

Que peut-on corrompre d’utile?

Comment contrôler ce qui est écrit?

- Ce qui est écrit: paquets RNDIS de réponse du host OS
- Par exemple, la réponse à RNDIS_QUERY_MSG peut contenir des données que nous controlons

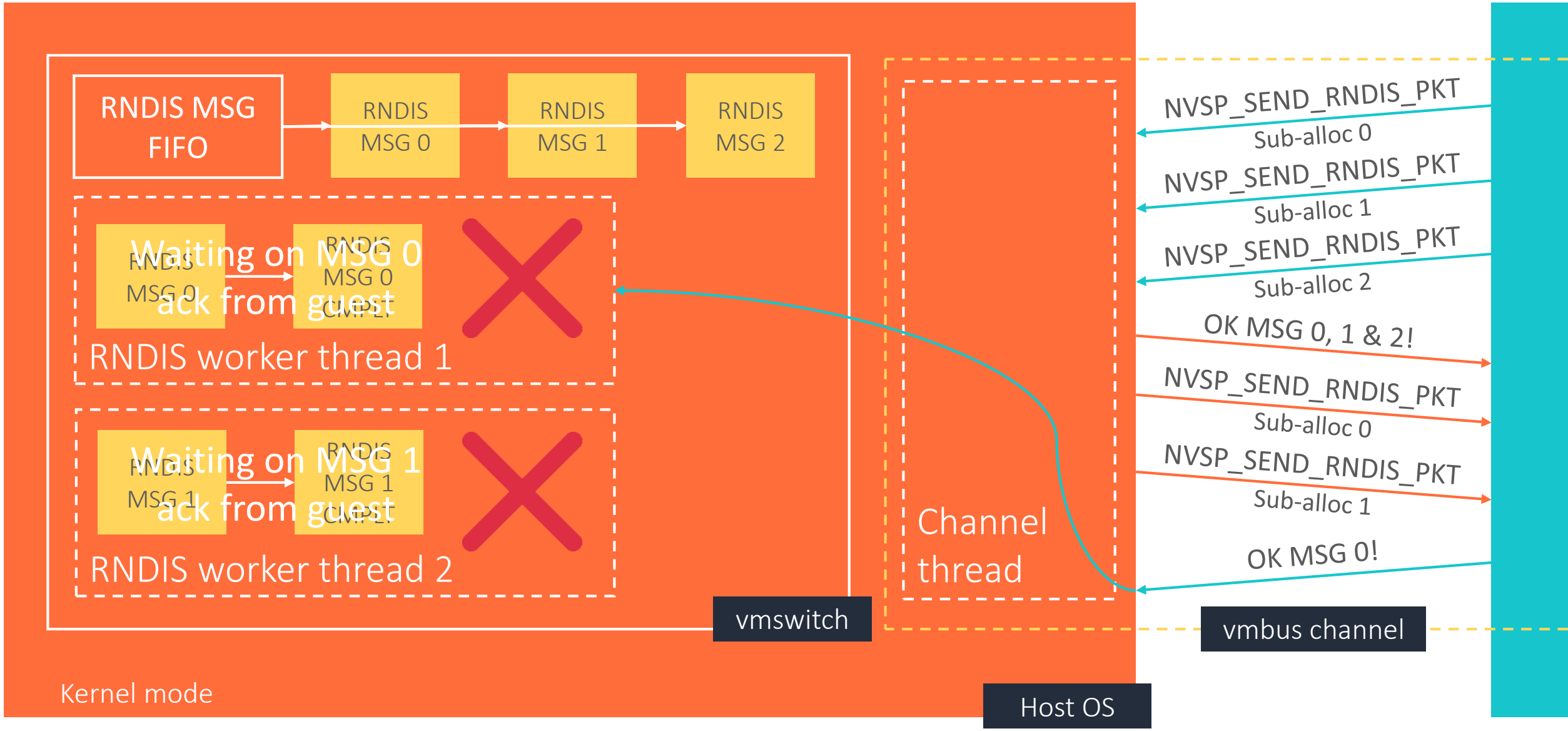
Offset	Size	Field
0	4	MessageType
4	4	MessageLength
8	4	RequestId
12	4	Status
16	4	InformationBufferLength
20	4	InformationBufferOffset

Comment exploiter le bug?


Comment contrôler ce qui est écrit?


Comment écrire exactement au bon moment?

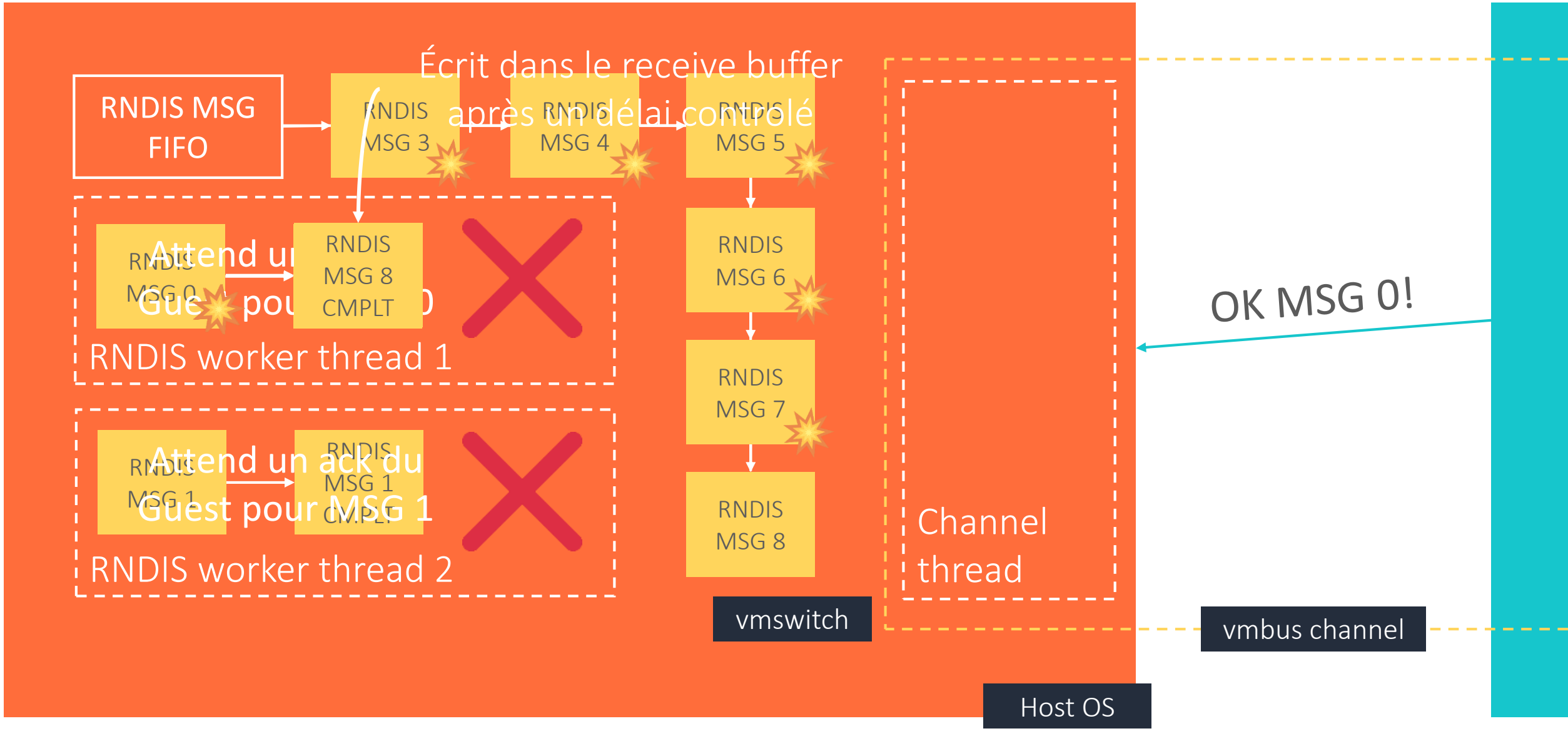
Que peut-on corrompre d’utile?



vmswitch: le traitement des messages RNDIS est asynchrone, mais pas complètement

Comment écrire exactement au bon moment: induire un délai sur un message RNDIS?

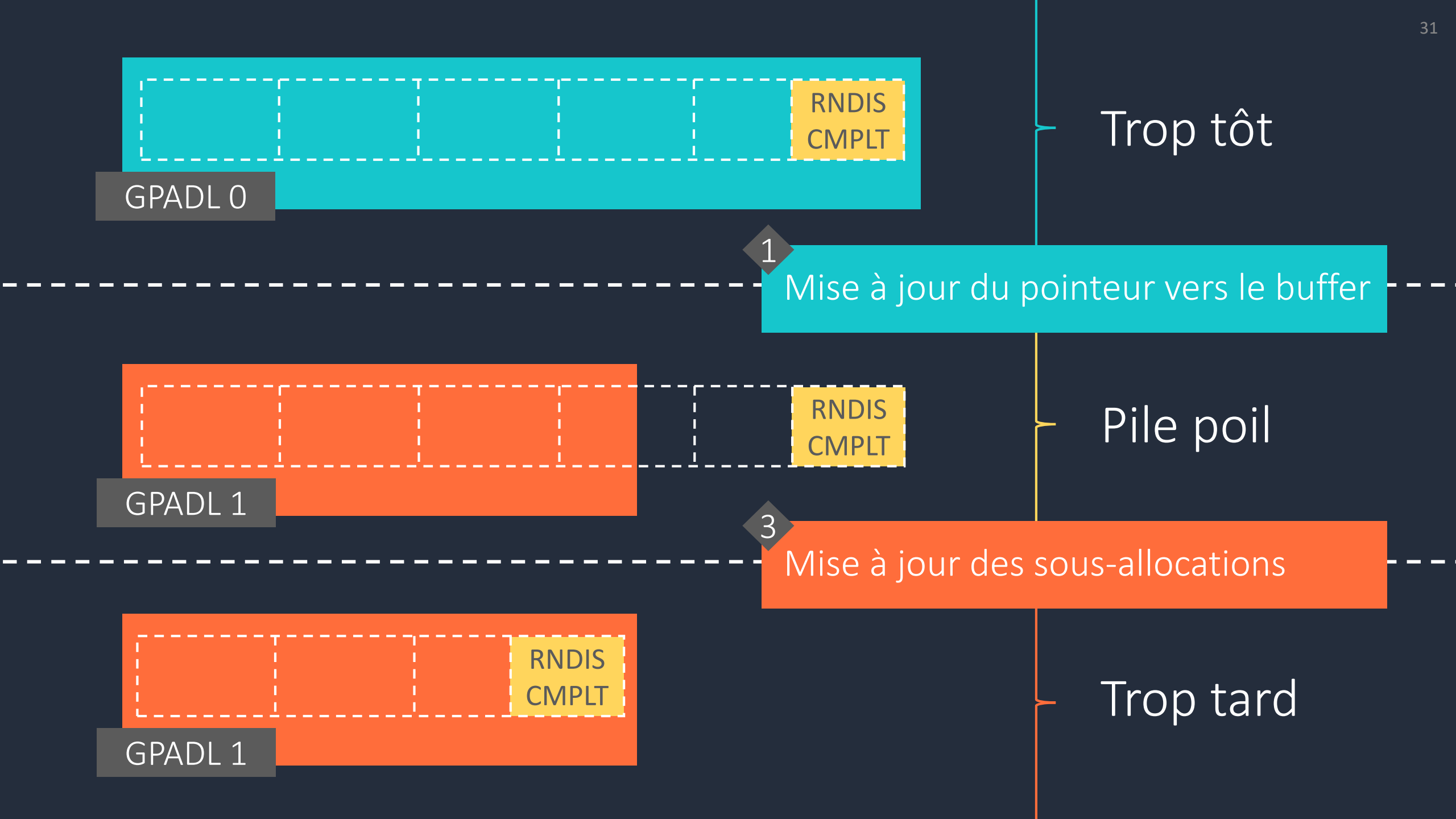
- On ne peut pas forcer le Host OS à écrire dans le receive buffer en continu…
  - Mais on n'a pas besoin de la faire en continu: il suffit d'écrire une seule fois au bon moment
  - => Peut-on induire un délai contrôlé sur le traitement d'un message RNDIS?
- A priori non: le protocole ne nous d'inclue pas cette fonctionnalité (surprenant…)
- Mais en pratique, toutes les opérations RNDIS ne nécessitent pas un ack du Guest
  - **Par exemple**: un paquet RNDIS_KEEPALIVE_MSG malformé est immédiatement rejeté
    - Après que ce message ait été rejeté, le Host OS va immédiatement traiter le prochain message en attente
- **Idée**: "cascade of failure"
  - On envoie un message RNDIS à chaque worker thread pour les bloquer
  - On enchaine *N* paquets RNDIS_KEEPALIVE_MSG malformés
  - On enchaine un unique paquet RNDIS valide à la fin, celui qui dont la réponse sera écrite dans le receive buffer



Comment écrire exactement au bon moment

Comment écrire exactement au bon moment: quel délai induire?

- On peut induire un délai de *N* messages malformés… mais quel est le bon *N*?
  - Chaque tentative induit une fuite de mémoire dans le Host OS, donc on ne peut pas essayer toutes les valeurs possibles de *N*
- Peut-on distinguer les différents résultats de nos tentatives?
  - Si oui, on pourrait chercher la bonne valeur de *N*
  - Si on a écrit trop tôt, on augmente *N*
  - Si on a écrit trop tard, on réduit *N*
  - Si on a écrit exactement au bon moment… on a gagné ☺


RNDIS CMPLT
GPADL 0
Trop tôt
1
Mise à jour du pointeur vers le buffer
RNDIS CMPLT
GPADL 1
Pile poil
3
Mise à jour des sous-allocations
RNDIS CMPLT
GPADL 1
Trop tard

Comment exploiter le bug?


Comment contrôler ce qui est écrit?


Comment écrire exactement au bon moment?


Que peut-on corrompre d’utile?

Que peut-on corrompre d'utile: d'autres GPADL/MDL et… des piles d'exécution?

```
0: kd> !address @@c++(ReceiveBuffer)
Usage:
Base Address:           ffffdd80`273d5000
End Address:            ffffdd80`27606000
Region Size:            00000000`00231000
VA Type:                SystemRange

0: kd> !address

...
ffffdd80`273c6000 ffffdd80`273cc000  0`00006000 SystemRange   Stack  Thread: ffffc903eed10800
ffffdd80`273cc000 ffffdd80`273cf000  0`00003000 SystemRange
ffffdd80`273cf000 ffffdd80`273d5000  0`00006000 SystemRange   Stack  Thread: ffffc903f182b080
ffffdd80`273d5000 ffffdd80`27606000  0`00231000 SystemRange
ffffdd80`27606000 ffffdd80`2760c000  0`00006000 SystemRange   Stack  Thread: ffffc903f181f080
ffffdd80`2760c000 ffffdd80`2760d000  0`00001000 SystemRange
ffffdd80`2760d000 ffffdd80`27613000  0`00006000 SystemRange   Stack  Thread: ffffc903ee878080
...
```

Que peut-on corrompre d'utile: piles d'exécution de thread kernel

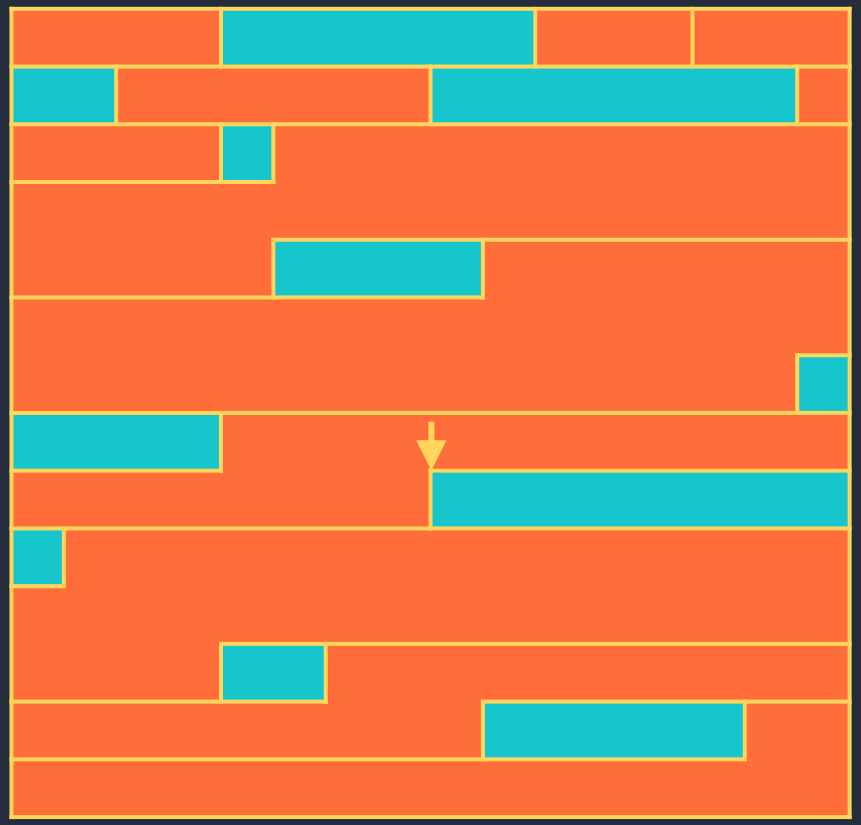
- Piles d'exécution de thread kernel Windows
  - Elles font toutes la même taille: 7 pages
    - 6 pages utilisées par la pile
    - 1 page de garde
  - Allouées dans la même région mémoire que les MDL/GPADL (« SystemPTE »)
  - Excellente cible de corruption mémoire: nous permet de corrompre une adresse de retour
- Difficultées
  - Comment l'allocateur de la région « SystemPTE » fonctionne-t-il?
  - Peut-on l'influencer pour placer une pile d'exécution juste à côté de notre receive buffer?
  - Sommes-nous même capables de « placer » une pile d'exécution? Ça nécessiterait de créer de nouveaux threads dans le Host OS sur demande…

SystemPTE « heap massaging »: primitives d'allocation

- **Receive/send buffers**: nous permettent de mapper autant de GPADL qu'on veut
  - « autant qu'on veut » : il y a une limite, mais elle est élevée
  - Le Guest contrôle la taille des GPADL, et là aussi il y a une limite mais elle est élevée
- Certains messages vmswitch sont traités sur des worker threads
  - Ces threads sont gérés par le kernel NT
  - Si tous les worker threads sont occupés, le kernel en crée de nouveaux
- **Idée**: si on peut forcer vmswitch à utiliser suffisamment de worker threads, nous pourrons peut-être forcer le kernel à créer de nouveaux threads
- On a de la chance: un bug dans vmswitch nous permettait de créer autant de threads deadlockés qu'on veut
- **Résultat**: on peut spray des thread dans le kernel du Host, et donc également des piles d'exécution

Allocateur SystemPTE

- Utilise une bitmap
  - Chaque bit représente une page
  - Bit 0: page libre, Bit 1: page allouée
- Utilise un « indice » pour allouer
  - Scan la bitmap en partant de l'indice
  - Reprend au début de la bitmap s'il en atteint la fin
  - Place le nouvel indice au bout de la nouvelle allocation
- La bitmap est étendue s'il n'y a pas assez de place
- Exemple 1: allocation de 5 pages



Page libre

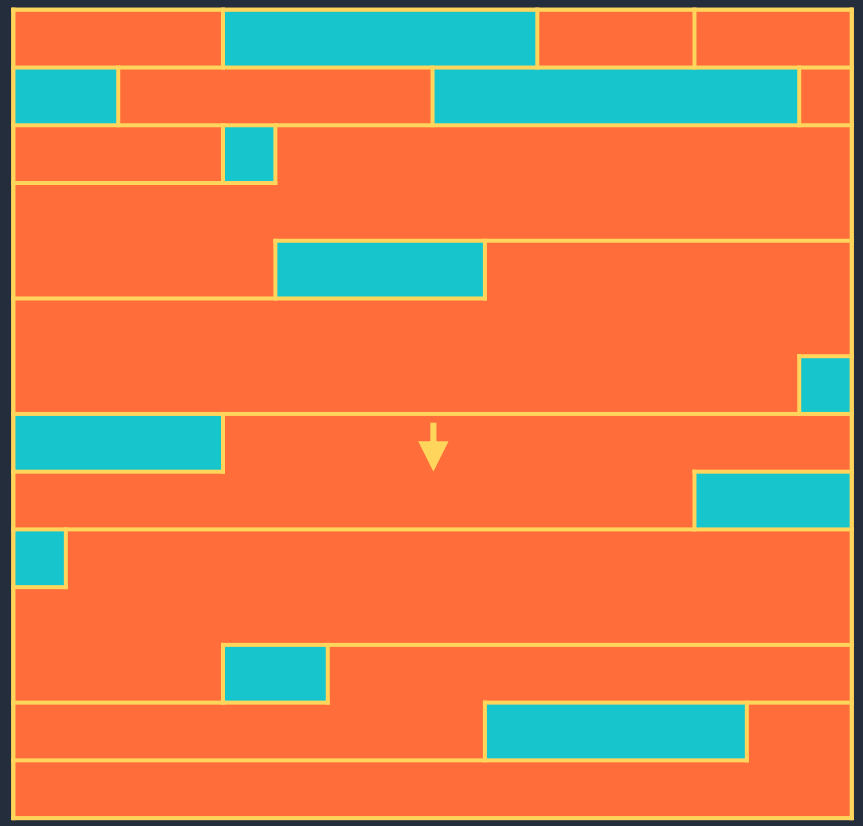
Page allouée

Indice bitmap

Bitmap d'allocation

Allocateur SystemPTE

- Utilise une bitmap
  - Chaque bit représente une page
  - Bit 0: page libre, Bit 1: page allouée
- Utilise un « indice » pour allouer
  - Scan la bitmap en partant de l'indice
  - Reprend au début de la bitmap s'il en atteint la fin
  - Place le nouvel indice au bout de la nouvelle allocation
- La bitmap est étendue s'il n'y a pas assez de place
- Exemple 1: allocation de 5 pages



Page libre

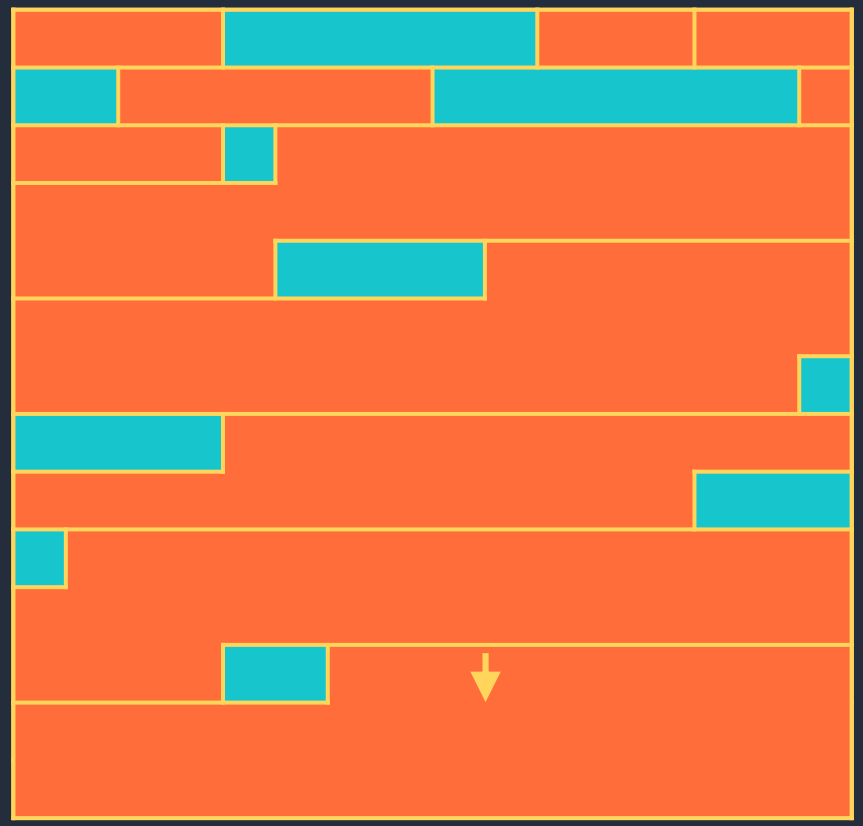
Page allouée

Indice bitmap

Bitmap d'allocation

Allocateur SystemPTE

- Utilise une bitmap
  - Chaque bit représente une page
  - Bit 0: page libre, Bit 1: page allouée
- Utilise un « indice » pour allouer
  - Scan la bitmap en partant de l'indice
  - Reprend au début de la bitmap s'il en atteint la fin
  - Place le nouvel indice au bout de la nouvelle allocation
- La bitmap est étendue s'il n'y a pas assez de place
- Exemple 1: allocation de 5 pages
- Exemple 2: nouvelle allocation de 5 pages



Page libre

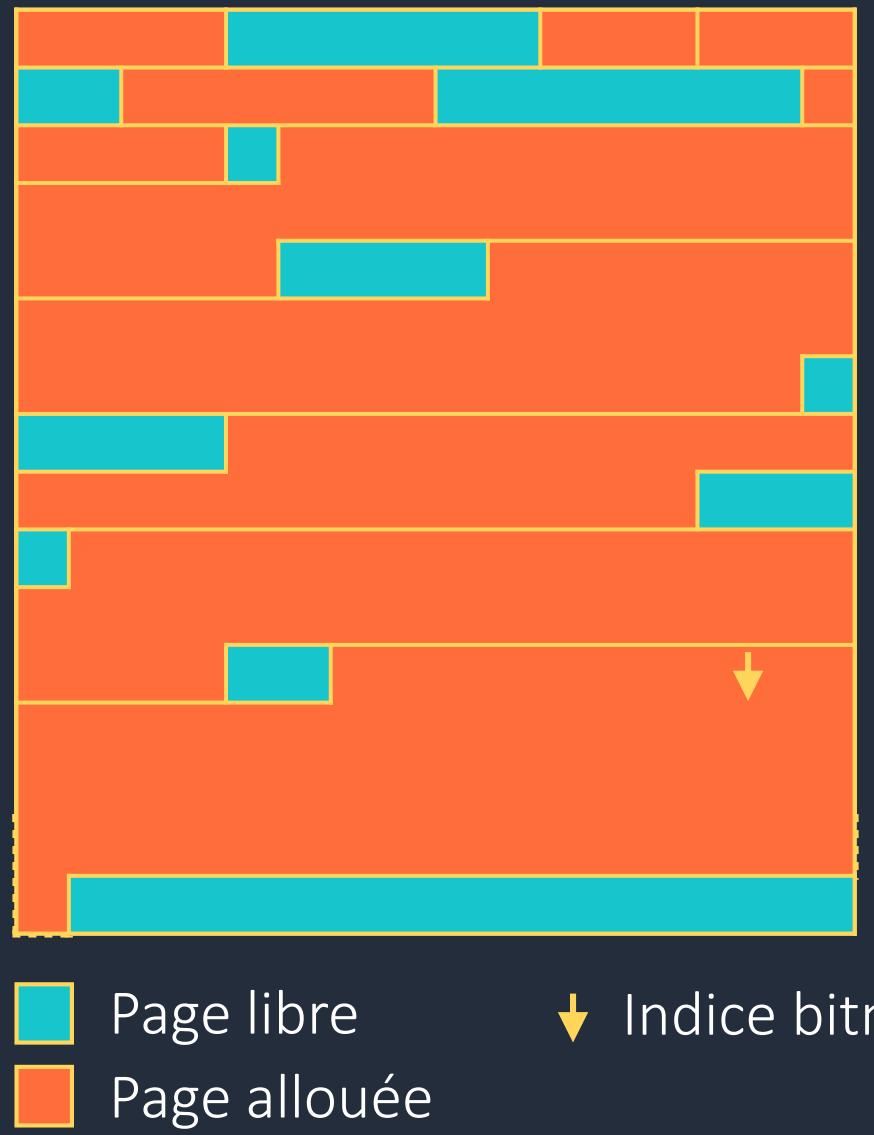
Page allouée

Indice bitmap

Bitmap d'allocation

Allocateur SystemPTE

- Utilise une bitmap
  - Chaque bit représente une page
  - Bit 0: page libre, Bit 1: page allouée
- Utilise un « indice » pour allouer
  - Scan la bitmap en partant de l'indice
  - Reprend au début de la bitmap s'il en atteint la fin
  - Place le nouvel indice au bout de la nouvelle allocation
- La bitmap est étendue s'il n'y a pas assez de place
- Exemple 1: allocation de 5 pages
- Exemple 2: nouvelle allocation de 5 pages
- Exemple 3: allocation de 17 pages

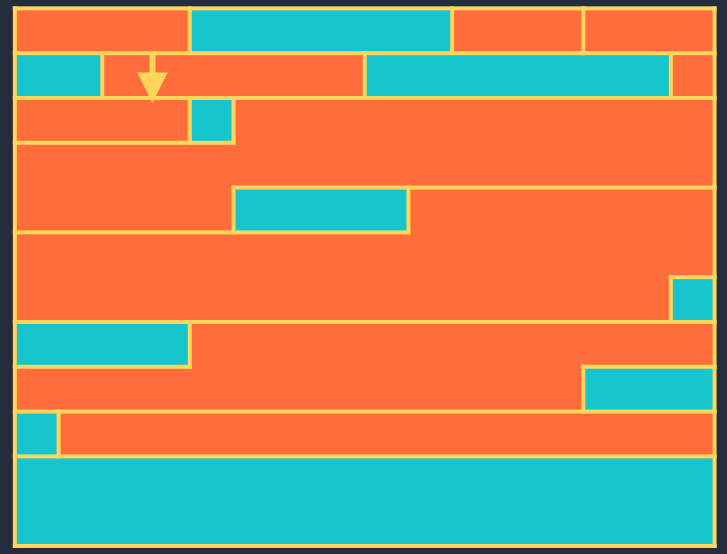


Bitmap d'allocation

SystemPTE « heap massaging »: stratégie

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution

Deux résultats possibles que l’on peut distinguer l’un de l’autre et gérer sans problème



Page libre

Indice bitmap

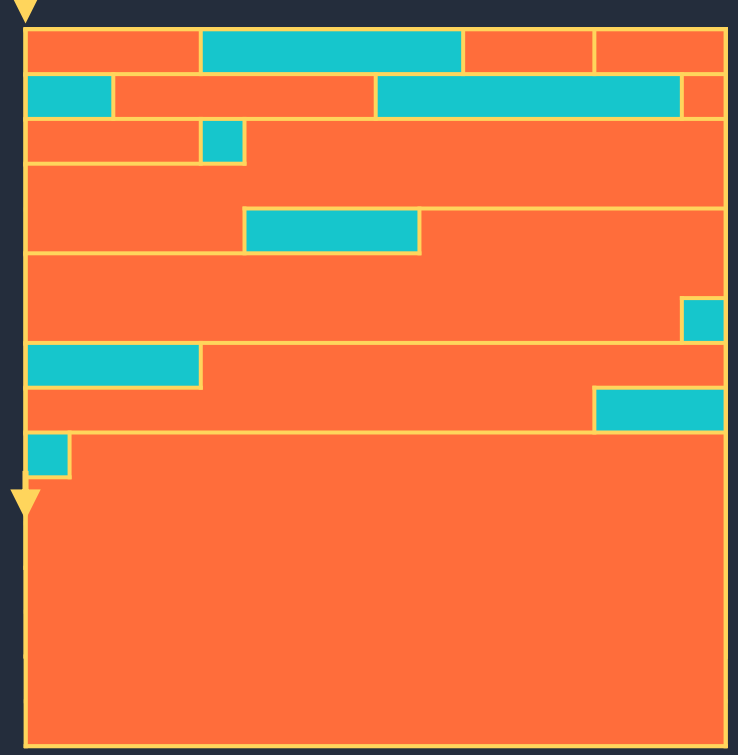
Page allouée

Bitmap d’allocation

SystemPTE « heap massaging »: stratégie

Résultat #1

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution



Page libre

Page allouée

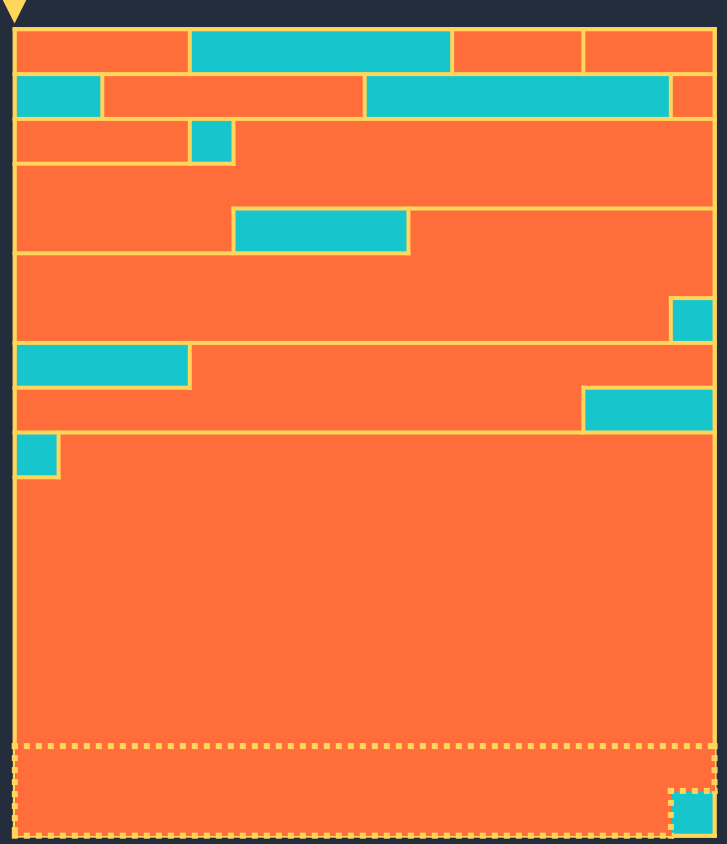
Indice bitmap

Bitmap d’allocation

SystemPTE « heap massaging »: stratégie

Résultat #1

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution



Page libre

Page allouée

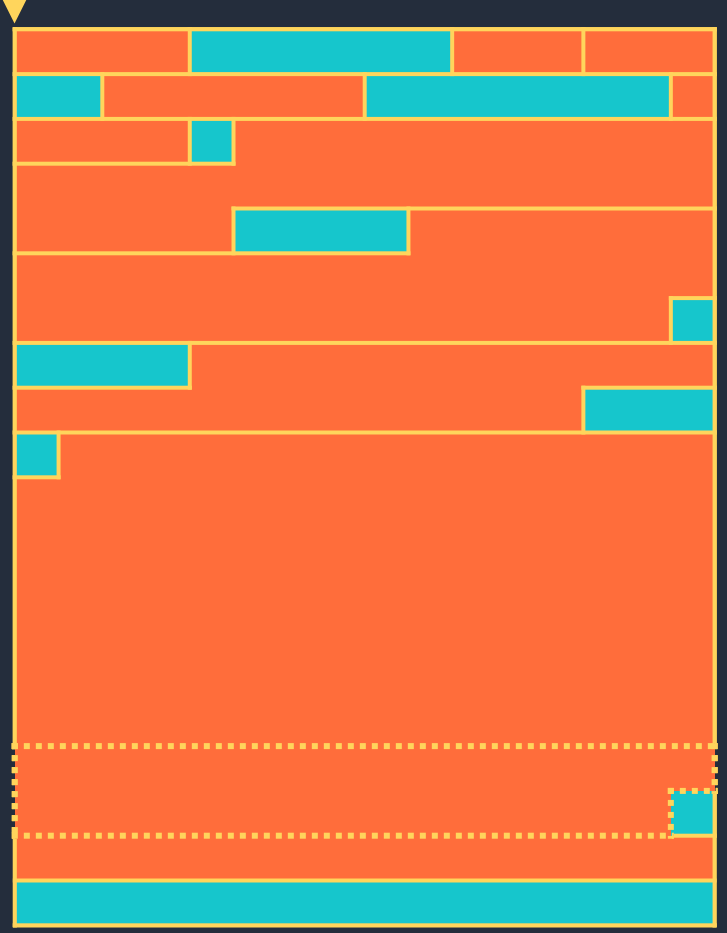
Indice bitmap

Bitmap d’allocation

SystemPTE « heap massaging »: stratégie

Résultat #1

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution

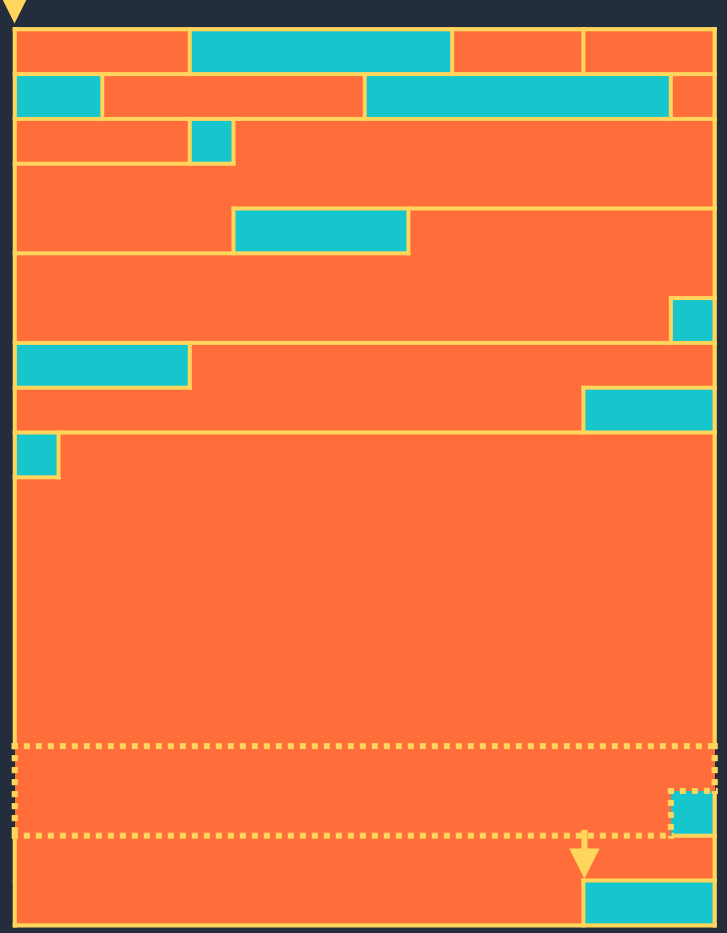


Bitmap d’allocation

SystemPTE « heap massaging »: stratégie

Résultat #1

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution



Page libre

Page allouée

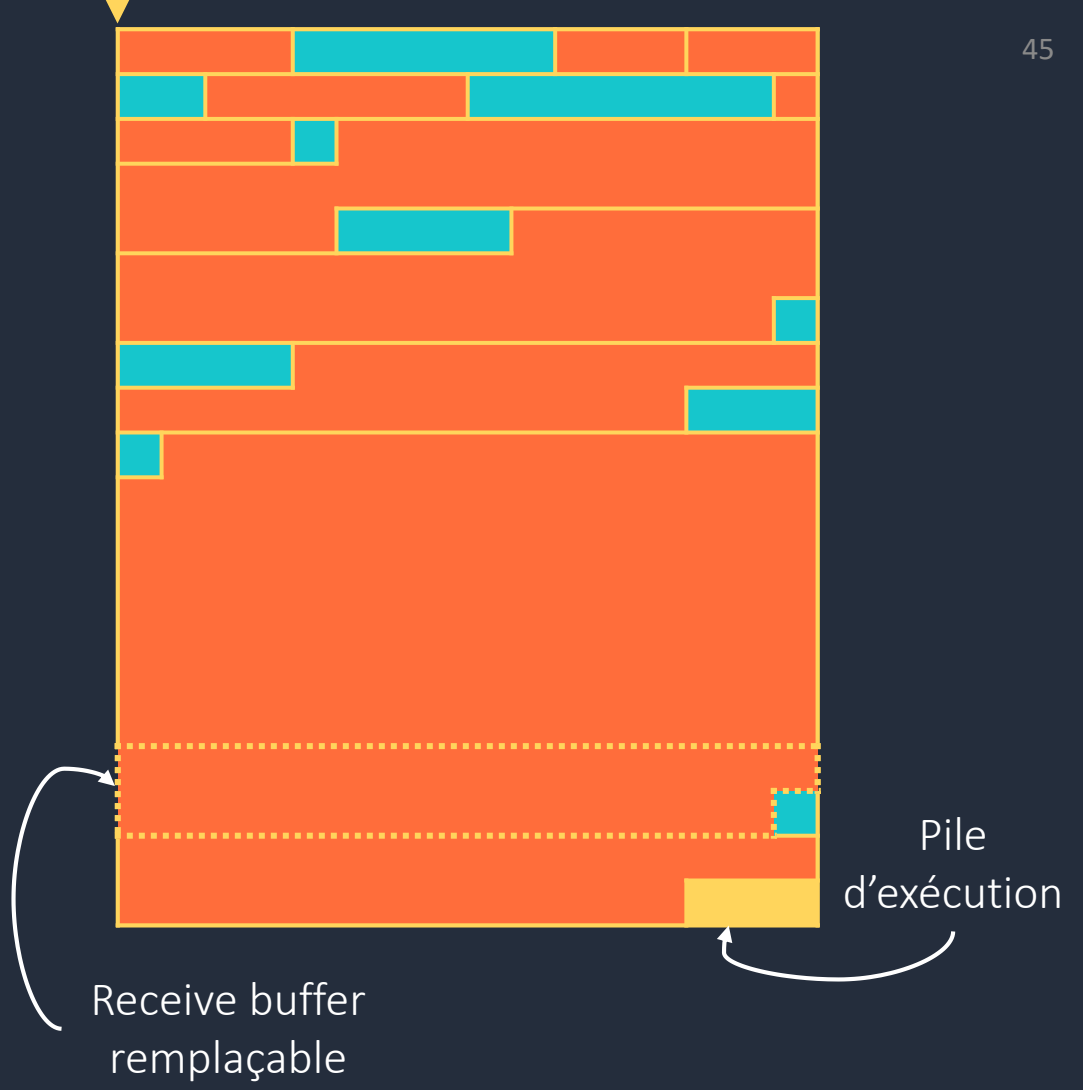
Indice bitmap

Bitmap d’allocation

SystemPTE « heap massaging »: stratégie

Résultat #1

1. On spray des buffers de 1MB
2. On alloue un buffer de 2MB - 1 page
   - (SystemPTE expansions are done in 2MB steps)
3. On alloue un buffer de 1MB
4. On alloue on buffer de 1MB - 7 pages
5. On spray des piles d’exécution



Comment exploiter le bug?

- Comment contrôler ce qui est écrit?
- Comment écrire exactement au bon moment?
- Que peut-on corrompre d’utile?
- Comment gérer KASLR?

KASLR dans le Host

Infoleak

```
struct nvsp_message {
    struct nvsp_message_header hdr;
    union nvsp_all_messages msg;
} __packed;

...

nvsp_message message;

message.hdr.msg_type = NVSP_MSG1_TYPE_SEND_RNDIS_PKT_COMPLETE;
message.v1_msg.Version1Messages.send_rndis_pkt_complete.status = NVSP_STAT_FAIL;

VmbChannelPacketComplete(completeContext, &message, sizeof(nvsp_message));
```

On peut extraire 32 octets de mémoire du Host

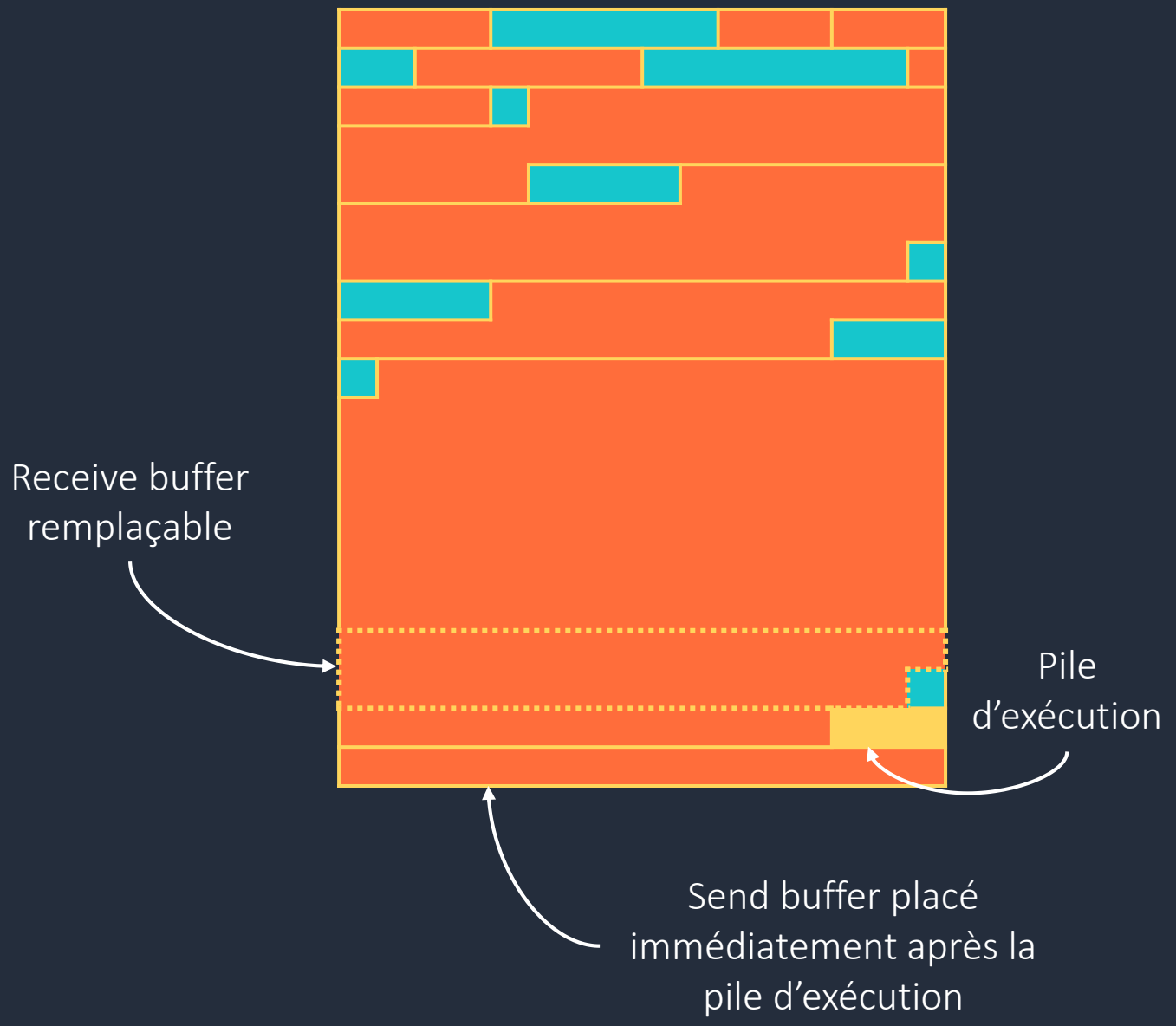
=> on peut utiliser ça pour extraire une adresse de retour dans vmswitch

On peut utiliser cette adresse de retour pour construire une ROP chain ☺

… ce bug existait sur Windows Server 2012 R2, mais pas Windows 10 ☹

Comment faire sans infoleak?

- Spectre!
  - On a prouvé qu'il était possible d'utiliser Spectre contre Hyper-V dans ce contexte…
  - ...mais cet exploit a été écrit dans l'ère pré-Spectre, donc on n'a pas pu l'utiliser
- Corruption partielle de pointeur?
  - Sur Windows, KASLR aligne les modules à 0x10000 octets…
- Exemple:
  - Adresse de retour: `0xfffff808e059f3be` (RndisDevHostDeviceCompleteSetEx+0x10a)
  - Corruption partielle: `0xfffff808e0598705` (ROP gadget: `pop r15; ret;`)
- En pratique, notre vulnérabilité écrit un unique bloc continu
  - => on ne peut faire qu'une seule corruption partielle… est-ce utile?



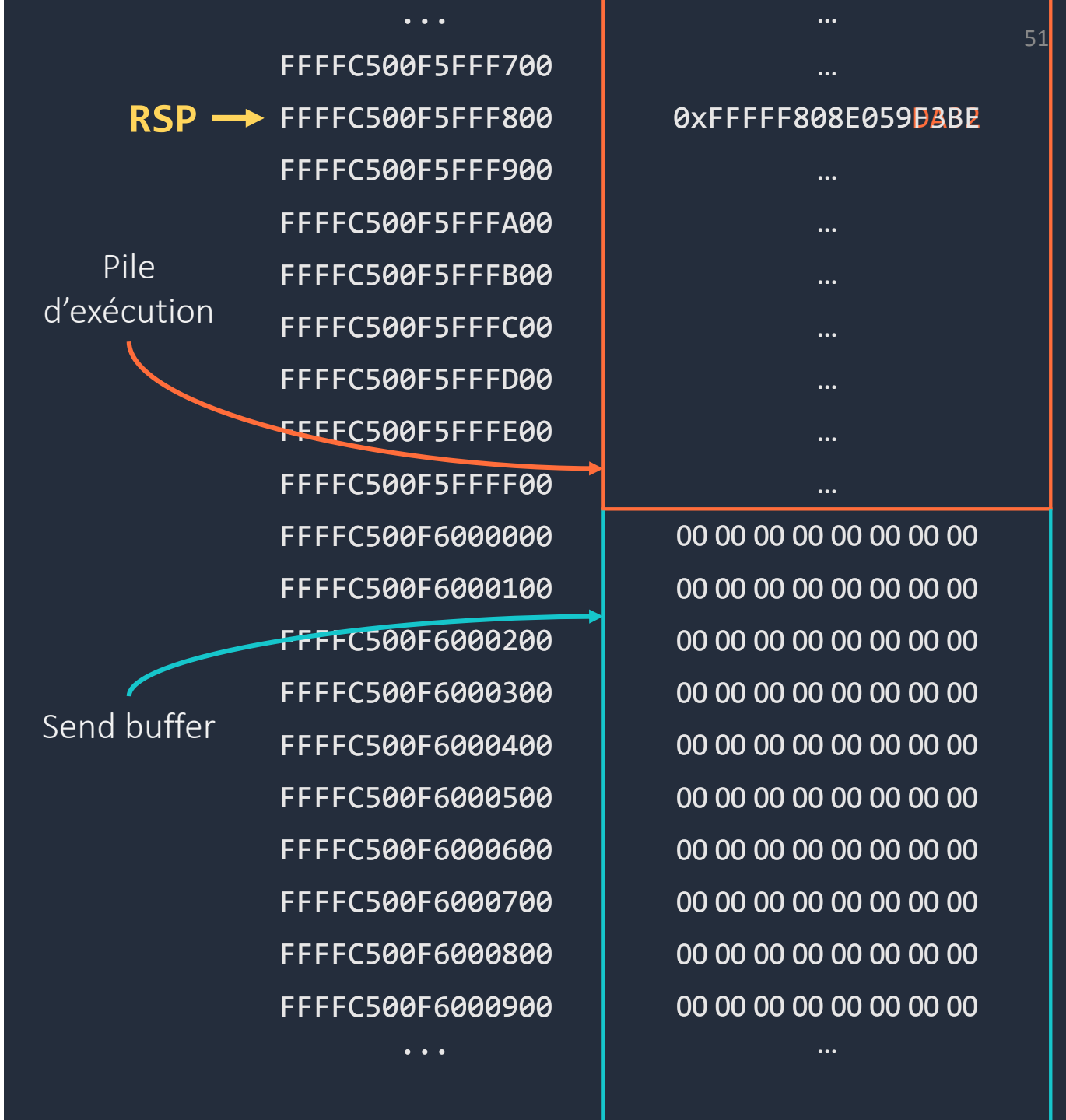
SystemPTE heap massaging

Corruption partielle

- Et si on l'utilise pour placer RSP dans notre send buffer?
  - Adresse de retour: `0xFFFFF808E059F3BE`
  - Corruption partielle: `0xFFFFF808E059DA32`

```
lea r11, [rsp+0E50h]
mov rbx, [r11+38h]
mov rbp, [r11+40h]
mov rsp, r11
...
retn
```

  - Ça revient à: `RSP += 0xE78`
- Ça nous donne que RSP est dans le send buffer… qui est un mapping de mémoire partagée avec le Guest

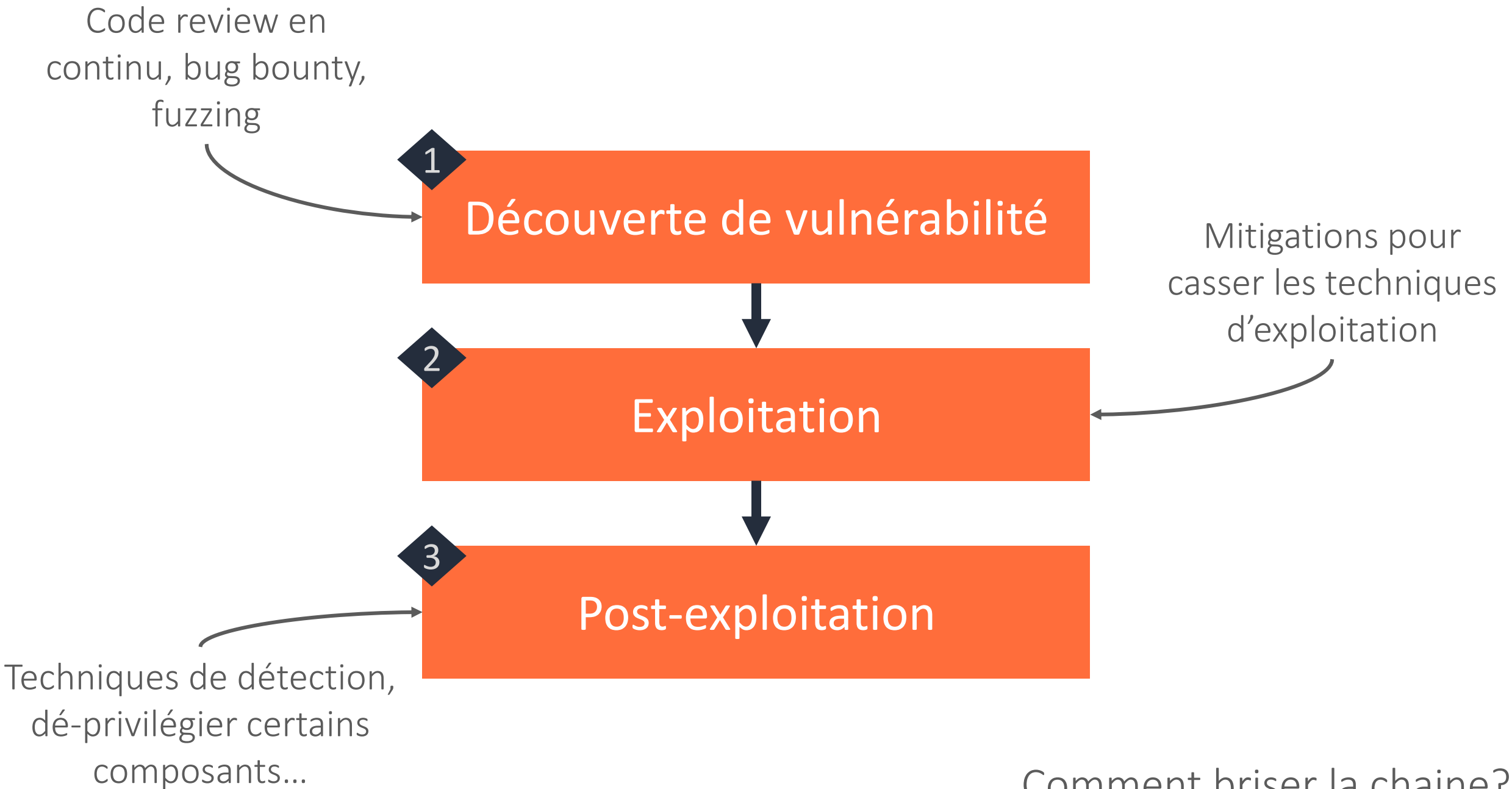


RSP pointe vers de la mémoire partagée… et maintenant?

1. Le Host crash: le CPU encourt une General Protection Fault (GPF)
   - Vu qu'on n'a pas d'infoleak, l'instruction RET crashe forcément…
2. L'adresse où la GPF est arrivée est placée sur la pile
   - Vu que la pile est en mémoire partagée, le Guest peut la lire
3. Le Windows du Host gère la GPF, utilisant toujours la pile en mémoire partagée
4. Pendant ce temps, le Guest peut:
   1. Déterminer l'adresse de ROP gadgets grace à l'adresse placée sur la pile en 2)
   2. Manipuler la pile pendant que le Host gère la GPF
      - Par exemple, il peut corrompre une adresse de retour
5. Résultat: on peut exécuter une ROP chain dans le Host sans bug infoleak ☺

Demo time 

Hardening Hyper-V



Comment briser la chaine?

Hardening: isolation des piles d'exécution

Pour éviter qu'une vulnérabilité puisse immédiatement corrompre une pile, celles-ci ont été placées dans leur propre région isolée de mémoire kernel

```
0: kd> !address
...
ffffae8f`050a8000 ffffae8f`050a9000  0`00001000  SystemRange
ffffae8f`050a9000 ffffae8f`050b0000  0`00007000  SystemRange  Stack  Thread: ffffbc8934d51700
ffffae8f`050b0000 ffffae8f`050b1000  0`00001000  SystemRange
ffffae8f`050b1000 ffffae8f`050b8000  0`00007000  SystemRange  Stack  Thread: ffffbc8934d55700
ffffae8f`050b8000 ffffae8f`050b9000  0`00001000  SystemRange
ffffae8f`050b9000 ffffae8f`050c0000  0`00007000  SystemRange  Stack  Thread: ffffbc8934d59700
ffffae8f`050c0000 ffffae8f`050c1000  0`00001000  SystemRange
ffffae8f`050c1000 ffffae8f`050c8000  0`00007000  SystemRange  Stack  Thread: ffffbc8934d5d700
...
```

Parait très spécifique, mais:

- A empêché plusieurs vulnérabilités trouvées depuis de directement corrompre des piles
- Pas d'impact perf mesurable

Hardening: InitAll

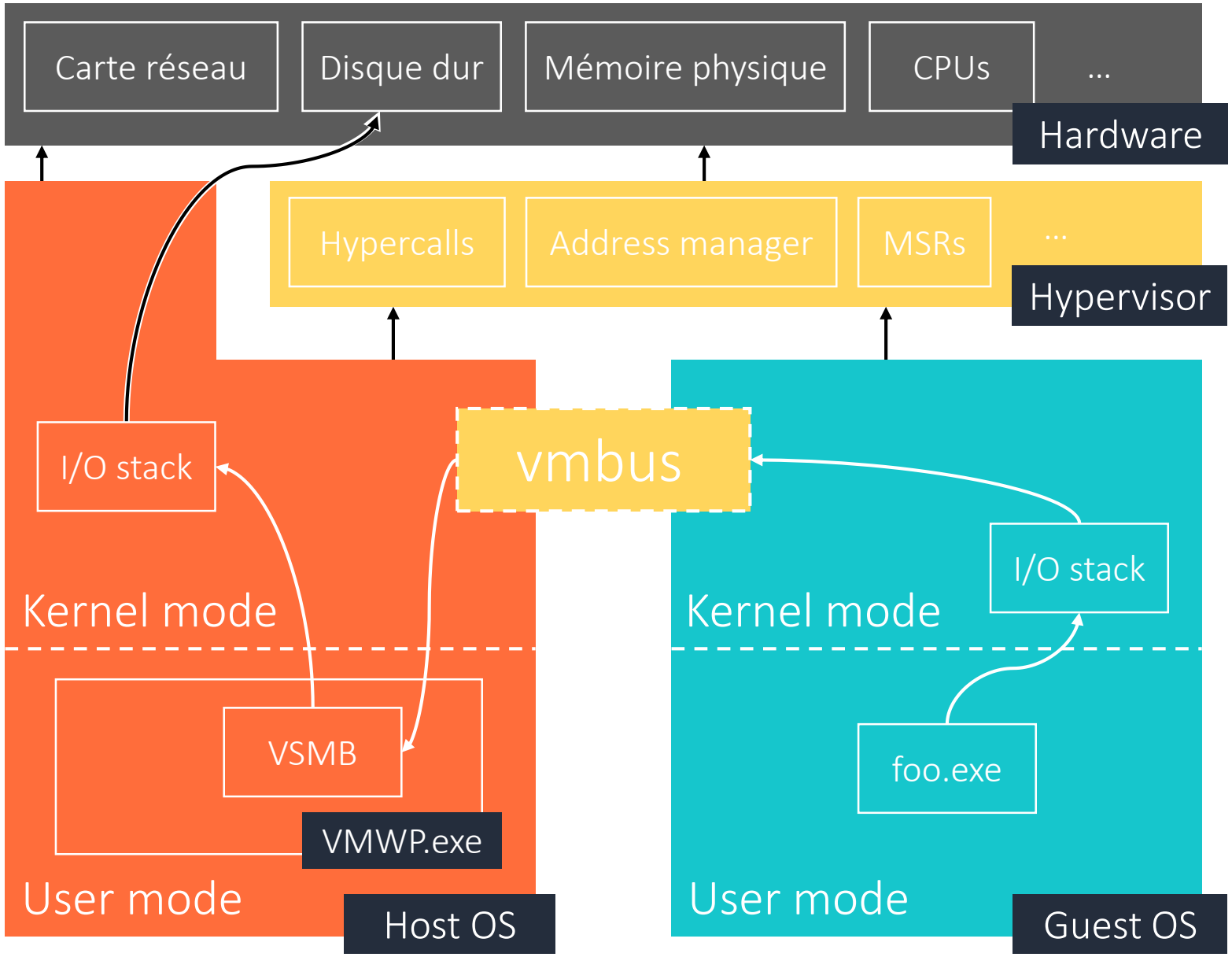
Le compilateur initialise certaines variables locales (POD) automatiquement

```
nvsp_message message;

message.hdr.msg_type = NVSP_MSG1_TYPE_SEND_RNDIS_PKT_COMPLETE;
message.v1_msg.Version1Messages.send_rndis_pkt_complete.status = NVSP_STAT_FAIL;

VmbChannelPacketComplete(completeContext, &message, sizeof(nvsp_message));
```

InitAll aurait initialisé « `message` » - plus d’infoleak dans cette fonction

Hardening: autres mitigations

- Hypervisor-enforced Code Integrity (HVCI)
  - Attaquants ne peuvent plus injecter de shellcode dans le Host
- Kernel-mode Control Flow Guard (KCFG)
  - Attaquants ne peuvent plus faire exécuter n'importe quel code en corrompant un pointeur de fonction
- Nous travaillons pour activer ces features par défaut
- Features en cours de développement
  - CET (hardware Intel)
    - Shadow stack assistée par le hardware pour complètement bloquer la ROP
  - XFG (software Windows)
    - Évolution de CFG pour limiter quelles fonctions sont accessibles



Hyper-V architecture: le role de VMWP pour la sécurité

Hardening: VM Worker Process

- Amélioration de sa sandbox
  - Déjà éliminé SeImpersonatePrivilege, d'autres améliorations en cours
- Autant de mitigations que possible
  - CFG export suppression
    - Réduction du nombre de fonctions accessibles via CFG
  - "Force CFG"
    - Seuls les modules compilés pour supporter CFG peuvent être chargés dans VMWP
  - ACG, CIG
    - Seul du code signé peut être chargé dans VMWP
  - À l'avenir: XFG
- Beaucoup d'efforts en cours pour sortir des composants Hyper-V du kernel et les placer dans VMWP

La bug bounty Hyper-V

- Paye jusqu'à $250,000!
  - Toutes vulnerabilités bienvenues: code execution, infoleak, denial of service
  - https://technet.microsoft.com/en-us/mt784431.aspx
- Par où commencer///?
  - *Joe Bialek* and *Nicolas Joly*'s talk: "A Dive in to Hyper-V Architecture & Vulnerabilities"
  - Hyper-V Linux integration services
    - Code open source et bien commenté disponible sur Github
    - Bonne façon de comprendre comment fonctionnent les différents VSP et de jouer avec
  - De plus en plus de symboles sont rendus publiques pour Hyper-V

Merci pour votre attention!

Merci en particulier à **Matt Miller**, **David Weston**, l'équipe **Hyper-V**, l'équipe **vmswitch**, l'équipe **MSRC**, le reste de l'équipe **OSR** et **tout le CO du SSTIC** pour une super conférence