

SDN-NFV 3

ESIR

Djob Mvondo

NFV: Network Function Virtualization

Les couches réseau et stockage

Optimiser le démarrage
des VNFs

Mise à jour sans
interruption de
service

La partie logique virtualisé – VNF – Virtual Network Function

Optimisation la couche réseau et stockage –
passage à l'échelle

Virtual Compute

Virtual Storage

Virtual Network

Virtualization Layer

Compute

Storage

Network

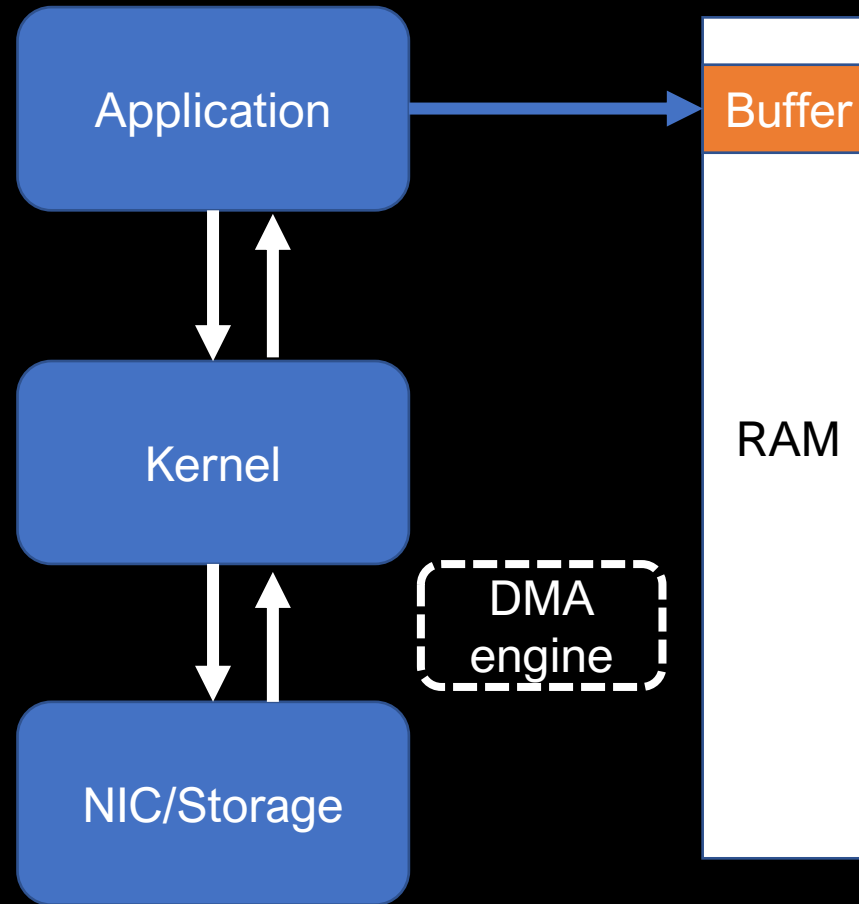
La partie matériel – NFVI – NFV Infrastructure

Surveillance
interopérable et
performante

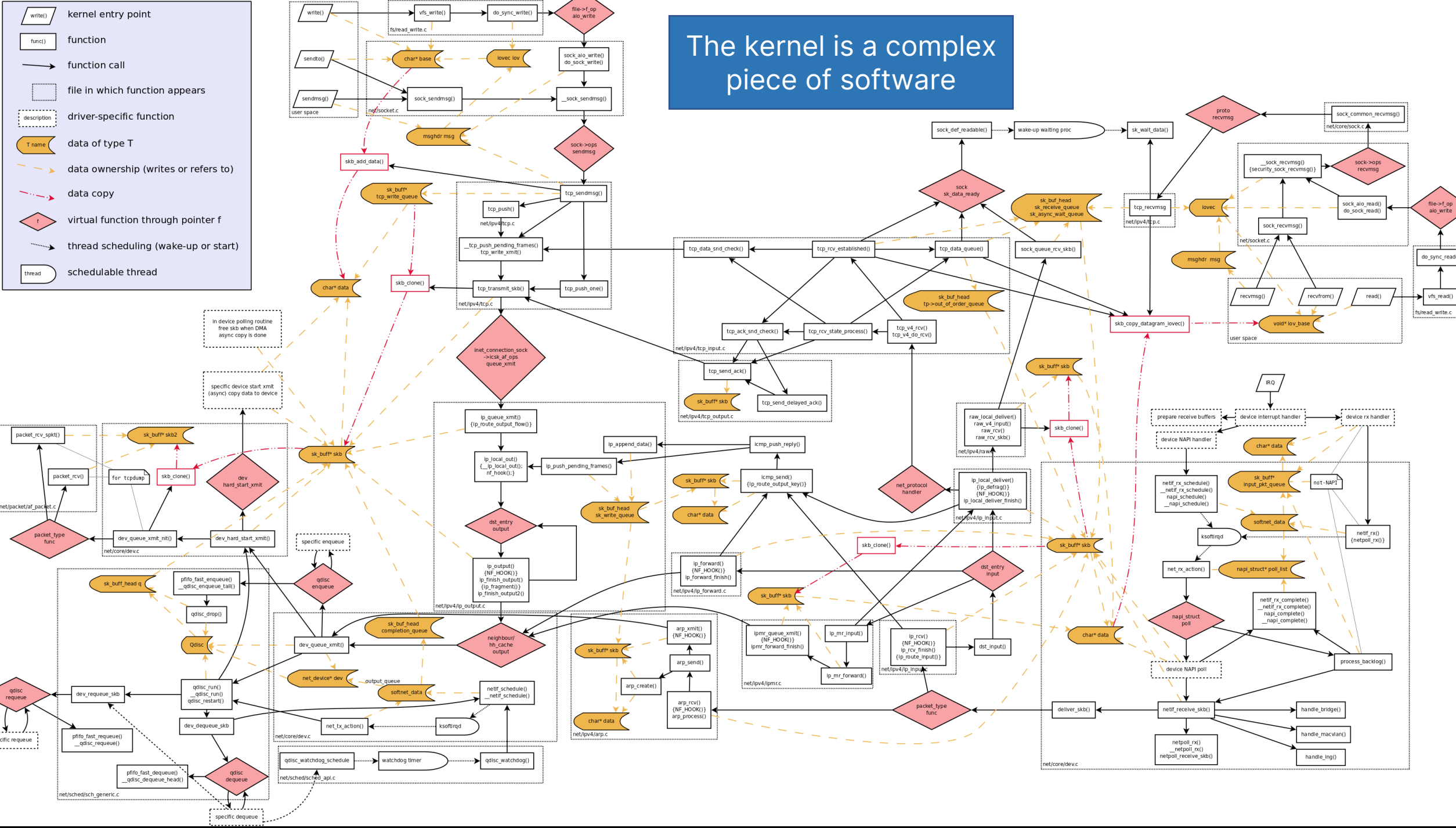
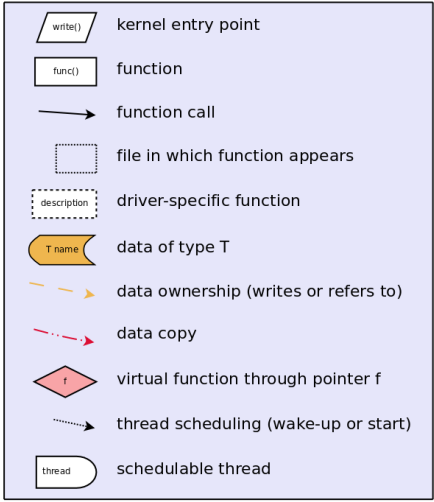
NFV MONA –
Management
and
Orchestration

NFV: Network Function Virtualization

Traditionnellement, comment fonctionne la couche réseau d'un OS ?

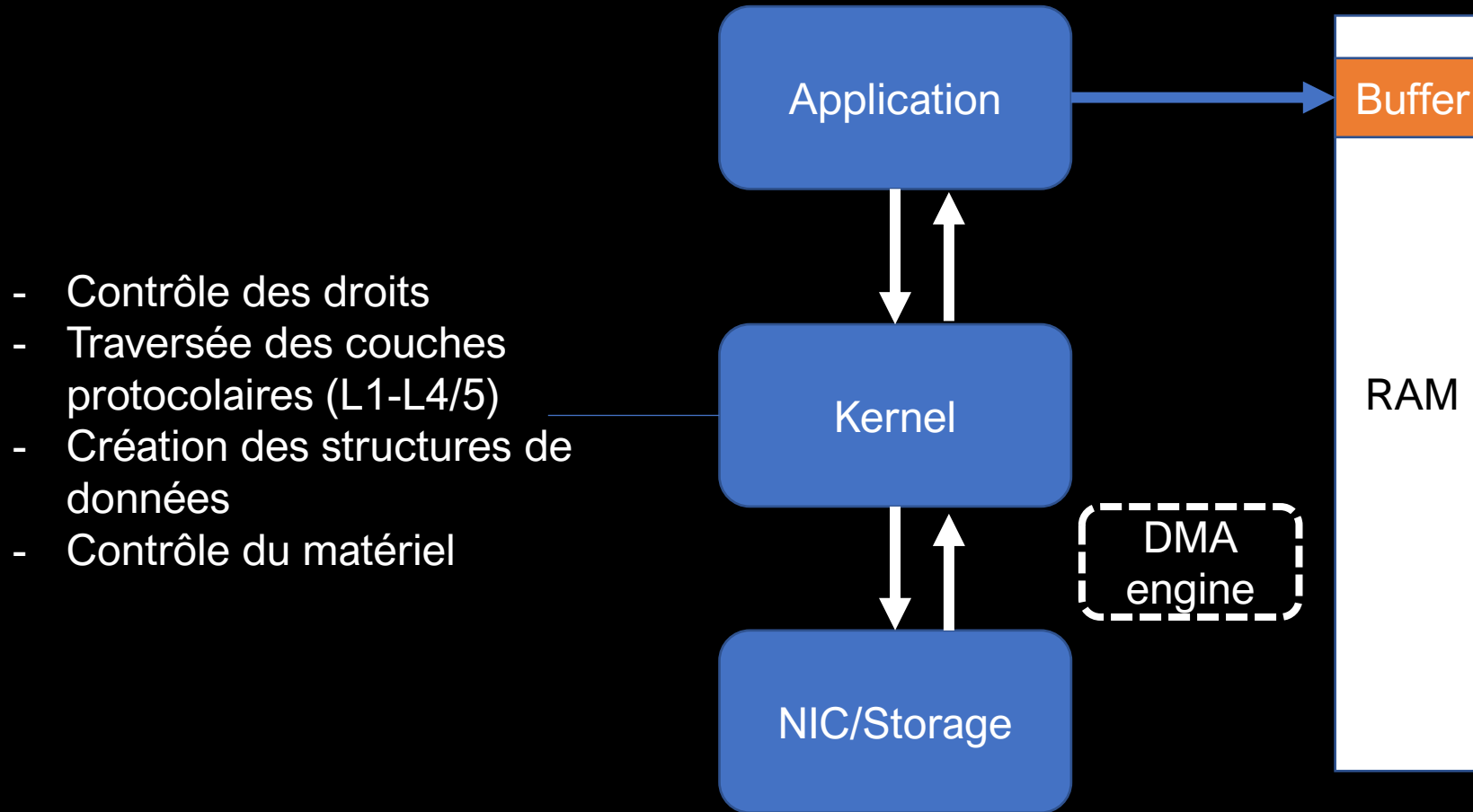


The kernel is a complex piece of software



NFV: Network Function Virtualization

Traditionnellement, comment fonctionne la couche réseau d'un OS ?



- Contrôle des droits
- Traversée des couches protocolaires (L1-L4/5)
- Création des structures de données
- Contrôle du matériel

- Le noyau prend un temps considérable lors du traitement des paquets/blocks de données
- Le noyau reste relativement rigide lors qu'il faut des mise à jour à la volée

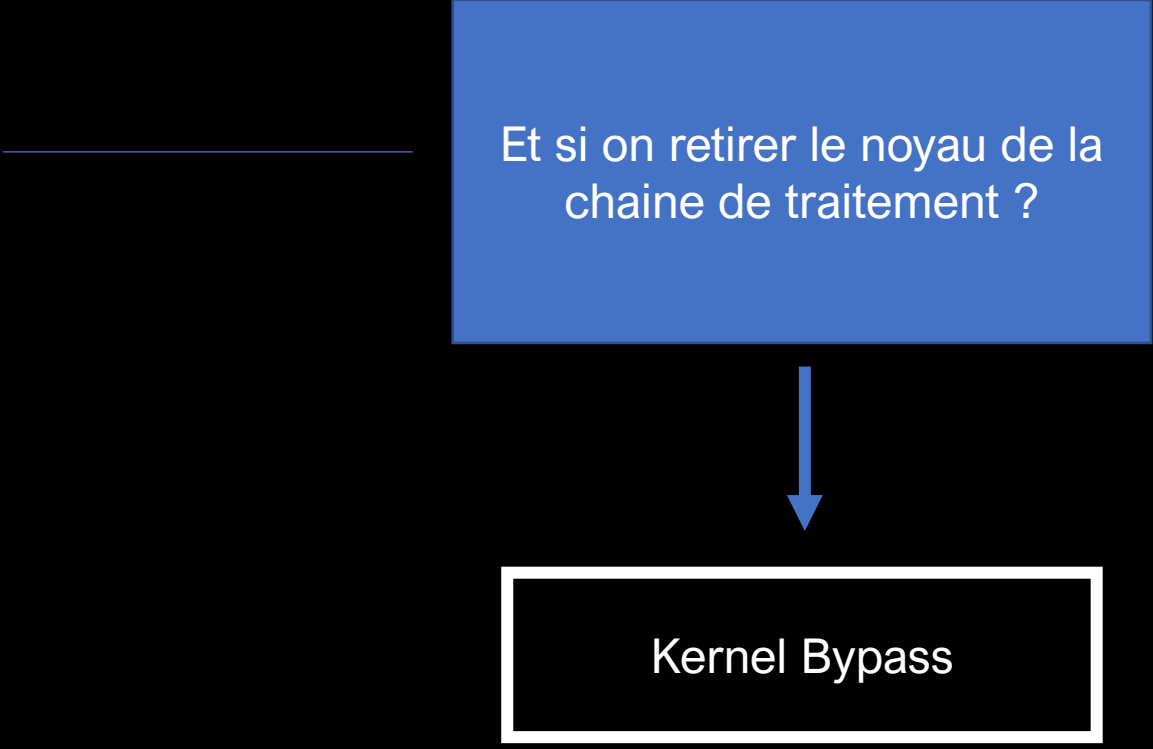
Alors comment faire ?

NFV: Network Function Virtualization

Traditionnellement, comment fonctionne la couche réseau d'un OS ?

- Le noyau prend un temps considérable lors du traitement des paquets/blocks de données
- Le noyau reste relativement rigide lors qu'il faut des mise à jour à la volée

Alors comment faire ?

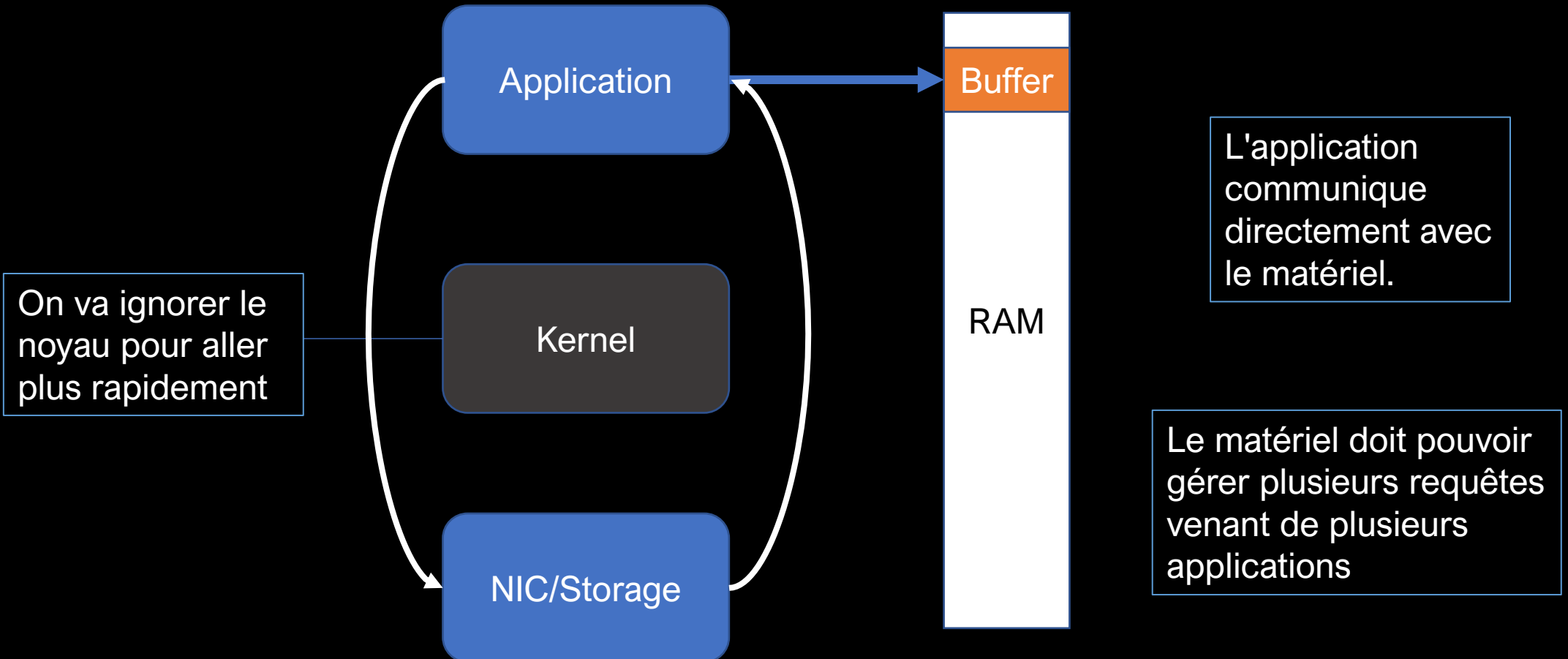


Et si on retire le noyau de la chaine de traitement ?

Kernel Bypass

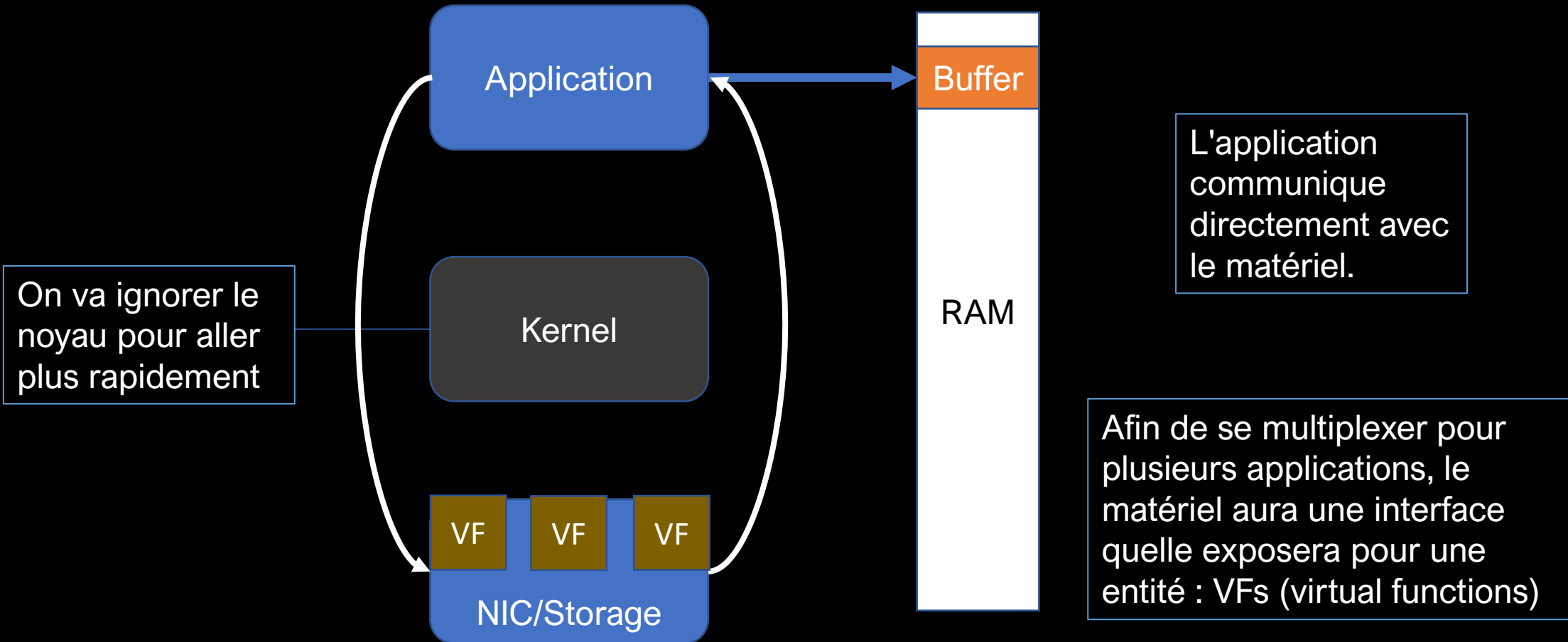
NFV: Network Function Virtualization

Kernel Bypass - Principles



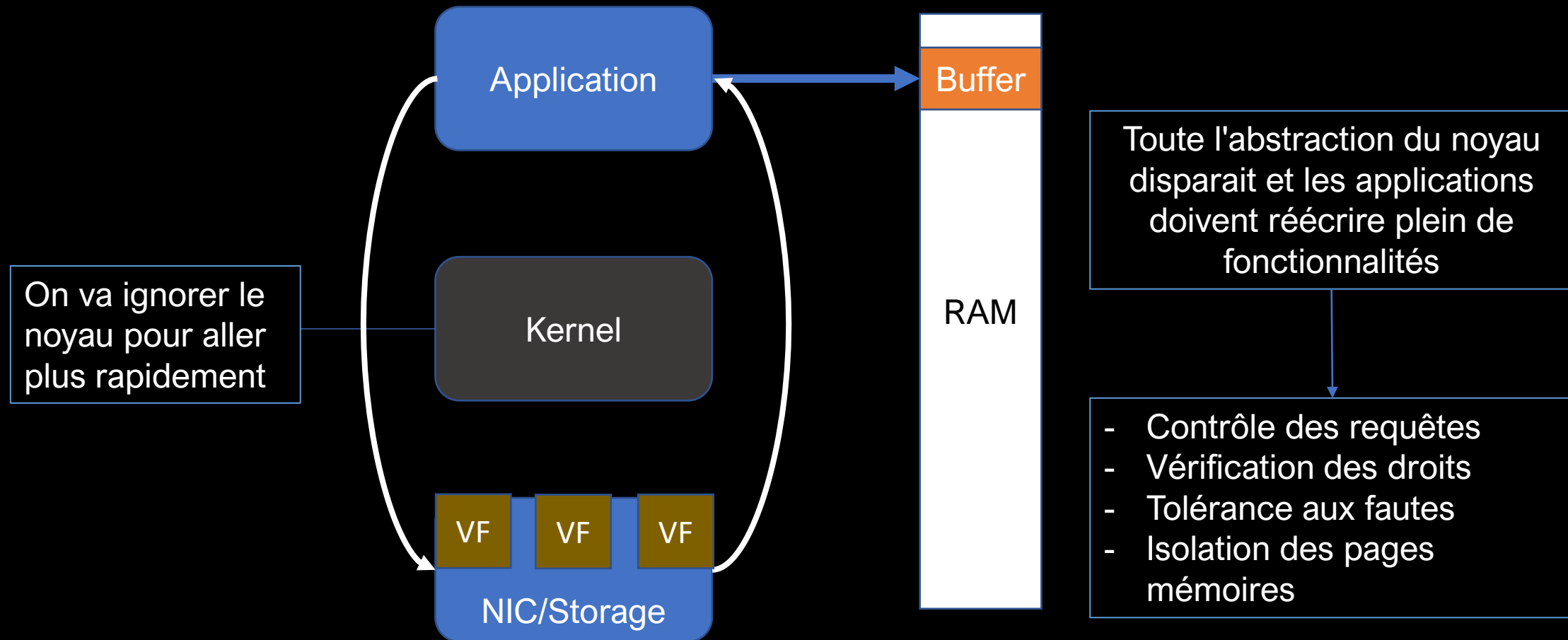
NFV: Network Function Virtualization

Kernel Bypass - Principles



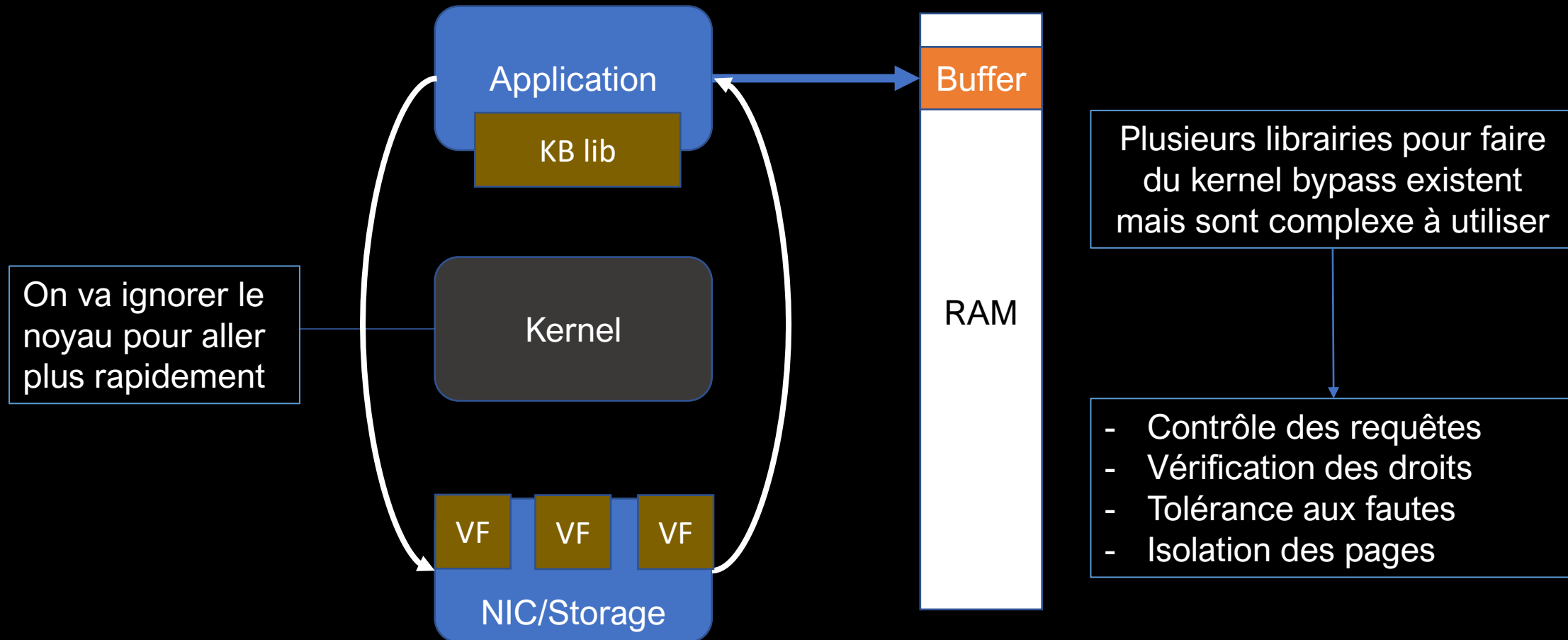
NFV: Network Function Virtualization

Kernel Bypass – Le noyau avait quand même un rôle important



NFV: Network Function Virtualization

Kernel Bypass – Le noyau avait quand même un rôle important



NFV: Network Function Virtualization

Traditionnellement, comment fonctionne la couche réseau d'un OS ?

- Le noyau prend un temps considérable lors du traitement des paquets/blocks de données
- Le noyau reste relativement rigide lors qu'il faut des mise à jour à la volée

Alors comment faire ?

Et si on changeait dynamiquement le comportement du noyau ?

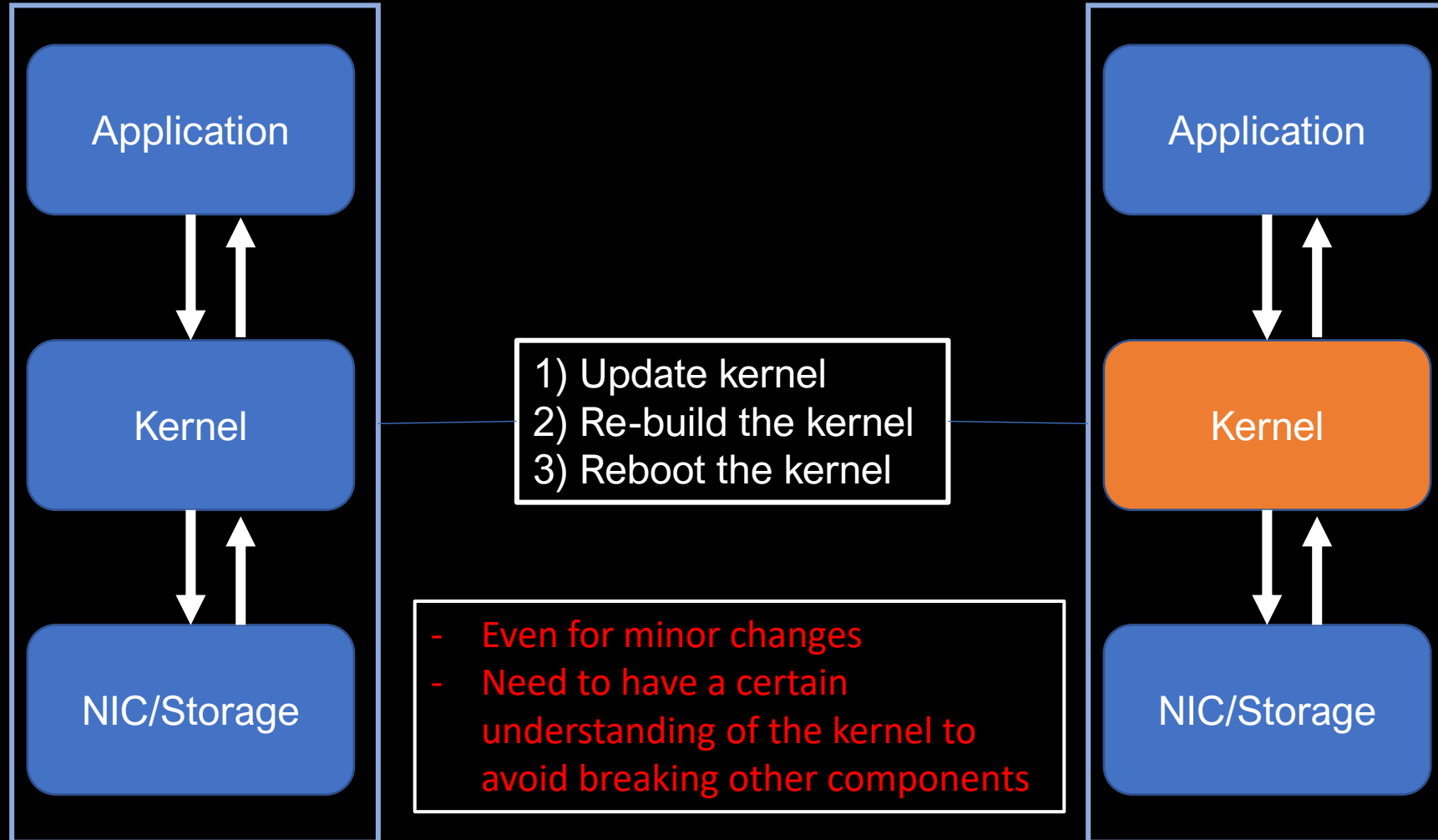
Modular kernel

Et si on retire le noyau de la chaîne de traitement ?

Kernel Bypass

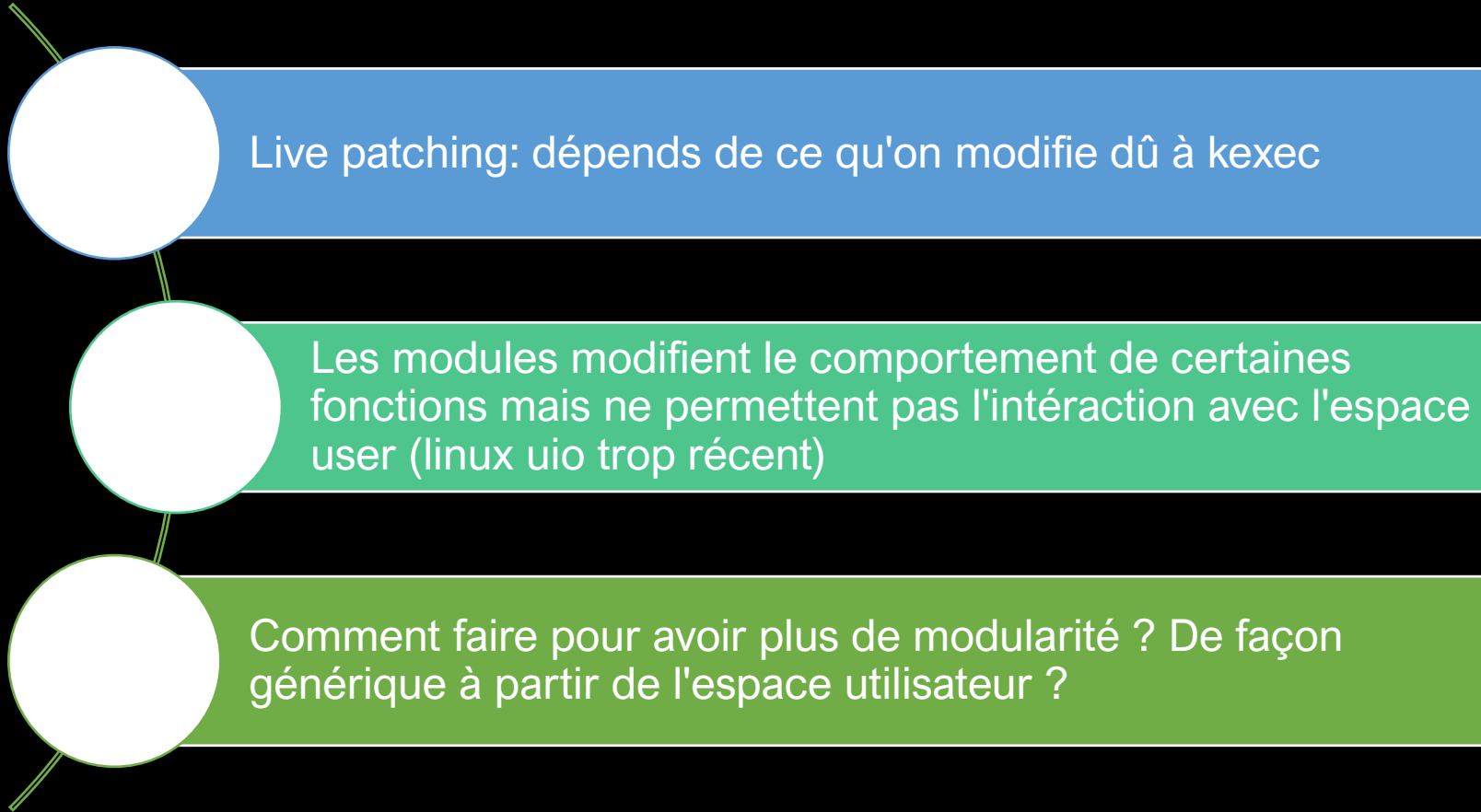
NFV: Network Function Virtualization

Comment faire si je peux optimiser ou personnaliser une partie du noyau ?



NFV: Network Function Virtualization

Quelques mécanismes existent: Live patching, modules



NFV: Network Function Virtualization

Quelques mécanismes existent: Live patching, modules

Live patching: dépend de ce qu'on modifie dû à kexec

Les modules modifient le comportement de certaines fonctions mais ne permettent pas l'interaction avec l'espace user (linux uio trop récent)

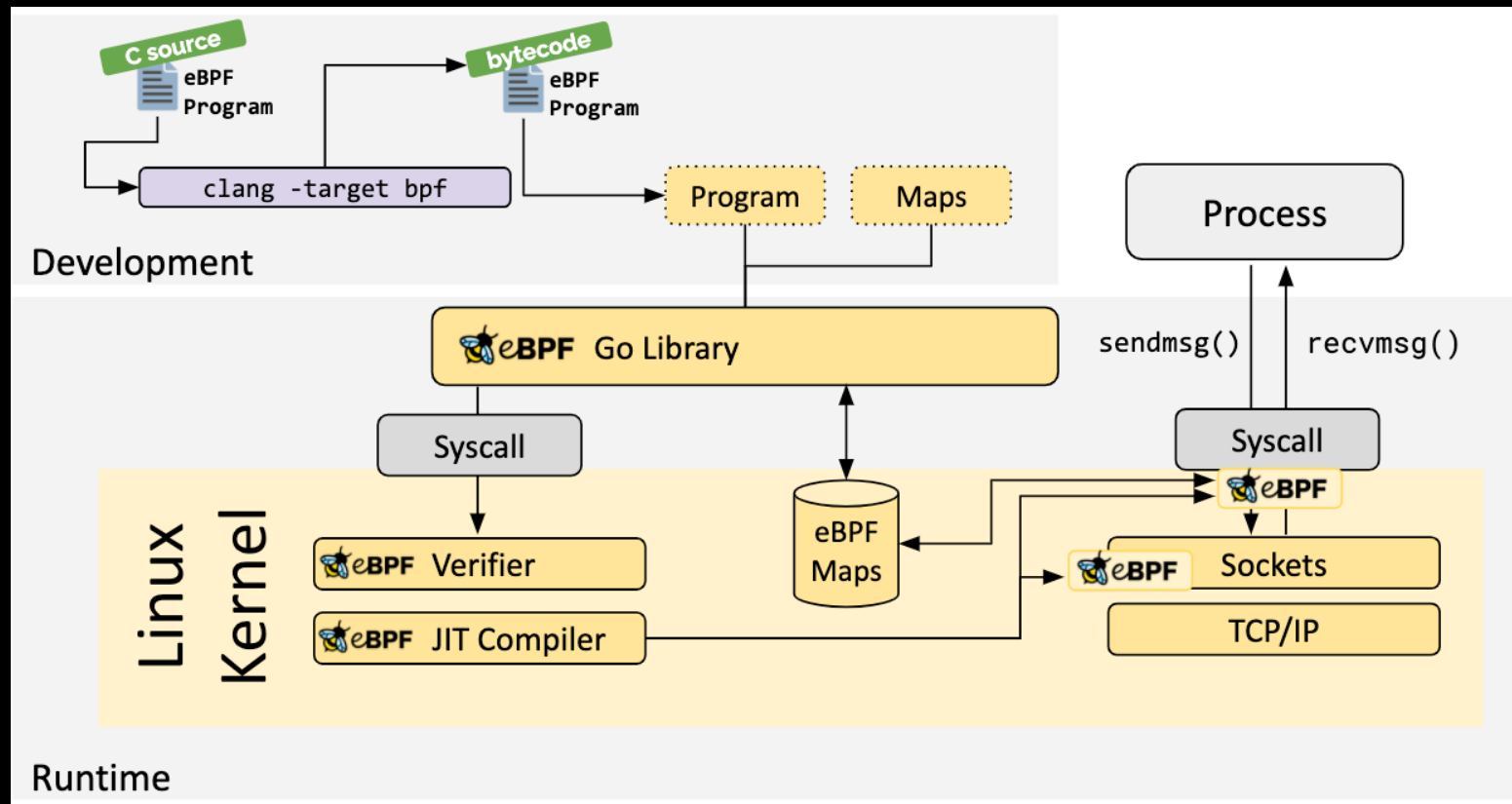
Comment faire pour avoir plus de modularité ? De façon générique à partir de l'espace utilisateur ?

eBPF (extended Berkeley Packet Filter)



NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)



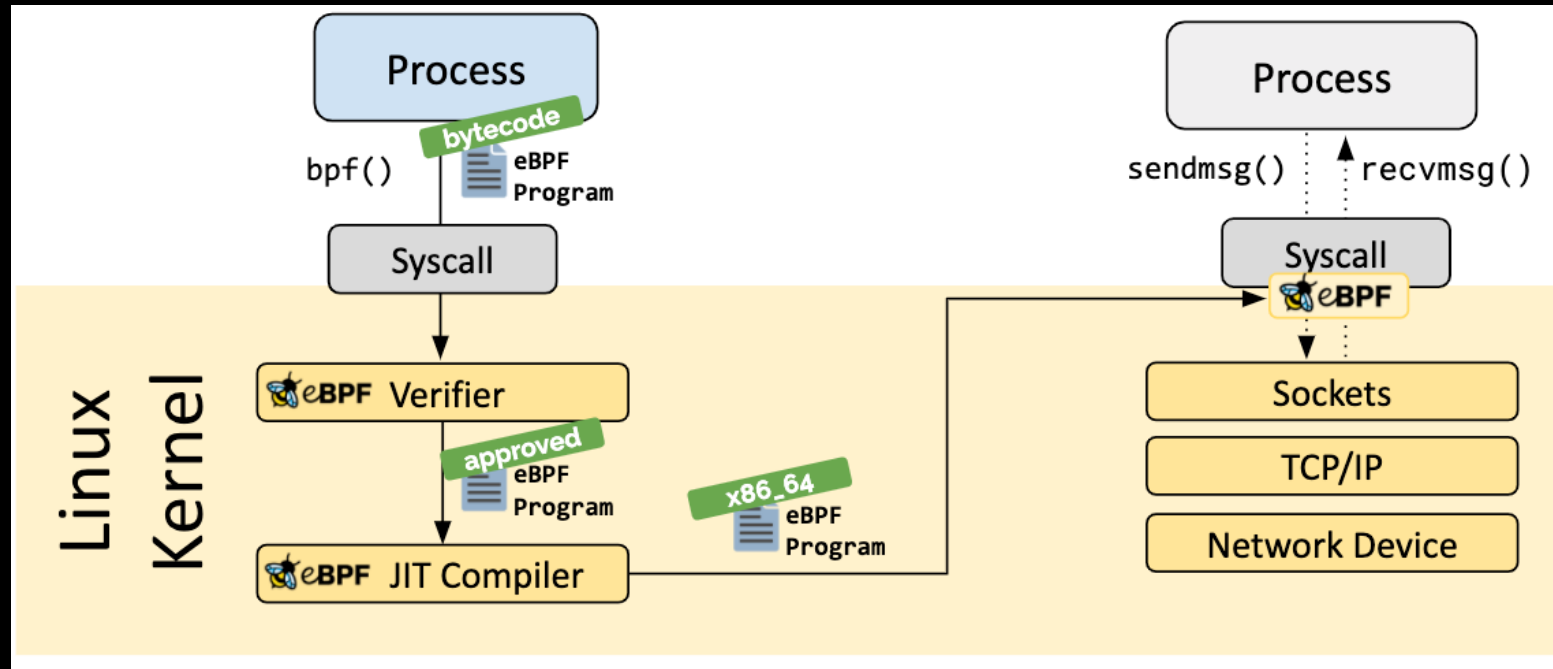
Une machine VM qui s'exécute lorsque votre noyau s'exécute en interprétant du bytecode.

En fonction des hooks que vous définissez, le code de votre programme eBPF est introduit à l'exécution via un **compilateur JIT (Just In Time)**

Votre programme s'exécute en mode noyau mais peut obtenir des informations du mode utilisateur grâce aux **Maps**

NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)

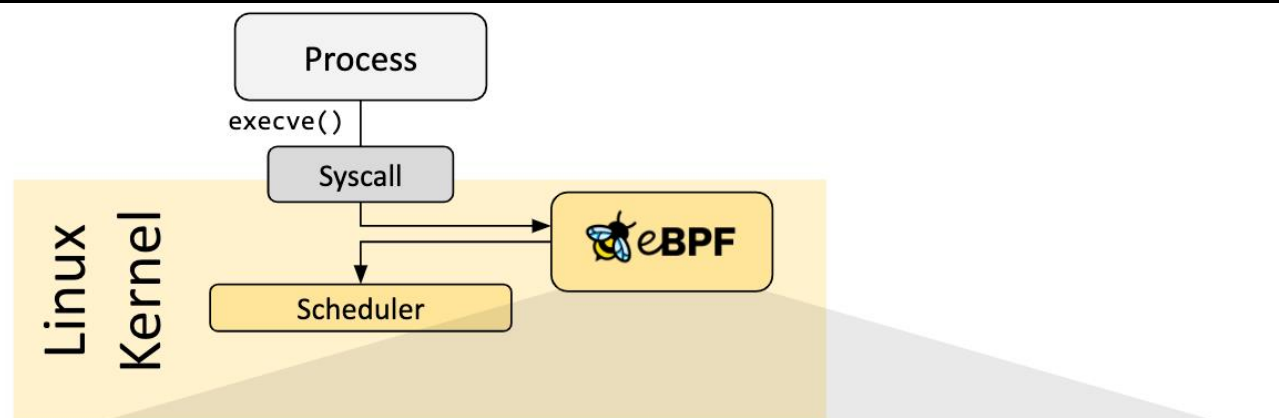


Chaque code eBPF est vérifié pour s'assurer de l'absence de bugs et de sa terminaison

Les structures de données sont optimisées avec les structure de contrôle comme les boucles (qui sont déroulés)

NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)



```
int syscall__ret_execve(struct pt_regs *ctx)
{
    struct comm_event event = {
        .pid = bpf_get_current_pid_tgid() >> 32,
        .type = TYPE_RETURN,
    };

    bpf_get_current_comm(&event.comm, sizeof(event.comm));
    comm_events.perf_submit(ctx, &event, sizeof(event));

    return 0;
}
```

Des **helpers** sont à disposition de l'utilisateur pour réaliser certaines opérations

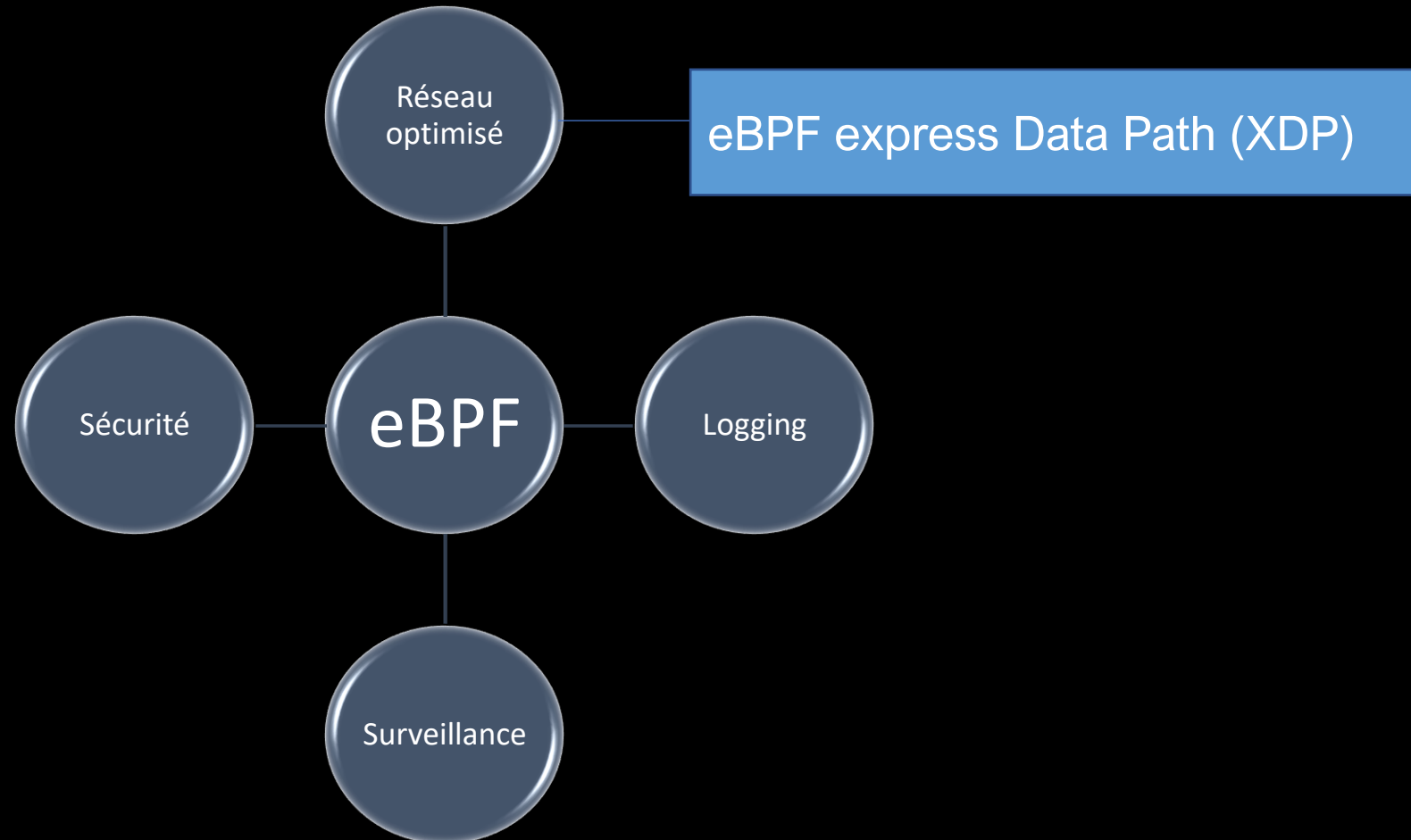
Une compréhension minimaliste du sous-système noyau cible est requise pour déterminer les **bons hooks**

man bpf-helpers

<https://ebpf.io/>

NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)



NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)



```
struct bpf_map_def sec("maps") processes = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(struct key_process_t),
    .value_size = sizeof(u64),
    .max_entries = PID_MAX_LIMIT,
};
```

```
SEC("prog")
__u32 int main_func(struct rq *ctx)
{
    struct rt_rq *rt_rq = &ctx->rt;
    struct rt_prio_array *array = &rt_rq->active;
    struct sched_rt_entity *next = NULL;
    struct list_head *queue;
    int idx;

    return sched_find_first_bit(array->bitmap);
}
```

Map definition and code example exploiting ebpf.

NFV: Network Function Virtualization

eBPF (extended Berkeley Packet Filter)



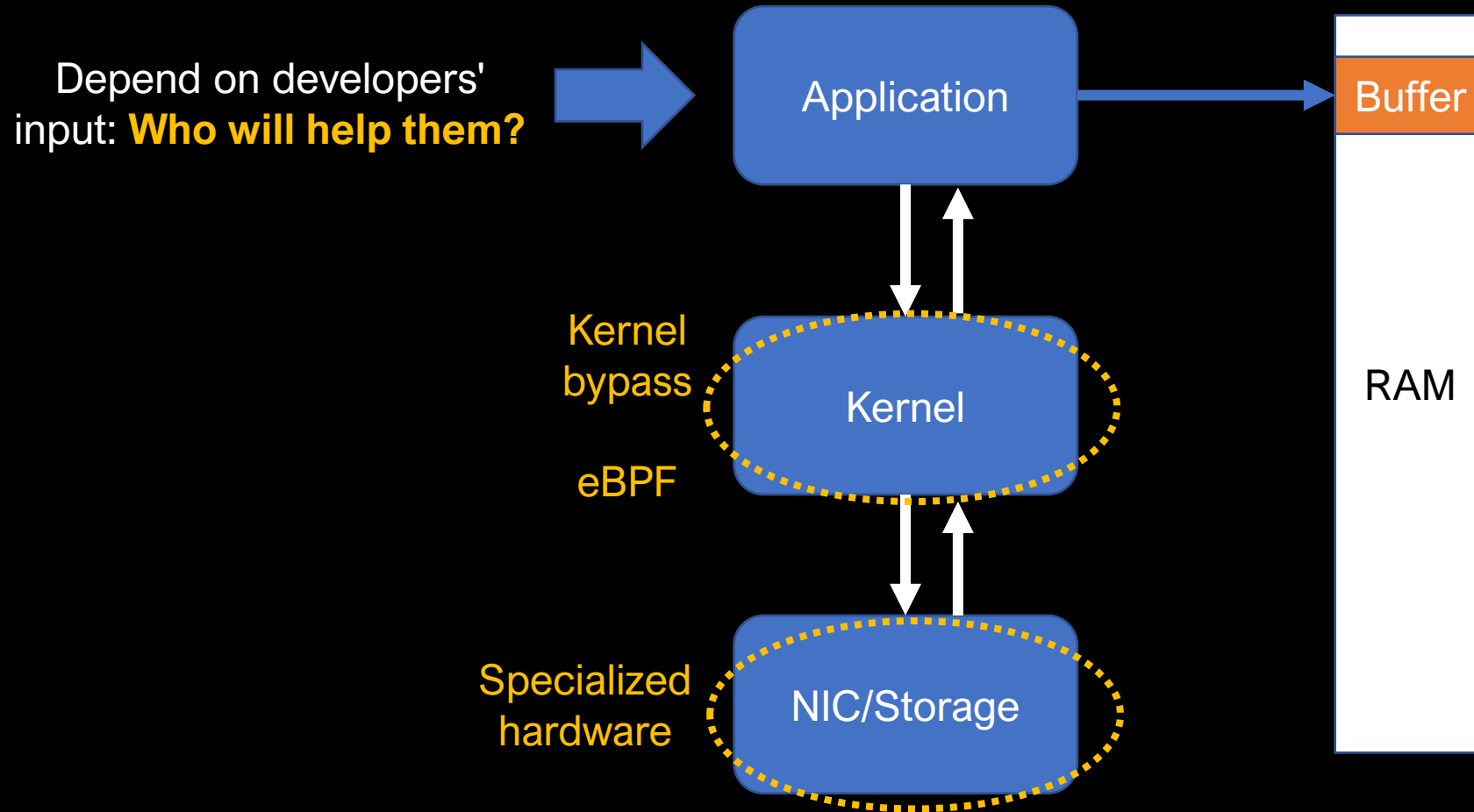
```
ret = bpf_prog_load("bpf_prog_fifo.o", BPF_PROG_TYPE_SCHED, &obj,
&progfd);
if (ret) {
    printf("Failed to load bpf program\n");
    exit(1);
}

ret = bpf_prog_attach(progfd, 0, BPF_SCHED, 0);
if (ret) {
    printf("Failed to attach bpf\n");
    exit(1);
}
```

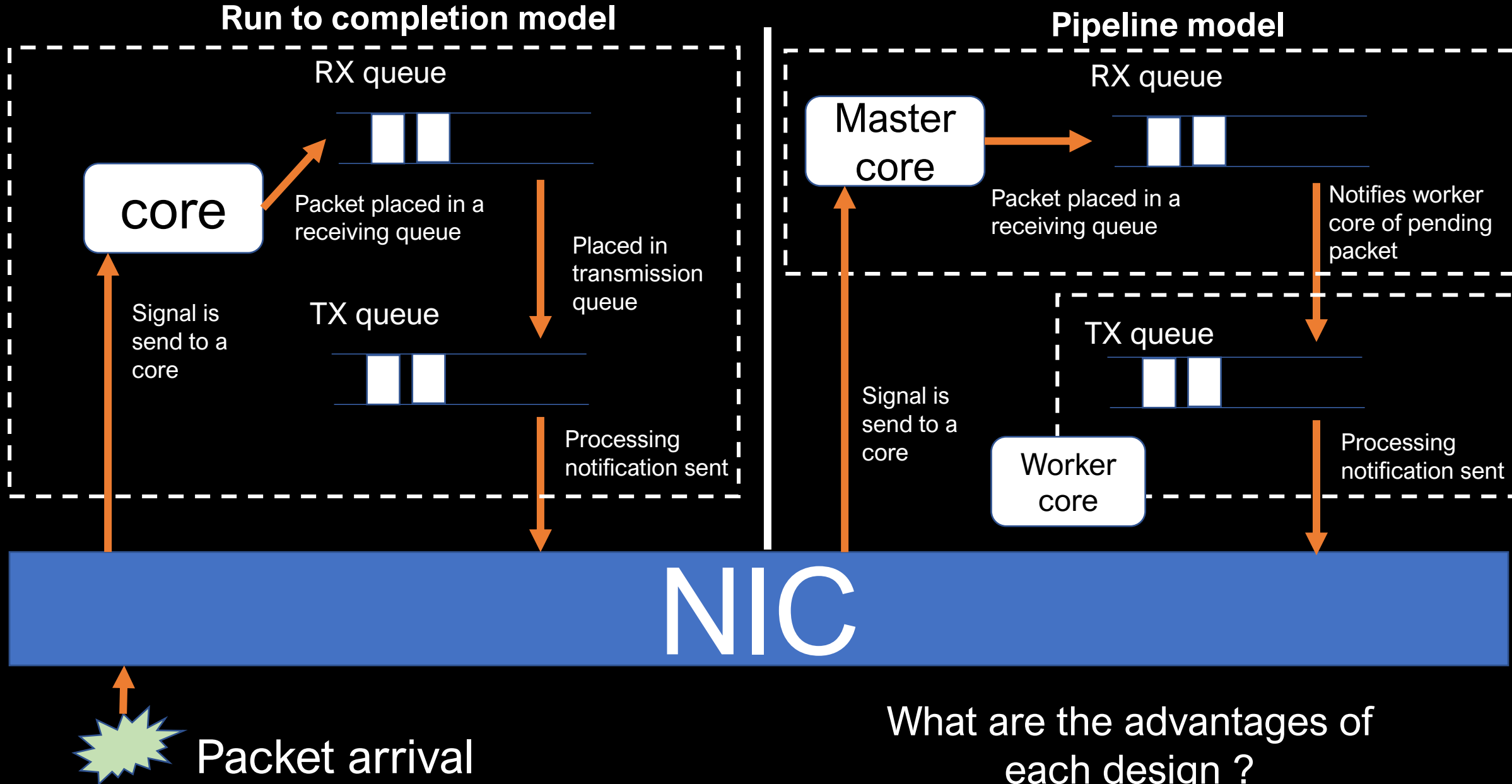
```
make
clang bpf_load.c -lbpf
./a.out
```

Building the function that uses eBPF with clang and running it.
More details: <https://github.com/djobiii2078/ebpf-sched-interface/>

NFV: Network Function Virtualization



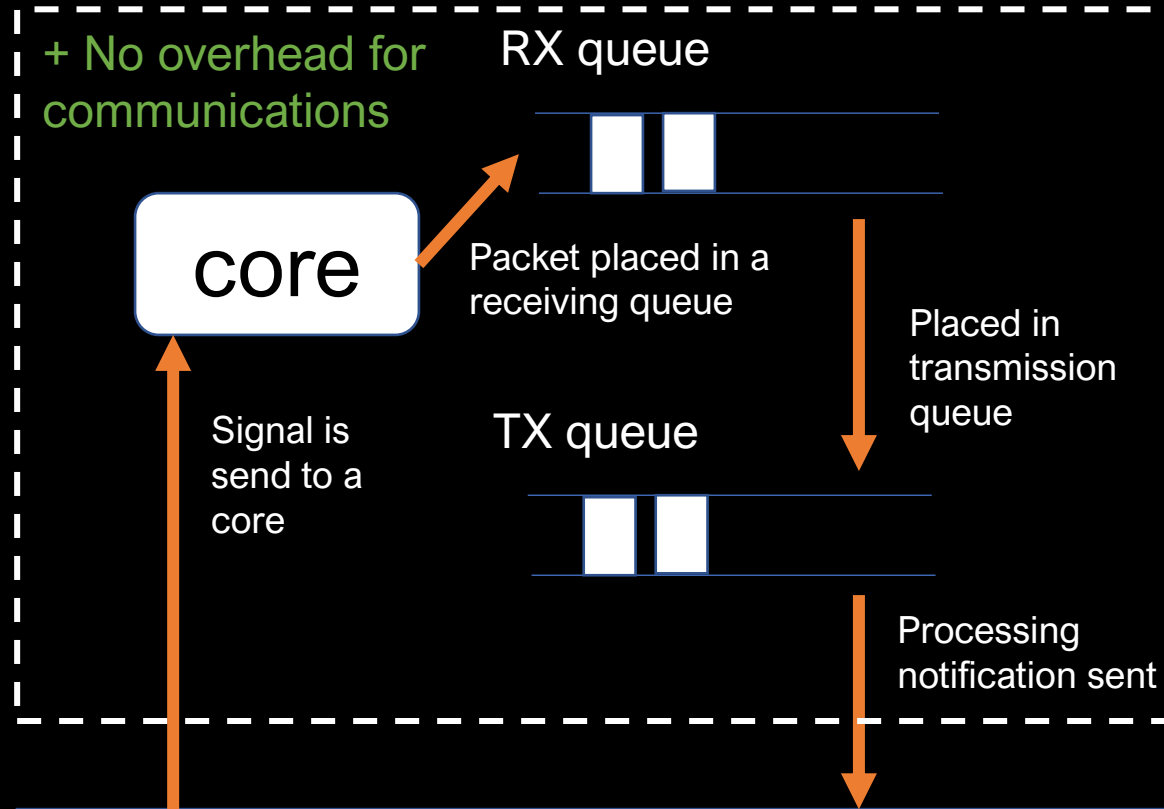
NFV: Network Function Virtualization



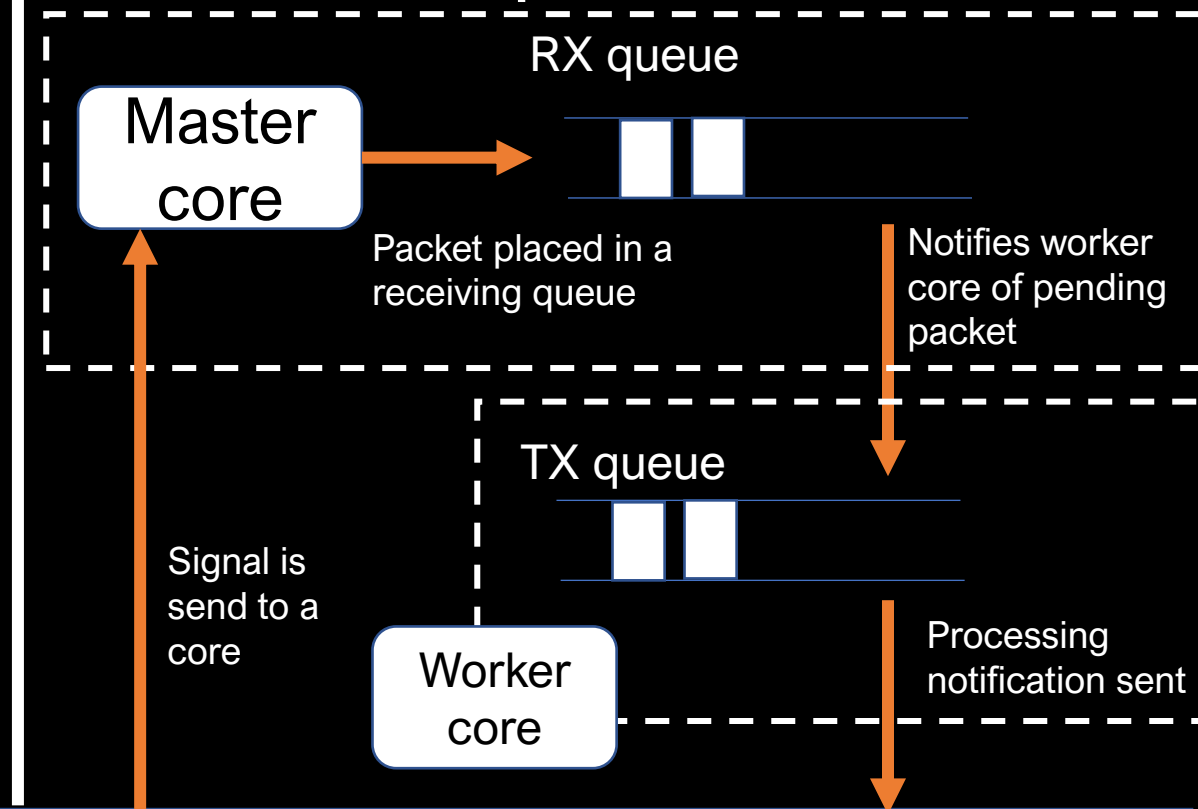
NFV: Network Function Virtualization

+ Scalable
+ No hardware support needed

Run to completion model

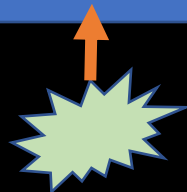


Pipeline model



NIC

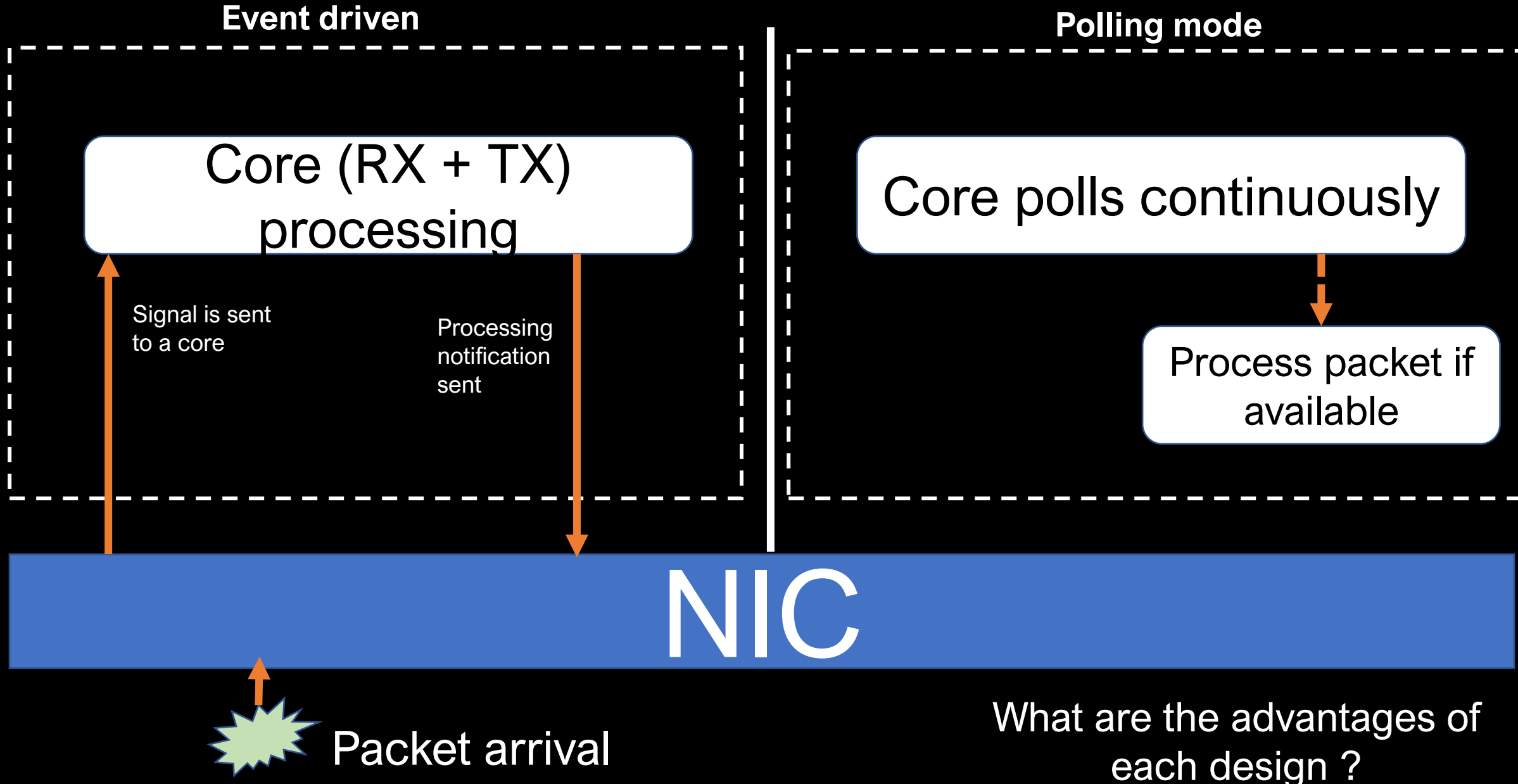
- Hardware support is needed



Packet arrival

- Overhead due to inter-core communications

NFV: Network Function Virtualization



NFV: Network Function Virtualization

Event driven

+ CPU time is preserved when there is no processing

Core (RX + TX)
processing

Signal is
sent to a
core

Processing
notification
sent

Polling mode

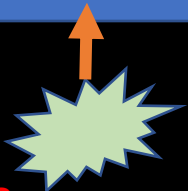
+ Performs well
for high load
since processing
is not halted

Core polls continuously

Process packet if
available

NIC

- Performs poorly on
high loads due to
continuous interrupts

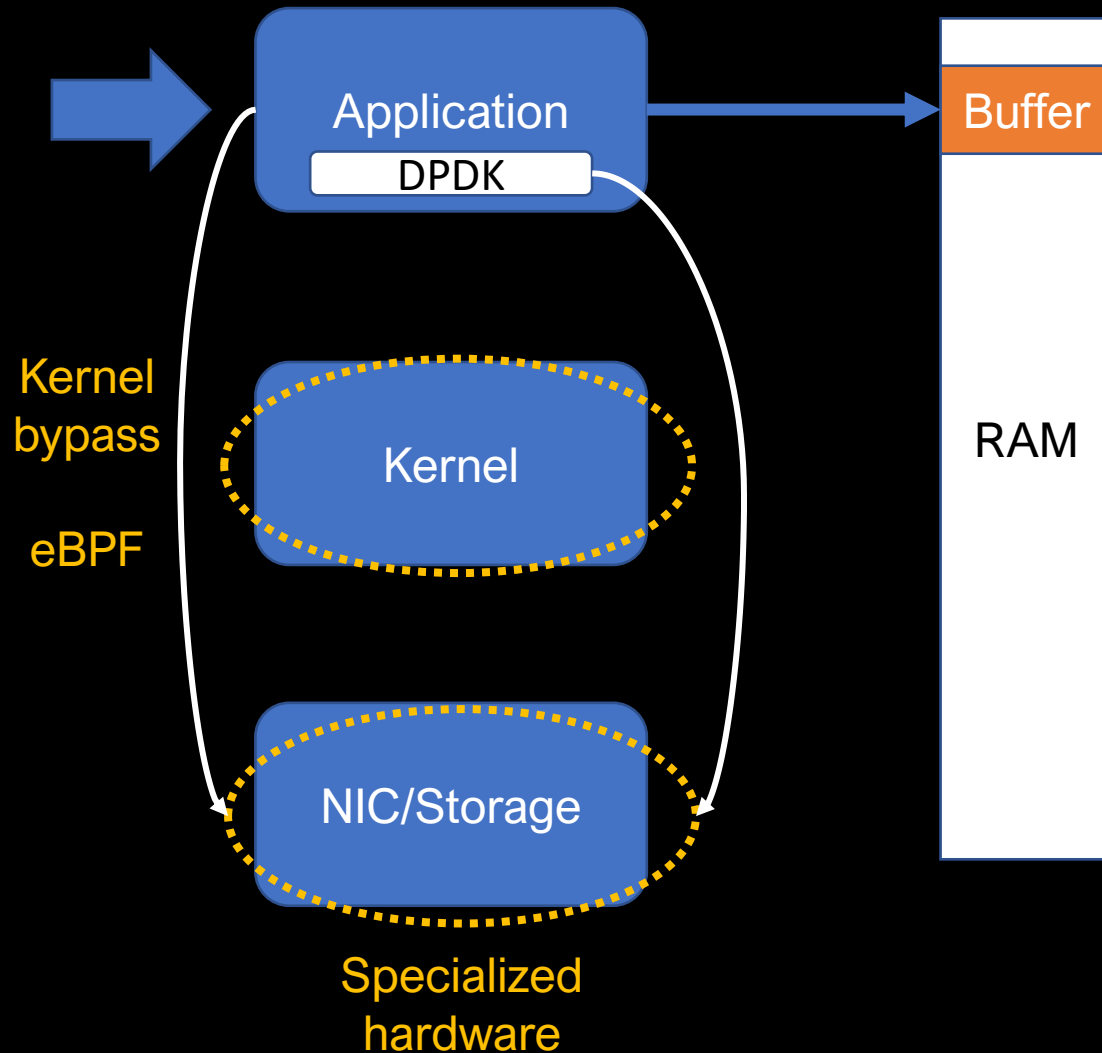


Packet arrival

- CPU time is stolen
even under no load

- Requires tuning
for correct period
configuration

NFV: Network Function Virtualization



DPDK : Data Plane Development Kit

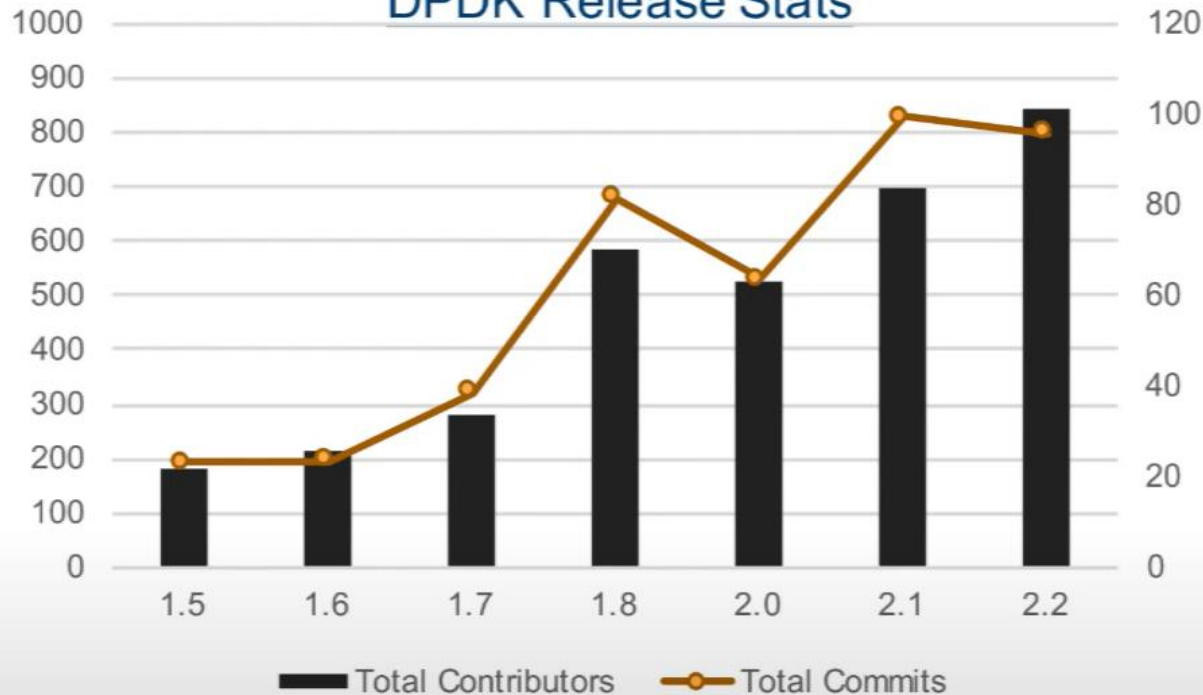
Eases configuration of kernel bypass techniques with fined tuned data structures and helpers to chooses the technique you need depending on the context

Started by Intel by **Venky Venkatesan** and can run on x86, IBM Power, and ARM CPUs (others are in road)

NFV: Network Function Virtualization

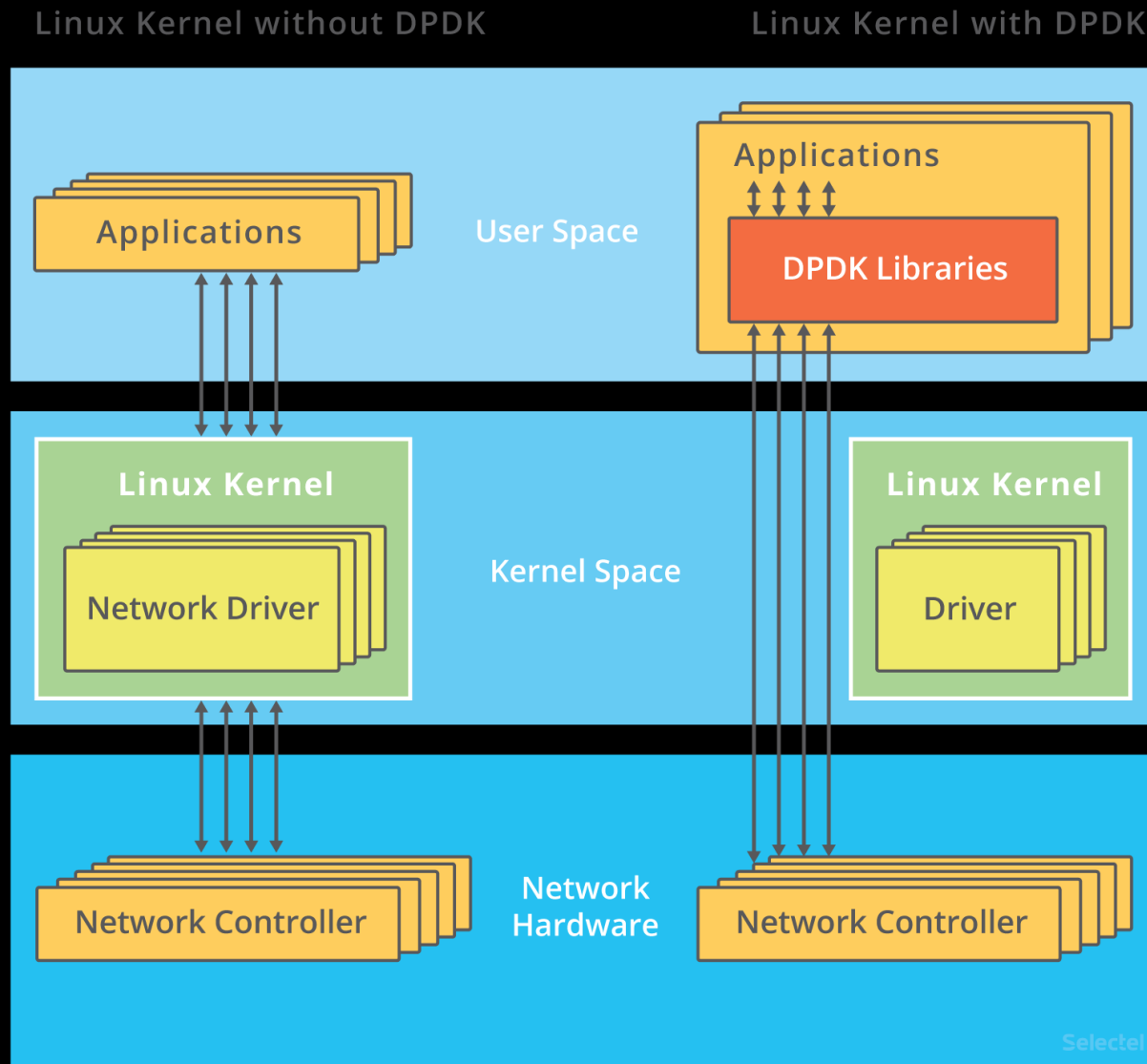
Open Source Stats

DPDK Release Stats



Fully open-source project with several major contributors: you can be one ☺

NFV: Network Function Virtualization

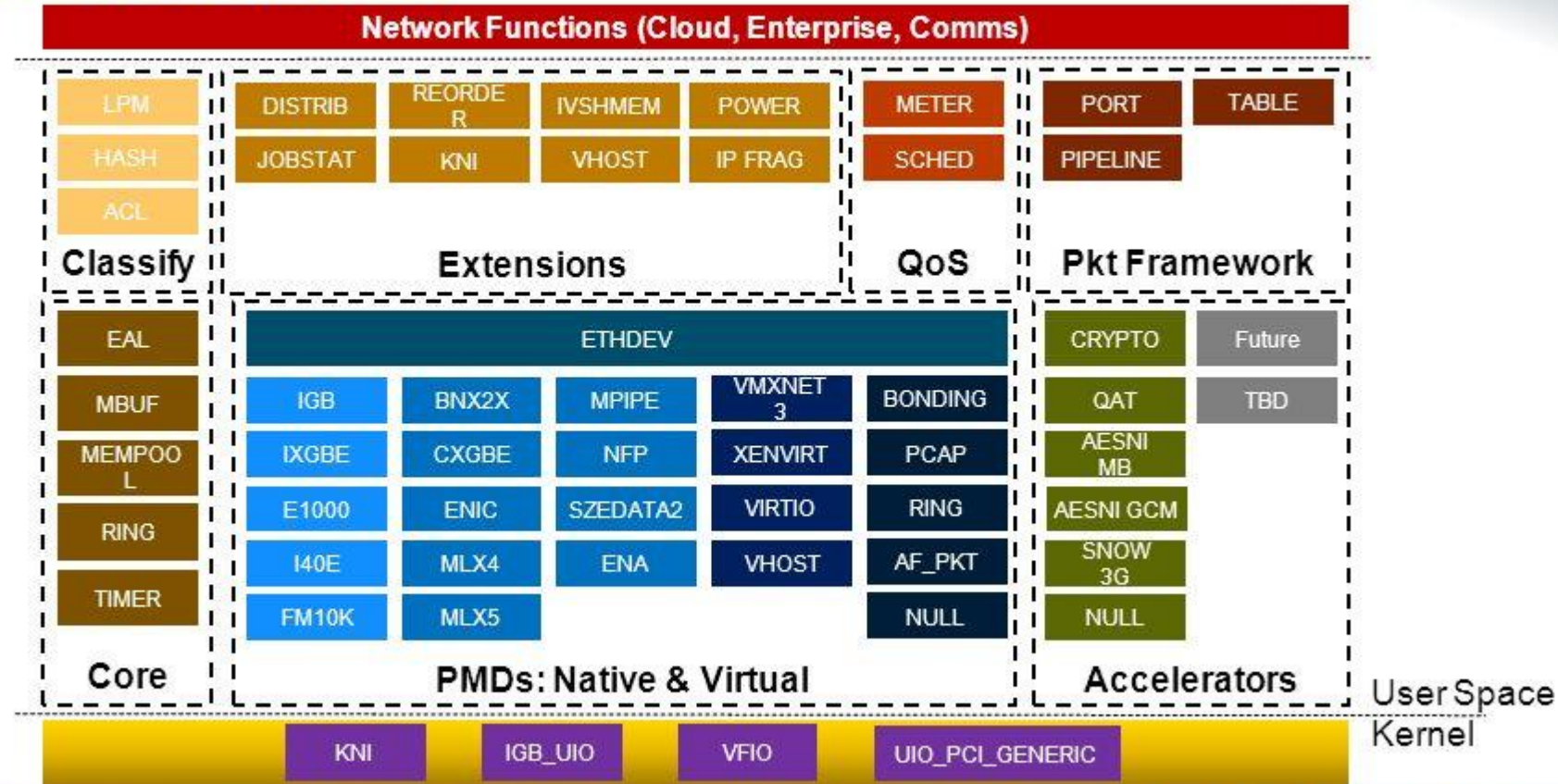


A simplified view of dpdk.

<https://selectel.ru>

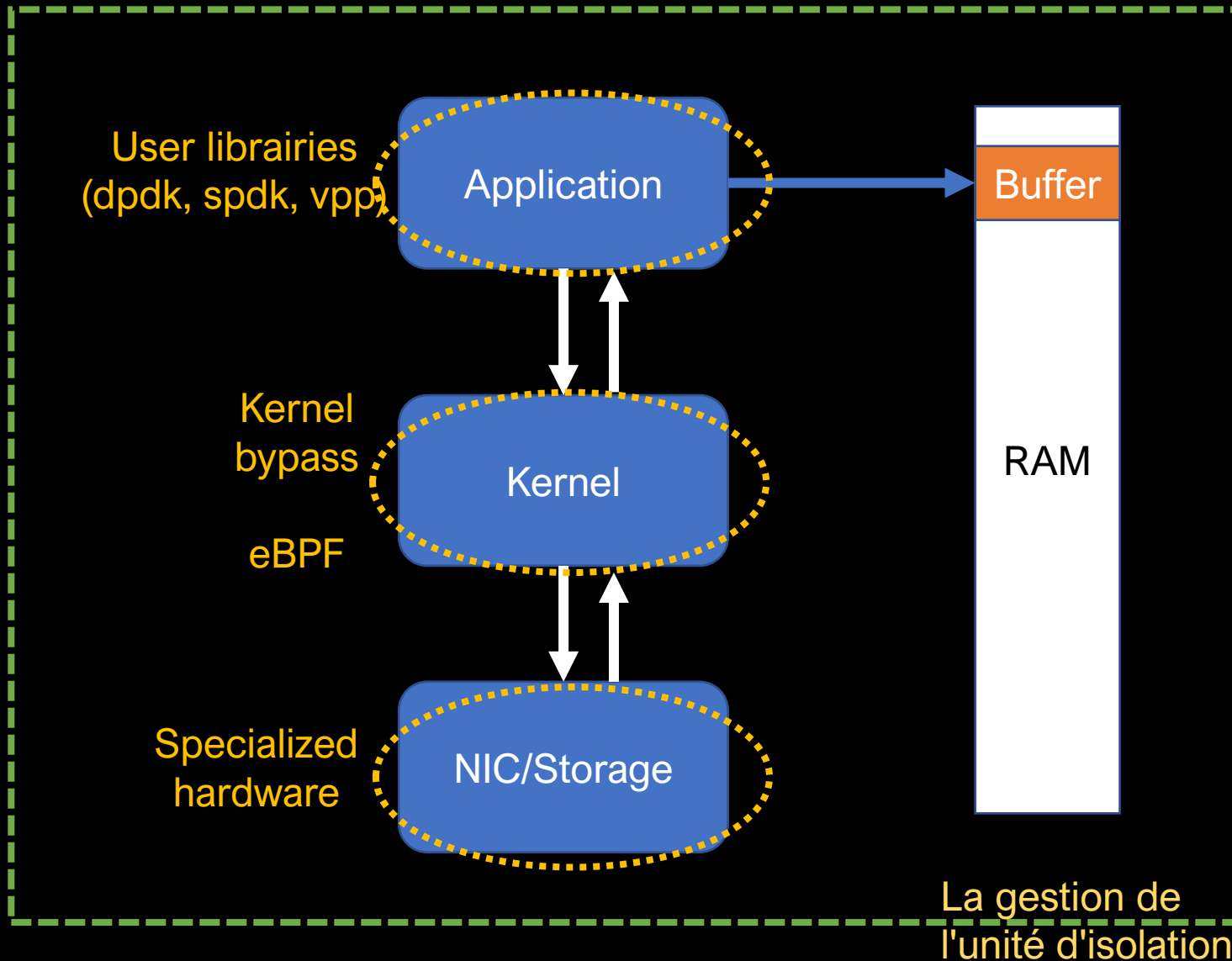
NFV: Network Function Virtualization

DPDK Framework



A detailed view of DPDK

NFV: Network Function Virtualization



TRAVAIL A FAIRE

Par groupe de $\frac{3}{4}$, choisissez une techno (dans la liste ci-dessous) et préparez une présentation sur cette dernière en précisant l'architecture, les cas d'utilisations possibles (dans le contexte des SDN/NFV), et une démo pas à pas qui montre la techno en action. La démo devra être reproduite (avec votre aide) par vos camarades. Technos:

- **DPDK**
- **SPDK**
- **VPP**
- **eBPF**
- **MWAIT/MONITOR**