6WINDGate Fast Path Implementation

6WIND

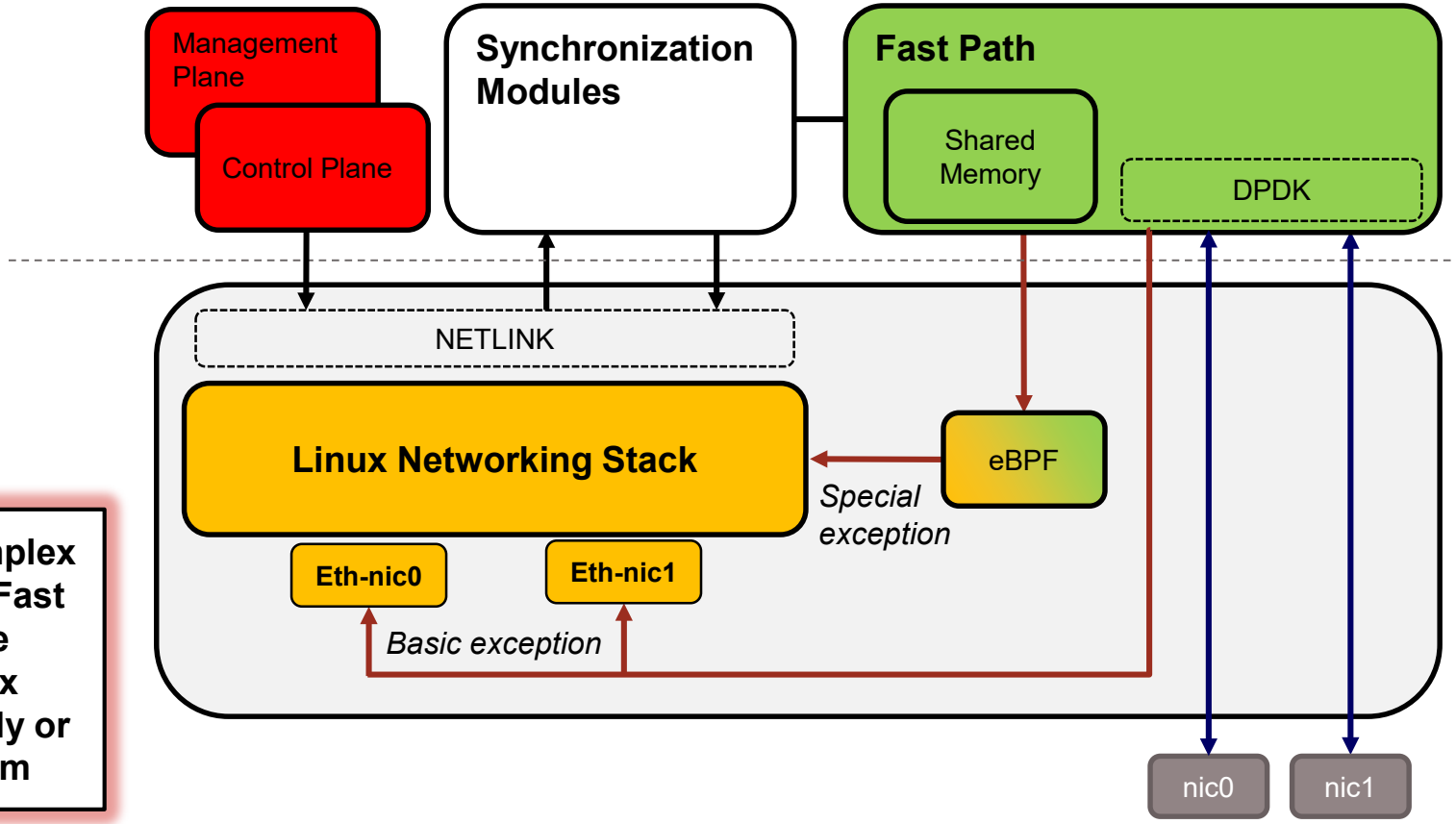
#SPEEDMATTERS For Serious Networks

6WINDGate Main Components

As a result, unmodified Linux applications transparently use the accelerated Data Plane as a standard Linux stack

Linux Networking Stack and Fast Path states are synchronized in a shared memory using Netlink

Dedicated optimized userland Data Plane running on top of DPDK



Packets that are too complex to be processed by the Fast Path (exceptions) are reinjected in the Linux Networking Stack directly or using an eBPF program

6WINDGate Detailed Architecture

1 Fast path

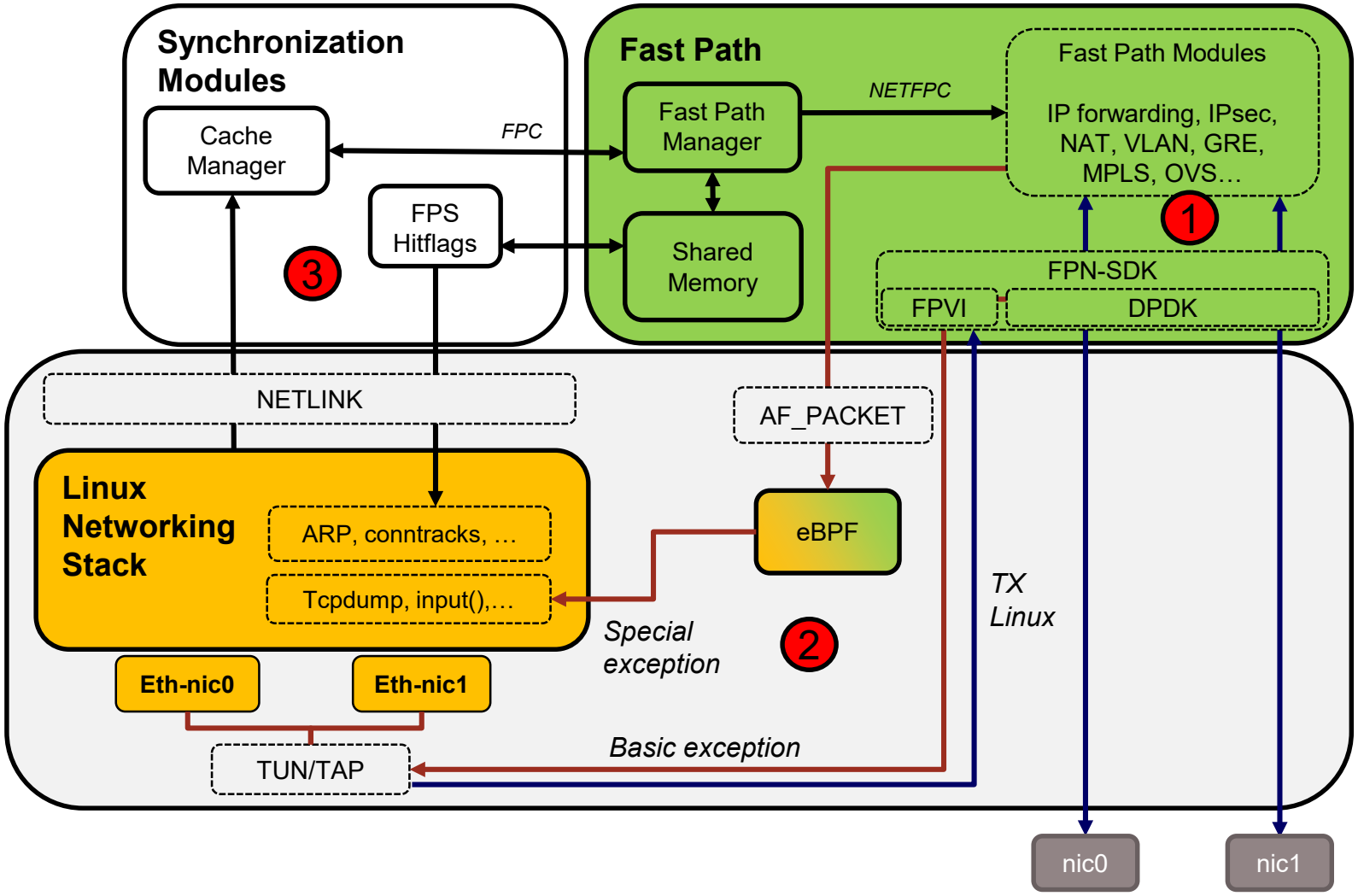
- Fast path modules on top of DPDK
- Process Linux TX packets
- Read configuration from shared memory and store usage and statistics

2 Exception path

- Basic RX for packets unmodified by fast path
- Special RX with eBPF for injecting packets modified by fast path in Linux networking stack

3 Synchronization

- Netlink monitoring to reflect kernel configuration into the shared memory
- FPS / Hitflags to update kernel states from shared memory



6WINDGate Detailed Architecture

1 Fast path

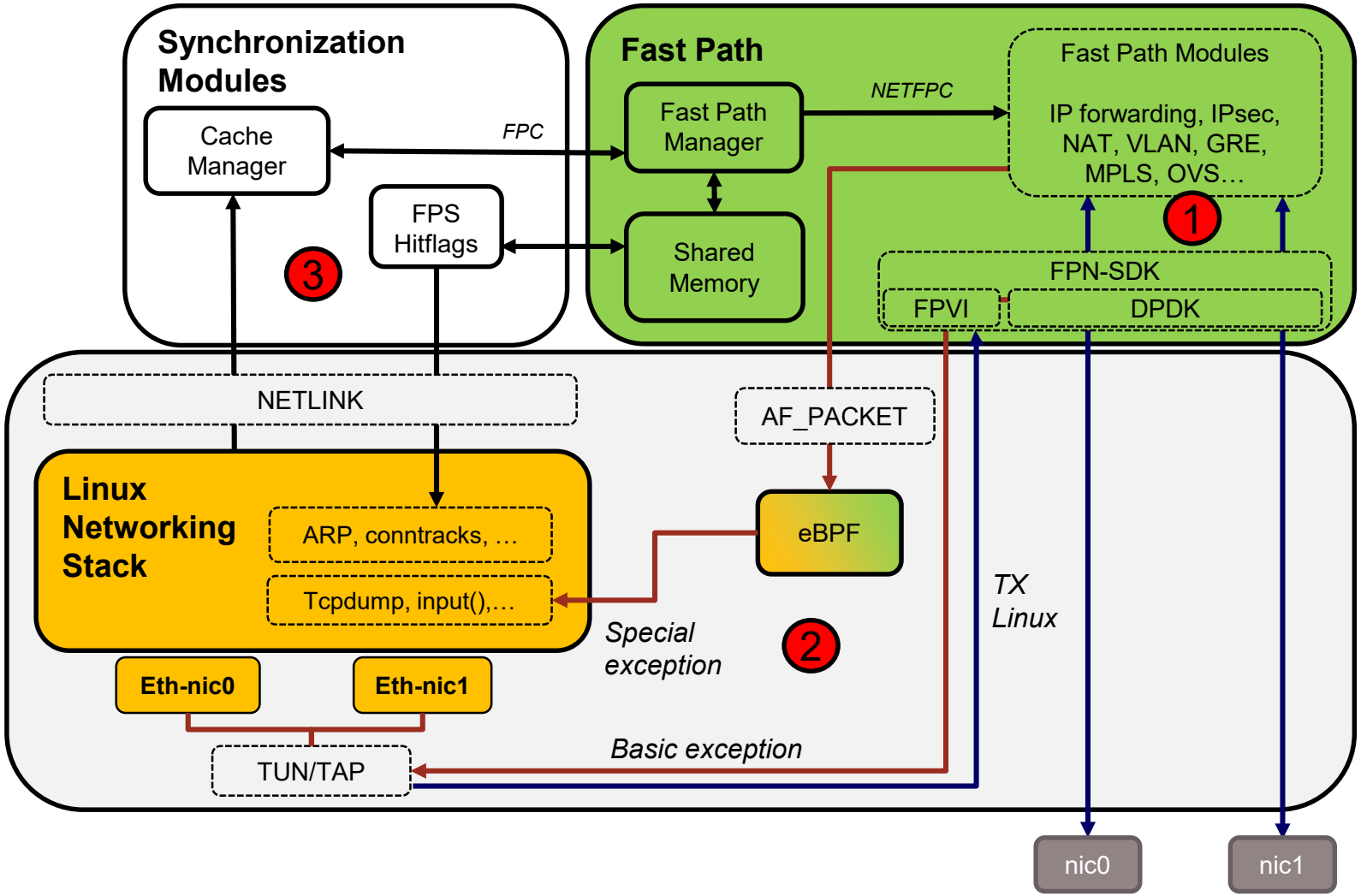
- Fast path modules on top of DPDK
- Process Linux TX packets
- Read configuration from shared memory and store usage and statistics

2 Exception path

- Basic RX for packets unmodified by fast path
- Special RX with eBPF for injecting packets modified by fast path in Linux networking stack

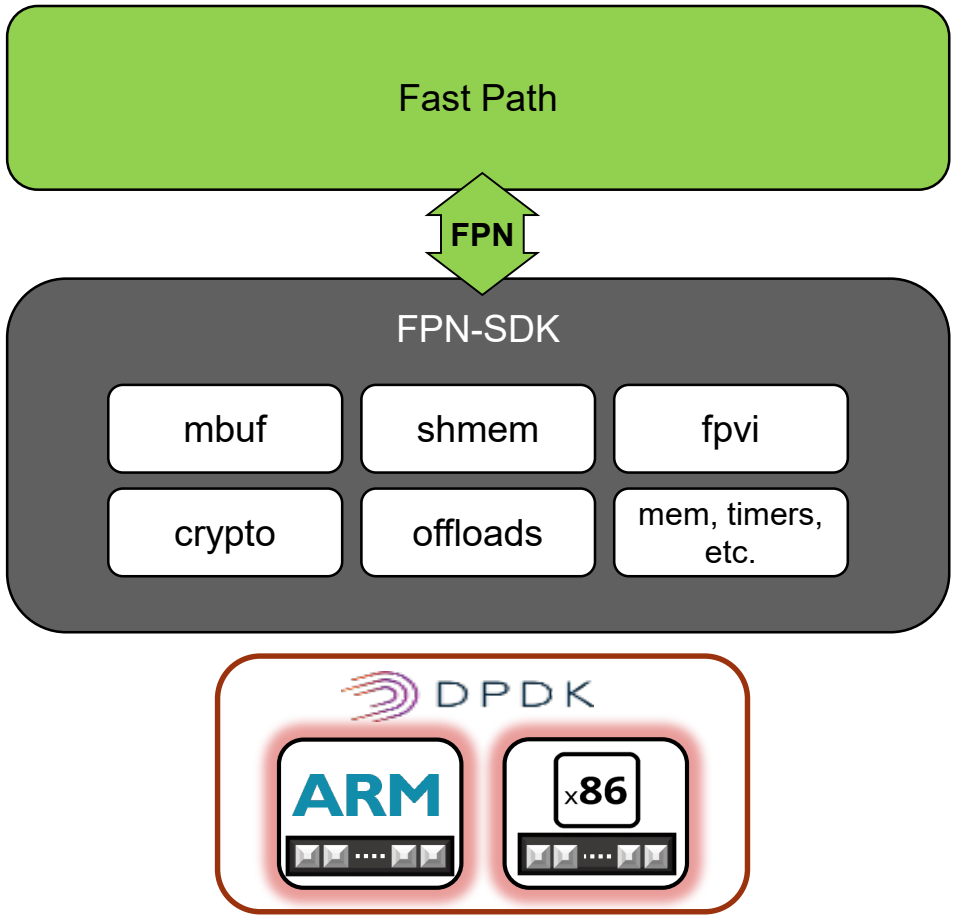
3 Synchronization

- Netlink monitoring to reflect kernel configuration into the shared memory
- FPS / Hitflags to update kernel states from shared memory



FPN-SDK Packet Processing Library

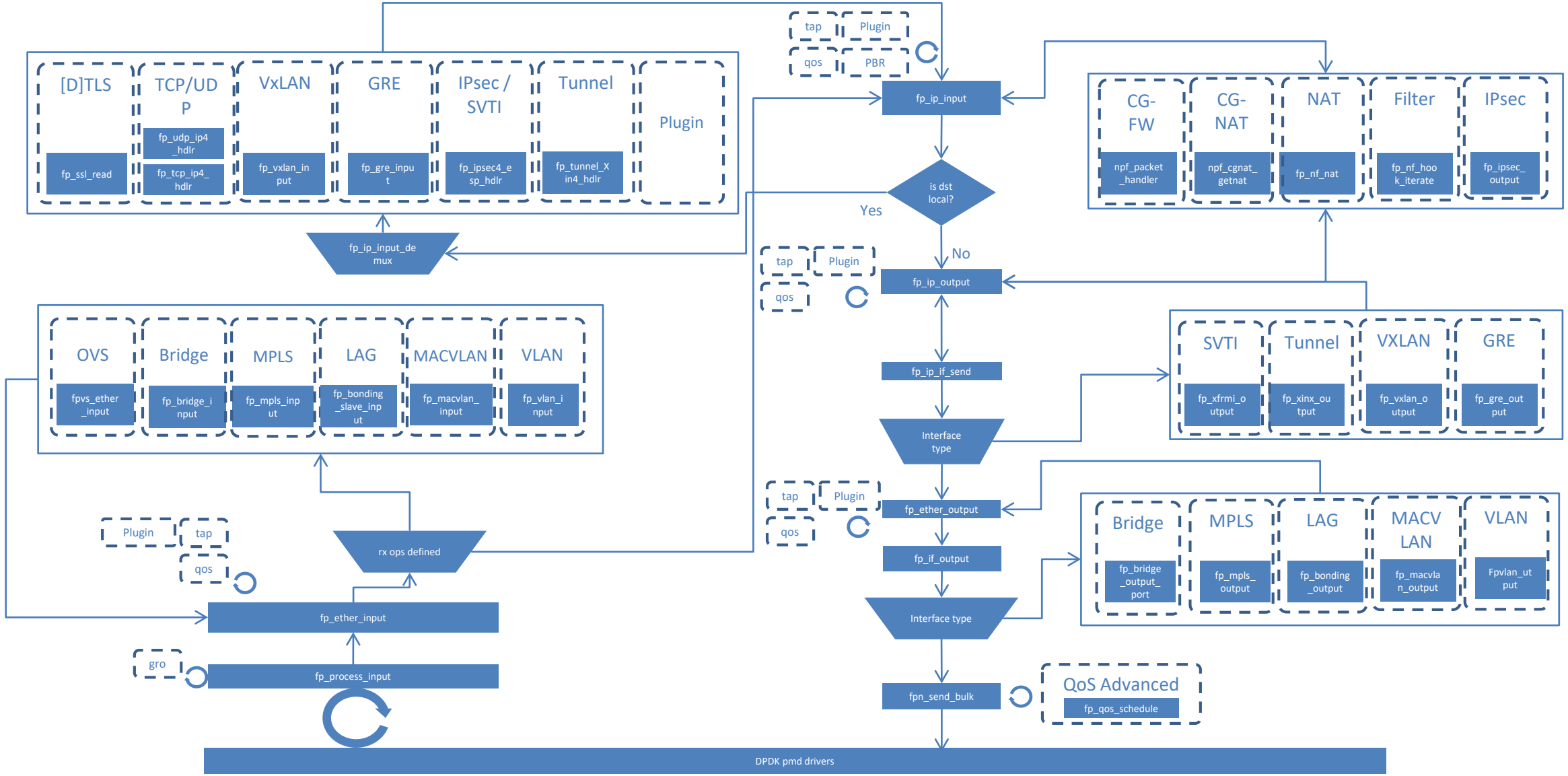
- **Hardware abstraction layer**
- **Northbound: FPN API for Fast Path modules**
  - Packet buffer (mbuf)
  - Shared Memory (userland / fast path)
  - FPVI: Linux interface abstraction
  - Crypto with HW support and SW fallback
  - Offloads: checksum, TCP (LRO, TSO)
  - And more: Control Plane protection, fast and scalable timers, memory pool and ring, lock and synchronization, atomic operations, CPU usage monitoring, function calls tracking for debugging, inter-core packet distribution
- **Southbound: hardware-specific SDK**



Protocol Implementation: Fast Path Modules

- **Generic software using the FPN-SDK generic API**
  - Same code used on supported hardware platforms
- **High performance architecture**
  - Run to completion model with pipeline capabilities when required (QoS…)
  - Implement only simple features to process 99 % of the traffic with maximum efficiency
  - Complex processing for the rest of the traffic is delegated to the Linux Networking Stack
- **Optimized code**
  - Straightforward case optimized (if xxx_likely())
  - Lockless, prefetch, cache usage
  - Statistics are implemented per core to minimize performance impact
- **Fast algorithms**
  - Route lookup with 8/8/8/8 trie
  - Security Policy Database lookup with automatic linear / trie switching
  - Load balancing of packets can be done (pipeline-hash plugin) if not provided by hardware
- **Use of hardware offloads abstracted by the FPN-SDK API**
  - Packet forwarding / sanity checks
  - Crypto, QoS….

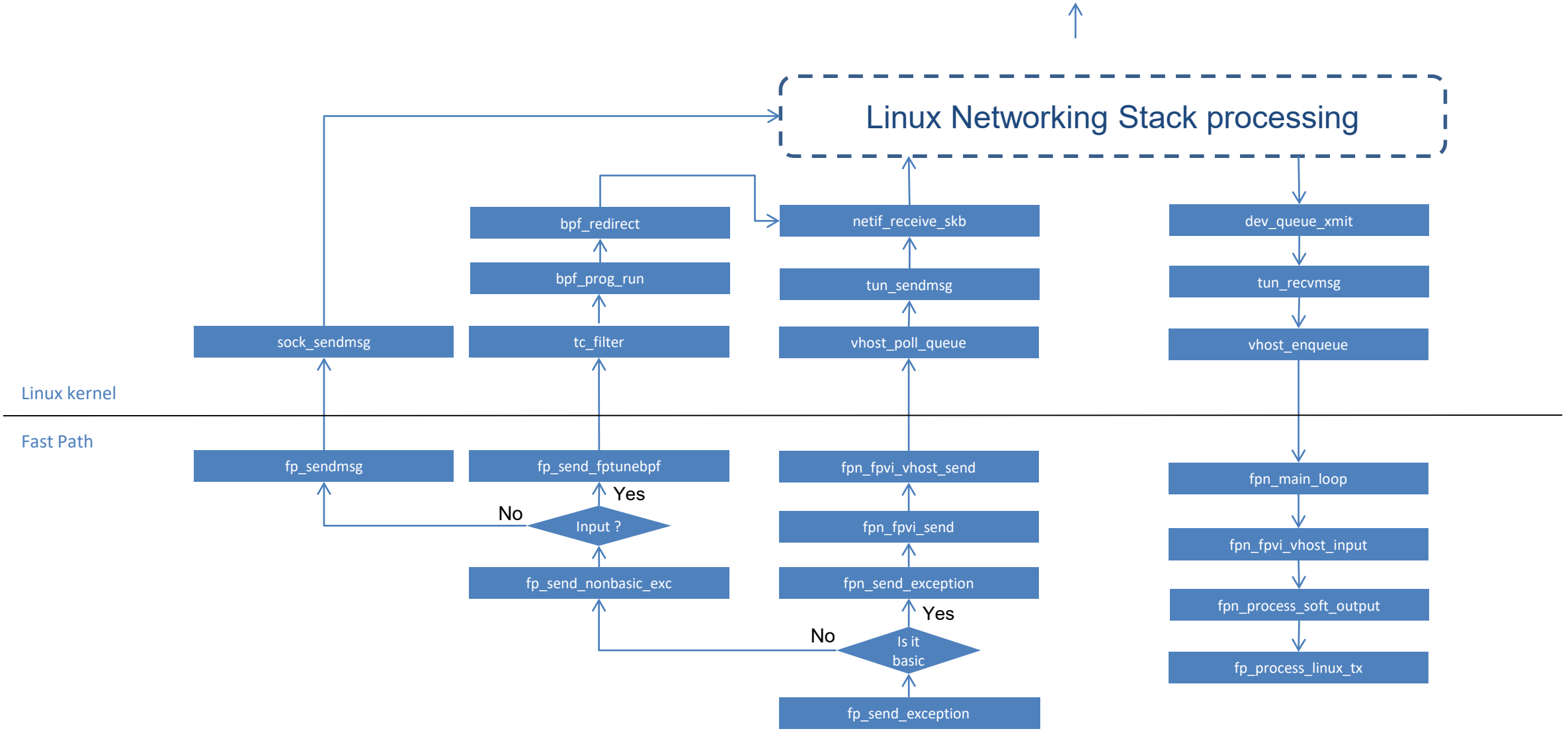
Packet Processing



Interface Between Fast Path And Linux Networking Stack: FPVI

- **Fast Path Virtual Interface**
  - Provides NIC representor in Linux for standard configuration (e.g. iproute2)
  - Implemented using FPN-SDK
    - For DPDK, it is using Linux tun/tap + DPDK PMD virtio-user
- **Fast path -> Linux : receive exception path**
  - Basic exception: packets are sent unmodified to get a standard processing by the Linux Networking stack
  - Special exception: packets have been preprocessed by the Fast Path and are injected at the right place into the Linux Networking Stack, thanks to an eBPF program
- **Linux -> Fast path : transmit exception path**
  - Packets are sent out the physical ports by the Fast Path which owns NIC drivers

Packet Exception Processing



Fast Path Commands: fp-cli

- **To configure the NIC**
  - Get / Set NIC flow control
  - Link speed
  - Offloads
- **To monitor the fast path**
  - Display statistics
  - Display debugging information
- **Command output is formatted in json**
- **API for extensions**

```
root@localhost:~# fp-cli iface
1:lo [VR-0] ifid=1 (virtual) <UP|RUNNING|FWD4|FWD6|MPLS> (0x3b)
        type=loop mac=00:00:00:00:00:00 mtu=0 no numa tcp4mss=0 tcp6mss=0
        IPv4 routes=0  IPv6 routes=0
        if_ops: rx_dev=none rx_early=none tx_dev=none ip_output=none
3:dp0 [VR-0] ifid=3 (port 0) <UP|RUNNING|FWD4|FWD6|MPLS> (0x3b)
        type=ether mac=de:ed:01:77:94:a2 mtu=1500 numa=0 tcp4mss=0 tcp6mss=0
        IPv4 routes=2  IPv6 routes=0
(…)
root@localhost:~# fp-cli iface-json
[
  {
    "vrfid": 0,
    "ifaces": [
      {
        "name": "lo",
        "ifid": 1,
        "mac": "00:00:00:00:00:00",
        "vrfid": 0,
        "mtu": 0,
        "numa": "no numa",
        "ipv4_routes": 0,
        "ipv6_routes": 0,
        "master": null,
        "port": "virtual",
        "type": "loop",
        "flags": [
          "up",
          "running",
          "fwd4",
          "fwd6",
          "mpls"
        ],

(…)
```

6WINDGate IP Forwarding Example: Call Flow