6WINDGate Exceptions and Linux - Fast Path Synchronization

#SPEEDMATTERS For Serious Networks

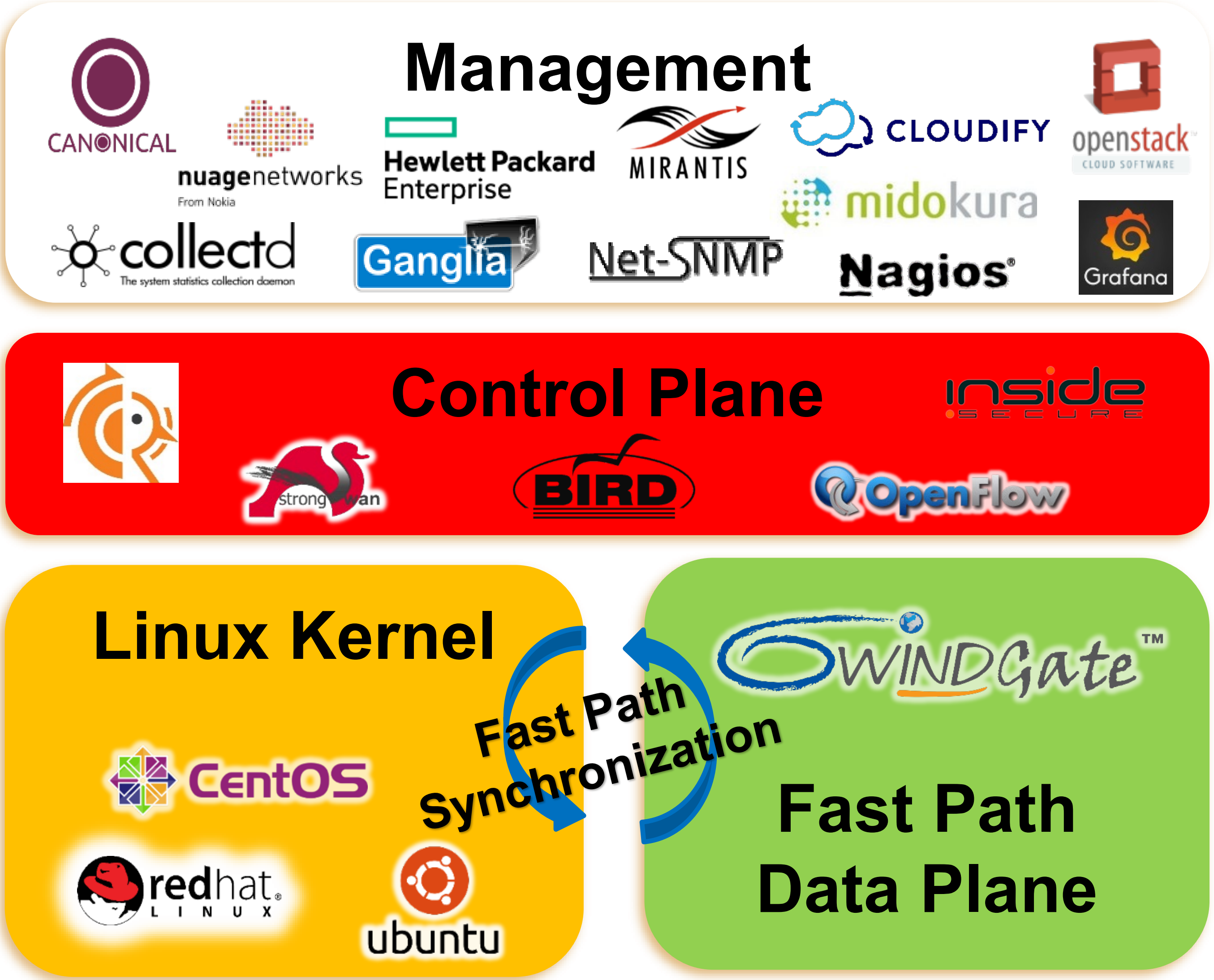
Integration of Fast Path with Control and Management Planes

- **There are two options to achieve the integration of a high performance isolated Fast Path with Linux Control and Management Planes**
  1. Redesign how Control and Management Planes interact with the Data Plane
     - Requires a significant amount of work to adapt and validate a large number of complex protocols
     - Used by VPP
  2. Reuse "as is" existing Linux Control and Management Planes
     - Rely on the design of a Linux-friendly Data Plane to let the Fast Path act as a transparent solution to Linux

- **This second option has been successfully implemented in 6WINDGate using Linux - Fast Path synchronization**

Benefits of Linux – Fast Path Synchronization

- Existing Linux applications are not modified and developing new applications is pure Linux development
- Compatible with third-party open source or commercial control plane applications that configure Linux (routing, IKE, ...)
- Linux management tools can be re-used (iproute, iptables, ipset, brctl, ovs-*ctl, tcpdump, etc.)
- Supports major Linux distributions



Management

Control Plane

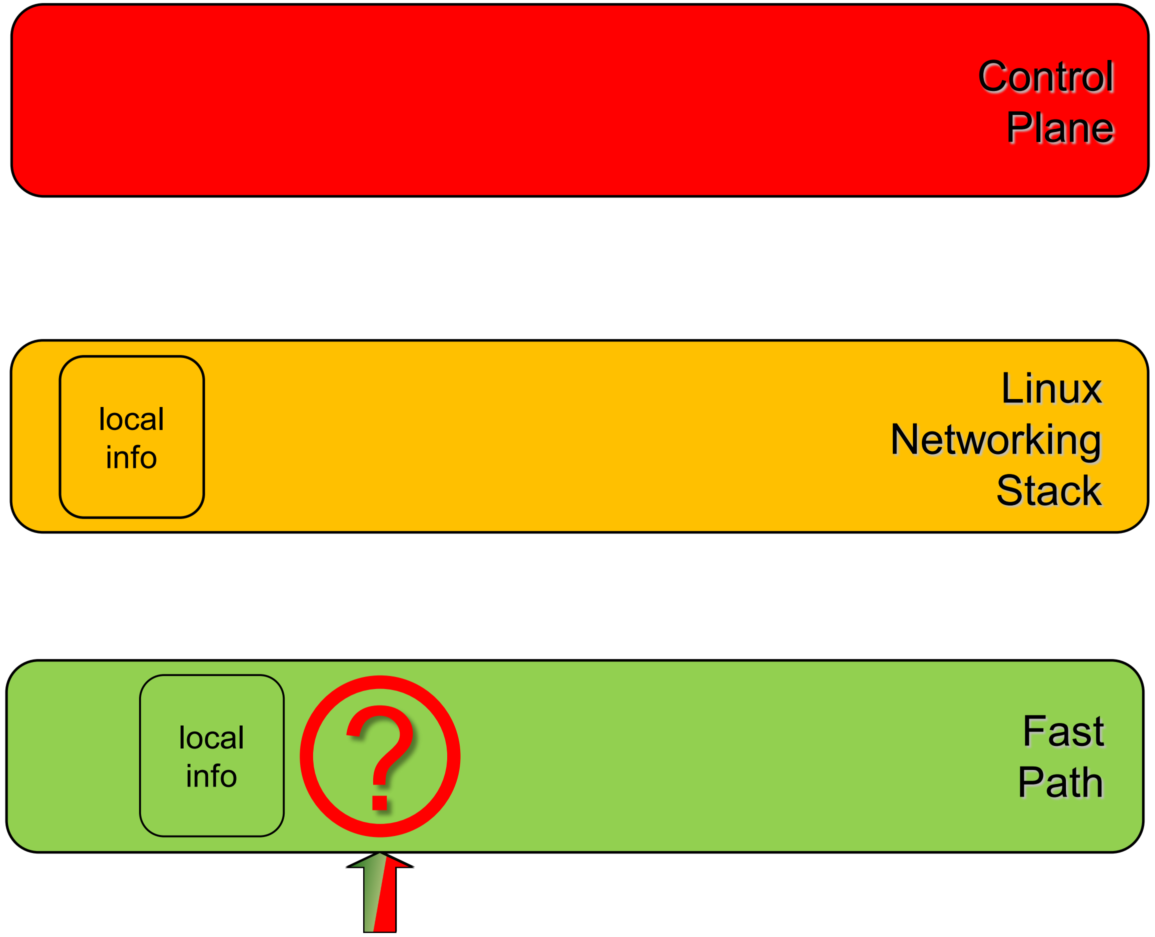
Linux Kernel

Fast Path Synchronization

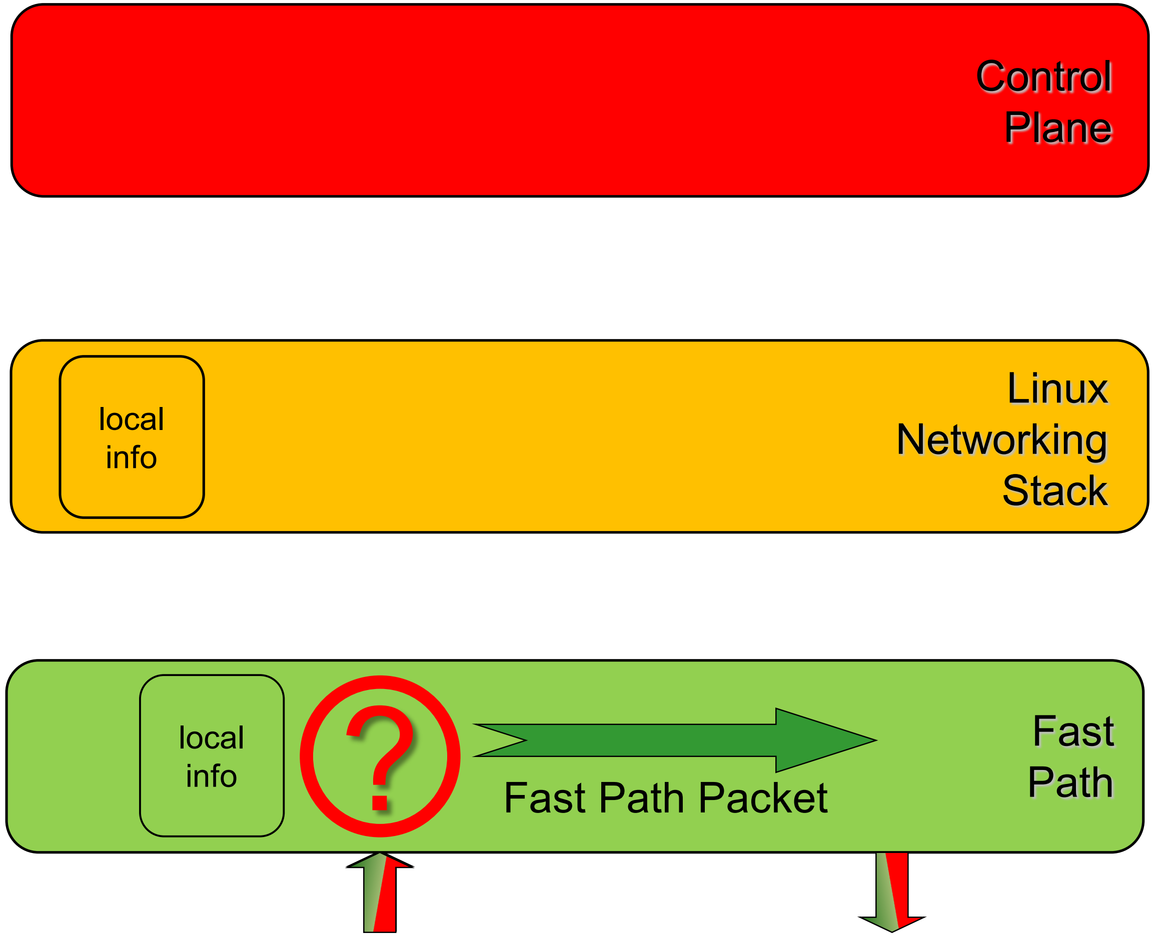
Fast Path Data Plane

Linux Running 6WINDGate is Linux

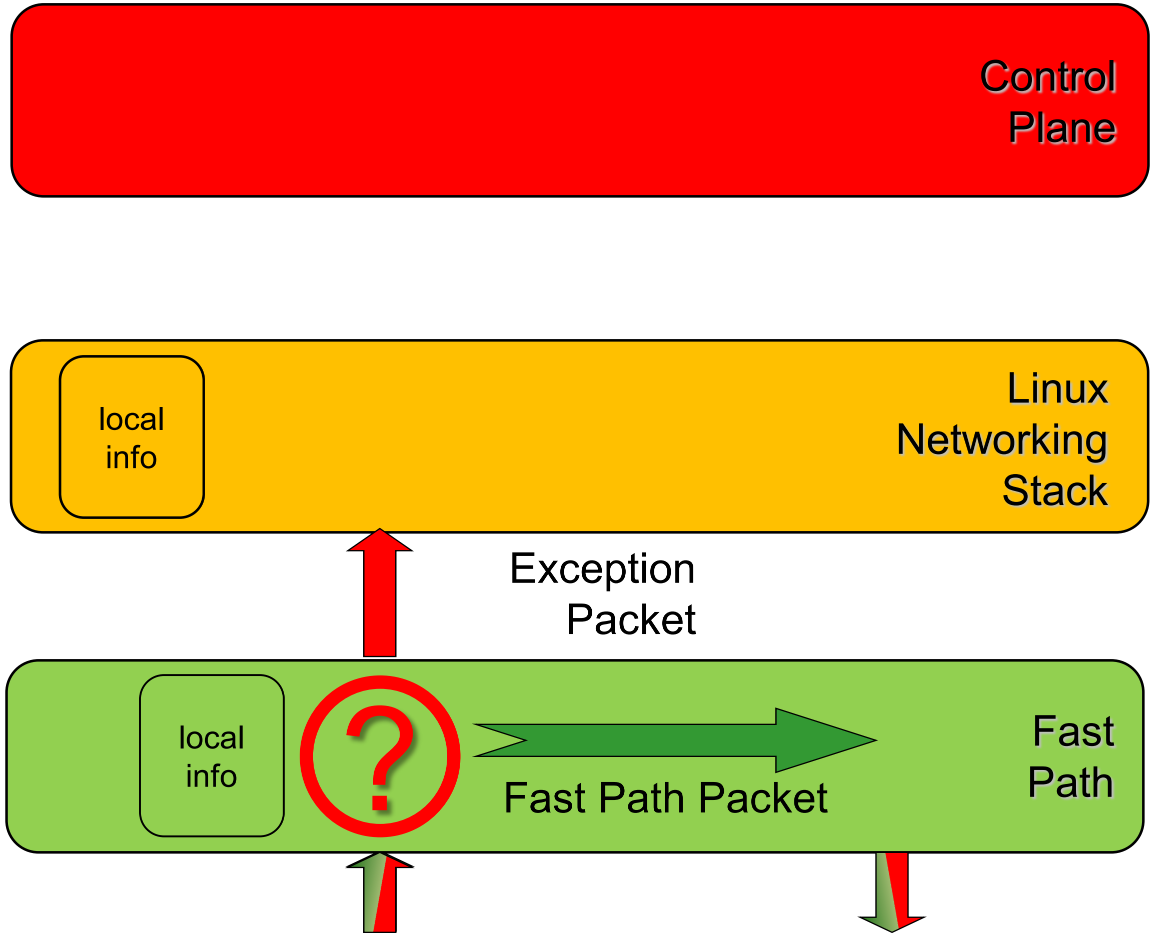
Exceptions And Continuous Synchronization



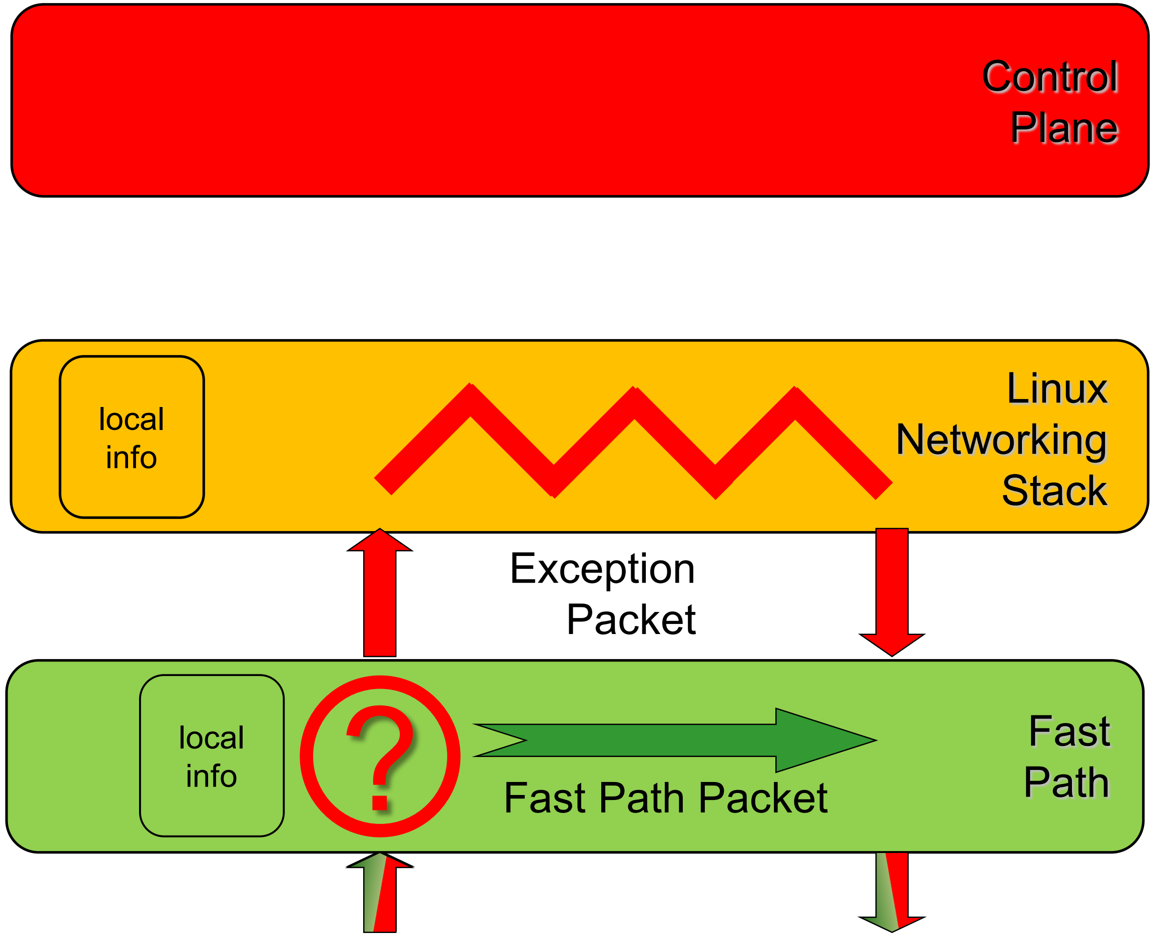
Exceptions And Continuous Synchronization



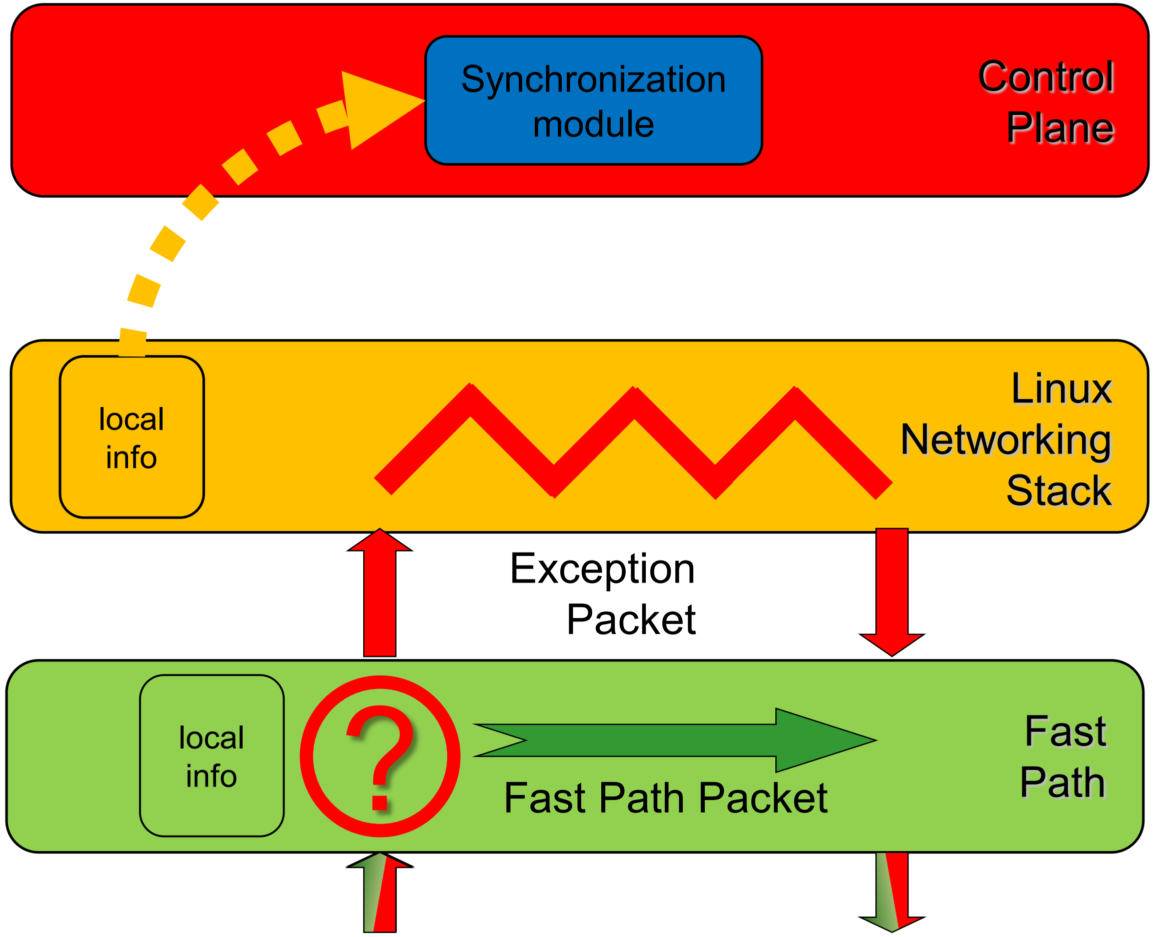
Exceptions And Continuous Synchronization



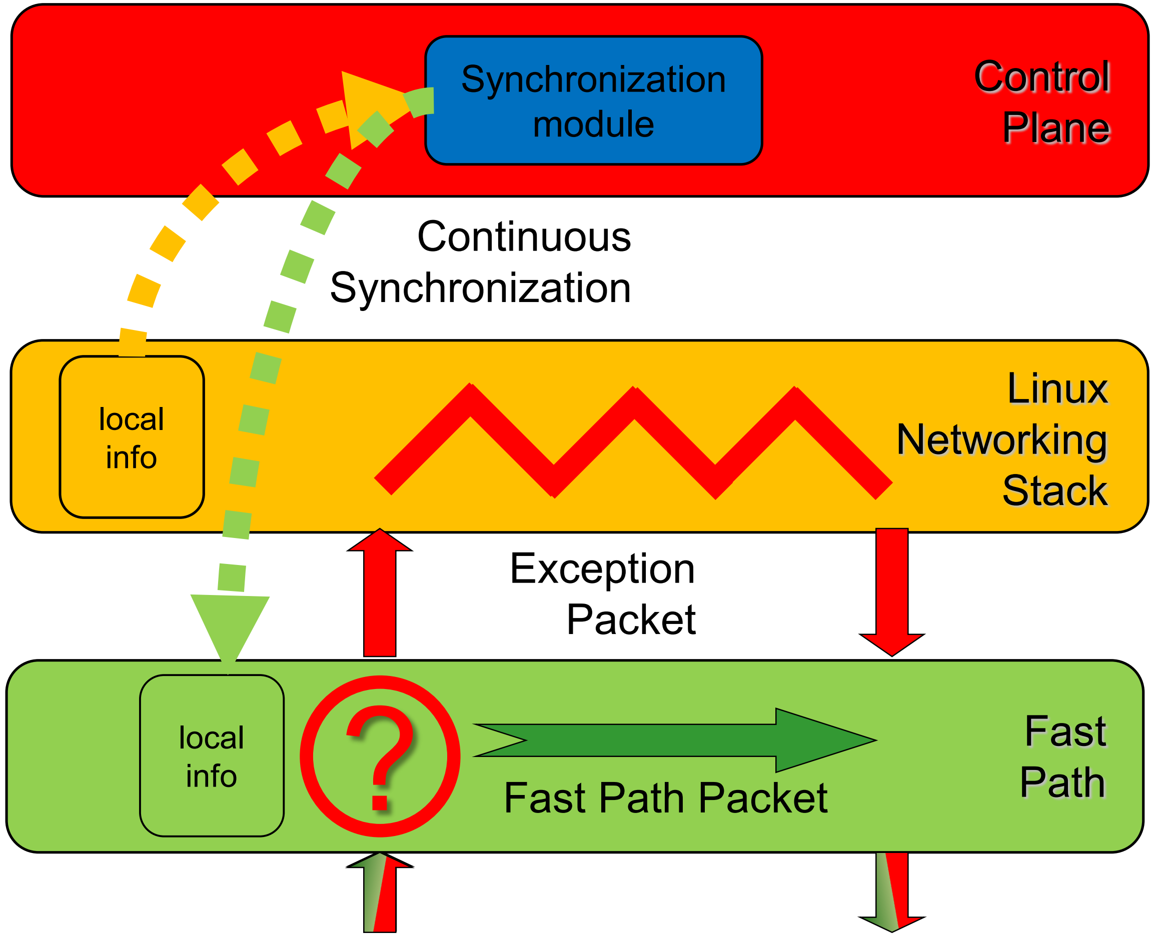
Exceptions And Continuous Synchronization



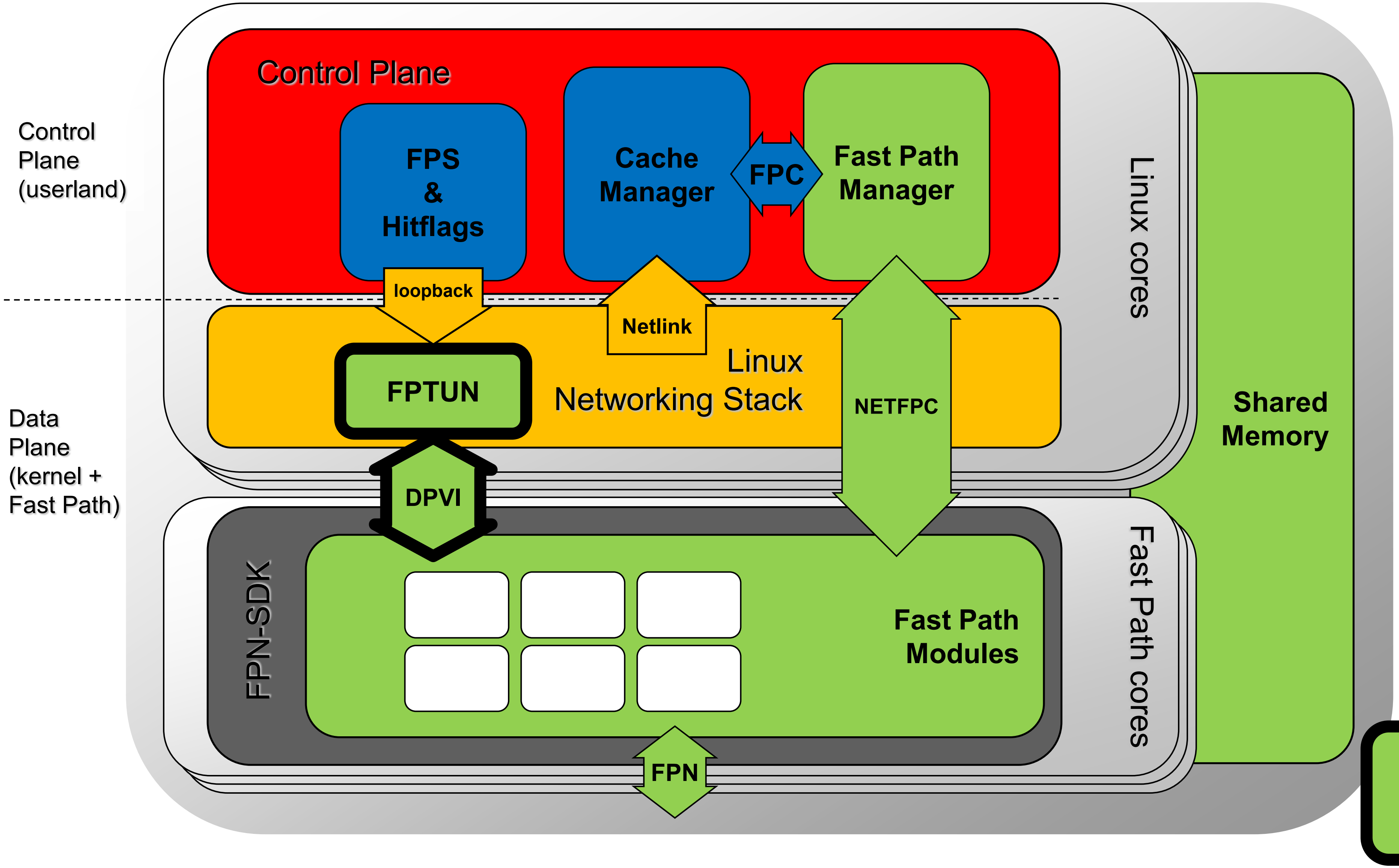
Exceptions And Continuous Synchronization



Exceptions And Continuous Synchronization



Linux - Fast Path Synchronization: Exceptions



Modules involved in exception handling

Exception Strategy

- **All packets are received by the Fast Path, but some are delegated to Linux**
  - Local destination
  - Missing processing information in Shared Memory (ARP, IPsec SA, etc.)
  - Unaccelerated protocol

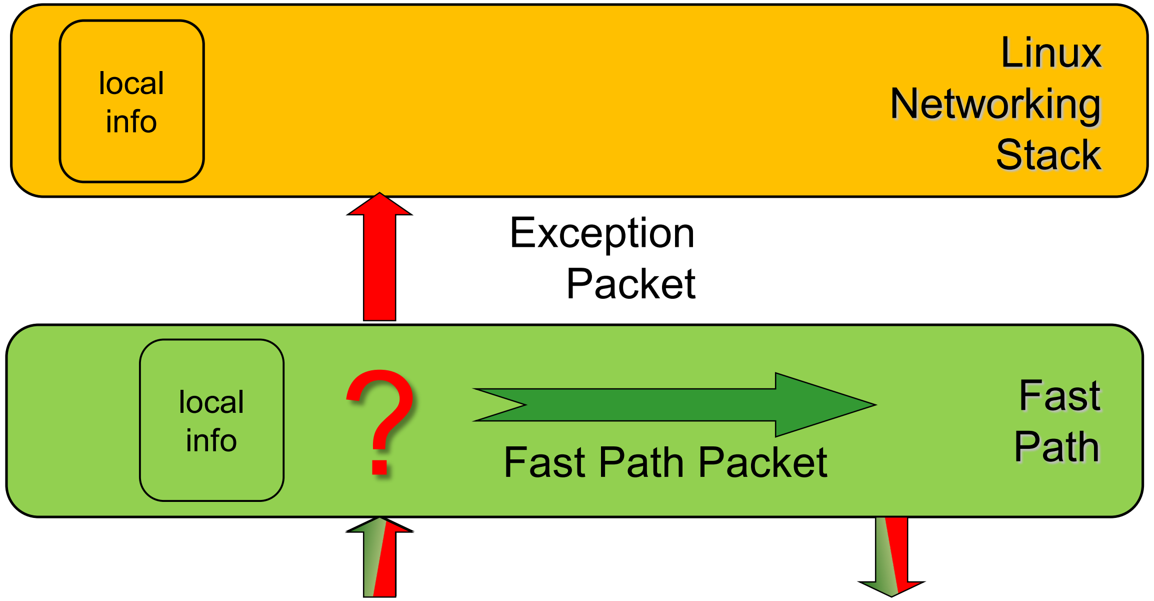
- **They are sent to Linux through DPVI/FPTUN**
  - DPVI = standard logical Linux netdevice
  - Basic exception for standard processing
  - Extended exception for packets that have been preprocessed by the Fast Path: thanks to FPTUN header, packets are injected at the right place into the Linux Networking Stack

- **Packets are then processed by the Linux Networking Stack**
  - Missing information (ARP, IPsec SA, etc.) is resolved by Linux and will be synchronized to the Fast Path (see next slides)

- **Benefits**
  - Complete networking stack, relying on Linux for unaccelerated protocols
  - Fast Path benefits from rich Linux Control Plane, no need to develop or change Control Plane daemons
  - No change to Linux

Exception Cases

- **Packets intended at Control Plane**
  - ICMP echo requests
  - Control Plane daemons (BGP, OSPF, IKE, etc.)
  - …
- **Missing info to process packet**
  - No L3 route available
  - No L2 address available for destination/gateway
  - No IPsec info (SP/SA)
  - Missing conntrack info
  - …
- **Protocols delegated to Linux**
  - ARP/NDP
  - ICMP stack (TTL expiration)
  - …

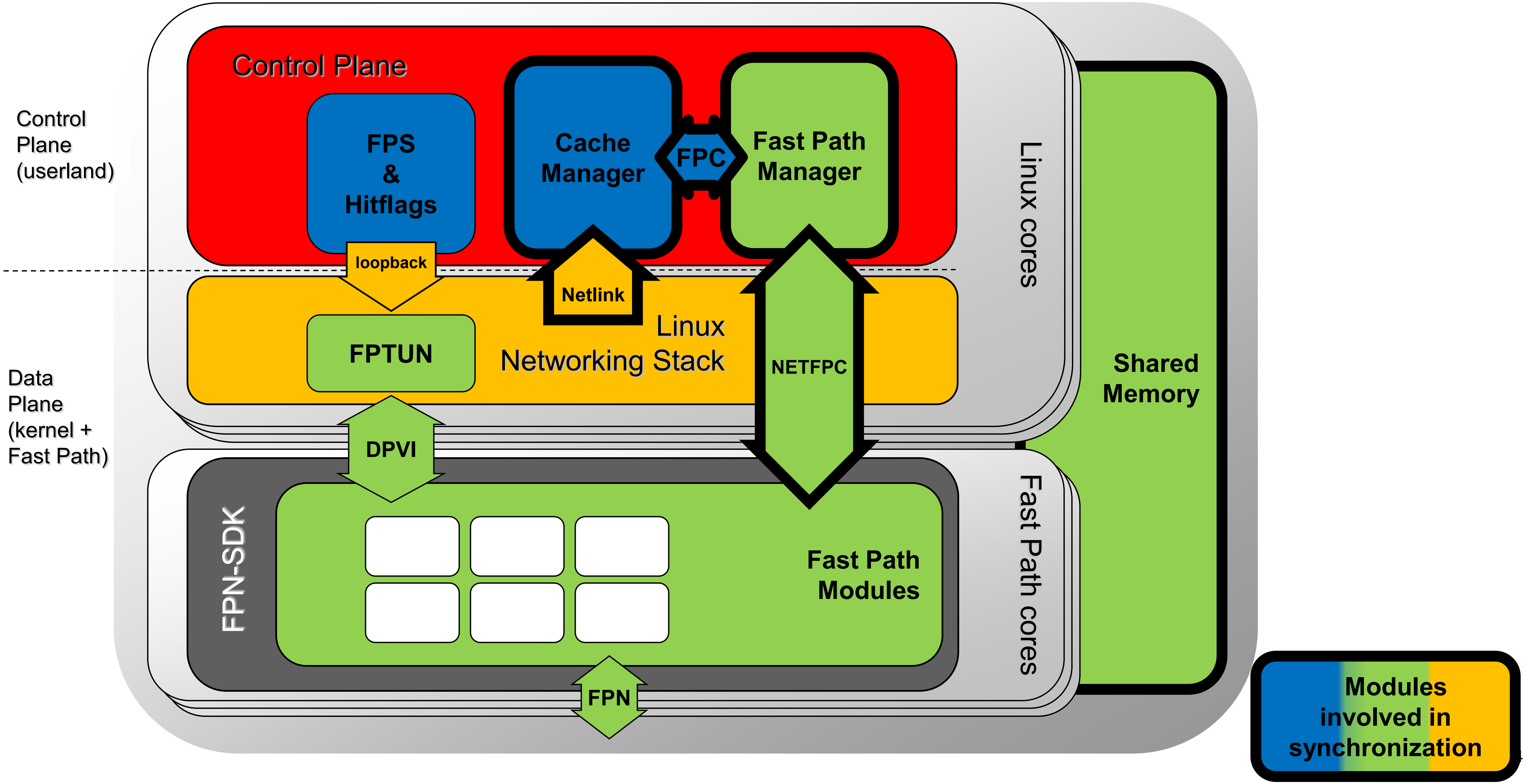


Exception Types

- **Basic exceptions**
  - Default case
  - Original packet sent to the Linux Networking Stack
    - Restore IPv4/IPv6 headers, L2 headers
  - Example: route lookup fails during forwarding

- **Extended exceptions**
  - Original packet cannot be restored
  - It is encapsulated with FPTUN
    - Specific header with meta-data + packet as-is
  - Will be processed by the FPTUN driver in Linux to inject it at the right place in the Linux Networking Stack
  - Example: route lookup fails on decrypted packet after IPsec processing

Linux - Fast Path Synchronization: Configuration



Configuration Synchronization Between Linux and Fast Path

- **Based on two applications**
  - Cache Manager (CM): cmgrd executable
  - Fast Path Manager (FPM): fpmd executable
  - Local or remote communication between CM and FPM is done by the Fast Path Control API (FPC API)
    - Allow distributed architectures with Control Plane and Data Plane running on different processors

- **Full synchronization path**

Linux Networking Stack ➔ *Netlink* ➔ Cache Manager ➔ *FPC Protocol* ➔ FPM ➔ *Shared Memory* ➔ Fast Path

FPM ➔ *NETFPC* ➔ Fast Path

Synchronization: Cache Manager (CM)

- Part of the Linux Fast Path Synchronization module
- Run as a Linux userland application
- Listen to the Netlink socket, for kernel internal states (Control Plane and configuration updates)
- Transform Netlink messages into FPC messages
- Control Plane modules (routing, IKE, PPP…) are not modified

Synchronization: FPC API

- **FPC API**
  - Interface between Cache Manager and Fast Path Manager
  - Define the exchange protocol and the structures of the configuration messages exchanged between the Cache Manager and the Fast Path Manager
  - Can work on distributed architectures with non co-localized Cache Manager and Fast Path Manager

- **Dedicated protocol**
  - UNIX or TCP socket
  - client/server
  - Common header
  - Type, sequence number (SN), report, length

Synchronization: Fast Path Manager (FPM)

- **Part of the Linux Fast Path synchronization module**

- **Run as a Linux userland application**

- **Application that translates FPC API messages to configure Fast Path modules using**
  - Read / write Shared Memory
  - Send / receive notifications to / from Fast Path through NETFPC

- **Can work on distributed architectures with non co-localized Cache Manager and Fast Path Manager**

Synchronization: Shared Memory

- **Contain structures for**
  - Physical ports
  - Forwarding table
  - Statistics
  - IPsec processing
  - etc.

- **Read/write access for**
  - FPM: writes local information received from CM through FPC messages
  - Fast Path: reads local information used for packet processing (L2/L3 entries, IPsec SAs, etc.) and writes statistics
  - FPS: reads statistics

- **Allocation is specific to processor architecture, contents are generic**

Synchronization: NETFPC

- **Used to trigger an event from Linux to Fast Path**
  - Set the MTU on an interface (the Fast Path owns the drivers)
  - Configure MAC address or promiscuous mode
  - …
- **Communication socket between FPM and Fast Path modules**
  - Point to point communication
  - IPv6 RAW
  - Link-local addresses
  - Dedicated network namespace, isolated from networking configuration

Synchronization: VRF

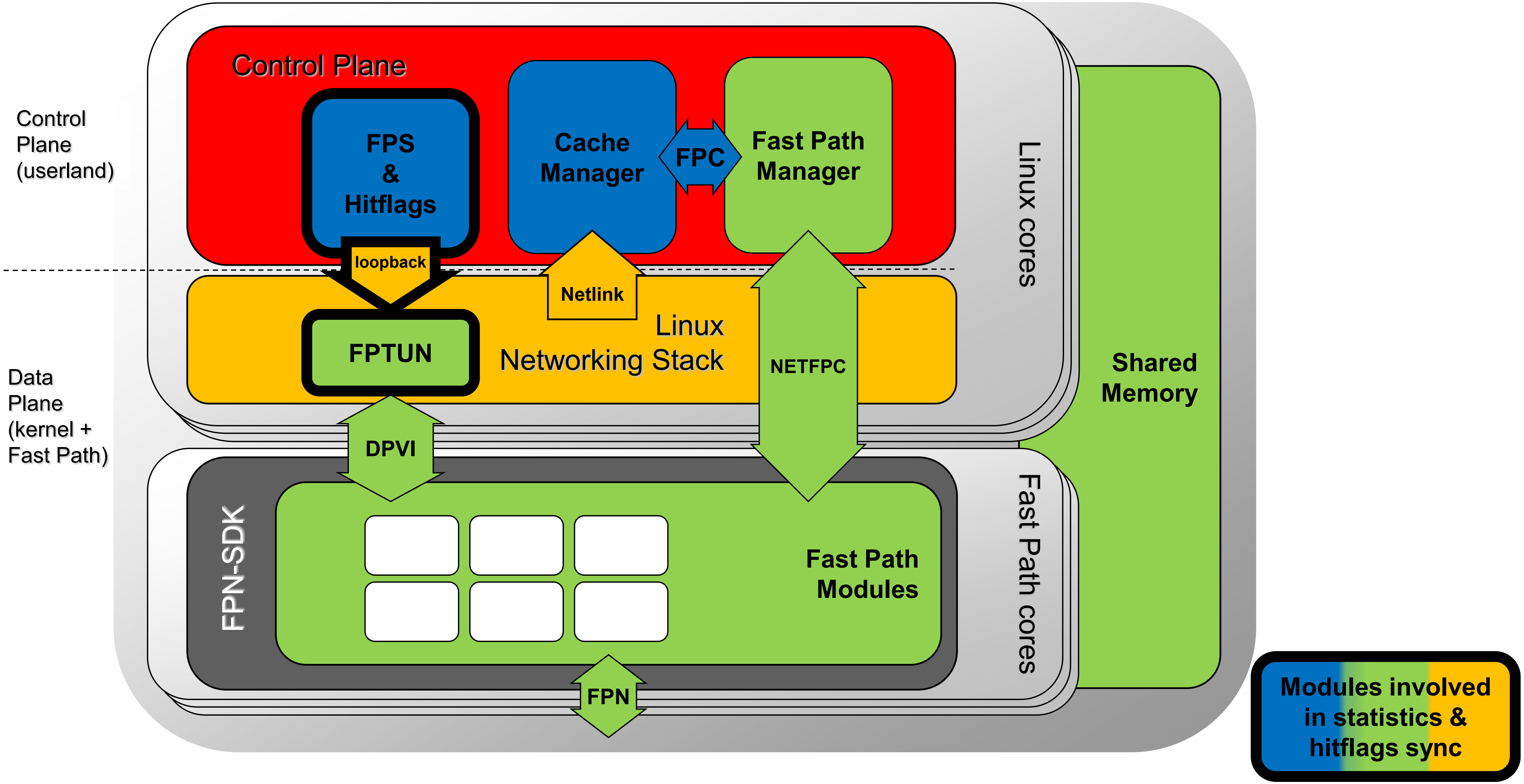
- **Virtual Routing and Forwarding (VRF): IP technology that allows multiple instances of a routing table to work simultaneously within the same router**

- **6WINDGate provides support for Virtual Routing and Forwarding (VRF) in all Fast Path modules**

- **In Linux, VRFs are configured using network namespaces**

- **The Linux / Fast Path Synchronization - VRF module implements synchronization of Linux netns to Fast Path VRFs**
  - Userland API: libvrf
    - This library allows to manage and monitor 6WINDGate VRFs from any Linux userland process
  - Kernel API: netns-vrf.ko
    - The kernel module translates Linux network namespaces to VRF instances at kernel level

Linux - Fast Path Synchronization: Statistics & Hitflags



Synchronization: Statistics (FPS)

- **Reports Fast Path statistics into the Linux Networking Stack**
  - Fast Path modules update the Shared Memory with statistics
  - FPS daemon reads Shared Memory statistics, and communicates them to the FPTUN kernel module through the loopback interface
  - FPTUN kernel module adds Fast Path statistics to Linux Networking Stack statistics
- **Linux applications are unchanged**
  - Linux applications read statistics as usual from the Linux kernel, which include kernel + Fast Path Statistics
  - Example: net-snmp used as-is with standard MIBs

Synchronization: Hitflags

- **When packets go through the Fast Path, the kernel object states are not updated**
  - Examples: ARP entries, conntracks, Linux bridge, …

- **The Fast Path Hitflags daemon updates hitflags into the Linux Networking Stack when packets hit the Fast Path**
  - Fast Path module updates Shared Memory with hitflags
  - Hitflags daemon reads Shared Memory entries (ARP for example), and communicates them to the FPTUN kernel module through the loopback interface
  - FPTUN kernel module updates states into the Linux Networking Stack

- **Linux applications are unchanged**
  - Linux applications can read states from the Linux kernel as usual

Thank You

6WIND.com

#SPEEDMATTERS For Serious Networks