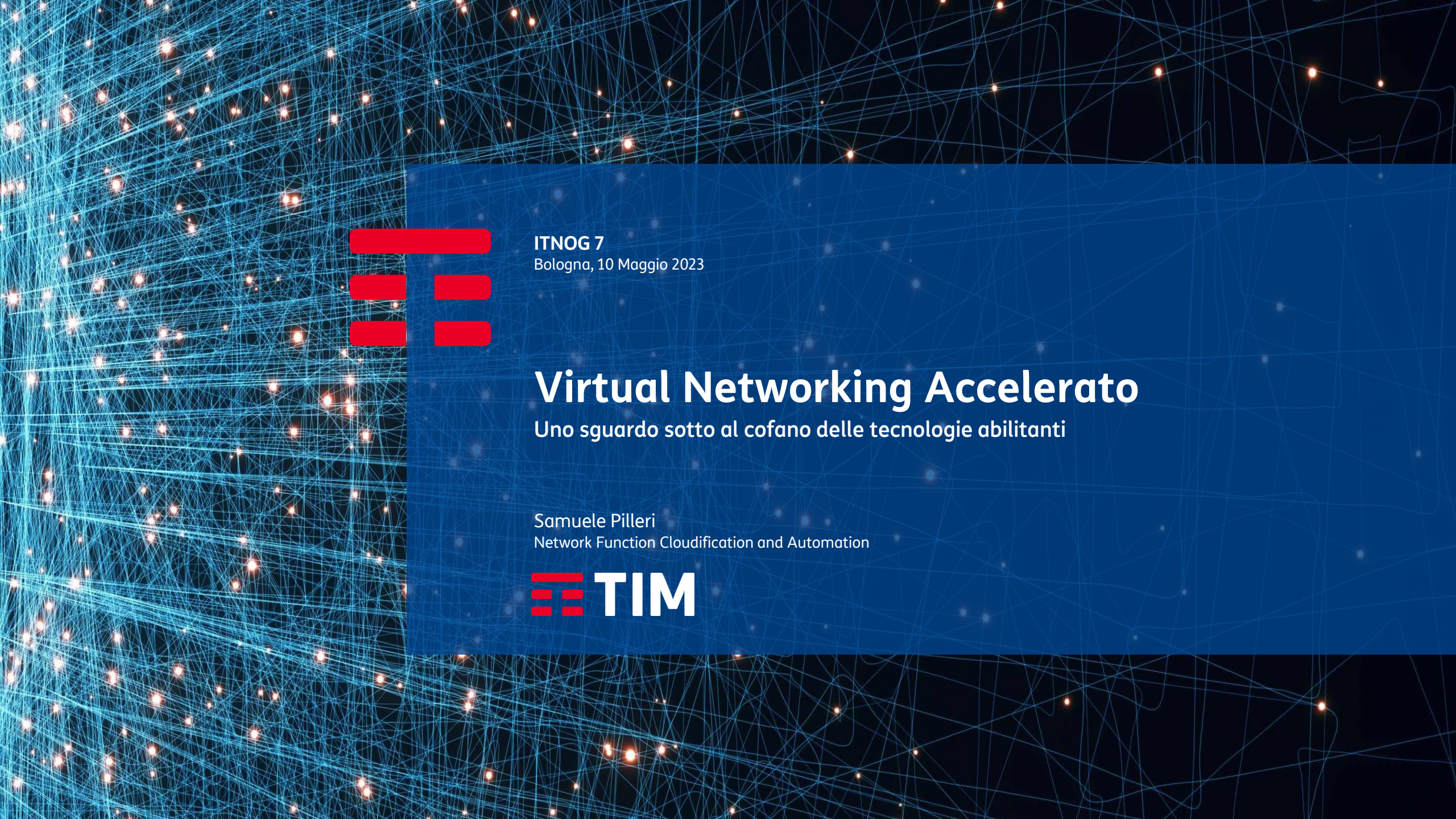


ITNOG 7
Bologna, 10 Maggio 2023

Virtual Networking Accelerato

Uno sguardo sotto al cofano delle tecnologie abilitanti

Samuele Pilleri
Network Function Cloudification and Automation

TIM

Agenda

Introduzione
Il data plane di Linux

Come i kernel tradizionali gestiscono l'I/O da dispositivi rete

eXpress Data Path
Un «bypass» per flussi speciali

Il linguaggio eBPF e XDP, seguiti da uno use case pratico di queste tecnologie nel contesto cloud native

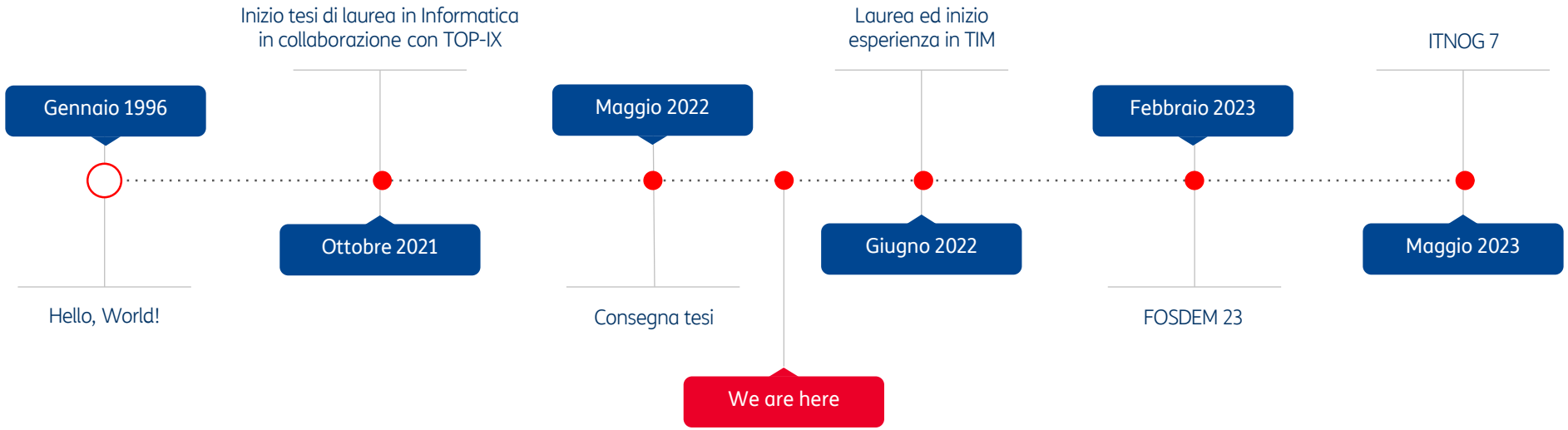
VPP/DPDK
La Ferrari del networking in software

Processing tradizionale vs vettoriale, driver in user space ed interazione tra le due tecnologie

Ottimizzazione
Ottenere il massimo dall'hardware a disposizione

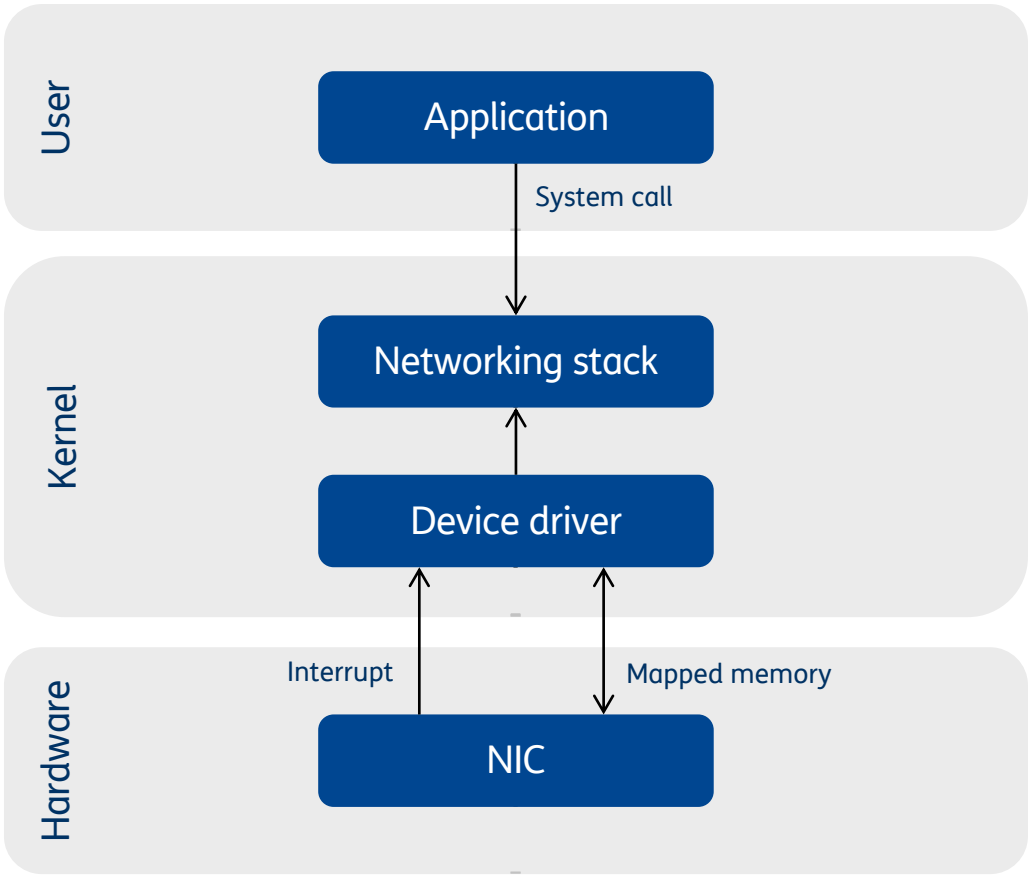
Conoscere i sistemi sottostanti è fondamentale per poter spremere ogni ciclo di clock: RSS, NUMA, e chi più ne ha più ne metta!

Timeline


Gennaio 1996
Hello, World!
Inizio tesi di laurea in Informatica in collaborazione con TOP-IX
Ottobre 2021
Maggio 2022
Consegna tesi
We are here
Laurea ed inizio esperienza in TIM
Giugno 2022
Febbraio 2023
FOSDEM 23
ITNOG 7
Maggio 2023

In principio fu Linux

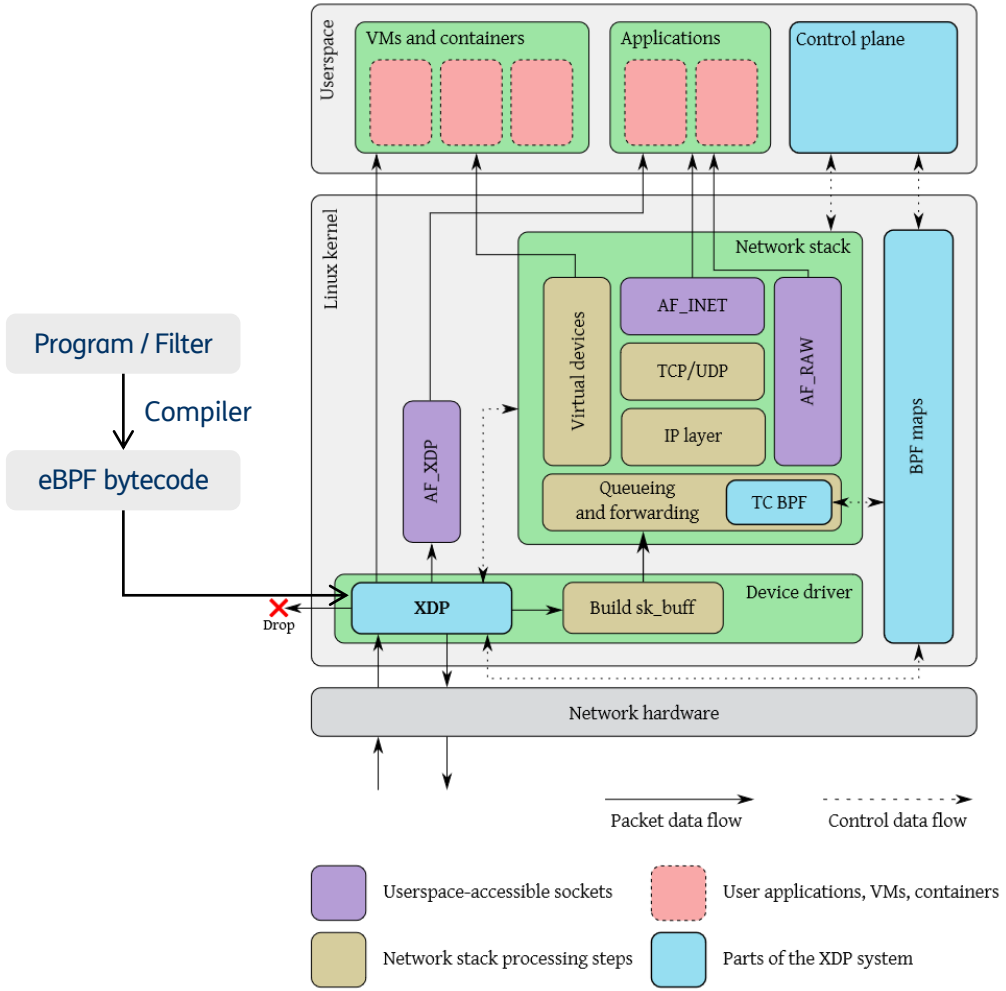
Anatomia dello stack di rete



- I sistemi operativi prevedono che dispositivi come le schede di rete segnalino la presenza di nuovi dati attraverso gli **interrupt**
- La NIC riceve i pacchetti, li salva in una zona della RAM condivisa col kernel e notifica il sistema operativo
- La gestione degli interrupt è particolarmente onerosa poiché stravolge il regime di funzionamento della CPU (**pipeline**)
- I pacchetti attraversano **uno ad uno** il data plane del kernel
- Se destinati ad un'applicazione, i dati vengono salvati in un buffer temporaneo in attesa di essere recuperati
- L'applicazione accede a questo buffer tramite le **system call**, anch'esse onerose per gli stessi motivi sopracitati
- In trasmissione il procedimento è analogo, con le frecce invertite

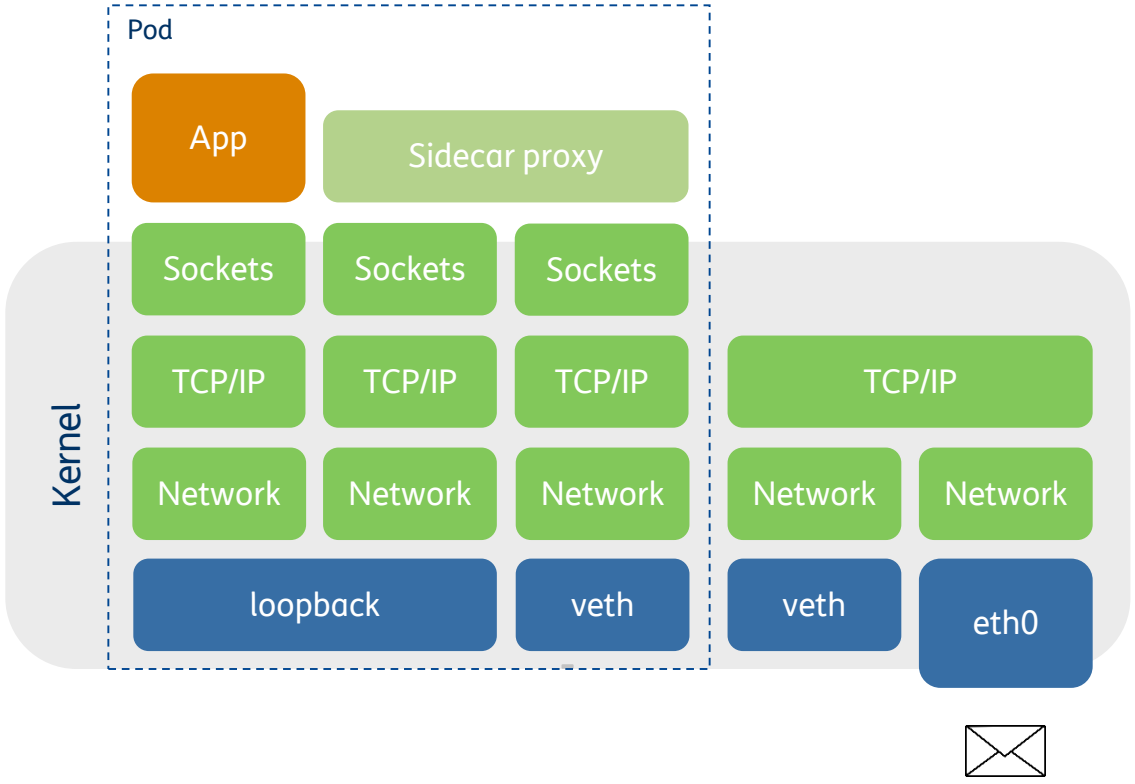
eXpress Data Path (XDP)

- Obiettivo: creare un **fast path** per velocizzare il processing di alcuni pacchetti
- Un **filtro** analizza i dati in arrivo prima che ne inizi il processing lato kernel e decide:
  - DROP: il pacchetto è da scartare
  - TX: il pacchetto va ritrasmesso sulla stessa interfaccia da cui è stato ricevuto, eventualmente dopo averlo modificato
  - REDIRECT: il pacchetto va spostato su un'altra interfaccia oppure inviato in user space tramite socket AF_XDP
  - PASS: prosegue lungo la pipeline normale
- Casi d'uso: firewalling (DDoS mitigation), Cloud NAT, k8s CNI, ...
- Il filtro non viene compilato direttamente in linguaggio macchina, ma in un bytecode intermedio (**eBPF**)
- Il programma così prodotto può essere **eseguito in sicurezza** all'interno del kernel oppure beneficiare dell'**offloading** sulle NIC compatibili (solo per alcune azioni)

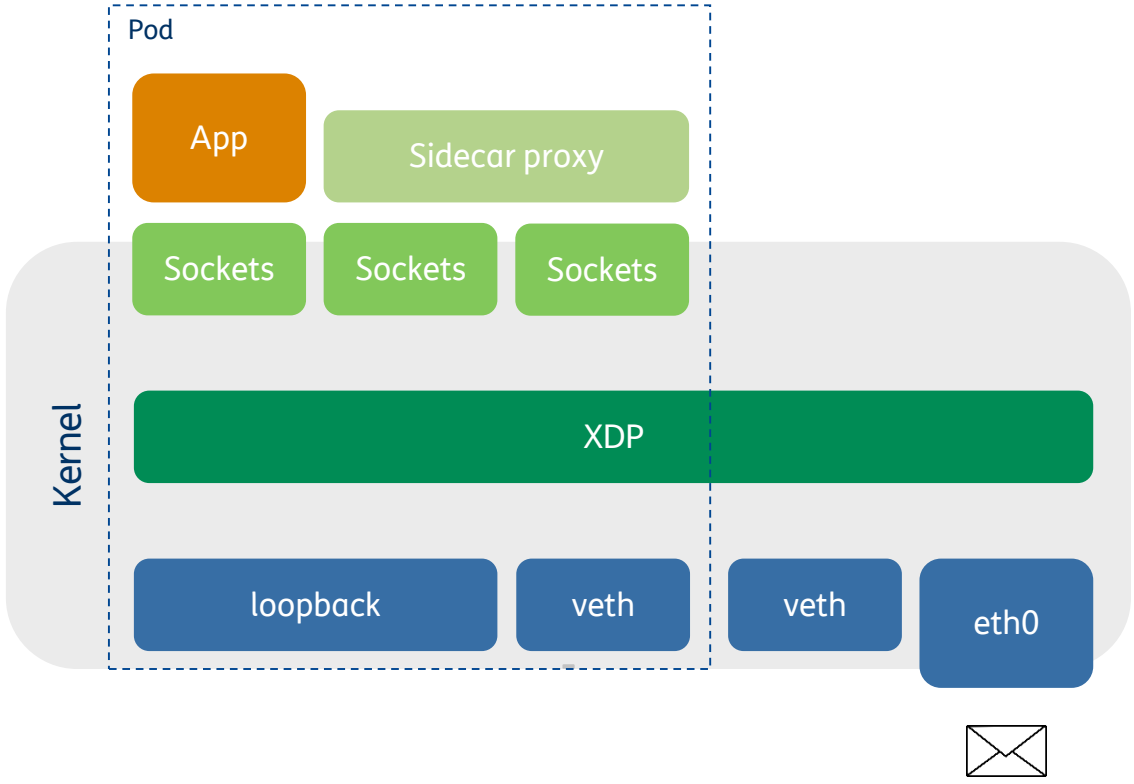


Fast-Forward: Kubernetes Service Mesh (Cilium)

Senza XDP



Con XDP



Punti ancora aperti

Superare il modello interrupt-driven Gli interrupt sono particolarmente onerosi da gestire perché vanificano il regime della pipeline della CPU ed impongono l'invalidazione di quasi tutte le cache hardware (effetto del context-switch)

Bypassare il kernel Lavorare a stretto contatto col kernel di un sistema operativo è complicato: occorre inserirsi in un ambiente preesistente molto articolato dove un eventuale difetto (bug) è potenzialmente in grado di causare il crash dell'intero sistema

Processing vettoriale Elaborare i pacchetti uno ad uno è inefficiente in termini di accessi alla memoria, i data plane ASIC operano su blocchi di pacchetti per migliorare parallelismo e throughput end-to-end

Virtualizzazione La soluzione dev'essere facilmente inseribile in un contesto cloud (macchine virtuali e container), garantendo un buon livello di disaccoppiamento

Cenni dell'architettura MIPS

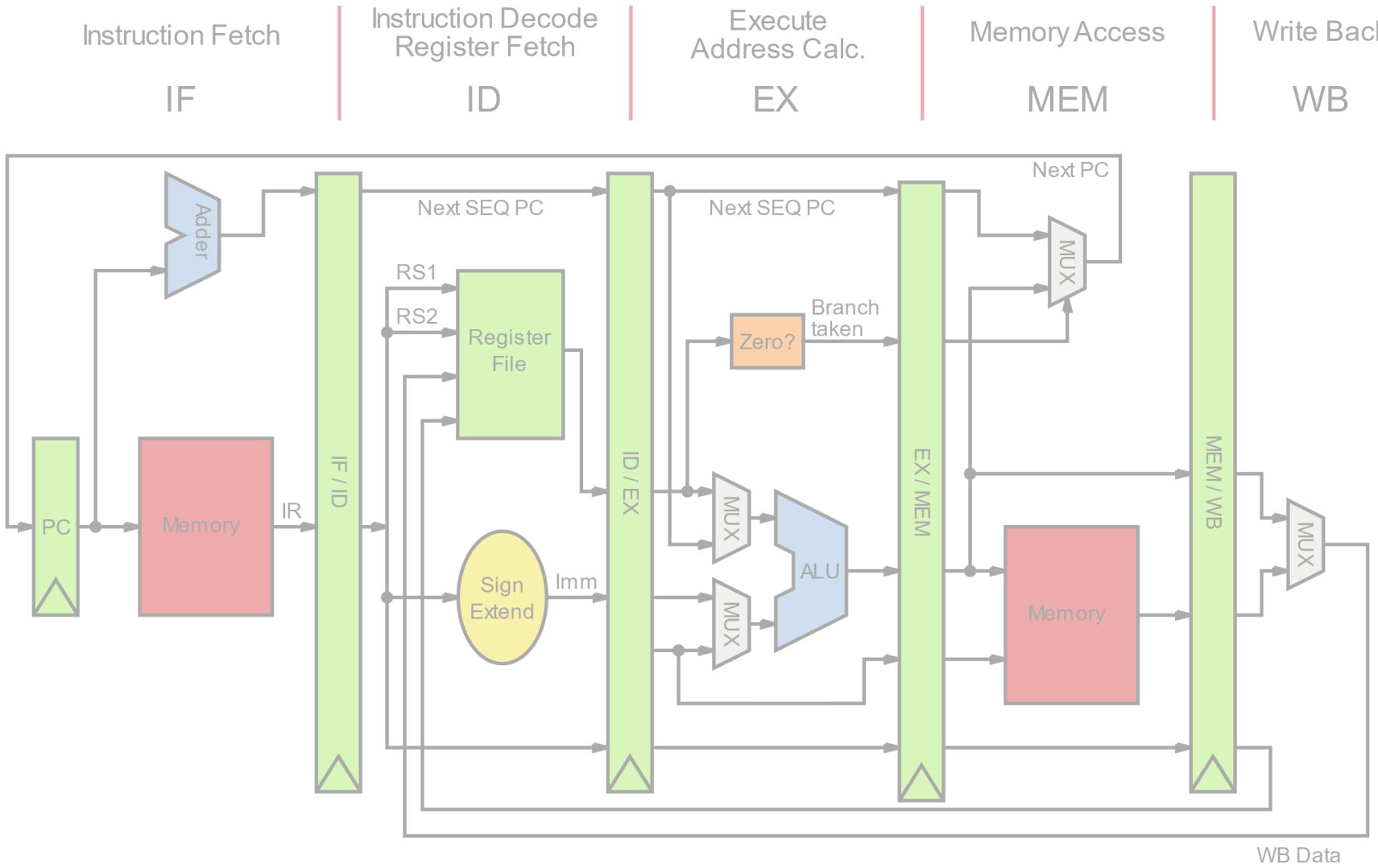
Istr 5

Istr 4

Istr 3

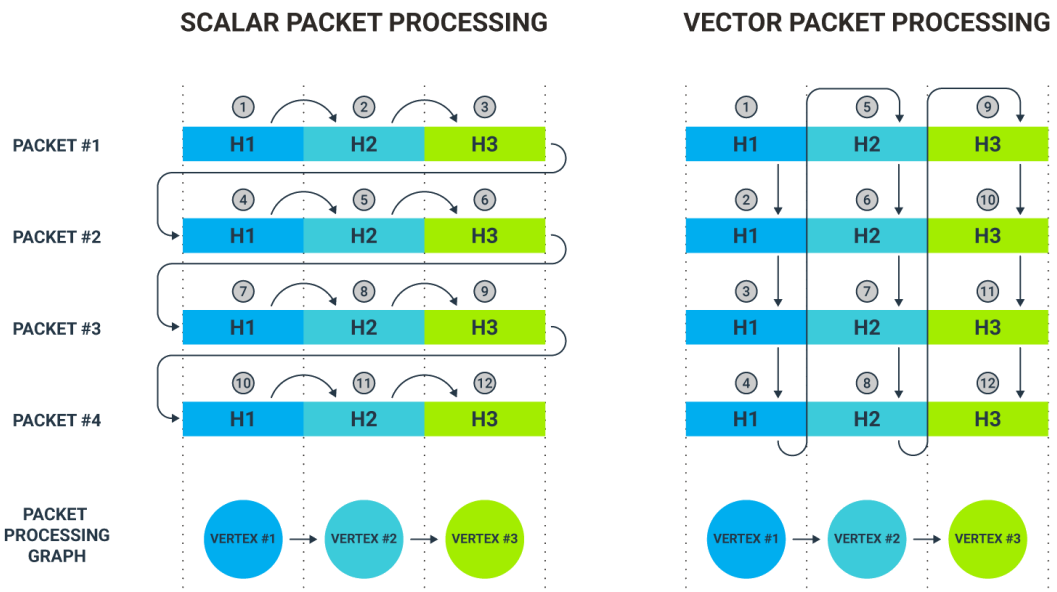
Istr 2

Istr 1



Come tenere il motore a regime??

Cosa vuol dire «Vector Packet Processing»?

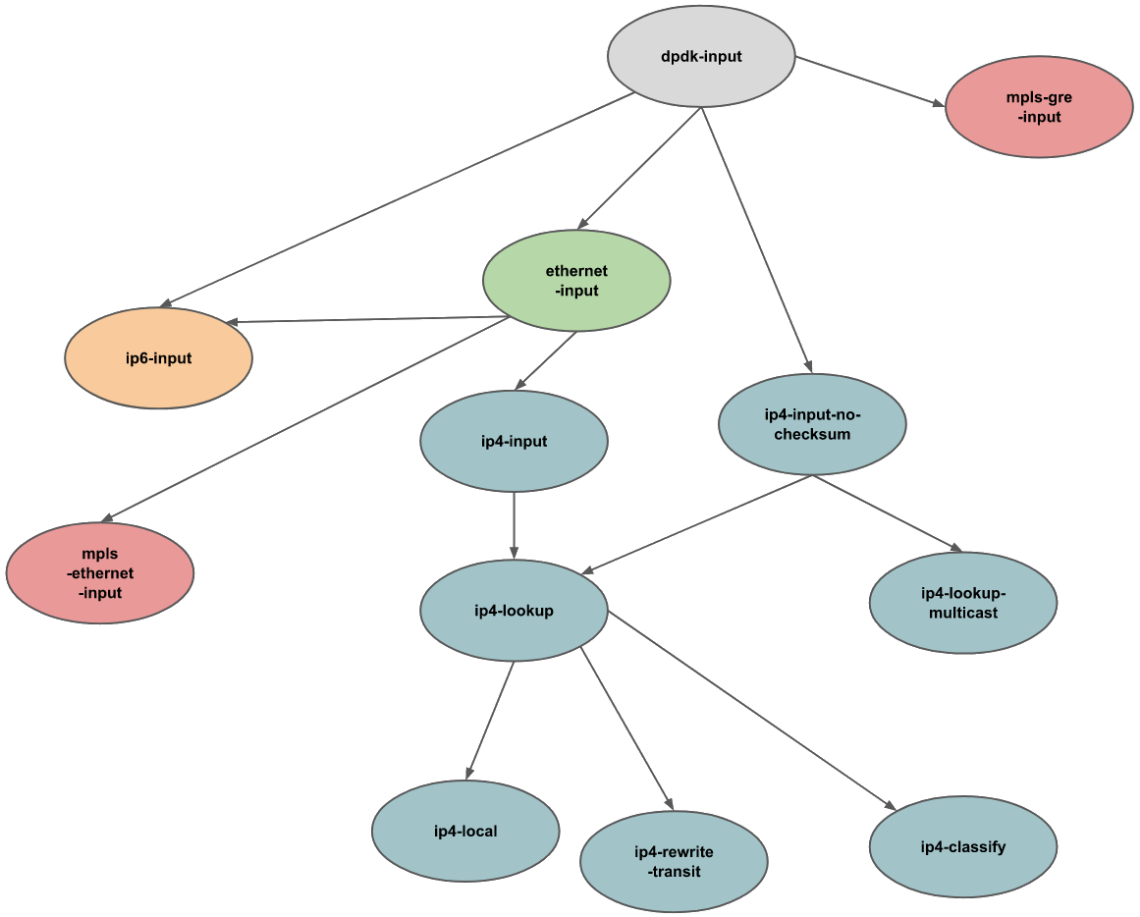


- Lo scopo è **ottimizzare** l'utilizzo ed in particolare **gli accessi alla cache** interna alla CPU
- Nell'approccio scalare classico gli header dei pacchetti sono esaminati in maniera sequenziale
- Così facendo la **data cache è sottoutilizzata** perché contiene al più un pacchetto
- Al contrario, la **instruction cache** vede il **susseguirsi di funzioni** che vengono richiamate una volta sola per poi lasciare spazio alle successive
- In caso di traffico eterogeneo la differenza è poco apprezzabile
- Il processing vettoriale invece punta a riempire la **data cache** con **dati dello stesso tipo**, tra loro **indipendenti** e **ben allineati**
- Le funzioni caricate nella **instruction cache** possono essere **riutilizzate centinaia di volte** su altrettanti pacchetti prima di essere swappate

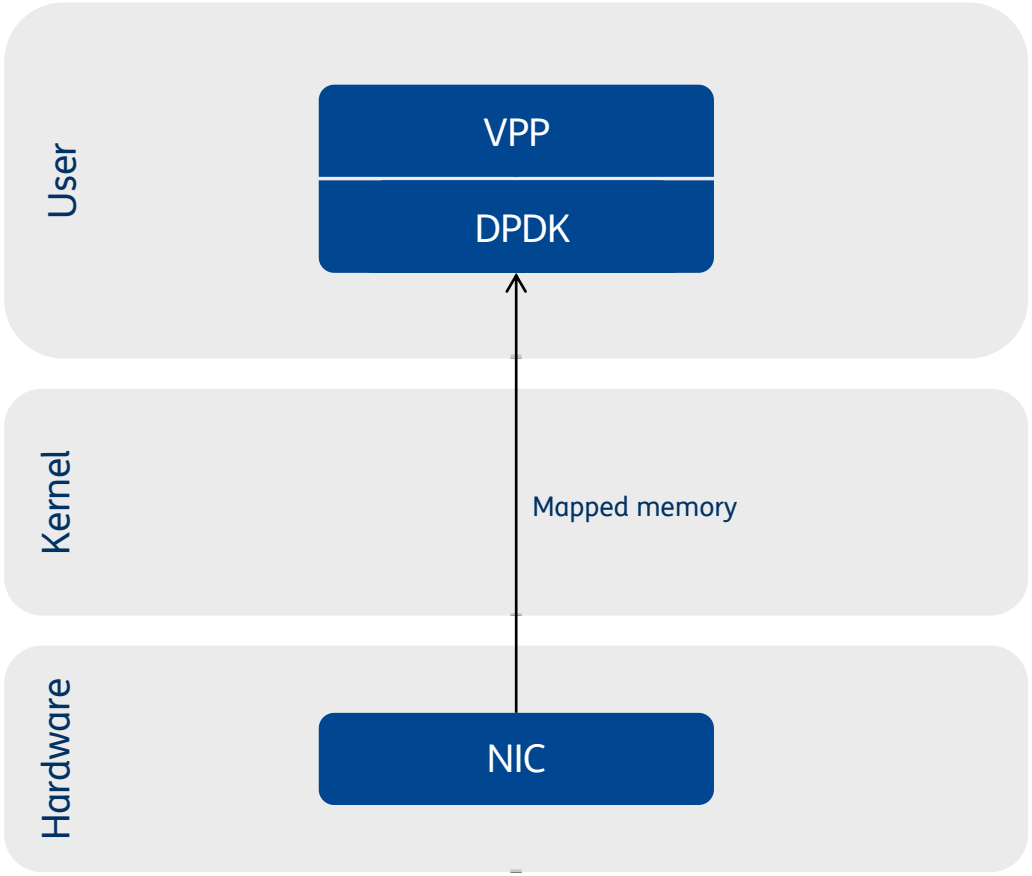
VPP (aka FD.io)

- Le funzionalità di VPP sono rappresentate da un **grafo diretto aciclico** (dag) formato da diversi nodi, detti plugin
- Ciascun pacchetto attraversa un sottoinsieme (lista) di stazioni di elaborazione: la sua **pipeline di processing**
- Il punto di forza di VPP è lavorare su **blocchi di pacchetti dello stesso tipo** (da cui Vector Packet Processing)
- Questa tecnica è molto versatile: è possibile realizzare router, gateway, firewall, ...
- A regime si raggiungono prestazioni nell'ordine di **diversi milioni di pacchetti al secondo** elaborati **per core**
- DPDK si occupa di interfacciarsi con la scheda di rete
- I pacchetti ricevuti da DPDK costituiscono l'**input** di VPP

Grafo di processing dei pacchetti (esempio)

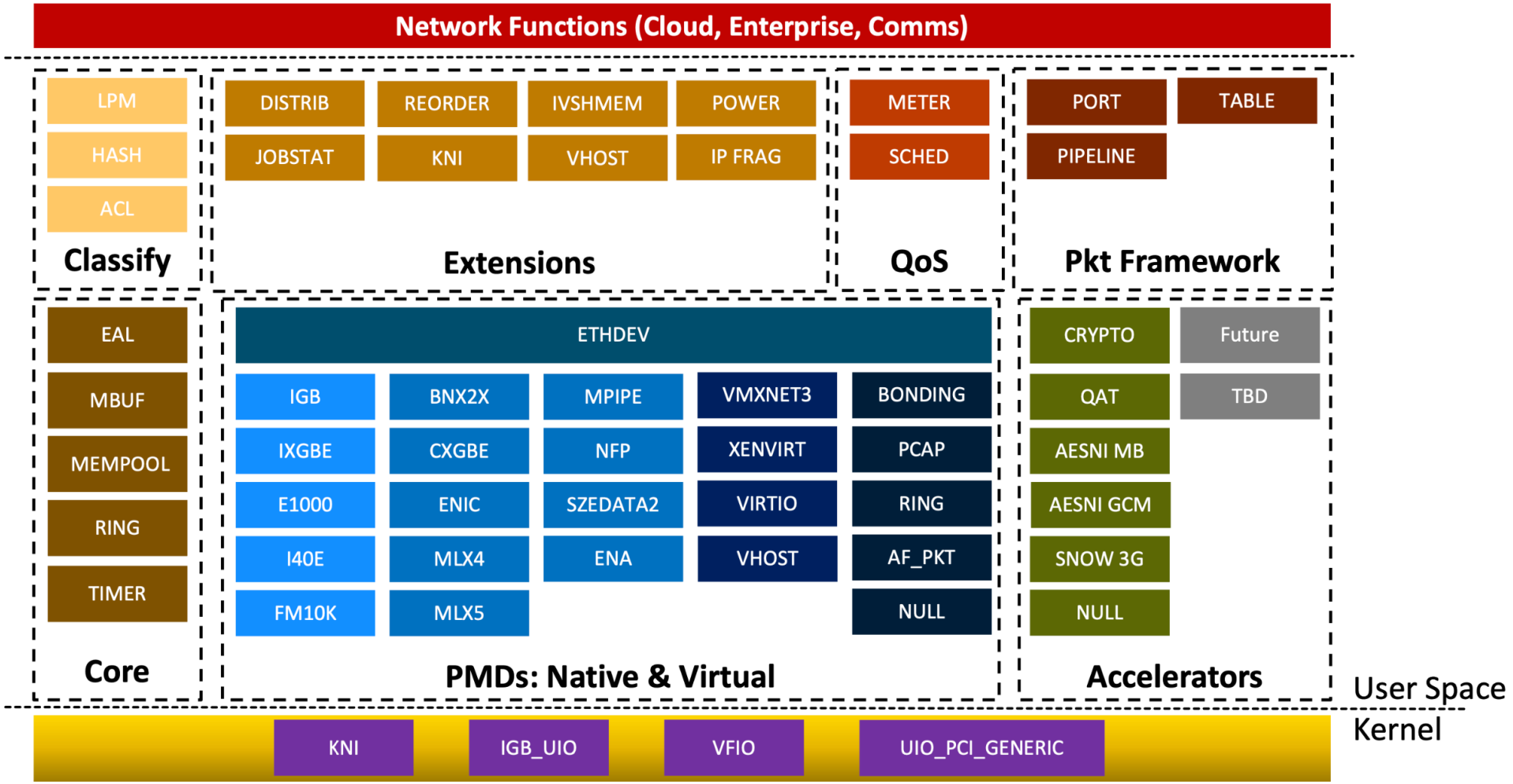


VPP/DPDK



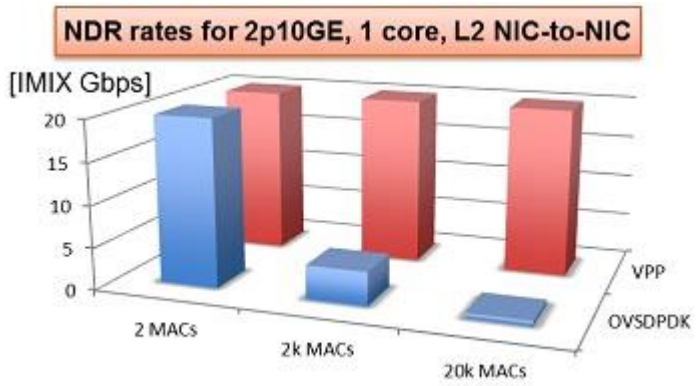
- Premessa: i dispositivi di rete non sono fatti per stare a riposo, sulle interfacce di una VNF/CNF vi sarà **sempre traffico**
- Segue logicamente che gli **interrupt non sono necessari** (perché notificare la presenza di dati che ci si aspetta arrivino comunque?)
- Introduzione del **poll-mode driver** (pmd)
- Spostare i buffer della NIC in **user space**, direttamente all'interno dell'applicazione: il data plane, a differenza dei programmi applicativi, è unico
- Gestire le particolarità dell'**interfacciamento con dispositivi diversi**
- Tutto questo è DPDK (Data Plane Development Kit)
- VPP si occupa di fornire l'ossatura per elaborare vettori di dati alla volta (Vector Packet Processing)

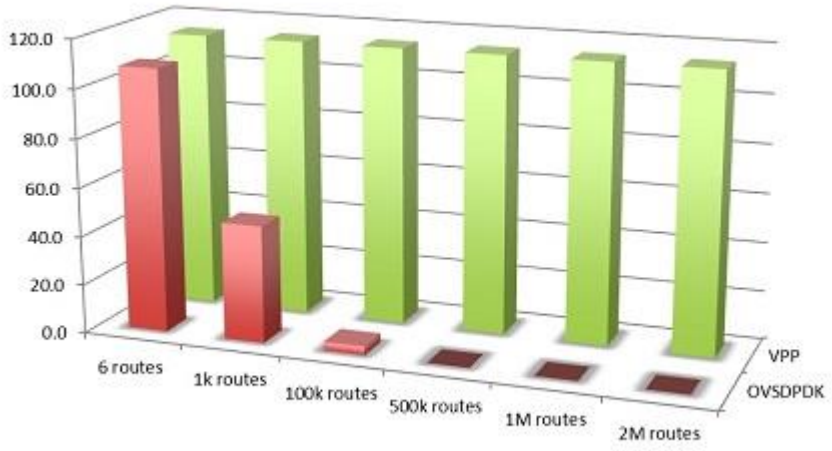
Architettura di DPDK



VPP/DPDK: Scaling

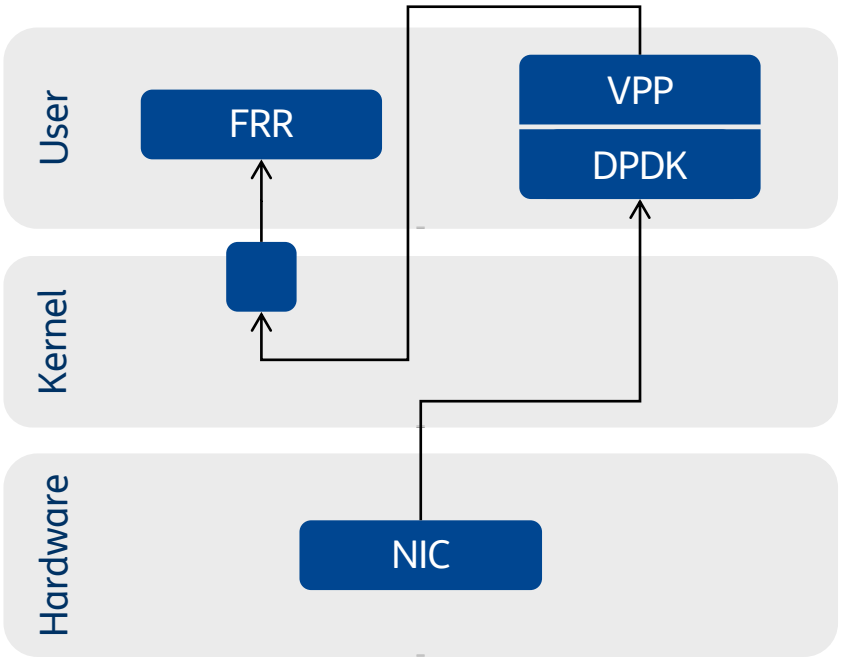
- **DPDK** è un framework che permette di **interfacciarsi con schede di rete** ad alte prestazioni bypassando diversi colli di bottiglia
- **VPP è un dataplane** estensibile che sfrutta il processing vettoriale per migliorare ulteriormente le performance e **si appoggia a DPDK** per l'integrazione con le NIC (fisiche o virtuali)
- Prestazioni nell'ordine di **x10 Gbps per core fisico** (traffico IMIX)
- Esistono applicazioni come T-rex (un traffic generator) che fanno uso solo di DPDK per l'accelerazione non necessitando di un dataplane completo
- Vi sono altresì soluzioni come OVS-DPDK che usano unicamente DPDK per implementare un sottoinsieme delle funzionalità di VPP
- In generale, **VPP rende DPDK scalabile!**



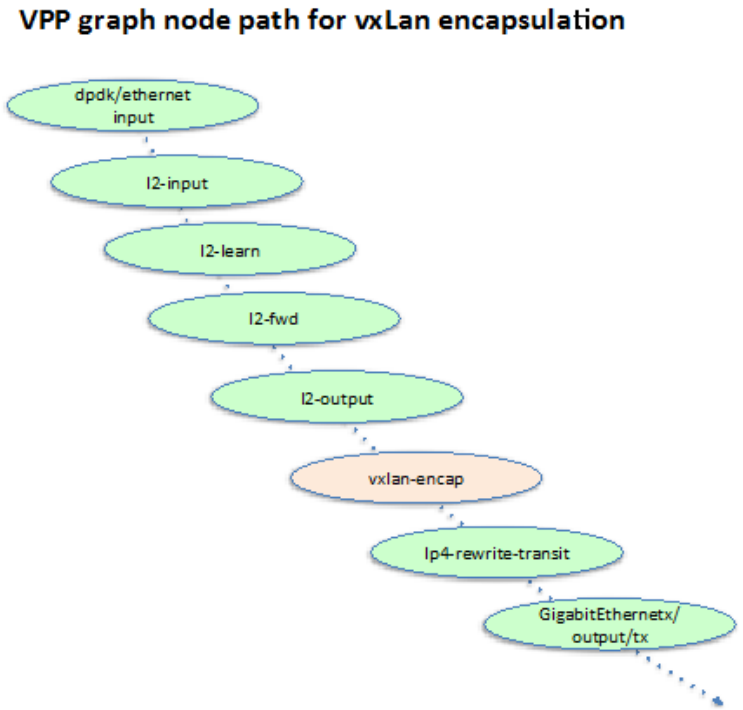


Use case end-to-end: EVPN/VxLAN con FRR e VPP/DPDK

- Quanto visto sinora è sufficiente per comprendere un caso pratico molto caro al mondo cloud: gli overlay VxLAN realizzati interamente in software



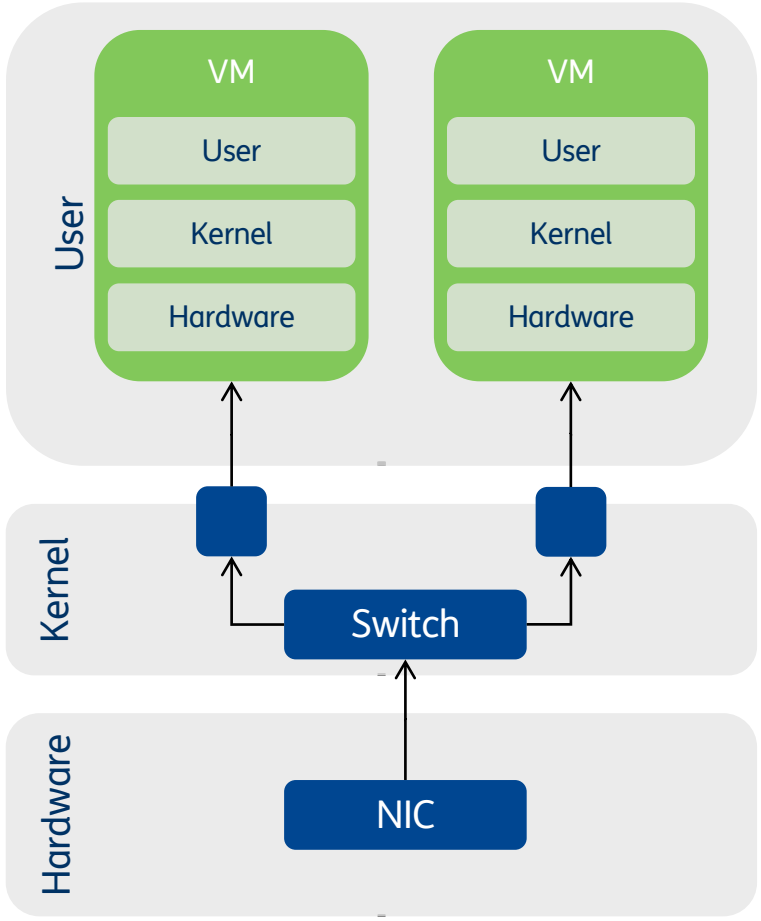
Grafo di elaborazione di VxLAN in VPP



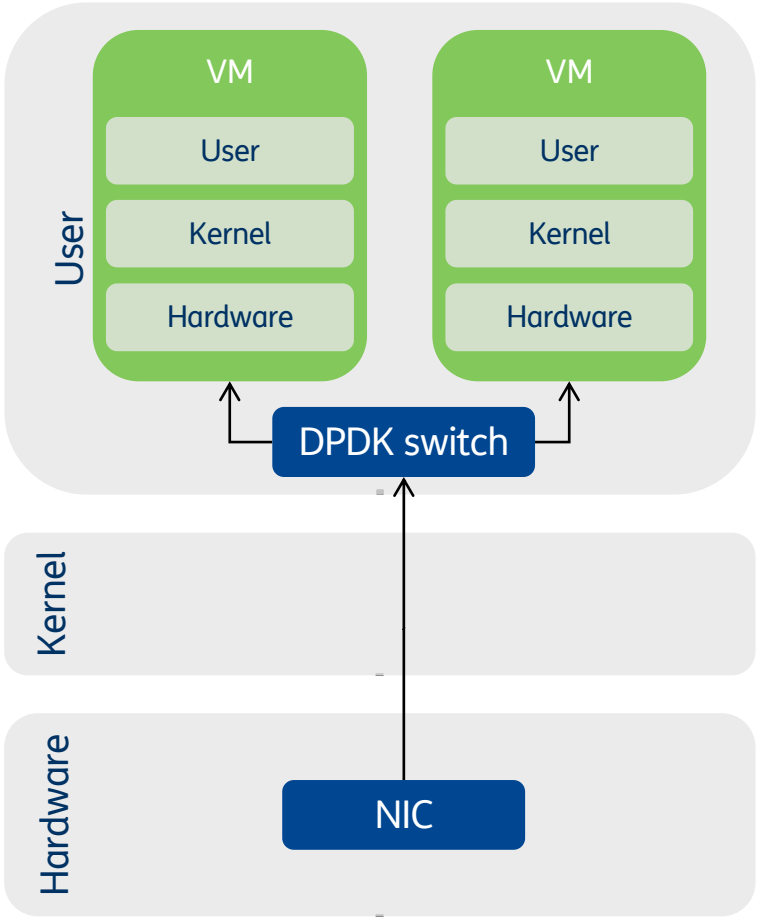
Ottimizzazione

Switch virtuali per tutti i gusti

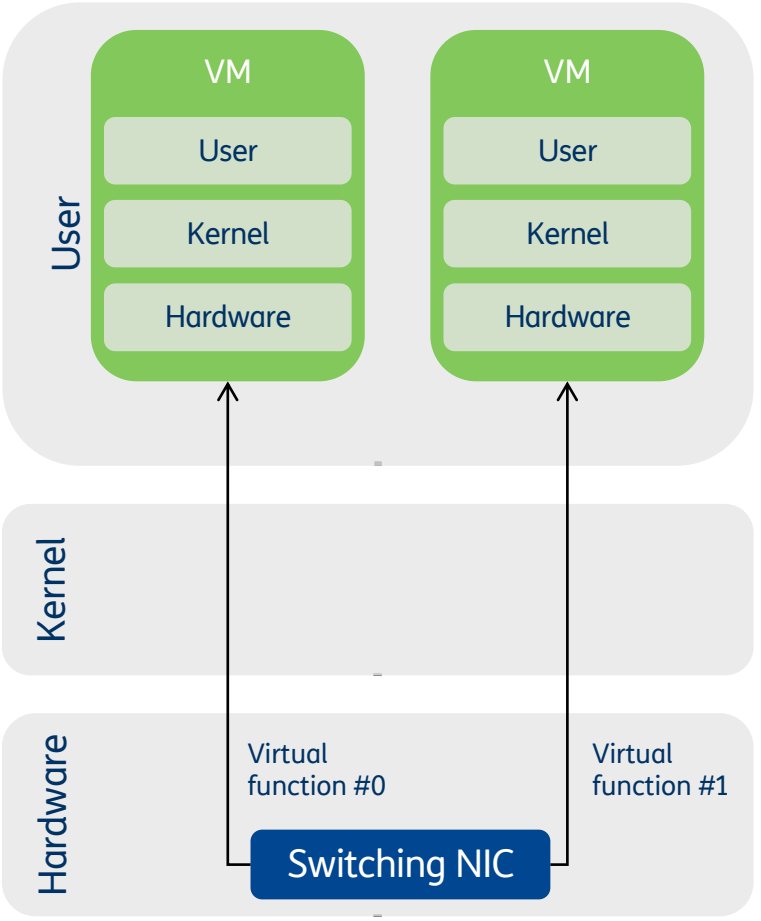
Bridge classico



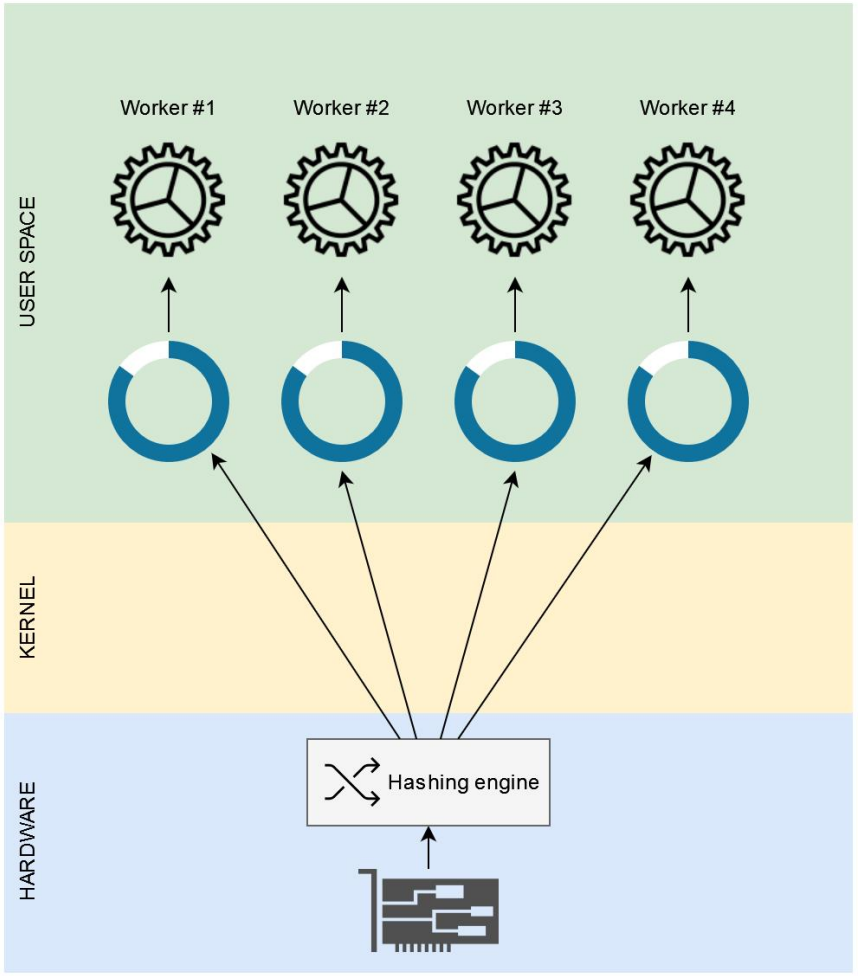
OVS–DPDK



SR–IOV



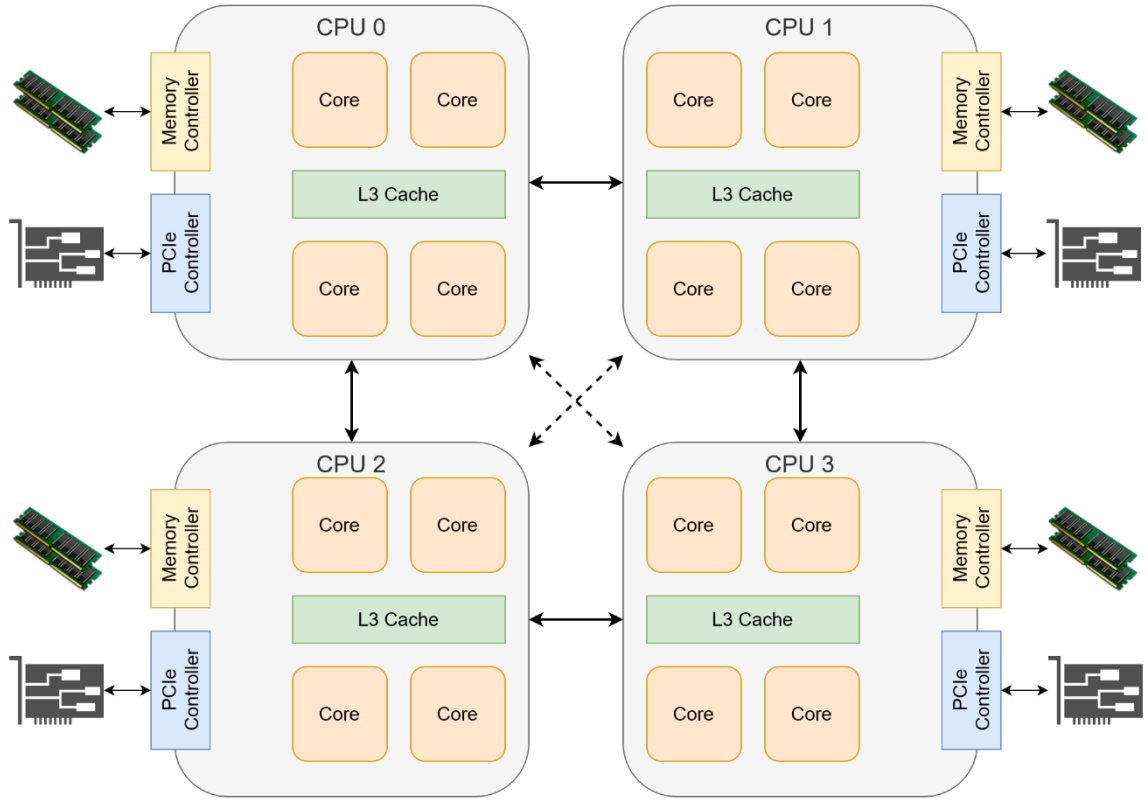
Receive Side Scaling



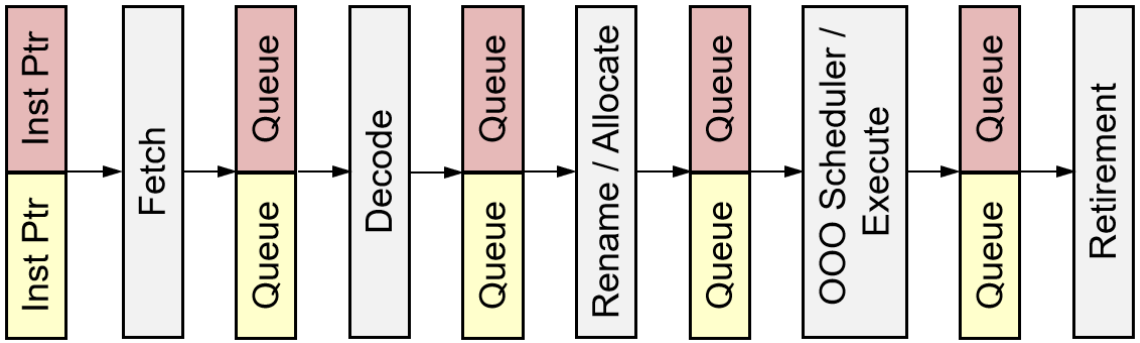
- Nessuna CPU, per quanto rapida, è in grado di scalare all'infinito
- Occorre un sistema efficace per **ripartire il traffico tra più code**, permettendone il **processing in parallelo**
- La ripartizione andrebbe effettuata mantenendo l'**integrità dei flussi** end-to-end per limitare il reordering
- Soluzione simile al funzionamento dei port channel: hashing di alcuni campi degli header L3 e L4 per identificare il flusso e riversarlo su una specifica coda
- **Non garantisce equa ripartizione**, ma adatto per flussi variegati
- Semplice da implementare in hardware
- Le schede più avanzate sono microprogrammabili, permettendo di customizzare l'algoritmo di hashing ad esempio per offrire diffserv

NUMA

- I server più potenti sono dotati di molteplici processori
- A ciascuna CPU sono collegati memoria e dispositivi
- L'accesso alla memoria di una CPU diversa dalla propria **comporta un costo aggiuntivo**
- Il bus di interconnessione si potrebbe saturare
- Occorre prevedere l'allocazione dei workload (CPU e memoria) vicino ai dispositivi che effettuano I/O



Simultaneous Multi-Threading



- Commercializzato da Intel col nome **Hyper-Threading**
- Permette di eseguire contemporaneamente istruzioni di thread diversi su uno stesso core fisico
- Le **risorse della pipeline**, quali cache, registri ed unità funzionali, sono **ripartite** tra i thread
- VPP performa tanto meglio quanti più pacchetti riesce ad elaborare in blocco, quindi le **prestazioni sono strettamente legate alla** disponibilità di **cache** sul core
- **SMT inficia pesantemente le performance di VPP!**
- Generalmente viene disabilitato, ma è possibile ottenere lo stesso effetto con una combinazione di core isolation e CPU pinning

...e molto di più!

- Hugepages & IOMMUs
- CPU & Memory pinning
- Core isolation
- «Seamless DPDK» (virtio-net-pmd)
- GPSS Timer
- ...

- Tutte queste tecniche di ottimizzazione si applicano sia in ambito virtualizzazione, dove spesso **l'effort va duplicato** nell'hypervisor e nella VM, sia in contesti cloud native
- Kubernetes è estendibile con tanti «**CNI plugins**» che consentono di implementare modelli di virtual networking accelerato senza dover sacrificare parte delle performance all'overhead delle VM

Grazie

samuele.pilleri@telecomitalia.it

TIM

Bibliografia

{ Samuele Pilleri, **«Virtualizzazione di Funzioni di Rete»**, giugno 2022, https://u.garr.it/jdO6D }.foreach

Raymond de Jong, **«Service MESH without the MESS**, *Latest of eBPF Powered Cilium Service Mesh*»,
FOSDEM23 (Network dev. room), Bruxelles, 5 febbraio 2023