

Open-NFP Summer Webinar Series: Session 4: P4, EBPF And Linux TC Offload

Dinan Gunawardena & Jakub Kicinski -
Netronome

August 24, 2016

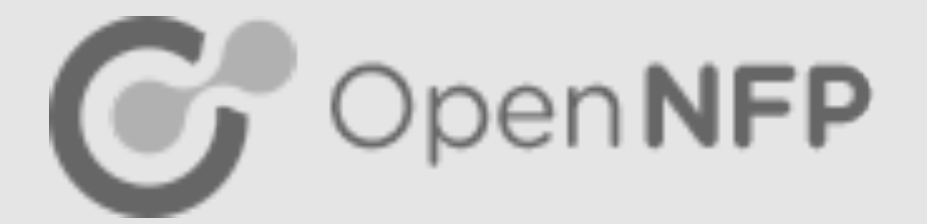
Support and grow reusable research in accelerating dataplane network functions processing

Reduce/eliminate the cost and technology barriers to research in this space

- Technologies:
 - P4, eBPF, SDN, OpenFlow, Open vSwitch (OVS) offload
- Tools:
 - Discounted hardware, development tools, software, cloud access
- Community:
 - Website (www.open-nfp.org): learning & training materials, active Google group <https://groups.google.com/d/forum/open-nfp>, open project descriptions, code repository
- Learning/Education/Research support:
 - Summer seminar series, P4DevCon conference, Tutorials (P4 Developer Day), research proposal support for proposals to the NSF, state agencies

Summer seminar series to further progress to our objective. Present applied reusable research.

P4DevCon Attendees/Open-NFP Projects*

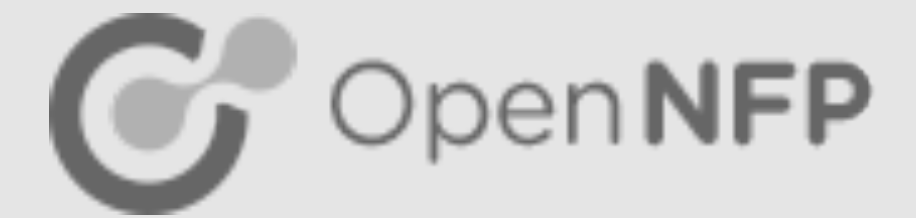


Universities

Companies

*This does not imply that these organizations endorse Open-NFP or Netronome

Session Agenda



1. Introduction
 - Objectives
2. Overview: The high level model for offload
 - What we are offloading – P4 / eBPF
 - Overall programmer model for transparent offload
3. Linux Kernel Infrastructure
 - The Traffic Classifier (TC)
 - eXpress Data Path (XDP)
 - Current eBPF translation on X86/ARM64/PPC64
4. Hardware Intro to NFP (Network Flow Processor) architecture
 - SmartNICs-Multi Core, Many Core
 - NUMA, Memory Hierarchy
5. Accelerating P4/eBPF in NFP : Implementation
 - Kernel core infrastructure
 - Map handling in the kernel
 - Translating instructions
 - Basic Map Support
 - Optimizations
6. & 7. Demo; Summary

A hand-drawn black arrow pointing to the right.

Session Objectives

Understanding how eBPF is relevant to P4

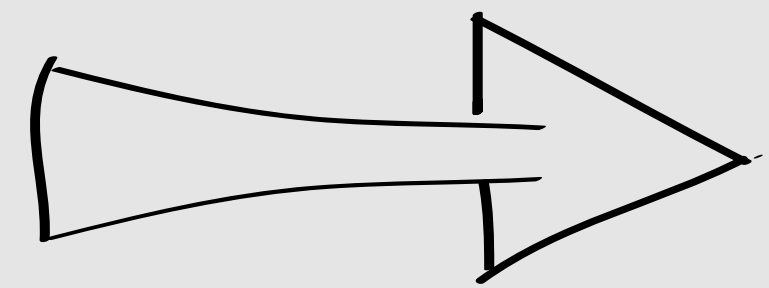
Understanding the support for offload and state of art in the Linux Kernel

The Code

- Understand the structure of a eBPF program
- Gain an insight into how this is translated and executed in hardware

Understanding how the NFP architecture on the Agilio CX enables high performing, fully programmable network offload

- The Many Core architecture and its advantages



Overview: High level model for offload

- What we are offloading – P4 / eBPF
- Overall programmer model for transparent offload

What are P4 and eBPF?

Domain specific languages for specifying forwarding behaviour of the data plane of network components

P4 - Programming Protocol-Independent Packet Processors

- Header format description
- Parse Graphs (FSM)
- Tables (<keys,actions>)
- Actions manipulate packet header/metadata
- Control flow – an imperative program, describing sequence of data dependent match/actions

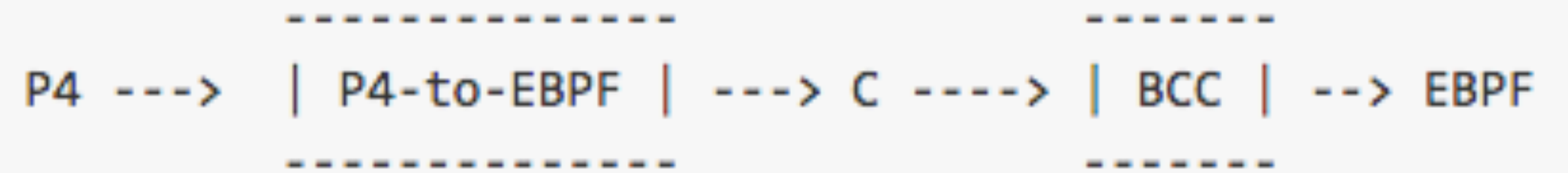
eBPF – Extended Berkley Packet Filters

- Low level (machine code like) language
- Code executed by a VM (restricted memory, no sleeps/locks, limited API to kernel)in the Kernel (TC)
- Code injected into netfilter hook points in kernel data plane
- Maps (<key, value> stores)
- Chained filter functions
- Match/action
- Static verification of safety, guaranteed to terminate

Translating P4->eBPF

John Fastabend P4 to eBPF compiler

Why translate P4 to eBPF?



P4 Construct	C Translation
header_type	struct type
header	struct instance with an additional valid bit
metadata	struct instance
parser state	code block
state transition	goto statement
extract	load/shift/mask data from packet buffer

P4 Construct	C Translation
table	2 EBPF tables: second one used just for the default action
table key	struct type
table actions block	tagged union with all possible actions
action arguments	struct
table reads	EBPF table access
action body	code block
table apply	switch statement
counters	additional EBPF table

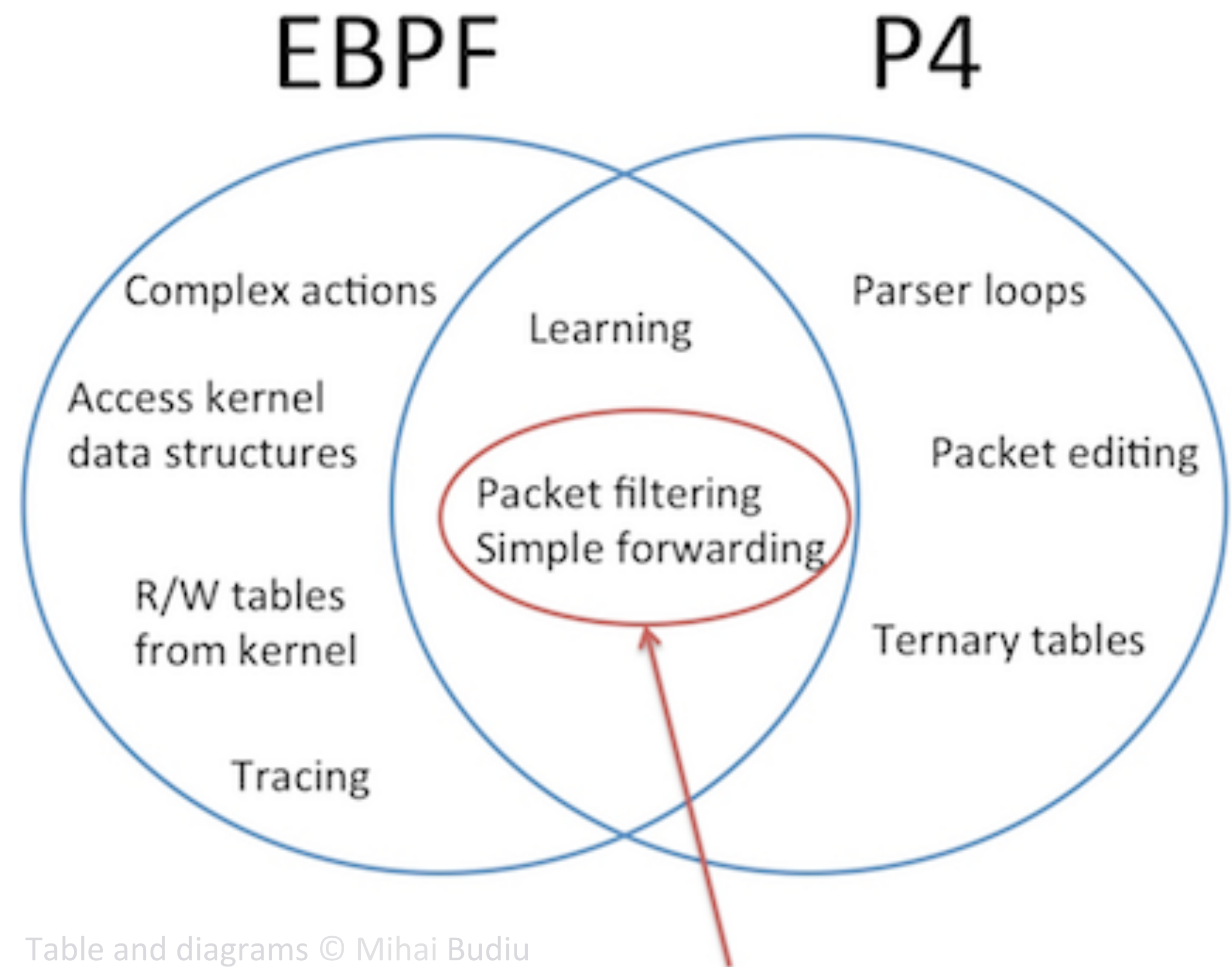


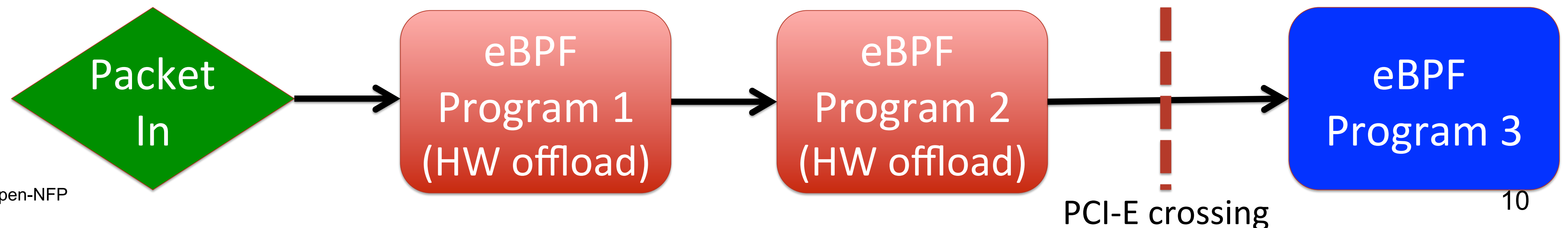
Table and diagrams © Mihai Budiu

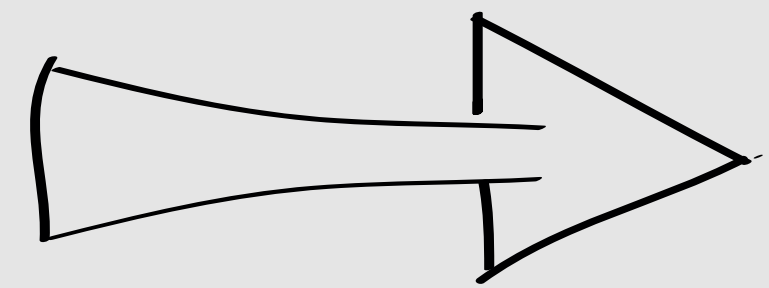
Scope of current compiler

Programmer / user is “unaware” that eBPF code is “offloaded”

Requirements

- Partial pipeline offload
- Fallback to software for any eBPF code block
- Transparent user mode / kernel mode access to tables





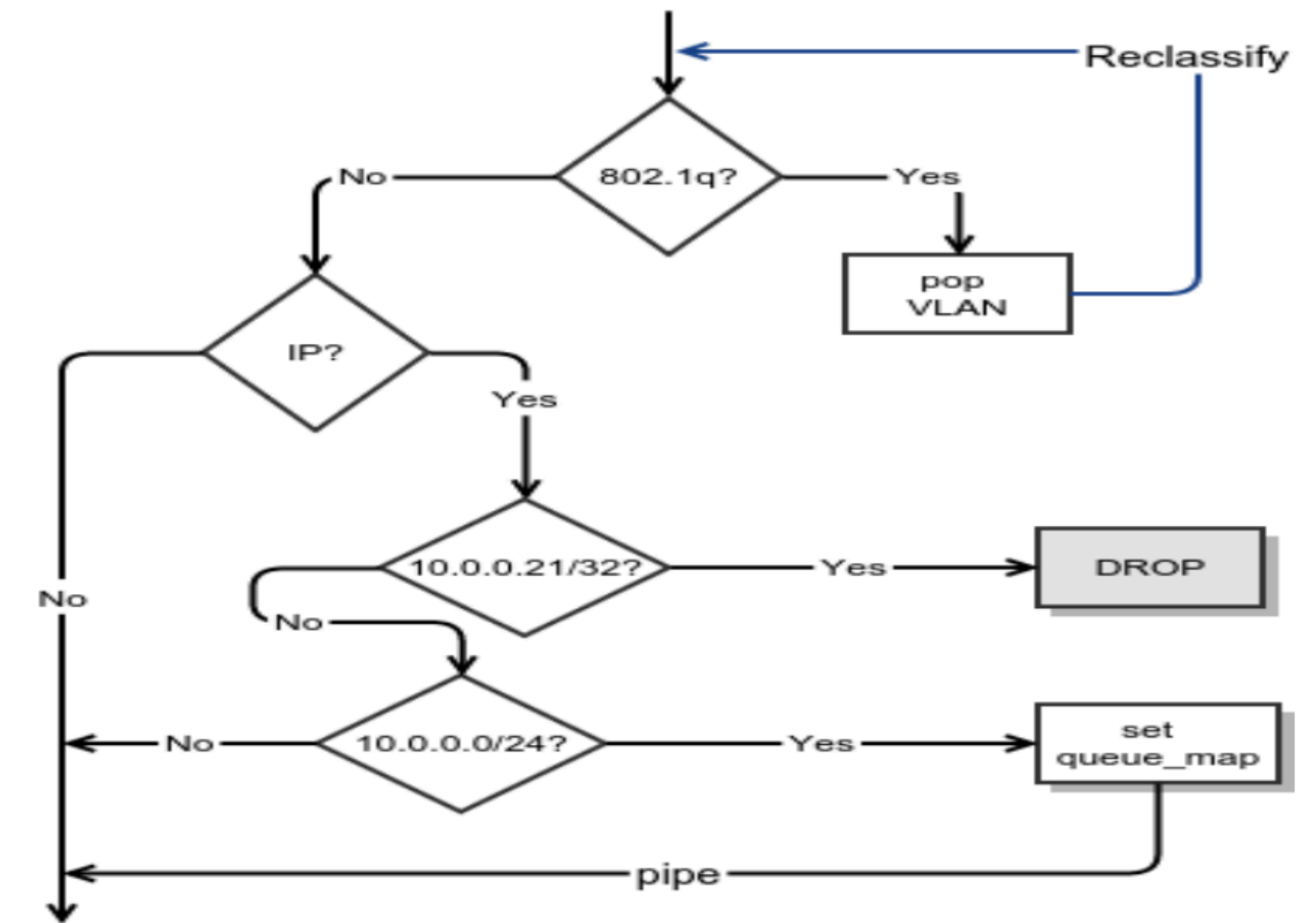
Linux Kernel Infrastructure

- The Traffic Classifier (TC)
- eXpress Data Path (XDP)

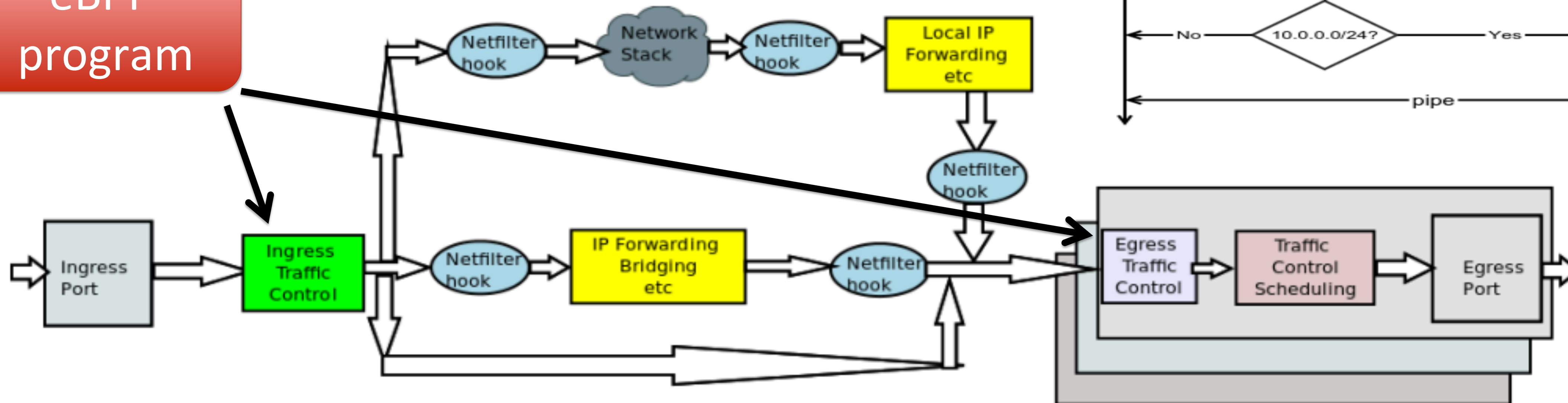
Linux Traffic Classifier (TC)

Component	Linux Component
Shaping	The class offers shaping capabilities
Scheduling	A qdisc acts as a scheduler e.g. FIFO
Classifying	The filter performs classification through a classifier object.
Policing	A policer performs policing within a filter
Dropping	The "drop" action occurs with a filter+policer
Marking	The dsmark qdisc is used for marking

A Simple Program



eBPF program



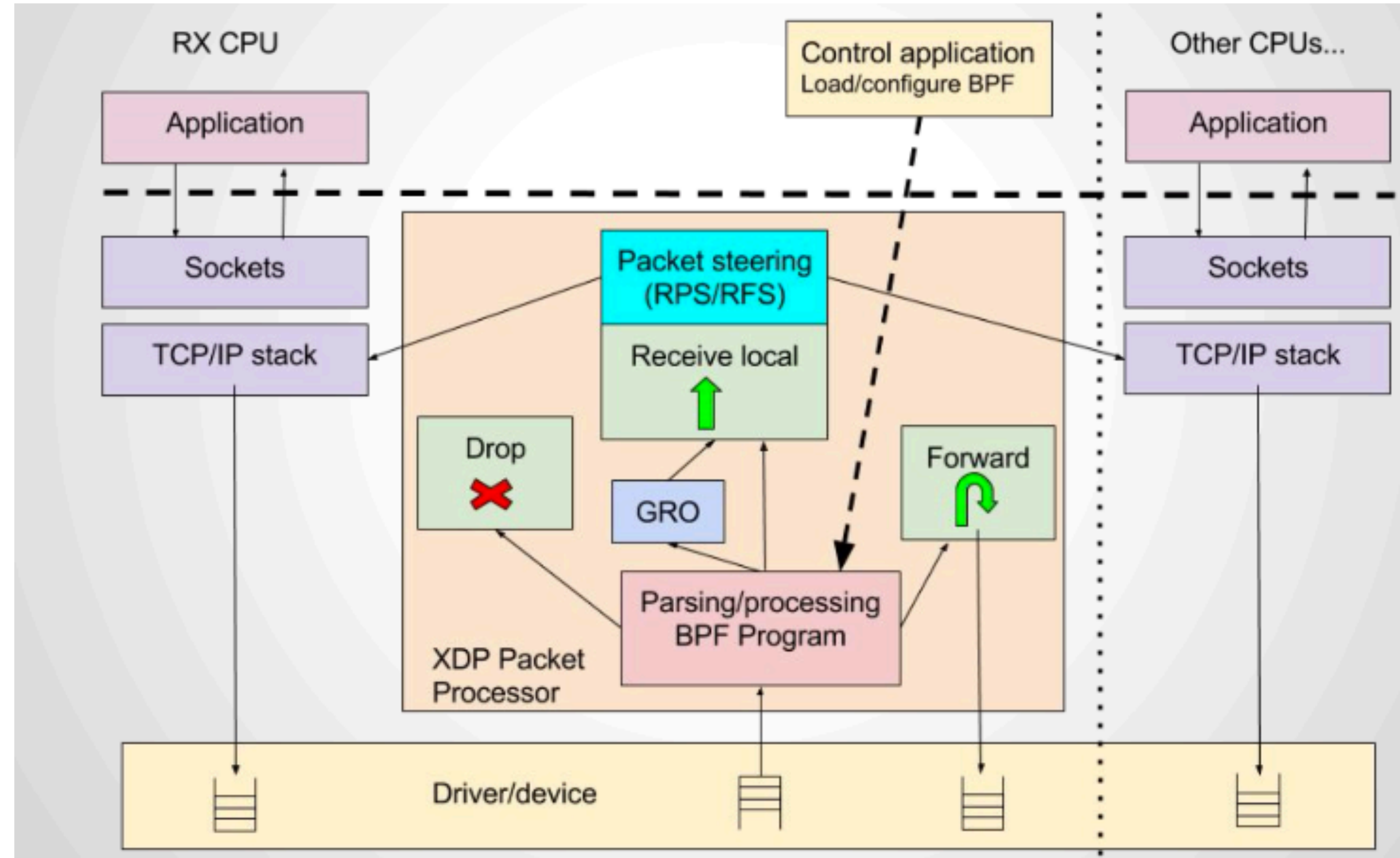
eXpress Data Path (XDP)

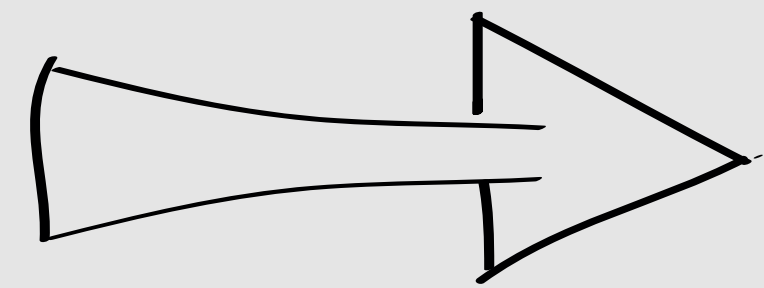
What

- High performance, programmable network data path

Utility

- Useful for packet processing
- forwarding
- load balancing
- DOS mitigation
- firewalls, etc.

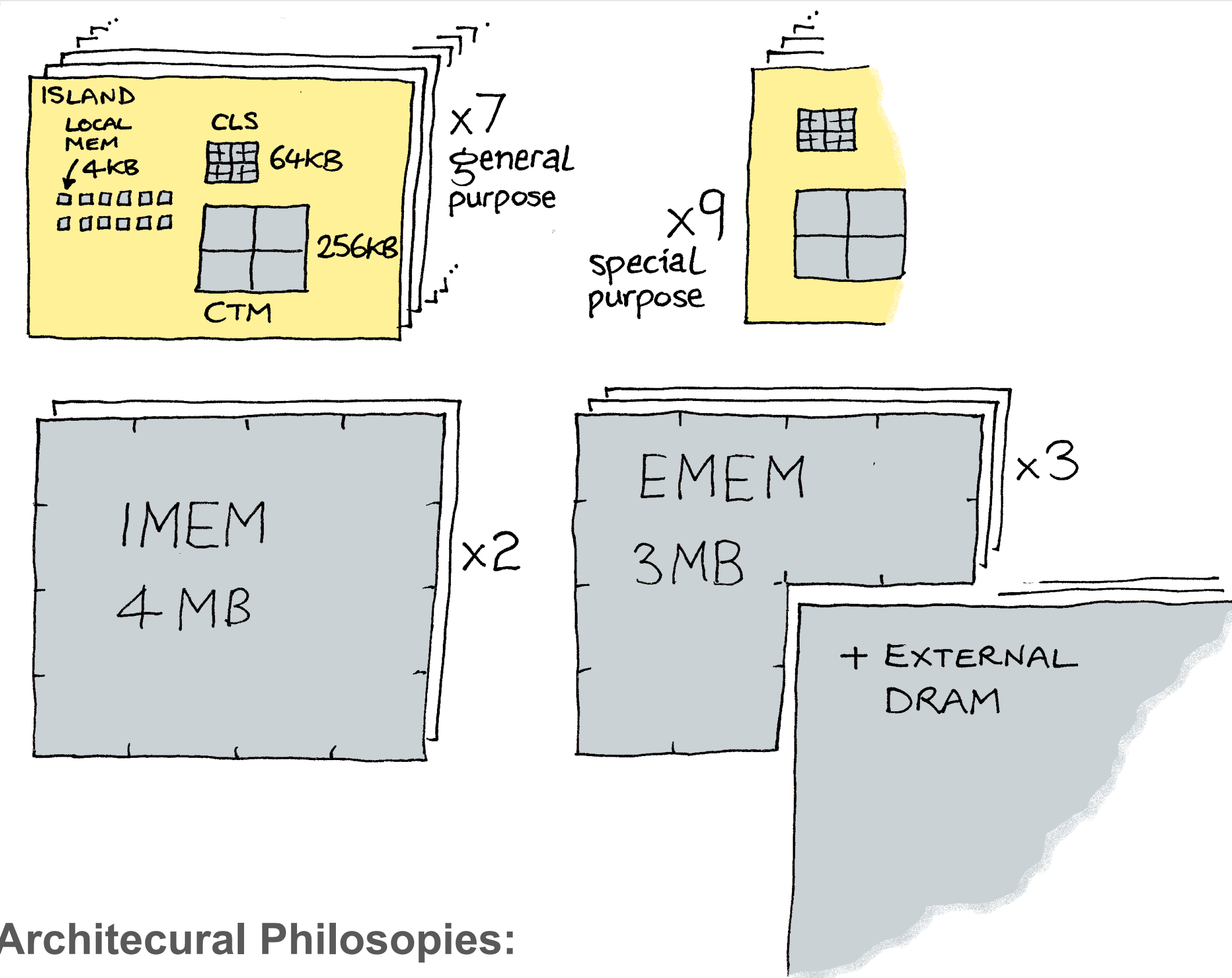
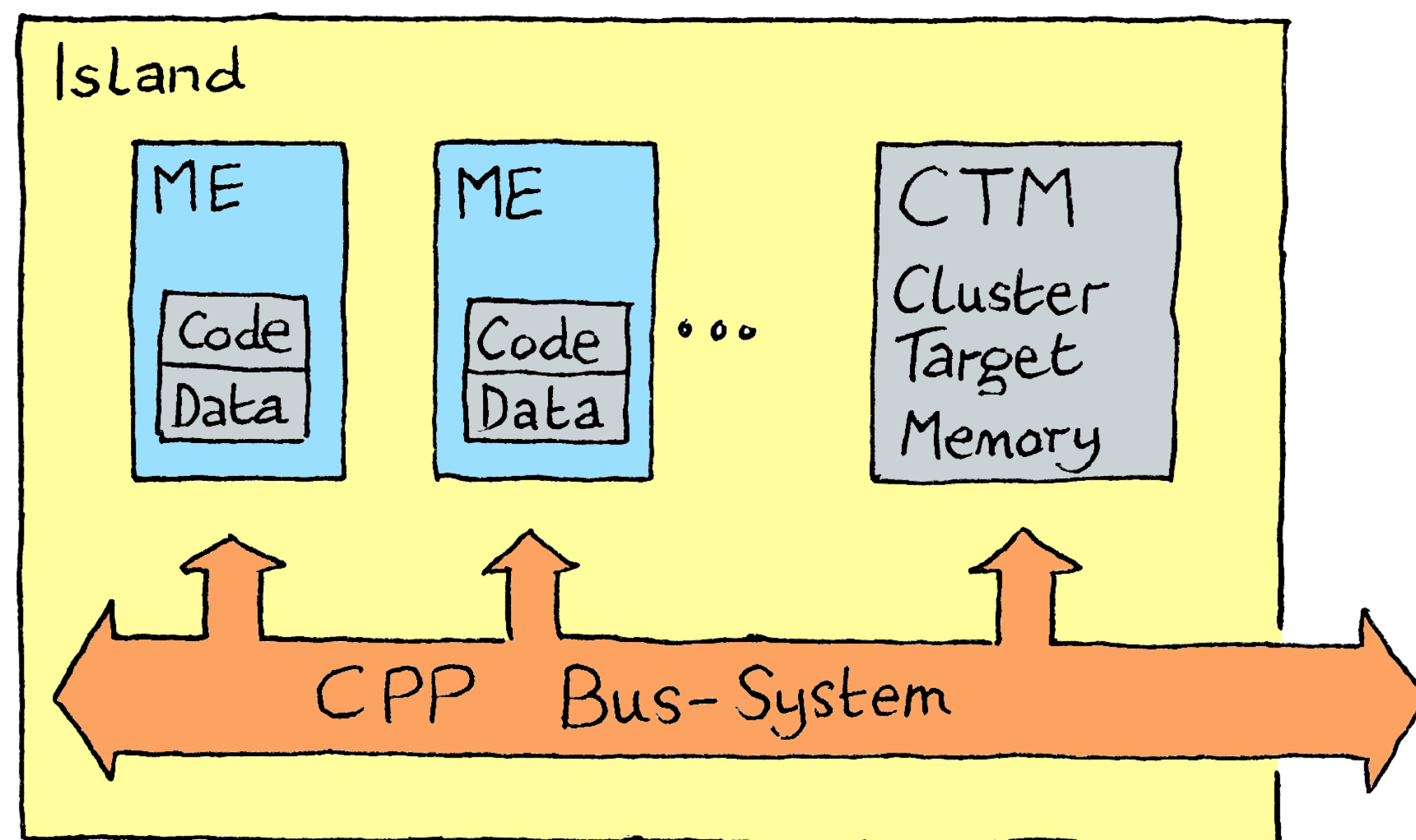
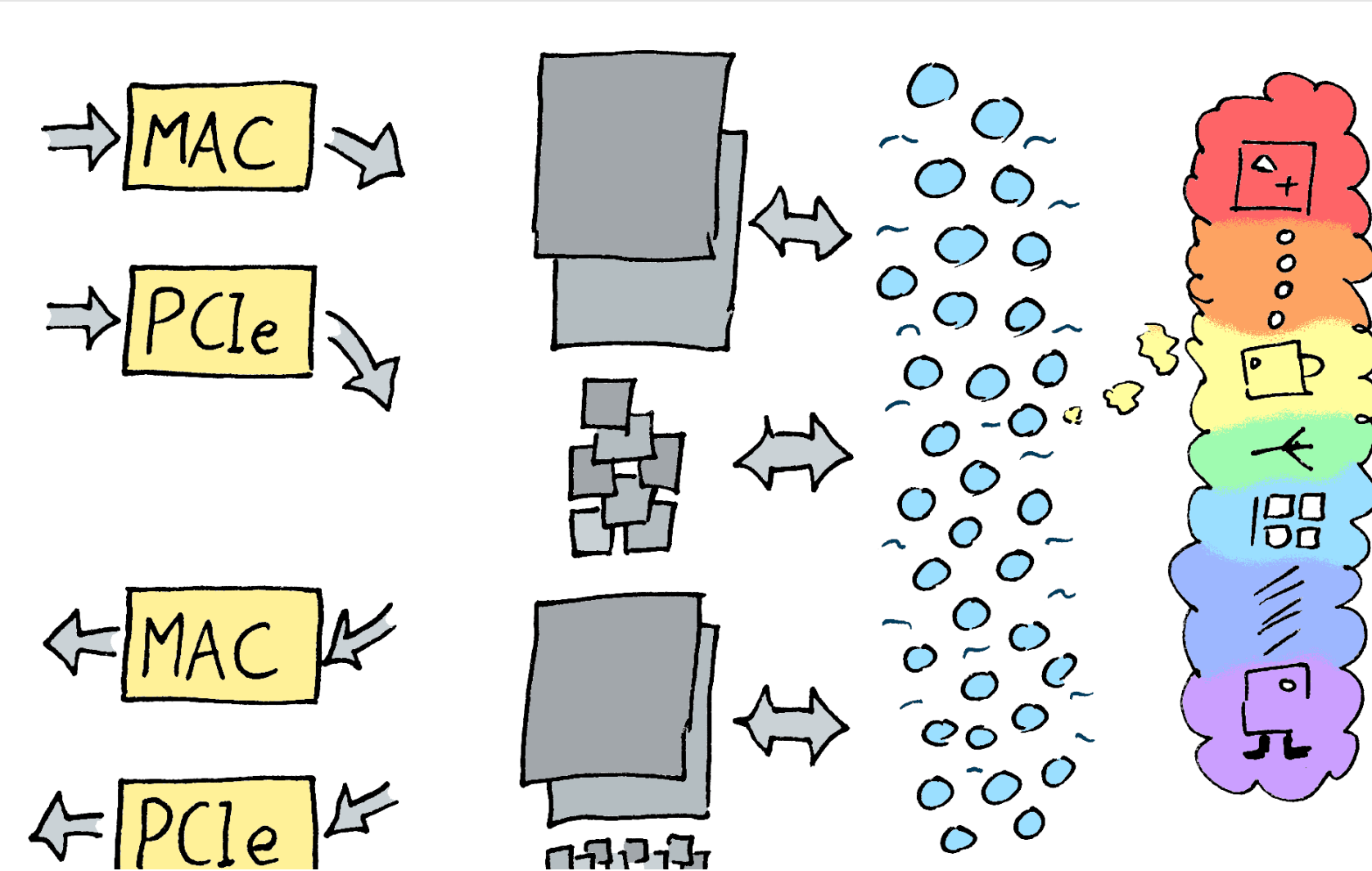




Hardware Intro to NFP (Network Flow Processor) architecture

- NUMA, Memory Hierarchy
- Current eBPF translation on X86/ARM64/PPC64

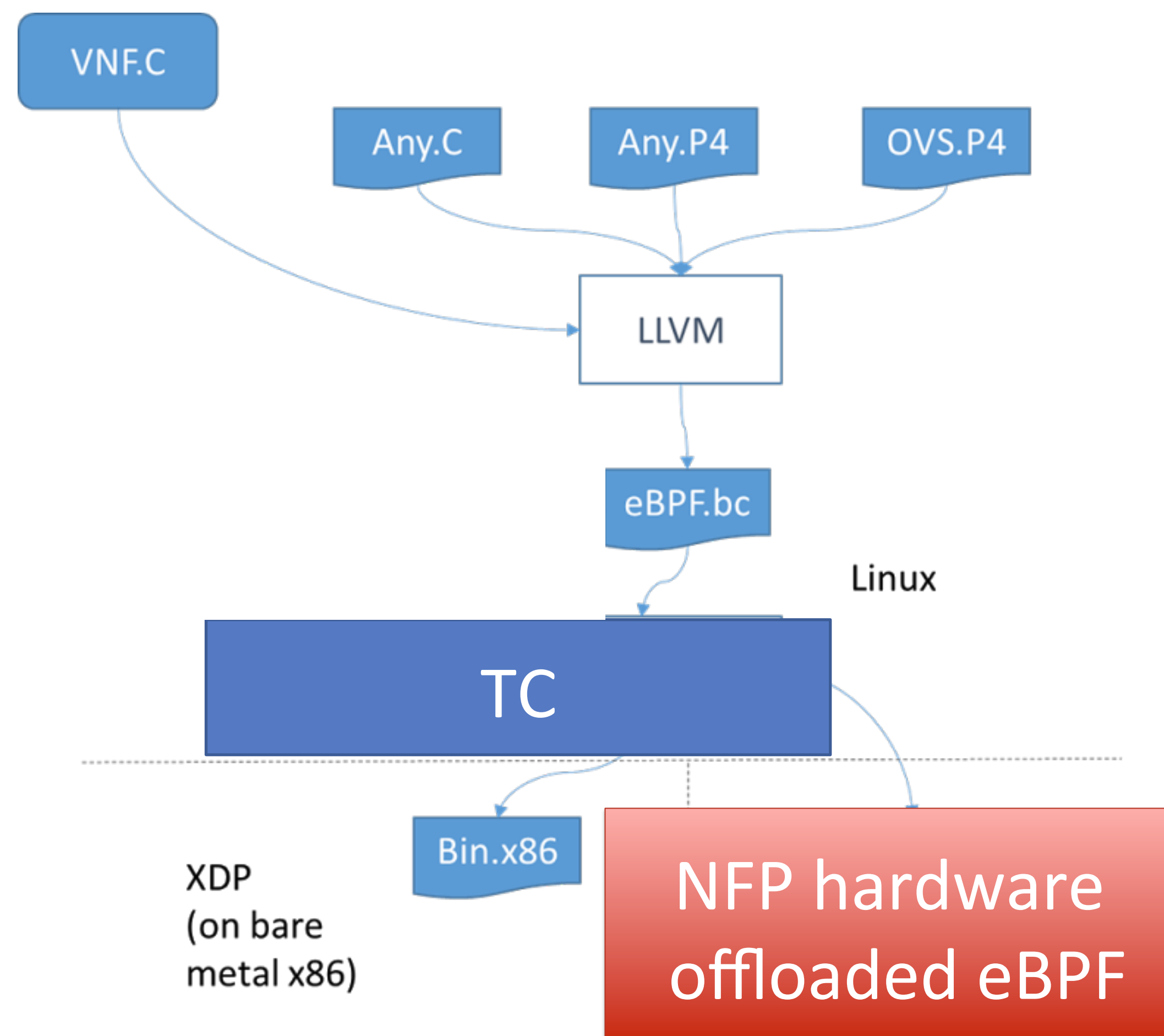
NUMA, Memory Hierarchy



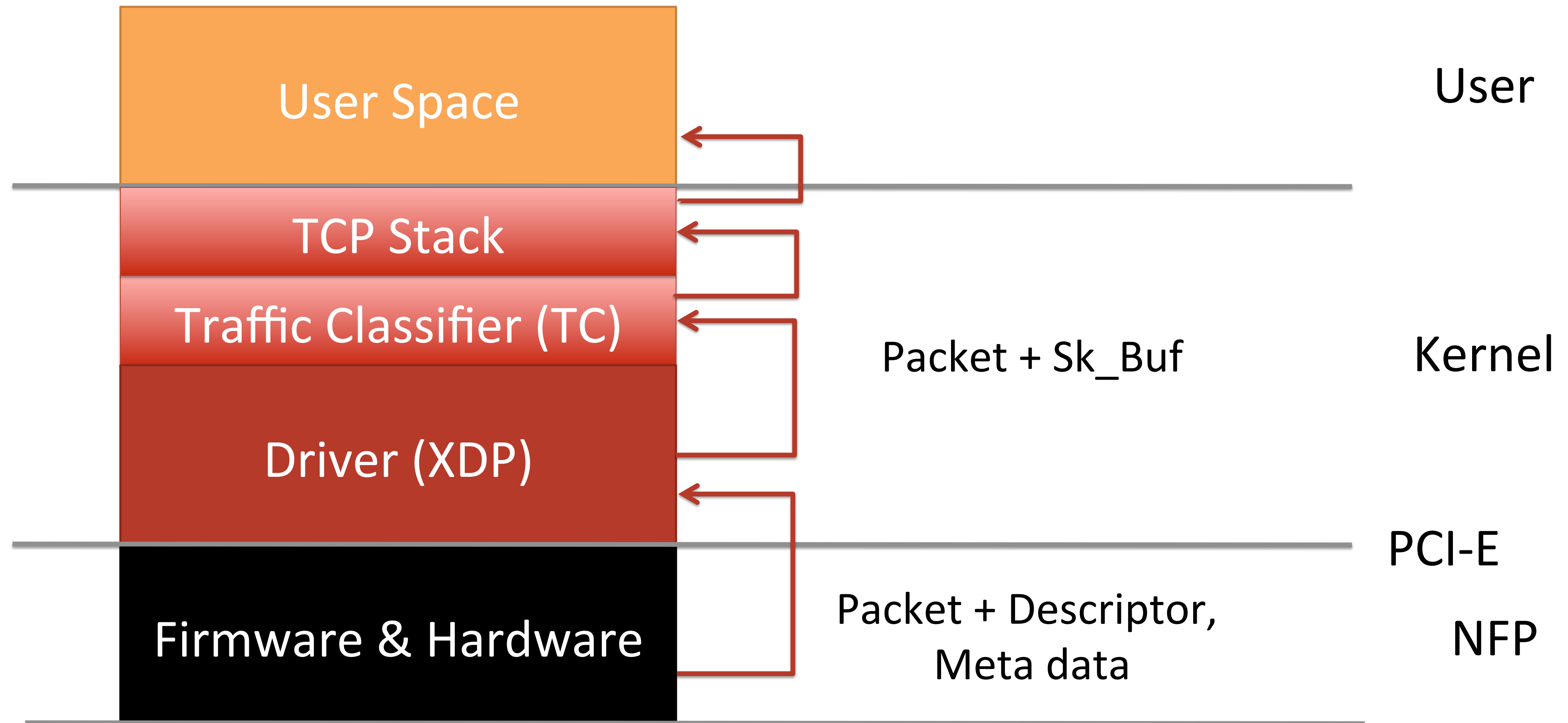
Architectural Philosophies:

- Bring the data close to where it needs to be processed
- Facilitate highly concurrent access to memories
- Mitigate branch costs and hide I/O & memory access latencies

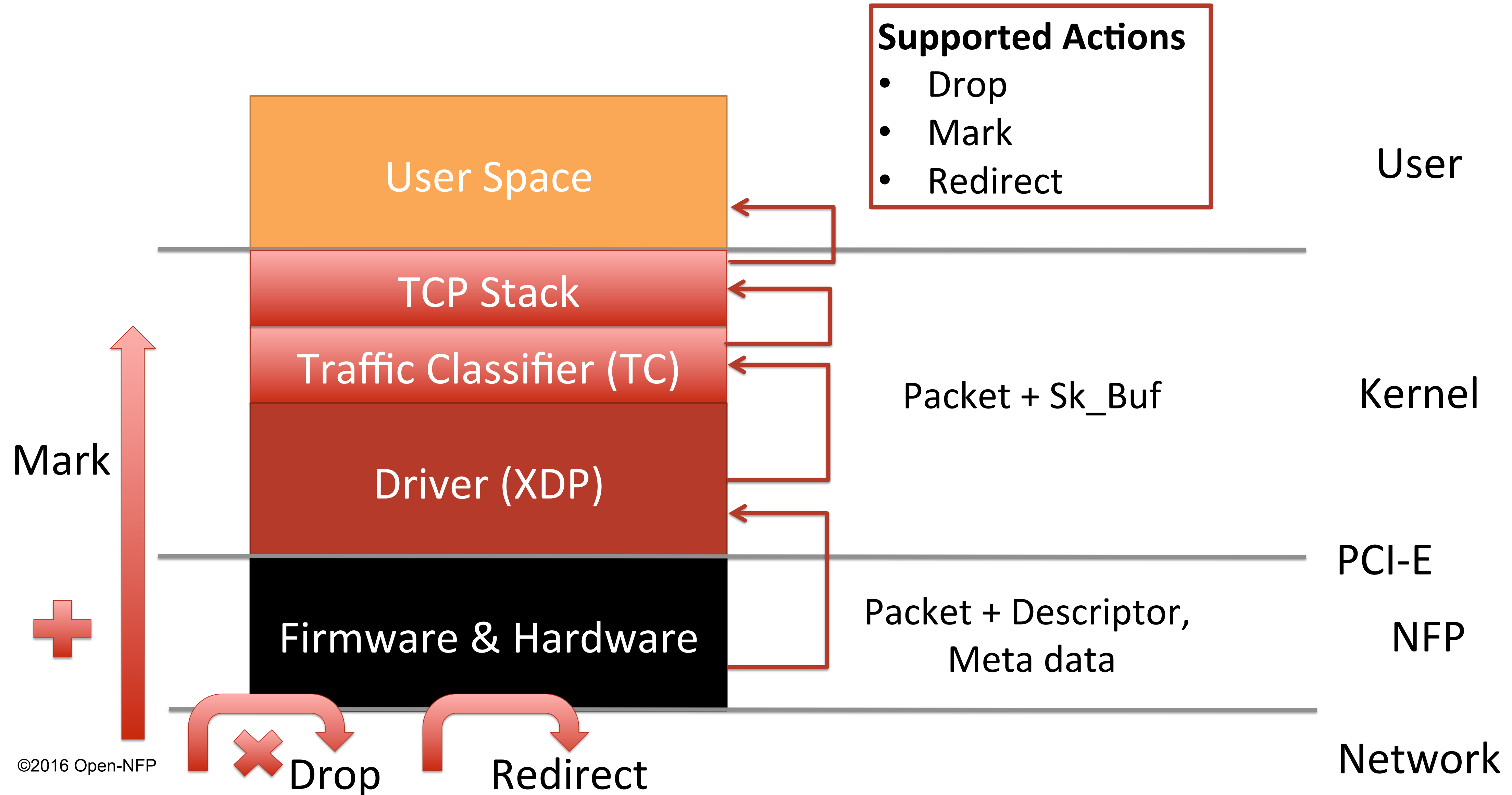
- 1) Write eBPF program as a simple C Program
 - 2) Compiled to eBPF byte code
 - 3) Loaded into the Linux TC
 - 4) Run through verifier
 - 5) Cross compiled to X86/ARM64/PPC64
- ... Or now NFP Byte Code!

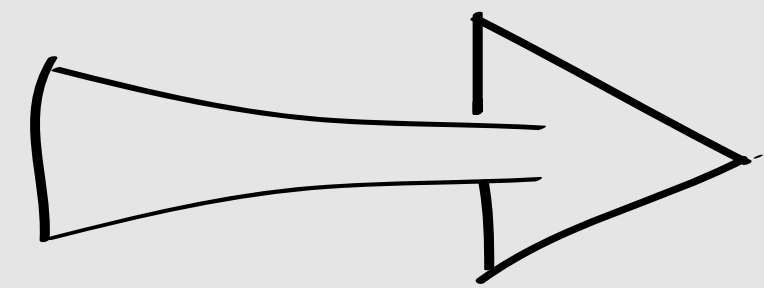


Dataflow



Supported Actions

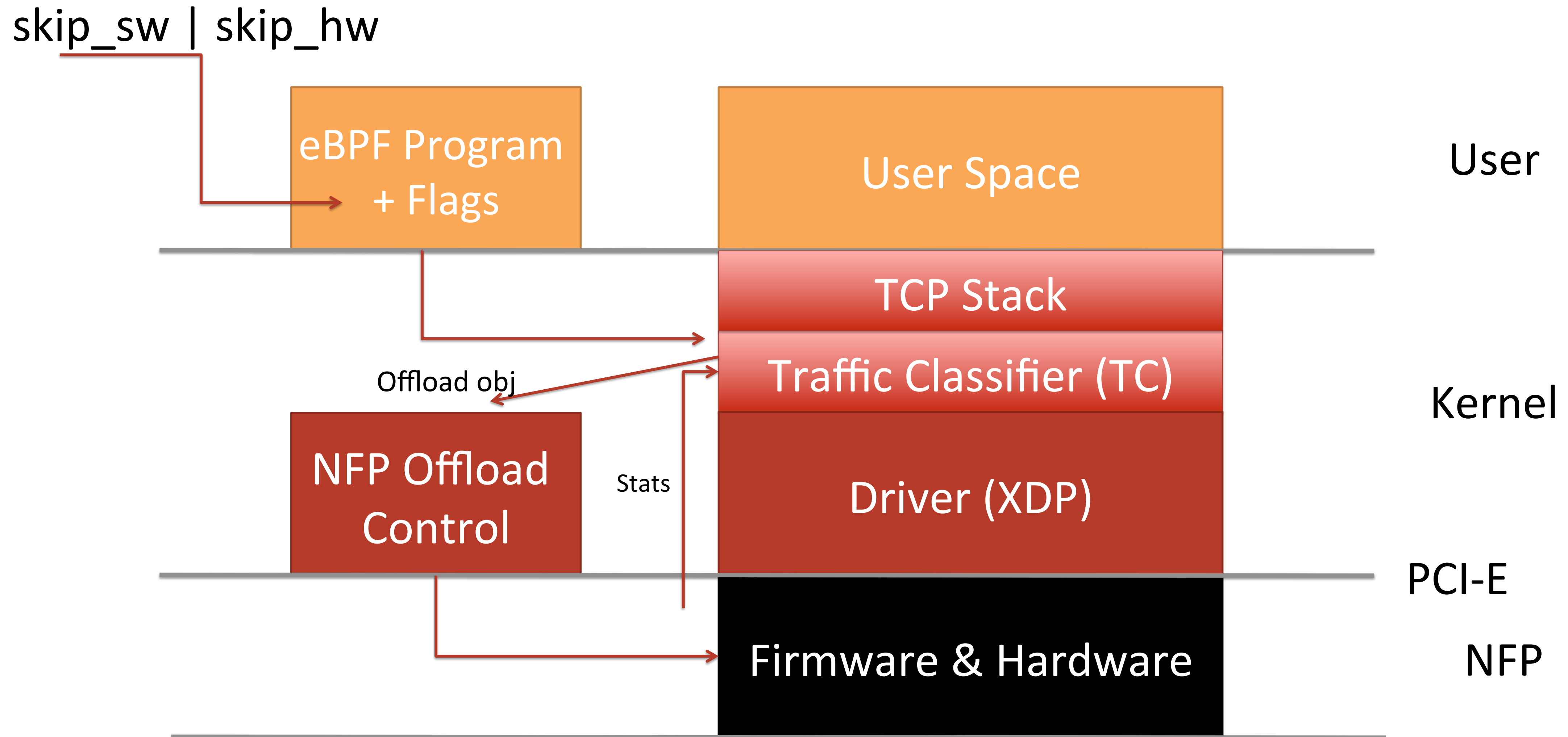




Accelerating P4/eBPF in NFP : Implementation

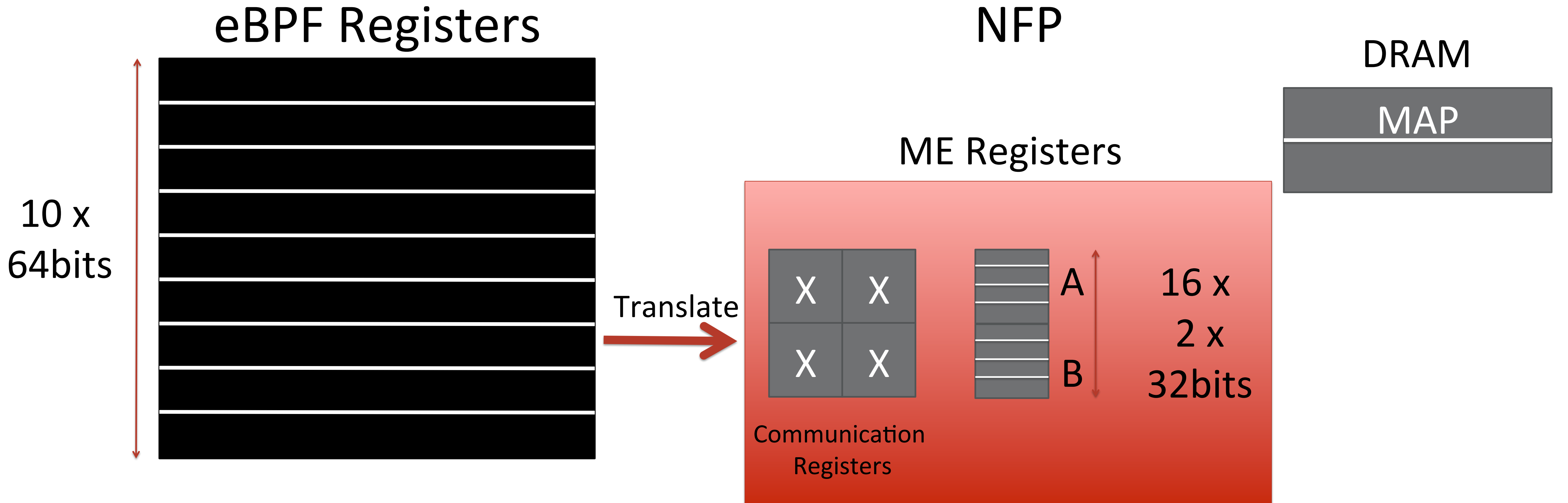
- Kernel core infrastructure
- Map handling in the kernel
- eBPF to NFP
- Map Support
- Optimizations

Kernel core infrastructure



eBPF to NFP

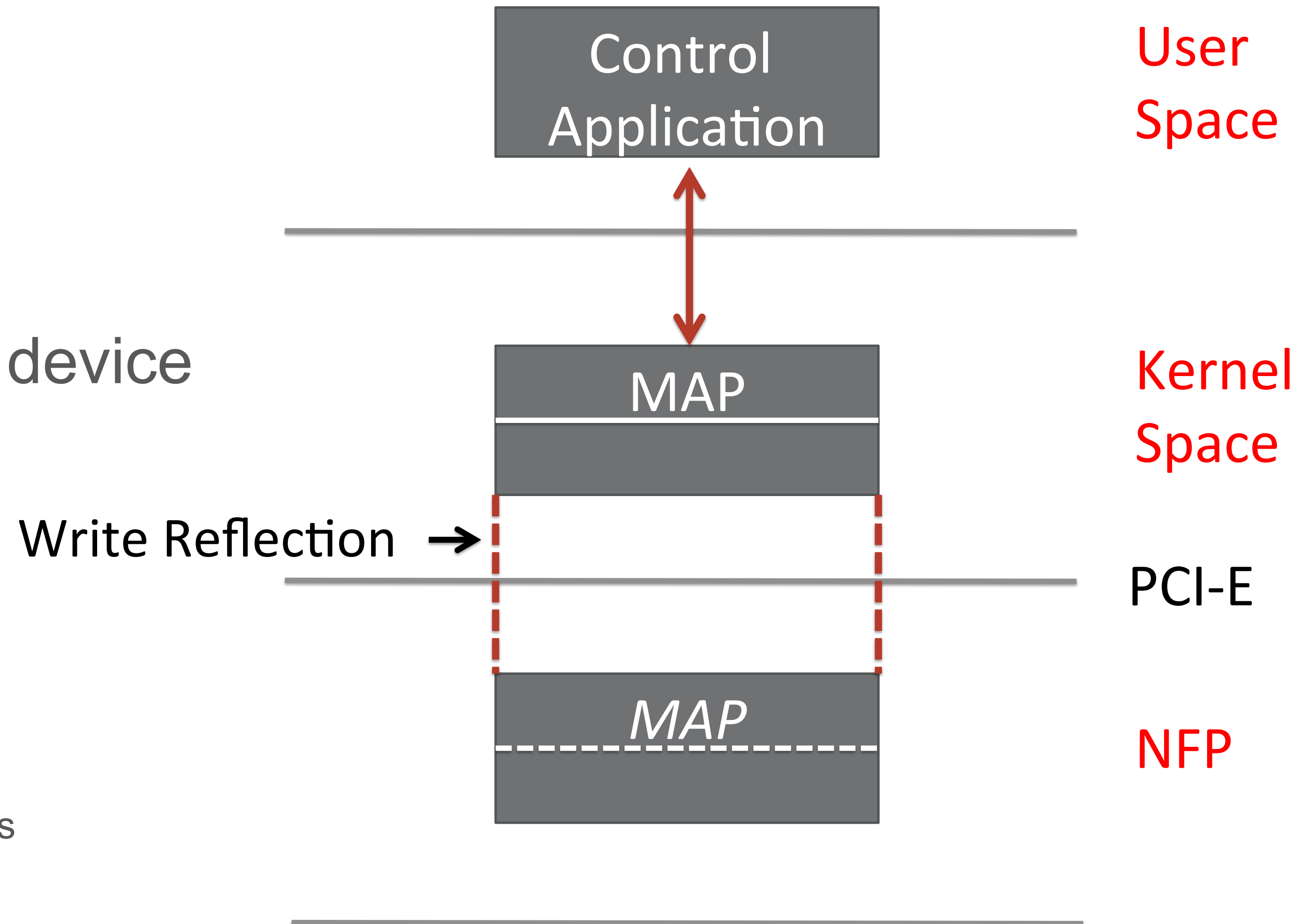
- Non-linear mapping
- 32 bit translation



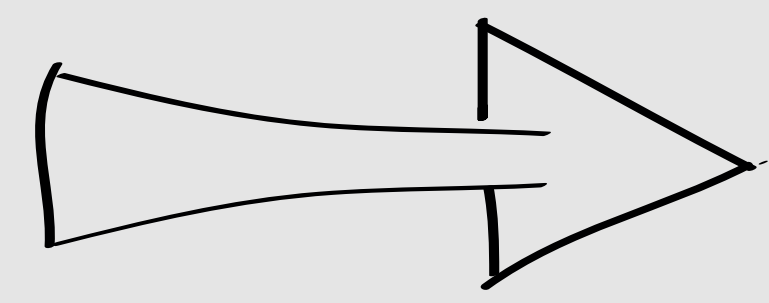
Map Handling in Kernel & Map Write Reflection

- Write interception
- Ownership
- Associating with a device

- Read-Only single-user maps
- Read-Only multi-user maps
- Write-Enabled single-user maps
- Write-Enabled multi-user maps

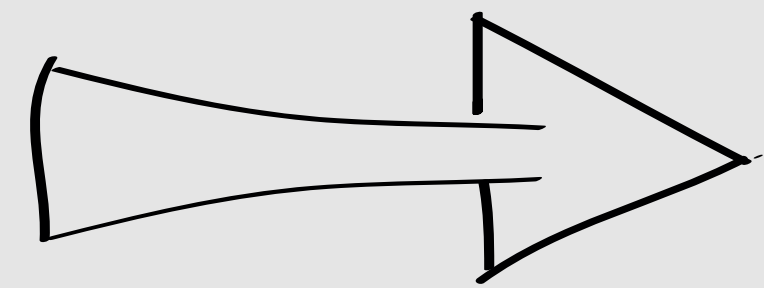


- Dealing with different memory types
- Peephole Optimizer
 - Dropping unnecessary jumps
 - Optimizing instructions with immediates
 - Fusing instructions
- Full A/B register allocations
- Liveness analysis with real state size tracking

A hand-drawn black arrow pointing to the right.

Demo

- Learnt the relationship between P4 and eBPF
- Discovered the infrastructure in the Linux Kernel for eBPF offload
- Learnt about how the Netronome Smart NIC architecture is optimised for network flow processing
- Explored how we implemented the eBPF offload in hardware



QUESTIONS?

Dinan Gunawardena
dinan.gunawardena@netronome.com

Jakub Kicinski
Jakub.Kicinski@netronome.com

A hand-drawn arrow pointing to the right, drawn with black outlines.

THANK YOU