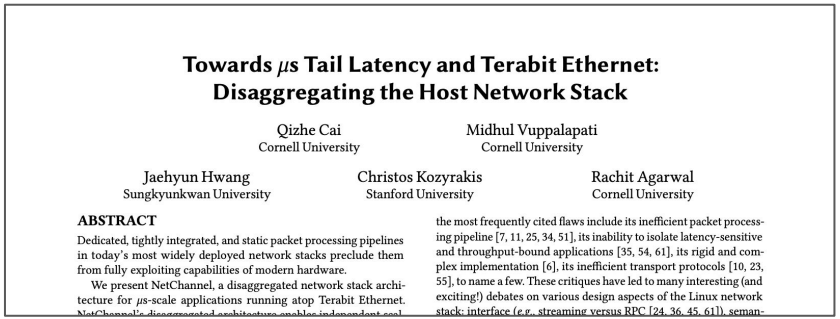
Towards Structurally Extensible Host Network Stacks

Kumar Kartikeya Dwivedi, Rishabh Iyer, Sanidhya Kashyap

EPFL

Host Network Stacks are Receiving Attention

- Recent designs achieve significant throughput gains and microsecond-scale tail latencies.



Towards μs Tail Latency and Terabit Ethernet: Disaggregating the Host Network Stack

Qizhe Cai, Cornell University
Midhul Vuppalapati, Cornell University
Jaehyun Hwang, Sungkyunkwan University
Christos Kozyrakis, Stanford University
Rachit Agarwal, Cornell University

ABSTRACT

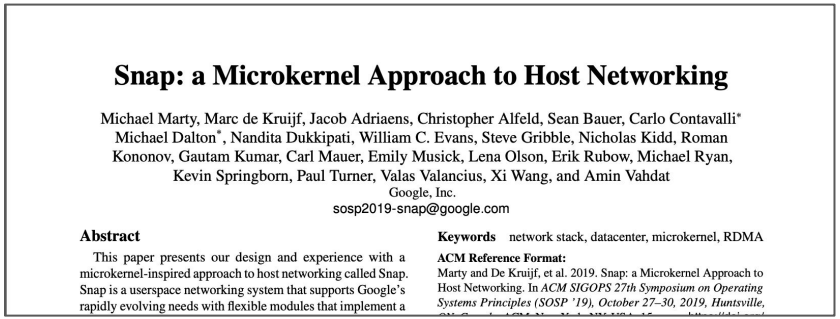
Dedicated, tightly integrated, and static packet processing pipelines in today's most widely deployed network stacks preclude them from fully exploiting capabilities of modern hardware.

We present NetChannel, a disaggregated network stack architecture for μs-scale applications running atop Terabit Ethernet.

the most frequently cited flaws include its inefficient packet processing pipeline [7, 11, 25, 34, 51], its inability to isolate latency-sensitive and throughput-bound applications [35, 54, 61], its rigid and complex implementation [6], its inefficient transport protocols [10, 23, 55], to name a few. These critiques have led to many interesting (and exciting!) debates on various design aspects of the Linux network stack: interface (e.g., streaming versus RPC [24, 36, 45, 61]), seman-

(NetChannel)

(Snap)



Snap: a Microkernel Approach to Host Networking

Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli*
Michael Dalton*, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat
Google, Inc.
sosp2019-snap@google.com

Abstract

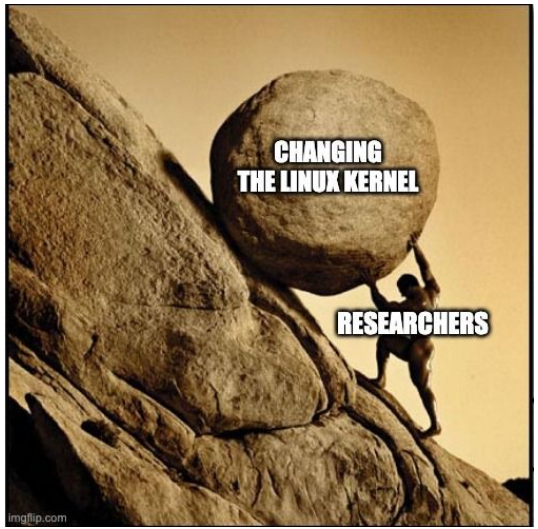
This paper presents our design and experience with a microkernel-inspired approach to host networking called Snap. Snap is a userspace networking system that supports Google's rapidly evolving needs with flexible modules that implement a

Keywords network stack, datacenter, microkernel, RDMA

ACM Reference Format:
Marty and De Kruijf, et al. 2019. Snap: a Microkernel Approach to Host Networking. In *ACM SIGOPS 27th Symposium on Operating Systems Principles (SOSP '19), October 27–30, 2019, Huntsville,*

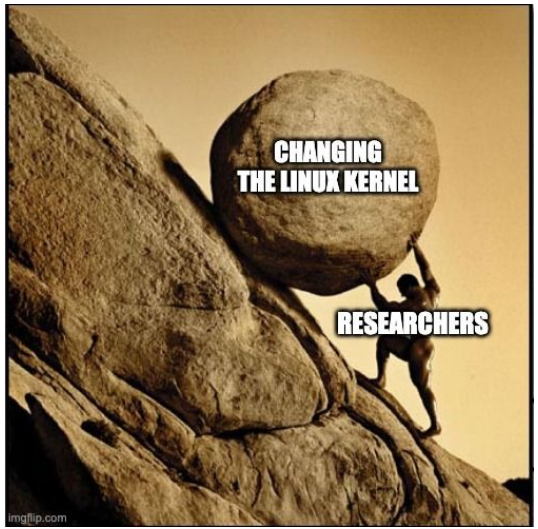
But They Largely Fail in Finding Adoption in Practice

- Modifying the kernel is hard, fragile, and requires expertise.
  - Existing mechanisms for extensibility, such as eBPF are insufficient.



But They Largely Fail in Finding Adoption in Practice

- Modifying the kernel is hard, fragile, and requires expertise.
  - Existing mechanisms for extensibility, such as eBPF are insufficient.
- Developers rebuild in user space or maintain kernel forks.
  - Both options fragment the ecosystem and hinder adoption.



Can we make host network stacks **sufficiently extensible** to support new designs **incrementally**?

Can We Make Network Stacks Sufficiently Extensible?

- What are the missing pieces today that recent designs need?

Can We Make Network Stacks Sufficiently Extensible?

- What are the missing pieces today that recent designs need?
- How can we allow developers to make these changes safely?

Example: NetChannel

- Send-side processing happens on application cores, i.e. tightly integrated with the application.

Example: NetChannel

- Send-side processing happens on application cores, i.e. tightly integrated with the application.
- When co-located latency-critical and batch applications contend for CPUs, tail latency suffers dramatically.

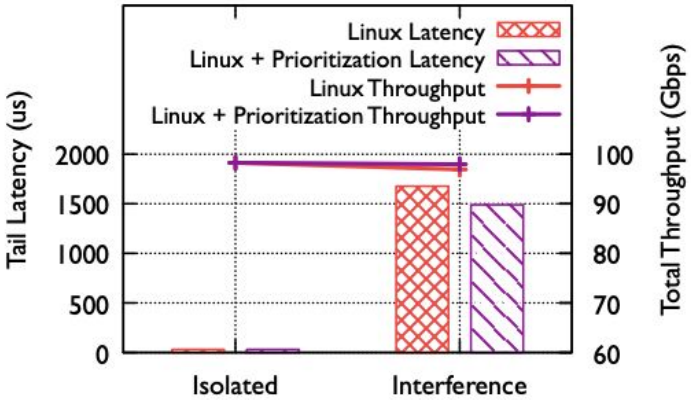


Figure 4, (a), NetChannel

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.
- Run on dedicated CPUs and arbitrate resources based on load.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.
- Run on dedicated CPUs and arbitrate resources based on load.

Encapsulation of work.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.
- Run on dedicated CPUs and arbitrate resources based on load.

Encapsulation of work.

Scheduling within the network stack.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.
- Run on dedicated CPUs and arbitrate resources based on load.

Encapsulation of work.

Scheduling within the network stack.

Scheduling outside the network stack.

Solution: NetChannel

- Identify the portion of send-side processing that is co-located.
- Decouple this unit of packet processing from application cores.
- Run on dedicated CPUs and arbitrate resources based on load.

The same three-pronged approach also applies to Snap, Shenango, Shinjuku.

Network Stacks Need to Be “Structurally Extensible”

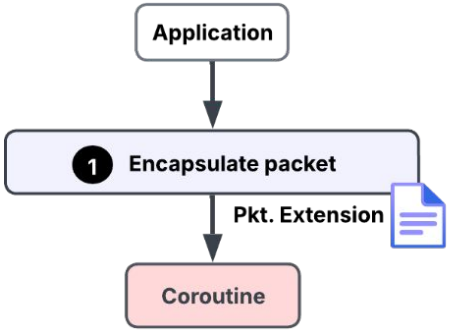
- Current extension mechanisms (eBPF) insufficient, only focus on “functional” changes.

Network Stacks Need to Be “Structurally Extensible”

- Current extension mechanisms (eBPF) insufficient, only focus on “functional” changes.
- We need to change “structural” aspects, i.e. controlling *what* work gets executed, and *where and when* it gets scheduled.

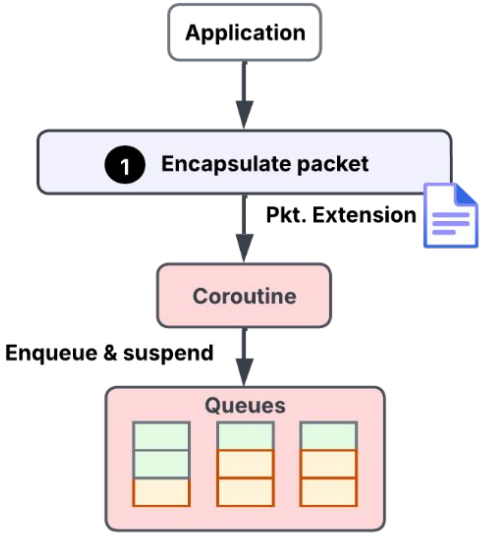
FlexNet

FlexNet



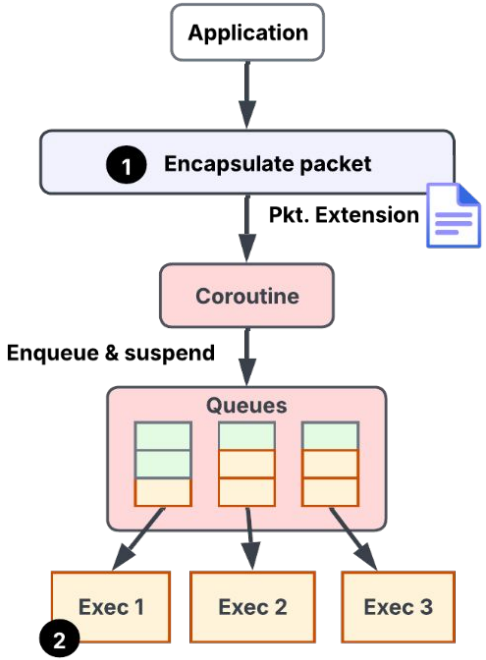
1. Encapsulate packet processing logic into a **coroutine...**

FlexNet



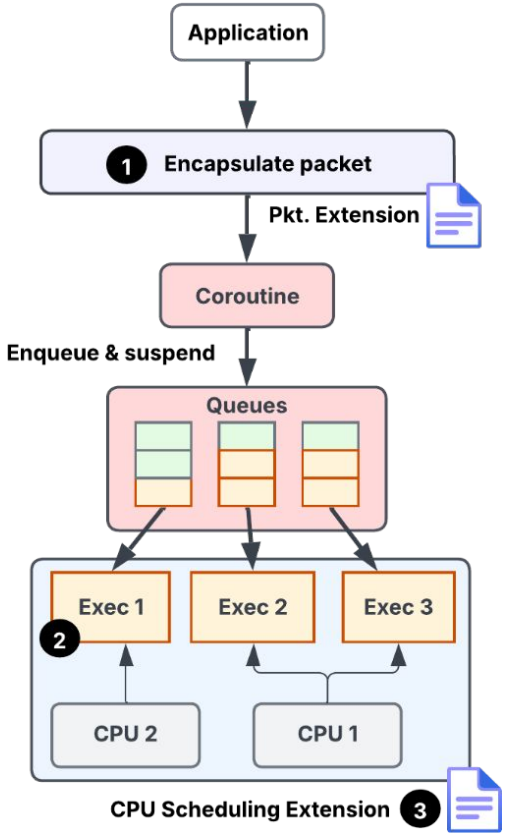
1. Encapsulate packet processing logic into a **coroutine**, and enqueue.

FlexNet



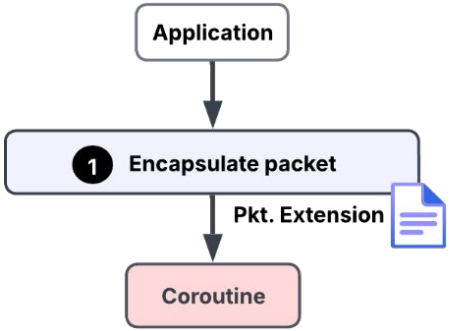
1. Encapsulate packet processing logic into a **coroutine**, and enqueue.
2. Dequeue and drive coroutines to completion to process packets.

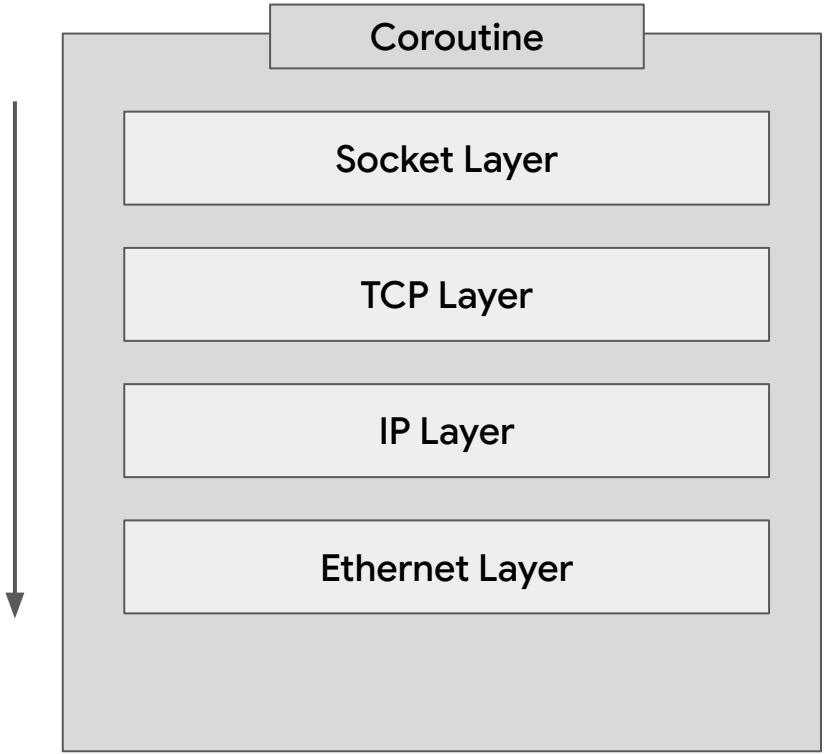
FlexNet



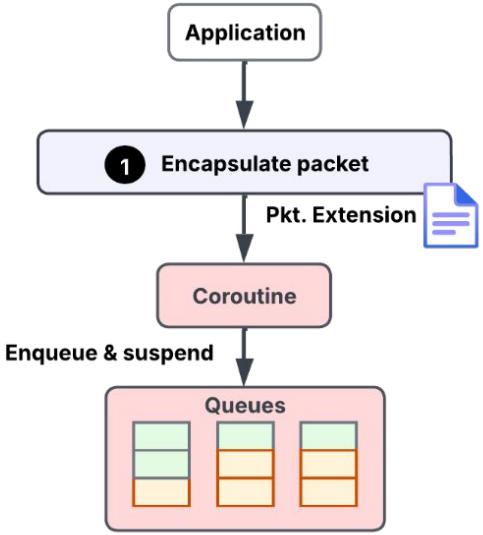
1. Encapsulate packet processing logic into a **coroutine**, and enqueue.
2. Dequeue and drive coroutines to completion to process packets.
3. Scale and compact cores for executors based on load.

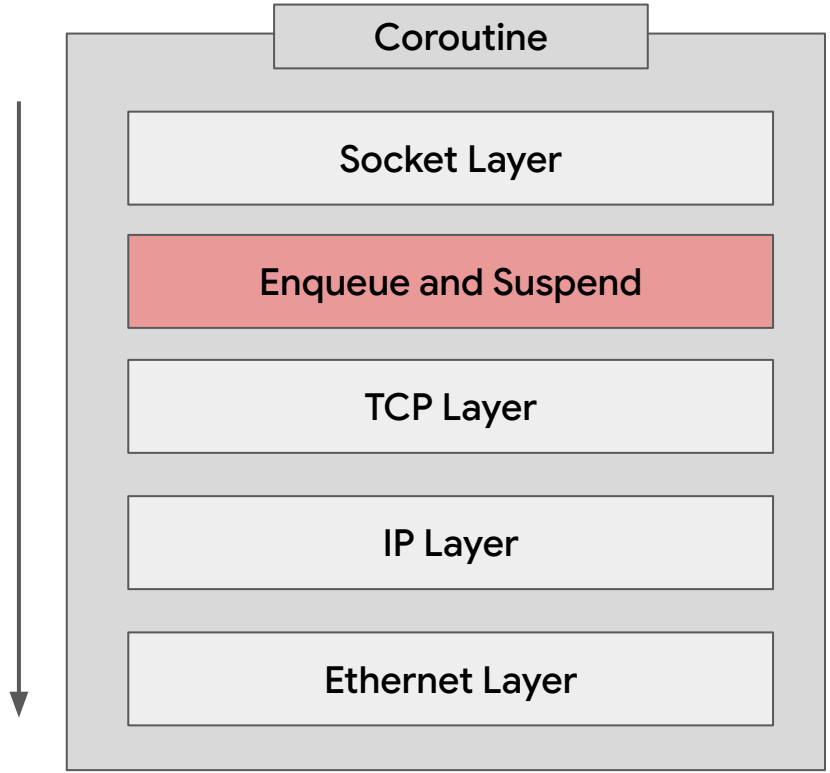
1 Encapsulate a Packet on Transmission



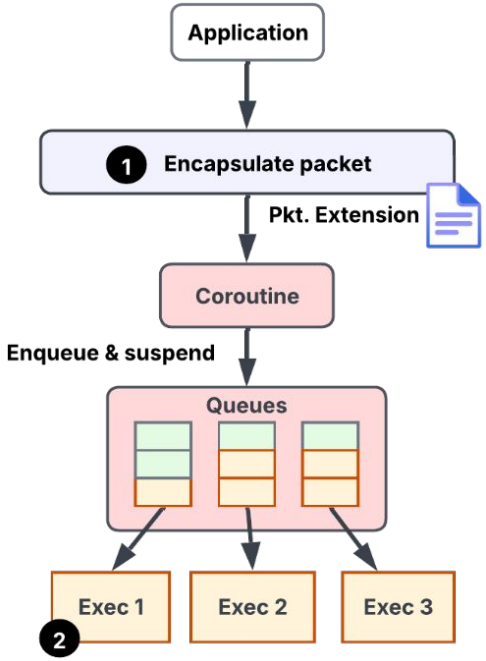


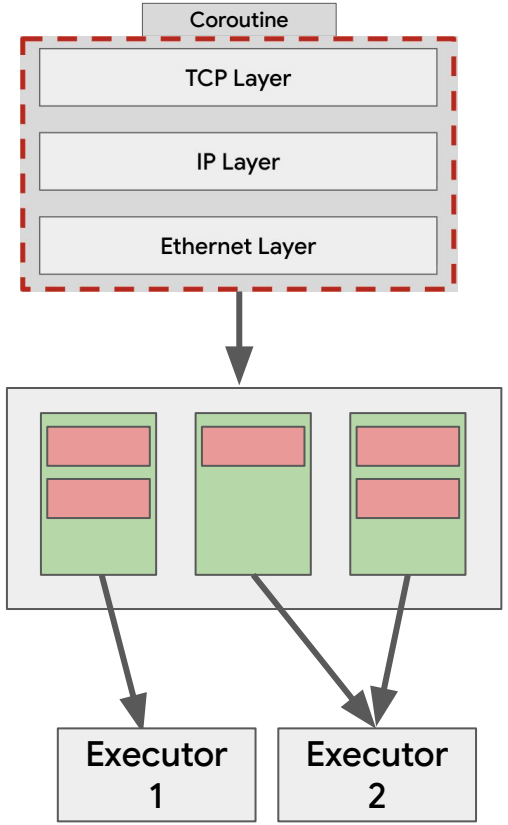
1 Encapsulate a Packet on Transmission



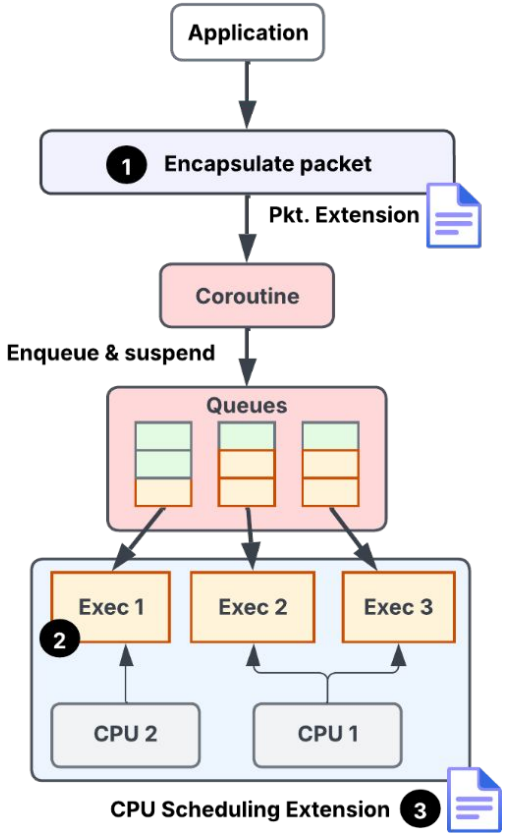


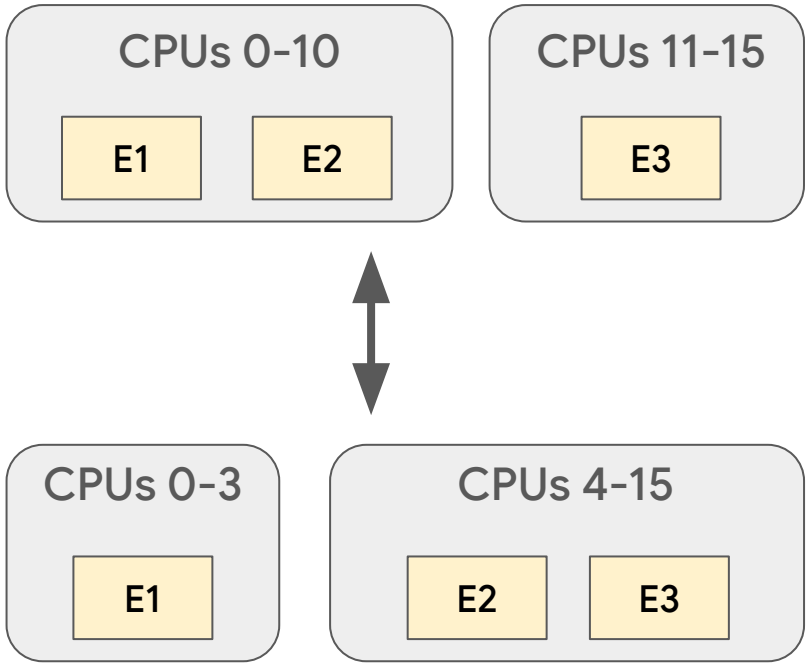
2 Dequeue and Resume in Executors





3 Scale CPUs Dedicated to Executors Based on Load





Takeaways

- Host network stacks should be *structurally* extensible.
  - Must give developers control over work encapsulation and scheduling.

Takeaways

- Host network stacks should be *structurally* extensible.
  - Must give developers control over work encapsulation and scheduling
- FlexNet enables such structural extensibility via coroutines.
  - Allows fine-grained encapsulation of work and programmable scheduling.