ExoPlane: An Operating System for On-Rack Switch Resource Augmentation



Daehyeok Kim Vyas Sekar Srinivasan Seshan

Microsoft and UT Austin *Carnegie Mellon University*

Two trends in in-network computing

Increasing number of applications: Academia & industry proposes many innovative applications [1]

Increasing workload size: Number of concurrent flows and traffic volume keep increasing (e.g., millions of concurrent flows) [2]

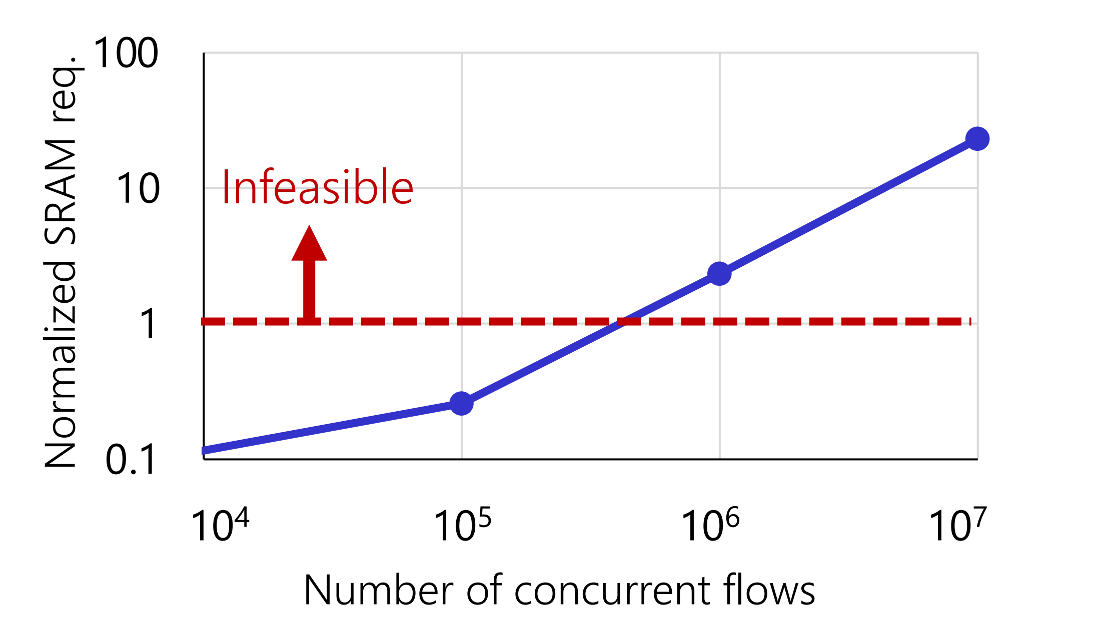
Is in-network computing ready for its prime time?

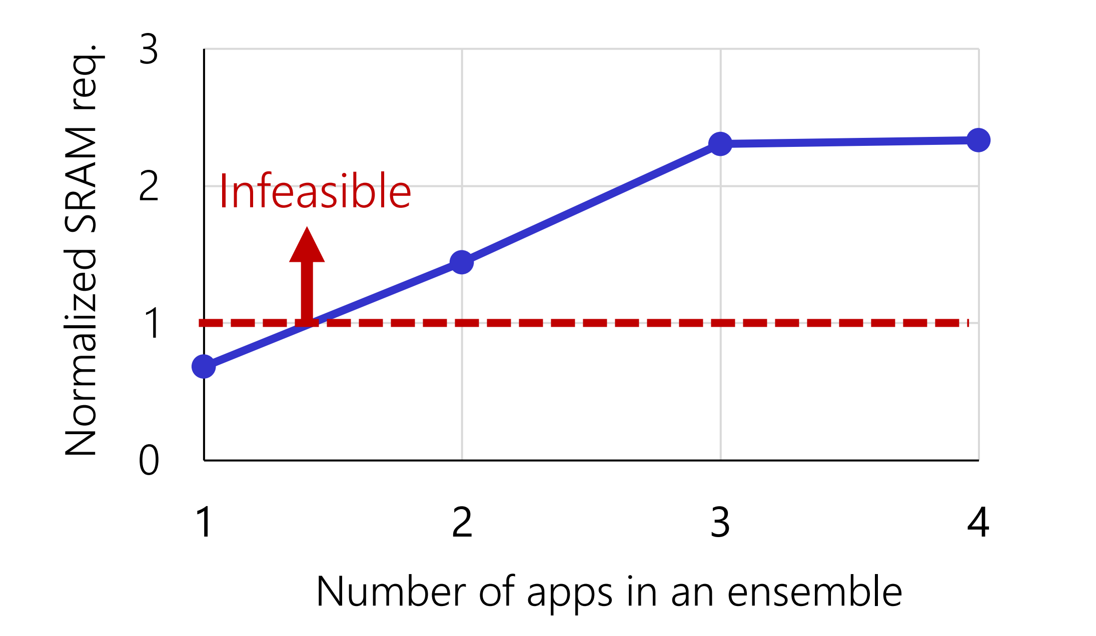
[1] Kfoury et al., An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, applications, challenges, and future trends. IEEE Access, 2021.
[2] Cisco. Cisco Global Cloud Index: Forecast and Methodology 2016–2021, White Paper, 2018.

Problem: Serving concurrent stateful apps on a switch

Example scenario in a datacenter:

Four apps (VPN gateway, NAT, ACL, Monitor) on a switch



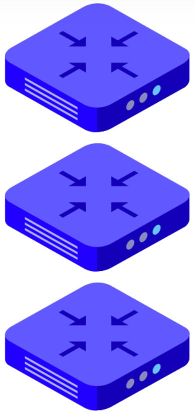


Root cause: Limited switch resources

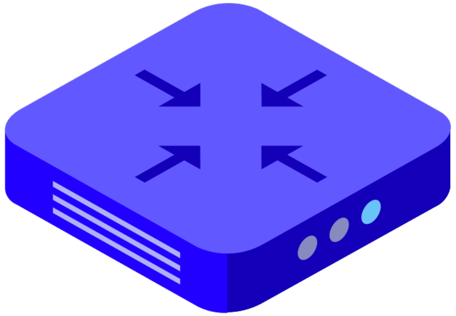
E.g., 10s MB of SRAM ≪ Million flow entries

Possible solutions and limitations

More switches



A beefier switch



Optimizing applications

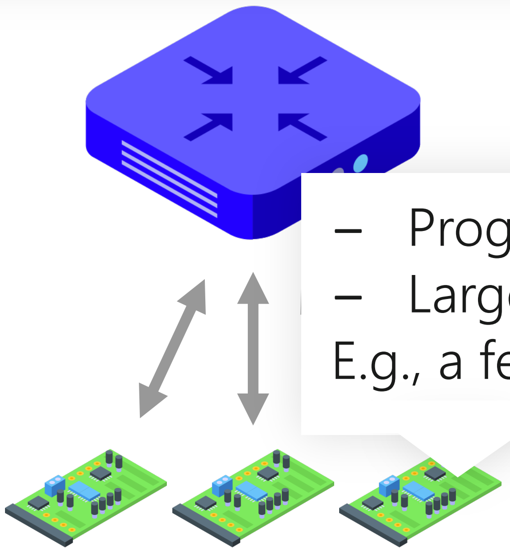
Resource-efficient design (e.g., using sketches)

Expensive
Hard to extend

Not generally applicable

Case for on-rack switch resource augmentation

On-rack resource augmentation:
A switch + resource on external devices



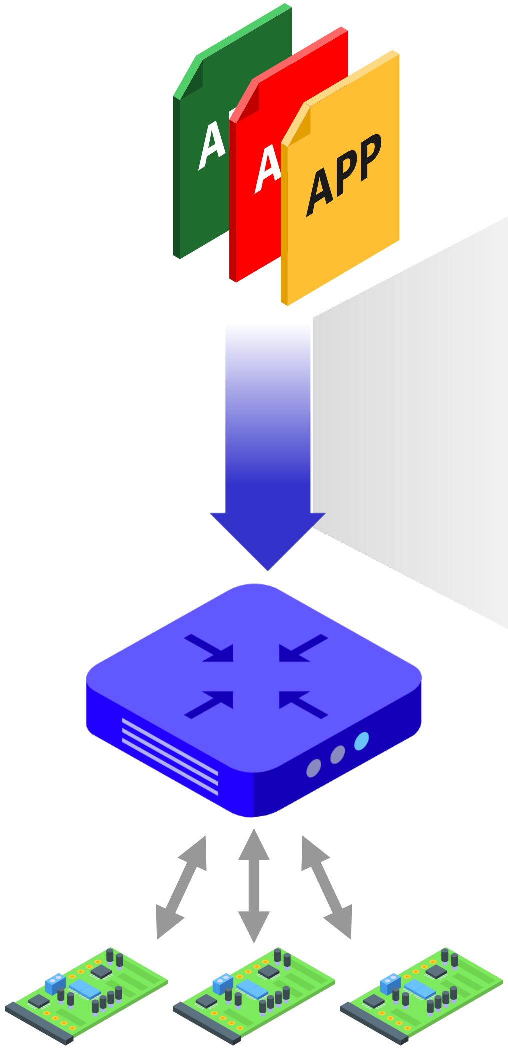
- Programmable
- Larger resources

E.g., a few GB of DRAM

Cost efficient
Easy to extend

What do we need for realizing it?

We need an OS [AD'12]!



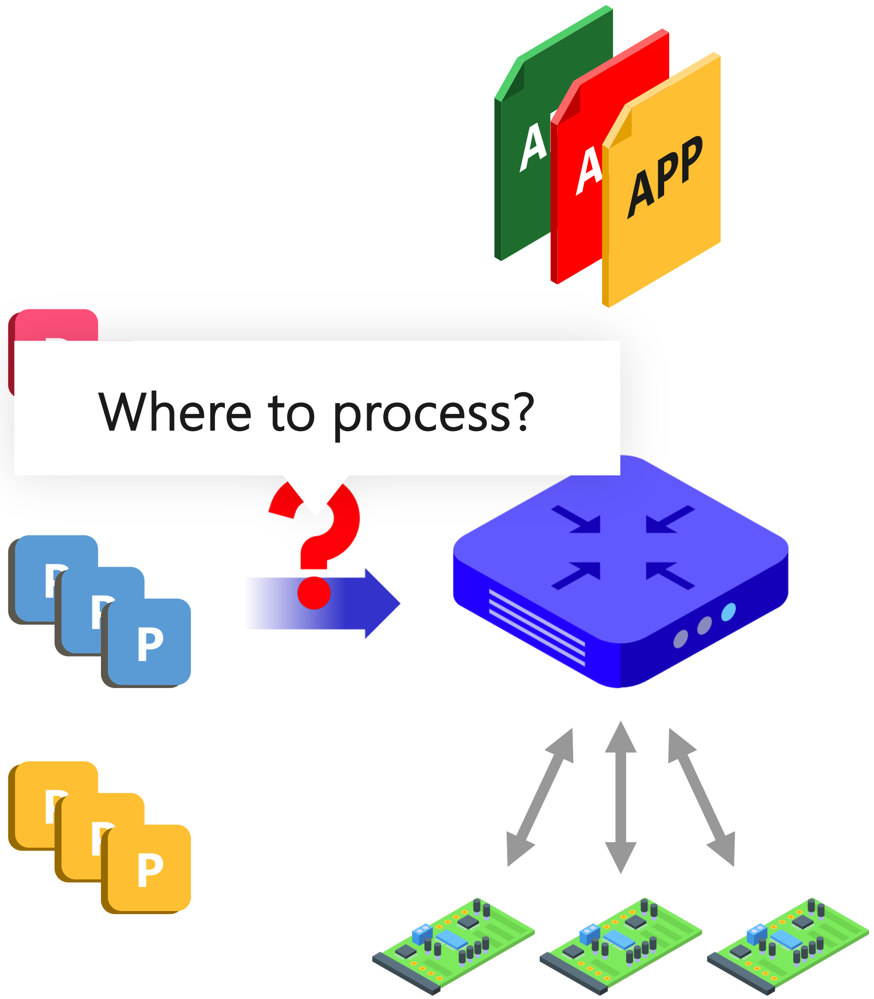
Providing abstractions of resources

Managing shared resource between apps

Facilitating the sharing of resources at runtime

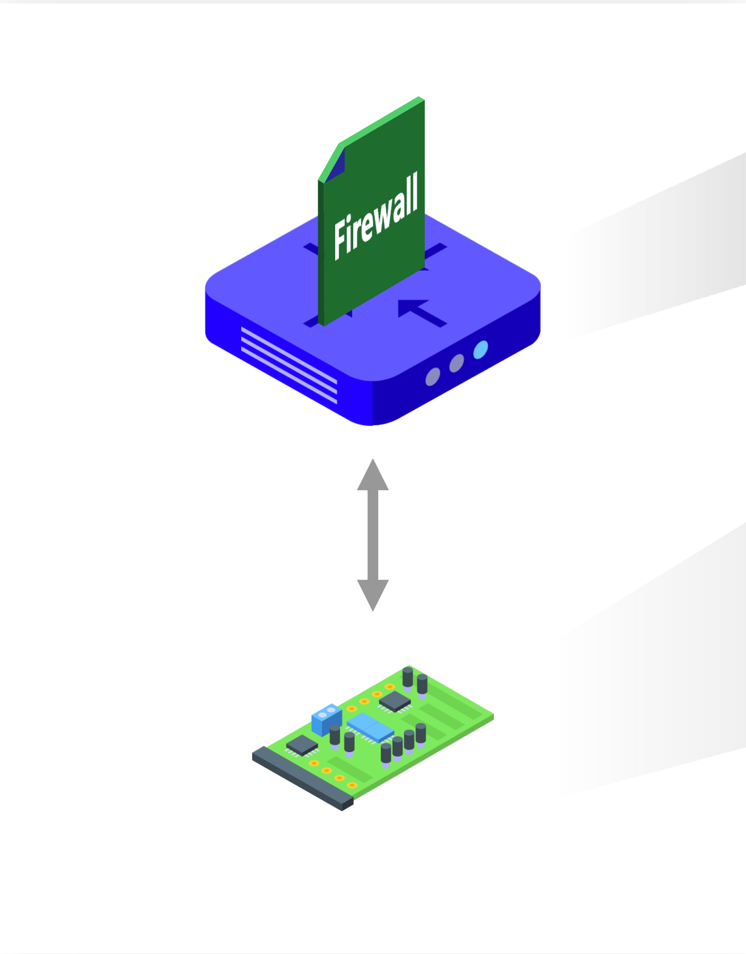
[AD'12] Anderson and Dahlin, Operating Systems: Principles and Practice, Recursive Books, 2012.

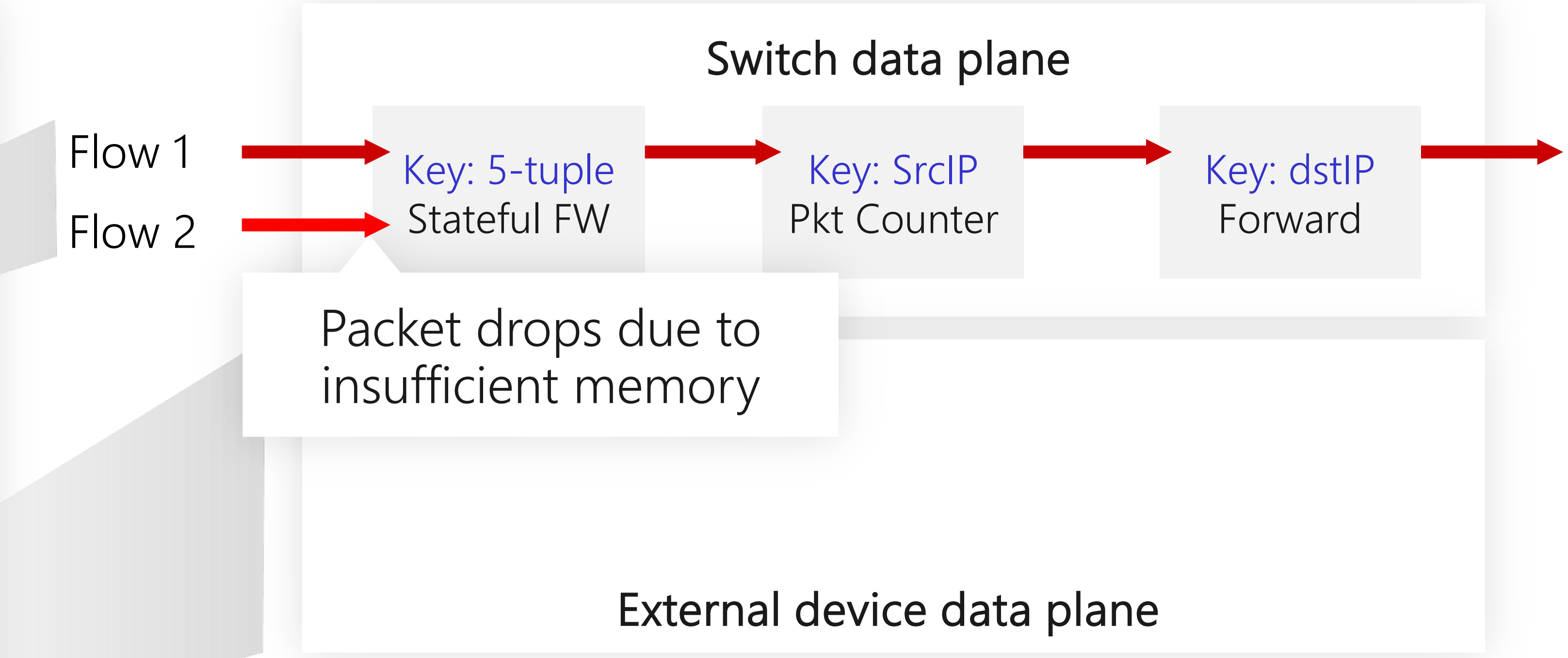
What should an “operating model” be?



Strawman model 1: App pinning

Pin an app to one device and process packets on that device

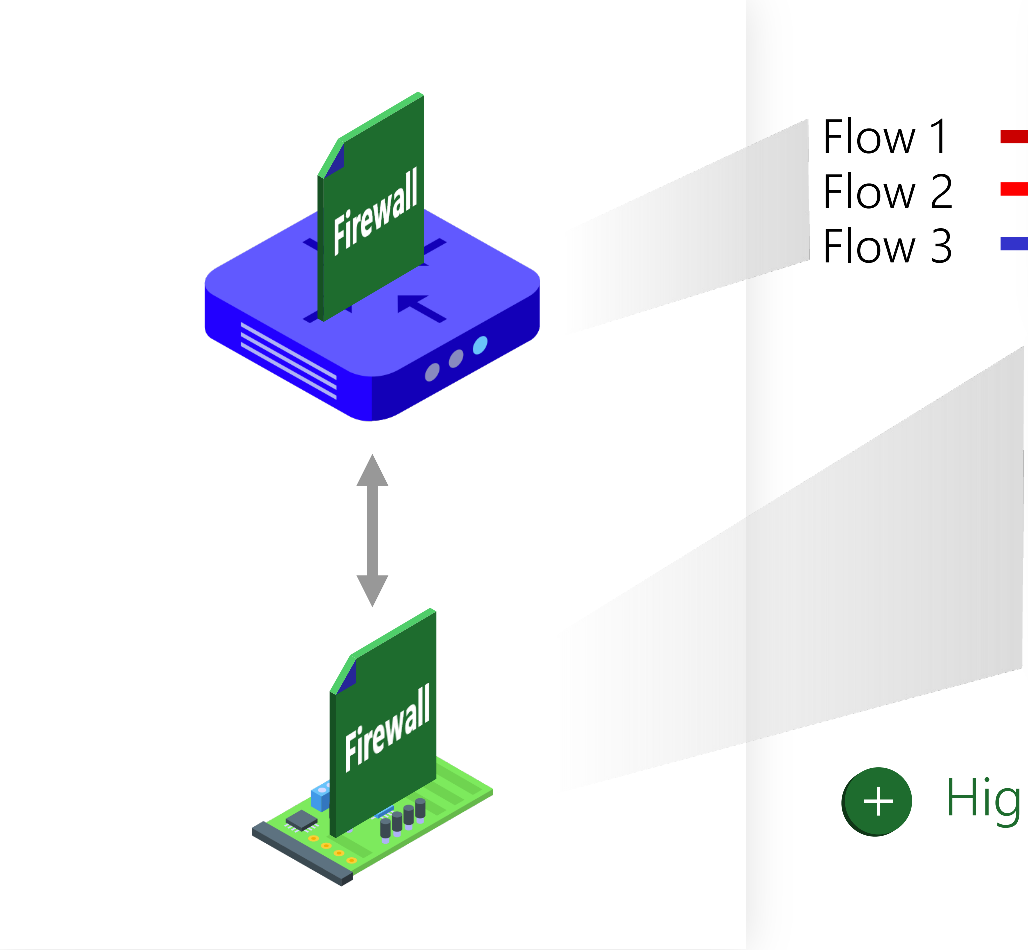


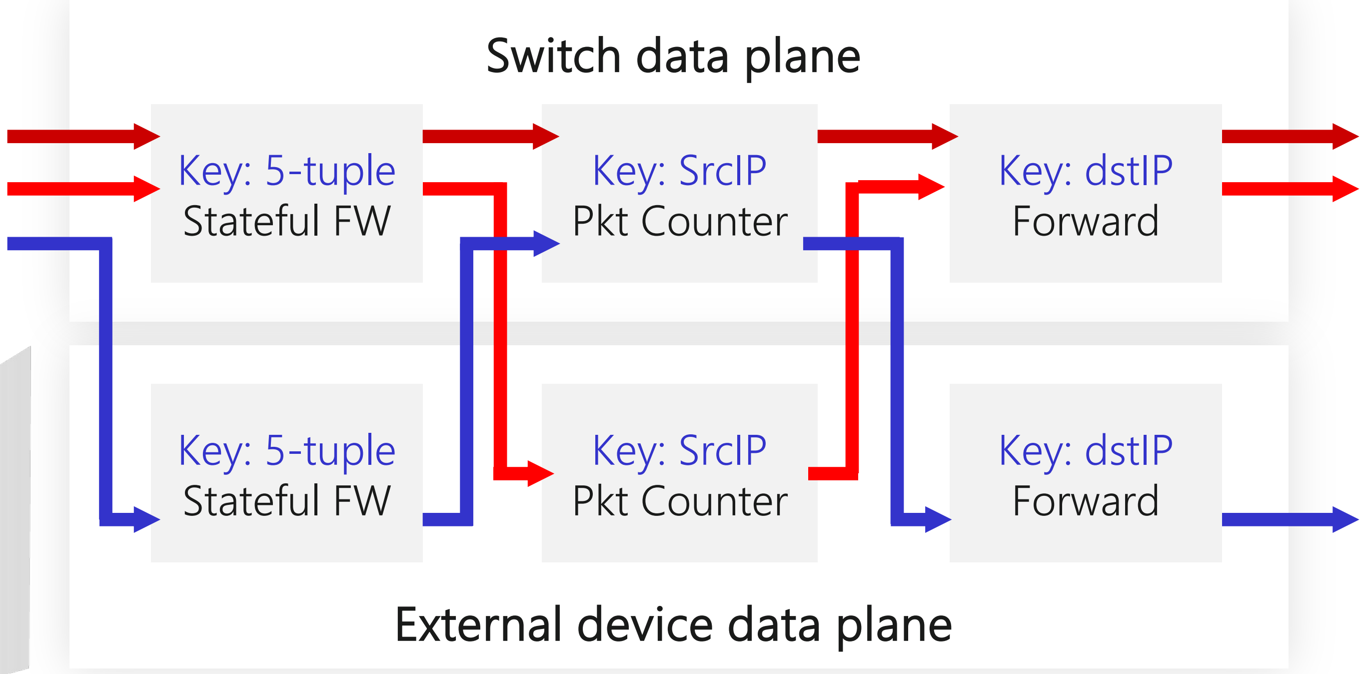


- \+ Low performance overhead
- \+ Low resource overhead
- – Resource underutilization

Strawman model 2: Full disaggregation

An app running on multi-devices and processing a packet on multi-devices

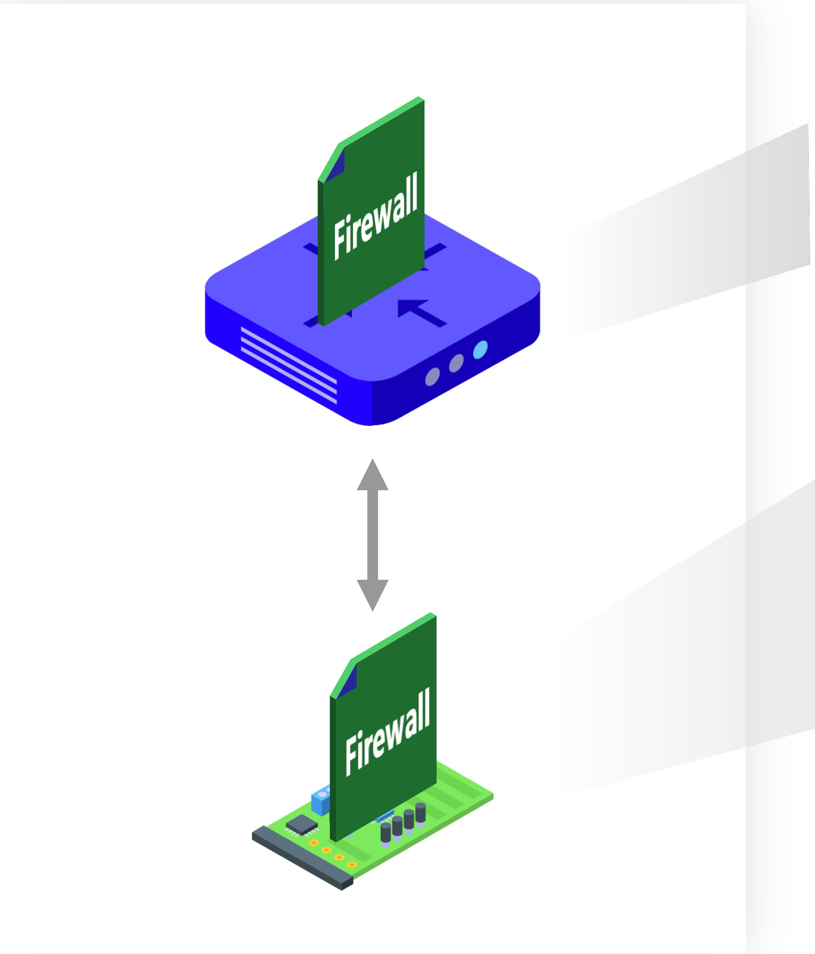


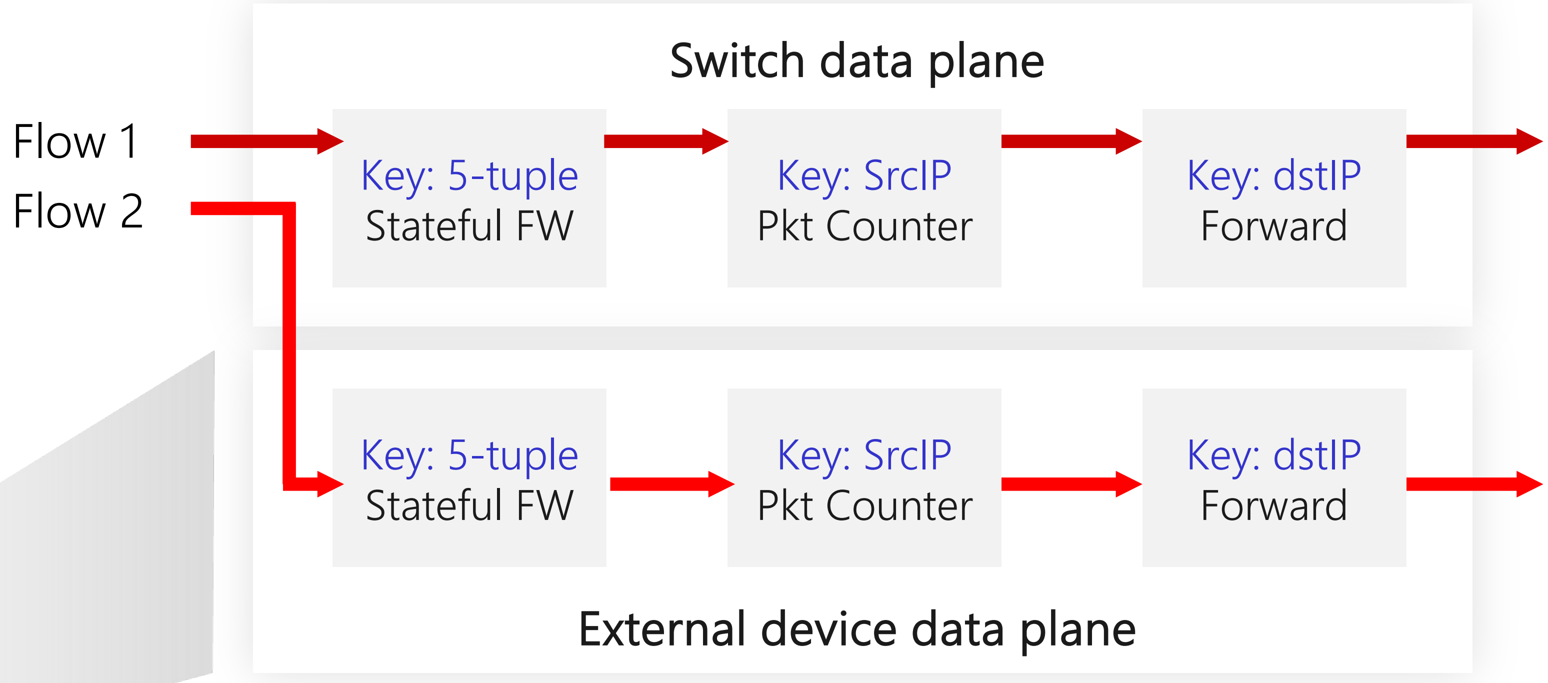


- + High resource utilization
- − High performance overhead
- − High resource overhead

Candidate model: Packet pinning

An app running on multi-devices and processing a packet on a single device

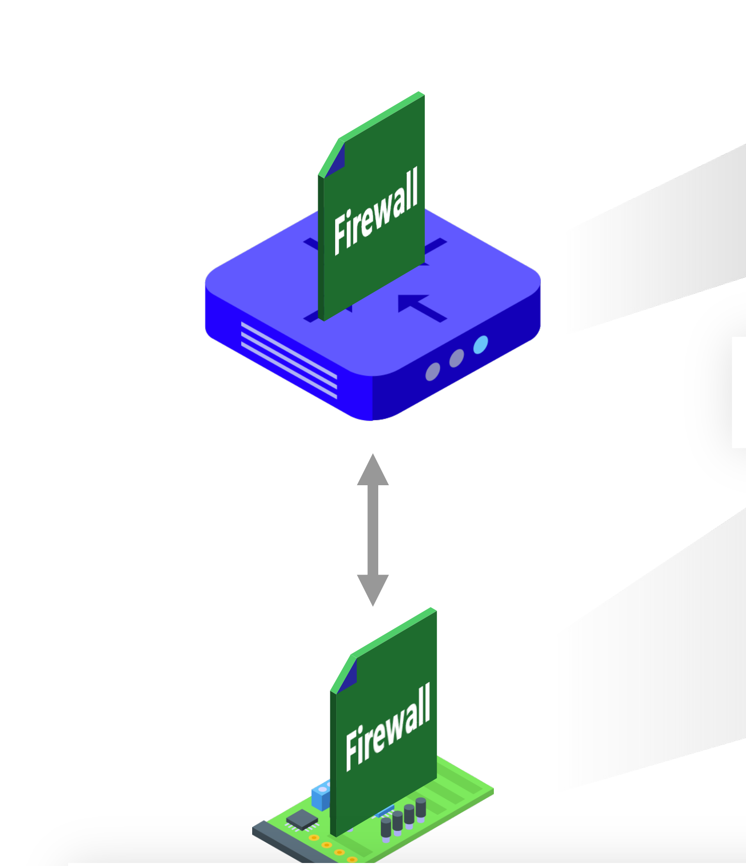


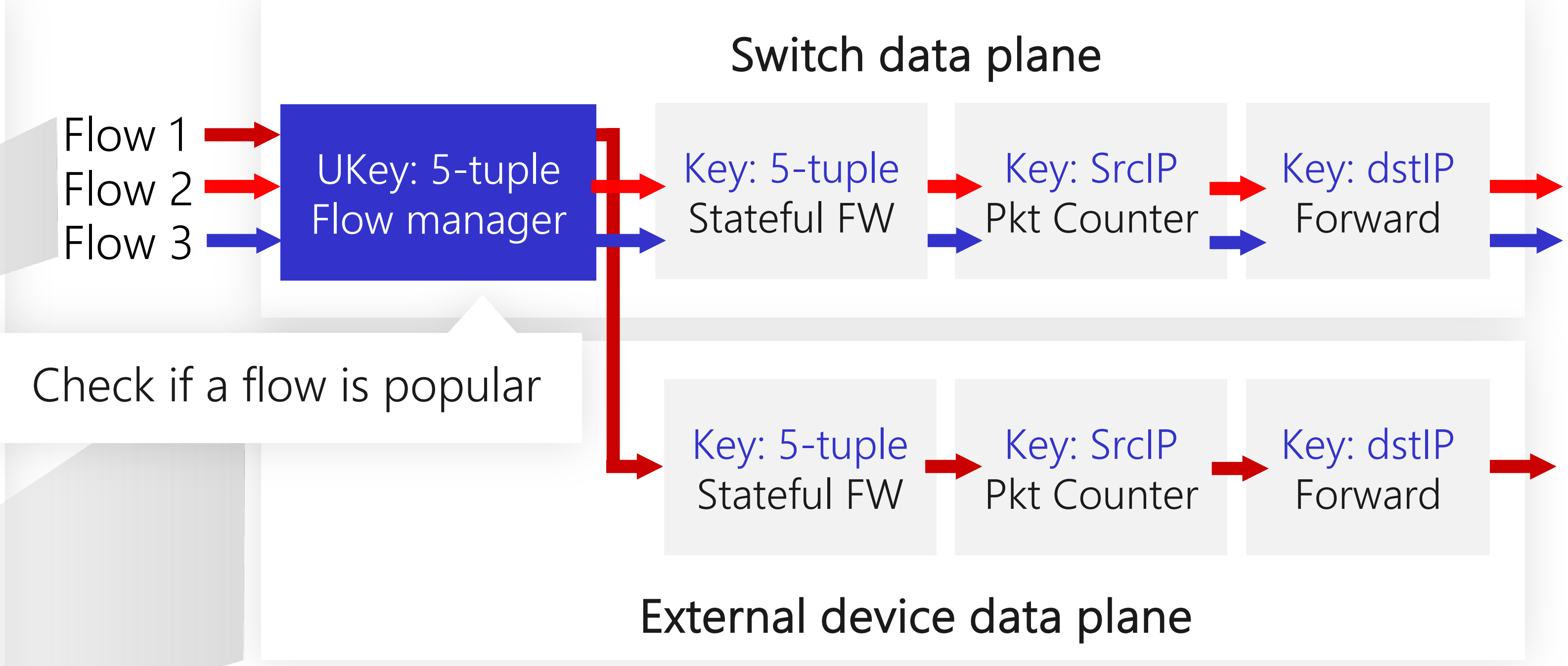


How to ensure that all necessary state is available on a device?

Our approach:
Packet pinning + Union key-based flow management

Union key: a union of key types of application objects



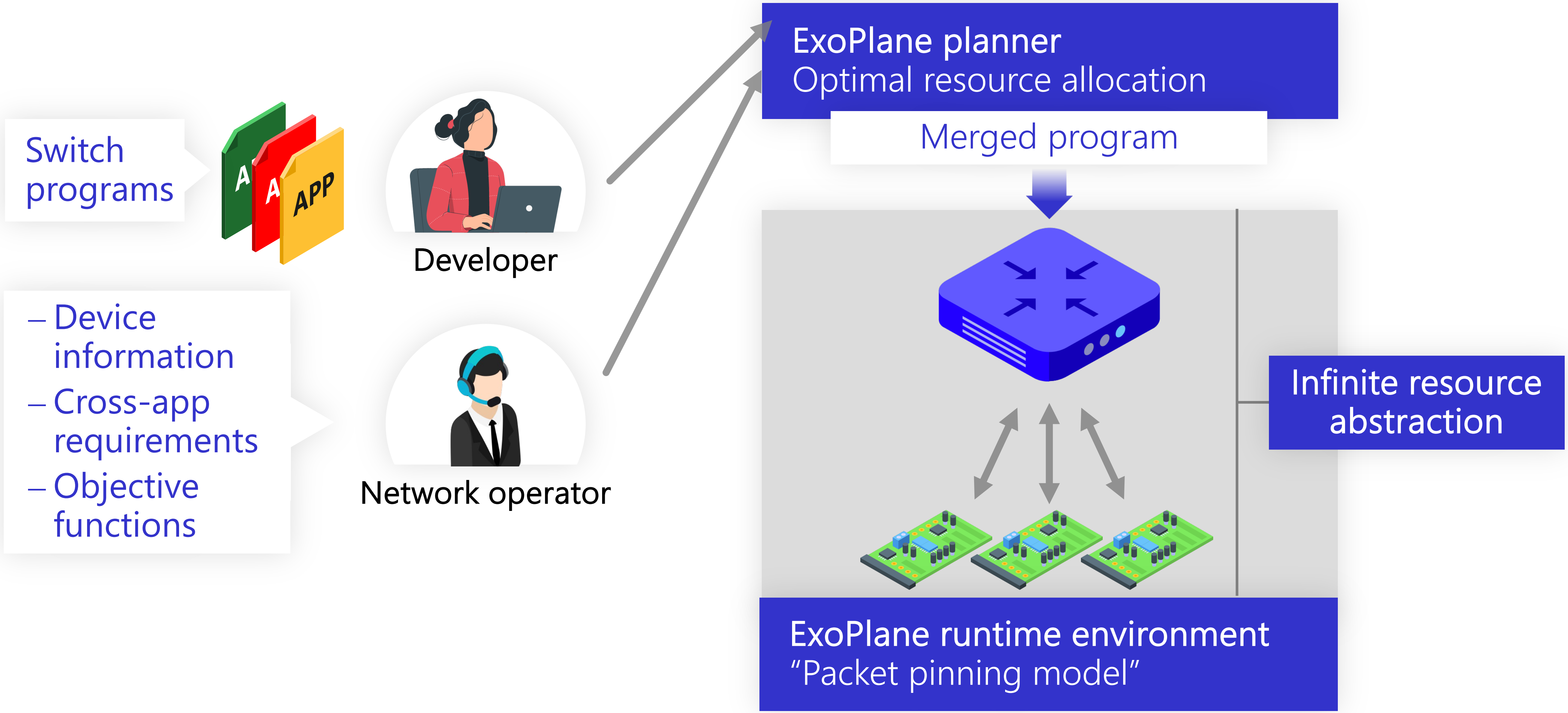


Key insight: Skewness of flow key distribution

E.g., 6% of flow keys takes 90% of total traffic

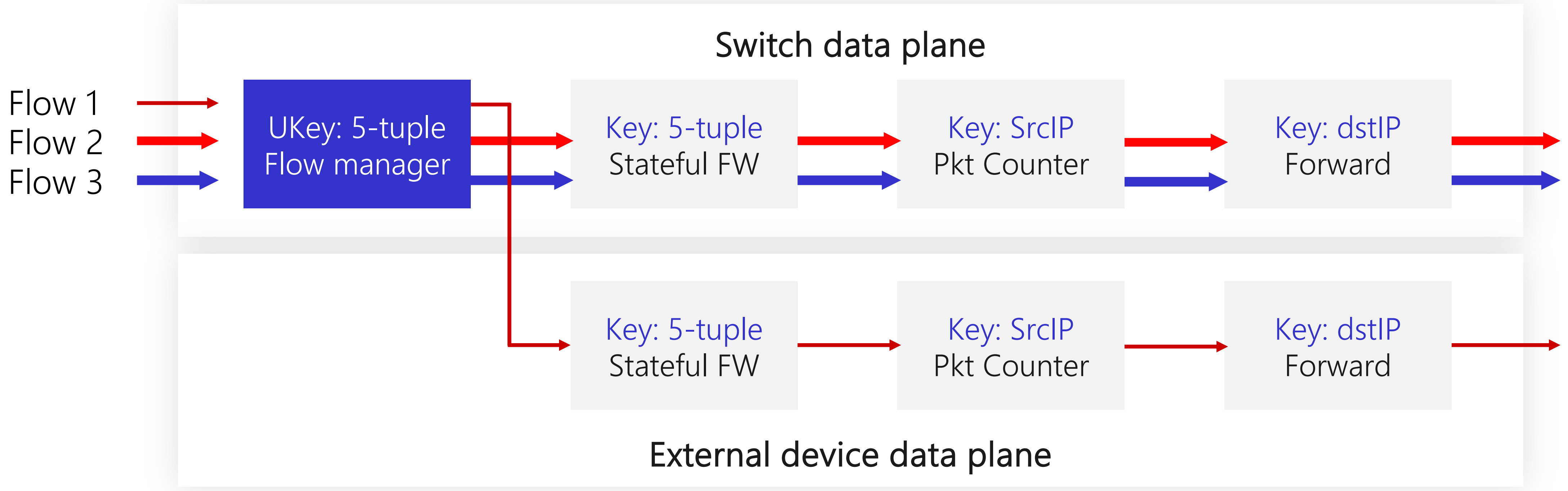
By placing popular keys on the switch, it can process most of the traffic while the remaining is processed at an external device

ExoPlane design overview



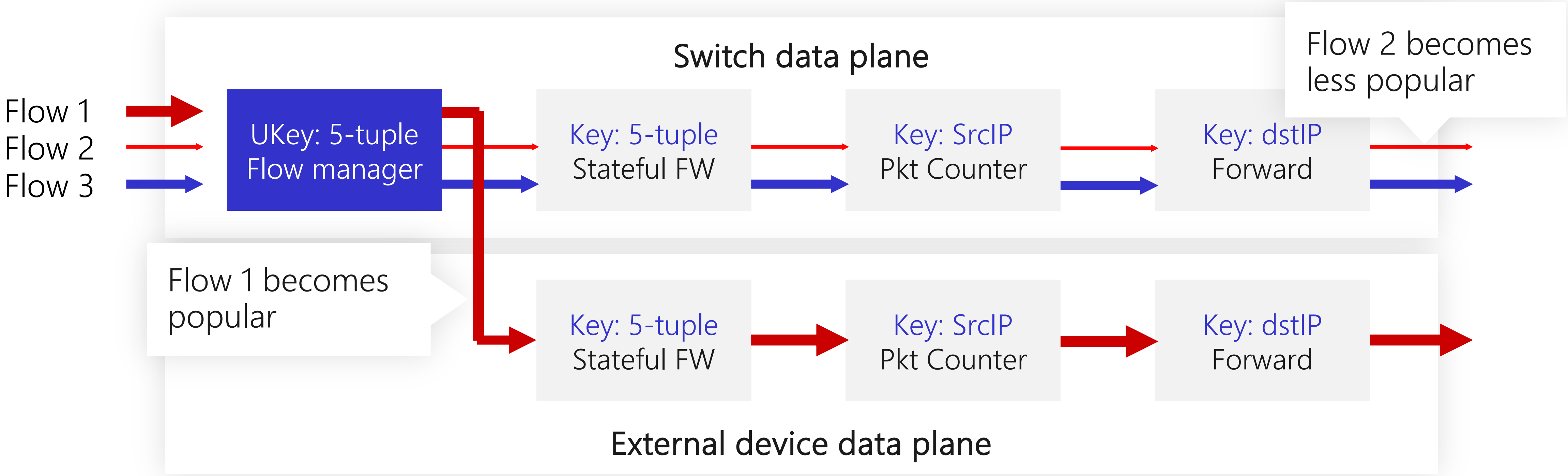
Challenge 1: Correctness under workload changes

1. New flows arrive → Insert entries of the flow
2. Flow popularity changes → Insert (evict) entries of popular (unpopular) flows

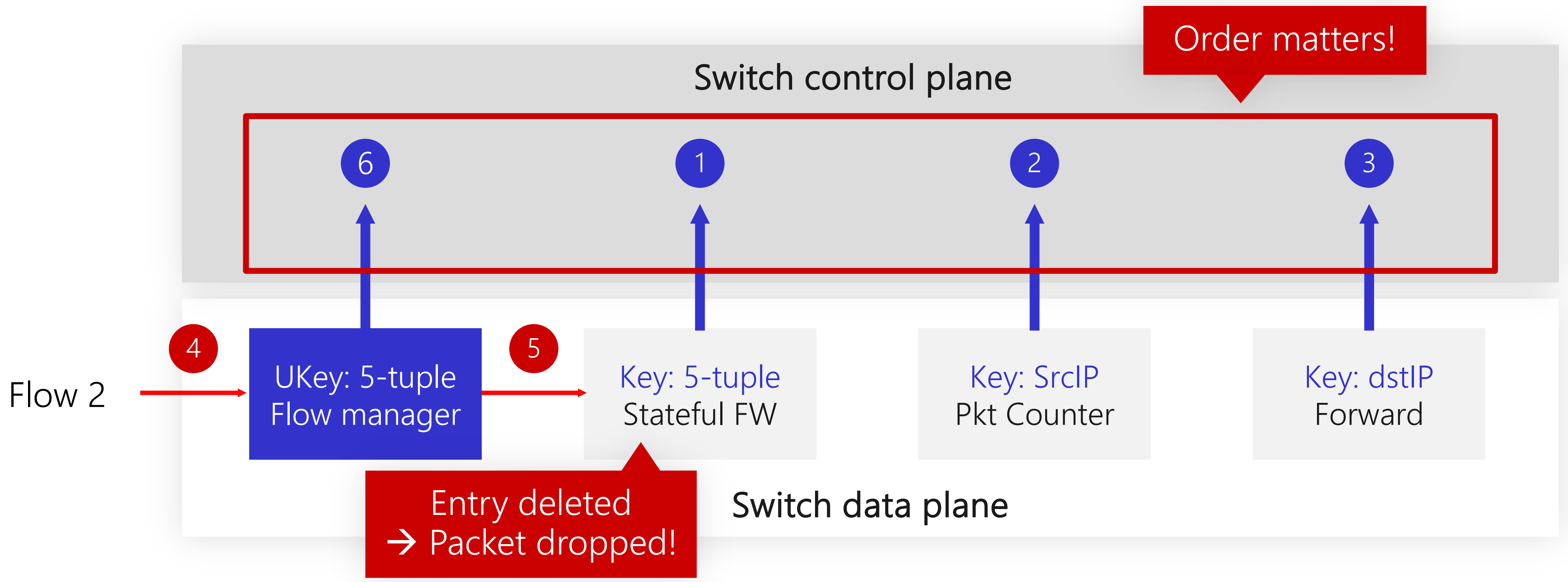


Challenge 1: Correctness under workload changes

1. New flows arrive → Insert entries of the flow
2. Flow popularity changes → Insert (evict) entries of popular (unpopular) flows

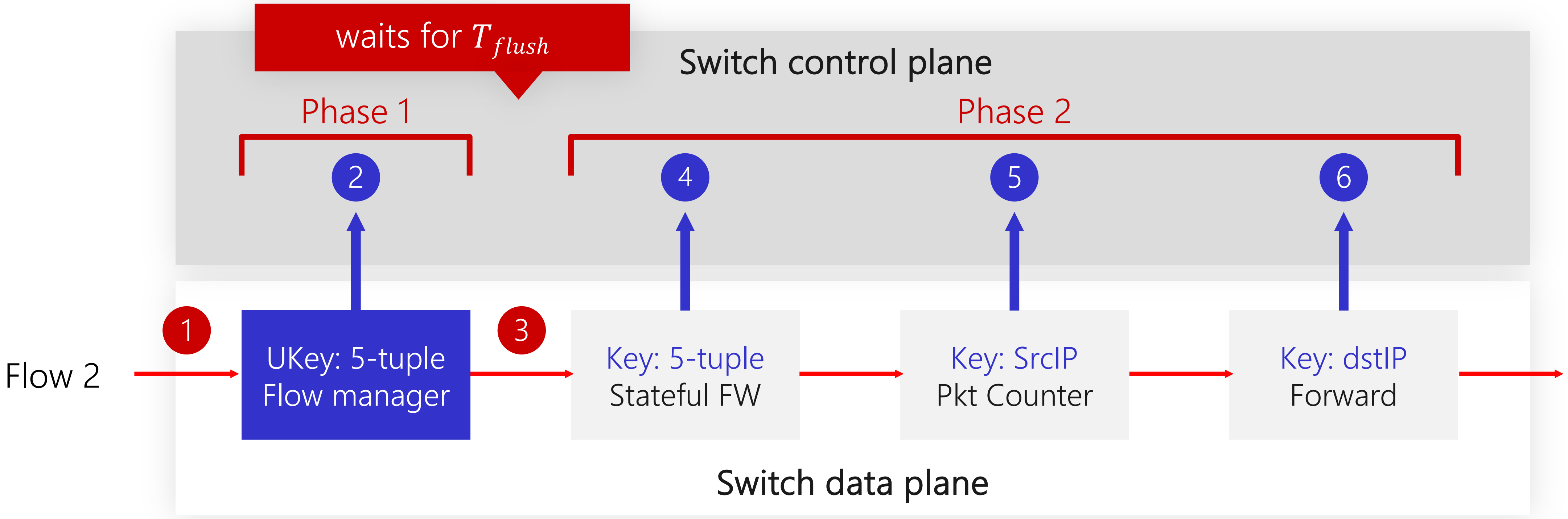


Problem: Incorrect state eviction



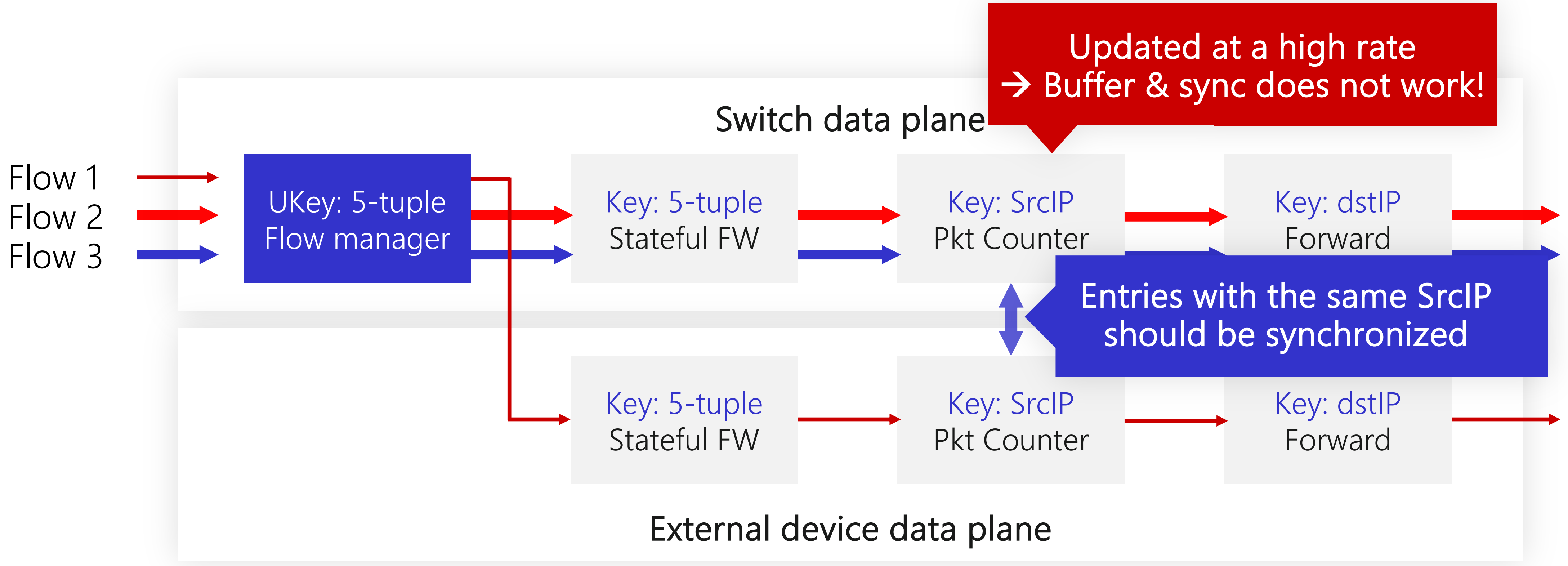
Similar issue can happen for insertion!

Our solution: Two-phase state update



Challenge 2: Synchronizing data plane-updatable states

Multiple copies of an object entry can be updated at different places

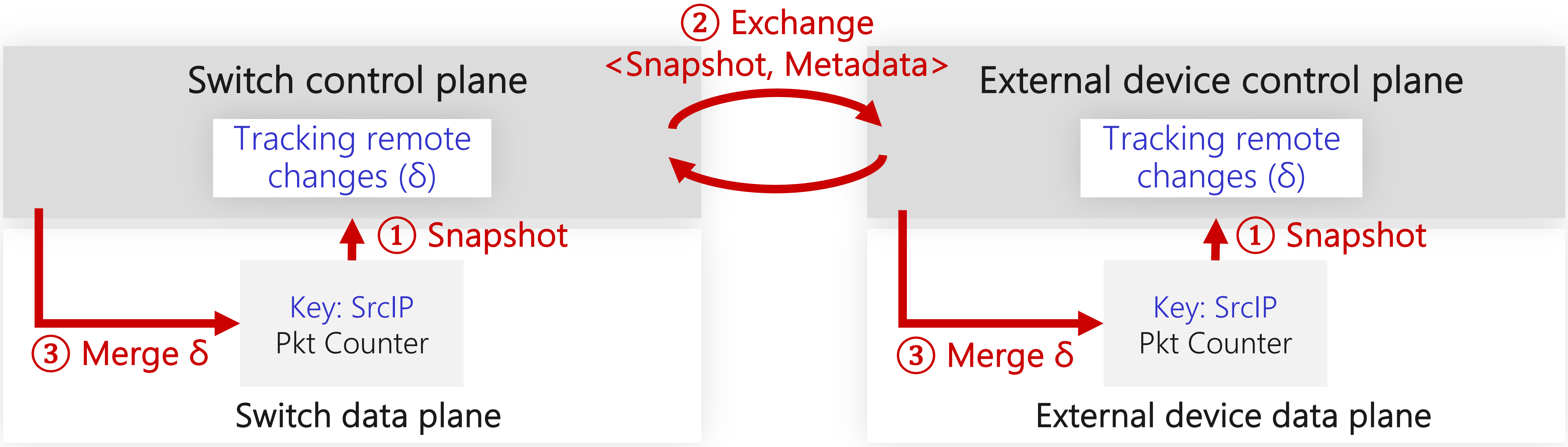


Bounded inconsistency via periodic synchronization

Observations on data plane-updatable state

- Approximate or statistical information
- Mergeable values

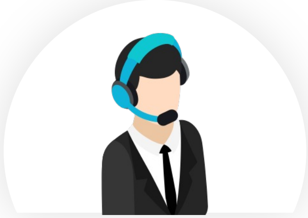
Our approach: bounded-inconsistency mode via periodic synchronization



Challenge 3: Meeting requirements across apps

App-specific requirements
(e.g., affinity to the switch)

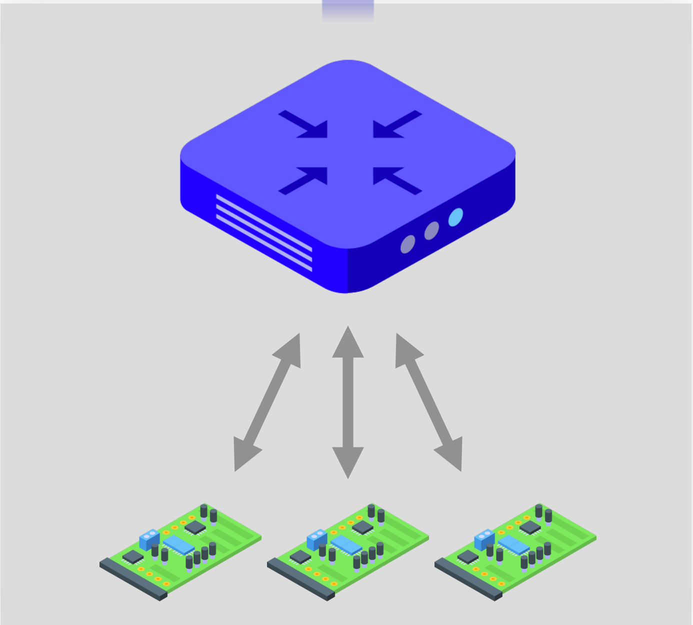
Developer



Network operator

– Cross-app requirements
– Objective functions

How to find an "optimal" resource allocation that satisfies all requirements?



Finding optimal resource allocation using ILP

– Switch program codes
– App-specific requirements

Developers

Network operator

– Device information
– Cross-app requirements
– Objective functions

Profiler

– Resource footprint
– Packet processing latency
– Compatibility matrix

Optimal resource allocation

Encode & solve resource allocation ILP

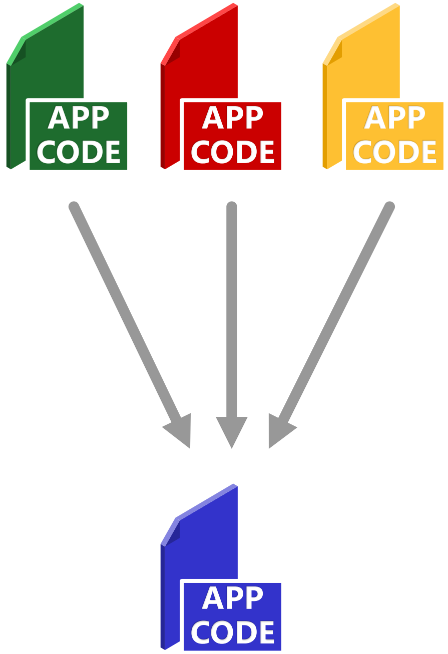
Objective:

Min. Expected Latency

Subject to:

– Resource constraint
– Compatibility constraint
– Workload assignment

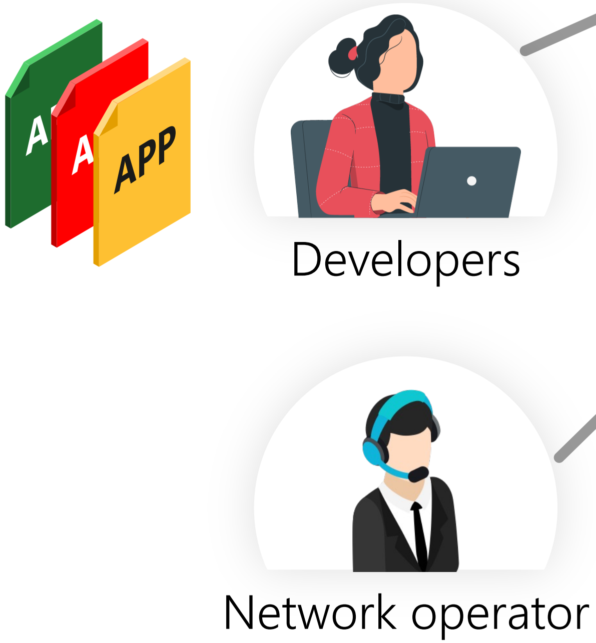
App merger

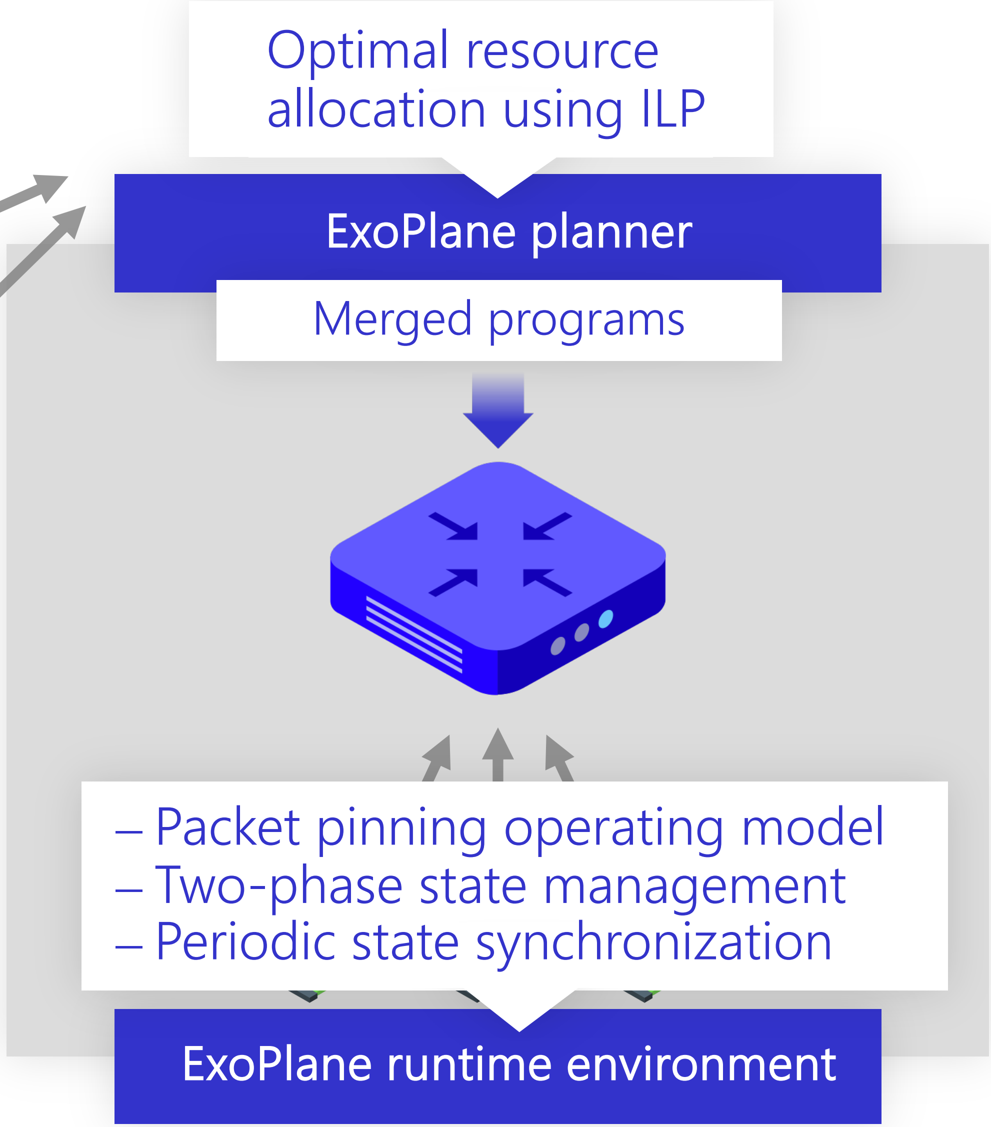


Loaded to the switch and external devices

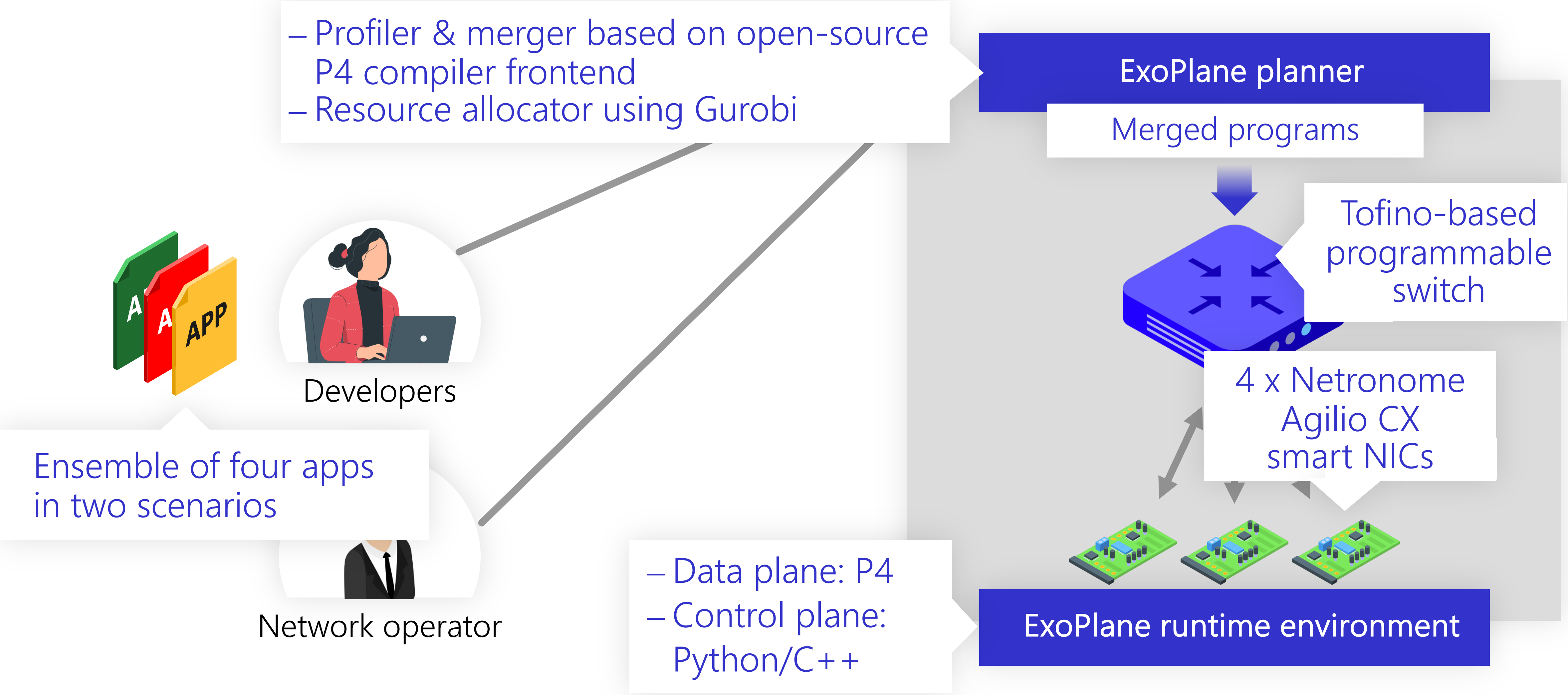
Putting it all together

ExoPlane provides an **infinite resource abstraction** to applications

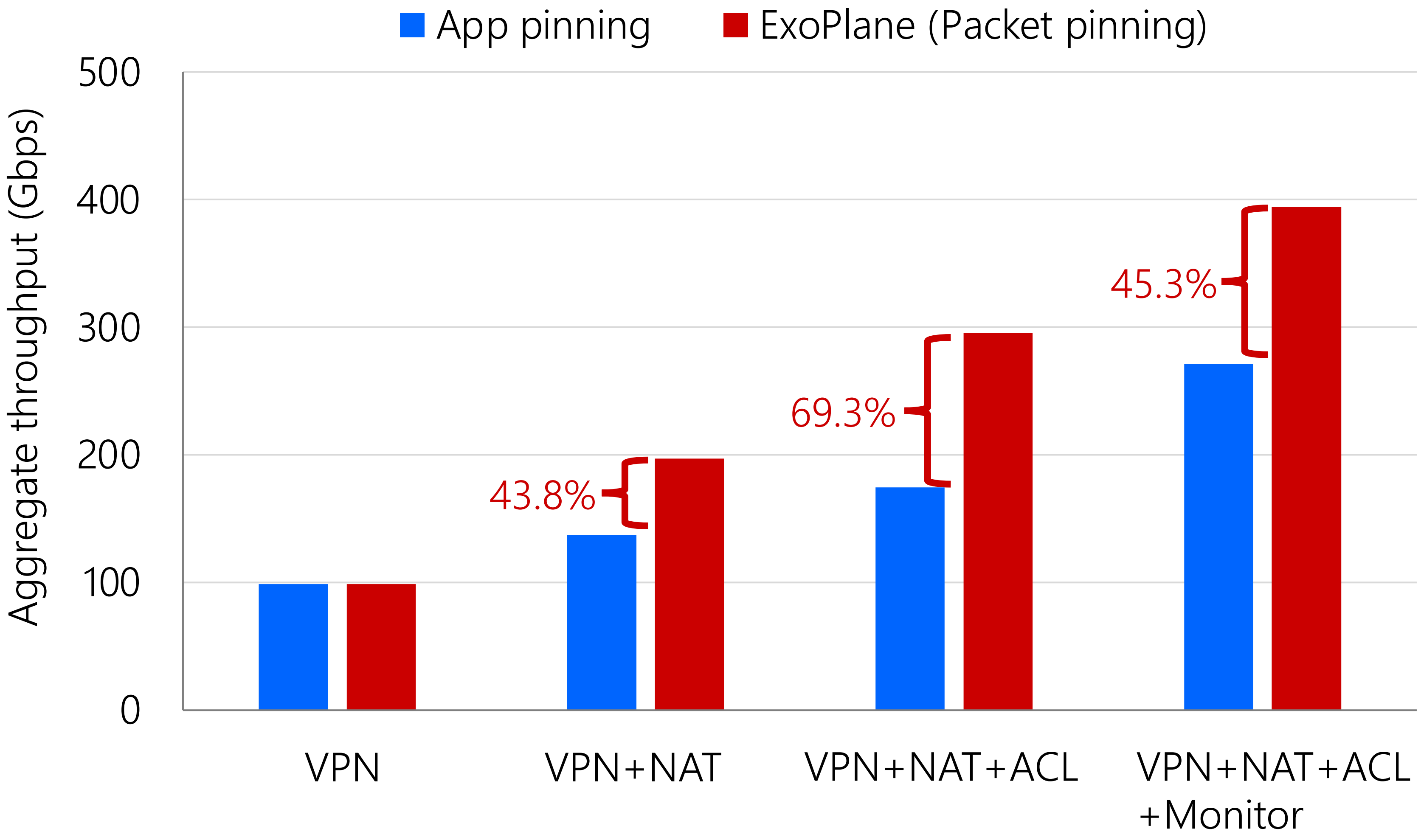




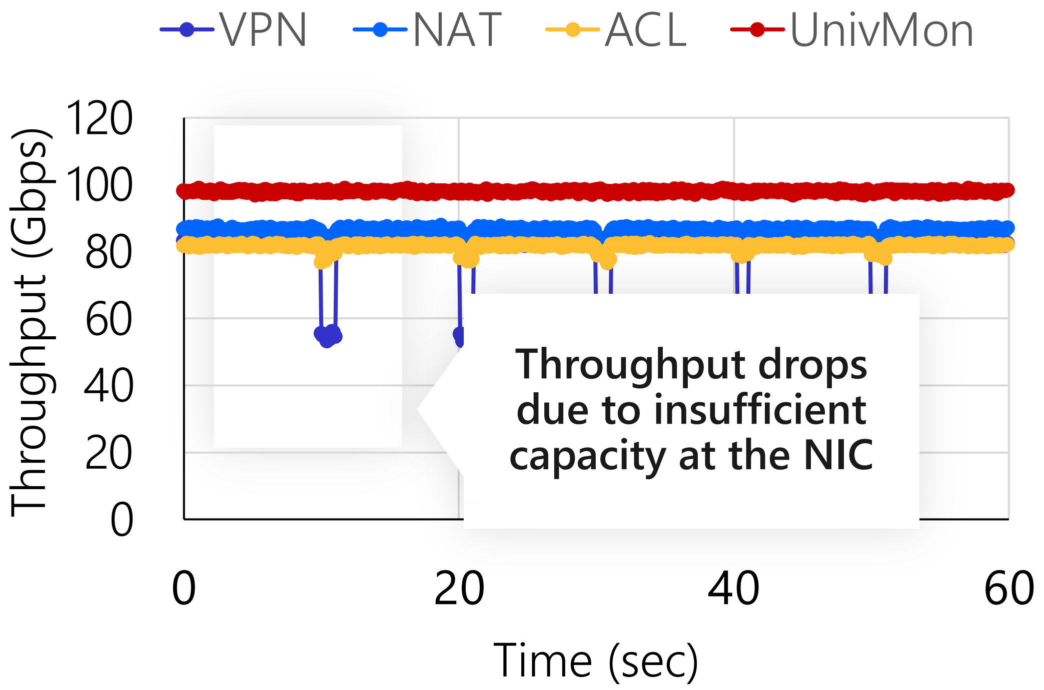
Implementation and evaluation setup



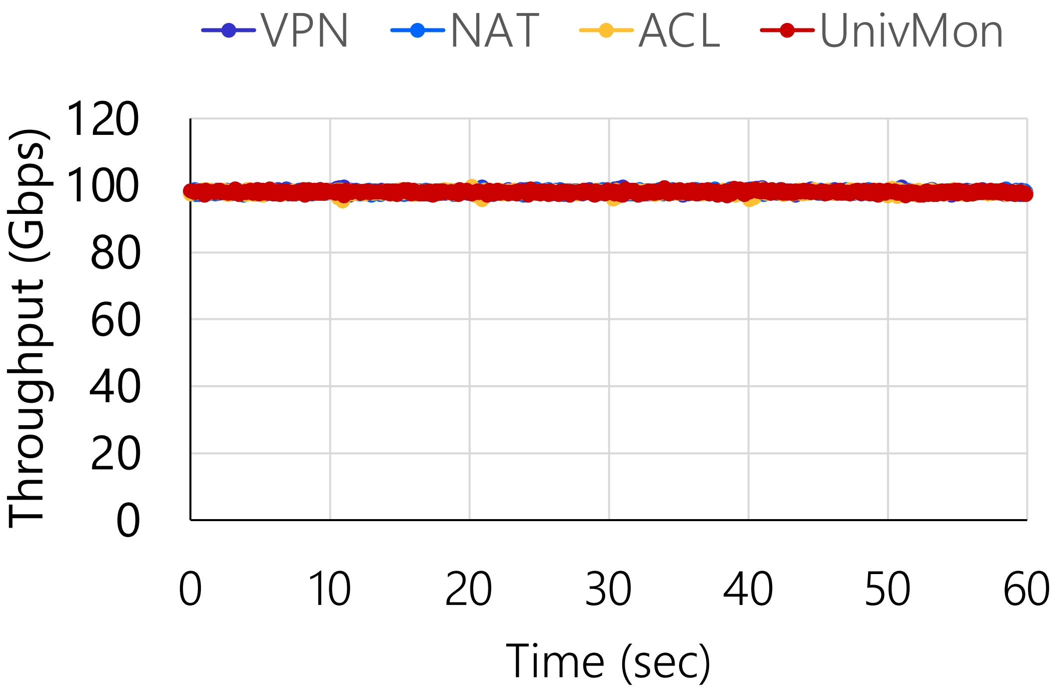
Does packet-pinning model work well?



How does ExoPlane work under dynamic workload?



Switch + a single external device



Switch + 4 x external devices

Limitations and future work

Supporting non-P4 programmable external devices

Supporting other types of resources on external devices

Enabling rapid runtime resource reallocation

What-if analysis of benefits from resource augmentation

Summary

Limited on-chip resources prevent concurrent stateful apps on programmable switches

---

ExoPlane provides OS abstractions for switch resource augmentation

- Packet pinning operating model
- Two-phase state management
- Periodic state synchronization
- Optimal resource allocation using ILP

Realizes resource augmentation with minimal performance and resource overhead

- Effectiveness of the packet pinning model
- Adapt to workload changes
- Low and predictable per-packet processing latency