

# The Design and Implementation of Open vSwitch

Ben Pfaff\*

Justin Pettit\*

Teemu Koponen\*

Ethan J. Jackson\*

Andy Zhou\*

Jarno Rajahalme\*

Jesse Gross\*

Alex Wang\*

Jonathan Stringer\*

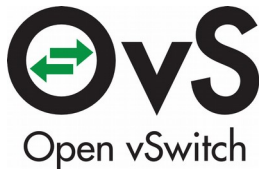
Pravin Shelar\*

Keith Amidon†

Martin Casado\*

\*VMware

†Awake Networks



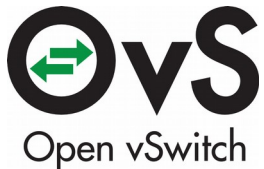
# What is Open vSwitch?

From [openvswitch.org](http://openvswitch.org):

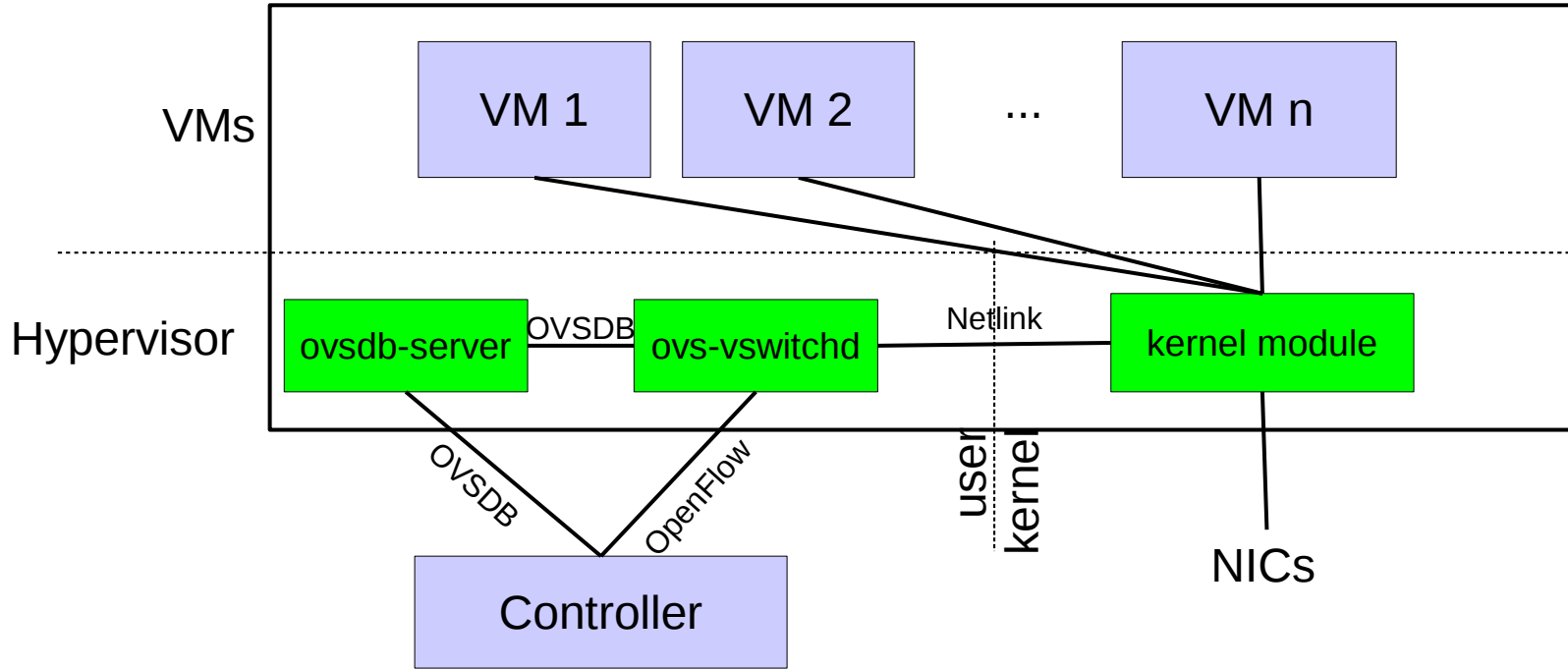
“Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).”

# Where is Open vSwitch Used?

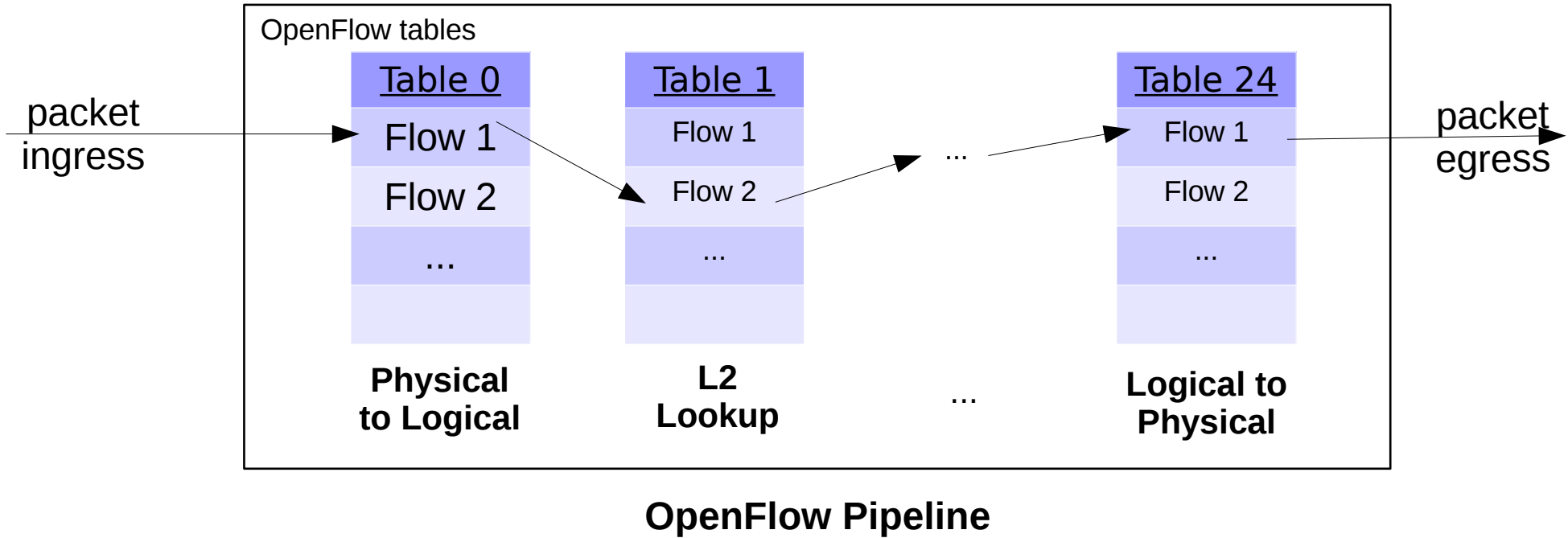
- Broad support:
  - Linux, FreeBSD, NetBSD, Windows, ESX
  - KVM, Xen, Docker, VirtualBox, Hyper-V, ...
  - OpenStack, CloudStack, OpenNebula, ...
- Widely used:
  - Most popular OpenStack networking backend
  - Default network stack in XenServer
  - 1,440 hits in Google Scholar
  - Thousands of subscribers to OVS mailing lists



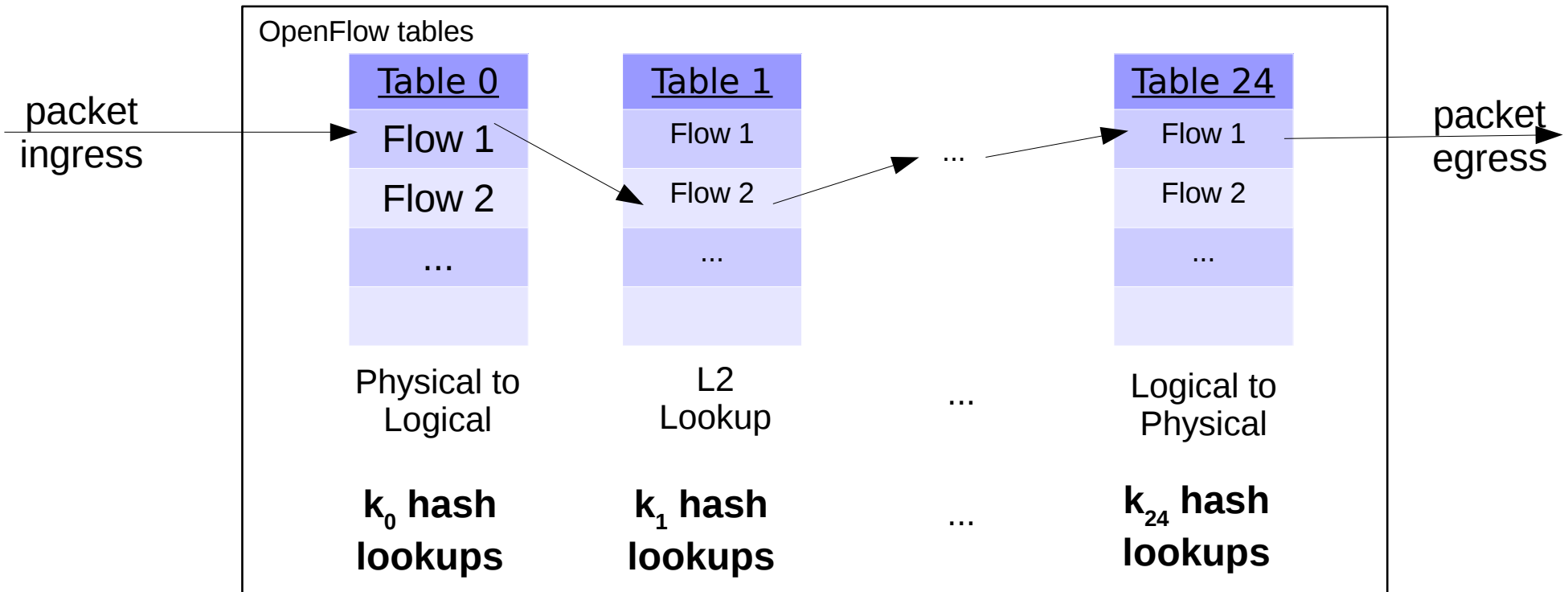
# Open vSwitch Architecture



# Use Case: Network Virtualization



# Implications for Forwarding Performance



100+ hash lookups per packet for tuple space search?

# Non-solutions

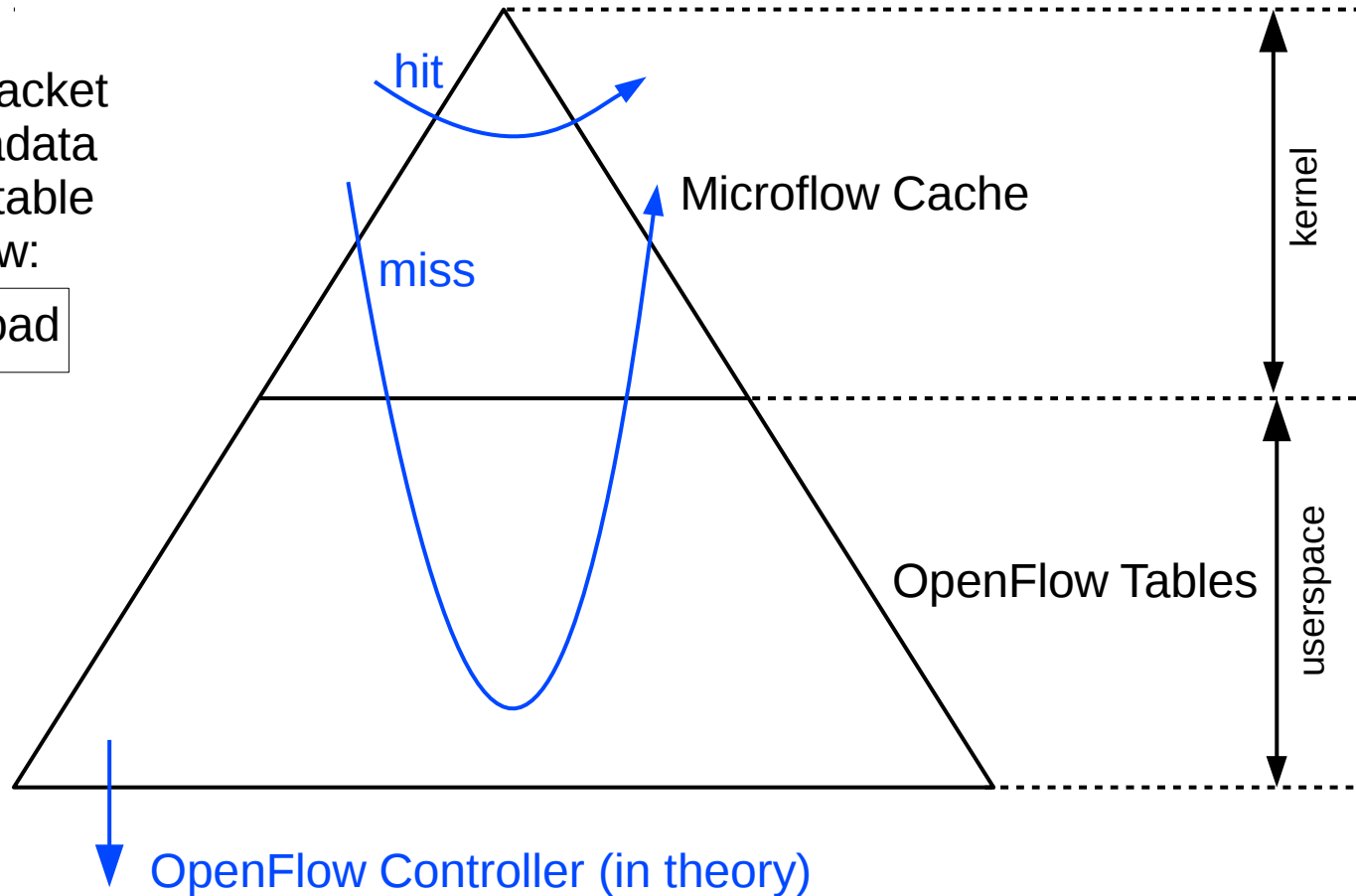
- All of these helped:
  - Multithreading
  - Userspace RCU
  - Batching packet processing
  - Classifier optimizations
  - Microoptimizations
- None of it helped enough: % versus x.

**Classification is expensive on general-purpose CPUs!**

# OVS Cache v1: Microflow Cache

## Microflow:

- Complete set of packet headers and metadata
- Suitable for hash table
- Shaded data below:



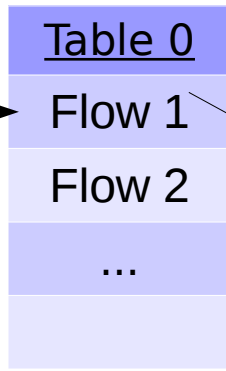


# Speedup with Microflow Cache

Microflow cache  
(1 hash lookup)

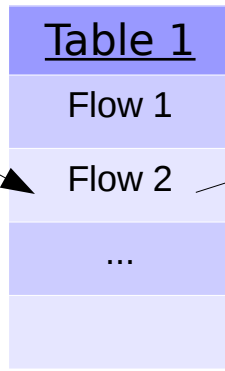
packet  
ingress

OpenFlow tables



Physical to  
Logical

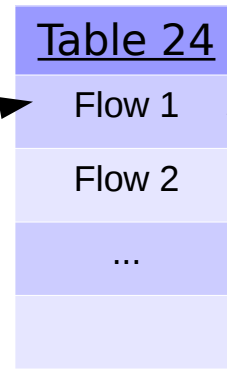
$k_0$  hash  
lookups



L2  
Lookup

$k_1$  hash  
lookups

...



Logical to  
Physical

$k_{24}$  hash  
lookups

packet  
egress

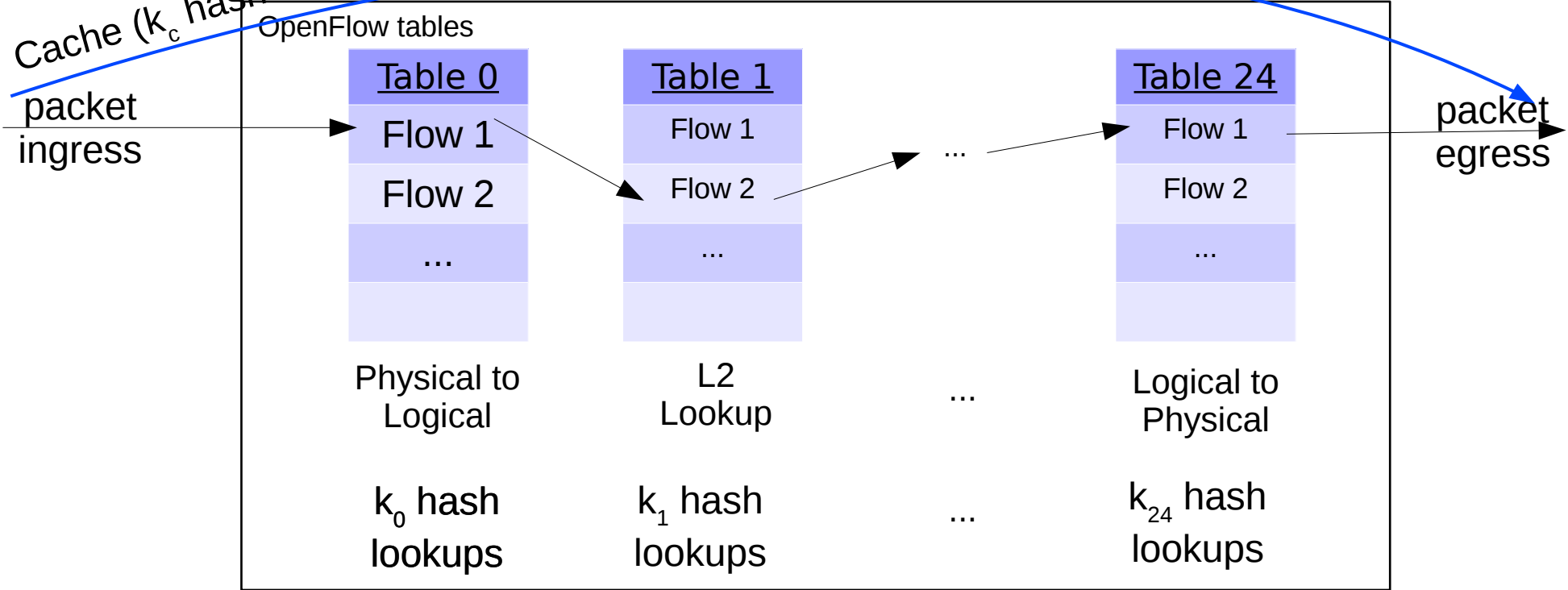
From 100+ hash lookups per packet, to just 1!

# Microflow Caching in Practice

- Tremendous speedup for most workloads
- Problematic traffic patterns:
  - Port scans
    - Malicious
    - Accidental (!)
  - Peer-to-peer rendezvous applications
  - Some kinds of network testing
- All of this traffic has lots of short-lived microflows
  - Fundamental caching problem: low hit rate

# Using a More Expensive Cache

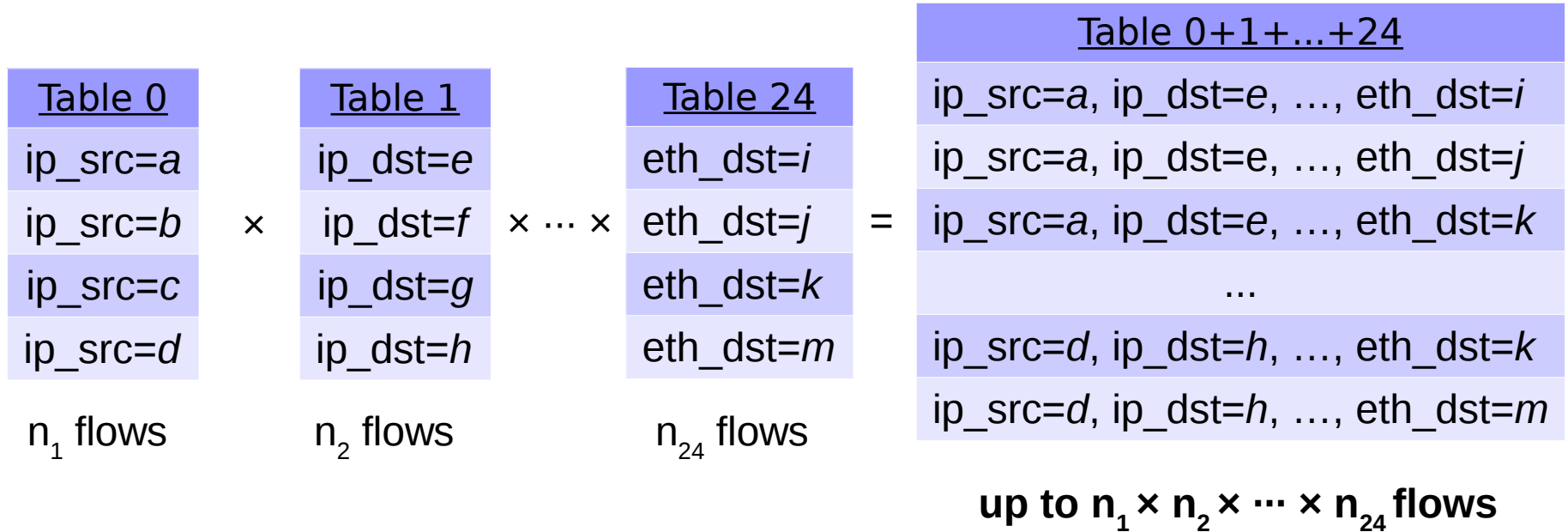
Cache ( $k_c$  hash lookups)



If  $k_c \ll k_0 + k_1 + \dots + k_{24}$ : benefit!

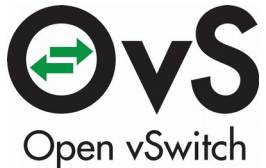
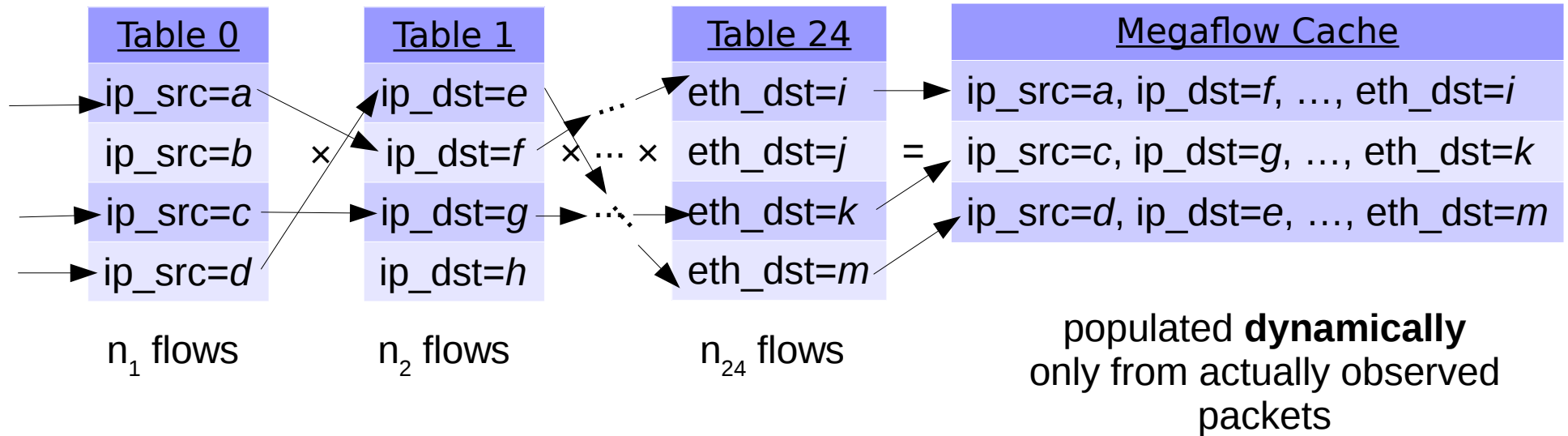
# Naive Approach to Populating Cache

Combine tables 0...24 into one flow table. Easy! Usually,  $k_c \ll k_0 + k_1 + \dots + k_{24}$ . But:



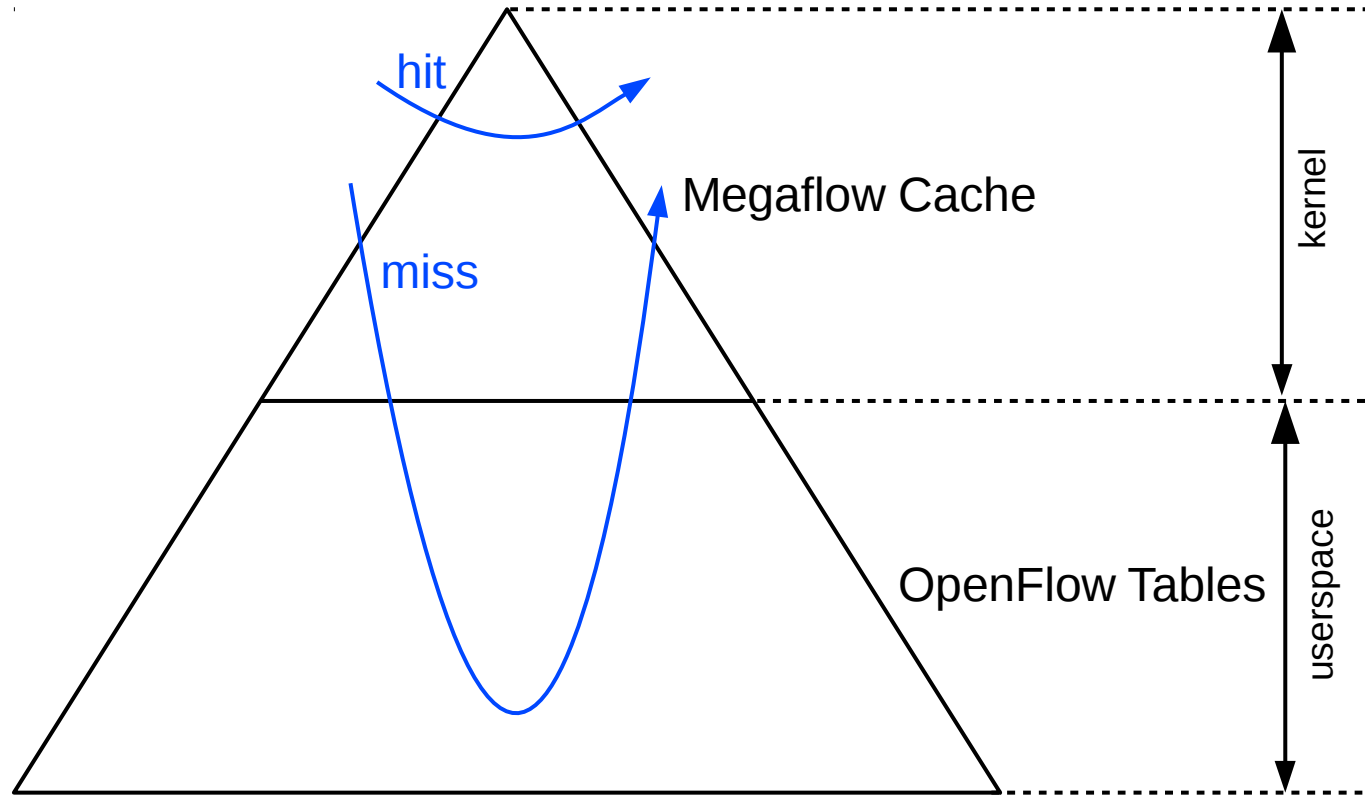
# Lazy Approach to Populating Cache

Solution: Build cache of combined “megafloWS” **lazily** as packets arrive.



**Same (or better!) table lookups as naive approach.  
Traffic locality yields practical cache size.**

# OVS Cache v2: “Megaflow” Cache



# Making MegafloWS Better

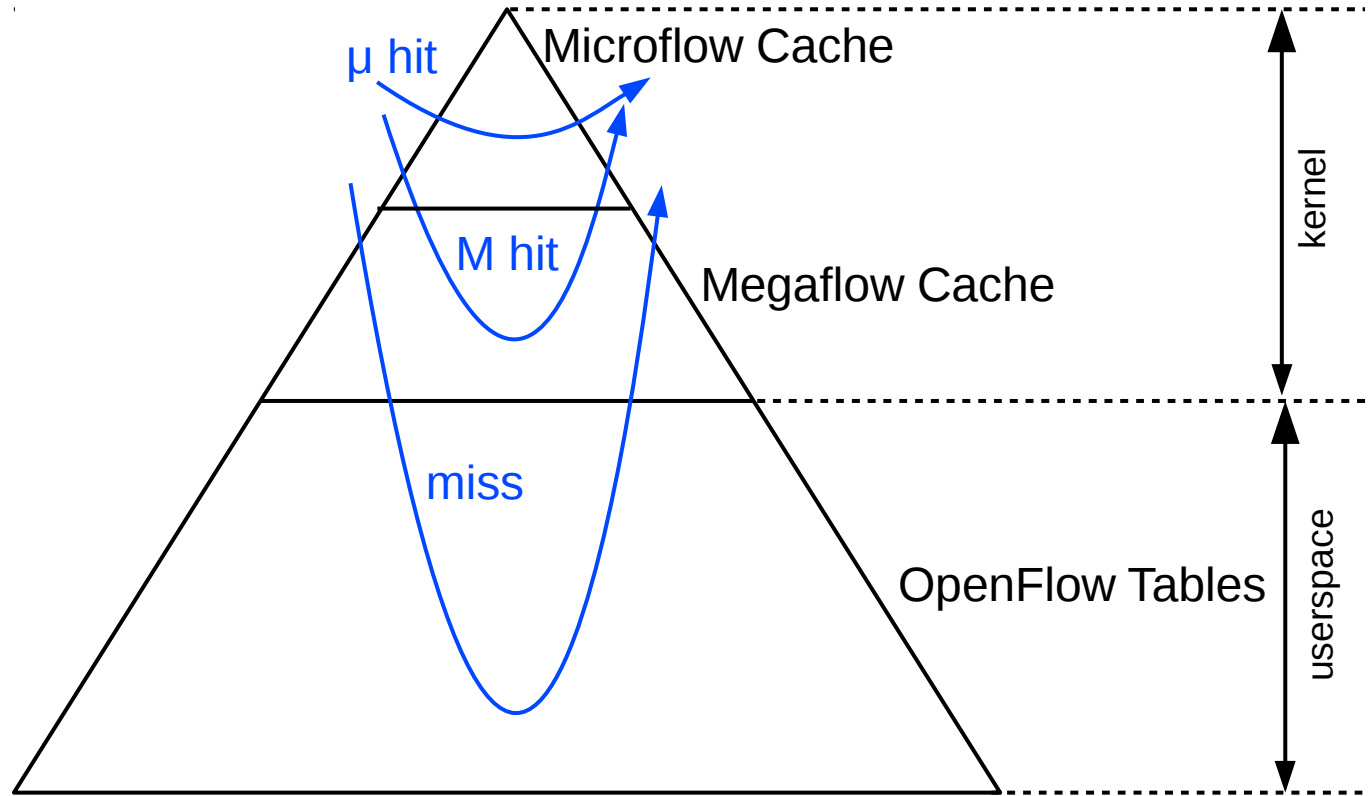
- MegafloWS are more effective when they match fewer fields.
  - MegafloWS that match TCP ports are almost like microfloWS!
  - Described approach matches every field that appears in any flow table
- Requirements:
  - online
  - fast
- Contribution: MegafloWS generation improvements (Section 5).

# Megaflow vs. Microflow Cache Performance

- Microflow cache:
  - $k_0 + k_1 + \dots + k_{24}$  lookups for first packet in microflow
  - 1 lookup for later packets in microflow
- Megaflow cache:
  - $k_c$  lookups for (almost) every packet
- $k_c > 1$  is normal, so megaflows perform worse in common case!
- Best of both worlds would be:
  - $k_c$  lookups for first packet in microflow
  - 1 lookup for later packets in microflow



# OVS Cache v3: Dual Caches



# Parting Thoughts

- Architectural tension: expressibility vs. performance
- OpenFlow is expressive but troublesome to make fast on x86
  - Performance requirements can make applications avoid OpenFlow
- Caching provides OVS with expressibility and performance
- Applications can freely evolve decoupled from performance
  - Specialized code would be **slower!**
- Starting from a more general problem produced better results