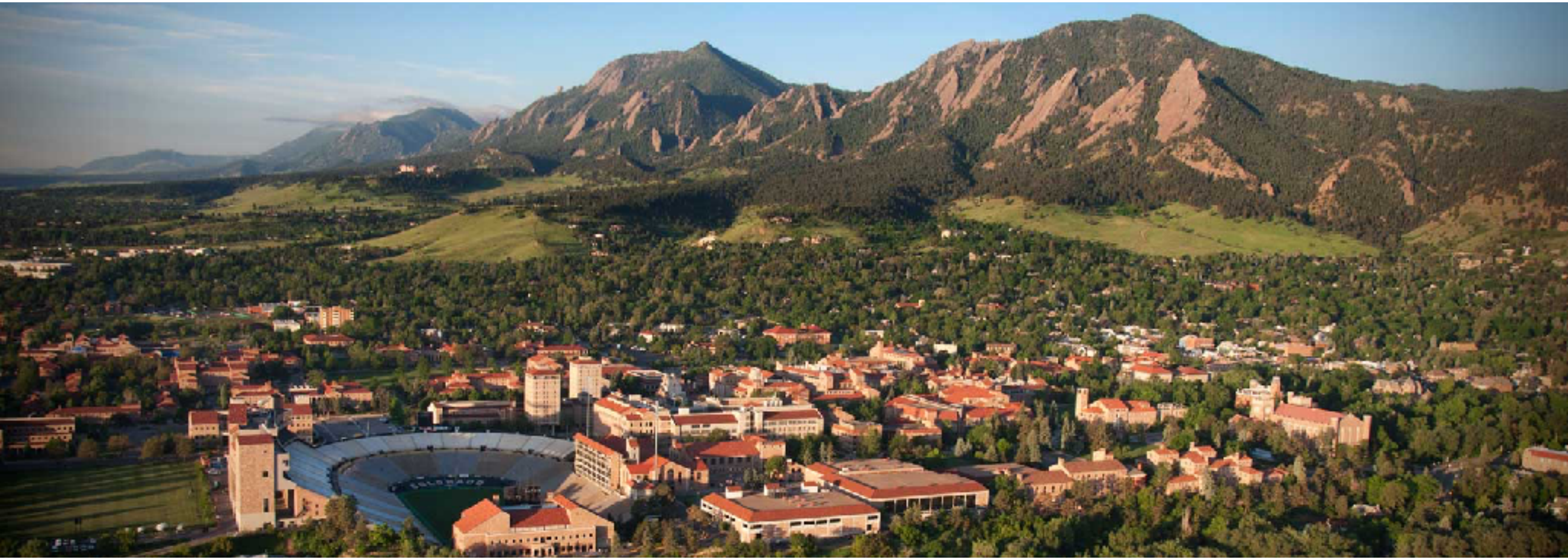
Packet-Level Network Analytics without Compromises

NANOG 73, June 26th 2018, Denver, CO

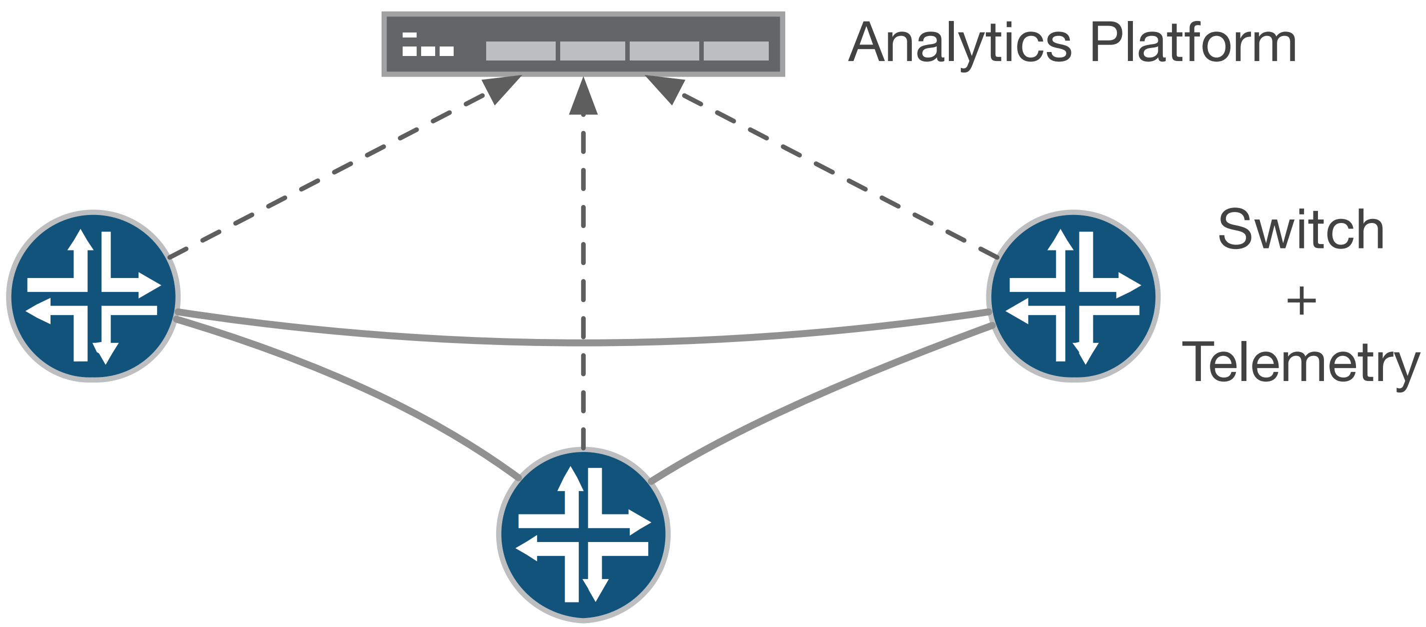


Oliver Michel

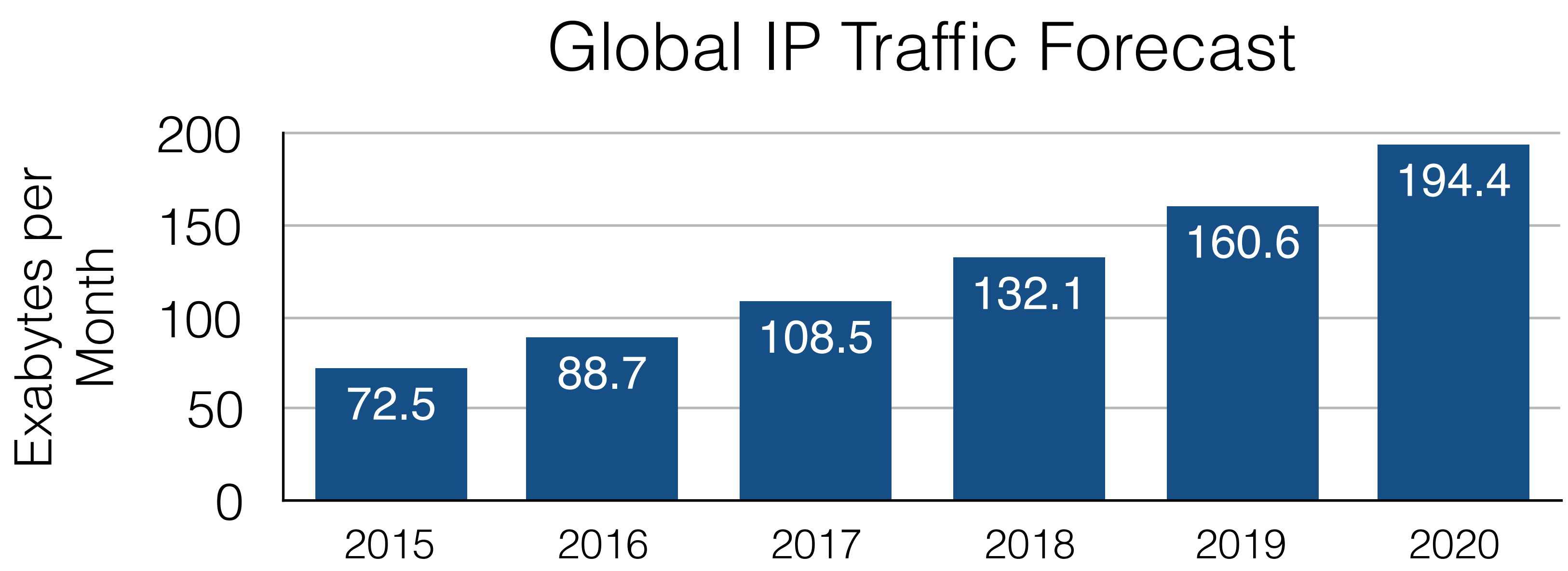
University of Colorado Boulder

Network monitoring is important

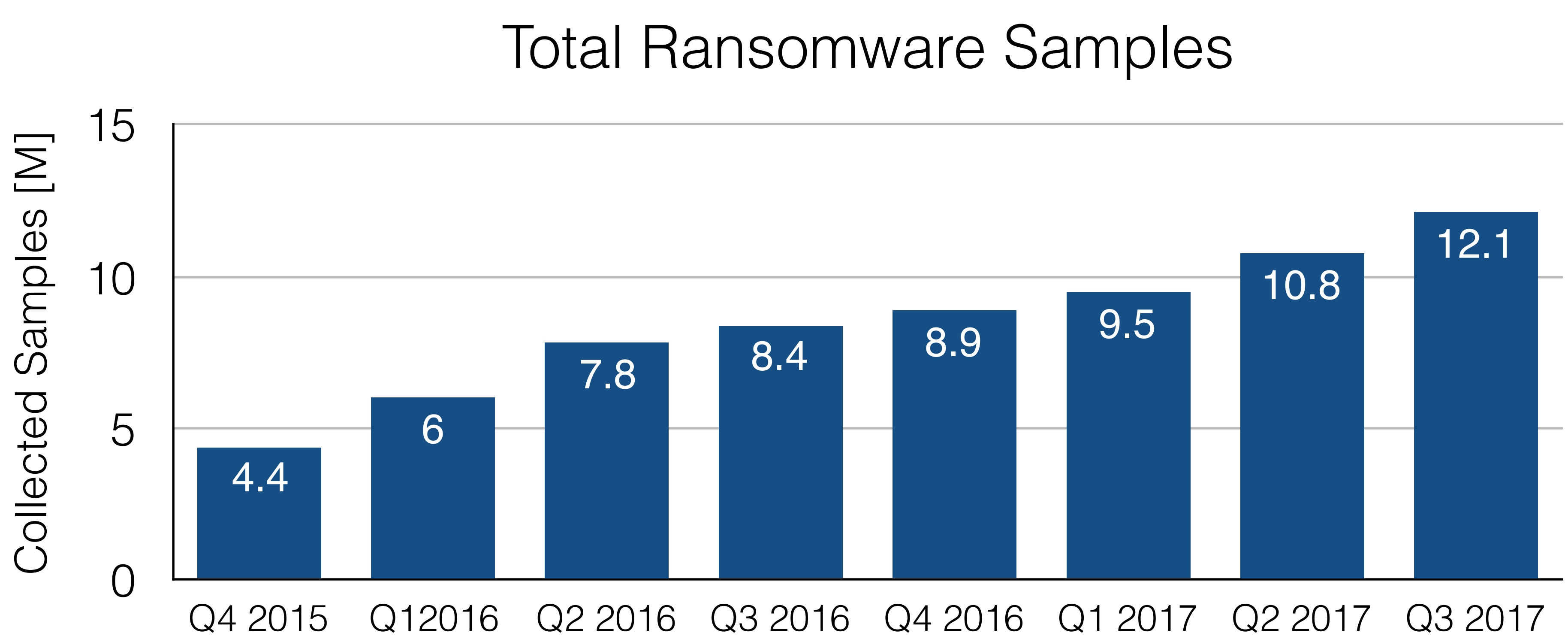
- Security issues
- Performance issues
- Equipment failure
- Misconfiguration



Network traffic and security threats grow rapidly

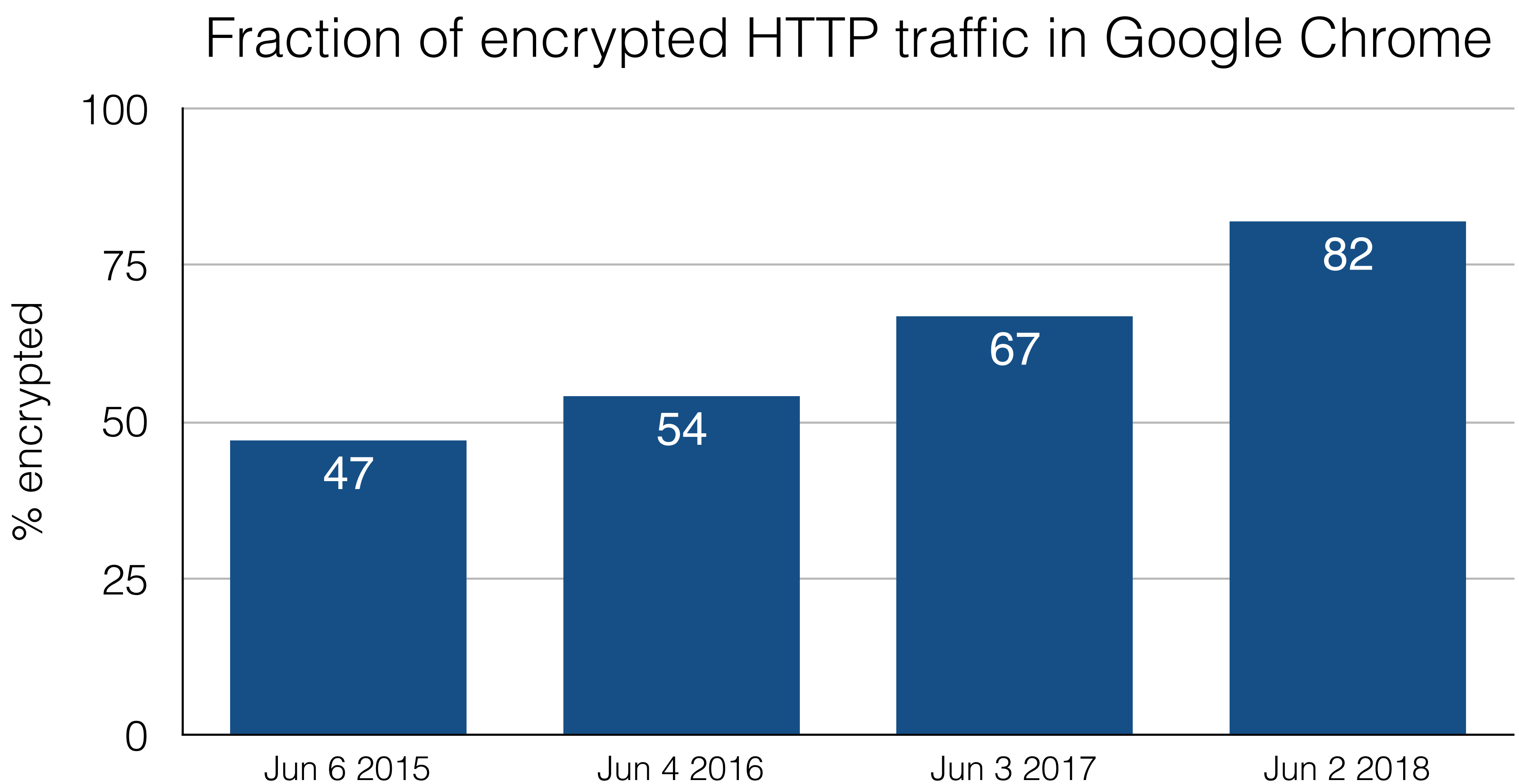


[Cisco Visual Networking Index 2017]



[McAfee Labs Thread Report Dec. 2017]

Traffic is commonly encrypted



[Google Transparency Report 2018]

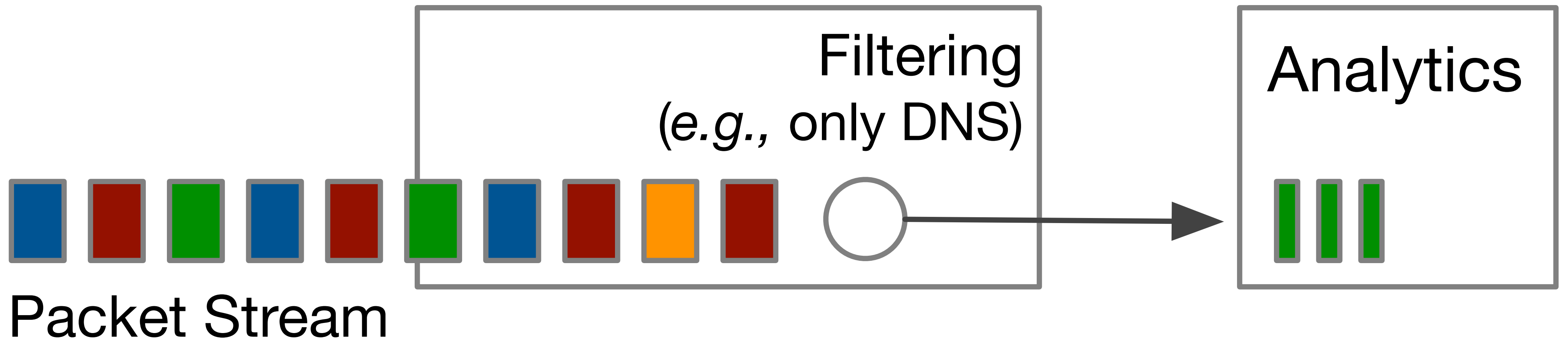
Network monitoring systems must match challenges

An ideal network monitoring system

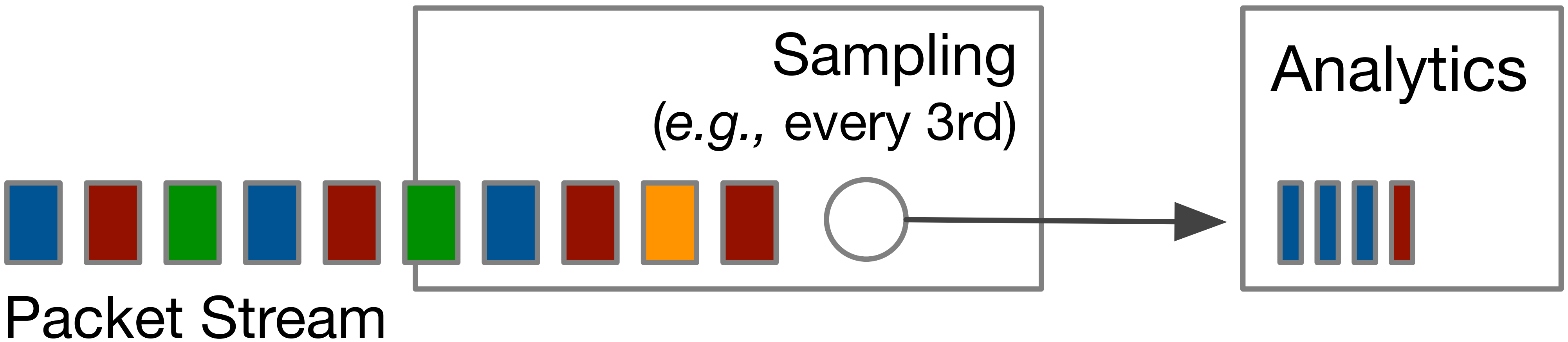
- record of every single packet
- full programmability
- DC scale performance

Existing systems make compromises

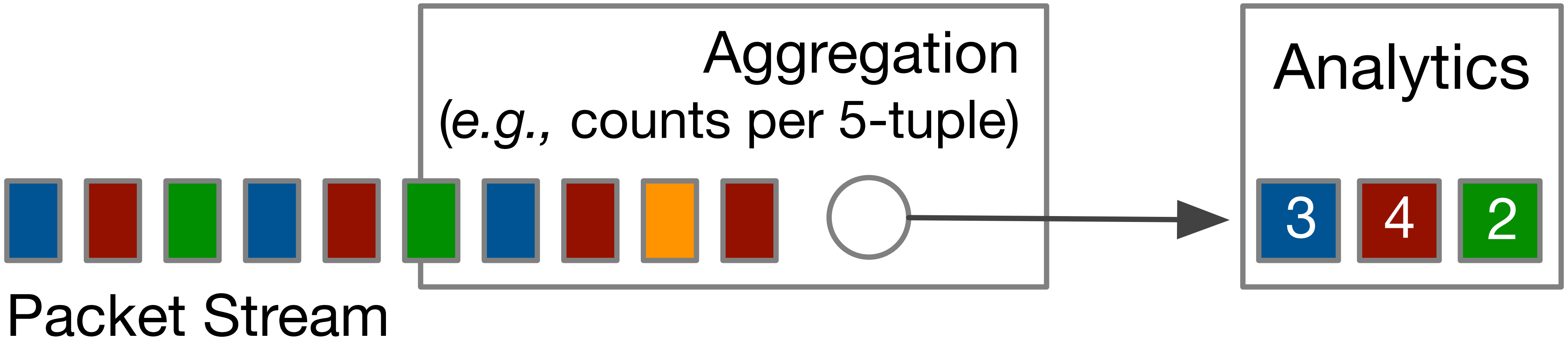
Filtering limits possible applications



Sampling can easily miss important packets

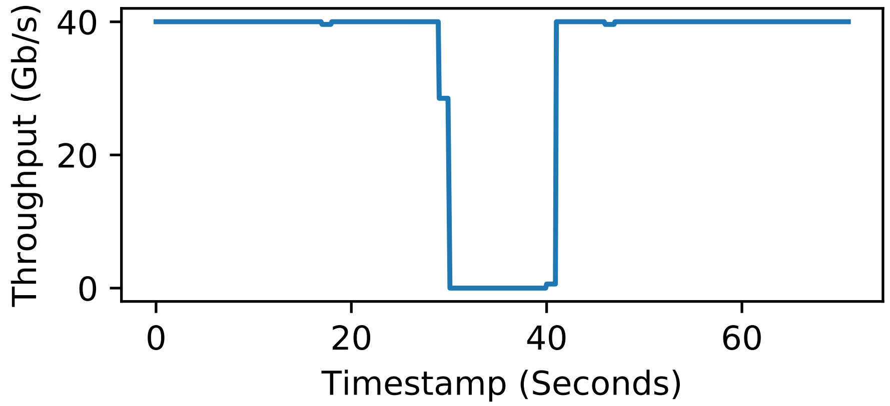


Aggregation limits information granularity and thus applications



Fixed hardware pipelines hinder expressiveness

Minimum downtime observed in 50 trials of reloading a Tofino PFE



Loss of information

Loss of capability

Why are these compromises made?

Case Study: Cisco Tetration for FB Data Center

Cisco Tetration-V:

- up to 200K flow events/s
- per instance requirements for Tetration-V ESXi: 128 CPU cores, 2TB RAM, 18TB storage
- 5 such servers for flow monitoring

Facebook web cluster (176 servers): 827K flows/s [roy. et. al. inside the social networks datacenter network 2015]

Is it possible to perform network analytics on cloud-scale infrastructures without compromises?

Two goals

Lossless telemetry at high rates	→	Flexible processing
~ 3 Tbit/s — 150M pps per-packet information		x86 / general purpose programming language runtime configurability ~ 10M pps per core
*flow	→	jetstream

Lossless telemetry at high rates

~ 3 Tbit/s — 150M pps

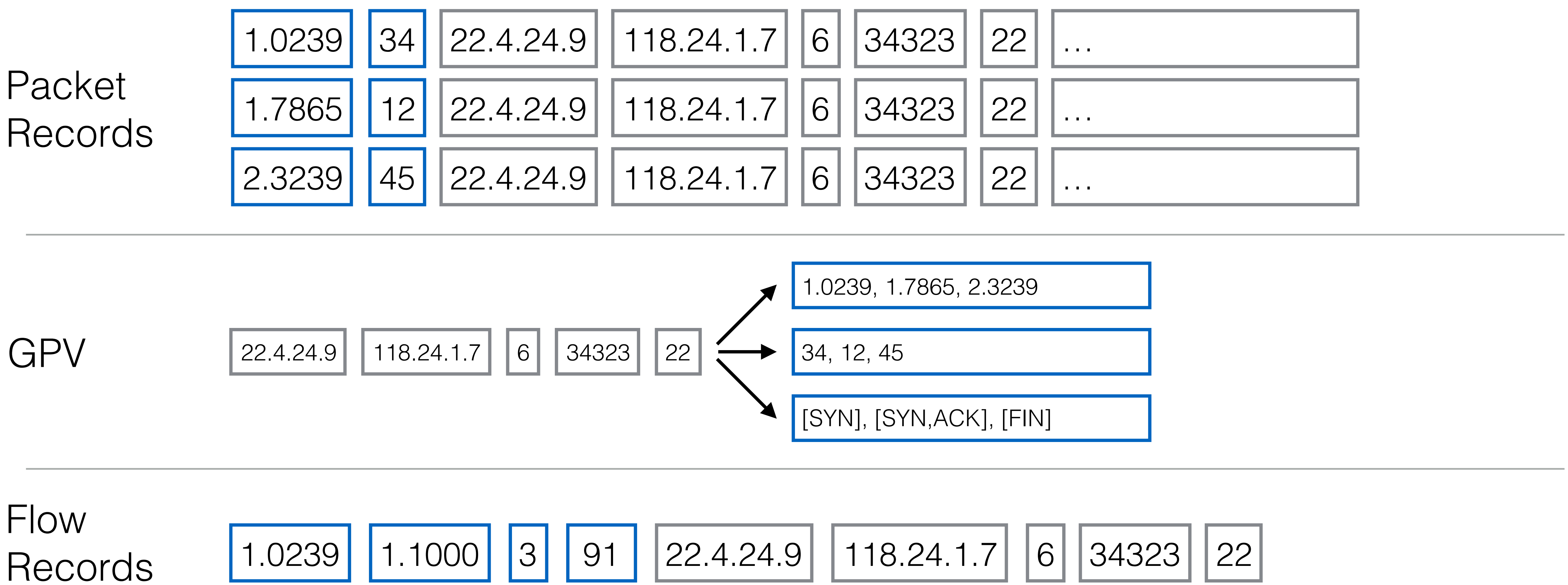
per-packet information

*flow

- Record format
- Hardware-assisted record generation

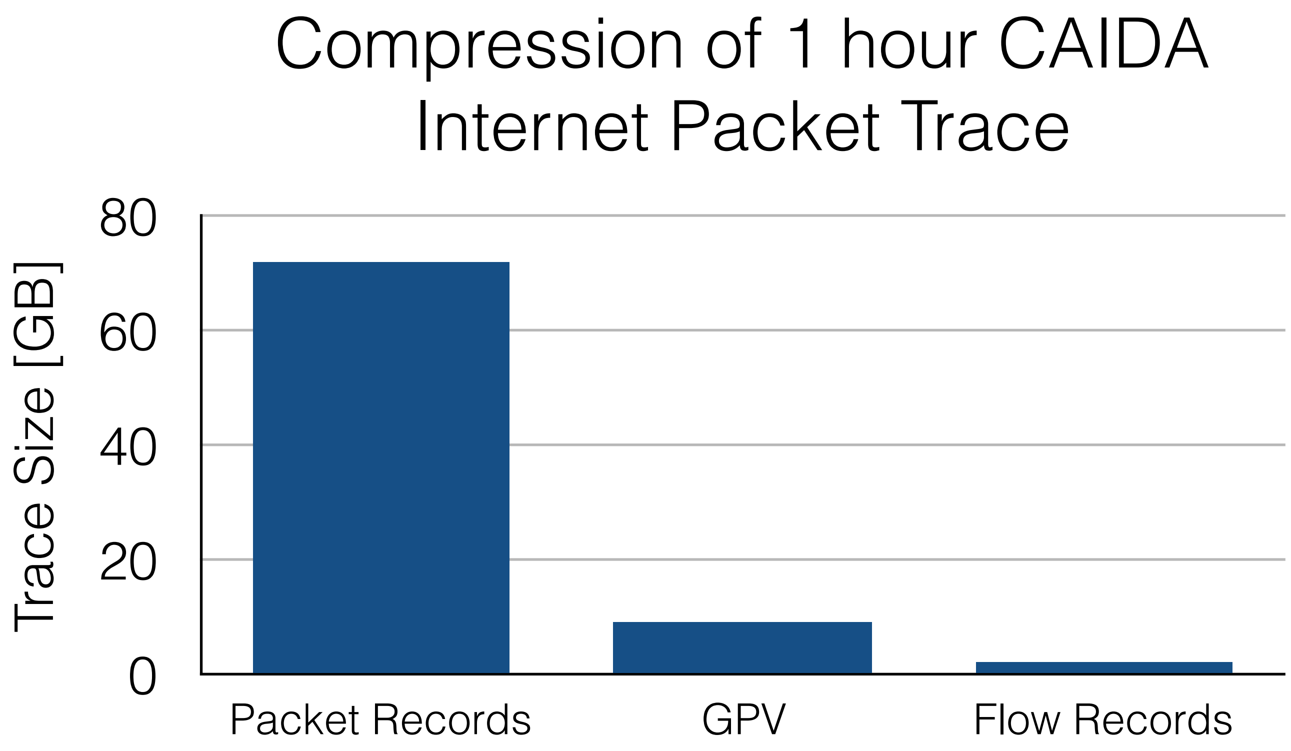
Grouped Packet Vectors (GPV)

- per-packet header fields
- meta data: *e.g.*, queue depth, ingress/egress timestamps



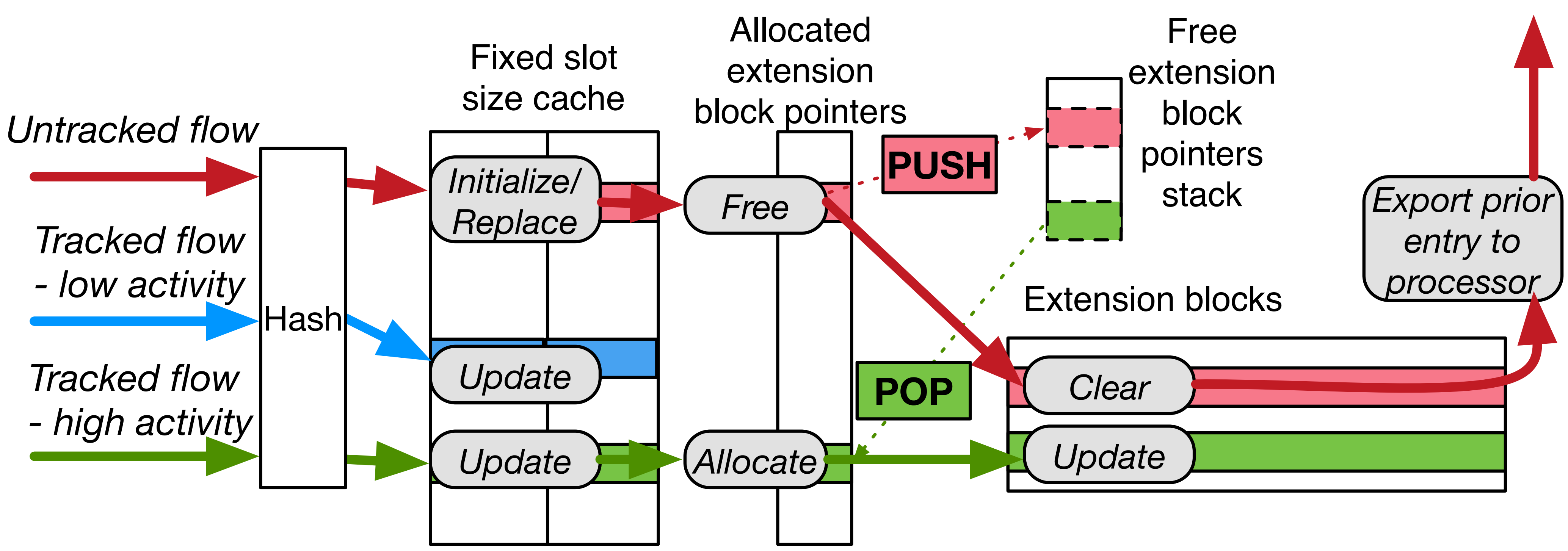
Grouped Packet Vectors (GPV)

- GPVs provide high compression while maintaining information richness

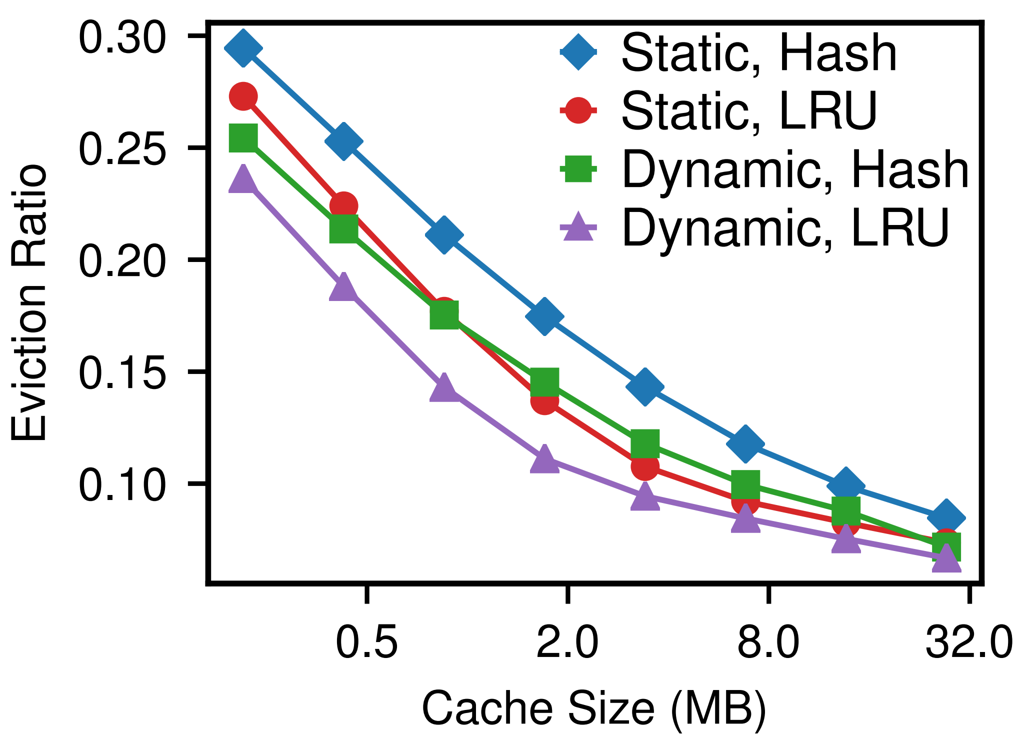


Generating GPVs at line rate

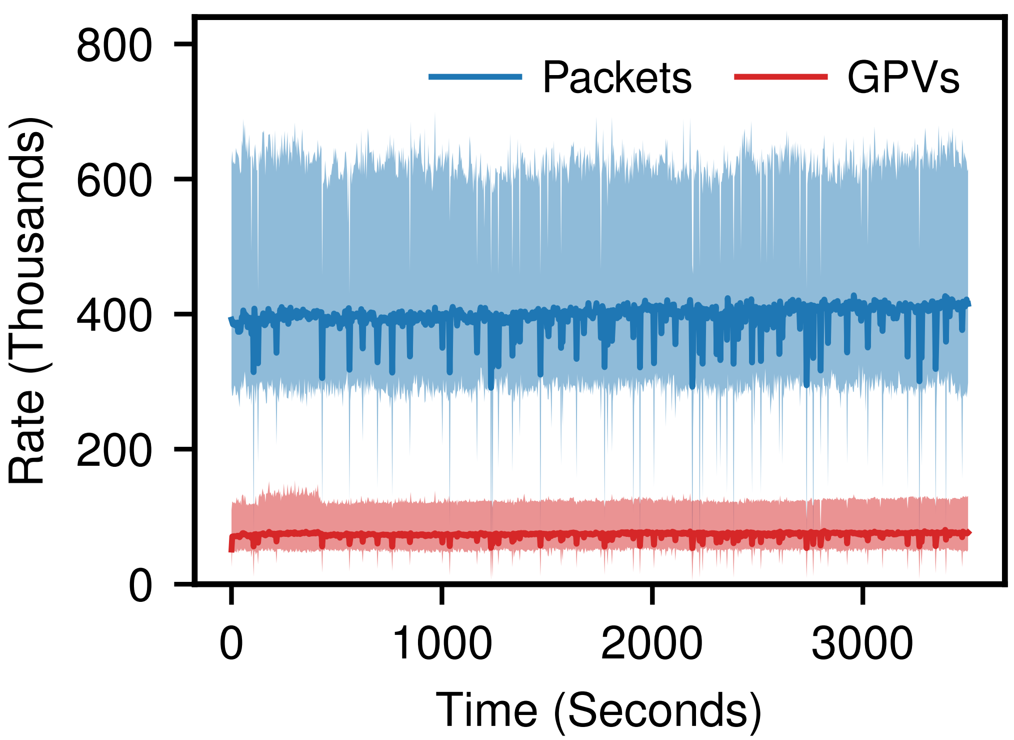
- Problem: GPVs have variable length, space is constrained
- Custom 2-level cache data structure
  1. Tall cache with narrow slots (many short flows)
  2. Small cache of wide slots (few long flows)



Resource usage



PFE memory vs. eviction rate



GPV eviction vs. packet rate

- Scalability
- Optimizations for packet record workloads
- Programming API

Flexible processing

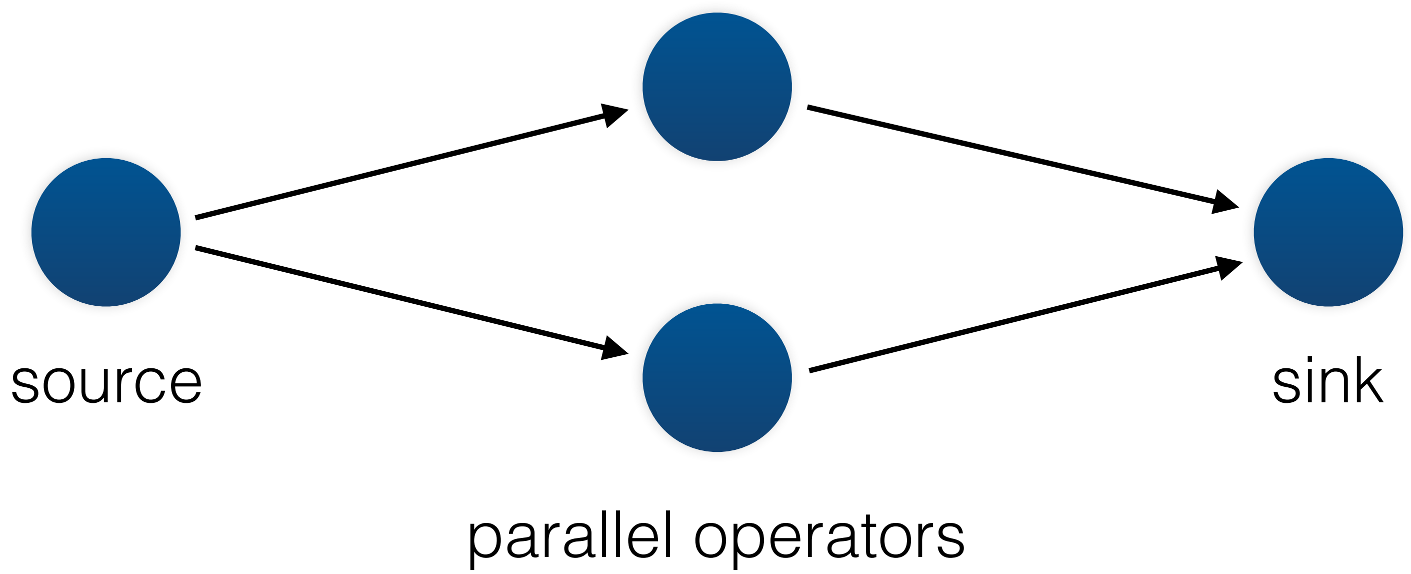
x86 / general purpose programming language

runtime configurability

~ 10M pps per core

jetstream

Leveraging parallel computation

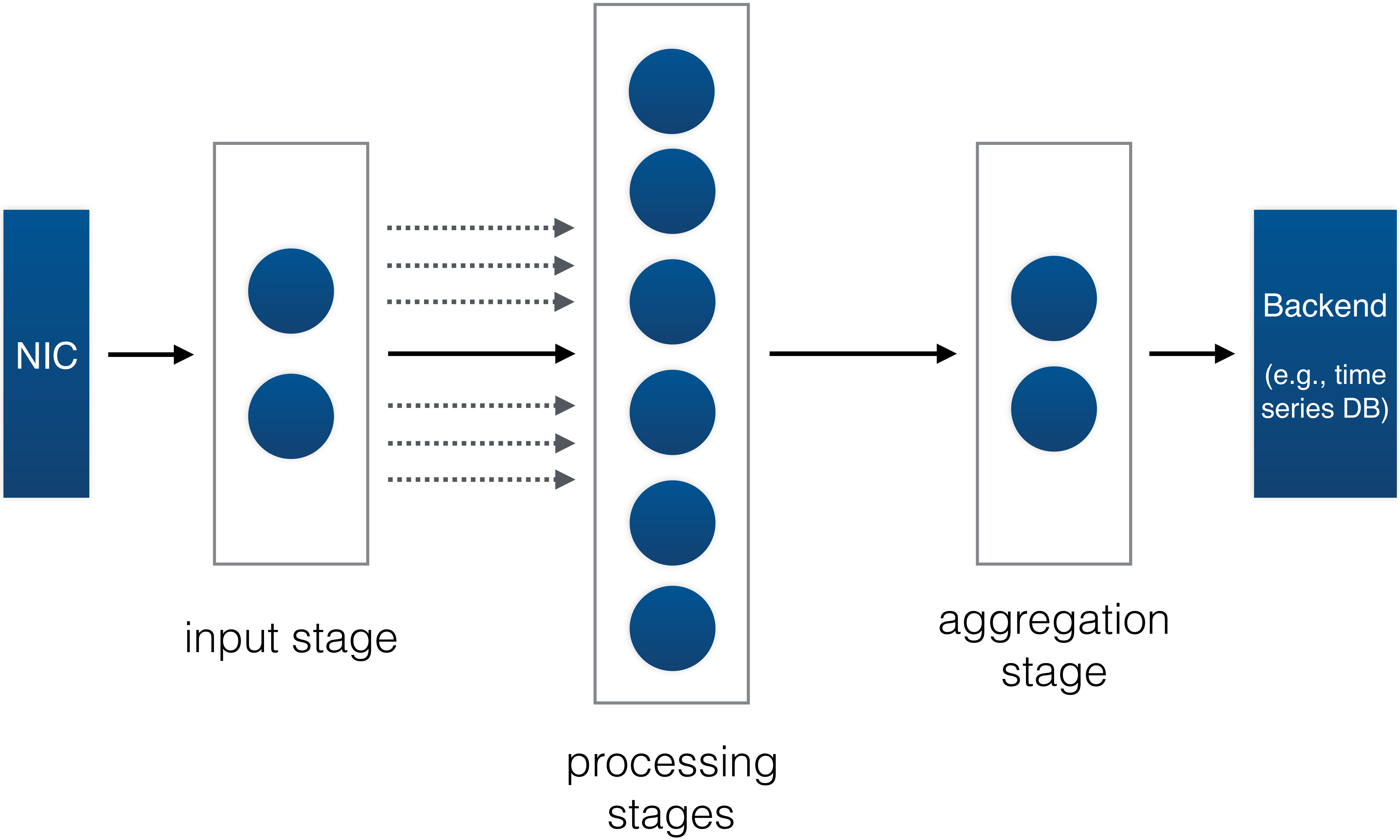


Flink

Spark Streaming

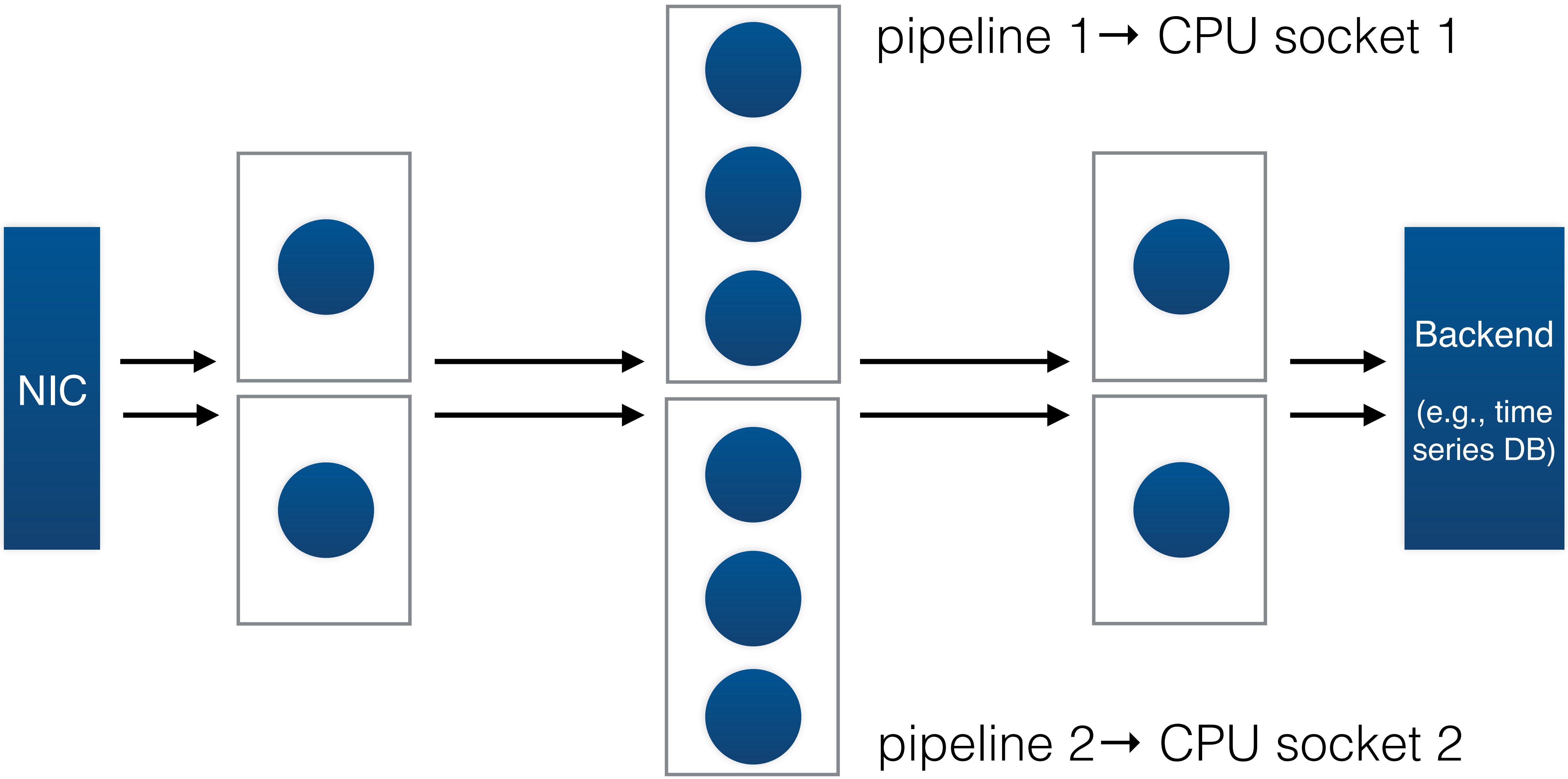
APACHE STORM™
Distributed · Resilient · Real-time

Jetstream architecture



Jetstream architecture

NUMA awareness

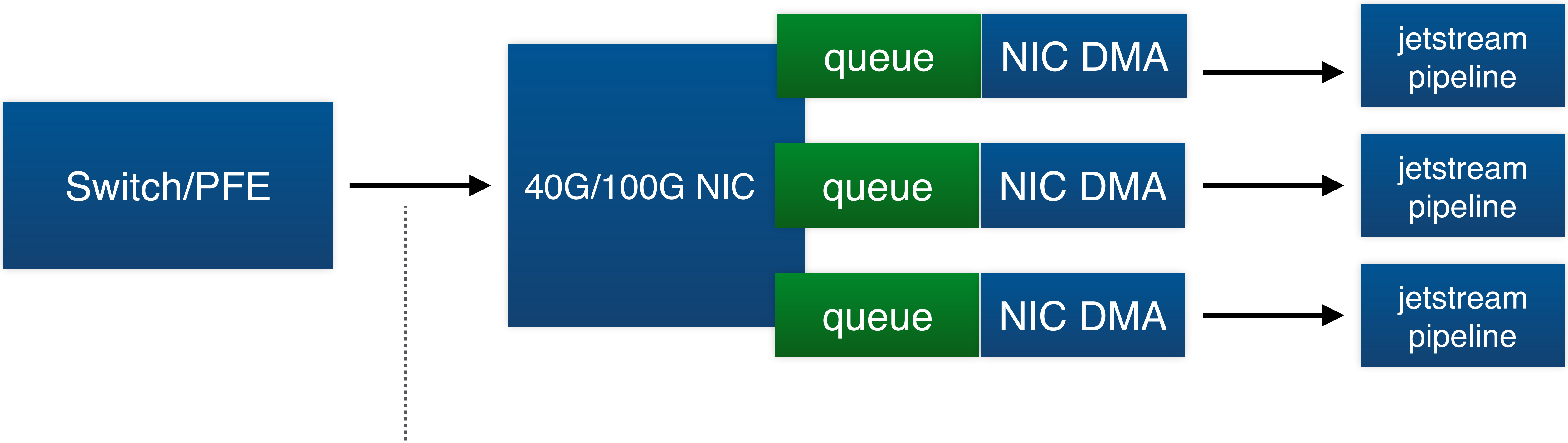


Characteristics of packet record workloads

Can we use properties of packet analytics workloads to our advantage?

- Network attached input
- Partitionability
- Small, simple, well-formed records
- Aggregation

Network attached input

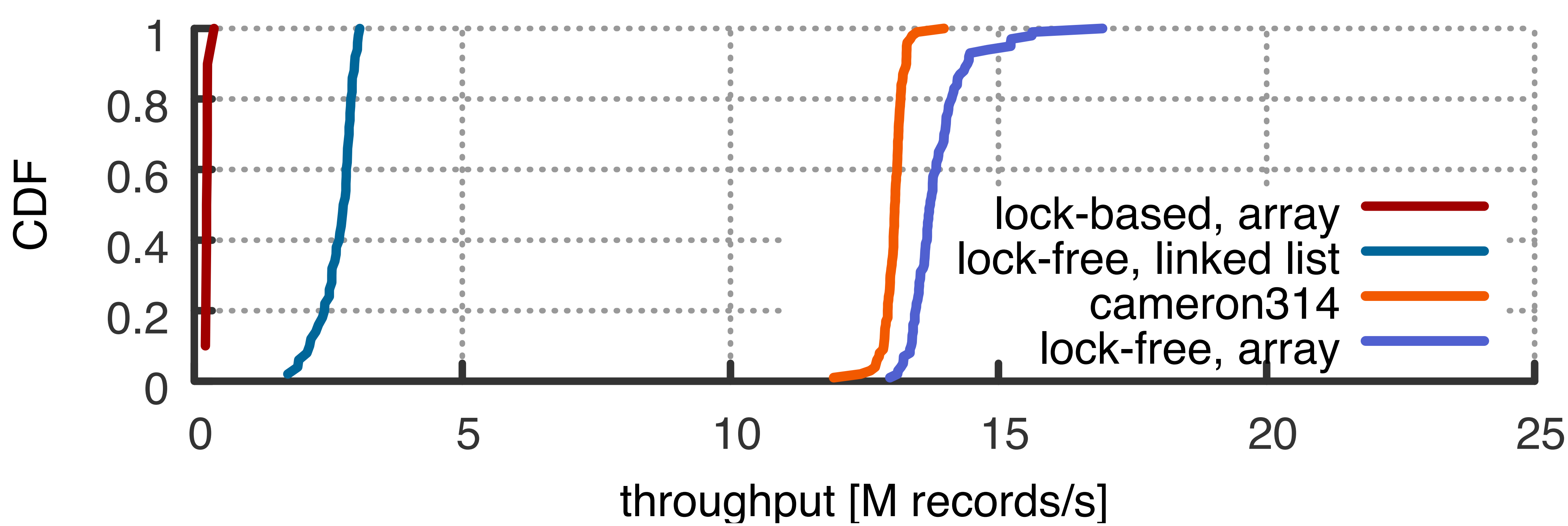


~ 131 M packet records/s
~ 41.9 Gbit/s
Barefoot Tofino PFE

Many small records

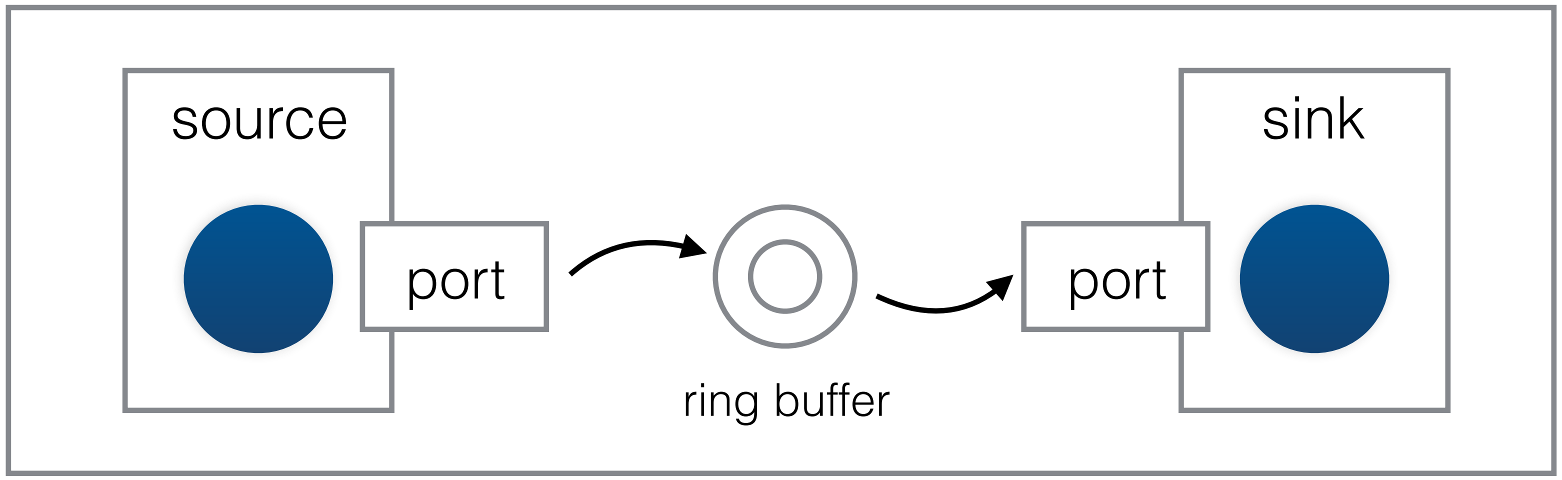
- Array vs. linked list
- Lock-free design
- Wait-free design
- Zero-copy operations

```
bool enqueue(const T& element_)

while (!q.enqueue(e)) { }

if (!q.enqueue(e))
    std::this_thread::yield();
```



Programming abstraction

Application definition




```
int main(int argc, char** argv)
{
    jetstream::app app;
    auto source = app.add_stage<source>(1, “enp6s0f0”);
    auto sink   = app.add_stage<sink>(1, std::cout);
    app.connect<jetstream::pkt_t>(source, sink);
    app();
    return 0;
}
```

Programming abstraction

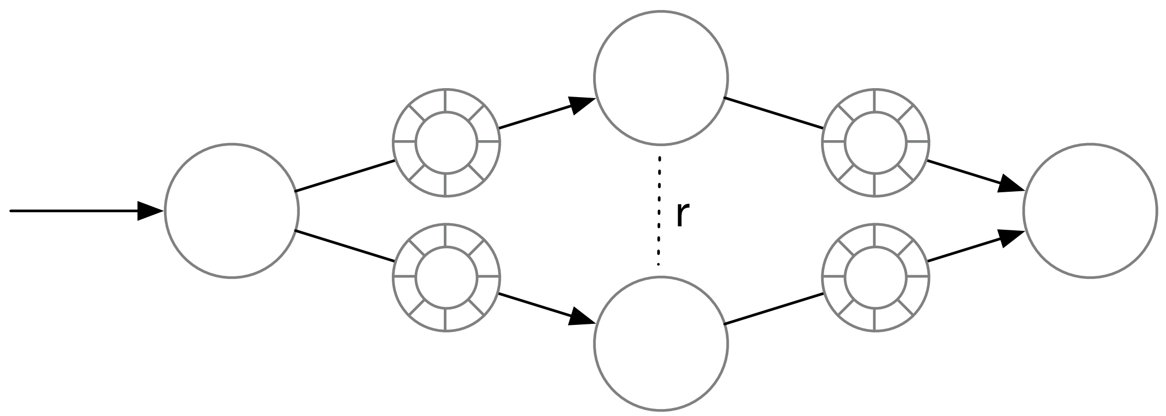
Processor definition

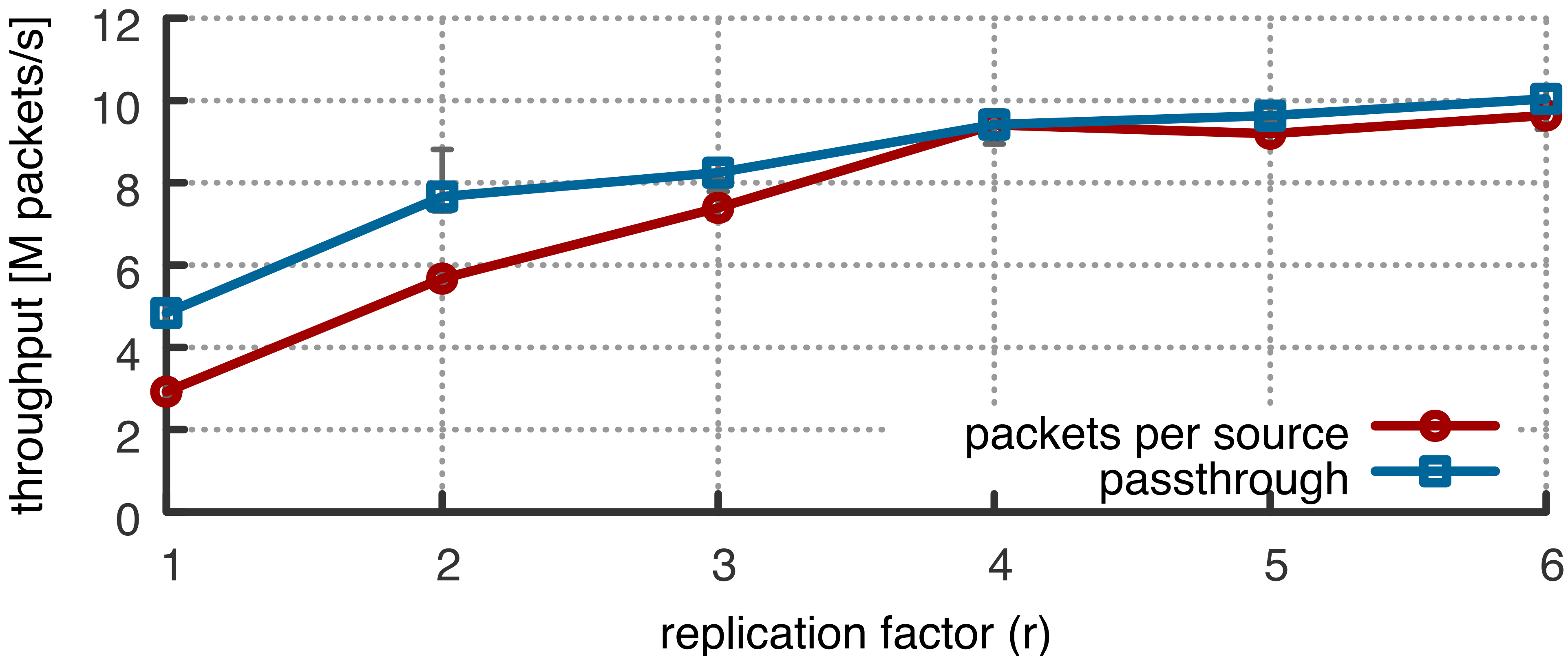
```
class source : public jetstream::proc {
  [...]
};
```

```
explicit source(const std::string& iface_name_) : proc() {
  add_out_port<jetstream::pkt_t>(0);
  [...]
}
```

```
jetstream::signal operator()() override {
    out_port<pkt_t>(0)->enqueue(read_from_nic(_pkt),
                                jetstream::signal::continue);
    return jetstream::signal::continue;
}
```

Performance

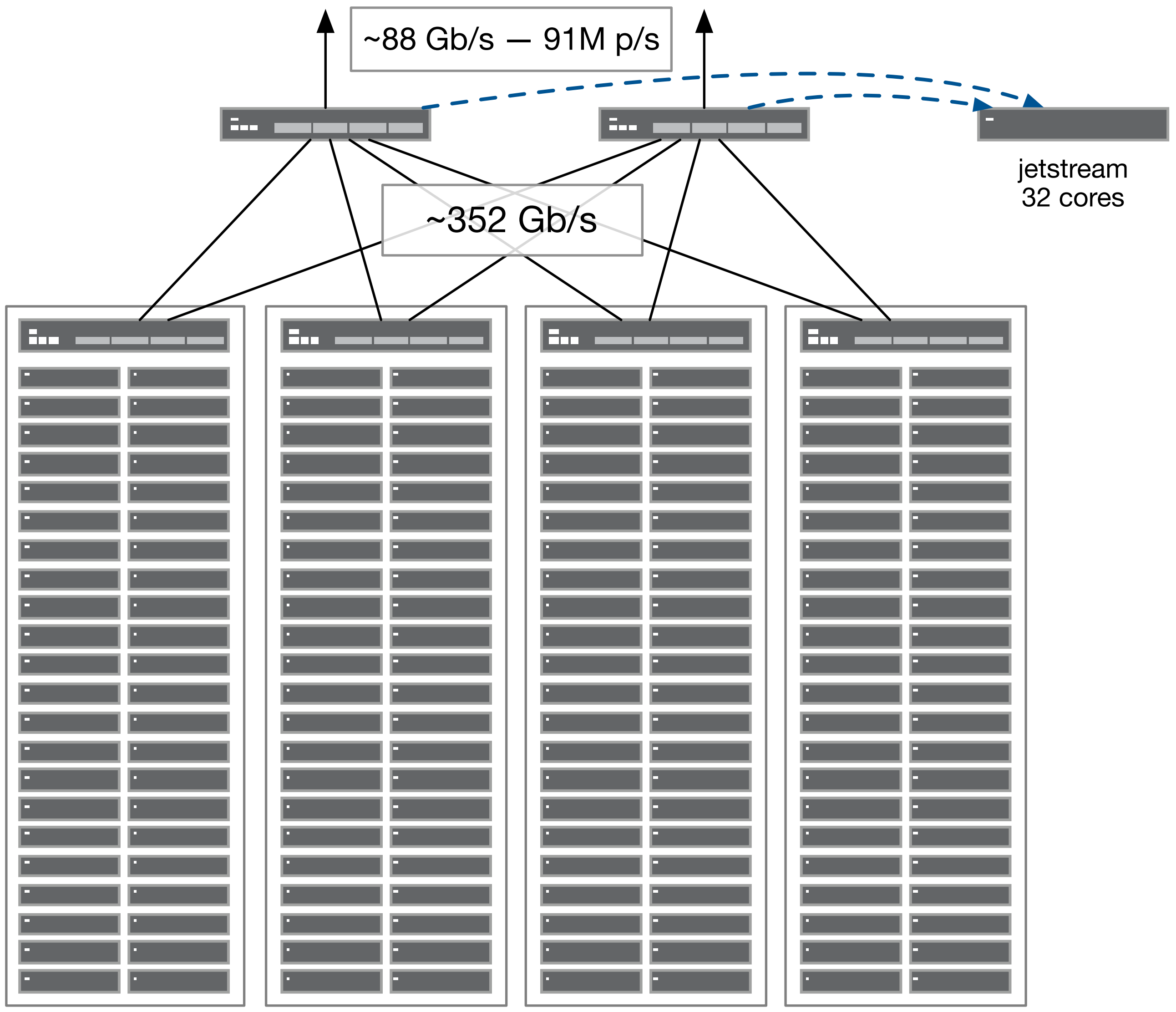




Evaluation

Facebook cluster study

- 2.9M packets/core: 32/64 cores for 4/8 racks
- StreamBox: 5096/10192 cores (163x)
- Single server: 1/176 ≅ 0.5% of cluster



[Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. SIGCOMM Comput. Commun. Rev. 45, 4 (August 2015), 123-137].

Conclusion

*flow → high-performance, hardware-accelerated network telemetry system

jetstream → high-performance, software network analytics platform

Conclusion

John Sonchack, Oliver Michel, Adam J. Aviv, Eric Keller, Jonathan M. Smith

Scaling Hardware Accelerated Monitoring to Concurrent and Dynamic Queries with *Flow

To appear: USENIX ATC 2018

Oliver Michel, John Sonchack, Eric Keller, Jonathan M. Smith

Packet-Level Analytics in Software without Compromises

To appear: USENIX HotCloud 2018

Scaling Hardware Accelerated Monitoring to Concurrent and Dynamic Queries With *Flow

John Sonchack[*], *Oliver Michel*[†], *Adam J. Aviv*[‡], *Eric Keller*[†], *and Jonathan M. Smith*[*]
[*]University of Pennsylvania, [‡]United States Naval Academy, and [†]University of Colorado, Boulder

Abstract

We introduce *Flow, a practical system for hardware accelerated traffic monitoring. *Flow is highly scalable and able to execute many concurrent and dynamically changing traffic queries with minimal network disruption. The design insight is to move query specific computation off of the switch ASIC and into software running on commodity servers. We evaluated *Flow on a 3.2 Tb/s Barefoot Tofino switch on which we developed a novel dynamic cache data structure to build and export to software flow records that contain per-packet information in a compact, *disaggregated* format that enables highly efficient software processing. We demonstrate *Flow's capability to efficiently support multiple concurrent queries at scale through a Raftlib stream pro-

and network resources required for the monitoring infrastructure [39]. There are two other important requirements that the compiled query model does not address: concurrency and dynamic queries.

First, support for *concurrent* traffic queries. In most networks, there are often multiple applications or operators observing the network concurrently but with different queries. A practical monitoring infrastructure needs to multiplex the PFE across all the concurrently active queries. This is a challenge when the entire query is compiled to the PFE. Each query requires different computation that, given the line-rate processing model of a PFE [49], must map to dedicated computational resources, which are limited in PFEs.

Equally important for practical deployment is support

Packet-Level Analytics in Software without Compromises

Oliver Michel
University of Colorado Boulder

John Sonchack
University of Pennsylvania

Eric Keller
University of Colorado Boulder

Jonathan M. Smith
University of Pennsylvania

Abstract

Traditionally, network monitoring and analytics systems rely on aggregation (e.g., flow records) or sampling to cope with high packet rates. This has the downside that, in doing so, we lose data granularity and accuracy, and in general limit the possible network analytics we can perform. Recent proposals leveraging software-defined networking or programmable hardware provide more fine-grained, per-packet monitoring but still are based on the fundamental principle of data reduction in the network, before analytics. In this paper, we provide a first step towards a cloud-scale, packet-level monitoring and analytics system based on stream processing entirely in software. Software provides virtually unlimited programmability and makes modern (e.g., machine-learning) net-

just couldn't process the information fast enough. These approaches, of course, reduce information – aggregation reduces the load of the analytics system at the cost of granularity, as per-packet data is reduced to groups of packets in the form of sums or counts [3, 16]. Sampling and filtering reduces the number of packets or flows to be analyzed. Reducing information reduces load, but it also increases the chance of missing critical information, and restricts the set of possible applications [30, 28].

Recent advances in software-defined networking (SDN) and more programmable hardware have provided opportunities for more fine-grained monitoring, towards packet-level network analytics. Packet-level analytics systems provide the benefit of complete insight into the network and open up opportunities for applications that

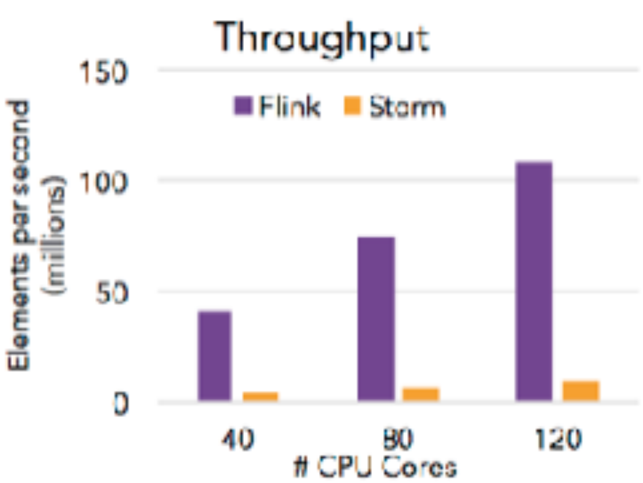
Q&A / DISCUSSION

Oliver Michel

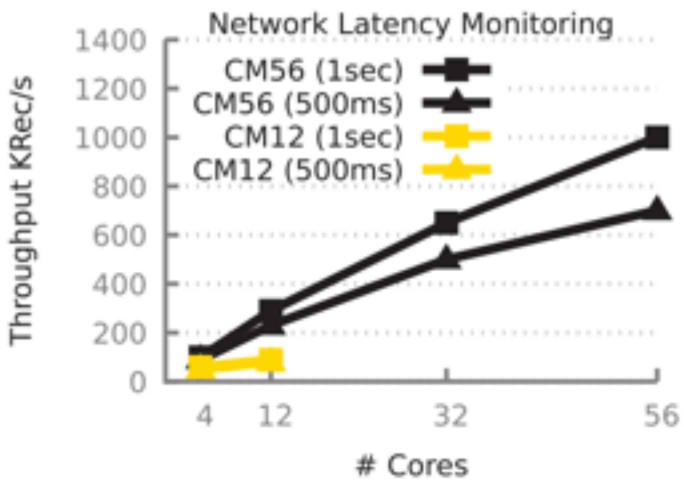
oliver.michel@colorado.edu
http://nsr.colorado.edu/oliver

University of Colorado Boulder

Backup Slides

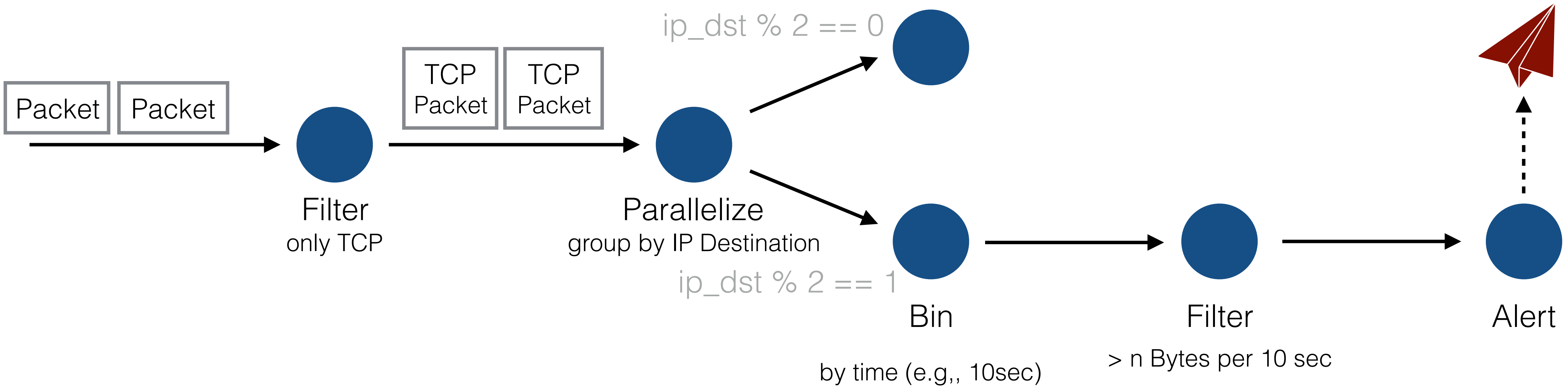


[Apache Flink]

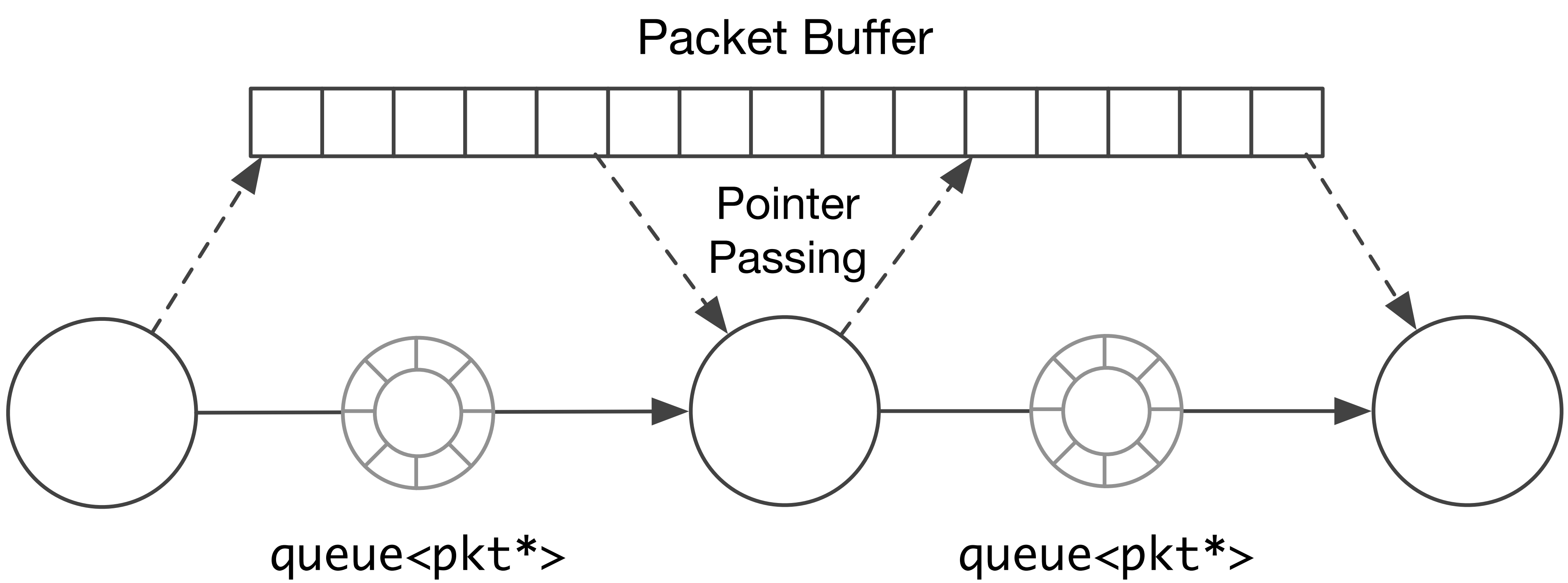


[StreamBox Miao '18]

Stream Processing

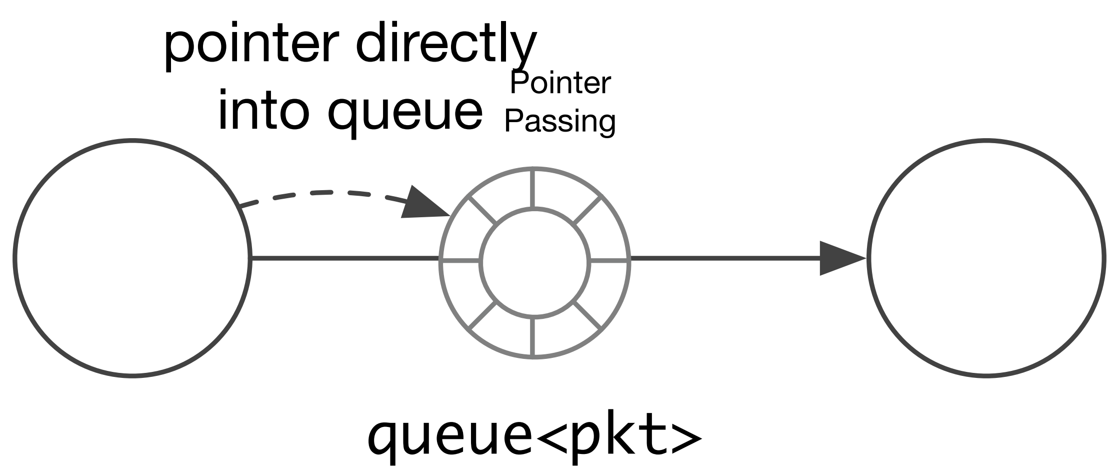


Reducing copy operations



Reducing copy operations

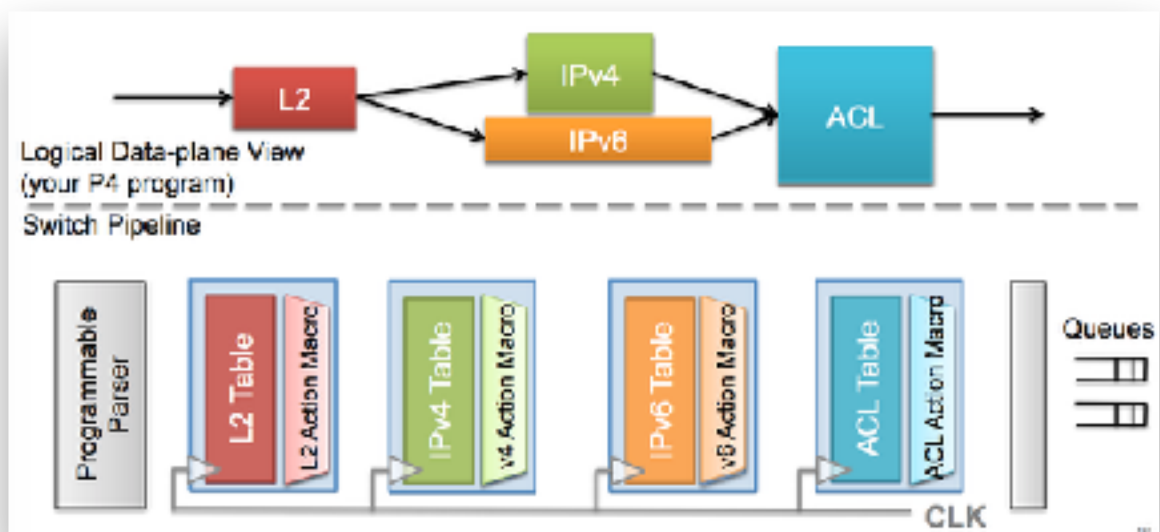
```
packet p;
p.ip_proto = 6;
q.enqueue(p);
```

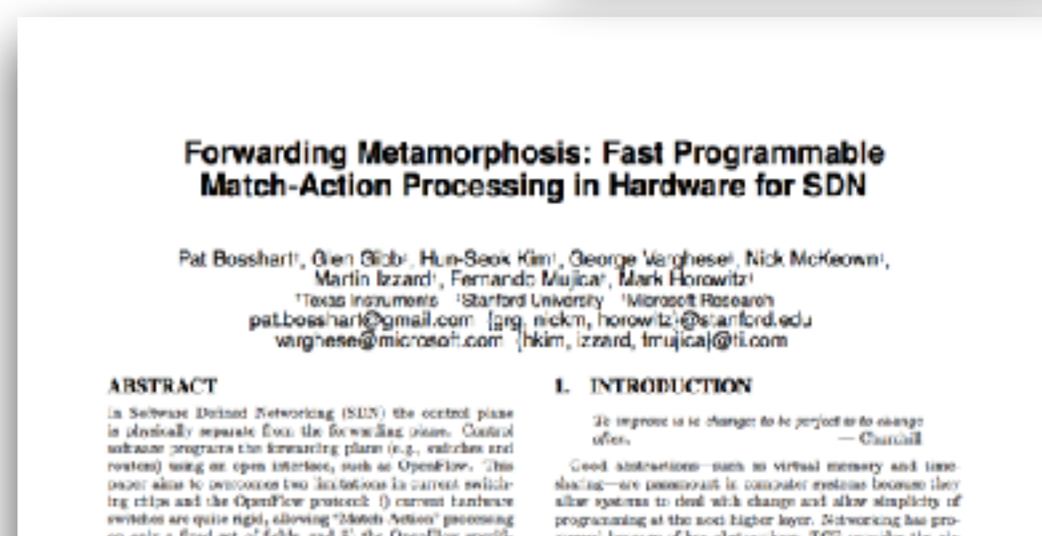



```
auto p = q.enqueue();
p->ip_proto = 6;
```

Technologies

- Programmable switches and PISA: Protocol Independent Switch Architecture
  - Reconfigurable match-action tables in hardware
  - multiple stages with TCAM/ALU pair, fixed processing time, guarantees line rate





Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, Mark Horowitz

ABSTRACT

1. INTRODUCTION

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Walker

Barefoot Networks, Intel, Stanford University, Princeton University, Google, Microsoft Research

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this