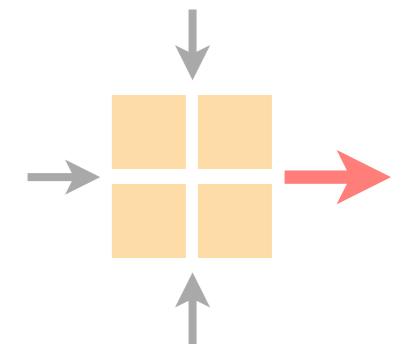


Advanced Topics in Communication Networks

Programming Network Data Planes



Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich
Oct 22 2019

Last week on
Advanced Topics in Communication Networks

*A bloom filter is a streaming algorithm
answering specific questions approximately.*

*A bloom filter is a streaming algorithm
answering **specific questions** approximately.*

Is X in the stream?

What is in the stream? Invertible Bloom Filter

*A bloom filter is a streaming algorithm
answering specific questions approximately.*

Is X in the stream?
What is in the stream?^{Invertible Bloom Filter}

What about other questions?



Is a certain flow in the stream?

Bloom Filter

What flows are in the stream?

Invertible Bloom Filter, HyperLogLog Sketch, ...

How frequently does an flow appear?

Count Sketch, CountMin Sketch, ...

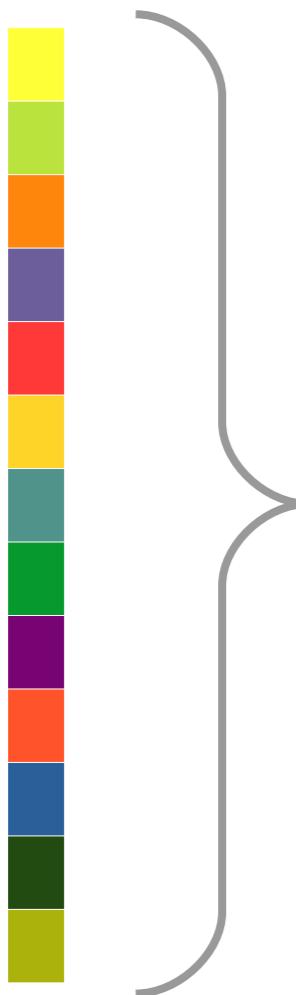
What are the most frequent elements?

Count/CountMin + Heap, ...

How many flows belong to a certain subnet?

SketchLearn SIGCOMM '18

In the worst case, an algorithm providing
exact frequencies requires **linear space**.



Data Stream

n elements in total

→ **n distinct elements**

(in the worst case)

→ **n counters** required? :(

Probabilistic datastructures can help again!

Bloom Filters

quickly “filter” only those elements that might be in the set

Save space by allowing false positives.

Sketches

provide approximate frequencies of elements in a data stream.

Save space by allowing mis-counting.

A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

$$Pr [\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1] \leq \delta$$

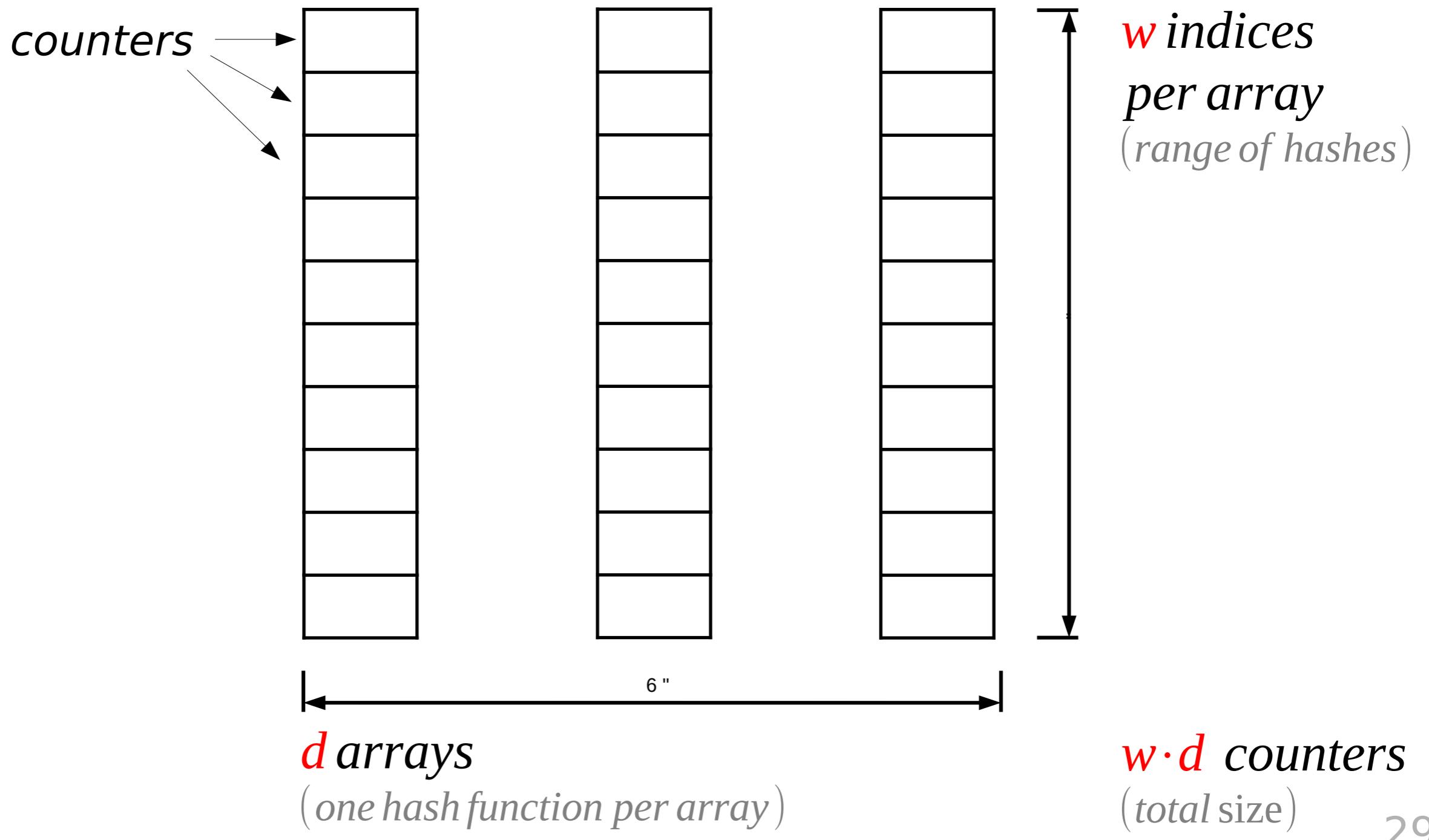
estimated frequency *true frequency* *sum of frequencies*

relative to L1 norm

*The estimation error **exceeds** $\varepsilon \|\mathbf{x}\|_1$*
*with a **probability smaller than** δ*

A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

A **CountMin** Sketch uses multiple arrays and hashes.



A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

CountMin sketch recipe

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|x\|_1$ with a probability less than δ

A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

→ only one design out of many!

A **Count sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L2 error bounds** for frequency queries.

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

CountMin sketch recipe

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|x\|_1$ with a probability less than δ

Count sketch recipe

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon^2} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|x\|_2$ with a probability less than δ

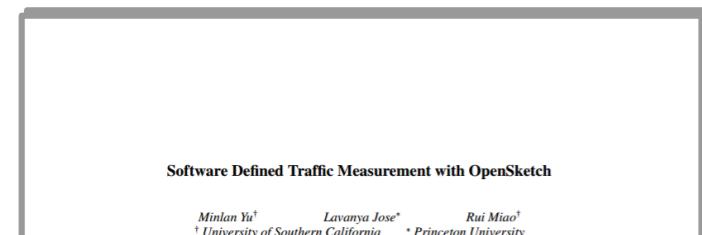
Sketches are the new black

...and many more!

OpenSketch

NSDI '13

[source]



Software Defined Traffic Measurement with OpenSketch

Minlan Yu[†] Lavanya Jose^{*} Rui Miao[†]
^{*}University of Southern California * Princeton University

Abstract

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software-defined traffic measurement system called OpenSketch, which performs the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFPGA, and the implementation of five measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

1 Introduction

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages: (1) measuring the network in real time (e.g., identifying traffic anomalies or large traffic aggregates) and then (2) adjusting the control of the network accordingly (e.g., routing, access control, and rate limiting). While there have been many efforts on designing the right APIs for network control (e.g., OpenFlow [29], ForCES [1], rule-based forwarding [33], etc.), little thought has gone into designing the right APIs for measurement. Since con-

UnivMon

SIGCOMM '16

[source]



One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon

Zaoxing Liu[†], Antonis Manousis^{*}, Gregory Vorsanger[†], Vyv Sekar^{*}, Vladimir Braverman[†]
[†]Johns Hopkins University * Carnegie Mellon University

ABSTRACT

Network management requires accurate estimates of metrics for many applications including traffic engineering [11, 32], attack and anomaly detection [49], and forensic analysis [46]. Each such task requires specific resource requirements and timely estimates on different application-level metrics of interest, e.g., the flow size distribution [37], heavy hitters [10], entropy measures [38, 50], or detecting changes in traffic patterns [44]. Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection) which is important for traffic engineering in data centers [6], NetFlow collects flow-level counts for sampled packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) thus requiring another way of changing NetFlow. Instead, we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements, and provide guarantees on the measurement accuracy.

CCS Concepts

• Networks → Network monitoring; Network measurement;

Keywords

Flow Monitoring, Sketching, Streaming Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for this work (c) 2016 ACM. This is the author's version of the work. It is made available with credit to the author. This version may differ from the final version in a journal or proceedings due to minor revision after presentation. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

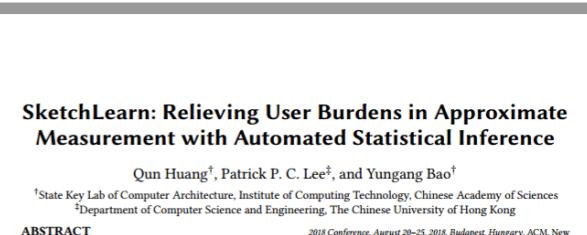
SIGCOMM '16, August 22–26, 2016, Fluminopolis, Brazil

© 2016 ACM. ISBN 978-1-4503-4854-6/16/08...\$15.00
DOI: <https://doi.org/10.1145/2923482.2934906>

SketchLearn

SIGCOMM '18

[source]



SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Qun Huang[†], Patrick P. C. Lee[‡], and Yungang Bao[†]

[†]State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences

[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong

ABSTRACT

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that leverages automated statistical inference properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

CCS CONCEPTS

• Networks → Network measurement;

KEYWORDS

Sketch; Network measurement

ACM Reference Format:
Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference*, August 20–25, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3230543.3230559>

Although theoretically sound, existing approximate measurement approaches are memory-expensive in practice. In such approaches, massive memory traffic competes for the limited resources, thereby introducing measurement errors due to resource conflicts (e.g., multiple flows are mapped to the same counter in sketch-based measurement). To mitigate errors, sufficient resources must be provisioned in approximate measurement based on its theoretical guarantees. Thus, there exists a tight binding between resource configurations and accuracy parameters. Such tight binding leads to several practical limitations (see §2.2 for details): (i) administrators need to reserve resources for particular traffic patterns tracking.

Today we'll talk about: important questions,
how 'sketches' answer them,
limitations of 'sketches',
and my master thesis :)

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

Let $\varepsilon = 0.01$, $\|x\|_1 = 10000$ ($\Rightarrow \varepsilon \cdot \|x\|_1 = 100$)

Assume two flows x_a , x_b ,

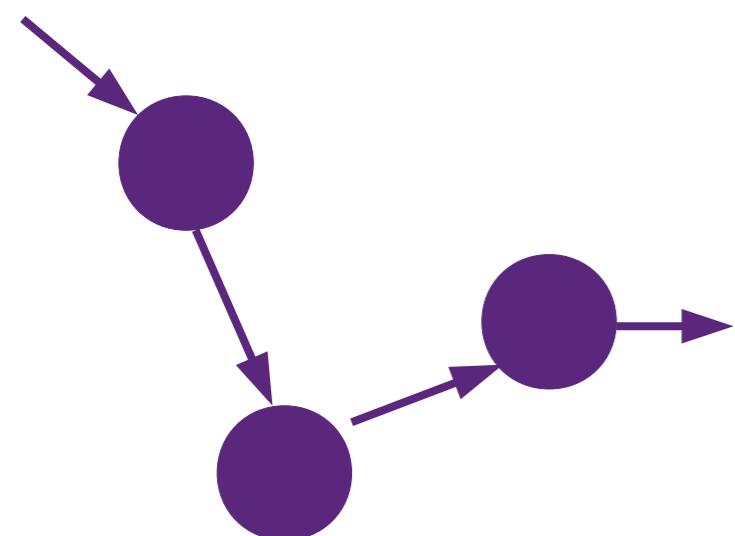
with $\|x_a\|_1 = 1000$, $\|x_b\|_1 = 50$

Error relative to **stream size**: 1%

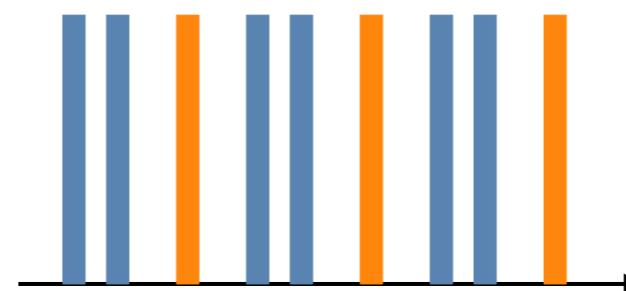
flow size: x_a : 10%, x_b : **200%**

Other Problems a Sketch **can't** handle

causality



patterns



rare things



This week on

Advanced Topics in Communication Networks

P4 hardware
target

P4-based
applications

How do we build a *fast*
reprogrammable switch?

What cool things
can we do with it?

P4 hardware
target

P4-based
applications

How do we build a *fast*
reprogrammable switch?

“Programmable switches are 10-100x slower than fixed-function switches. They cost more and consume more power.”

Conventional wisdom in networking

How can we allow network programmability in the field,
at reasonable cost, and without **sacrificing speed**

supporting Tbps of
backplane throughput

Let's look at a concrete design: Reconfigurable Match Tables (RMT)

The screenshot shows a PDF document titled "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN". The document is presented in a window with a toolbar at the top, including icons for zoom, search, and file operations. The title is centered above the author information, which lists the authors and their institutions: Texas Instruments, Stanford University, and Microsoft Research. The email addresses for the authors are provided below their names. The abstract section follows, describing the RMT model and its benefits. The introduction section begins with a Churchill quote about change.

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[‡]
†Texas Instruments ‡Stanford University §Microsoft Research
pat.bosshart@gmail.com {grg, nickm, horowitz}@stanford.edu
varghese@microsoft.com {hkim, izzard, fmujica}@ti.com

ABSTRACT

In Software Defined Networking (SDN) the control plane is physically separate from the forwarding plane. Control software programs the forwarding plane (e.g., switches and routers) using an open interface, such as OpenFlow. This paper aims to overcomes two limitations in current switching chips and the OpenFlow protocol: i) current hardware switches are quite rigid, allowing “Match-Action” processing on only a fixed set of fields, and ii) the OpenFlow specification only defines a limited repertoire of packet processing actions. We propose the RMT (reconfigurable match tables) model, a new RISC-inspired pipelined architecture for switching chips, and we identify the essential minimal set of action primitives to specify how headers are processed in hardware. RMT allows the forwarding plane to be changed in the field without modifying hardware. As in OpenFlow, the programmer can specify multiple match tables of arbitrary width and depth, subject only to an overall resource limit, with each table configurable for matching on arbitrary fields. However, RMT allows the programmer to modify *all* header fields much more comprehensively than in OpenFlow. Our paper describes the design of a 64 port by 10 Gb/s switch chip implementing the RMT model. Our concrete design demonstrates, contrary to concerns within the community, that flexible OpenFlow hardware switch implementations are feasible at almost no additional cost or power.

1. INTRODUCTION

To improve is to change; to be perfect is to change often. — Churchill

Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues between endpoints, and IP provides a simple datagram abstraction from an endpoint to the network edge. However, routing and forwarding *within* the network remain a confusing conglomerate of routing protocols (e.g., BGP, ICMP, MPLS) and forwarding behaviors (e.g., routers, bridges, firewalls), and the control and forwarding planes remain intertwined inside closed, vertically integrated boxes.

Software-defined networking (SDN) took a key step in abstracting network functions by separating the roles of the control and forwarding planes via an *open* interface between them (e.g., OpenFlow [27]). The control plane is lifted up and out of the switch, placing it in external software. This programmatic control of the forwarding plane allows network owners to add new functionality to their network, while replicating the behavior of existing protocols. OpenFlow has become quite well-known as an interface between the control plane and the forwarding plane based on the approach known as “Match-Action”. Roughly, a subset of packet bytes

[SIGCOMM'13]

Let's look at a concrete design: Reconfigurable Match Tables (RMT)

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

Texas Instruments, Stanford University, Microsoft

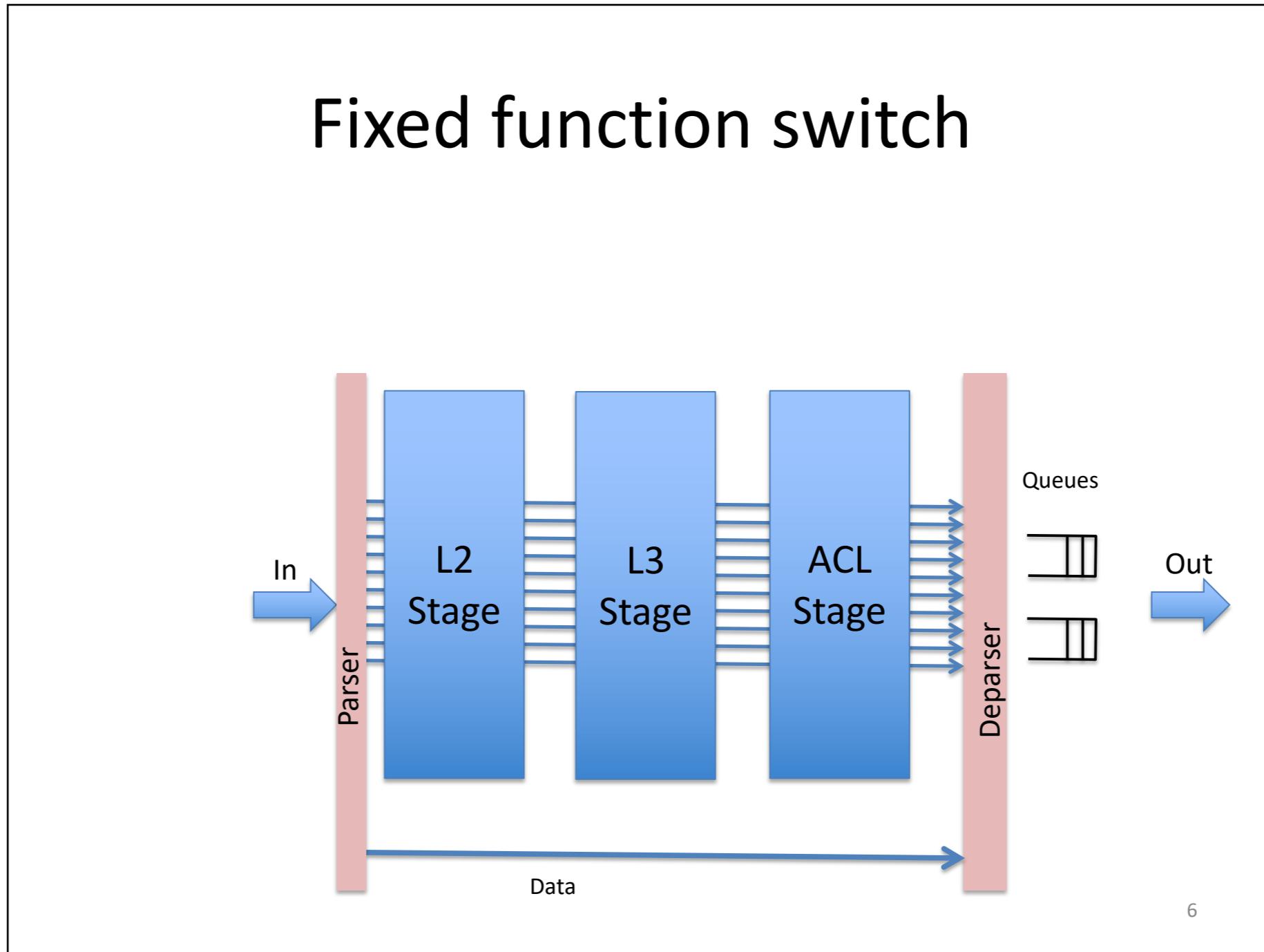
Source: Presentation slides from SIGCOMM'2013 (also available online)

The paper argues that flexibility does **not** come at the price of performance or cost

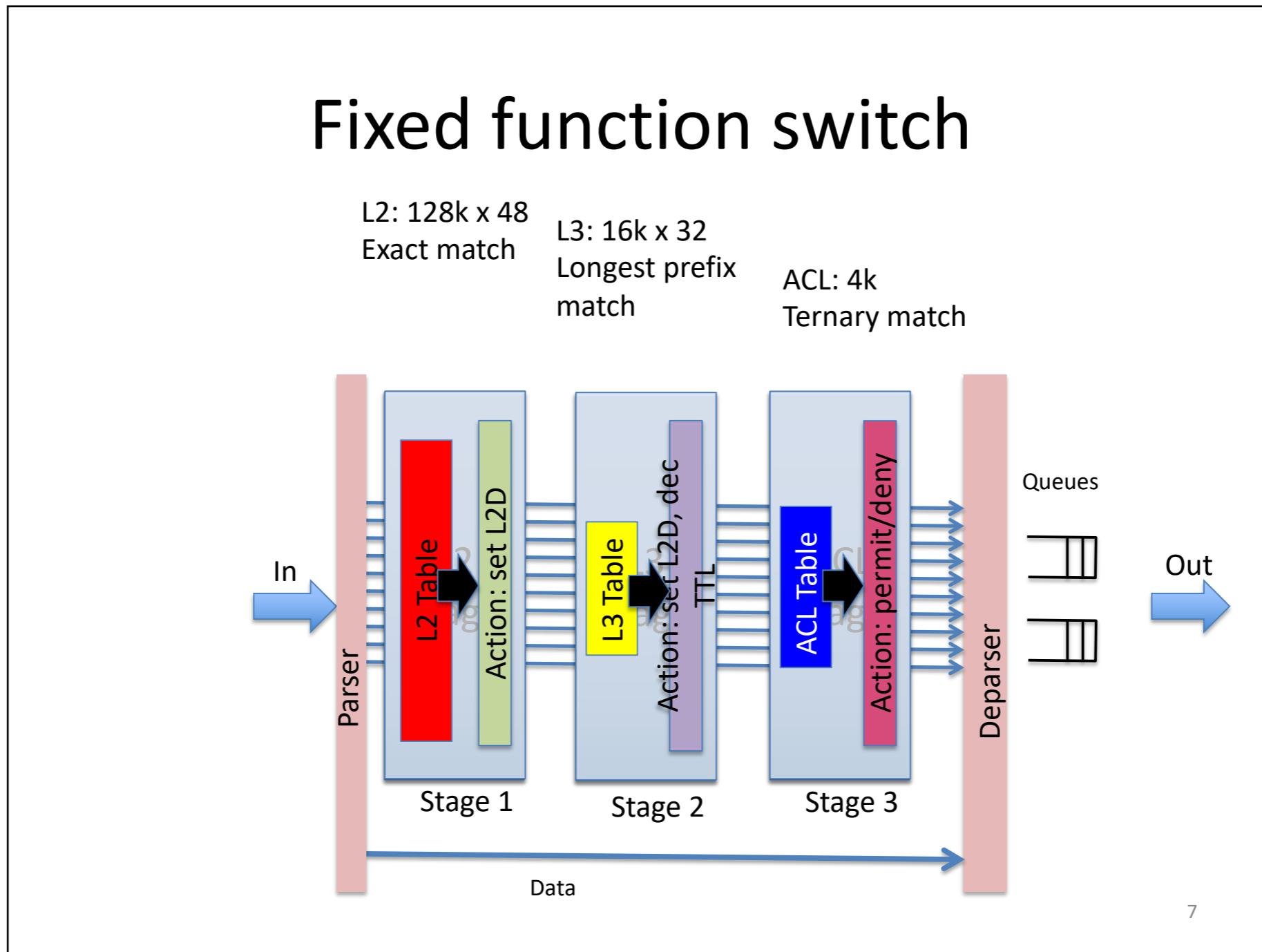
Outline

- Conventional switch chips are inflexible
- SDN demands flexibility...sounds expensive...
- How do we do it: The RMT switch model
- Flexibility costs less than 15%

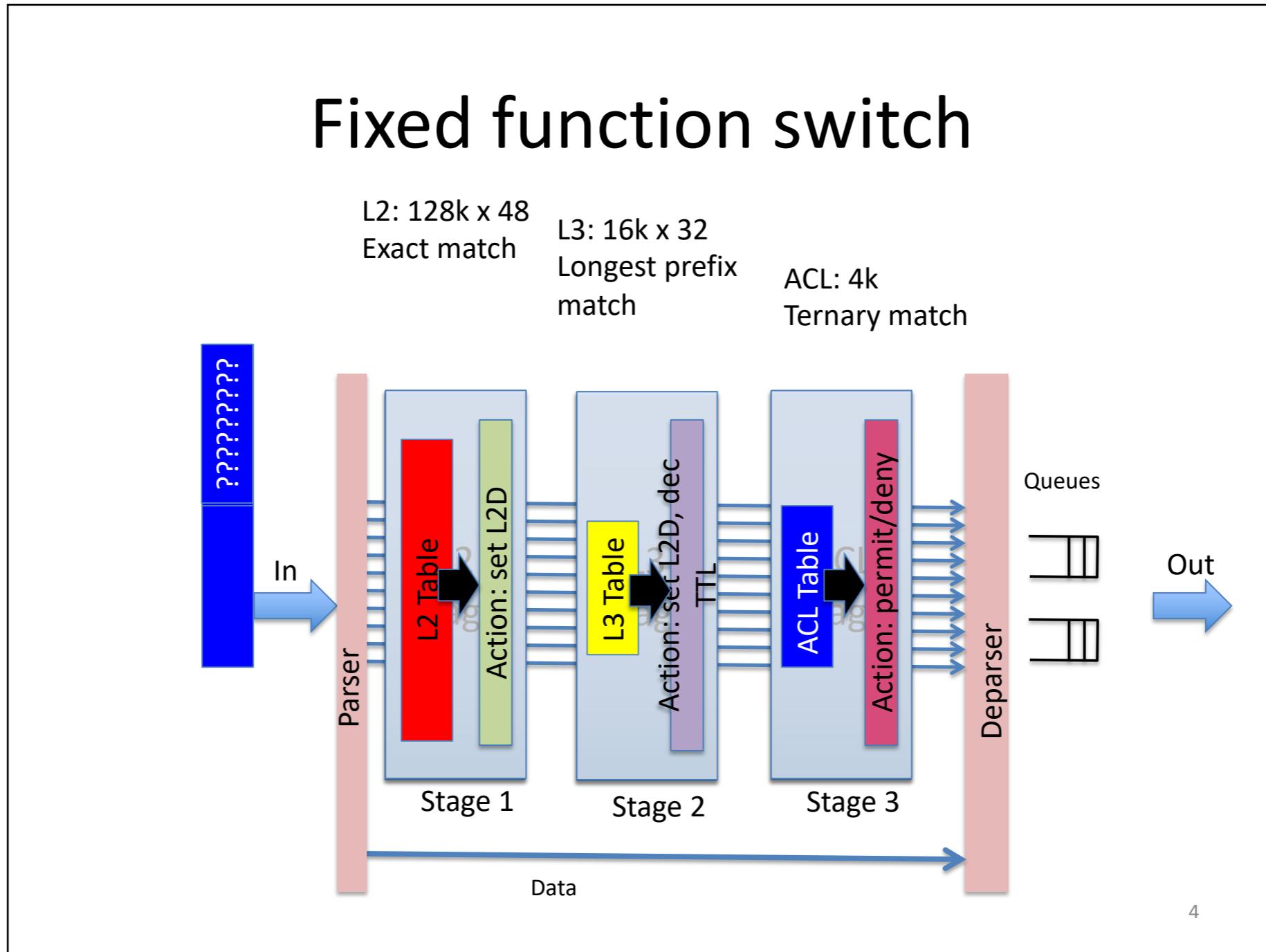
Let's look first at a fixed-function switch composed of a (de-)parser and a sequence of processing stages



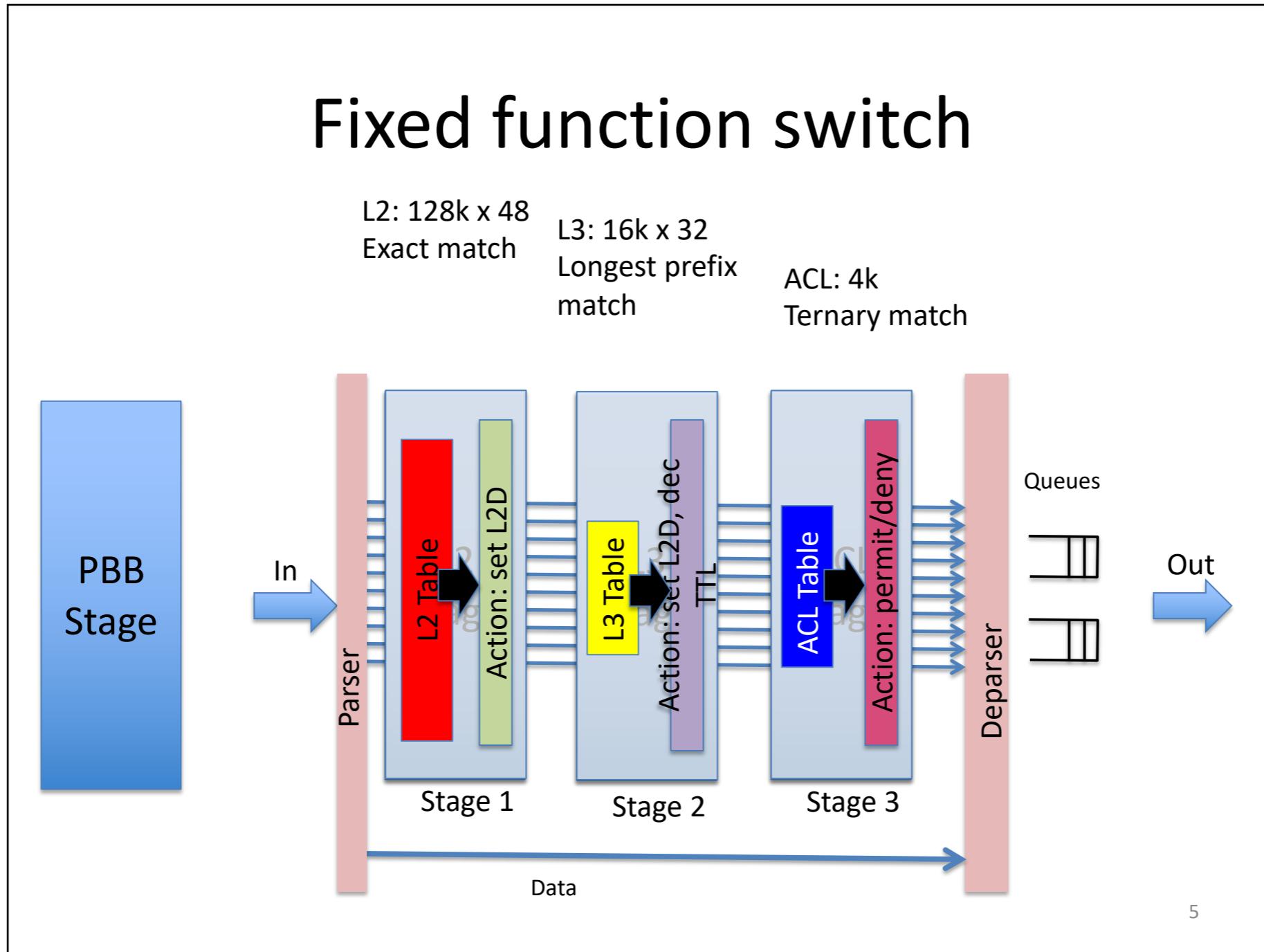
In such a switch,
each stage is particularized to its usage



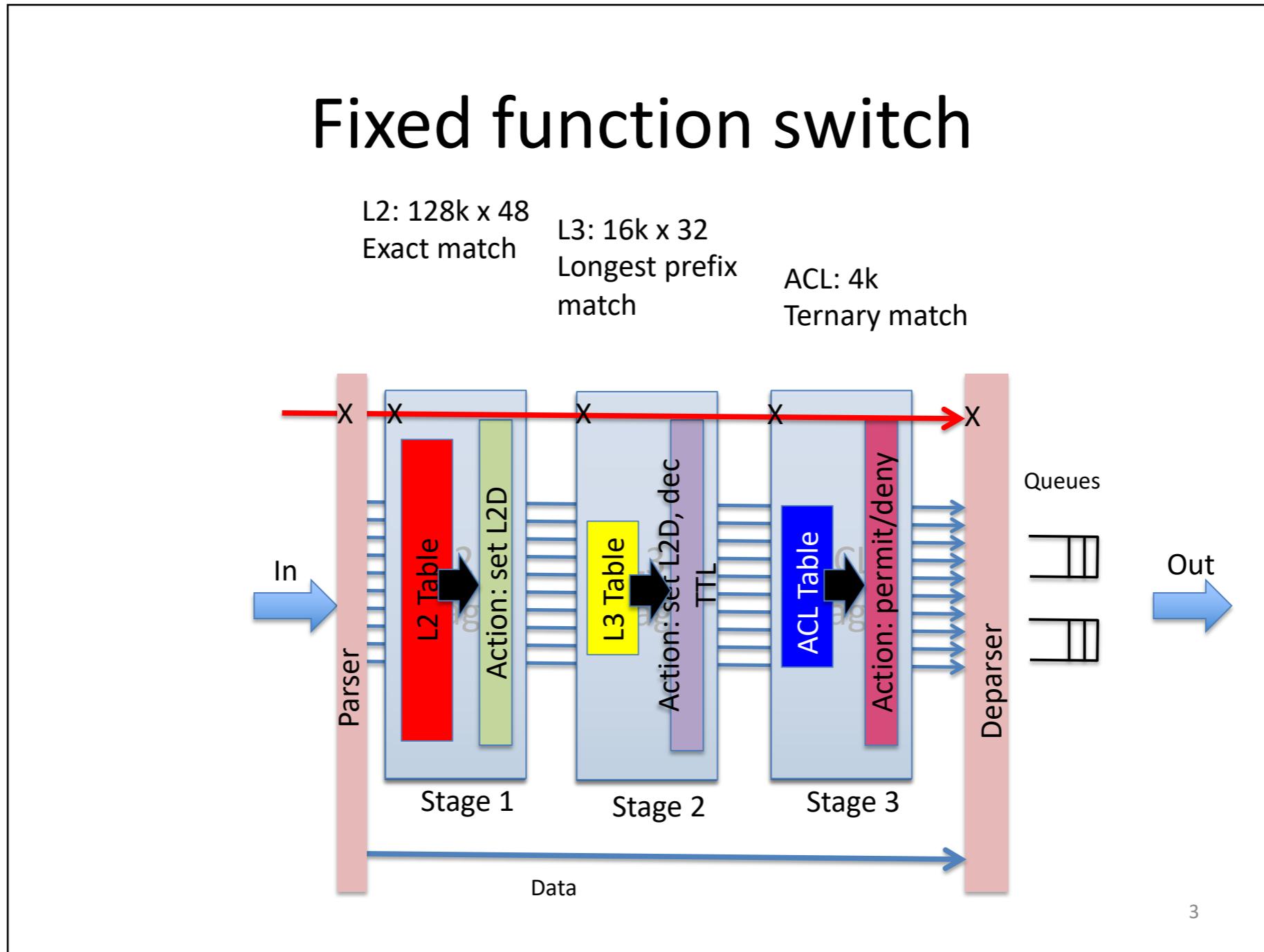
This specificity makes it impossible to...
trade memory size for another



This specificity makes it impossible to...
add a new table



This specificity makes it impossible to...
support new headers or new actions



What if you need flexibility?

- Flexibility to:
 - Trade one memory size for another
 - Add a new table
 - Add a new header field
 - Add a different action
- SDN accentuates the need for flexibility
 - Gives programmatic control to control plane, expects to be able to use flexibility

What does SDN want?

- Multiple stages of match-action
 - Flexible allocation
- Flexible actions
- Flexible header fields
- No coincidence OpenFlow built this way...

Alternative ways to enable flexibility don't compare in terms of cost-performance ratio

What about Alternatives?
Aren't there other ways to get flexibility?

- Software? 100x too slow, expensive
- NPUs? 10x too slow, expensive
- FPGAs? 10x too slow, expensive

What We Set Out To Learn

- How do I design a flexible switch chip?
- What does the flexibility cost?

Unsurprisingly...
building flexible switching chipset *is* challenging

What's Hard about a Flexible Switch Chip?

- Big chip
- High frequency
- Wiring intensive
- Many crossbars
- Lots of TCAM
- Interaction between physical design and architecture
- Good news? No need to read 7000 IETF RFC's!

Enter...

Reconfigurable Match Tables (RMT)

Outline

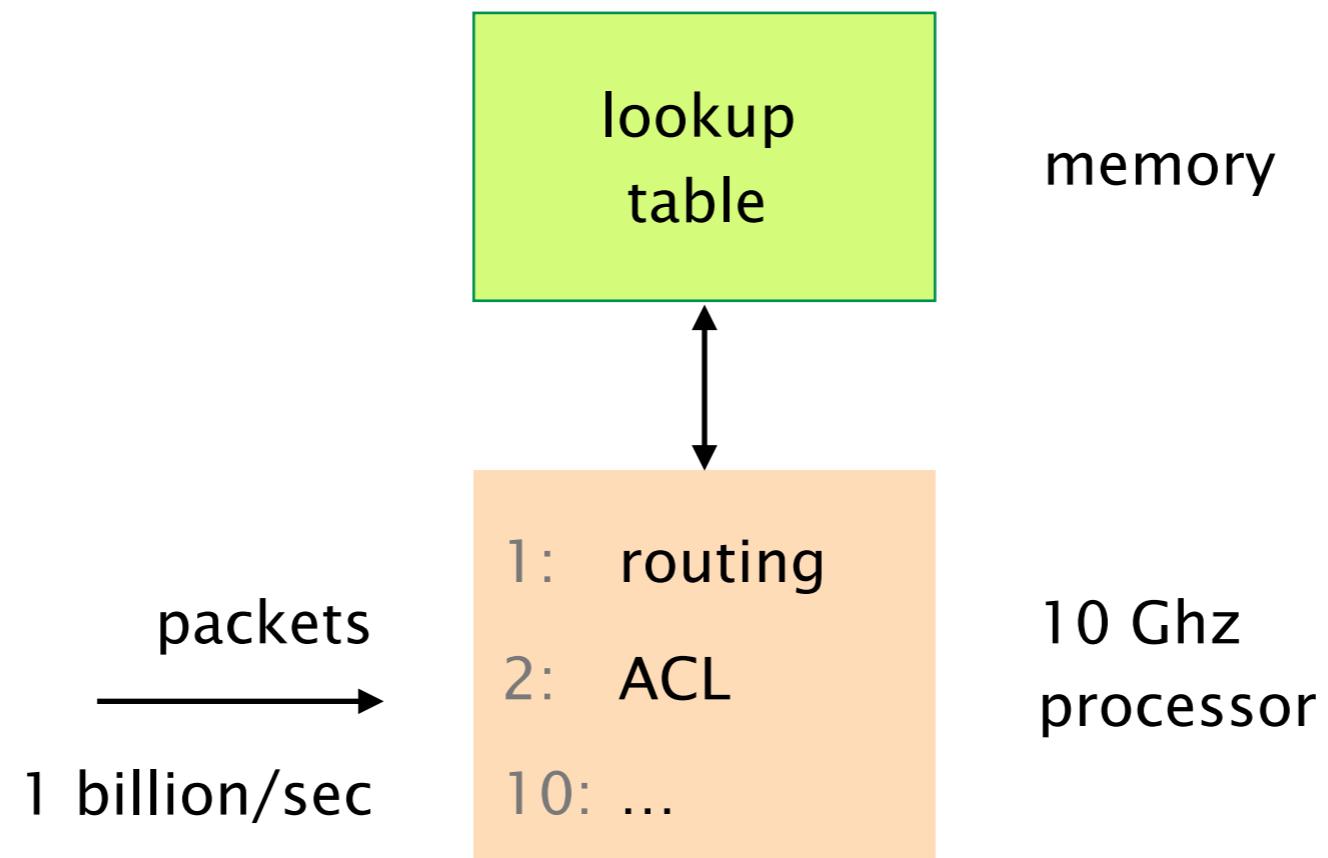
- Conventional switch chip are inflexible
- SDN demands flexibility...sounds expensive...
- **How do we do it: The RMT switch model**
- Flexibility costs less than 15%

What kind of switch architecture could support flexibility and yet run at Terabits per second?

Throughput aggregate	1 Tbps
Packet size average	1000 bits
# operations per packet (avg.)	10
Requirements	10 billion op./second

If our switch has a single processor,
this would require us to run it at **10 Ghz**...

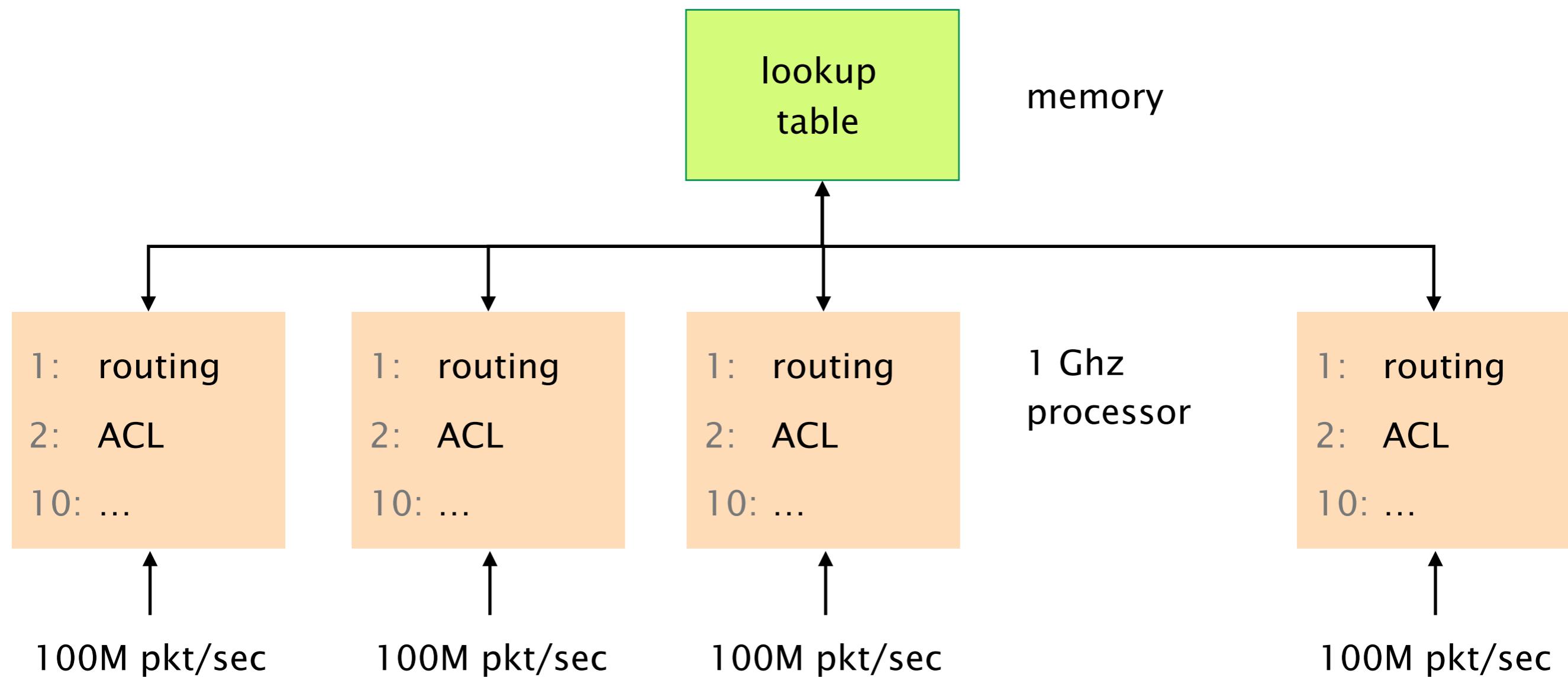
not feasible



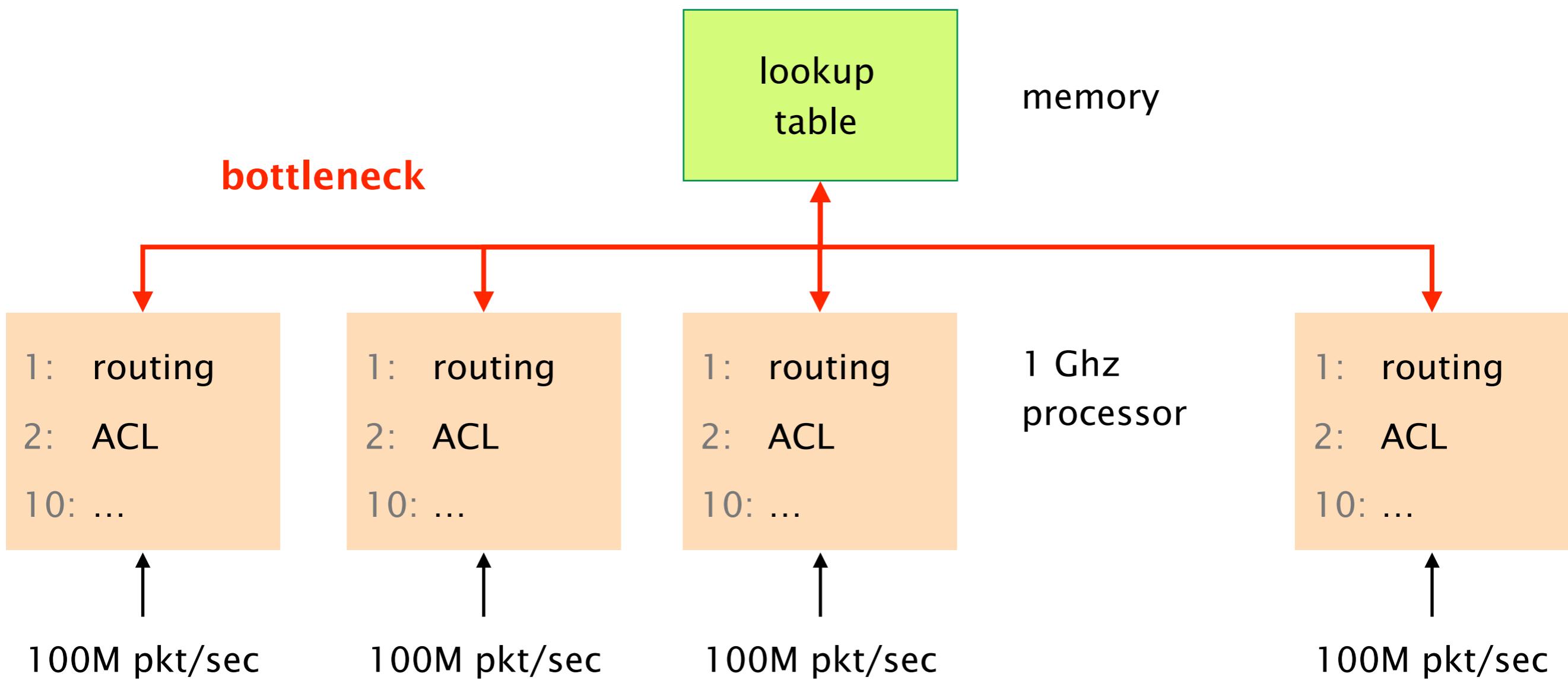
Let's parallelize things with a
packet-parallel architecture

What about we duplicate the processing units?

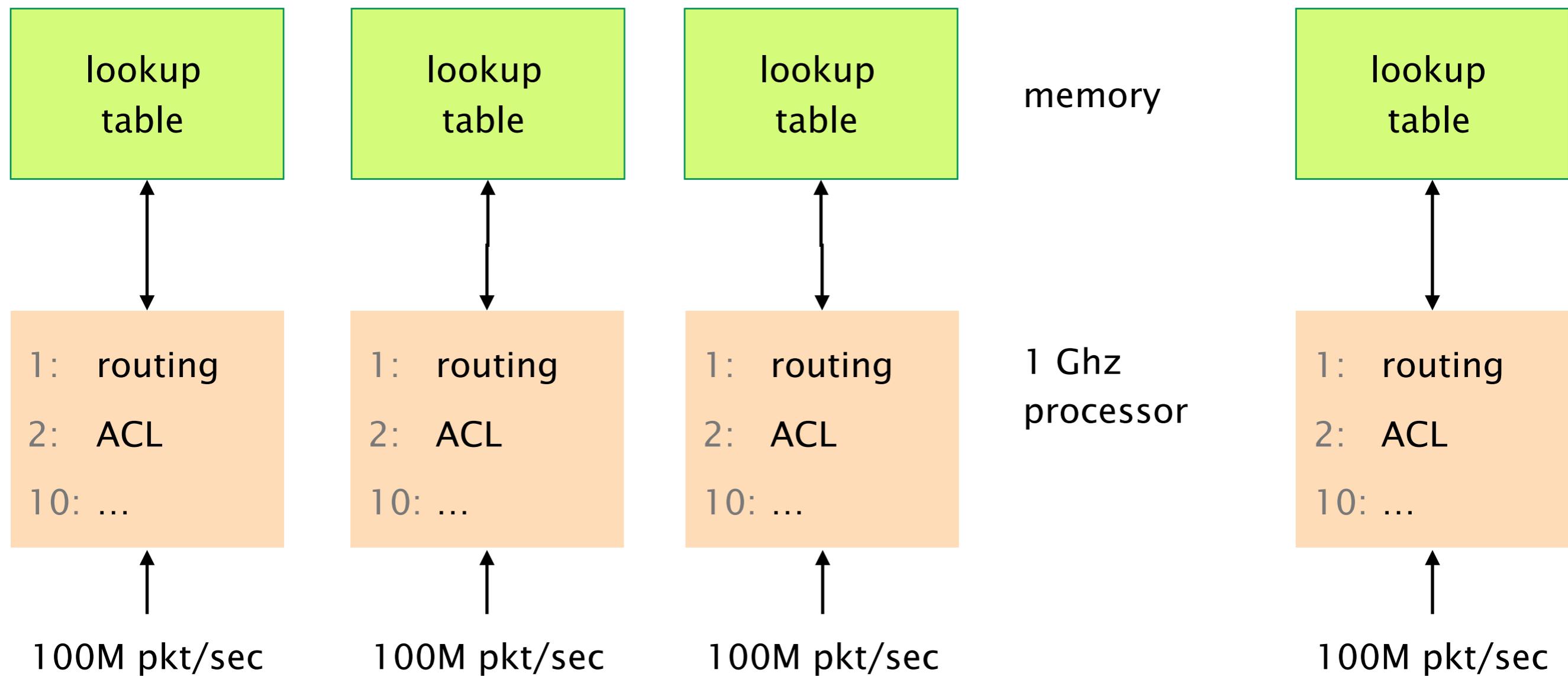
Each of which clocked at 1 Ghz



One issue though is to scale
the memory-to-CPU bandwidth

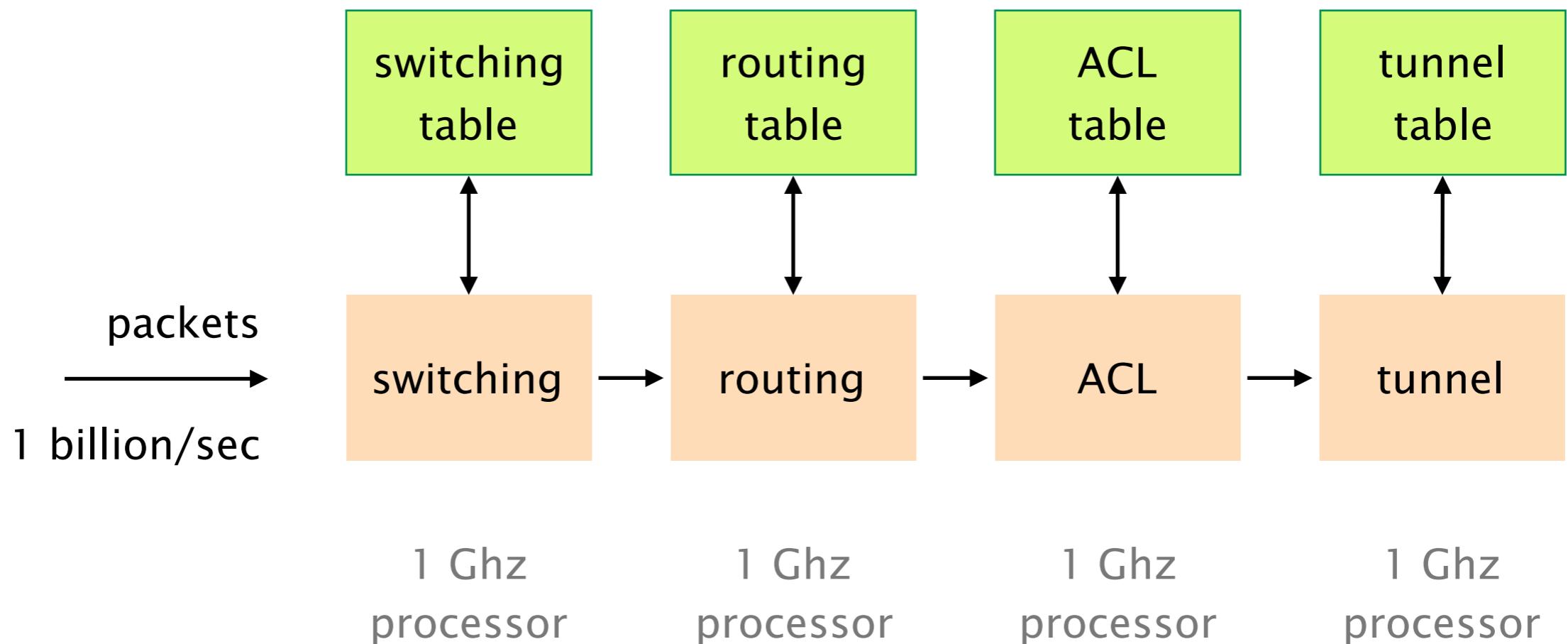


We could replicate the memory of course...
but that comes at **a huge costs in die area**

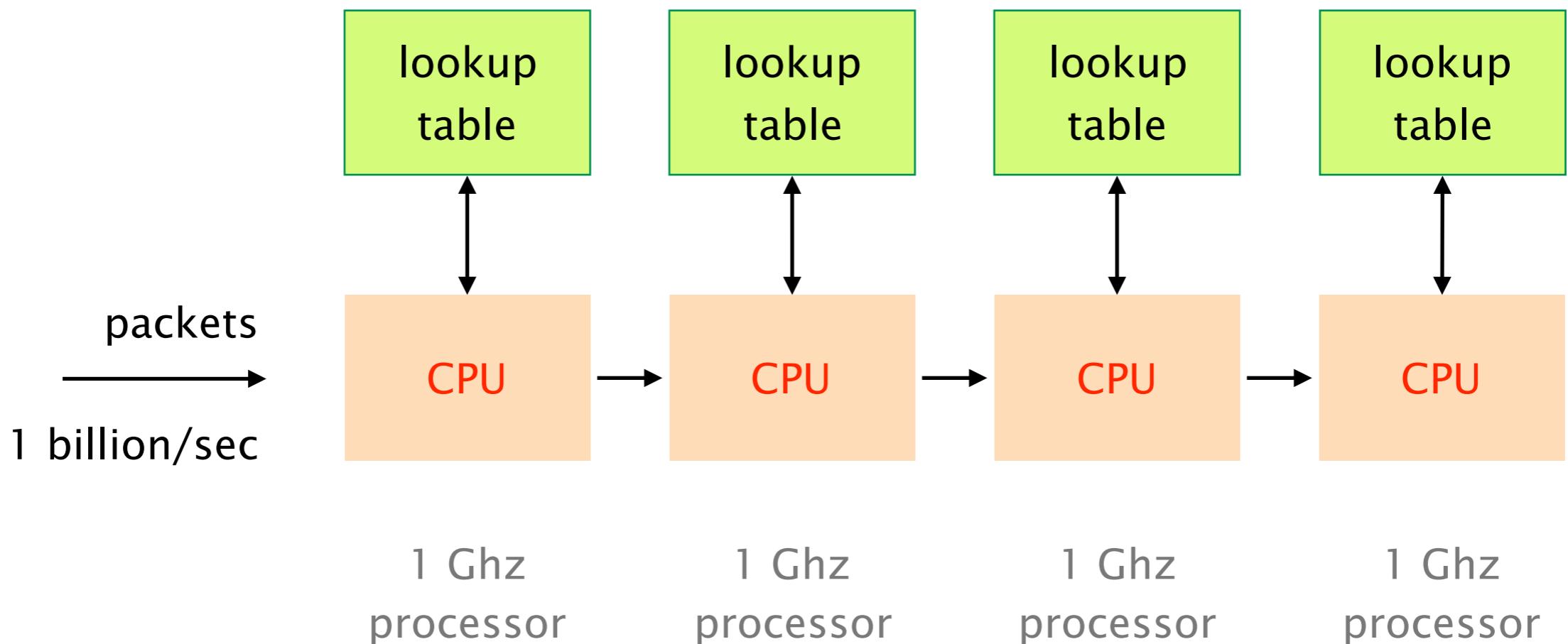


What if we organize the processing
as a **pipeline** instead?

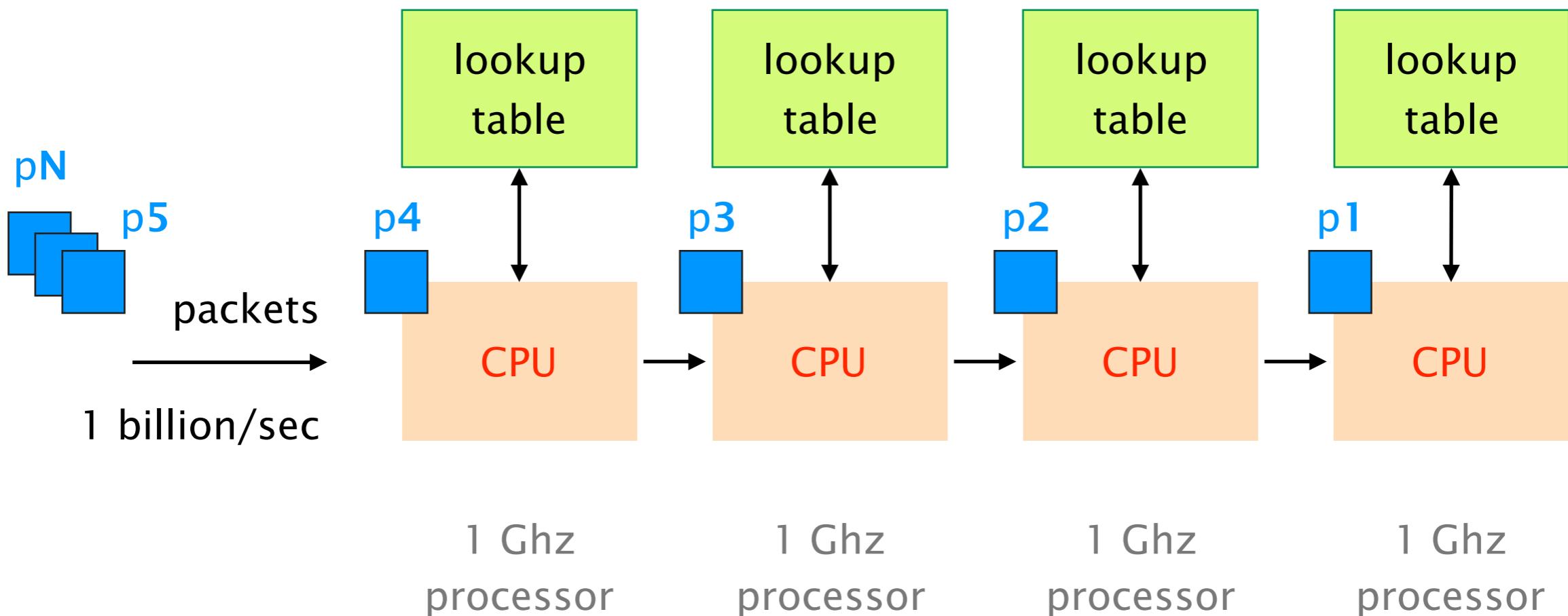
Pipelined architectures organize processing through a sequence of processing units and local memory



For flexibility,
each processing unit/memory can be made generic



Each CPU can process distinct packets, with up to 10 packets going through the pipeline simultaneously



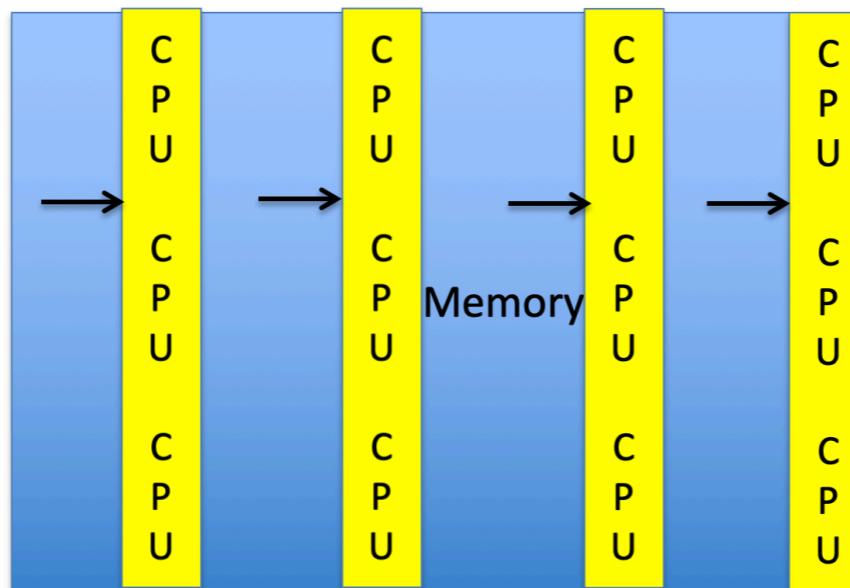
Performance vs Flexibility

- Multiprocessor: memory bottleneck
- Change to pipeline
- Fixed function chips specialize processors
- Flexible switch needs general purpose CPUs

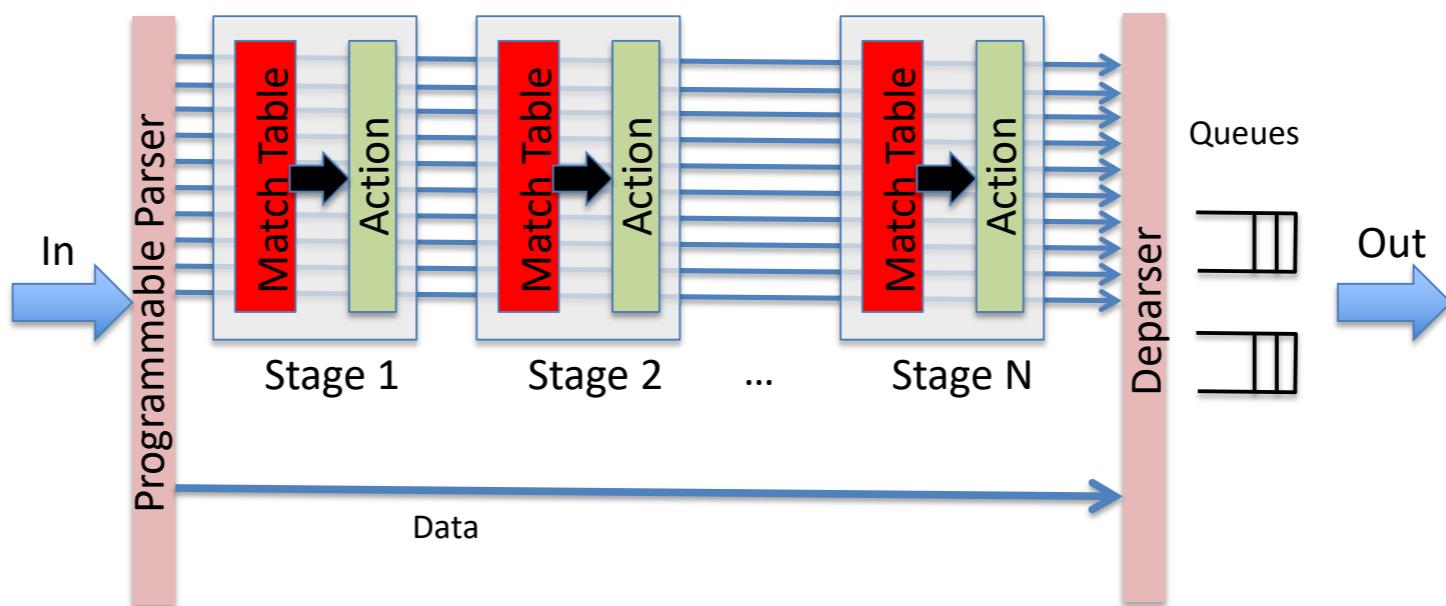


How We Did It

- Memory to CPU bottleneck
- Replicate CPUs
- More stages for finer granularity
- Higher CPU cost ok



Match/Action Forwarding Model



The runtime behavior of the parser & the match stages is defined through the RMT abstract model

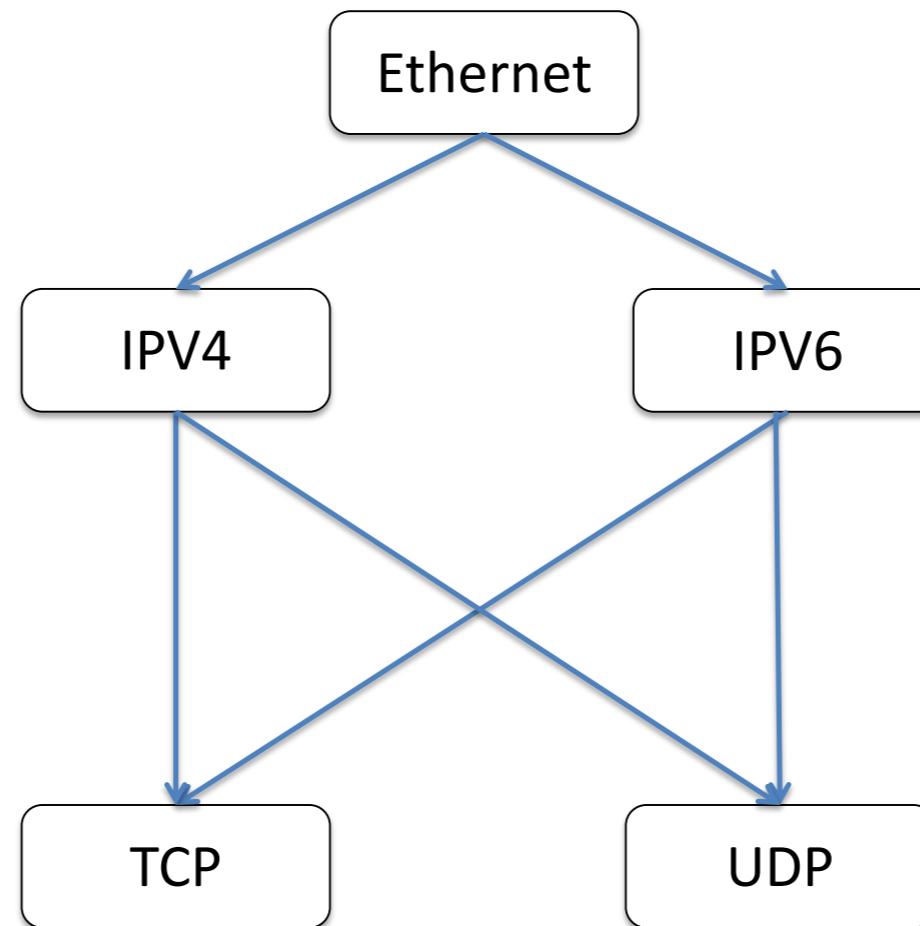
The RMT Abstract Model

- Parse graph
- Table graph

The parse graph contains nodes which corresponds to a header field and identifies the next field that follows

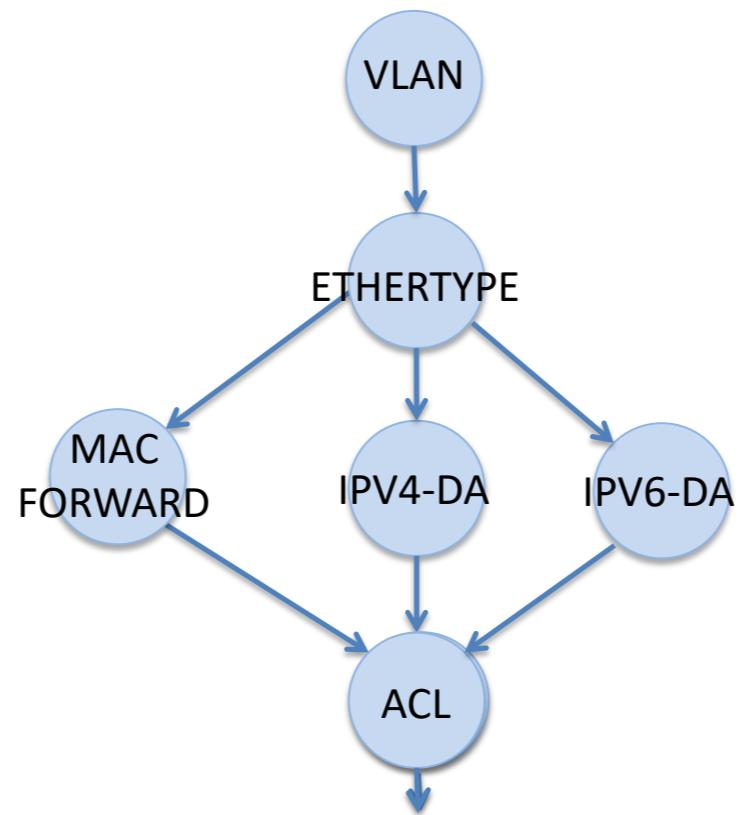
Arbitrary Fields: The Parse Graph

Packet: Ethernet IPV4 TCP



The table graph contains nodes,
each of which represents a match table

Reconfigurable Match Tables: The Table Graph



How do we implement in hardware
a programmable parser and a logical pipeline?

How do we implement in hardware a programmable parser and a logical pipeline?

The screenshot shows the first page of a PDF document titled 'Design Principles for Packet Parsers'. The page includes the authors' names (Glen Gibb, George Varghese, Mark Horowitz, Nick McKeown), their institutions (Stanford University, Microsoft Research), and their email addresses. The abstract discusses the challenges of packet parsing in network devices. Categories and subject descriptors are listed, along with keywords. The introduction section is also visible.

Design Principles for Packet Parsers

Glen Gibb[†], George Varghese[‡], Mark Horowitz[†], Nick McKeown[†]
†Stanford University ‡Microsoft Research
`{grg, horowitz, nickm}@stanford.edu` `varghese@microsoft.com`

ABSTRACT

All network devices must parse packet headers to decide how packets should be processed. A 64×10 Gb/s Ethernet switch must parse one billion packets per second to extract fields used in forwarding decisions. Although a necessary part of all switch hardware, very little has been written on parser design and the trade-offs between different designs. Is it better to design one fast parser, or several slow parsers? What is the cost of making the parser reconfigurable in the field? What design decisions most impact power and area?

In this paper, we describe trade-offs in parser design, identify design principles for switch and router designers, and describe a parser generator that outputs synthesizable Verilog that is available for download. We show that i) packet parsers today occupy about 1-2% of the chip, and ii) while future packet parsers will need to be programmable, this only doubles the (already small) area needed.

Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*Network Communications*

Keywords

Parsing; Design principles; Reconfigurable parsers

1. INTRODUCTION

Despite their variety, *every* network device examines fields

The diagram illustrates the structure of a TCP packet. It starts with an 'Ethernet' layer (Len: 14B) at the bottom, followed by an 'IPv4' layer (Len: 20B), then a 'TCP' layer (Len: 20B), and finally a 'Payload' layer. Arrows point from left to right between the layers, indicating the flow of the packet. The 'Next:' field is shown in red for the IPv4 and TCP layers.

Figure 1: A TCP packet.

In practice, packets often contain many more headers. These extra headers carry information about higher level protocols (e.g., HTTP headers) or additional information that existing headers do not provide (e.g., VLANs¹ in a college campus, or MPLS² in a public Internet backbone). It is common for a packet to have eight or more different packet headers during its lifetime.

To parse a packet, a network device has to identify the headers in sequence before extracting and processing specific fields. A packet parser seems straightforward since it knows *a priori* which header types to expect.

In practice, designing a parser is quite challenging:

1. **Throughput.** Most parsers must run at line-rate, supporting continuous minimum-length back-to-back packets. A 10 Gb/s Ethernet link can deliver a new packet every 70 ns; a state-of-the-art Ethernet switch ASIC with 64×40 Gb/s ports must process a new packet every 270 ps.

[ANCS'13]

Parsing is the (complex) process of identifying and extracting the appropriate fields in a packet header

Throughput

Parser must run at line-rate
parse 1 packet every 70 ns on a 10 Gbps link

Dependency

Parsing involves sequential processing
as headers typically point to the next one

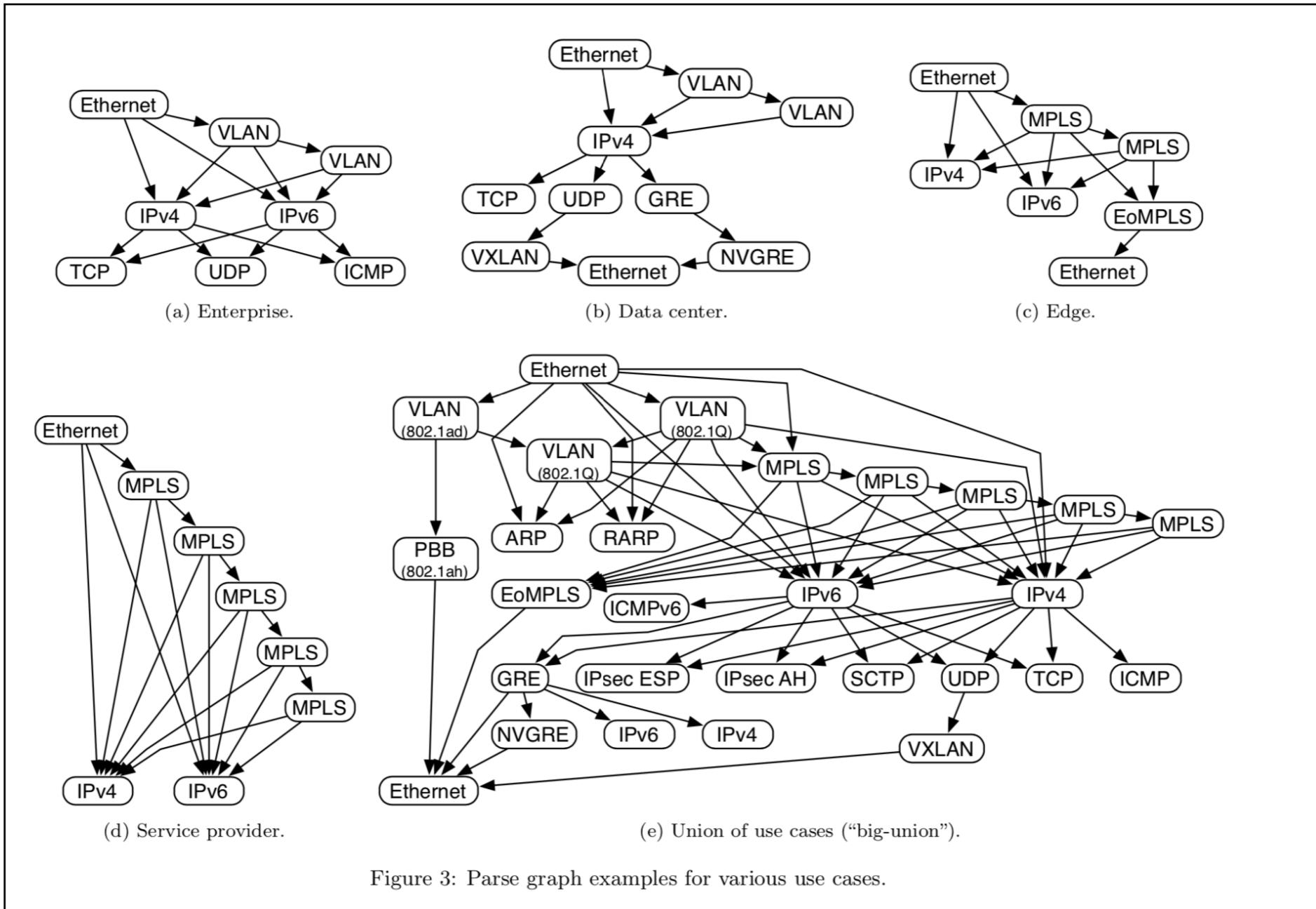
Incompleteness

Some headers do not even identify
the subsequent header

Heterogeneity

Many header formats exist that
can appear in various orders/locations

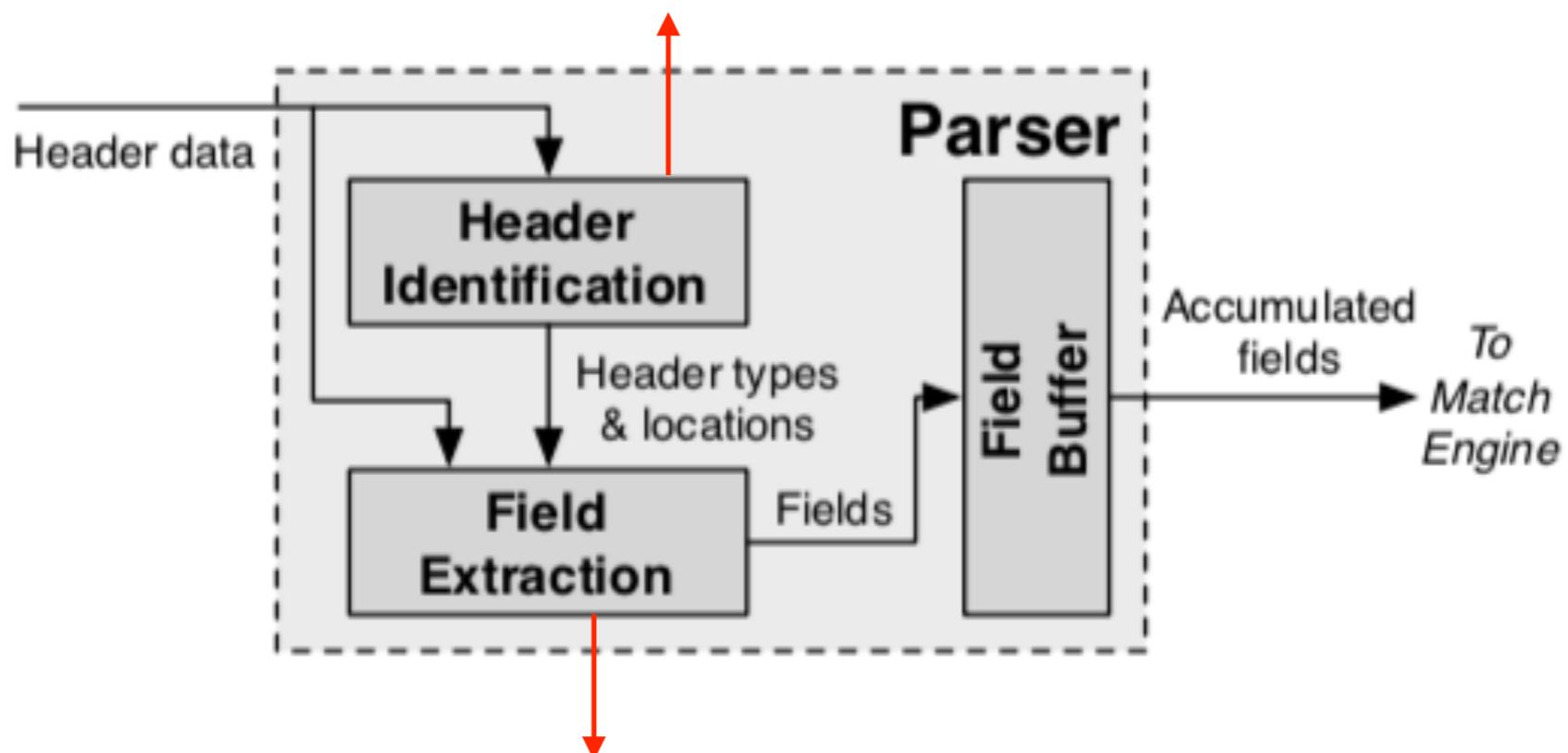
Parse graphs are directed acyclic graphs encoding header types and their sequence



Source: Design Principles for Packet Parsers, Gibb et al.

A parser can be divided into two separate blocks:
header identification and field extraction

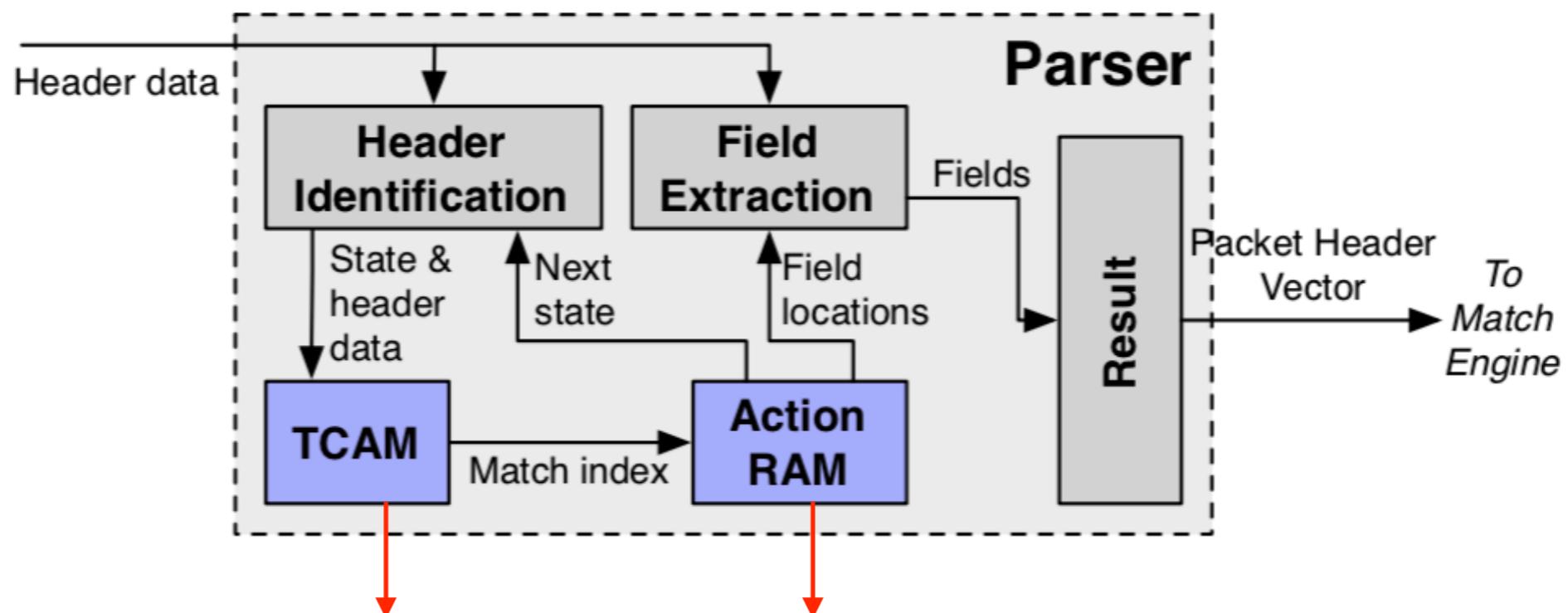
implements the parse graph's
state machine



extracts the chosen fields
from identified headers

In a programmable parser, the two modules rely on
runtime information instead of hard-coded logic

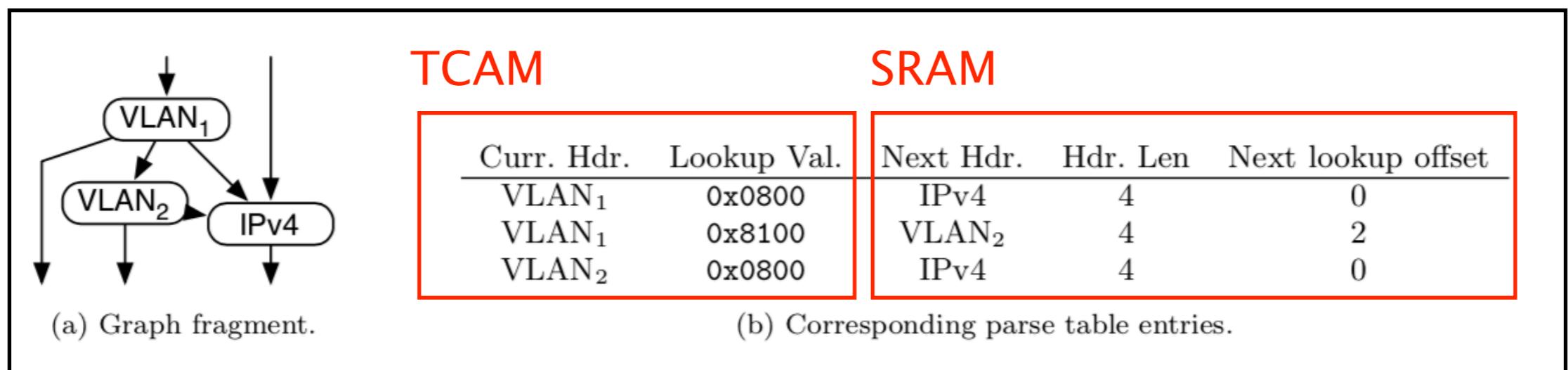
stored in memory,
e.g. in RAM and/or TCAM



stores the bit sequences
that identify the headers

stores the next state,
the fields to extract,
and any other data (if any)

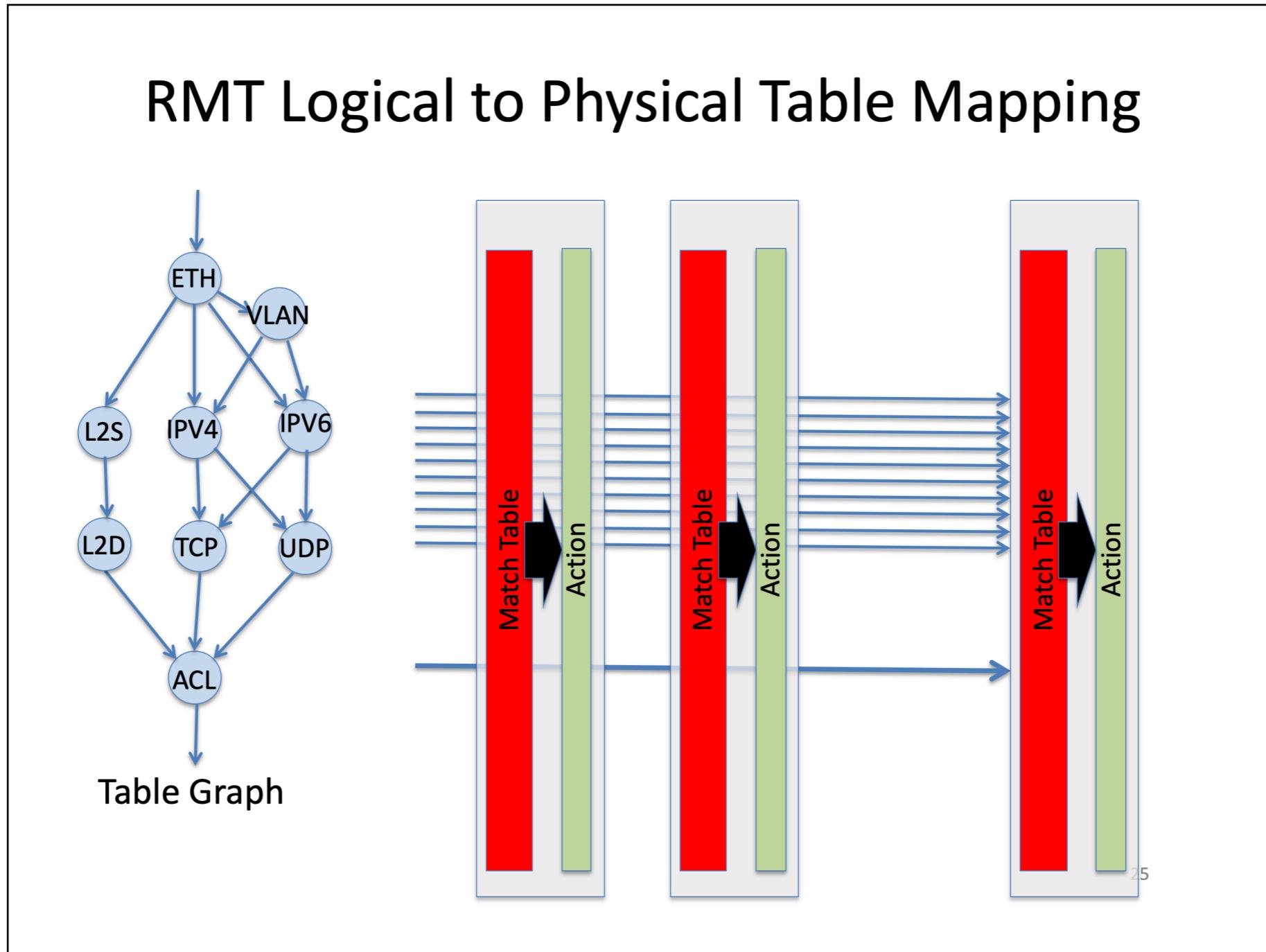
Linked together, a SRAM and TCAM can encode the transition table attached to a parsing graph



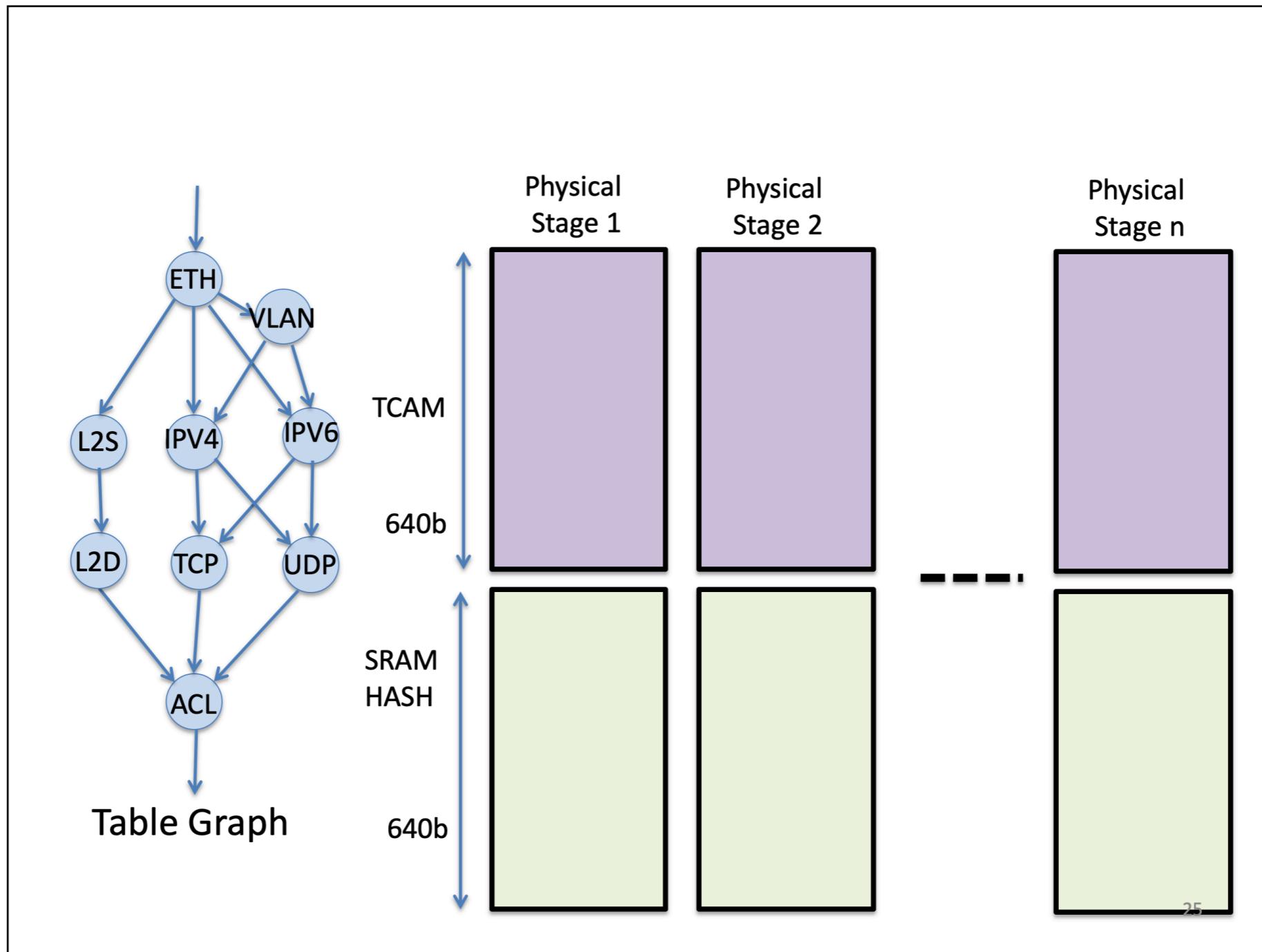
Source: Design Principles for Packet Parsers, Gibb et al.

How do we implement in hardware
a programmable parser and a logical pipeline?

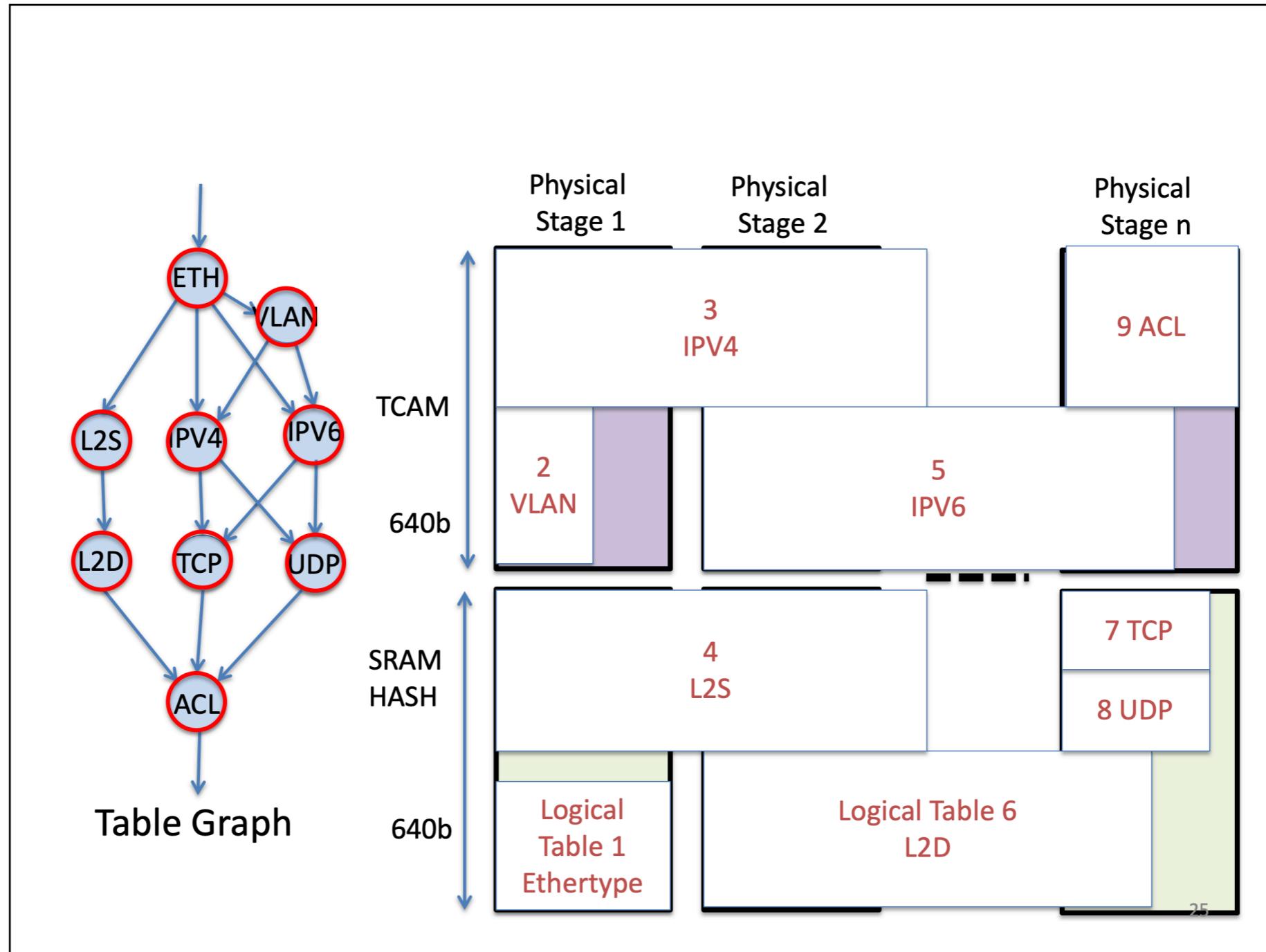
A compiler translates a given RMT logical pipeline (specified in P4) into a physical one



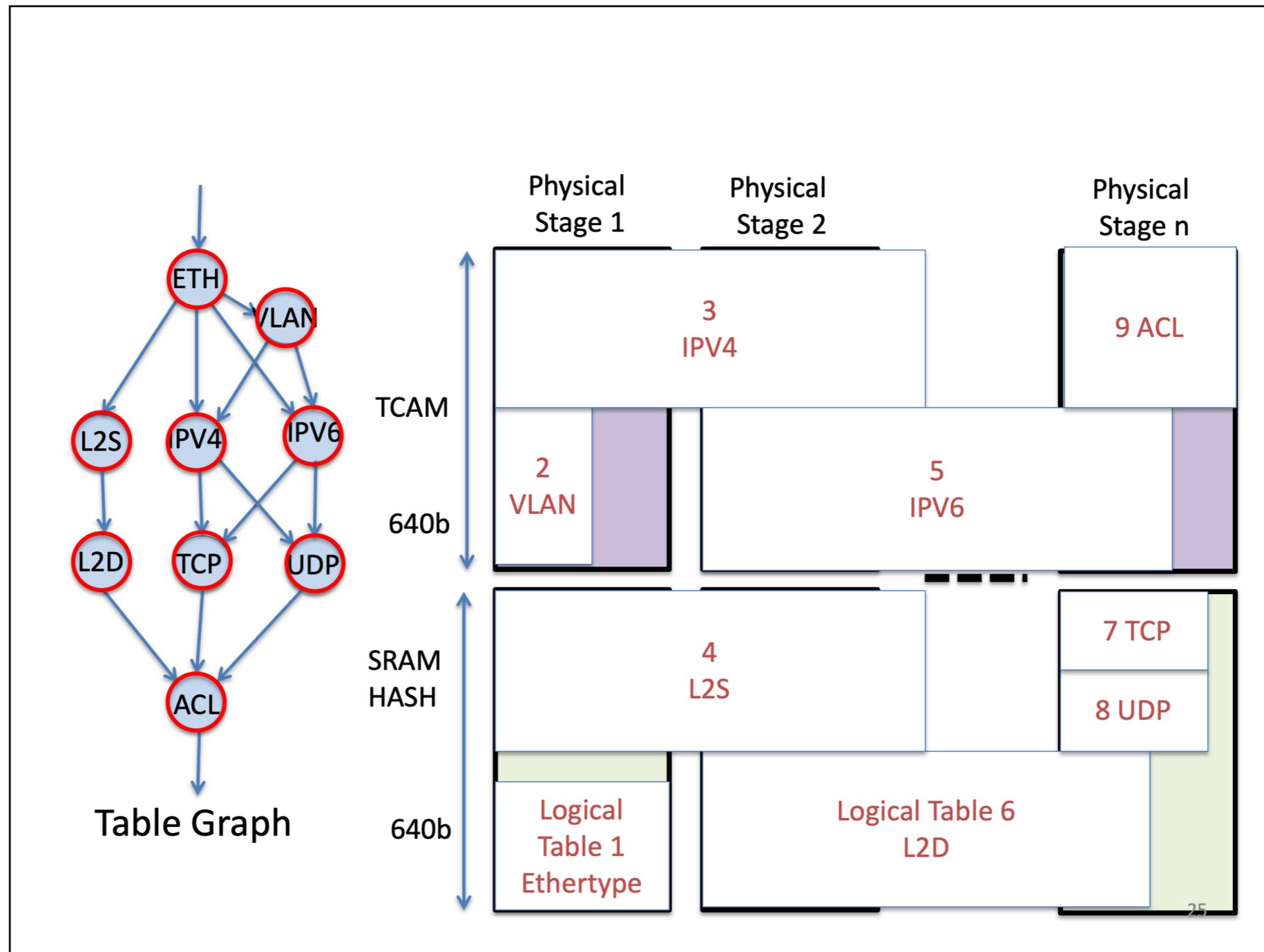
Each physical stage contains dedicated SRAM,
for exact matches, and TCAM, for ternary matches



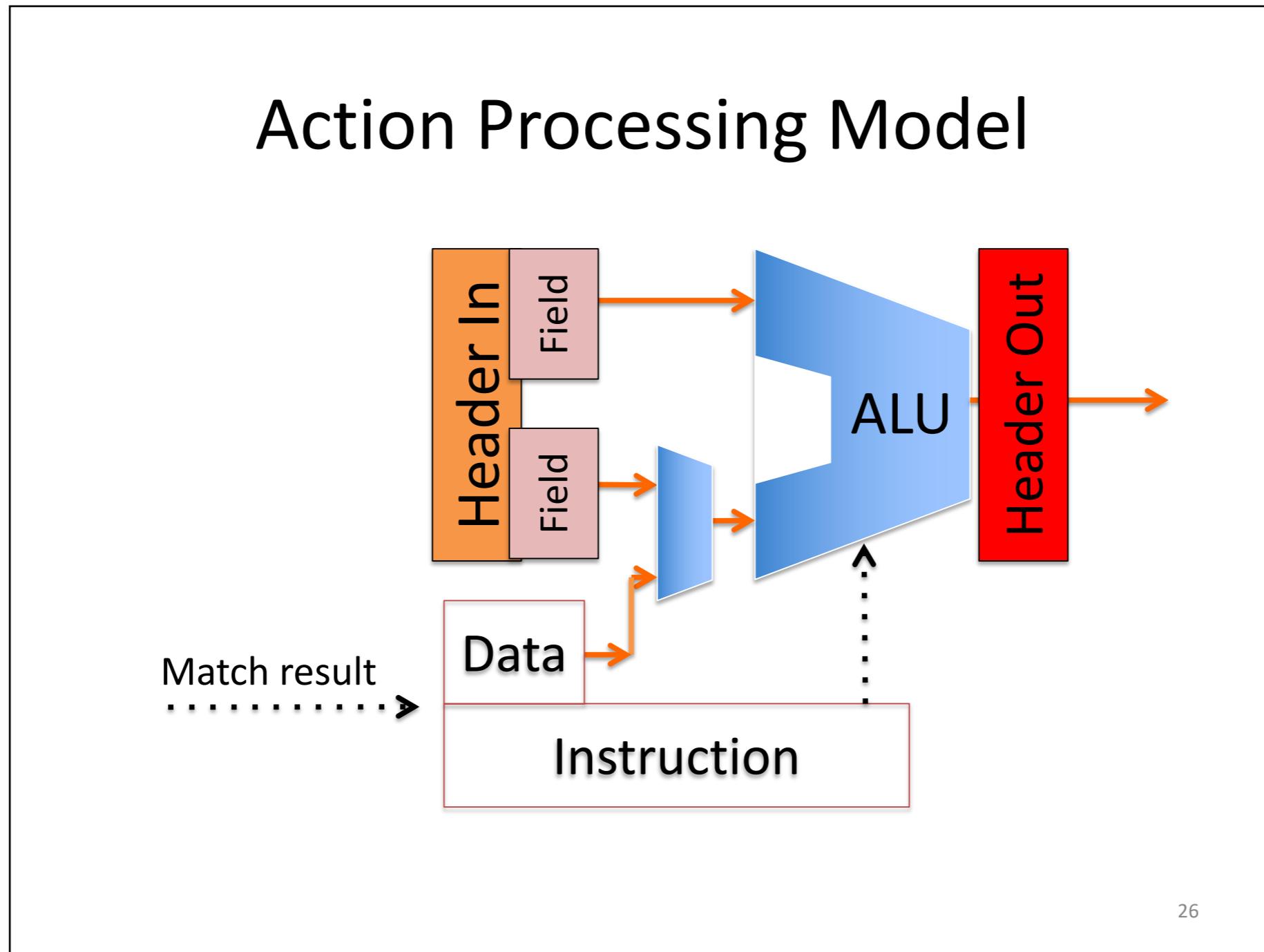
The compiler maps each individual logical stage to one or more physical stage.



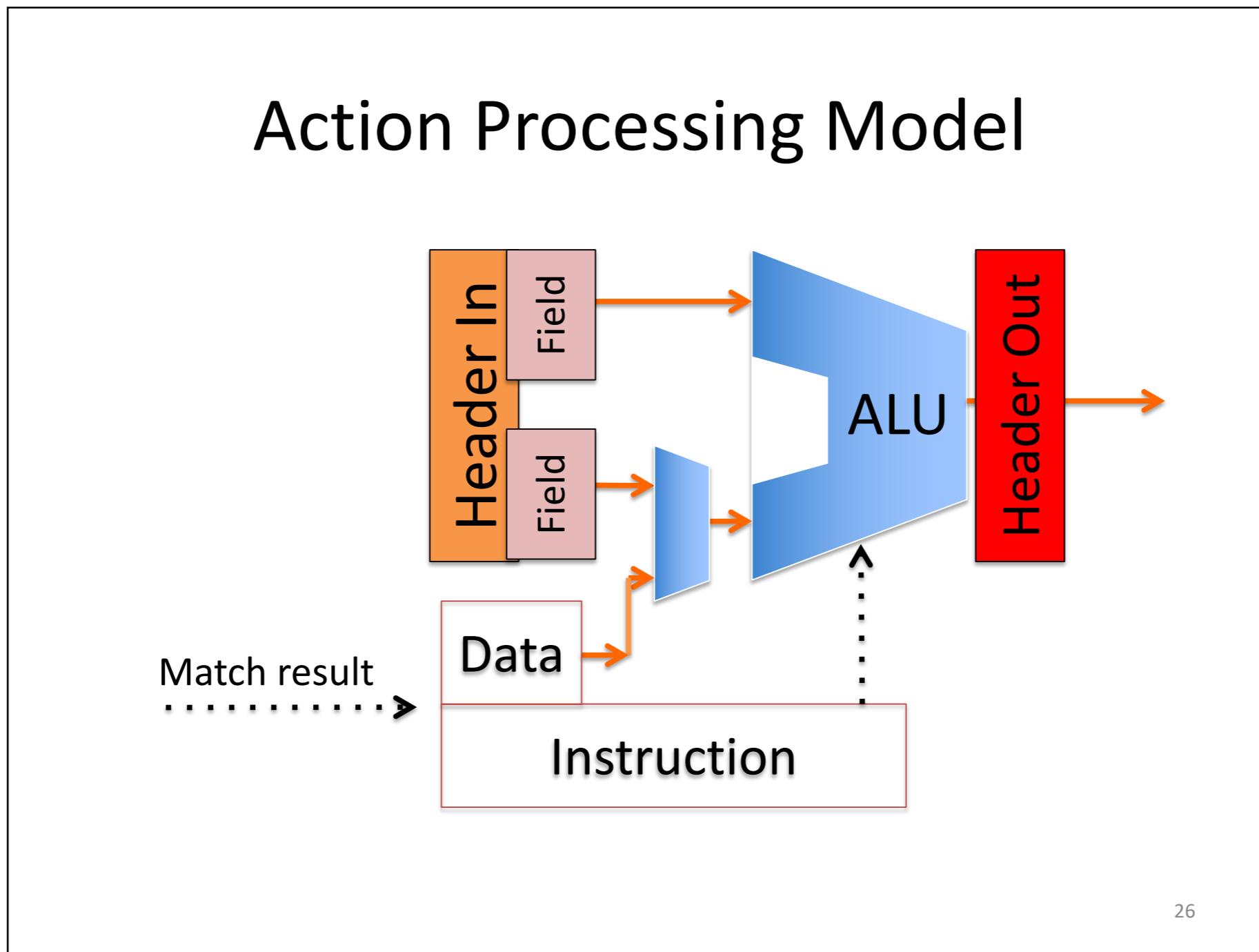
Small tables can share a stage (up to 16 per stage),
while large tables can span multiple ones



The RMT pipeline relies on many Arithmetic Logic Units (ALU) to perform actions on the result of a match

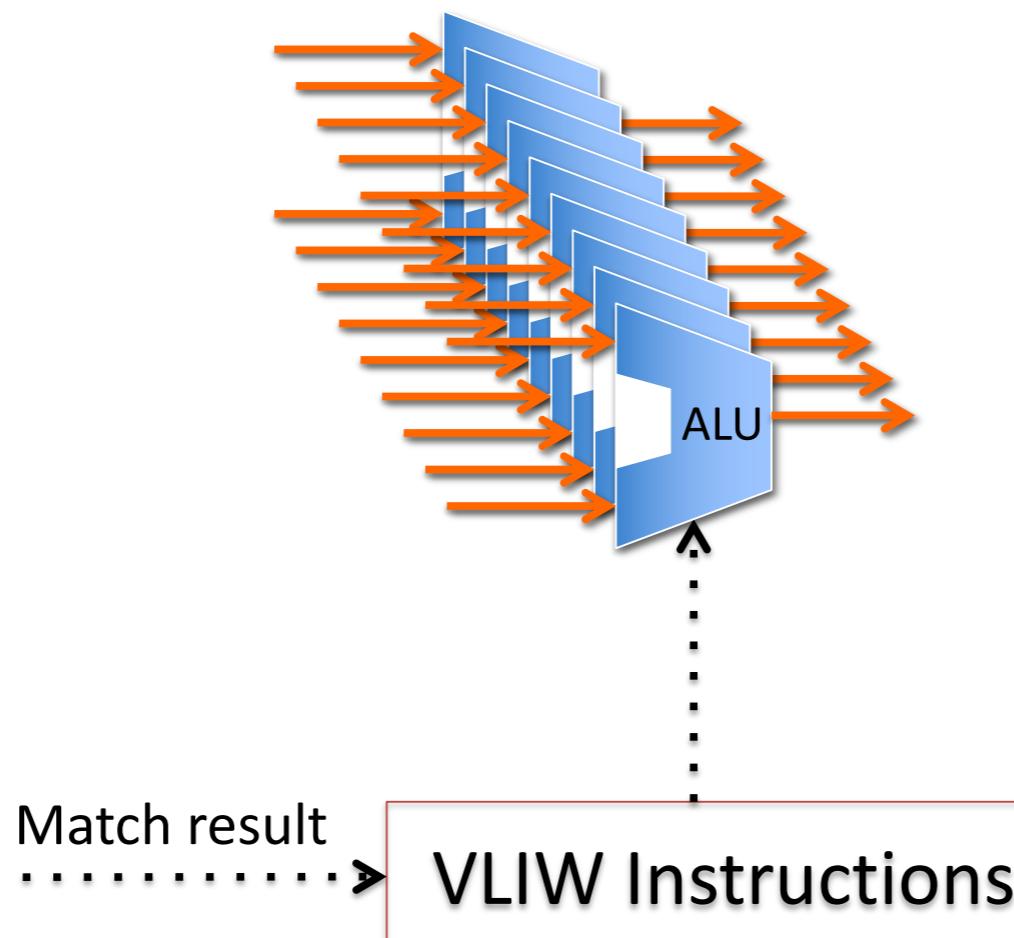


Each ALU modifies only one word of a header
(a header is composed of *many* words)



Each stage of the RMT pipeline contains one ALU per word of the header vector (that's *a lot* of ALUs)

Modeled as Multiple VLIW CPUs per Stage



The RMT pipeline in a few statistics

Our Switch Design

- 64 x 10Gb ports
 - 960M packets/second
 - 1GHz pipeline
- Programmable parser
- 32 Match/action stages
- Huge TCAM: 10x current chips
 - 64K TCAM words x 640b
- SRAM hash tables for exact matches
 - 128K words x 640b
- 224 action processors per stage
- All OpenFlow statistics counters

Building a RMT pipeline is **only 15% more expensive**
than building a fixed-function switching pipeline

Outline

- Conventional switch chip are inflexible
- SDN demands flexibility...sounds expensive...
- How do I do it: The RMT switch model
- **Flexibility costs less than 15%**

The biggest cost is the memory...
not the processing logic

Cost of Configurability: Comparison with Conventional Switch

- Many functions identical: I/O, data buffer, queueing...
- Make extra functions optional: statistics
- Memory dominates area
 - Compare memory area/bit and bit count
- RMT must use memory bits efficiently to compete on cost
- Techniques for flexibility
 - Match stage unit RAM configurability
 - Ingress/egress resource sharing
 - Table predication allows multiple tables per stage
 - Match memory overhead reduction
 - Match memory multi-word packing

In terms of die area, flexibility is not very expensive
at least, not anymore... mainly thanks to Moore's law

Chip Comparison with Fixed Function Switches

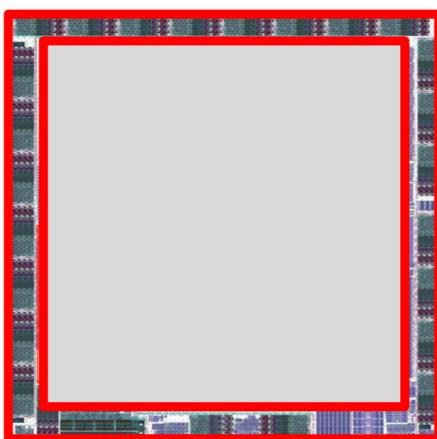
Area

Section	Area % of chip	Extra Cost
IO, buffer, queue, CPU, etc	37%	0.0%
Match memory & logic	54.3%	8.0%
VLIW action engine	7.4%	5.5%
Parser + deparser	1.3%	0.7%
Total extra area cost		14.2%

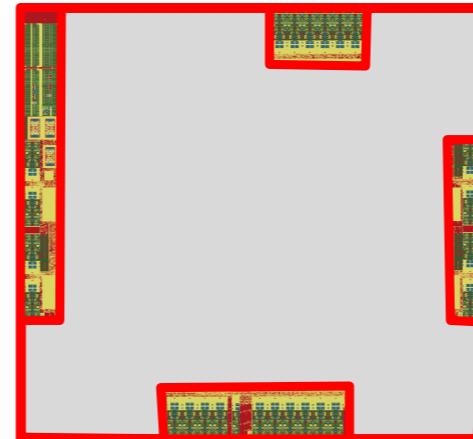


Serializer/Deserializer (SerDes) usually account for 30% of the area

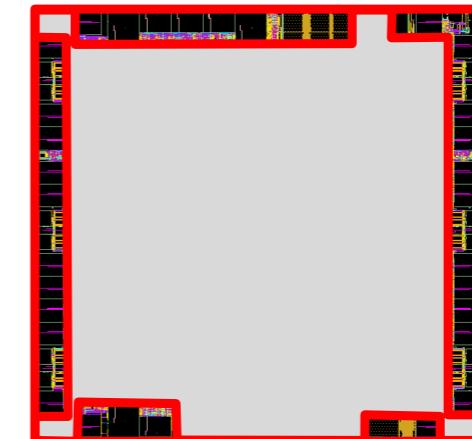
Serial I/O: About 30% of switch chip area



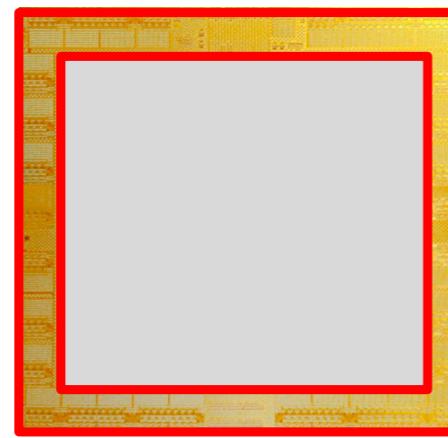
Intel Alta (2011)



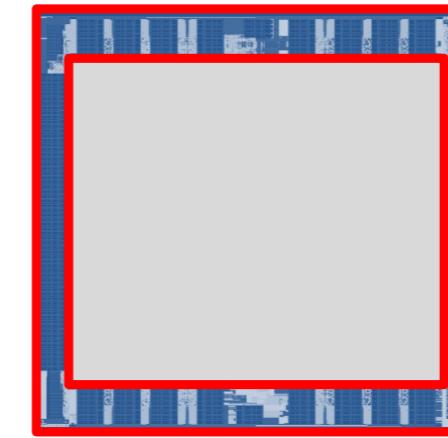
Cisco (2011)



Ericsson Spider (2011)

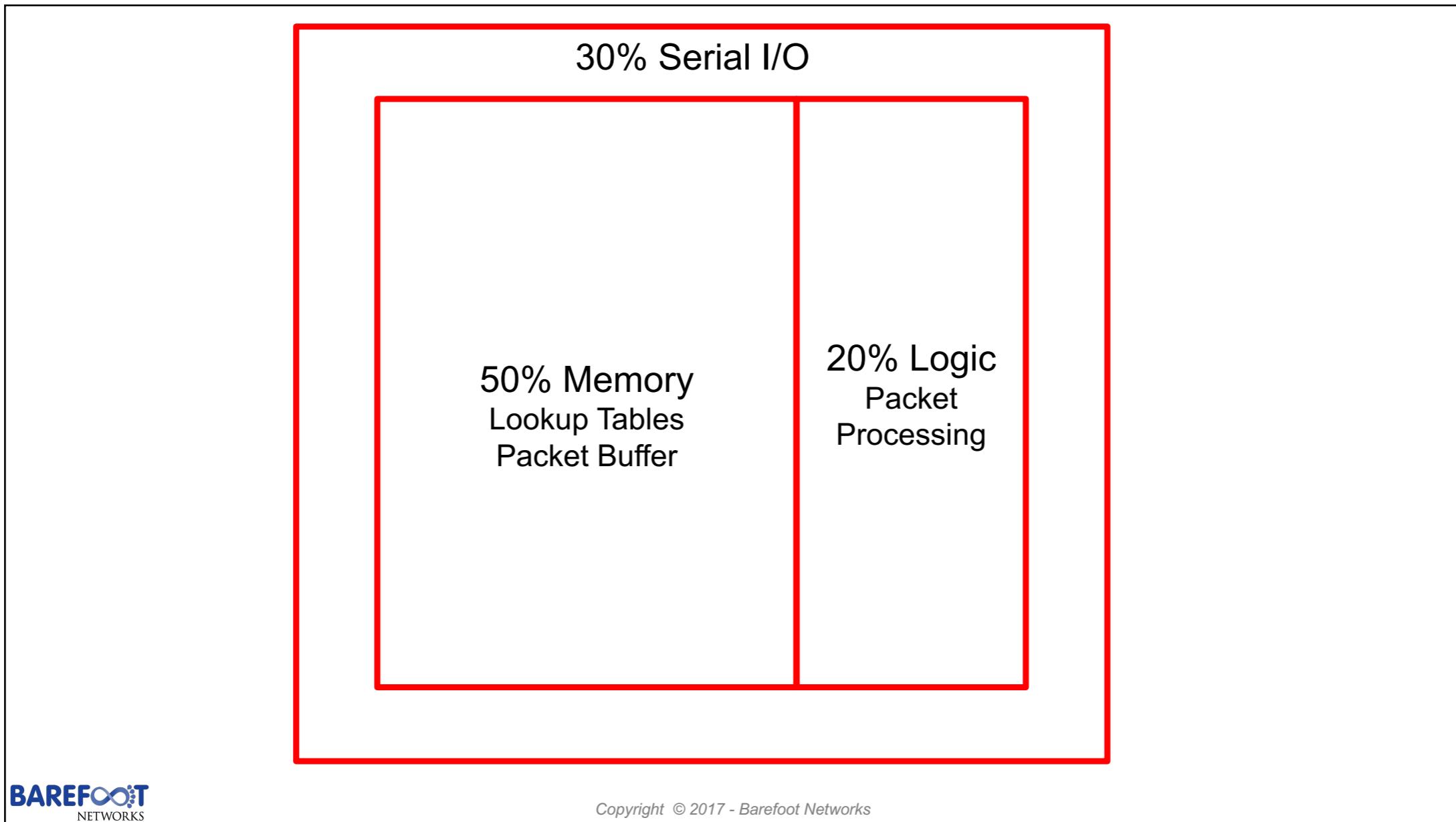


Broadcom Tomahawk (2014)



Barefoot Tofino (2016)

Memory usually account for ~50% of the die area,
leaving us around 20% for the processing logic



Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

As SerDes and memory technologies progress, the relative area dedicated to logic shrinks

Observations

With every new generation of network devices,
people expect larger speeds and more memory

Consequences

relative areas of SerDes/memory stay roughly equivalent
logic shrinks

Even with an increased space for logic,
the device tends to be relatively the same

Chip Comparison with Fixed Function Switches

Area

Section	Area % of chip	Extra Cost
IO, buffer, queue, CPU, etc	37%	0.0%
Match memory & logic	54.3%	8.0%
VLIW action engine	7.4%	5.5%
Parser + deparser	1.3%	0.7%
Total extra area cost		14.2%



The same lesson applies for power

Chip Comparison with Fixed Function Switches

Area		
Section	Area % of chip	Extra Cost
IO, buffer, queue, CPU, etc	37%	0.0%
Match memory & logic	54.3%	8.0%
VLIW action engine	7.4%	5.5%
Parser + deparser	1.3%	0.7%
Total extra area cost		14.2%

Power		
Section	Power % of chip	Extra Cost
I/O	26.0%	0.0%
Memory leakage	43.7%	4.0%
Logic leakage	7.3%	2.5%
RAM active	2.7%	0.4%
TCAM active	3.5%	0.0%
Logic active	16.8%	5.5%
Total extra power cost		12.4%

Conclusion

- How do we design a flexible chip?
 - The RMT switch model
 - Bring processing close to the memories:
 - pipeline of many stages
 - Bring the processing to the wires:
 - 224 action CPUs per stage
- How much does it cost?
 - 15%
- Lots of the details how we designed this in 28nm CMOS are in the paper

That was just an academic paper
Let's look at a real flexible pipeline



A small subset of our lab @ITET with two Tofino 3.2 Tbps, 32x 100 GbE QSFP28

That was just an academic paper
Let's look at a real flexible pipeline



Programmable Data Plane at Terabit Speeds

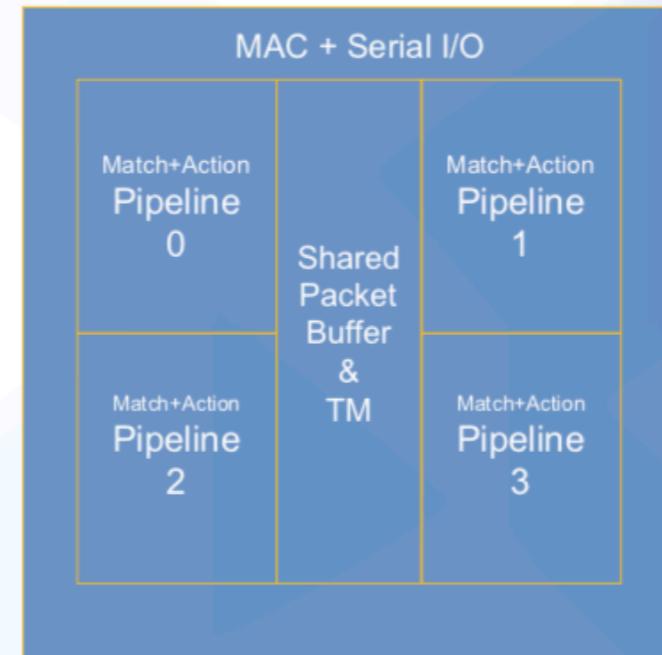
Vladimir Gurevich
May 16, 2017

Barefoot Tofino 6.5 Tbps backplane

several billion packets per second at line rate

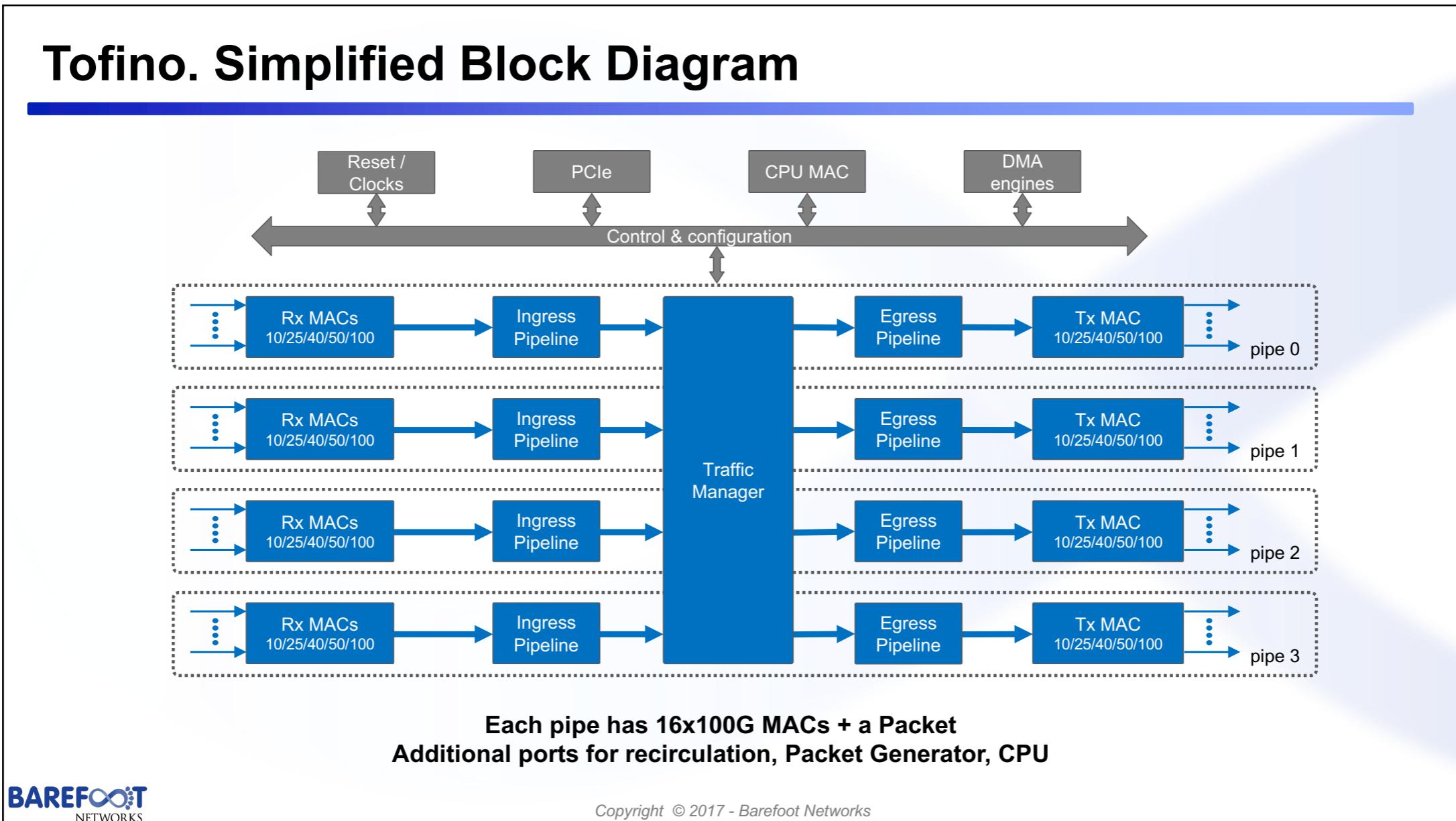
6.5Tb/s Tofino™ Summary

- **State of the art design**
 - Single Shared Packet Buffer
 - TSMC 16nm FinFET+
- **Four Match+Action Pipelines**
 - Fully programmable PISA Embodiment
 - All compiled programs run at line-rate.
 - Up to 1.3 million IPv4 routes
- **Port Configurations**
 - 65 x 100GE/40GE
 - 130 x 50GE
 - 260 x 25GE/10GE
- **CPU Interfaces**
 - PCIe: Gen3 x4/x2/x1
 - Dedicated 100GE port



Barefoot Tofino 6.5 Tbps backplane

several billion packets per second at line rate

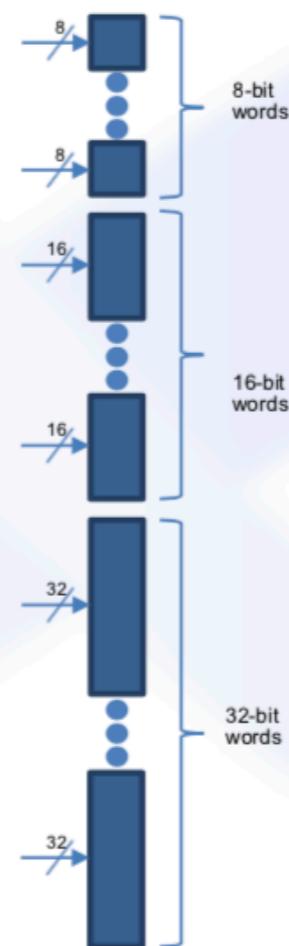


Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Tofino relies on Packet Header Vector (PHV) to pass states between stages

Packet Header Vector (PHV)

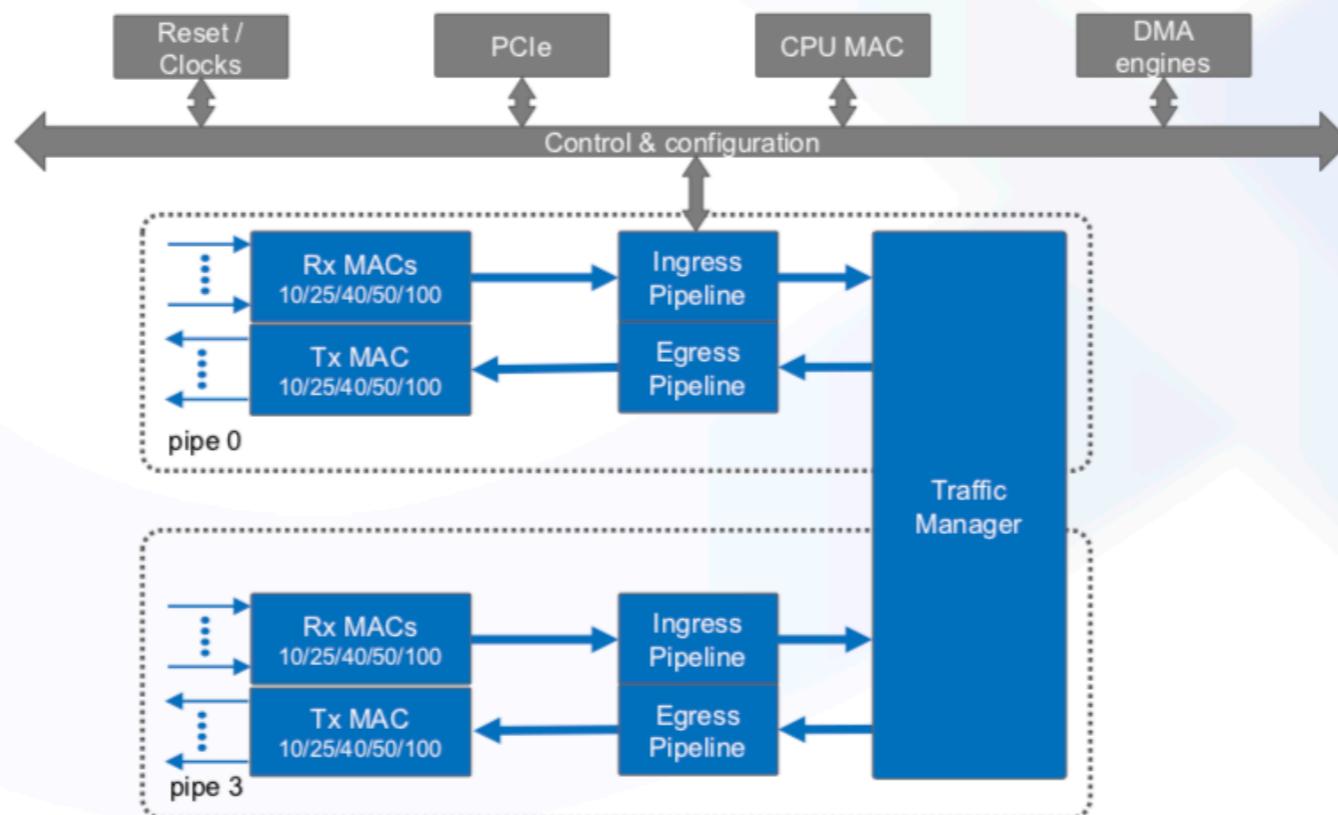
- A set of uniform containers that carry the headers and metadata along the pipeline
- Fields can be packed into any container or their combination
- PHV Allocation step in the compiler decides the actual packing



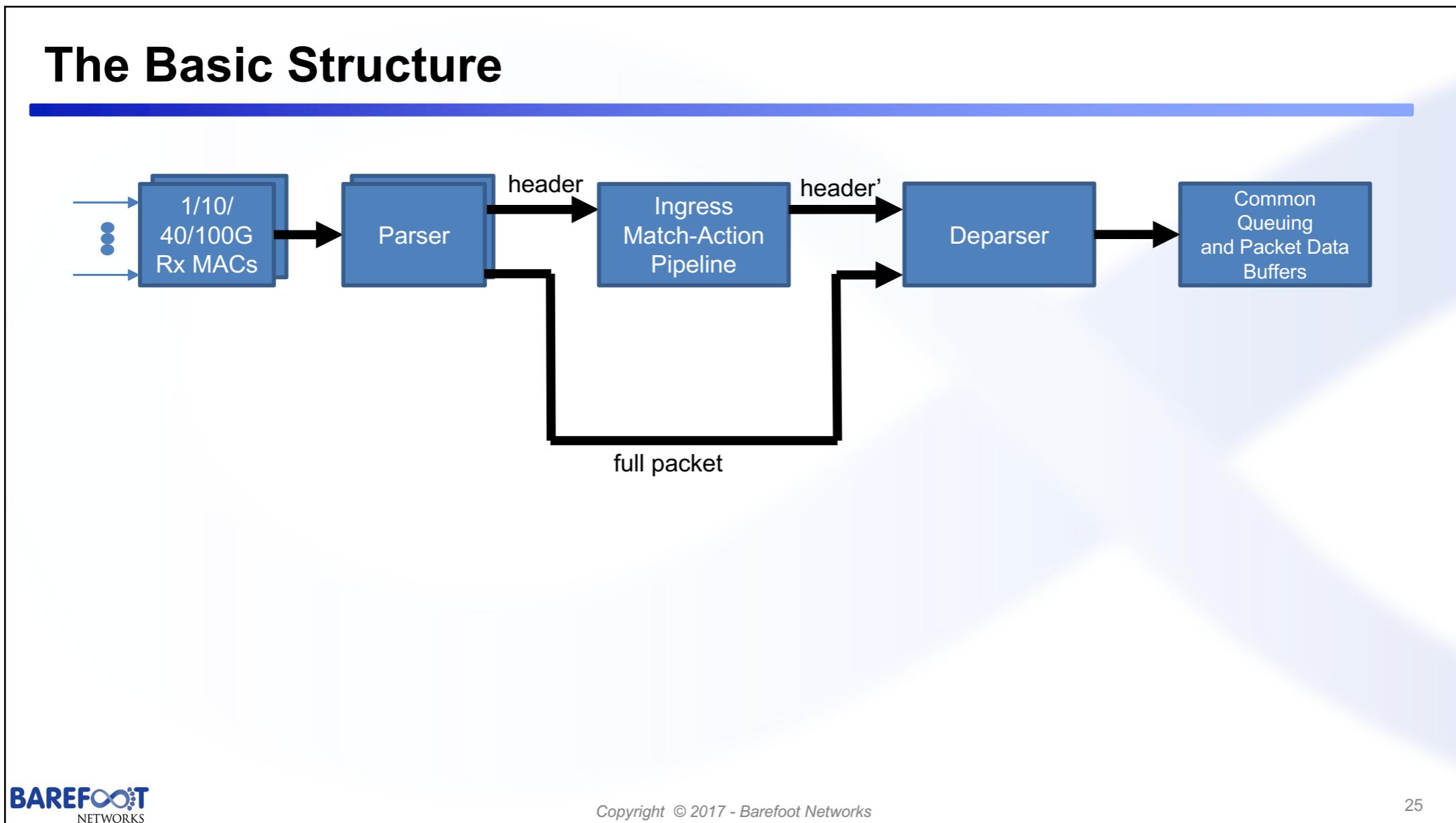
Tofino uses a folded pipeline in which the *same* stages are used for both the ingress and the egress pipeline

Unified Pipeline

- There is no difference between ingress and egress processing
 - The same blocks can be efficiently shared

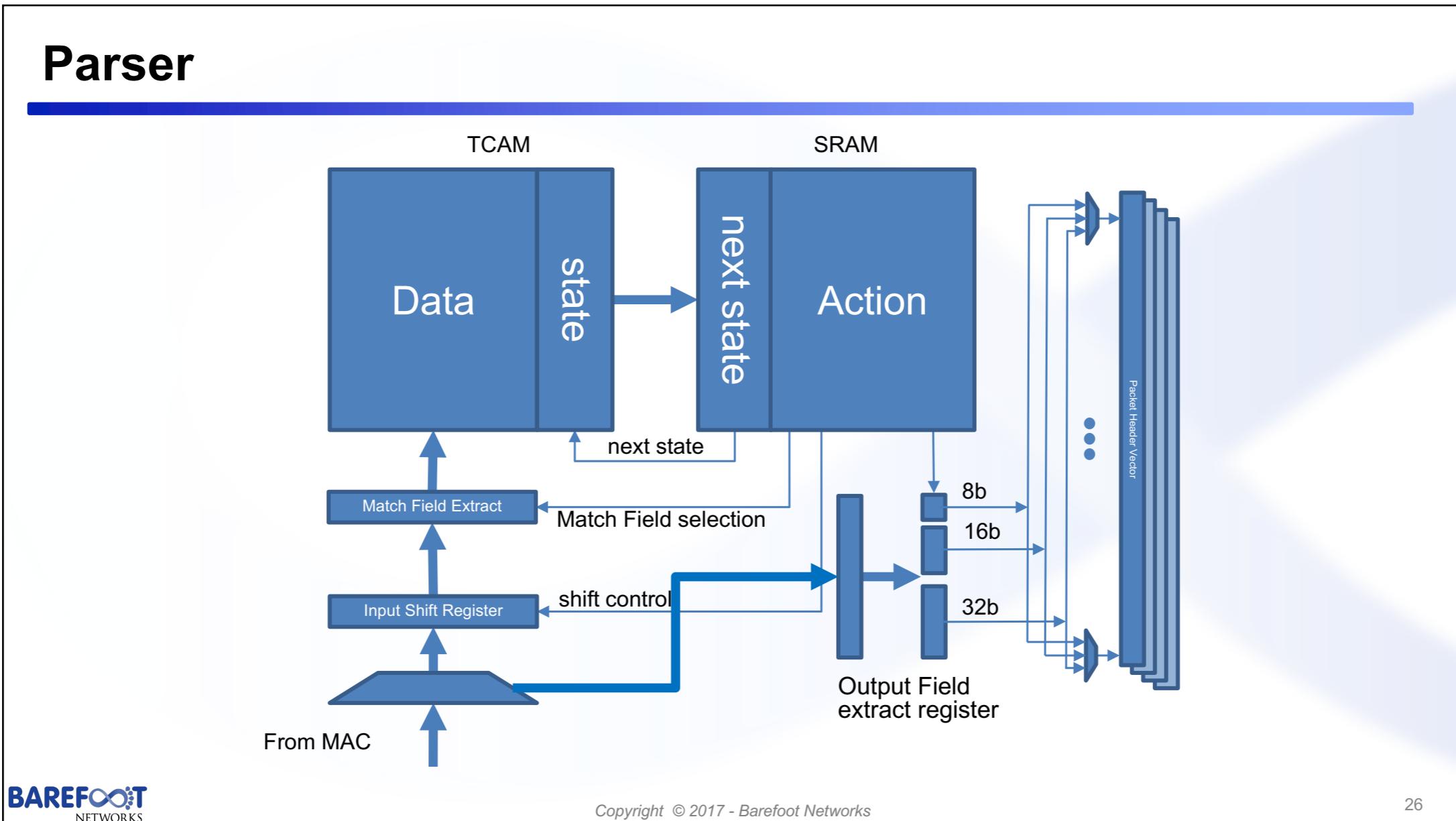


In terms of structure,
Tofino basically follows the RMT pipeline



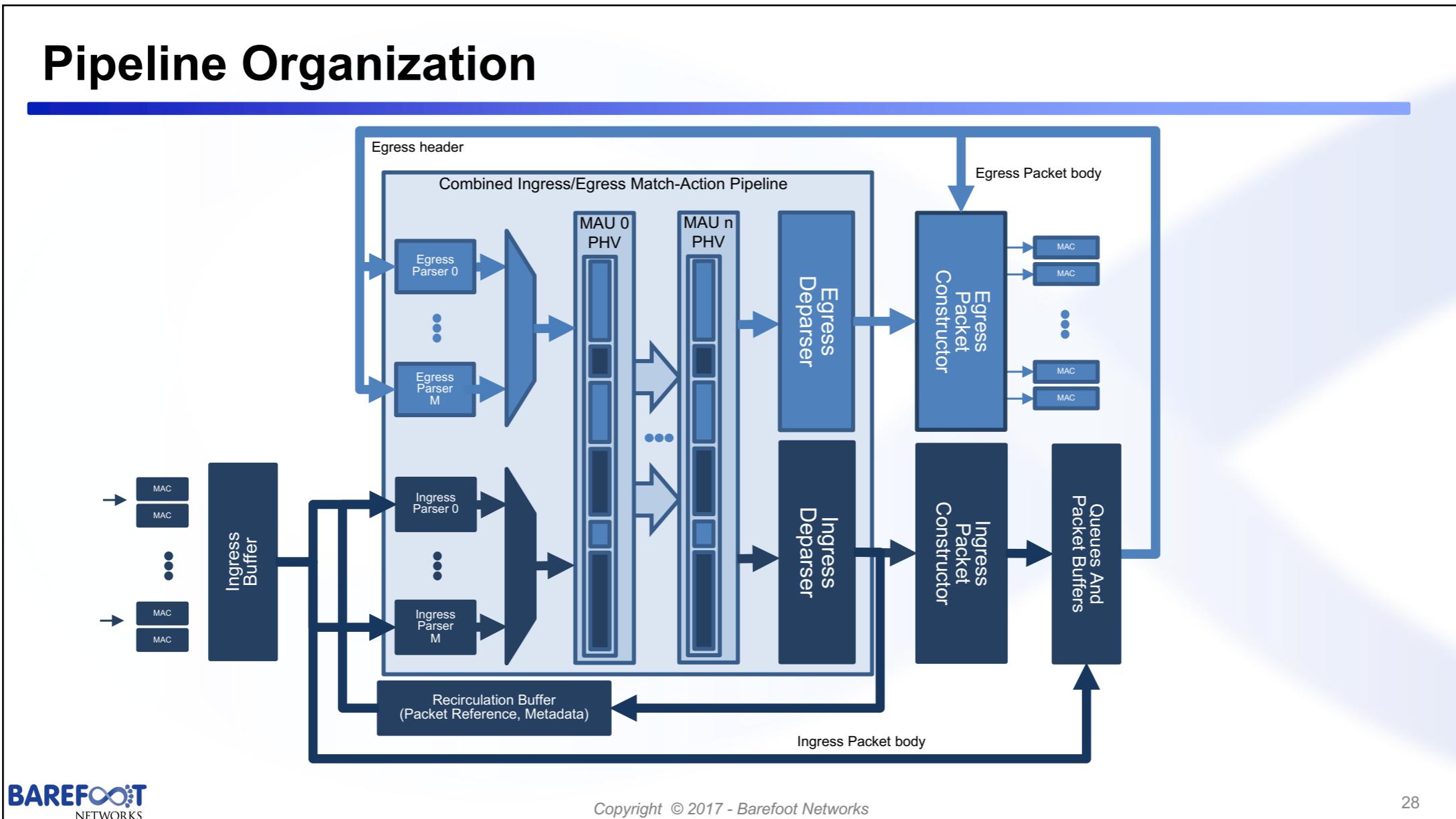
Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

The same goes for the design of the programmable parser



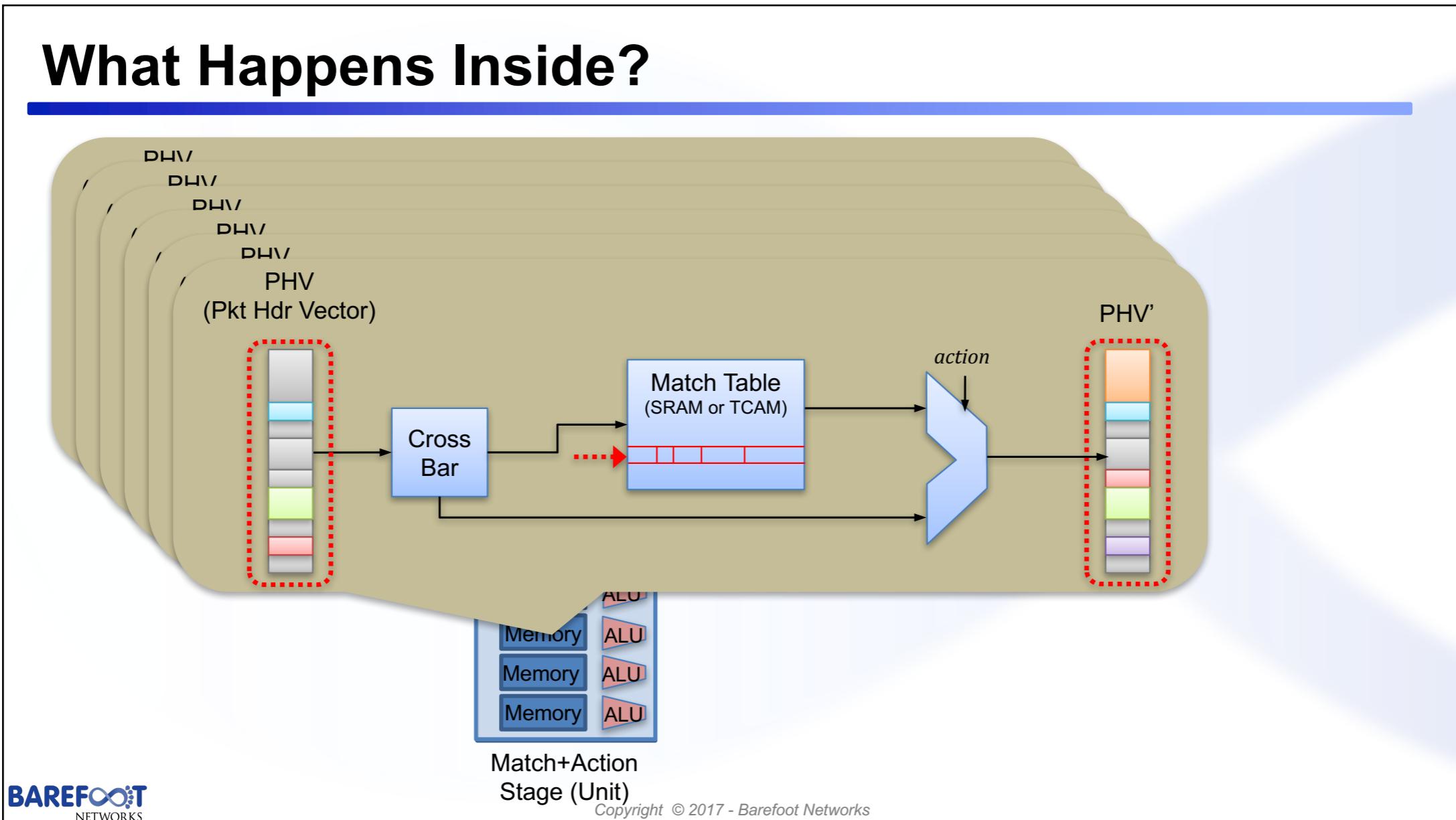
Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Putting everything together



Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Each match action stage is regularly structured around:
crossbars, memory units, and ALUs



Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Parallelism in P4

```
apply {
    /* Parallel lookups possible */
    subnet_vlan.apply();
    mac_vlan.apply();
    protocol_vlan.apply()
    port_vlan.apply();

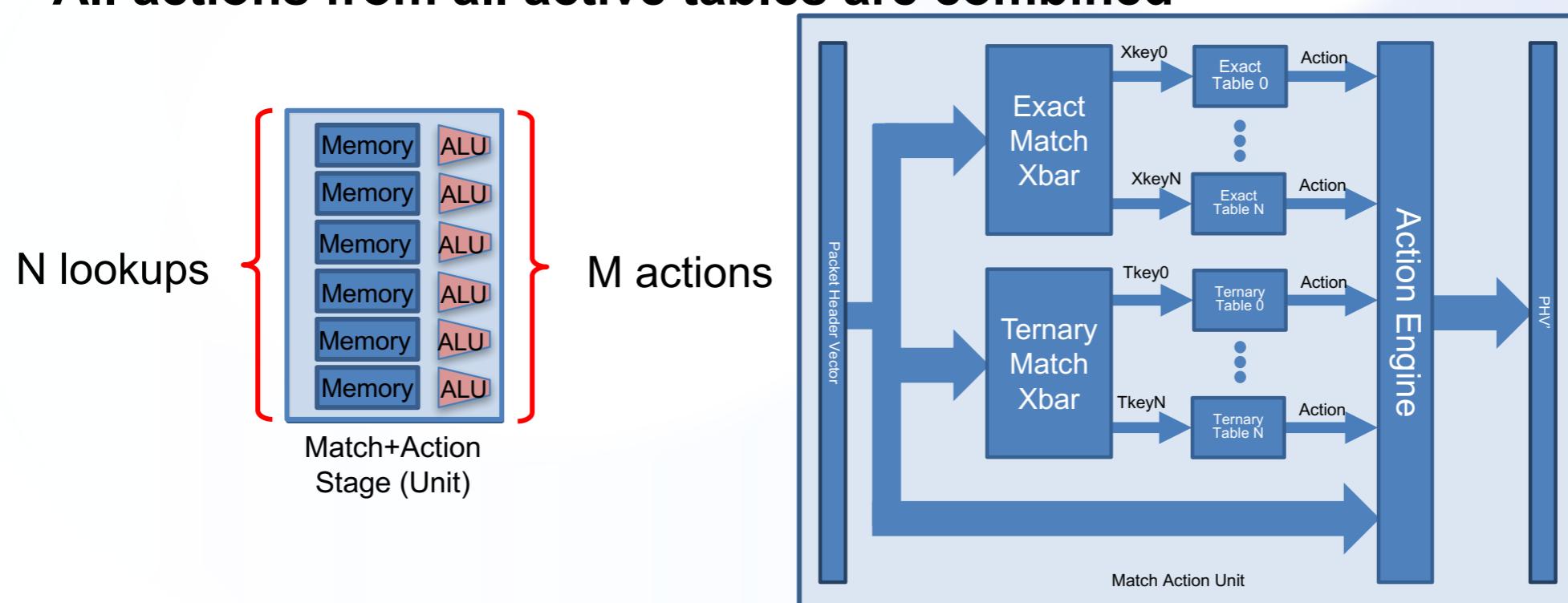
    /* Resolution in next stage */
    resolve_vlan.apply();
}

apply {
    if (!subnet_vlan.apply().hit) {
        if (!mac_vlan.apply().hit) {
            if (!protocol_vlan.apply().hit) {
                port_vlan.apply();
            }
        }
    }
}
```

- **Most P4 programs have inherent parallelism**
- **Others can be executed speculatively**
- **Switch.p4**
 - ~100 tables and if() statements
 - ~22 stages divided between ingress and egress
 - Degree of parallelism ~4.5

How Tofino Supports Parallel Processing

- Multiple tables mean multiple parallel lookups
- All actions from all active tables are combined



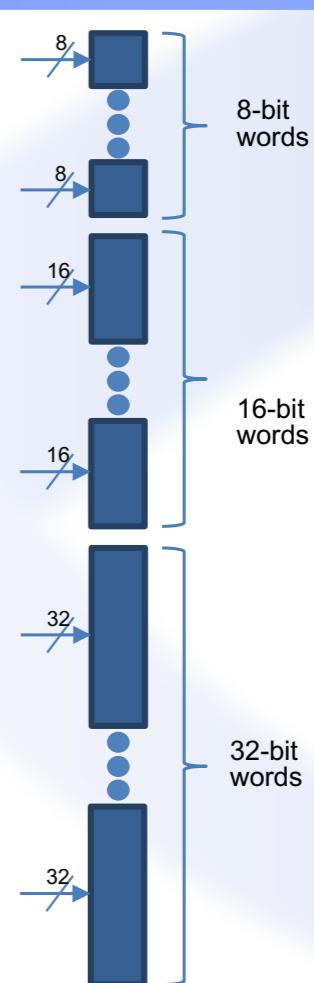
Parallelism in P4

```
action ipv4_in_mpls(in bit<20> label1, in bit<20> label2) {
    hdr.mpls[0].setValid();
    hdr.mpls[0].label = label1;
    hdr.mpls[0].exp  = 0;
    hdr.mpls[0].bos  = 0;
    hdr_mpls[0].ttl  = 64;

    hdr.mpls[1].setValid();
    hdr.mpls[1] = { label2, 0, 1, 128 };

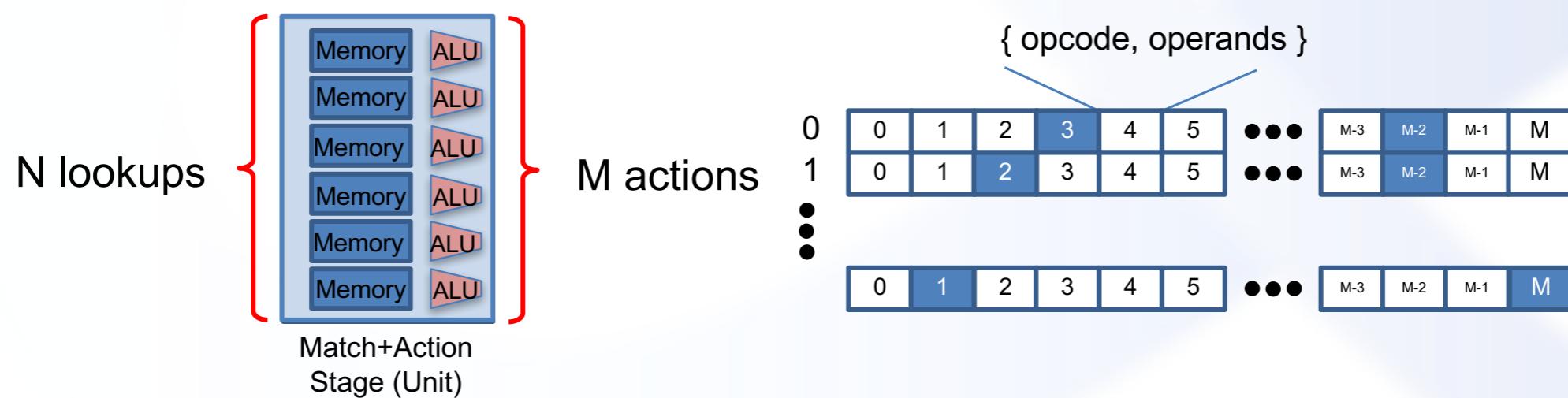
    if (hdr.vlan_tag.isValid()) {
        hdr.vlan_tag.etherType = 0x8847;
    } else {
        hdr.ethernet.etherType = 0x8847;
    }
}
```

- Most actions can be easily parallelized
 - This action can be executed in 1 cycle
 - Number of parallel operations: 12
- Keep fields in separate containers



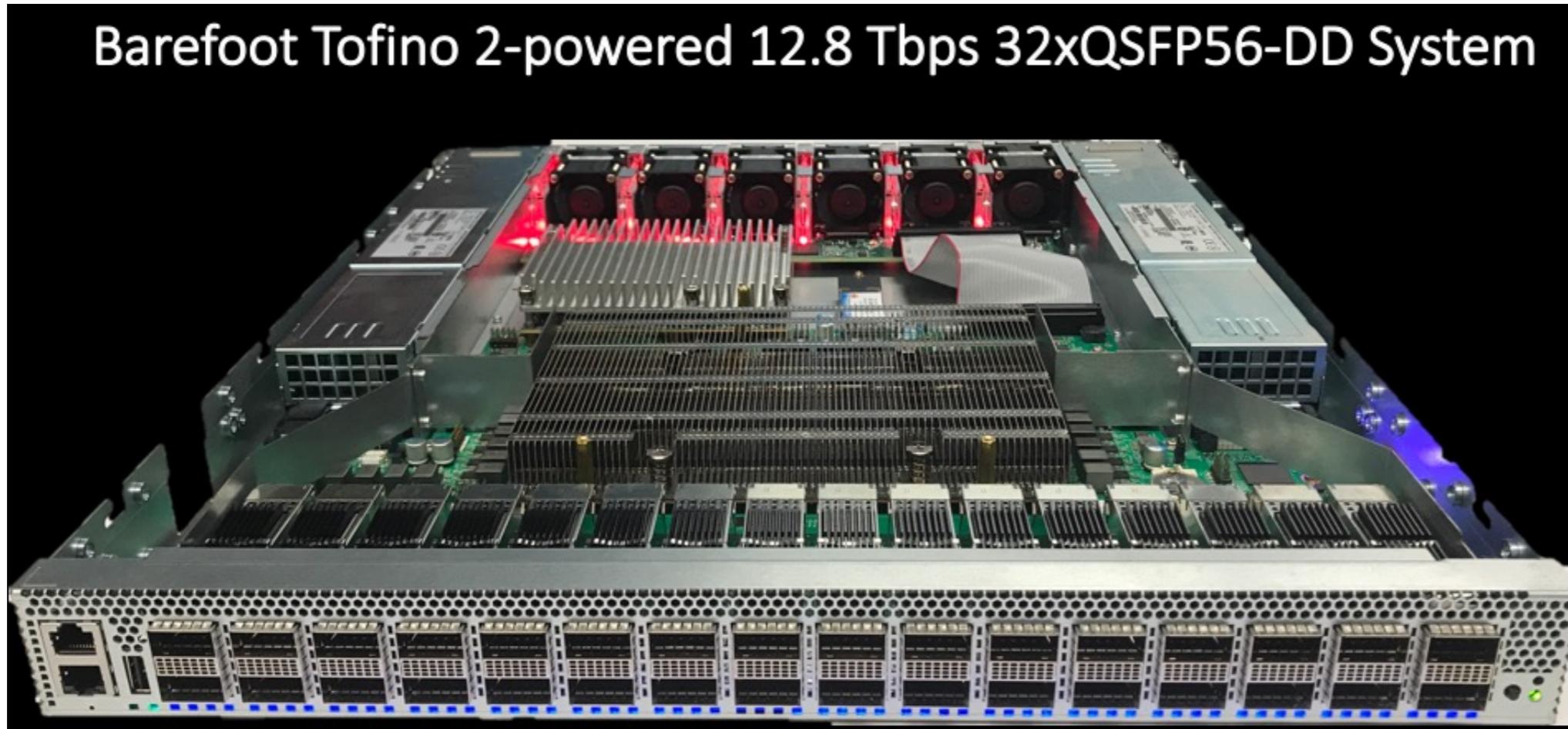
How Tofino Supports Parallel Processing

- Multiple tables mean multiple parallel lookups
- All actions from all active tables are combined
- Each PHV container has its own, independent processor



What's next?

Tofino 2: 12.8 Tbps (7 nm switching ASIC)



<https://www.barefoottnetworks.com/press-releases/barefoot-networks-unveils-tofino-2-the-next-generation-of-the-worlds-first-fully-p4-programmable-network-switch-asics/>

P4 hardware
target

P4-based
applications

What cool things
can we do with it?

A high-level, **non-exhaustive overview** of the research surrounding data plane programmability

A high-level, non-exhaustive overview of the research surrounding data plane programmability

Data plane for
programmability

Performance
Monitoring
Applications offloading

Platforms for Data plane
Correctness programmability
Management

Data plane
programmability

for

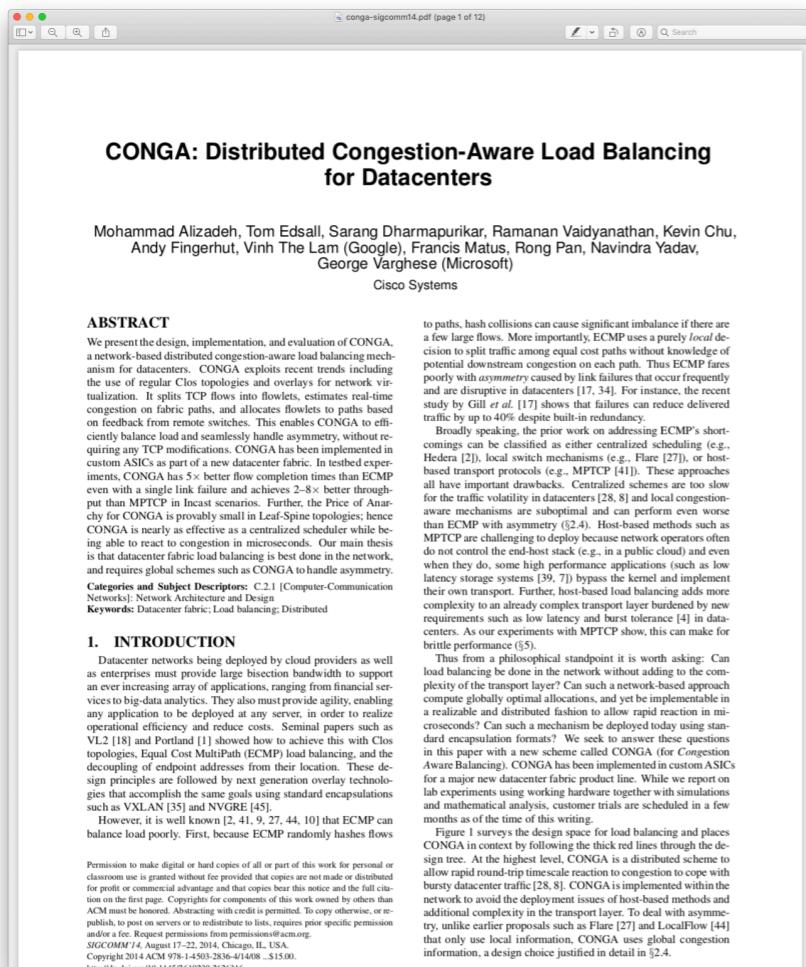
Performance
Monitoring
Applications offloading

Platforms
Correctness
Management

for

Data plane
programmability

A large set of papers on programmable data planes aim at improving performance, esp. load balancing

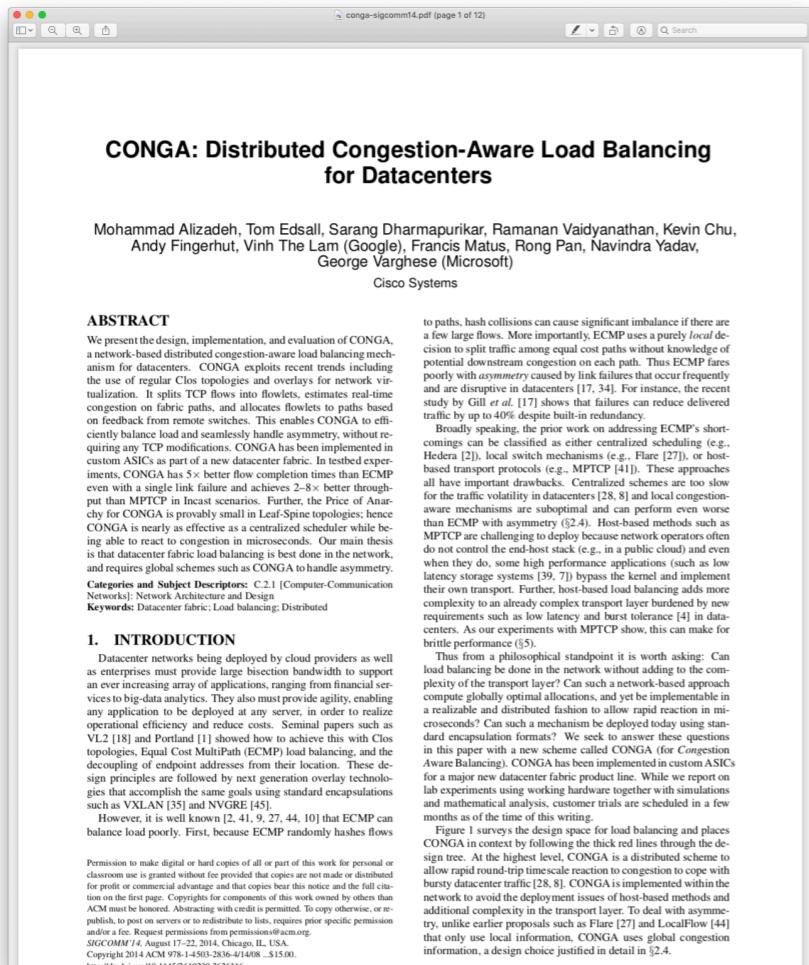


CONGA [SIGCOMM'14]



LetFlow [NSDI'17]

A large set of papers on programmable data planes aim at improving performance, esp. load balancing



CONGA [SIGCOMM'14]

LetFlow [NSDI'17]

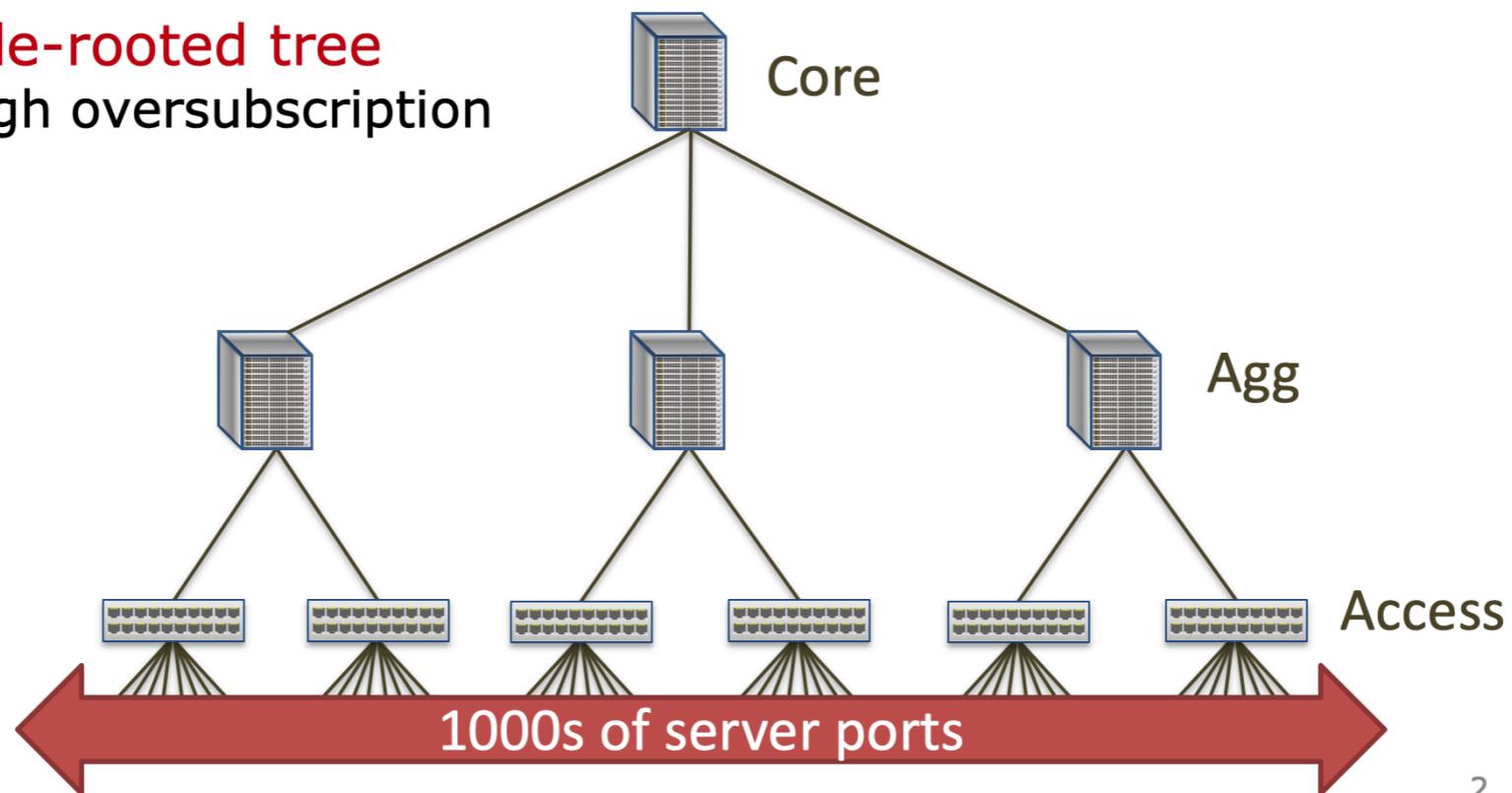
DRILL [SIGCOMM'17]

Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

Single-rooted tree

- High oversubscription



2

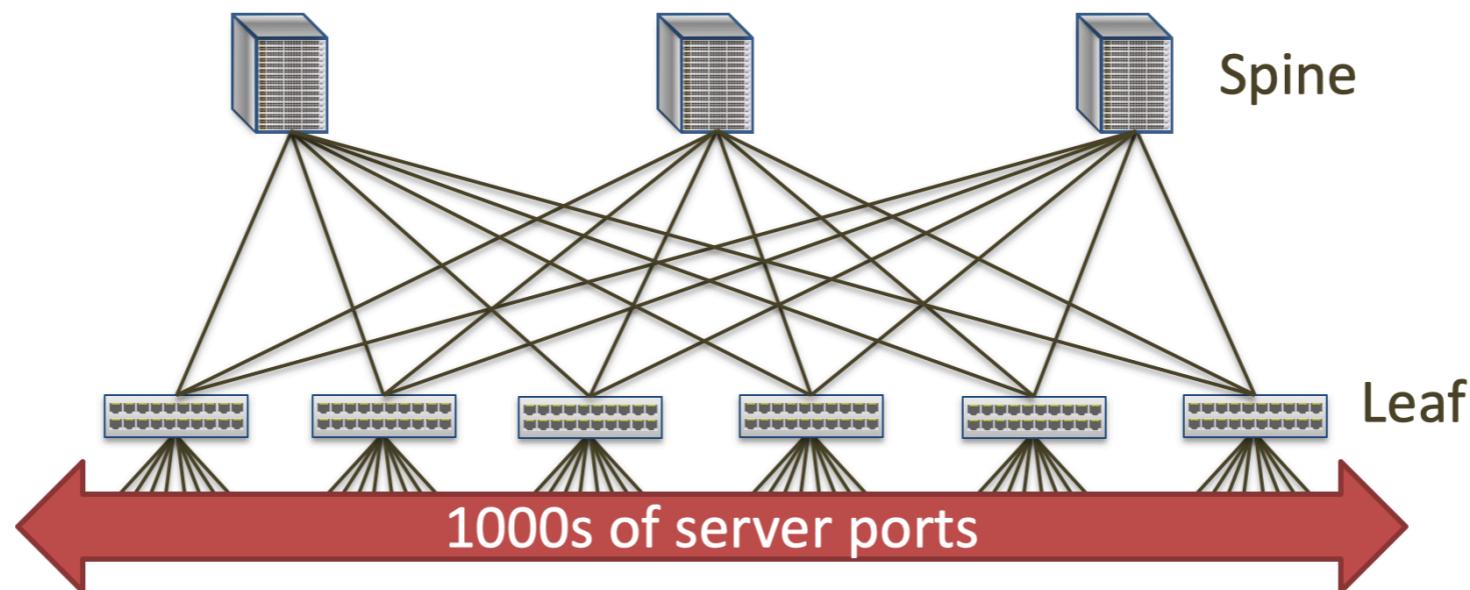
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

Multi-rooted tree [Fat-tree, Leaf-Spine, ...]

- Full bisection bandwidth, achieved via multipathing

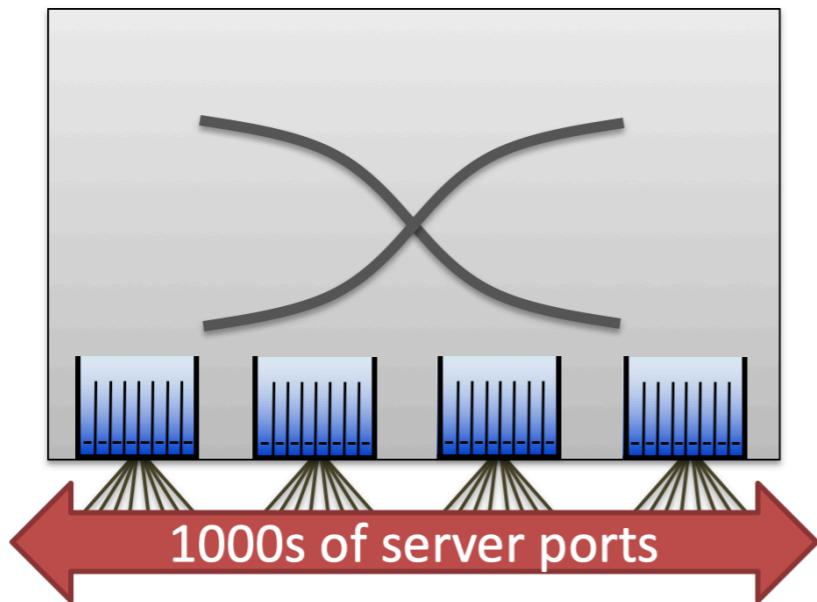


2

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

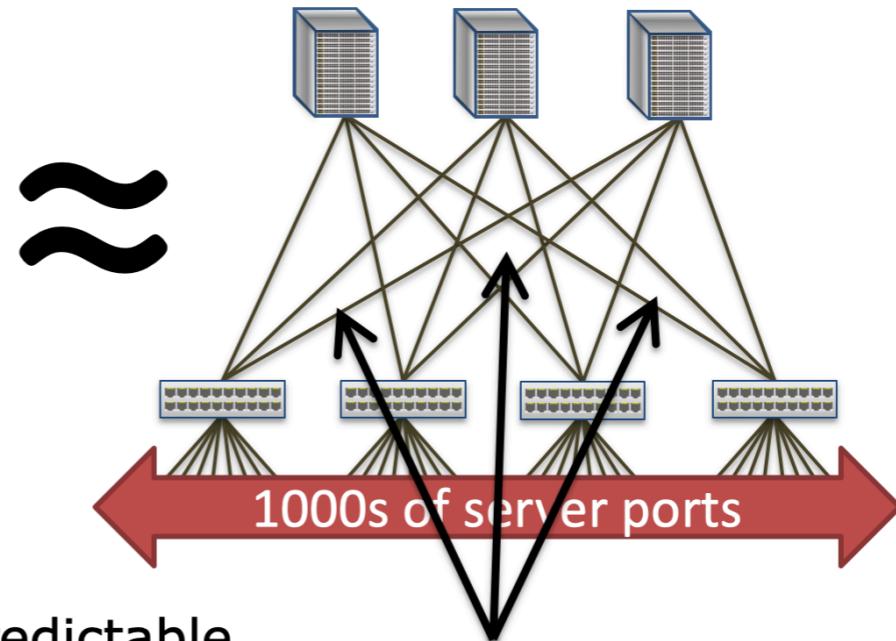
Multi-rooted != Ideal DC Network

Ideal DC network:
Big output-queued switch



- No internal bottlenecks → predictable
- Simplifies BW management
[EyeQ, FairCloud, pFabric, Varys, ...]

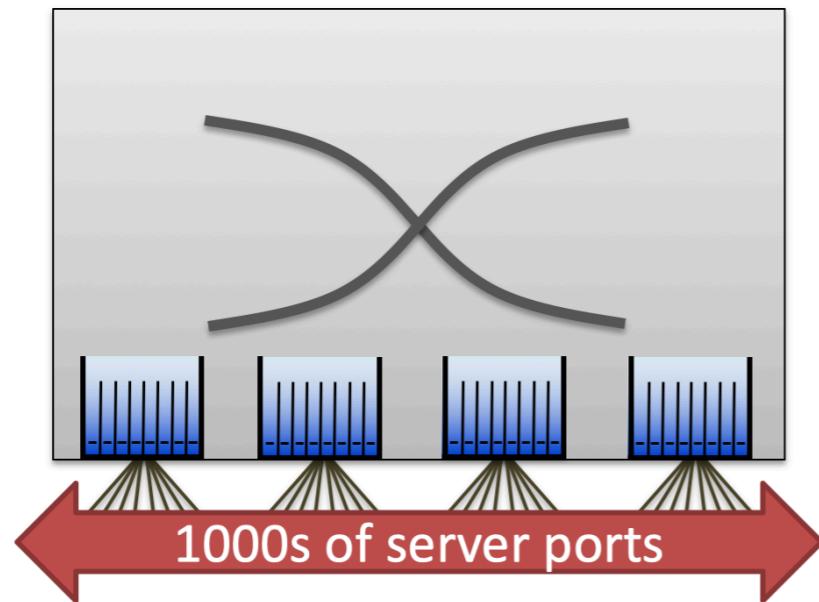
Multi-rooted tree



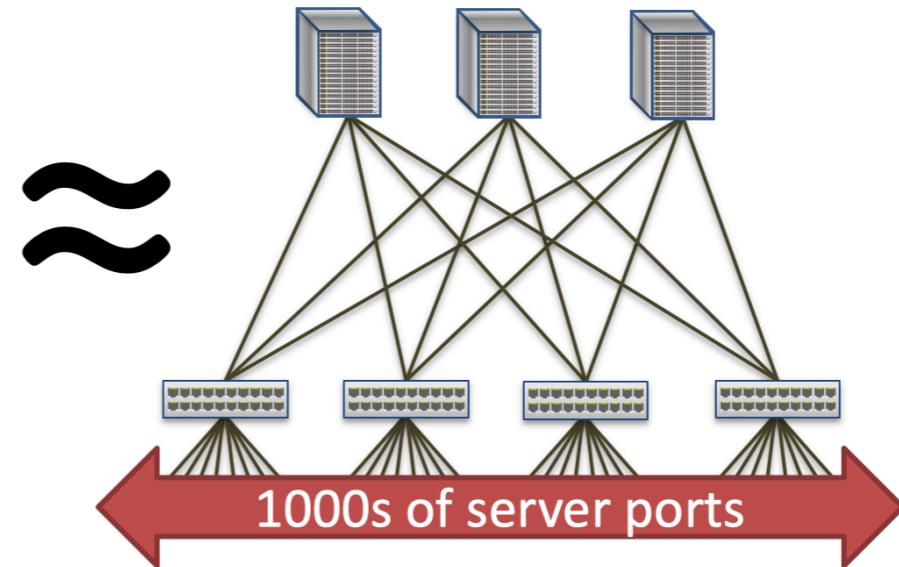
Possible
bottlenecks

Multi-rooted != Ideal DC Network

Ideal DC network:
Big output-queued switch



Multi-rooted tree



Need precise load balancing

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

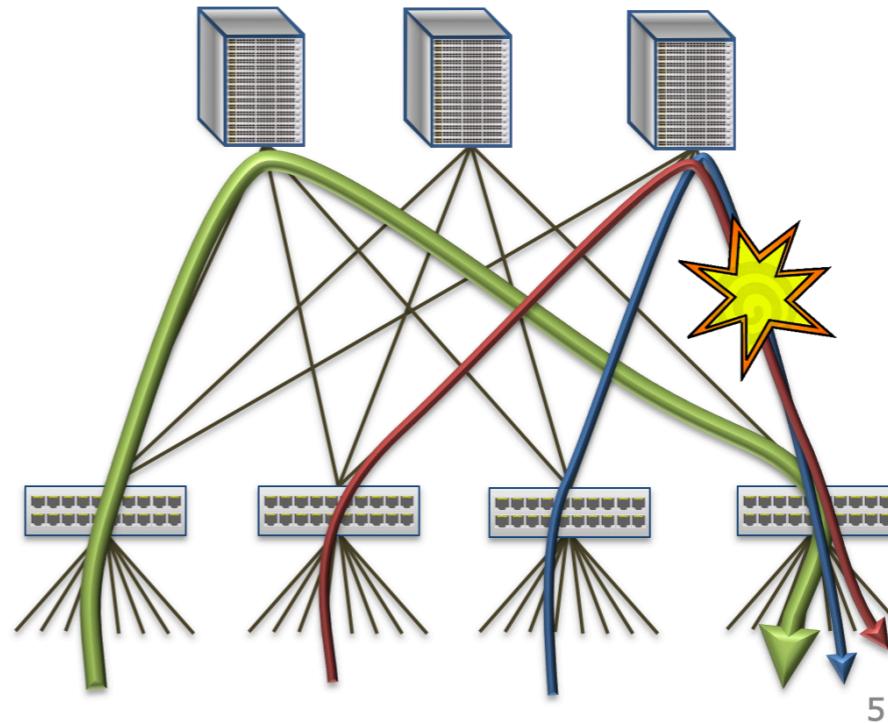
Today: ECMP Load Balancing

Pick among equal-cost paths by a **hash** of 5-tuple

- Approximates Valiant load balancing
- Preserves packet order

Problems:

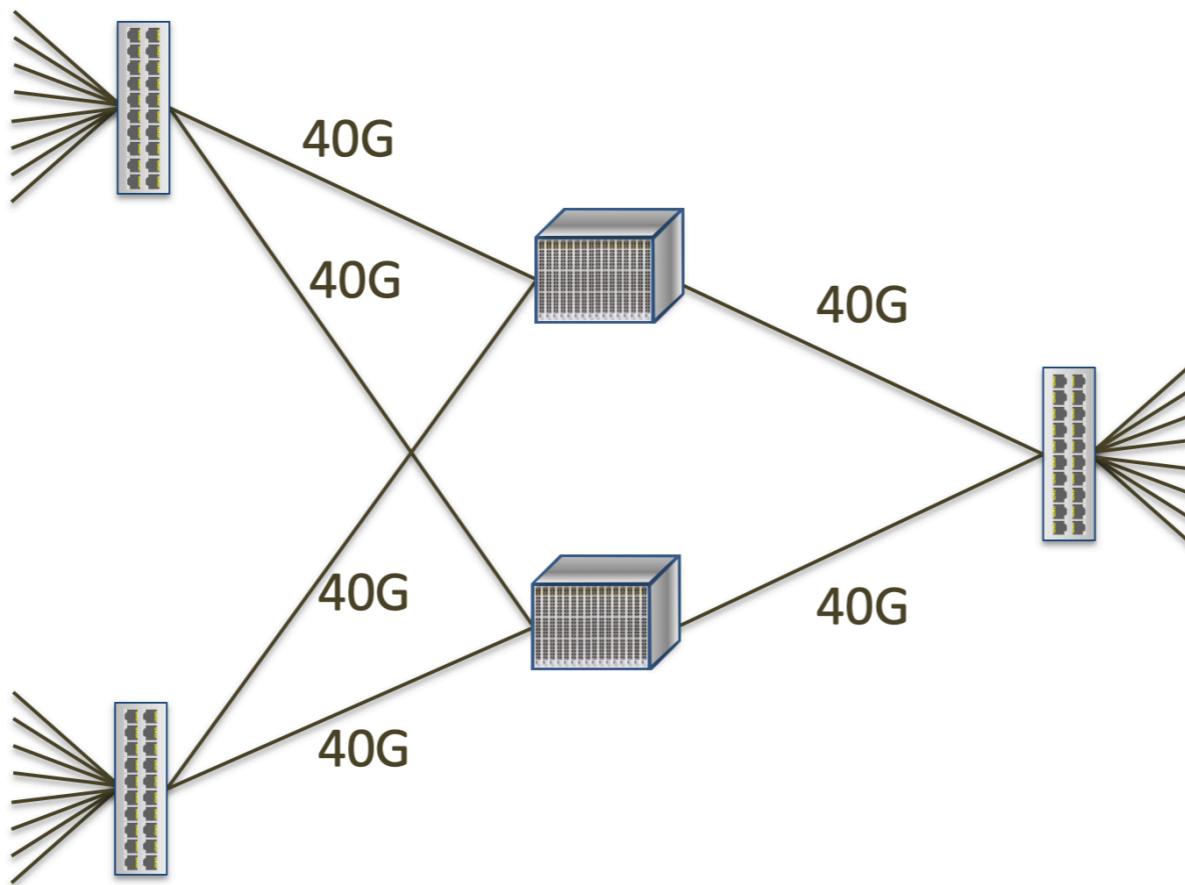
- Hash collisions
(coarse granularity)
- Local & stateless
(v. bad with asymmetry
due to link failures)



Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

Dealing with Asymmetry

Handling asymmetry needs non-local knowledge

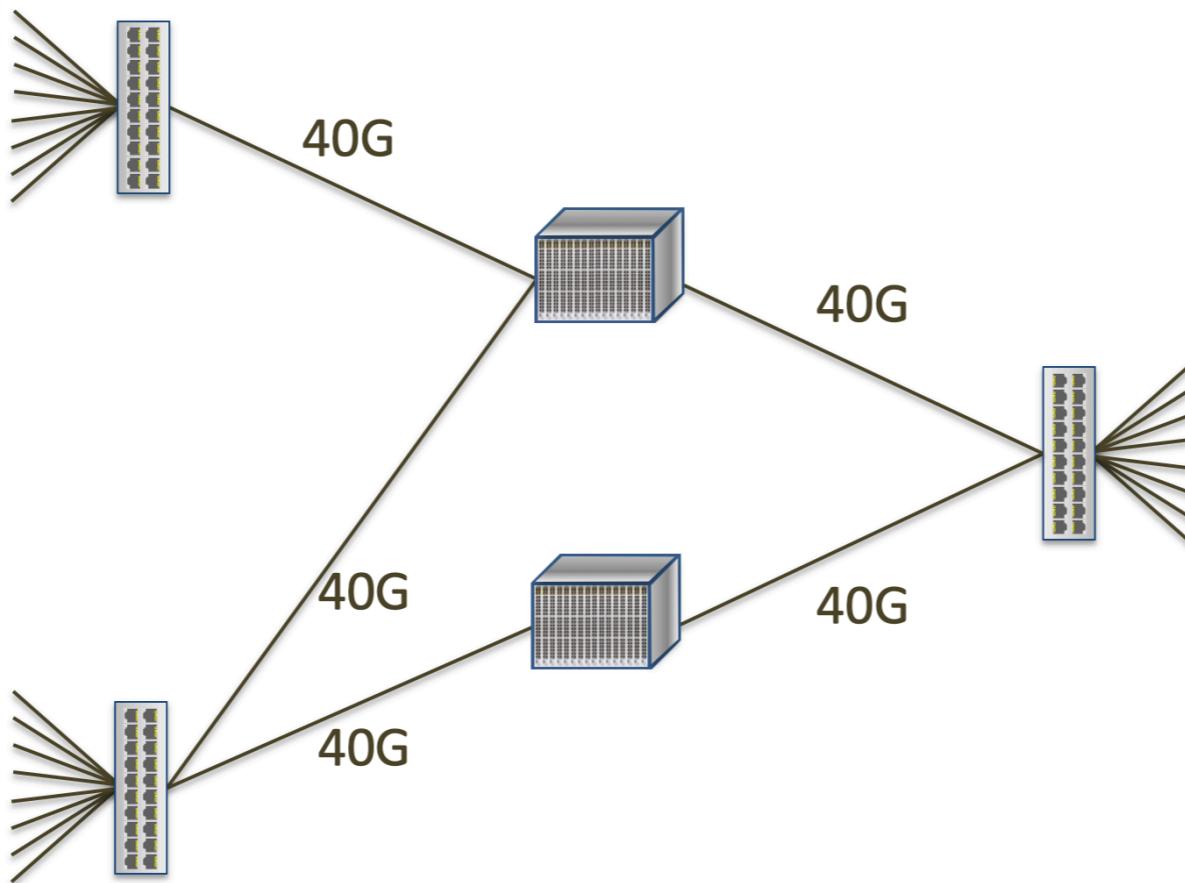


6

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

Dealing with Asymmetry

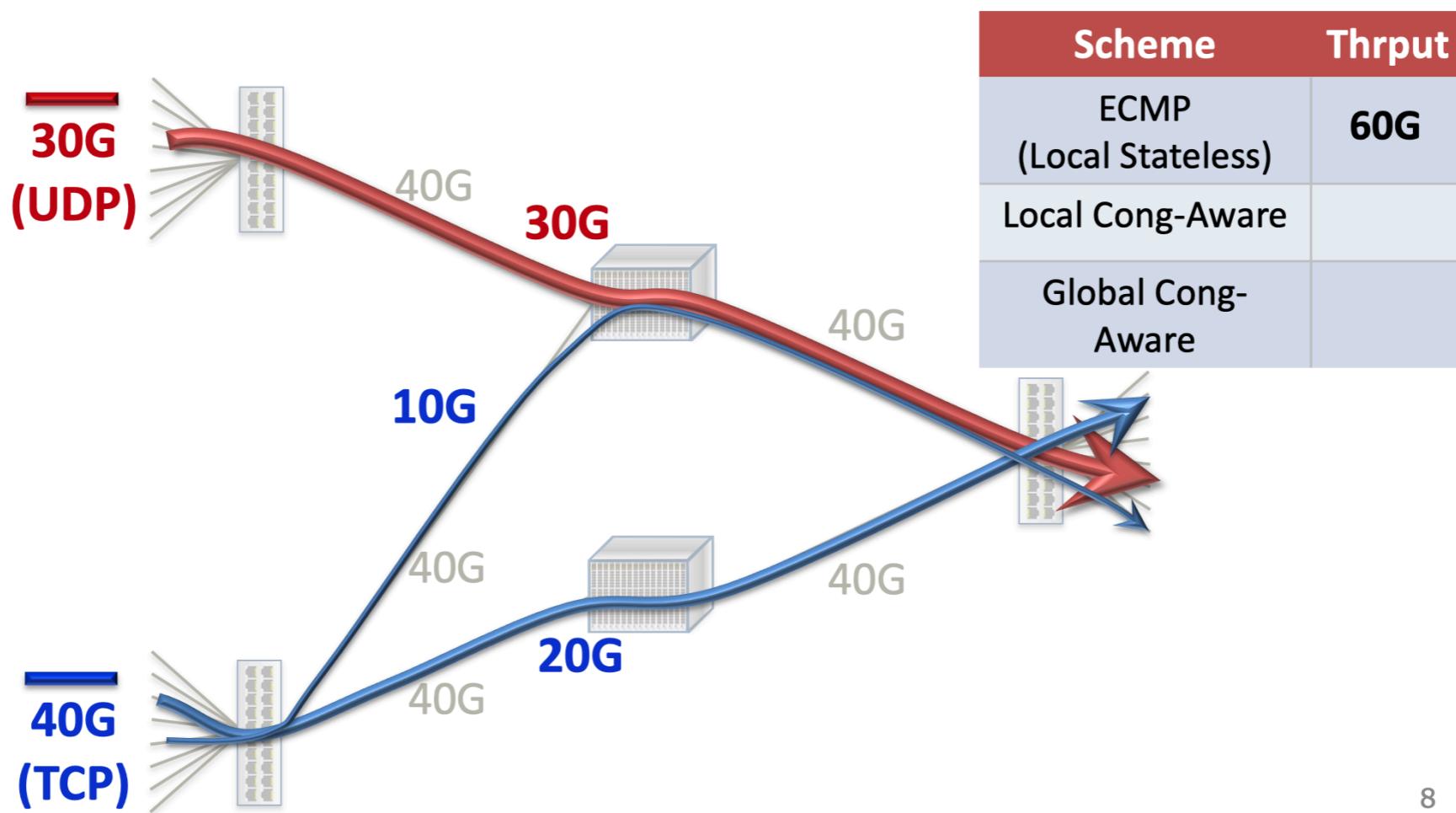
Handling asymmetry needs non-local knowledge



6

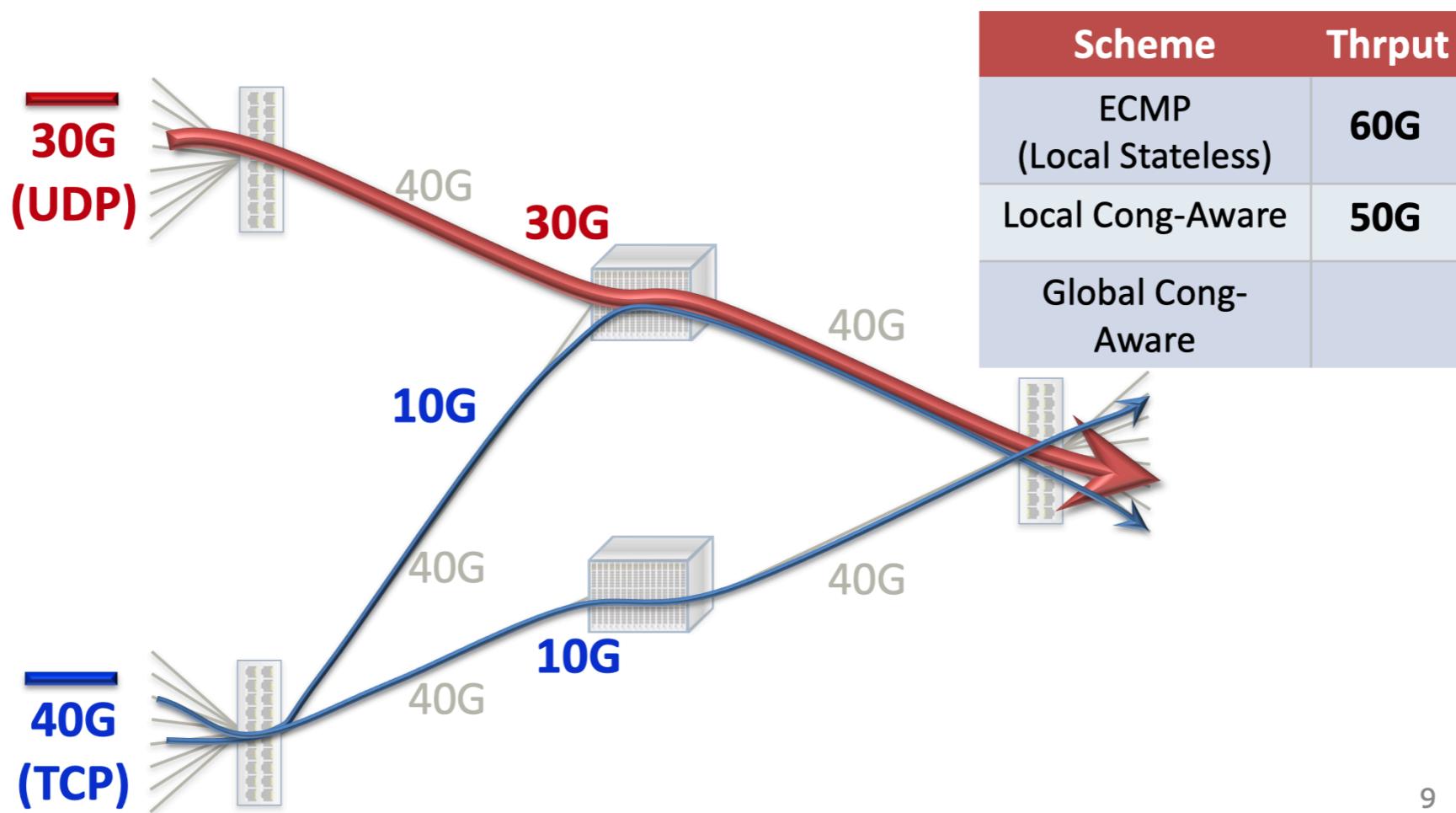
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

Dealing with Asymmetry: ECMP



Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

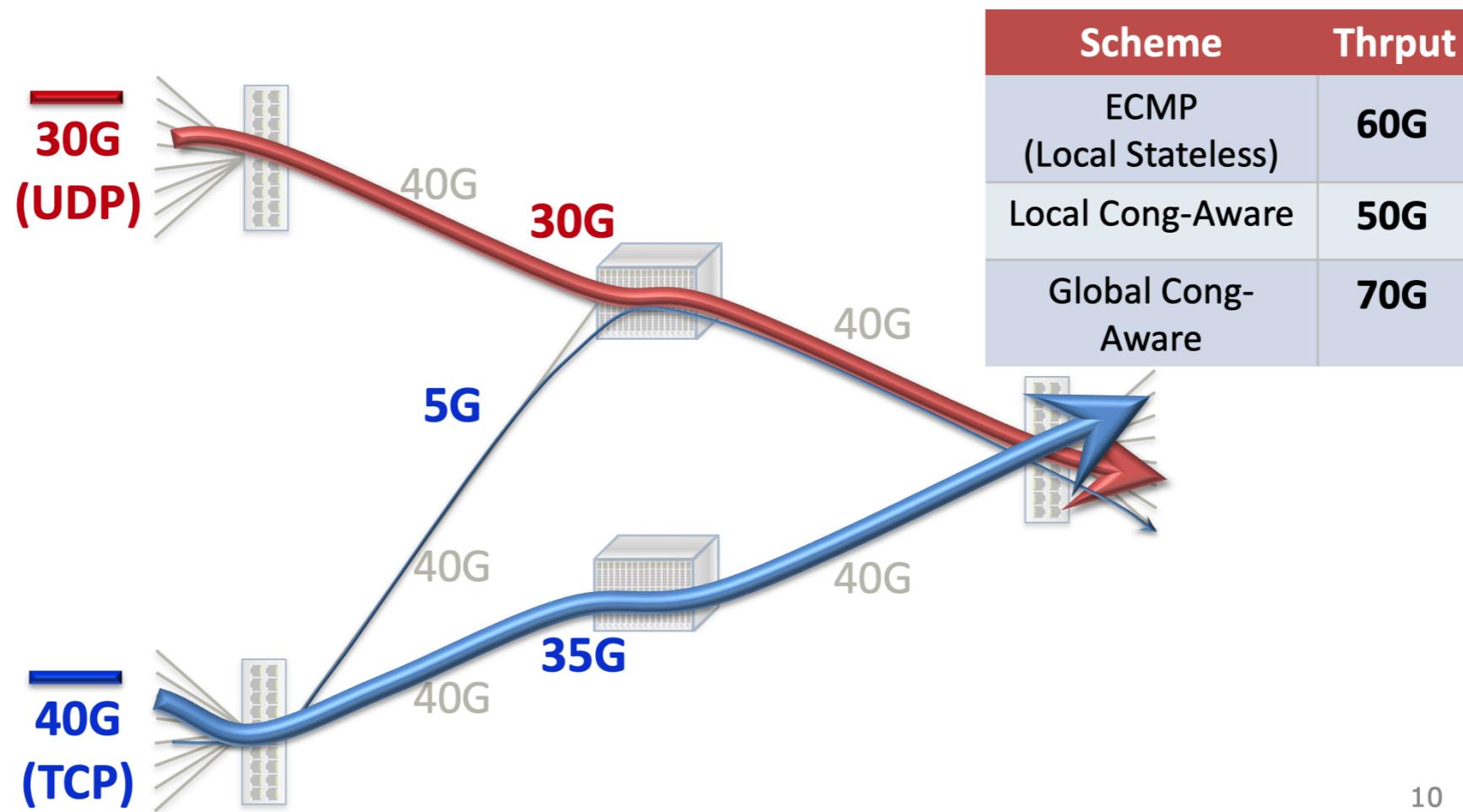
Dealing with Asymmetry: Local Congestion-Aware



9

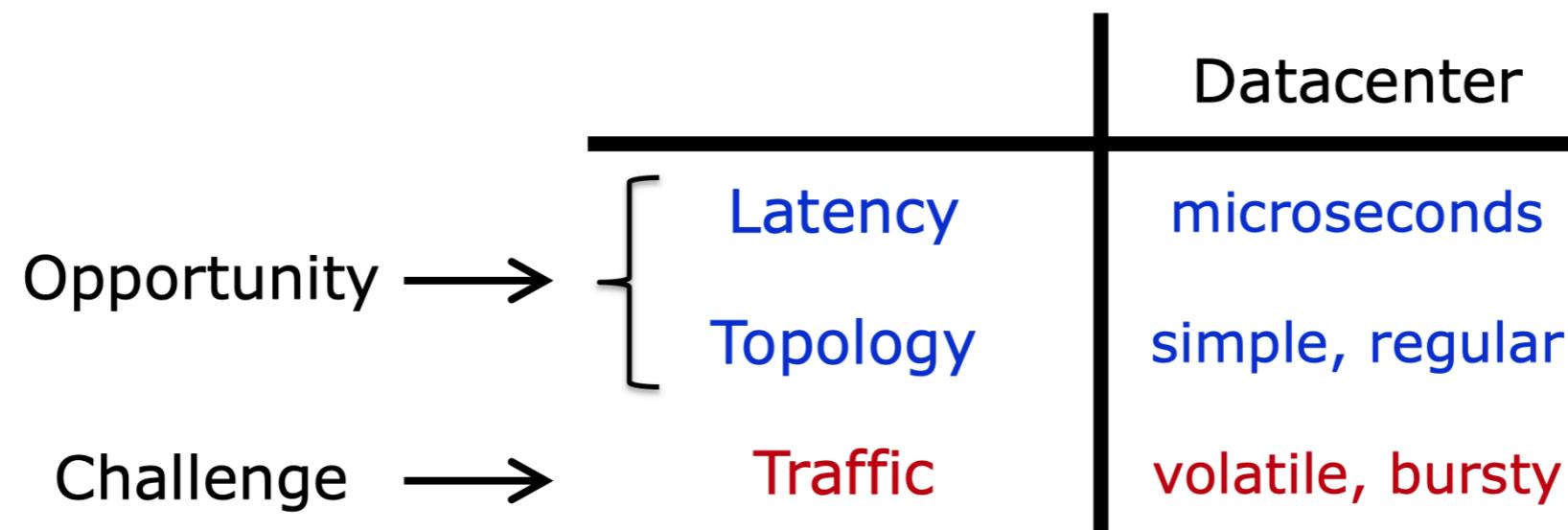
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

Dealing with Asymmetry: Global Congestion-Aware



Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

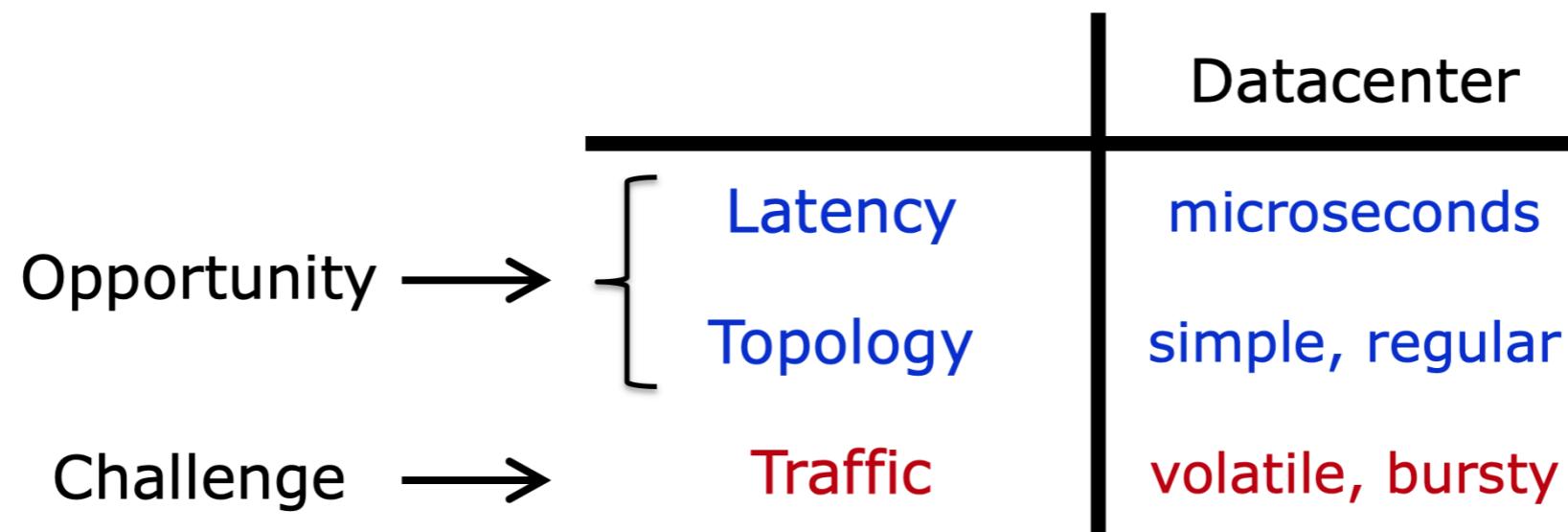
Global Congestion-Awareness (in Datacenters)



11

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

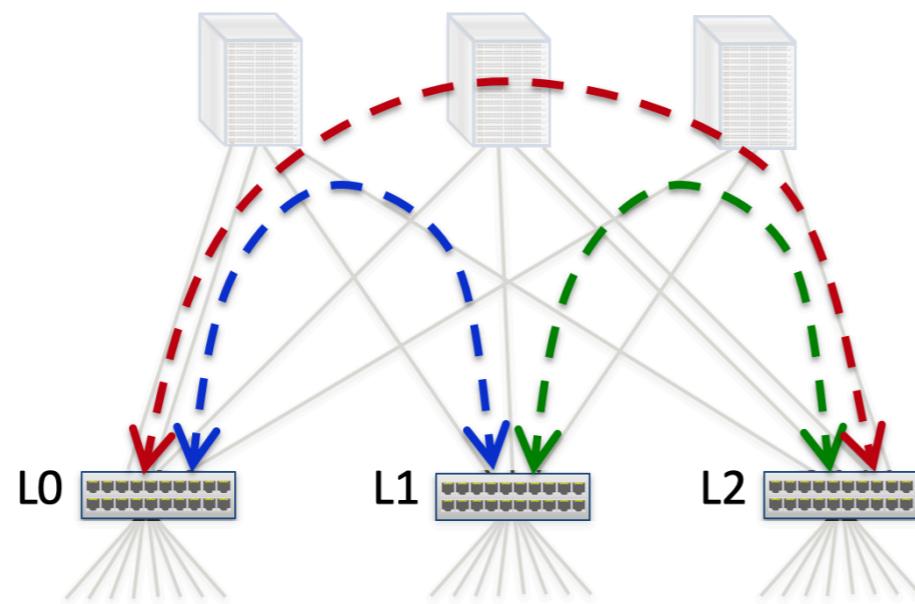
Global Congestion-Awareness (in Datacenters)



Key Insight:
Use *extremely fast, low latency*
distributed control

CONGA in 1 Slide

1. Leaf switches (top-of-rack) track congestion to other leaves on different paths **in near real-time**
1. Use greedy decisions to minimize bottleneck util



Fast feedback loops
between leaf switches,
directly in dataplane

12

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,
Mohammad Alizadeh et al., 2014

A large set of papers on programmable data planes aim at improving performance, esp. load balancing

P4-based data-plane load-balancing
with better scalability than CONGA

HULA [SOSR'16]



DRILL [SIGCOMM'17]

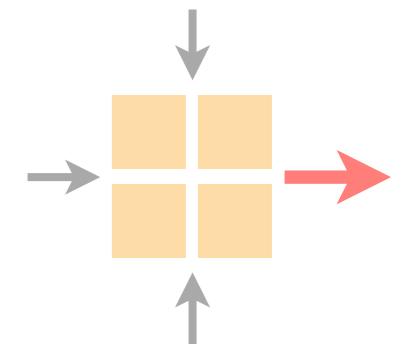
"micro" load balancing,
packet-by-packet,
can deal with micro-bursts

stateless, yet congestion-aware
load-balancing decision

LetFlow [NSDI'17]

Advanced Topics in Communication Networks

Programming Network Data Planes



Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich
Oct 22 2019