PADERBORN UNIVERSITY

Advanced Networked Systems SS24

Programmable Switch Architecture

Prof. Lin Wang, Ph.D.
Computer Networks Group
Paderborn University
https://cs.uni-paderborn.de/cn

RMT and P4

RMT: reconfigurable match tables model (a RISC-inspired pipelined architecture)

P4: a domain-specific language for programming protocol-independent packet processors

P4: Programming Protocol-Independent Packet Processors

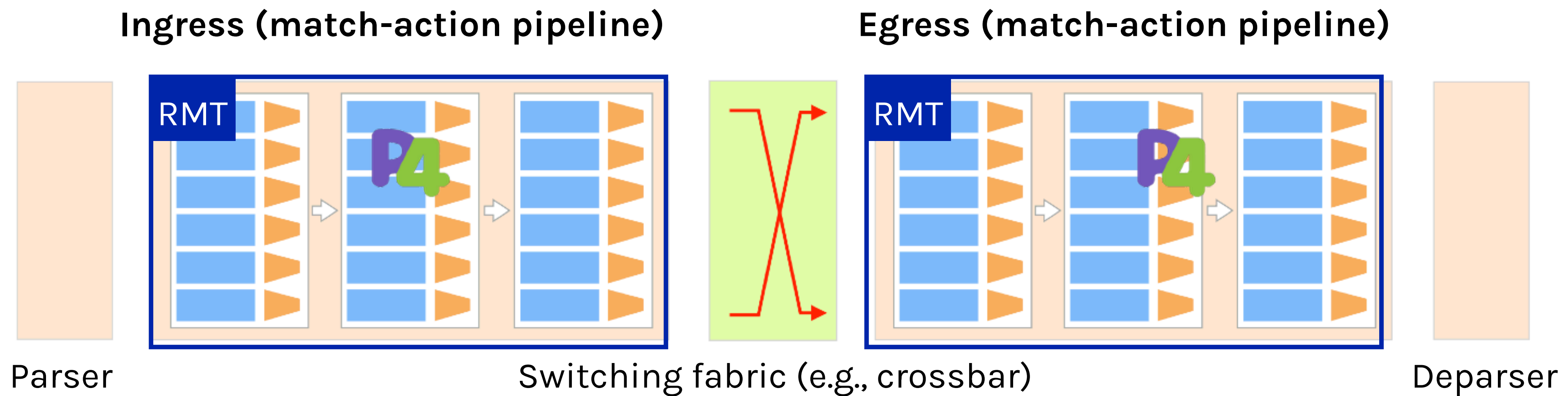
Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

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 paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open inter-



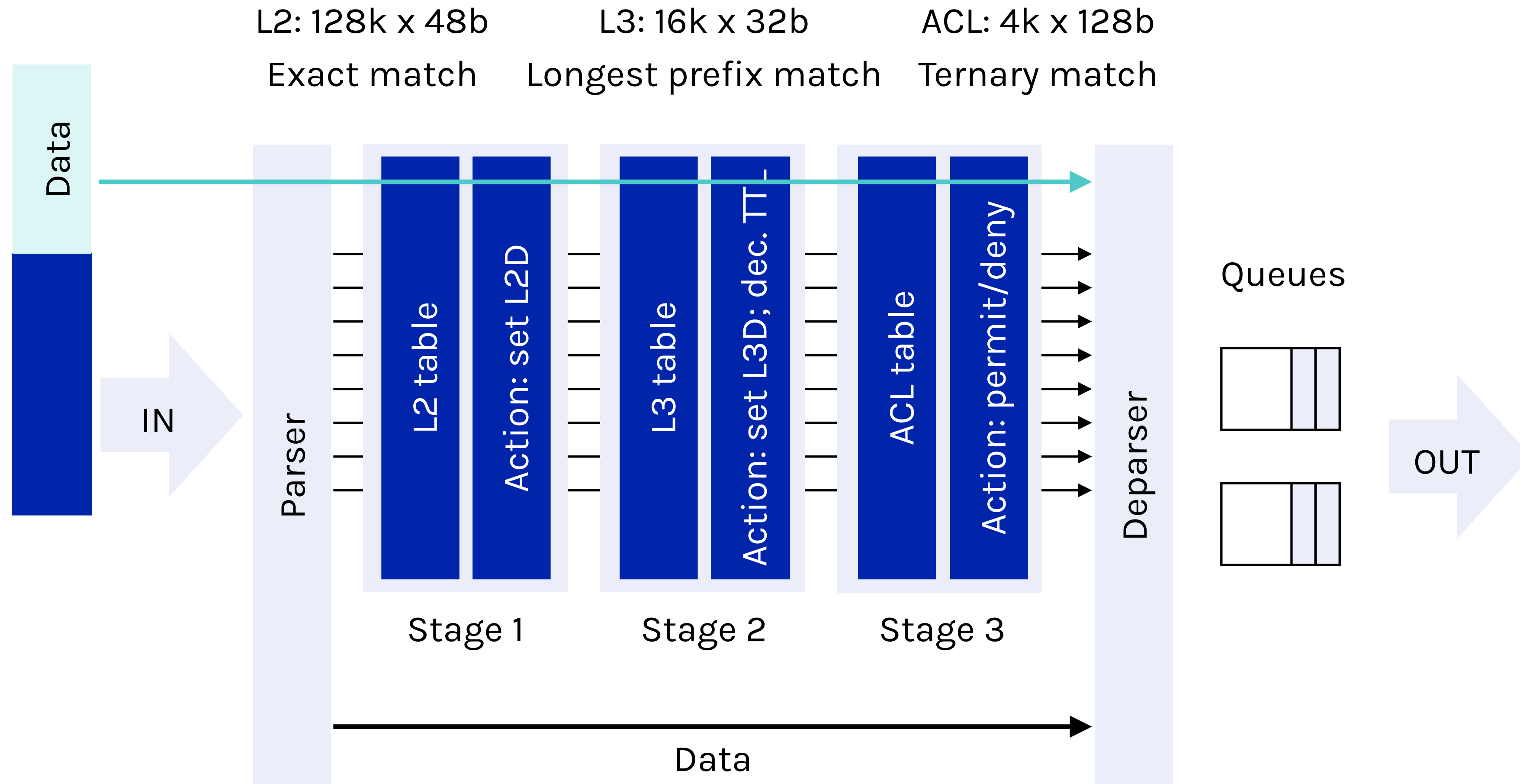
Learning objectives

How to **implement programmable data planes in hardware**?

How to **improve resource efficiency** of programmable data planes?

Implementing programmable data planes in hardware

Fixed function switch architecture



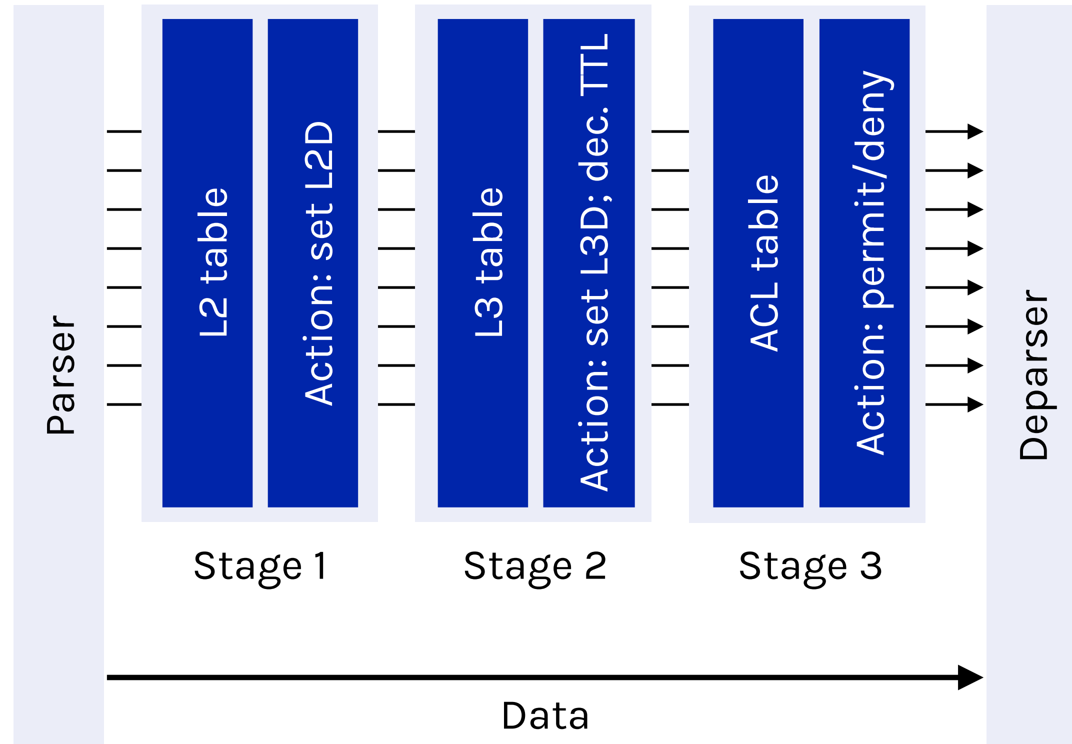
Limited flexibility

Hard to

- Trade one **memory size** for another
- Add a new **table**
- Add a new **header field**
- Add a different **action**

SDN pushes for flexibility

- Programmatic control to control plane
- **Data plane flexibility** demanded



SDN flexibility demands

Multiple stages of match-action

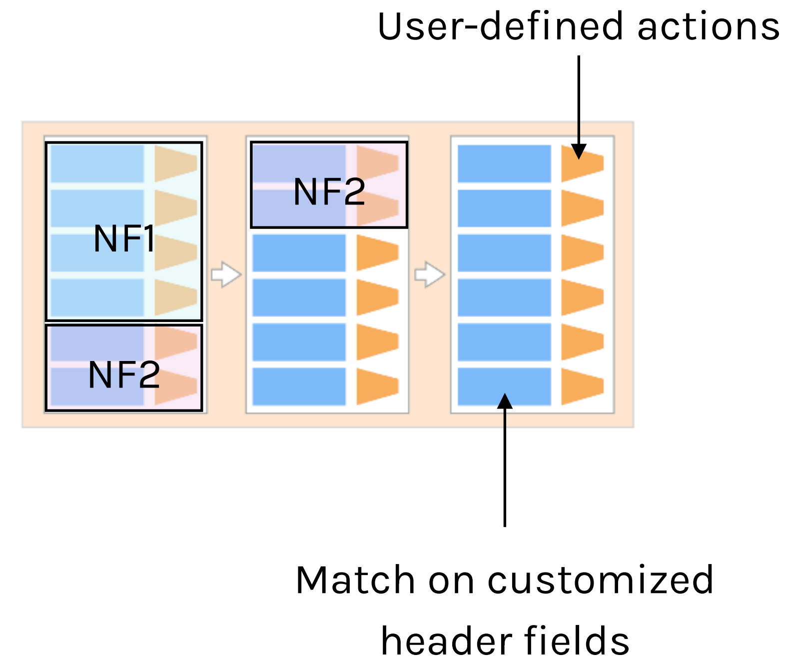
- Flexible allocation of memory to different functionalities

Flexible actions

- User-defined actions instead of hard-baked ones

Flexible header fields

- Allowing the customizable header fields instead of being bounded by the known protocols



Different ways to achieve flexibility

NF1 NF2 NF3

Software: 100x too slow, expensive

NF1 NF2 NF3

NPUs: 10x too slow, expensive

NF1 NF2 NF3

FPGAs: 10x too slow, expensive

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

Designing a flexible switch chip is hard

Bad news

- Big chip (memory, compute, I/O)
- High frequency (line rate of 100 Gbps)
- Wiring intensive (match-action logic)
- Many crossbars (header selectors)
- Lots of TCAM (fast matching)
- Interaction between physical design and architecture

Good news

- No need to read 9k+ IETF RFCs

RFC Index

Num Information

9327 **Control Messages Protocol for Use with Network Time Protocol Version 4** B. Haberman [November 2022] (HTML, TEXT, PDF, XML) (Status: HISTORIC) (Stream: IETF, Area: int, WG: ntp) (DOI: 10.17487/RFC9327)

9326 **In Situ Operations, Administration, and Maintenance (IOAM) Direct Exporting** H. Song, B. Gafni, F. Brockners, S. Bhandari, T. Mizrahi [November 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, Area: tsv, WG: ippm) (DOI: 10.17487/RFC9326)

9323 **A Profile for RPKI Signed Checklists (RSCs)** J. Snijders, T. Harrison, B. Maddison [November 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, Area: ops, WG: sidrops) (DOI: 10.17487/RFC9323)

9322 **In Situ Operations, Administration, and Maintenance (IOAM) Loopback and Active Flags** T. Mizrahi, F. Brockners, S. Bhandari, B. Gafni, M. Spiegel [November 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, Area: tsv, WG: ippm) (DOI: 10.17487/RFC9322)

9321 **Signature Validation Token** S. Santesson, R. Housley [October 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: INDEPENDENT) (DOI: 10.17487/RFC9321)

9319 **The Use of maxLength in the Resource Public Key Infrastructure (RPKI)** Y. Gilad, S. Goldberg, K. Sriram, J. Snijders, B. Maddison [October 2022] (HTML, TEXT, PDF, XML) (Also BCP0185) (Status: BEST CURRENT PRACTICE) (Stream: IETF, Area: ops, WG: sidrops) (DOI: 10.17487/RFC9319)

9318 **IAB Workshop Report: Measuring Network Quality for End-Users** W. Hardaker, O. Shapira [October 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: IAB) (DOI: 10.17487/RFC9318)

9317 **Operational Considerations for Streaming Media** J. Holland, A. Begen, S. Dawkins [October 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: IETF, Area: ops, WG: mops) (DOI: 10.17487/RFC9317)

9316 **Intent Classification** C. Li, O. Havel, A. Olariu, P. Martinez-Julia, J. Nobre, D. Lopez [October 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: …

9315 **Intent-Based Networking - C…** …2] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (…

9314 **YANG Data Model for Bidire…** …i, G. Mirsky [September 2022] (HTML, TEXT, PDF, XML) (U… (DOI: 10.17487/RFC9314)

9313 **Pros and Cons of IPv6 Transit…** …rd, R. Patterson, I. Farrer [October 2022] (HTML, TEXT, PDF, X… RFC9313)

9312 **Manageability of the QUIC T…** …XML) (Status: INFORMATIONAL) (Stream: IETF, Area: tsv, WG: quic) (DOI: 10.17487/RFC9312)

Only 9327 of them as of November 27, 2022!

9311 **Running an IETF Hackathon** C. Eckel [September 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: IETF, Area: gen, WG: shmoo) (DOI: 10.17487/RFC9311)

9309 **Robots Exclusion Protocol** M. Koster, G. Illyes, H. Zeller, L. Sassman [September 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, WG: NON WORKING GROUP) (DOI: 10.17487/RFC9309)

9308 **Applicability of the QUIC Transport Protocol** M. Kühlewind, B. Trammell [September 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: IETF, Area: tsv, WG: quic) (DOI: 10.17487/RFC9308)

9307 **Report from the IAB Workshop on Analyzing IETF Data (AID) 2021** N. ten Oever, C. Cath, M. Kühlewind, C. S. Perkins [September 2022] (HTML, TEXT, PDF, XML) (Status: INFORMATIONAL) (Stream: IAB) (DOI: 10.17487/RFC9307)

9306 **Vendor-Specific LISP Canonical Address Format (LCAF)** A. Rodriguez-Natal, V. Ermagan, A. Smirnov, V. Ashtaputre, D. Farinacci [October 2022] (HTML, TEXT, PDF, XML) (Updates RFC8060) (Status: EXPERIMENTAL) (Stream: IETF, Area: rtg, WG: lisp) (DOI: 10.17487/RFC9306)

9305 **Locator/ID Separation Protocol (LISP) Generic Protocol Extension** F. Maino, J. Lemon, P. Agarwal, D. Lewis, M. Smith [October 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: lisp) (DOI: 10.17487/RFC9305)

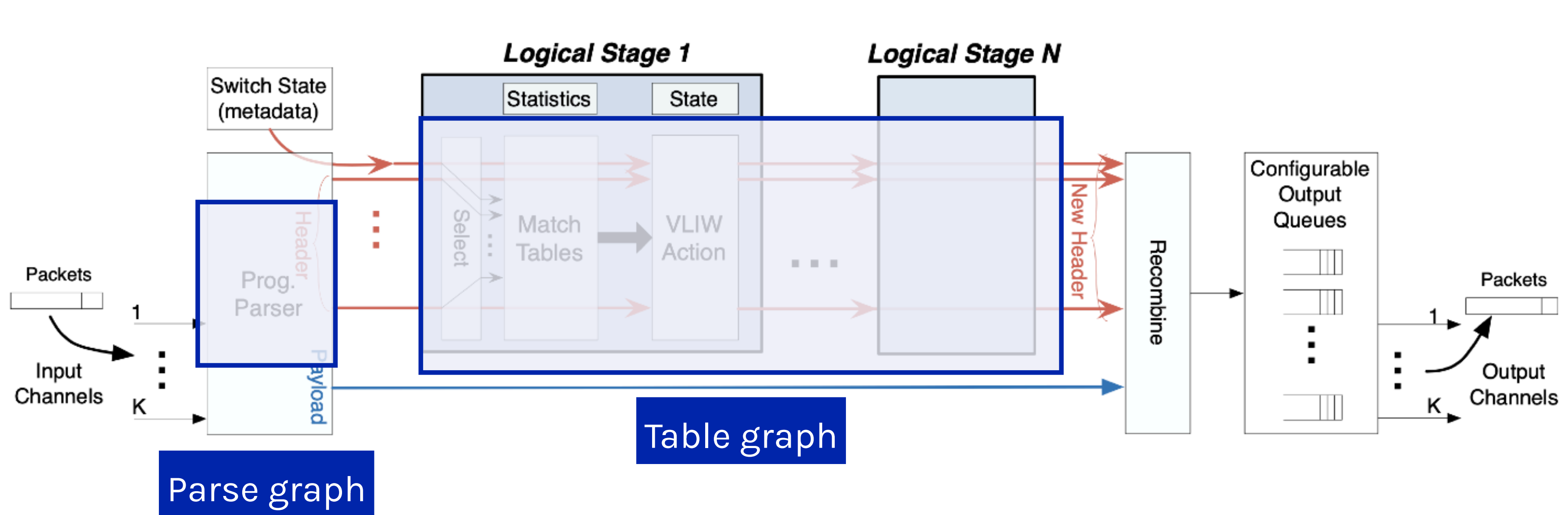
9304 **Locator/ID Separation Protocol (LISP): Shared Extension Message and IANA Registry for Packet Type Allocations** M. Boucadair, C. Jacquenet [October 2022] (HTML, TEXT, PDF, XML) (Obsoletes RFC8113) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: lisp) (DOI: 10.17487/RFC9304)

9303 **Locator/ID Separation Protocol Security (LISP-SEC)** F. Maino, V. Ermagan, A. Cabellos, D. Saucez [October 2022] (HTML, TEXT, PDF, XML) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: lisp) (DOI: 10.17487/RFC9303)

9302 **Locator/ID Separation Protocol (LISP) Map-Versioning** L. Iannone, D. Saucez, O. Bonaventure [October 2022] (HTML, TEXT, PDF, XML) (Obsoletes RFC6834) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: lisp) (DOI: 10.17487/RFC9302)

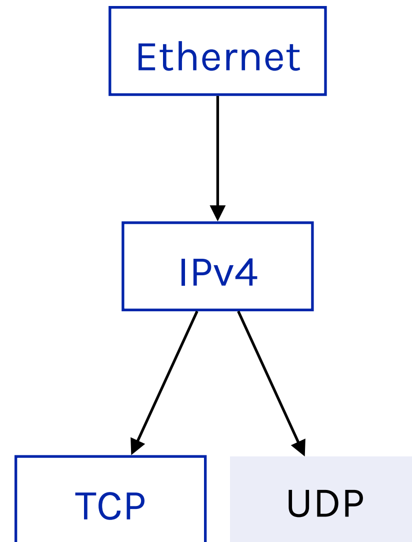
https://www.rfc-editor.org/rfc-index-100d.html

Reconfigurable match table (RMT) abstract model



Parse graph: arbitrary fields

Packet:



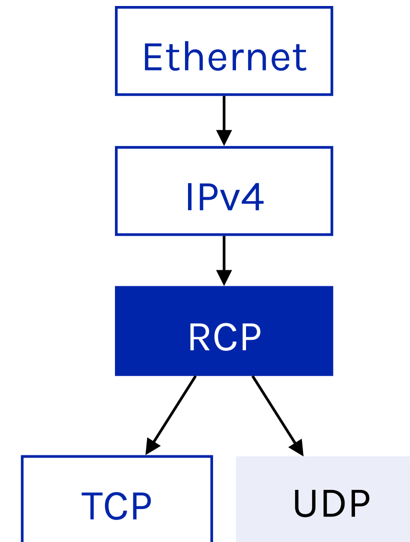
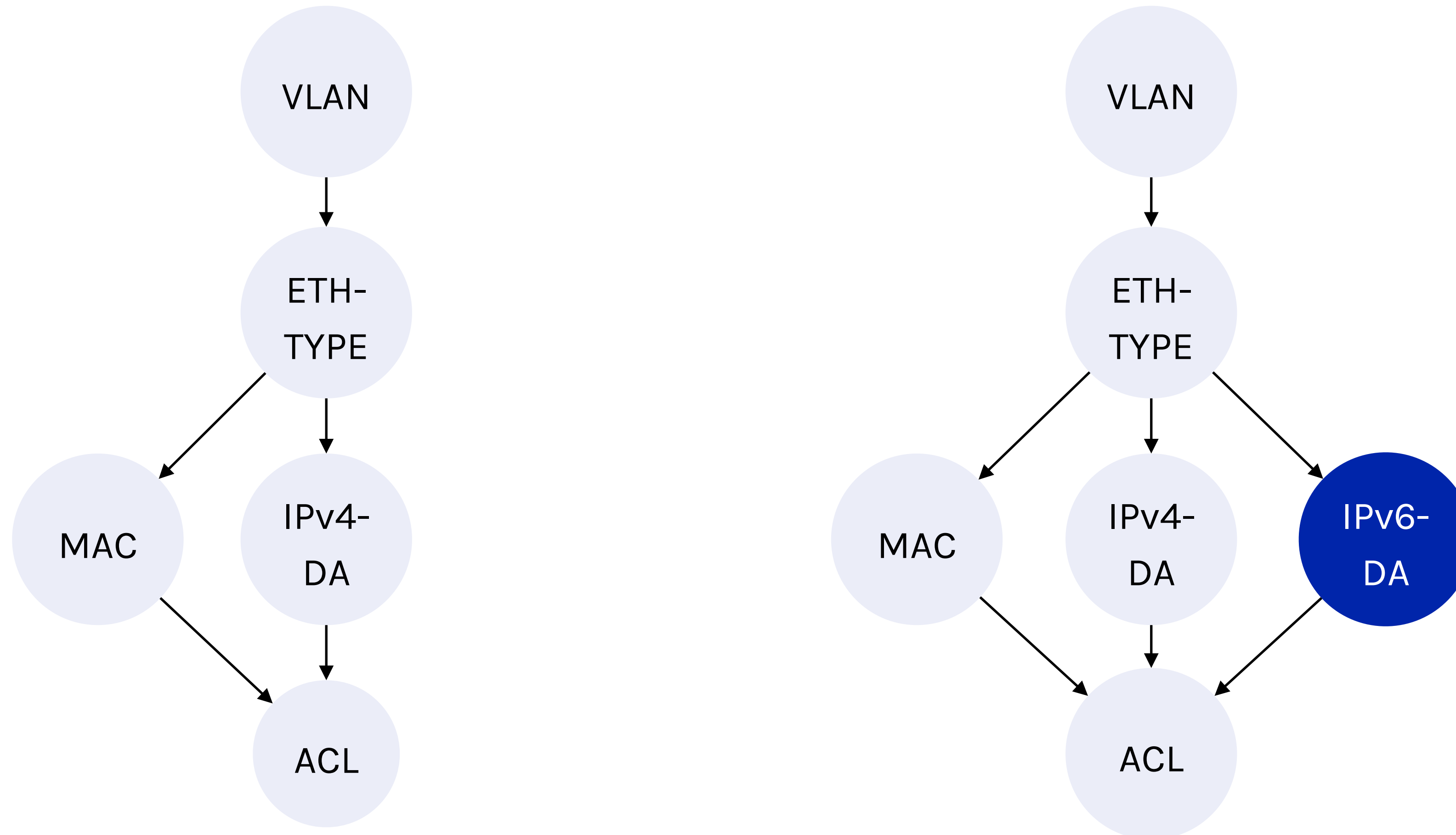
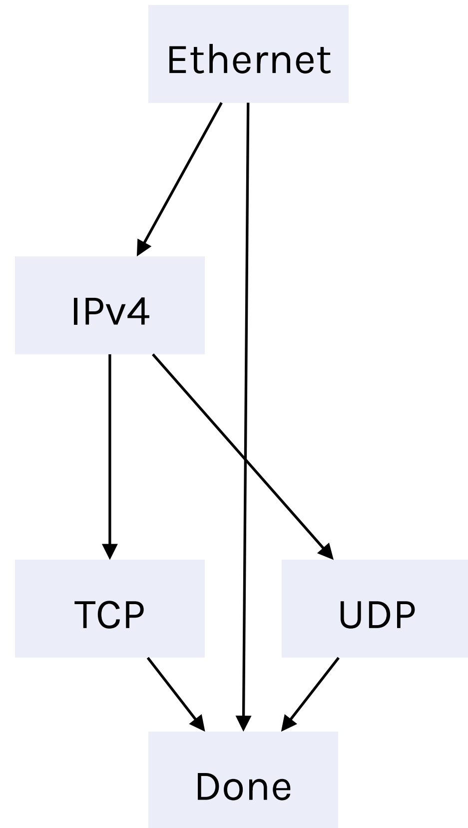


Table graph: reconfigurable match tables



Changes to parse graph and table graph



Parse graph

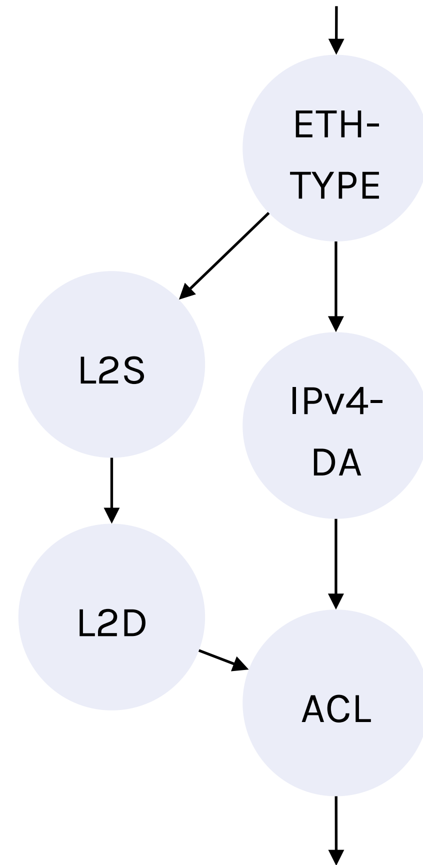
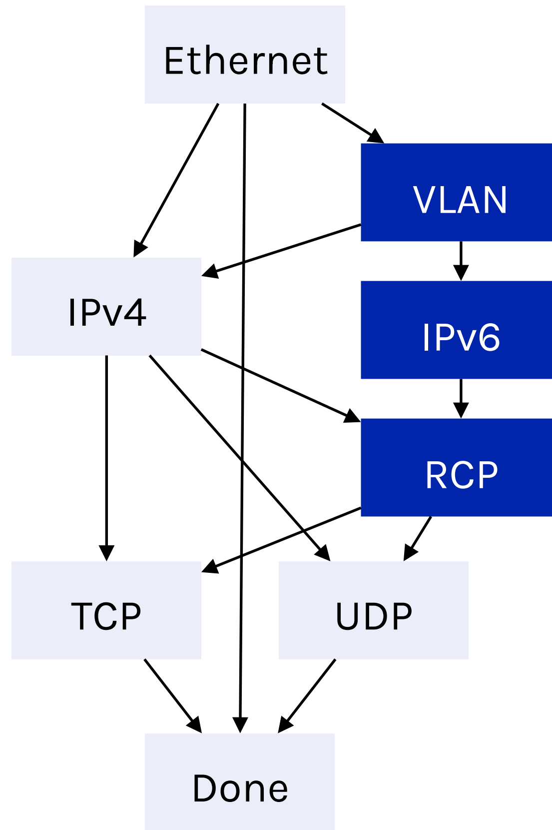


Table graph

Changes to parse graph and table graph



Parse graph

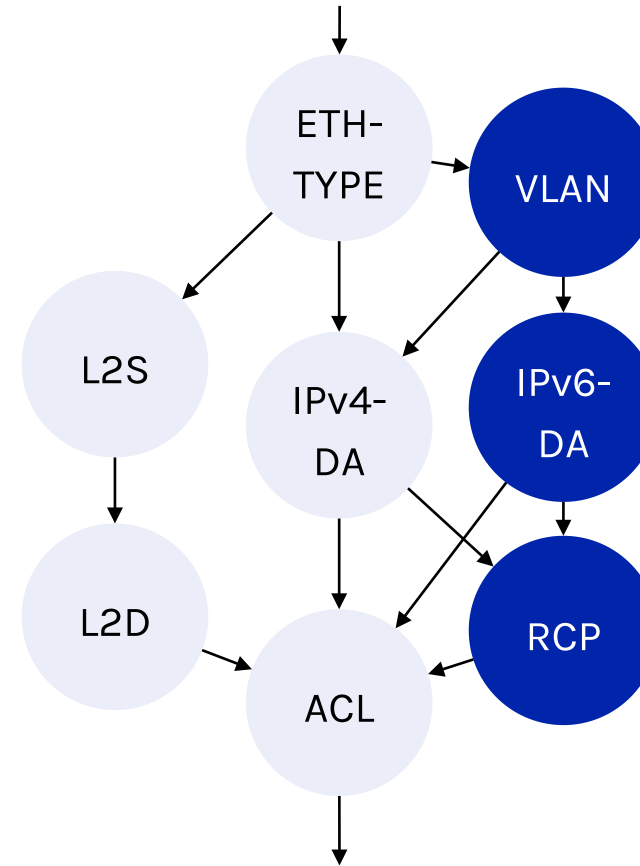
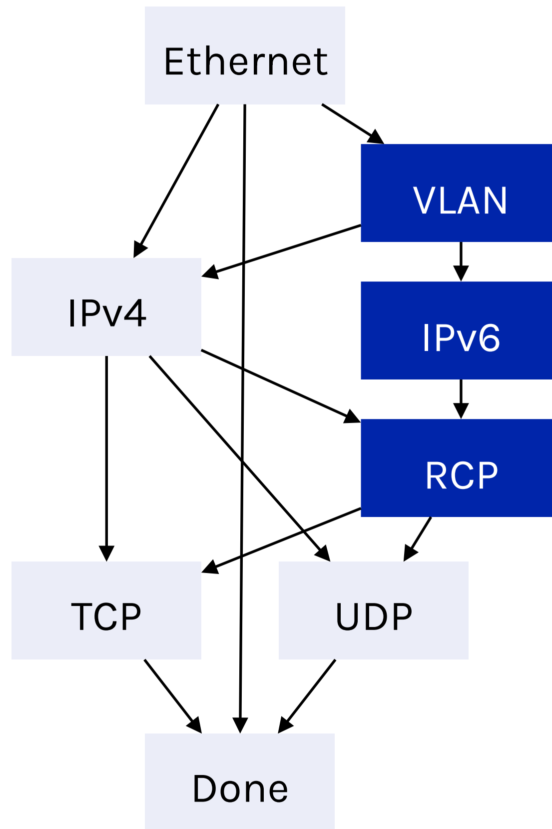


Table graph

Changes to parse graph and table graph



Parse graph

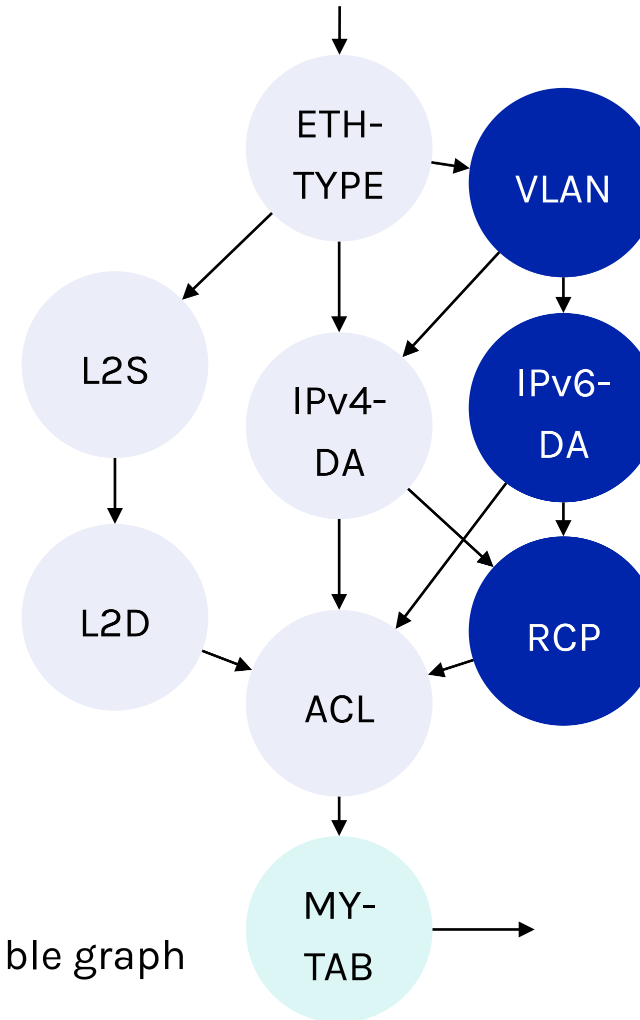
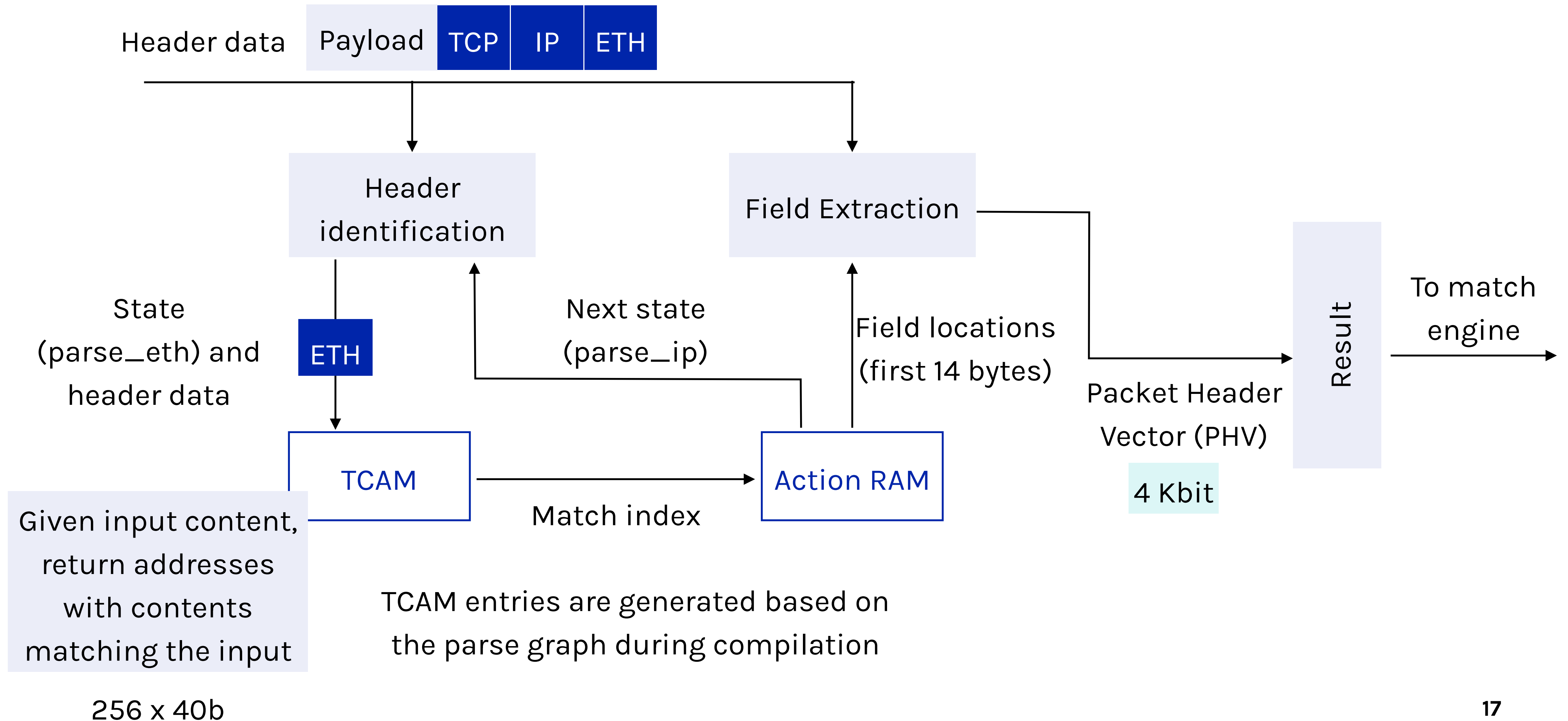


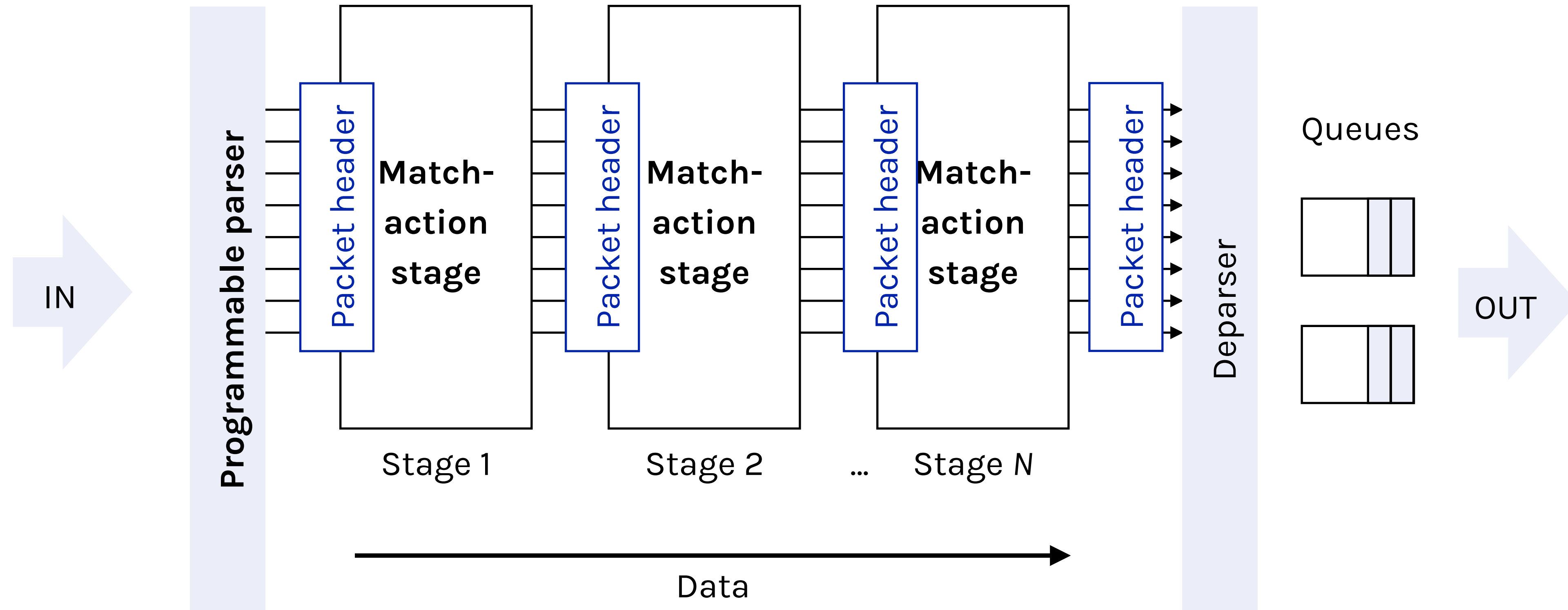
Table graph

How to turn the parse graph and table graph into a switch?

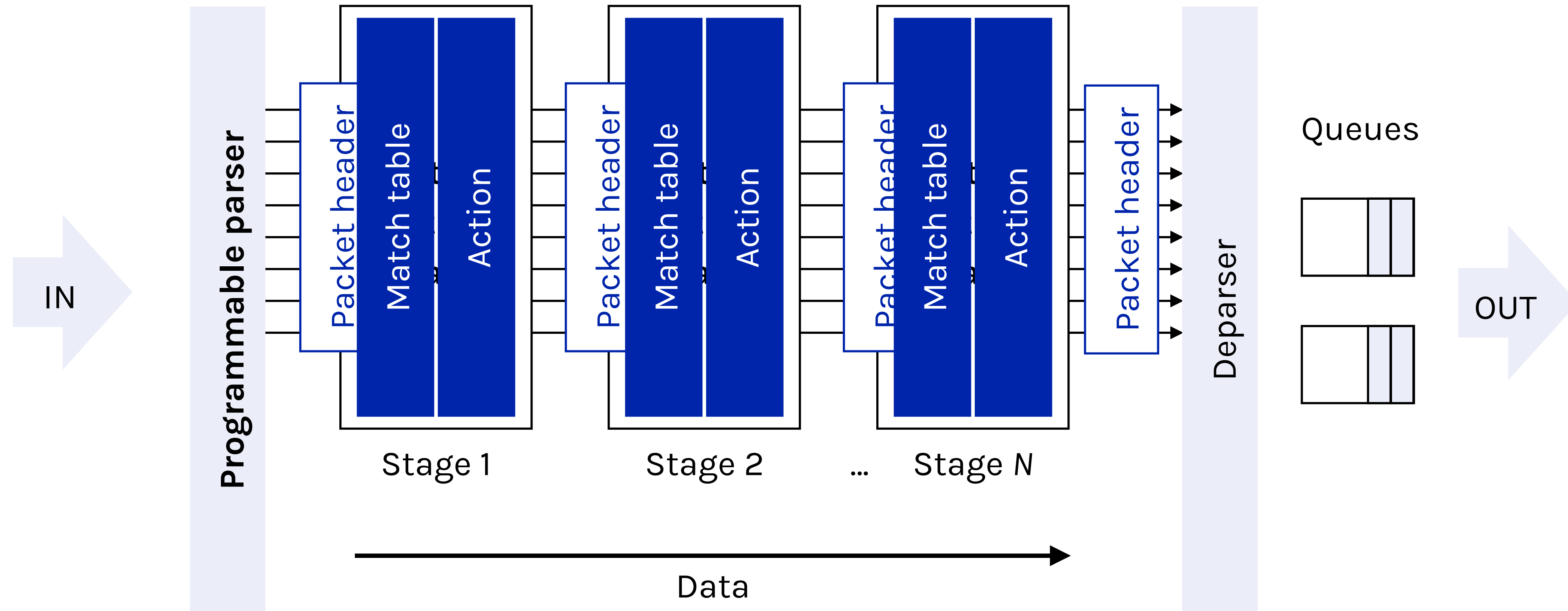
Programmable parser model



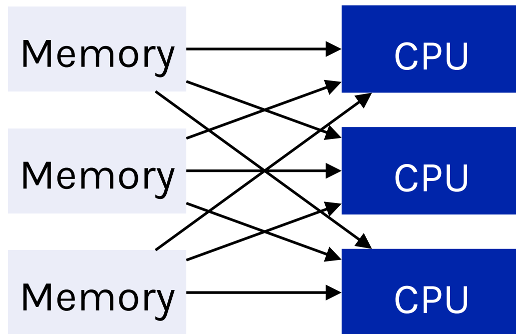
Match-action forwarding model



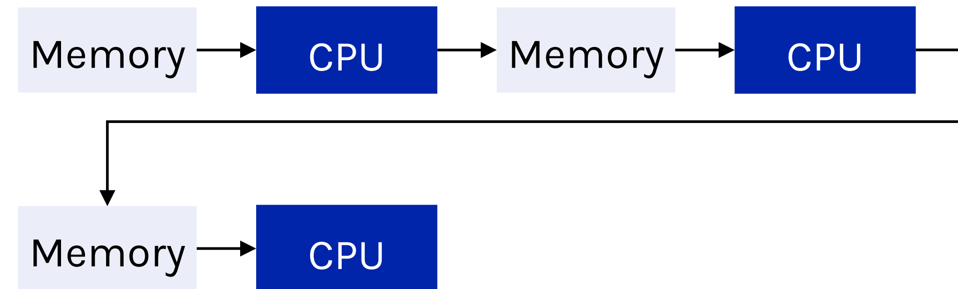
Match-action forwarding model



Match-action table performance vs. flexibility



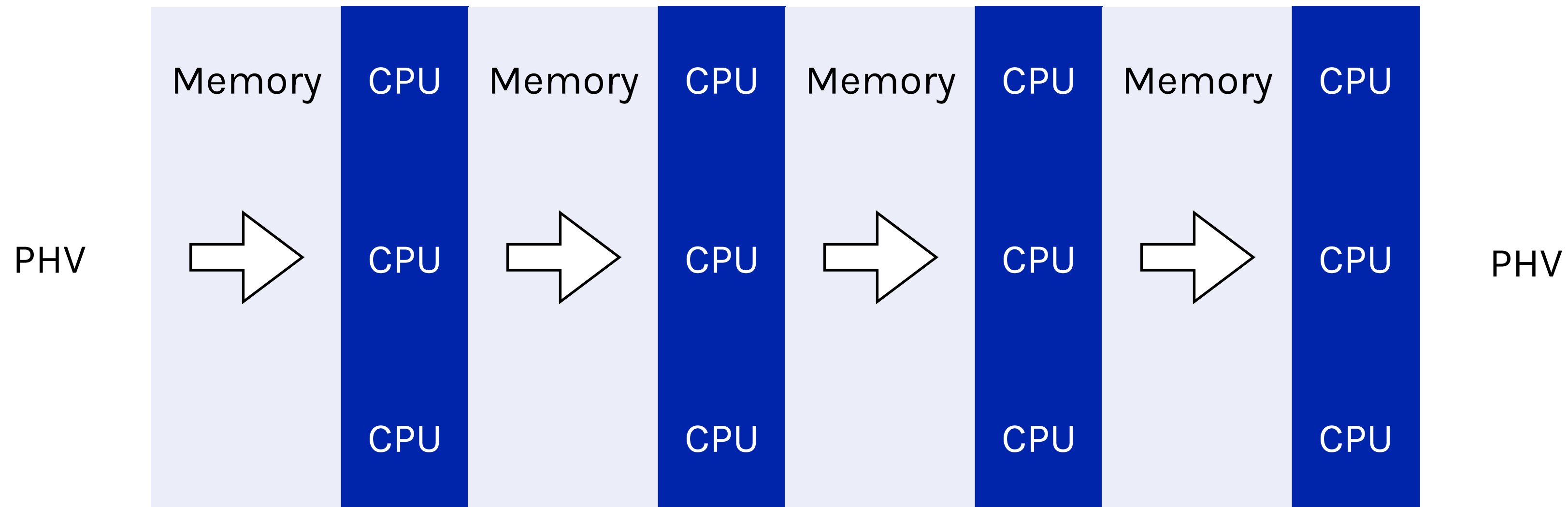
Multiprocessor:
memory bottleneck



Pipeline: similar to fixed-function switches, but with general-purpose CPUs for customizability

VLIW stages

VLIW: very large instruction words



Replicate CPUs and add **more stages** for finer granularity

VLIW processors

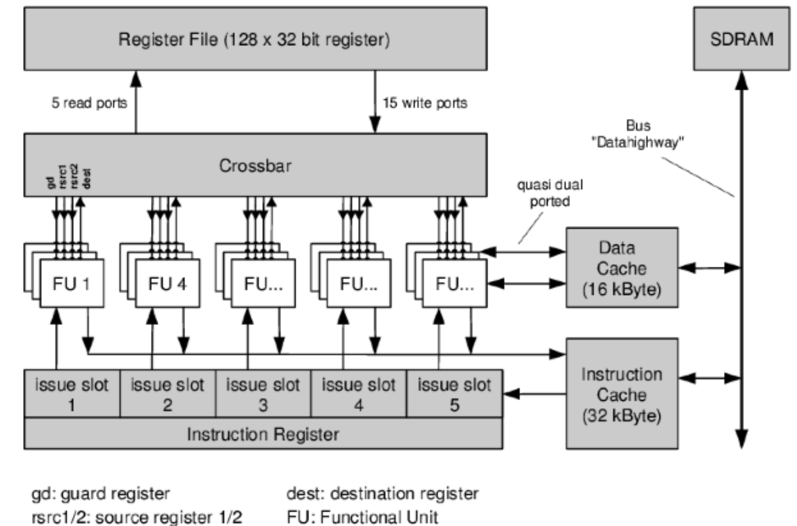
A fixed number of operations are formatted as one big instruction (called a bundle)

- Usually LIW (3 operations)
- Change in the instruction set architecture (ISA), i.e., one program counter points to one bundle (not one operation)

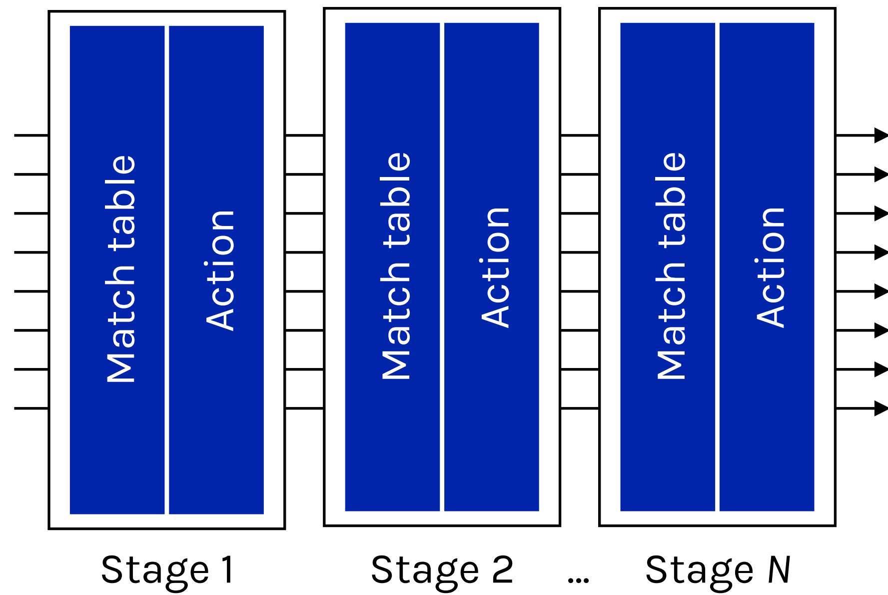
Operations organized in bundles to issue in parallel

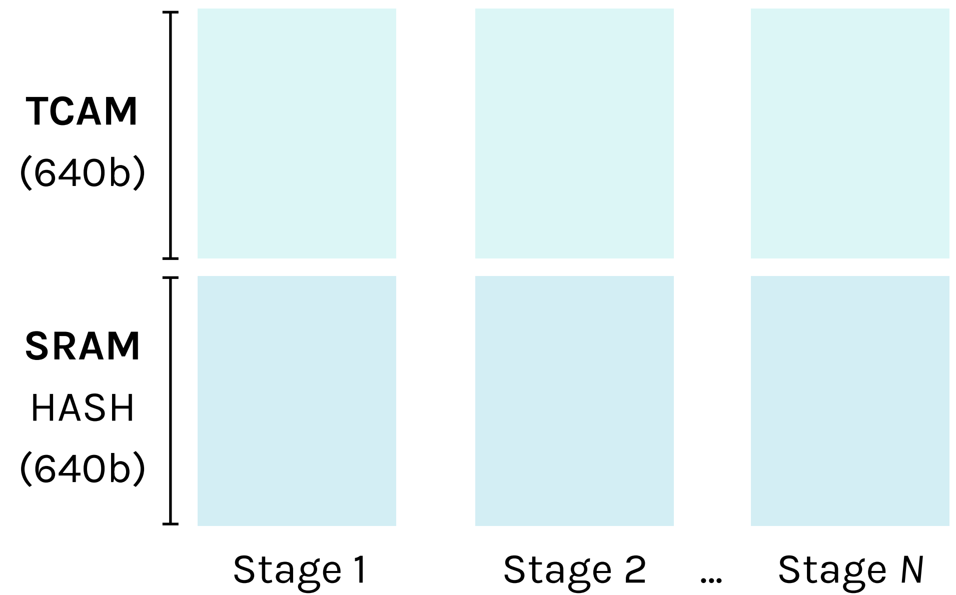
- Fixed format so could decode operations in parallel
- Enough FUs for types of operations that can issue in parallel

Instructions are scheduled by the compiler



RMT memory layout





RMT logical to physical table mapping

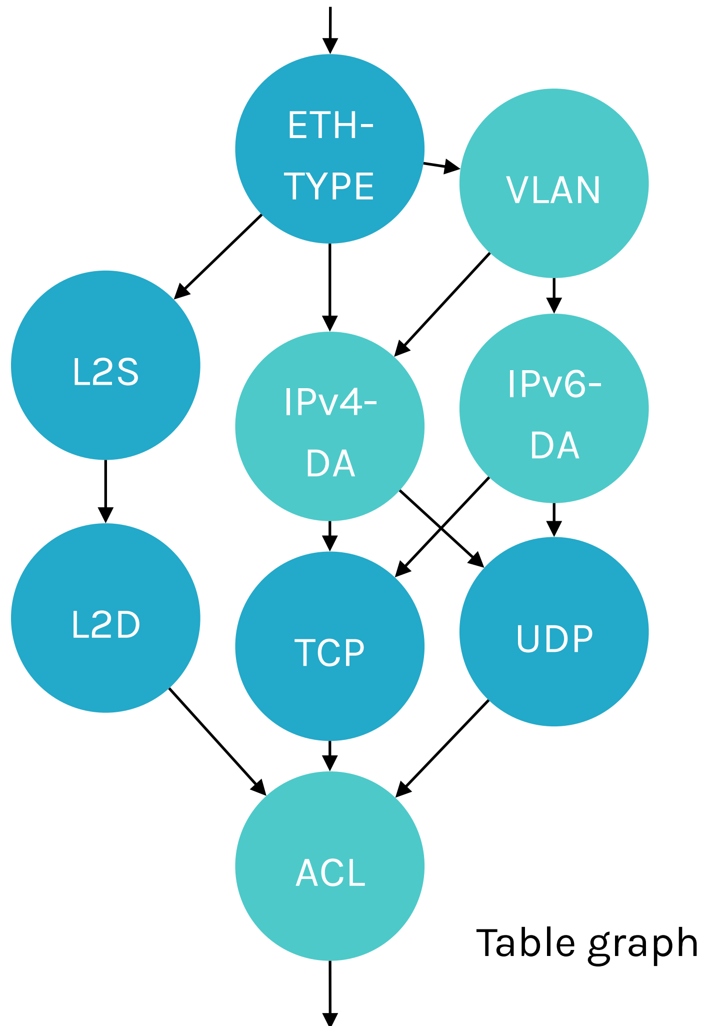
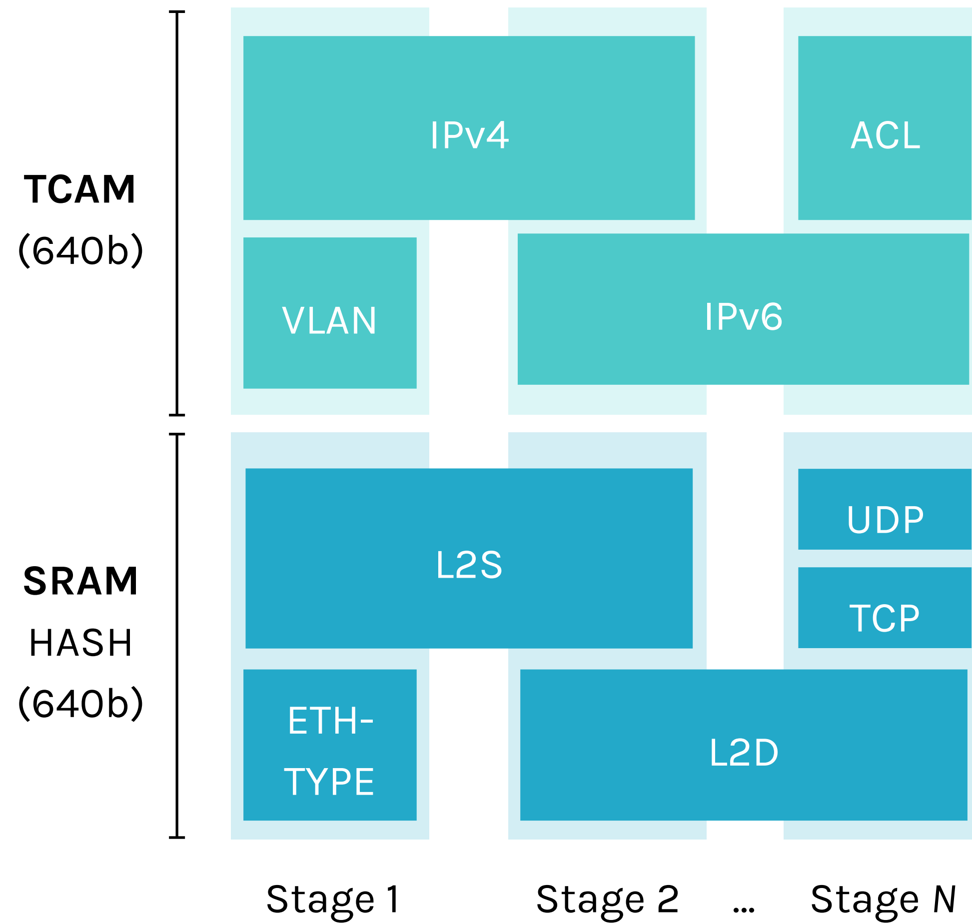
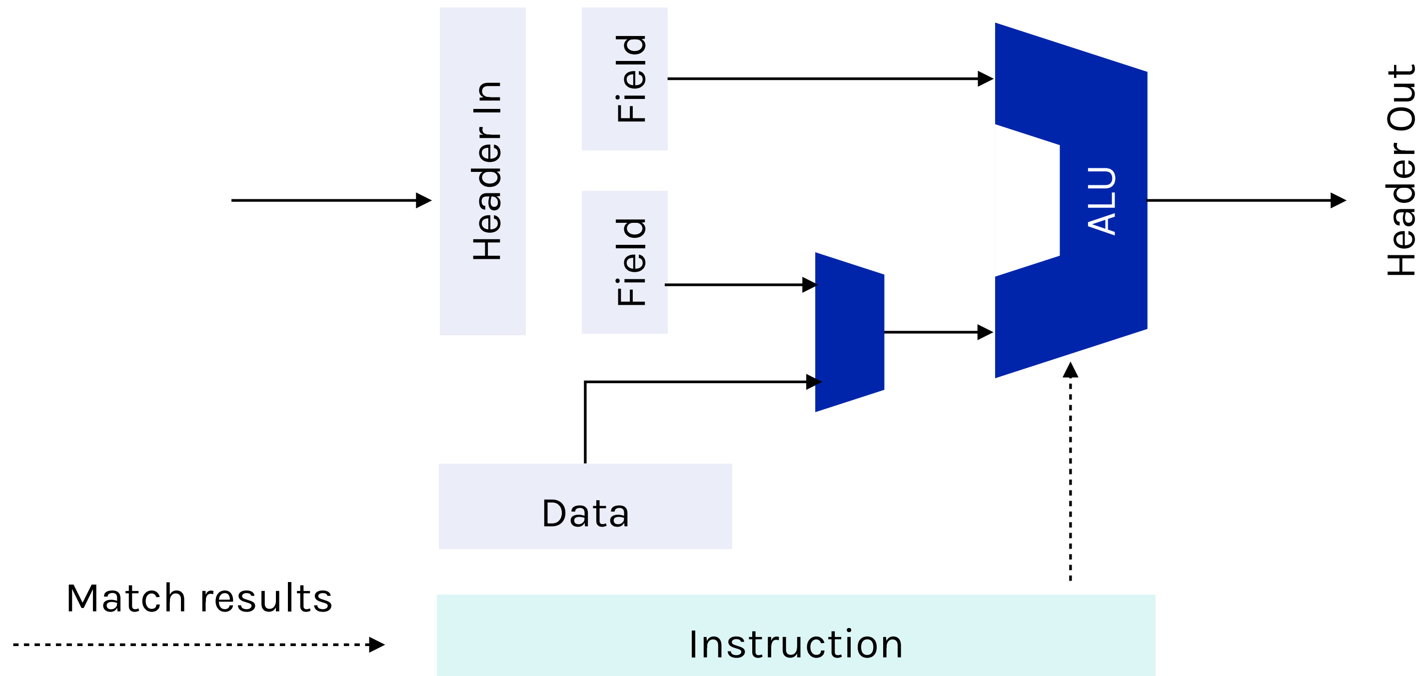


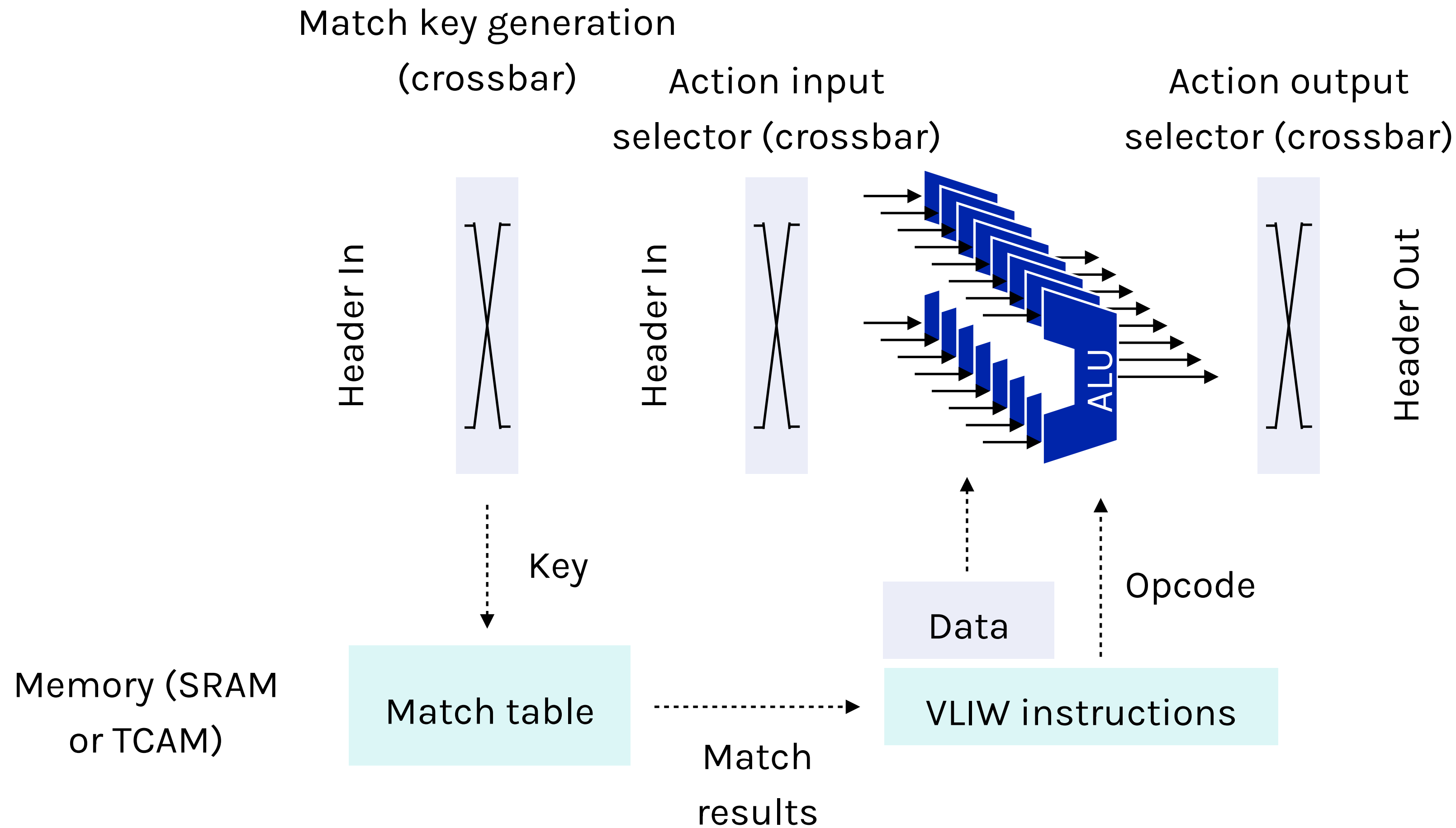
Table graph



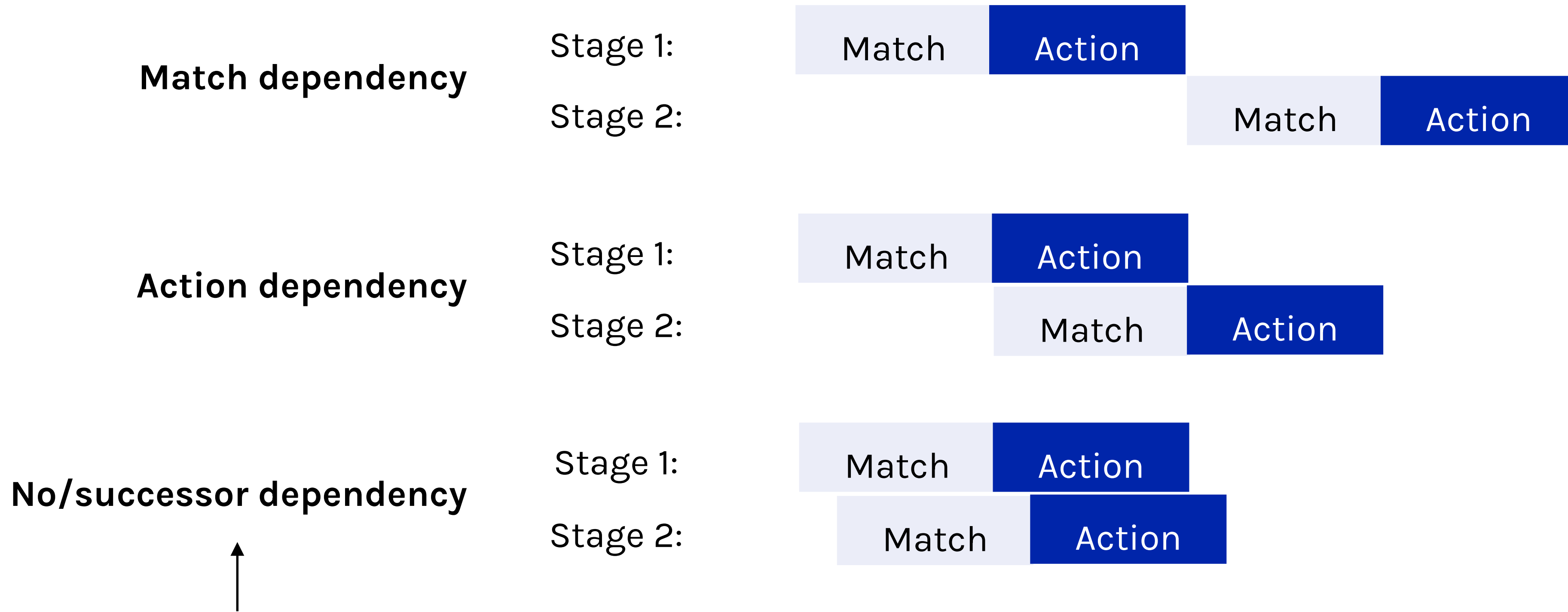
Action processing model



Multiple VLIW processors per stage



Cross stage parallelism via dependency analysis



Speculative execution and predication resolved before side effects are committed

Switch design and flexibility cost

64 x 10 Gbps ports

- 960M packets/second
- 1 GHz pipeline

Programmable parser

32 match-action stages

Huge TCAM: 10x current chips

- 64K TCAM words x 640b

SRAM has tables for exact matches

- 128K words x 640b

224 action processors per stage

All OpenFlow statistics counters

Total extra area cost: 14.2%, total extra power cost: 12.4%

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

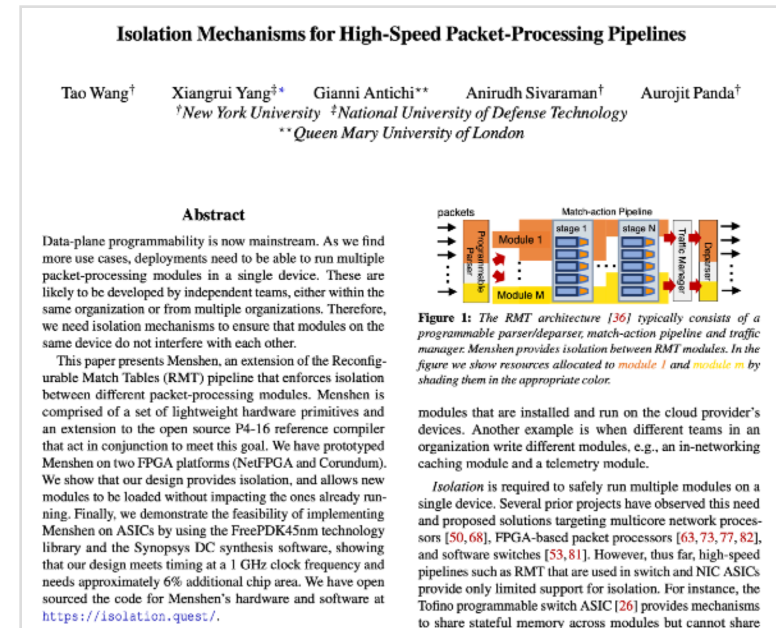
Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues 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

How to support isolation?

Isolation requirements

- **Behavior isolation:** one program cannot impact another's behavior or performance
- **Resource isolation:** resources should be allocated independently
- **Performance isolation:** one module's behavior should not affect the throughput and latency of another module
- **Lightweight:** low overhead to the high performance network device
- **Rapid reconfiguration:** quick update of the module program
- **No disruption:** during reconfiguration, must not disrupt the behavior of other unchanged modules



Isolation Mechanisms for High-Speed Packet-Processing Pipelines

Tao Wang[†] Xiangrui Yang[‡*] Gianni Antichi[**] Anirudh Sivaraman[†] Aurojit Panda[†]
[†]*New York University* [‡]*National University of Defense Technology*
[**]*Queen Mary University of London*

Abstract

Data-plane programmability is now mainstream. As we find more use cases, deployments need to be able to run multiple packet-processing modules in a single device. These are likely to be developed by independent teams, either within the same organization or from multiple organizations. Therefore, we need isolation mechanisms to ensure that modules on the same device do not interfere with each other.

This paper presents Menshen, an extension of the Reconfigurable Match Tables (RMT) pipeline that enforces isolation between different packet-processing modules. Menshen is comprised of a set of lightweight hardware primitives and an extension to the open source P4-16 reference compiler that act in conjunction to meet this goal. We have prototyped Menshen on two FPGA platforms (NetFPGA and Corundum). We show that our design provides isolation, and allows new modules to be loaded without impacting the ones already running. Finally, we demonstrate the feasibility of implementing Menshen on ASICs by using the FreePDK45nm technology library and the Synopsys DC synthesis software, showing that our design meets timing at a 1 GHz clock frequency and needs approximately 6% additional chip area. We have open sourced the code for Menshen's hardware and software at https://isolation.quest/.

Figure 1: *The RMT architecture [36] typically consists of a programmable parser/deparser, match-action pipeline and traffic manager. Menshen provides isolation between RMT modules. In the figure we show resources allocated to module 1 and module m by shading them in the appropriate color.*

modules that are installed and run on the cloud provider's devices. Another example is when different teams in an organization write different modules, e.g., an in-networking caching module and a telemetry module.

Isolation is required to safely run multiple modules on a single device. Several prior projects have observed this need and proposed solutions targeting multicore network processors [50, 68], FPGA-based packet processors [63, 73, 77, 82], and software switches [53, 81]. However, thus far, high-speed pipelines such as RMT that are used in switch and NIC ASICs provide only limited support for isolation. For instance, the Tofino programmable switch ASIC [26] provides mechanisms to share stateful memory across modules but cannot share

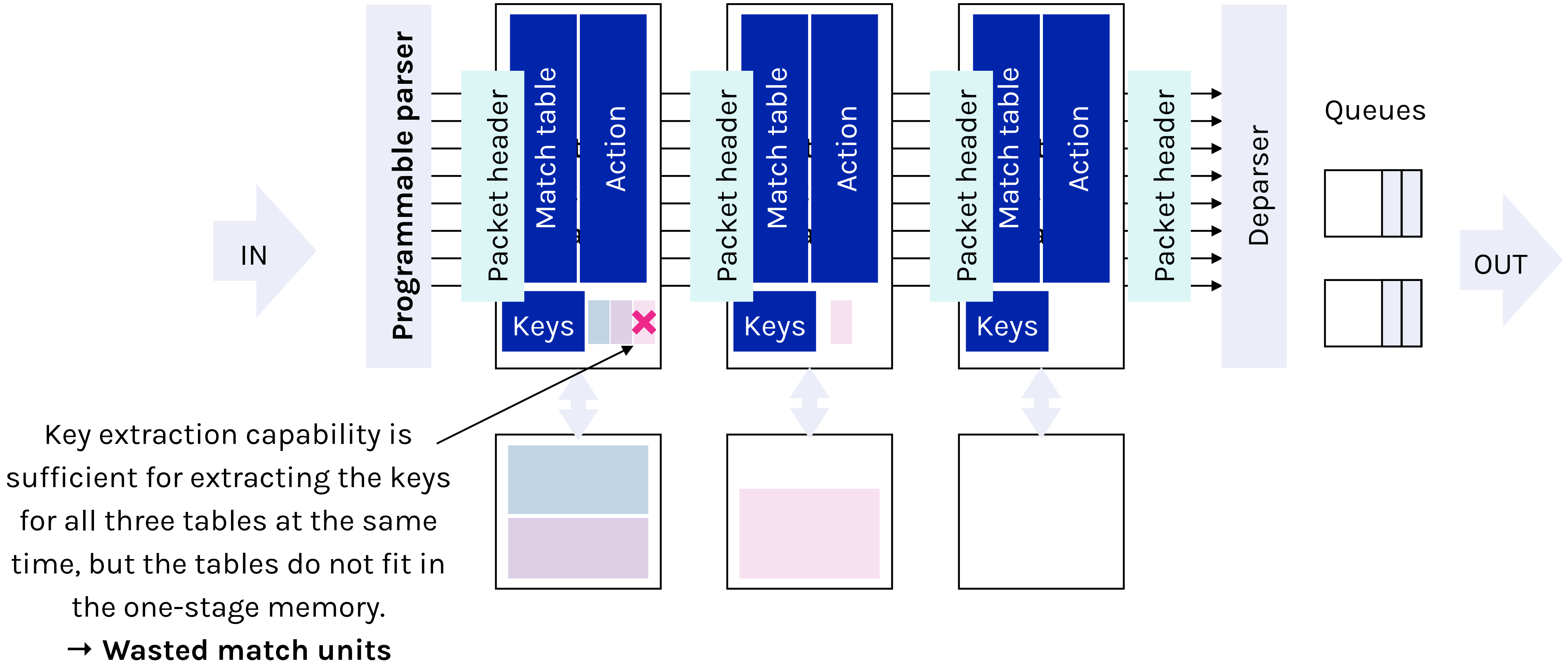
Menshen: an RMT extension for enforcing isolation

How to improve the resource efficiency of programmable data planes?

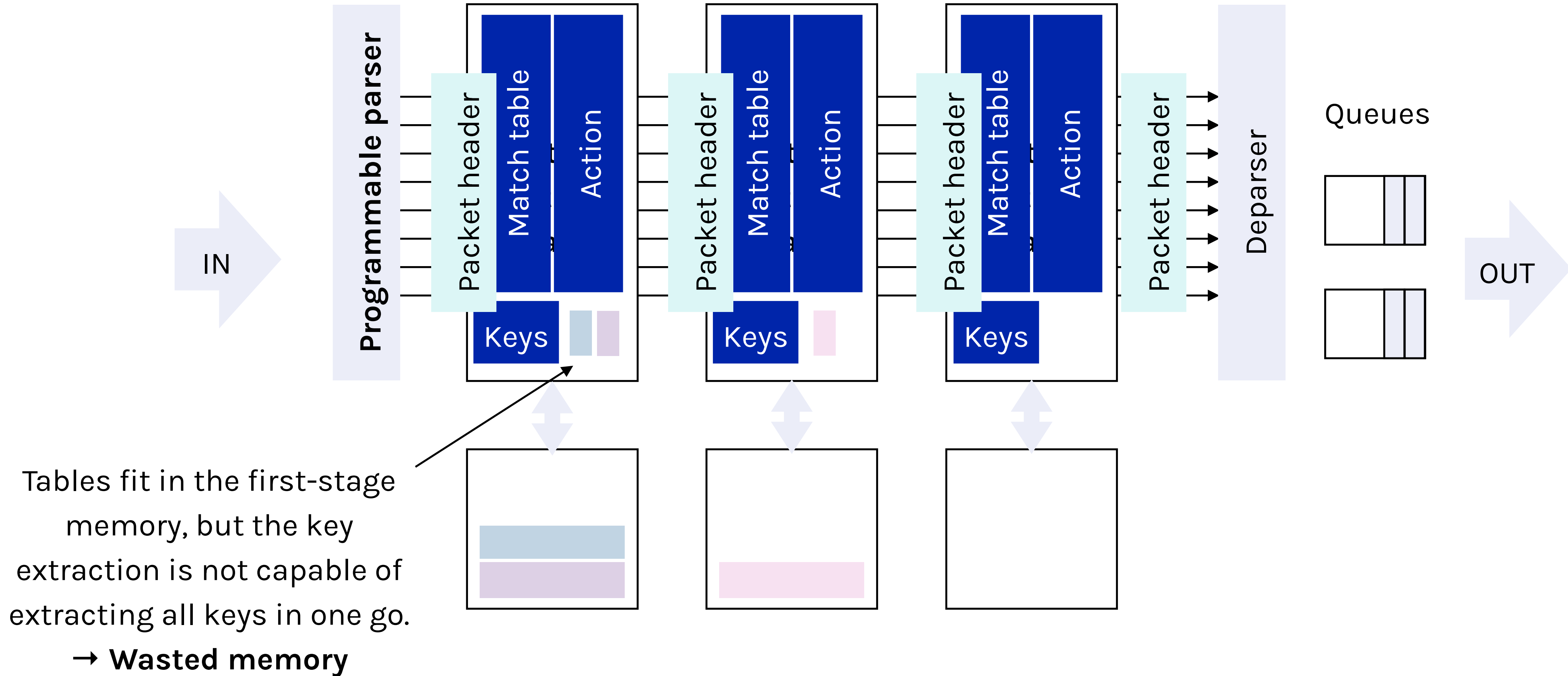
RMT recap

Resources are aggregated into stages that provide a **fixed ratio** of memory:match:action resources.

RMT limitations: misaligned hardware utilization

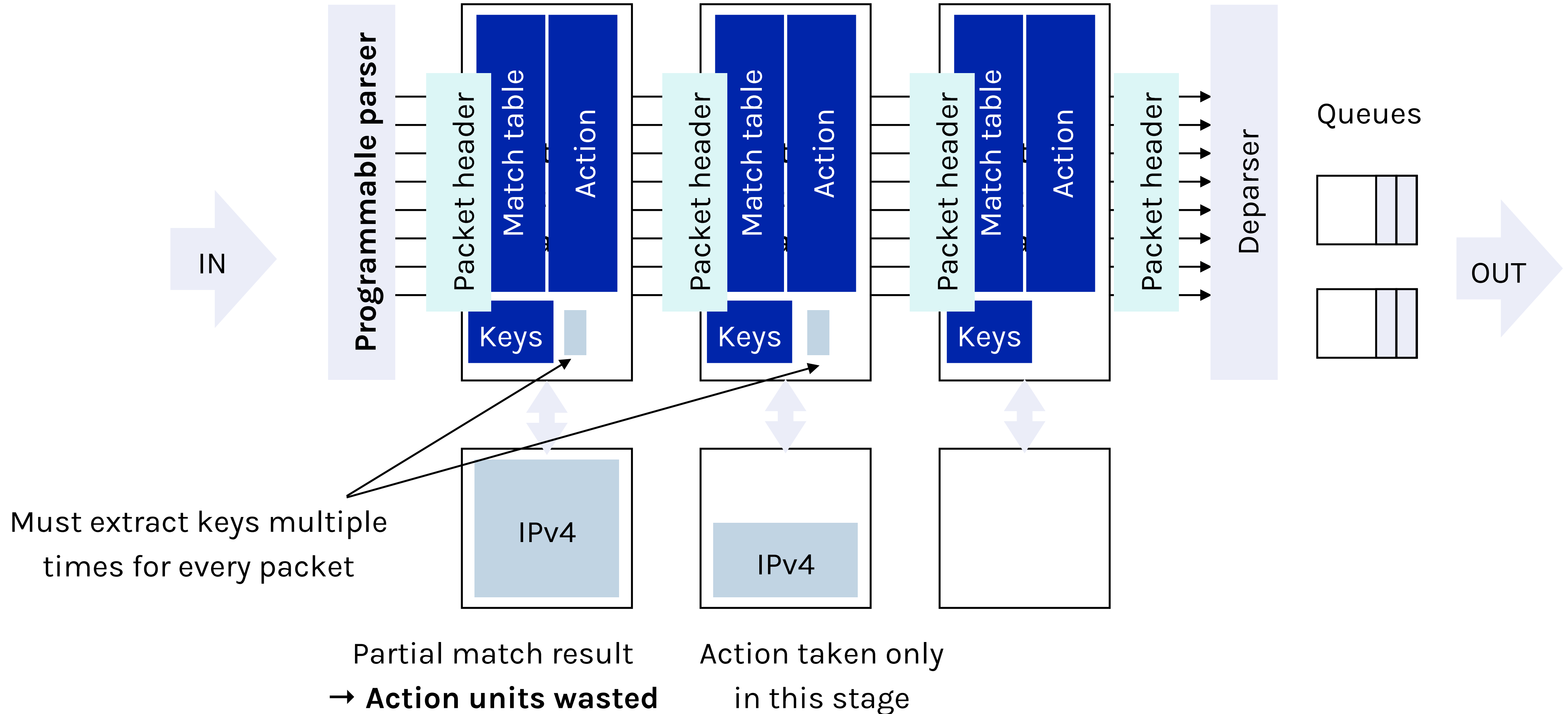


RMT limitations: misaligned hardware utilization

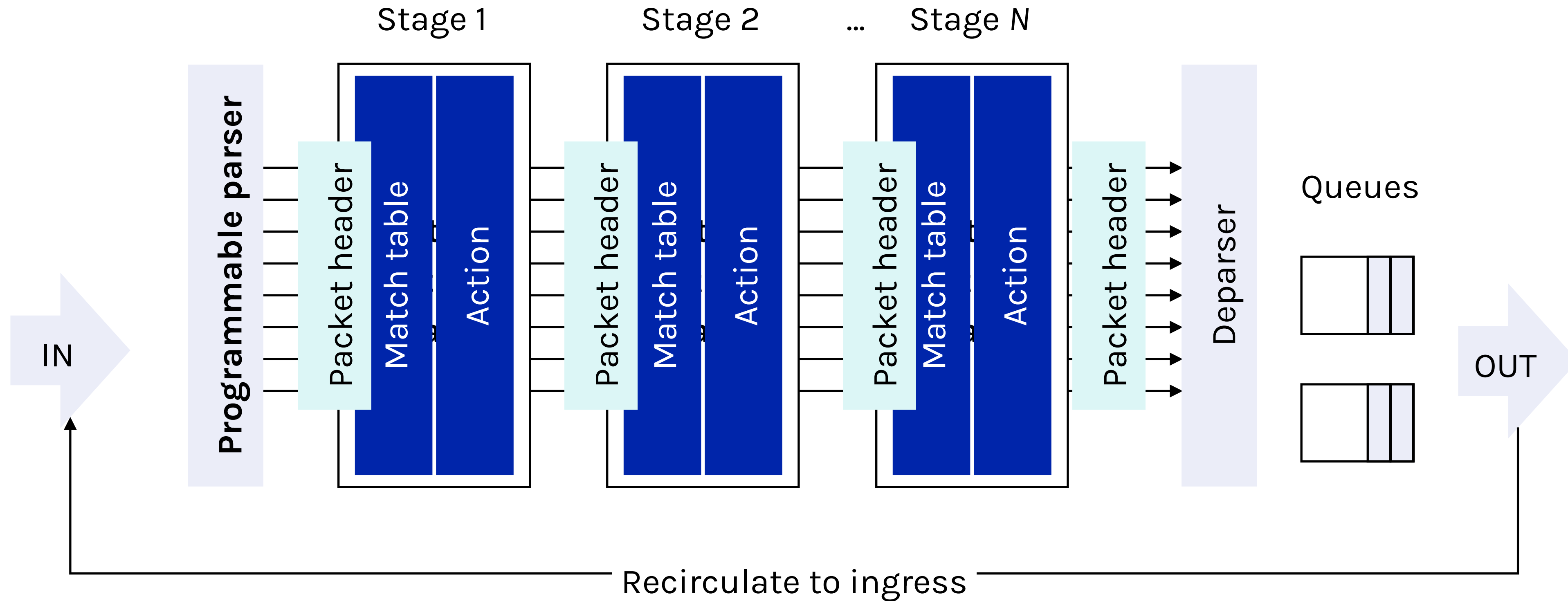


Tables fit in the first-stage memory, but the key extraction is not capable of extracting all keys in one go.
→ **Wasted memory**

More RMT limitations



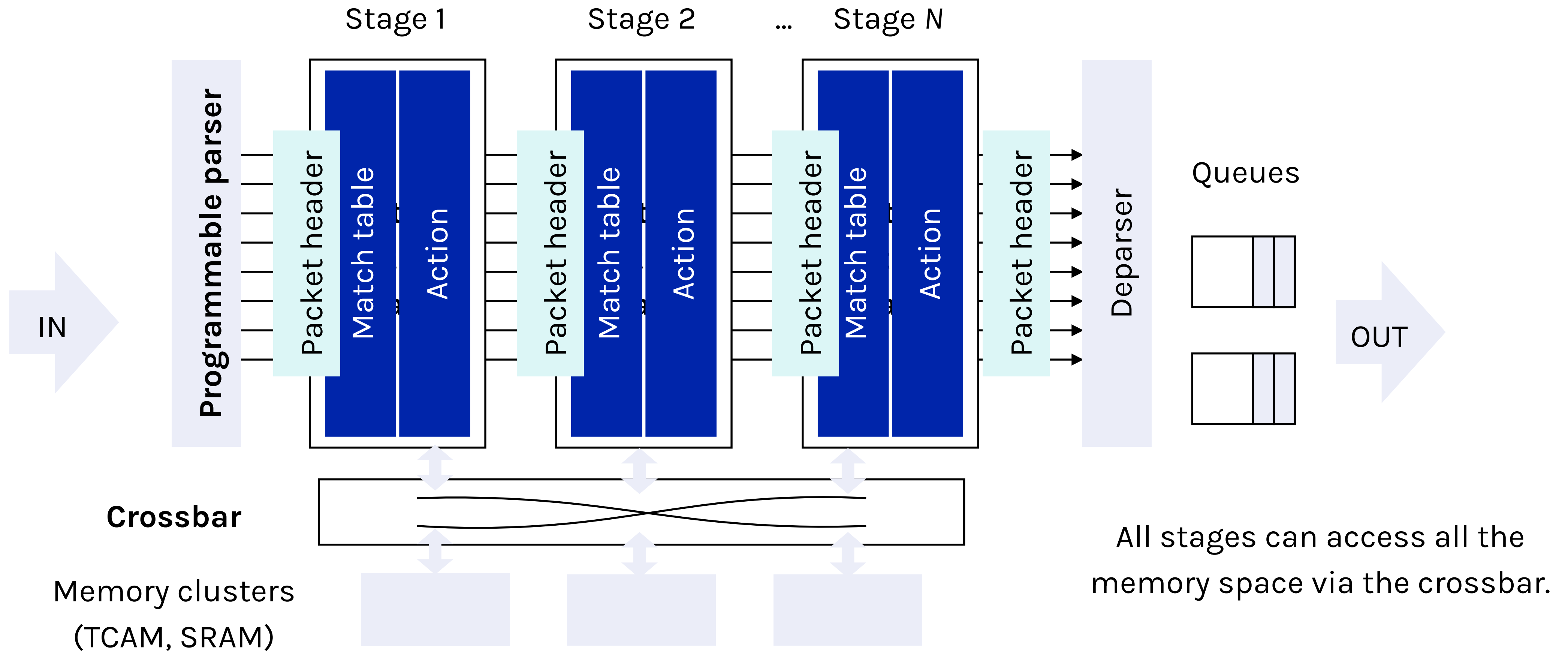
More RMT limitations



If the program does not fit, we need to recirculate packets to “extend” the pipeline → Throughput cut in half

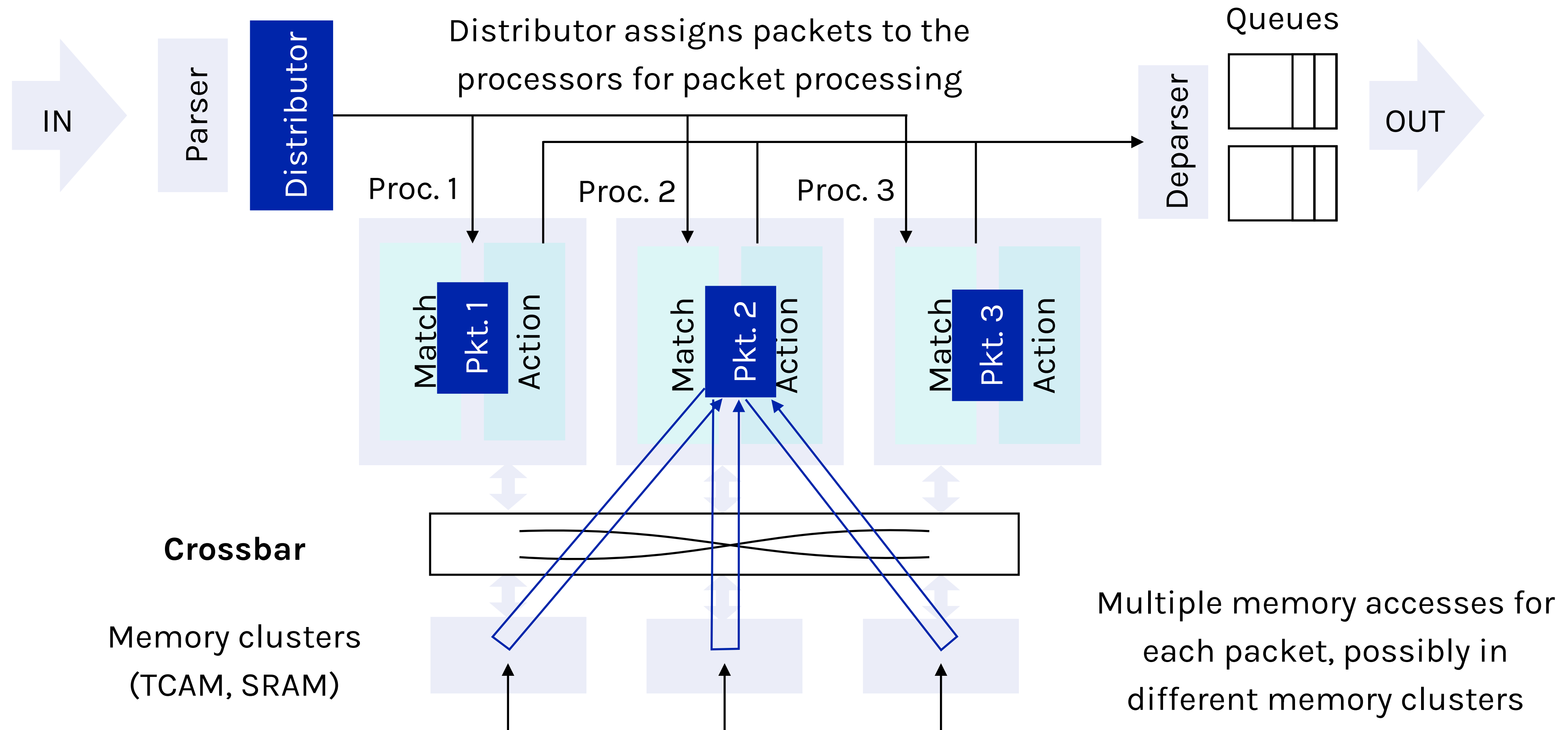
Improve resource efficiency via resource disaggregation

dRMT memory disaggregation

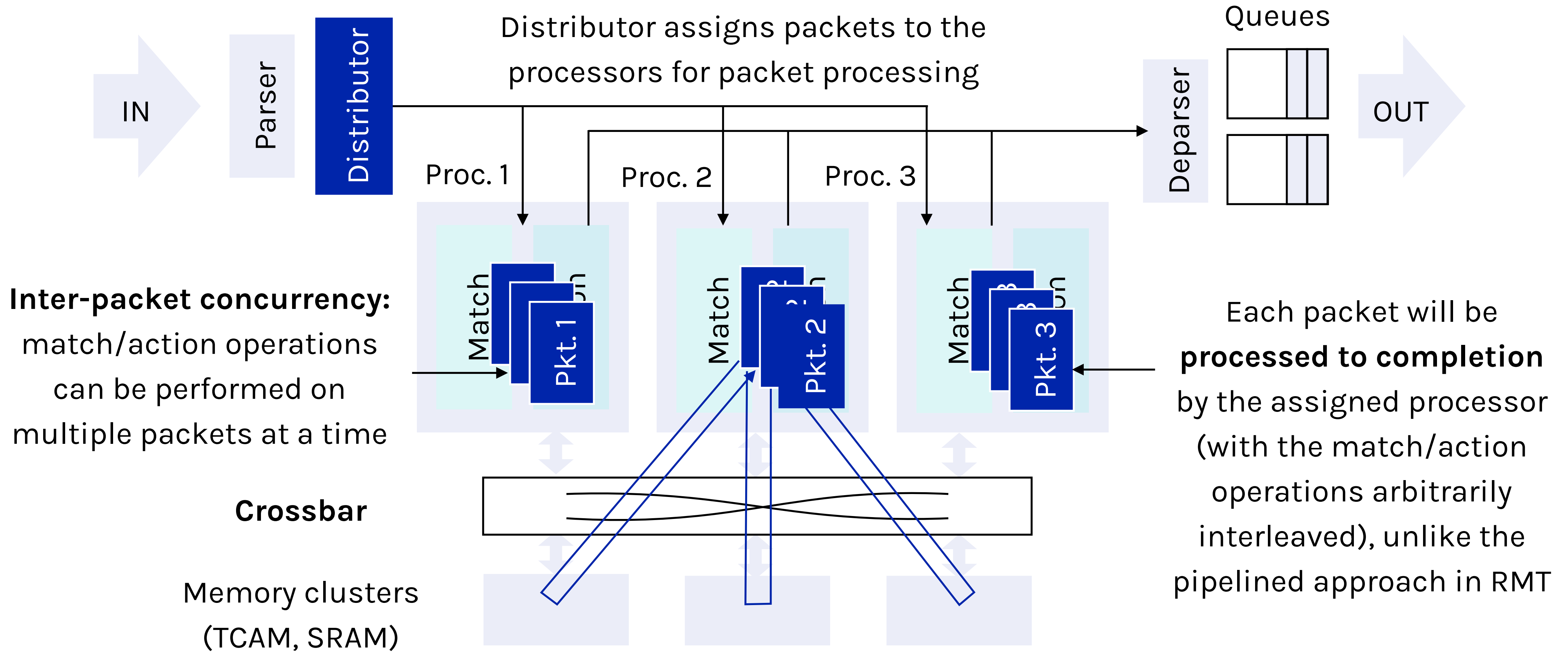


All stages can access all the memory space via the crossbar.

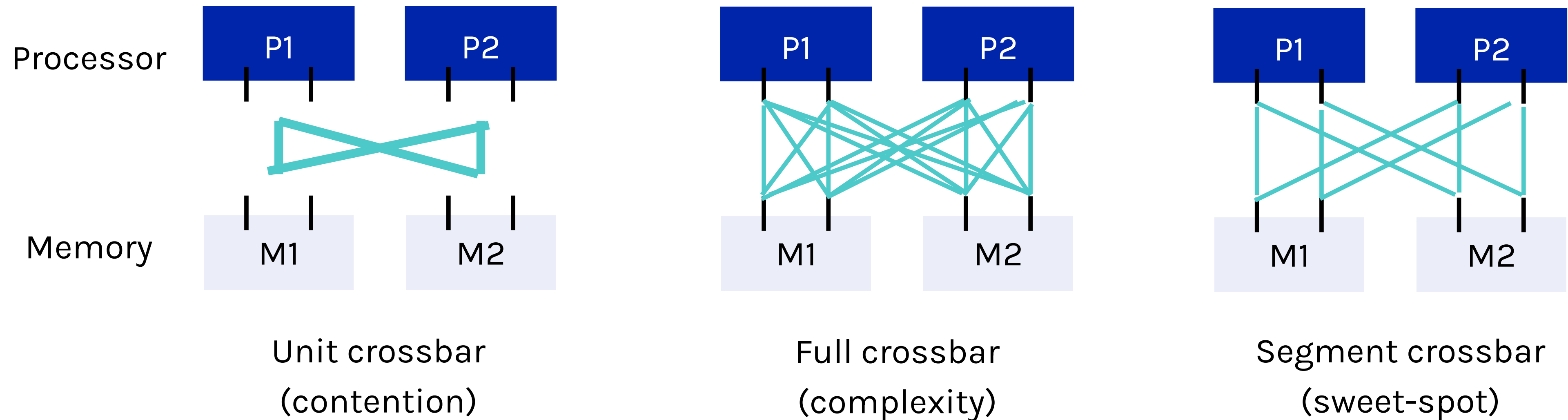
dRMT compute disaggregation



dRMT compute disaggregation

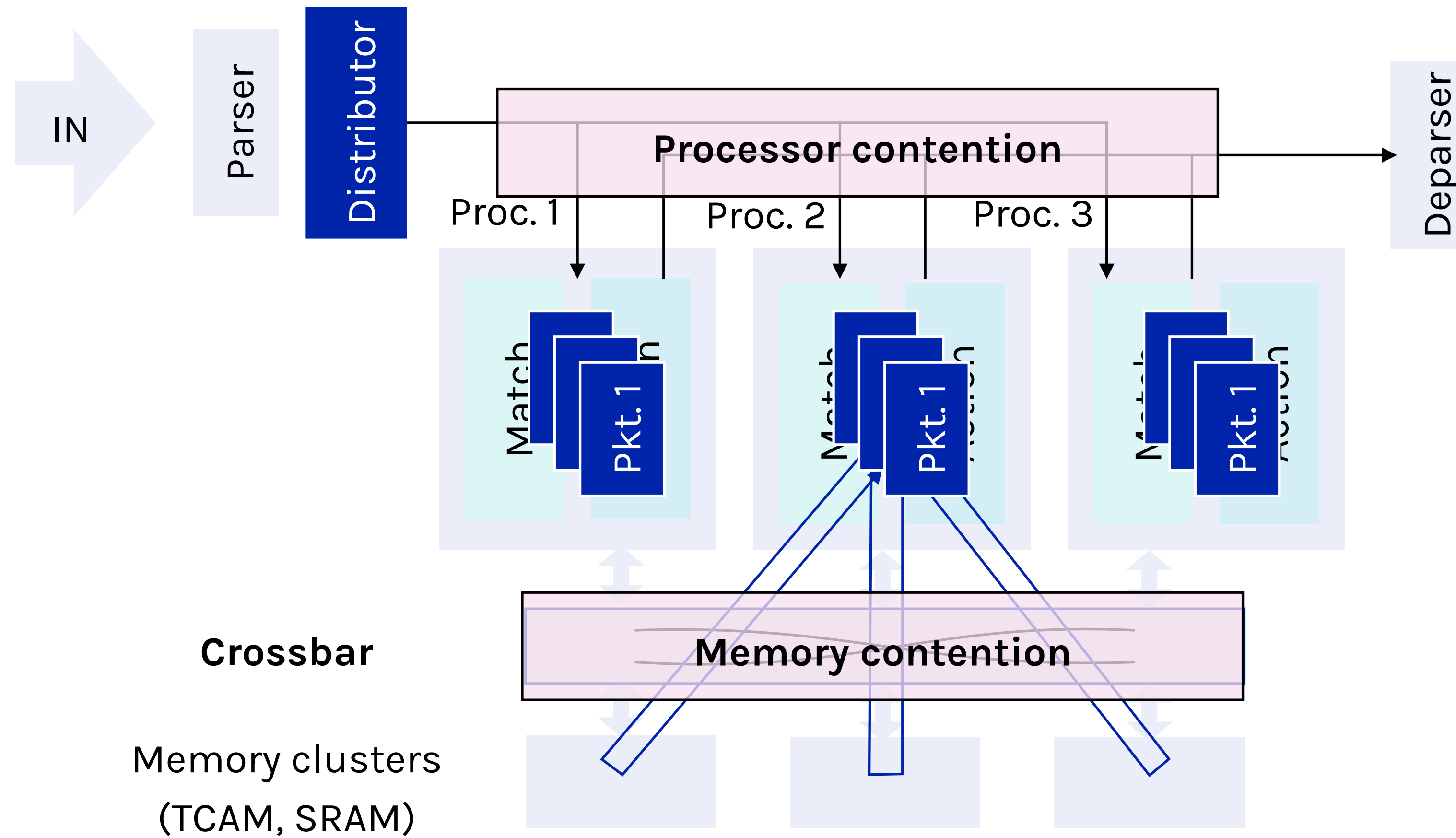


Crossbar design

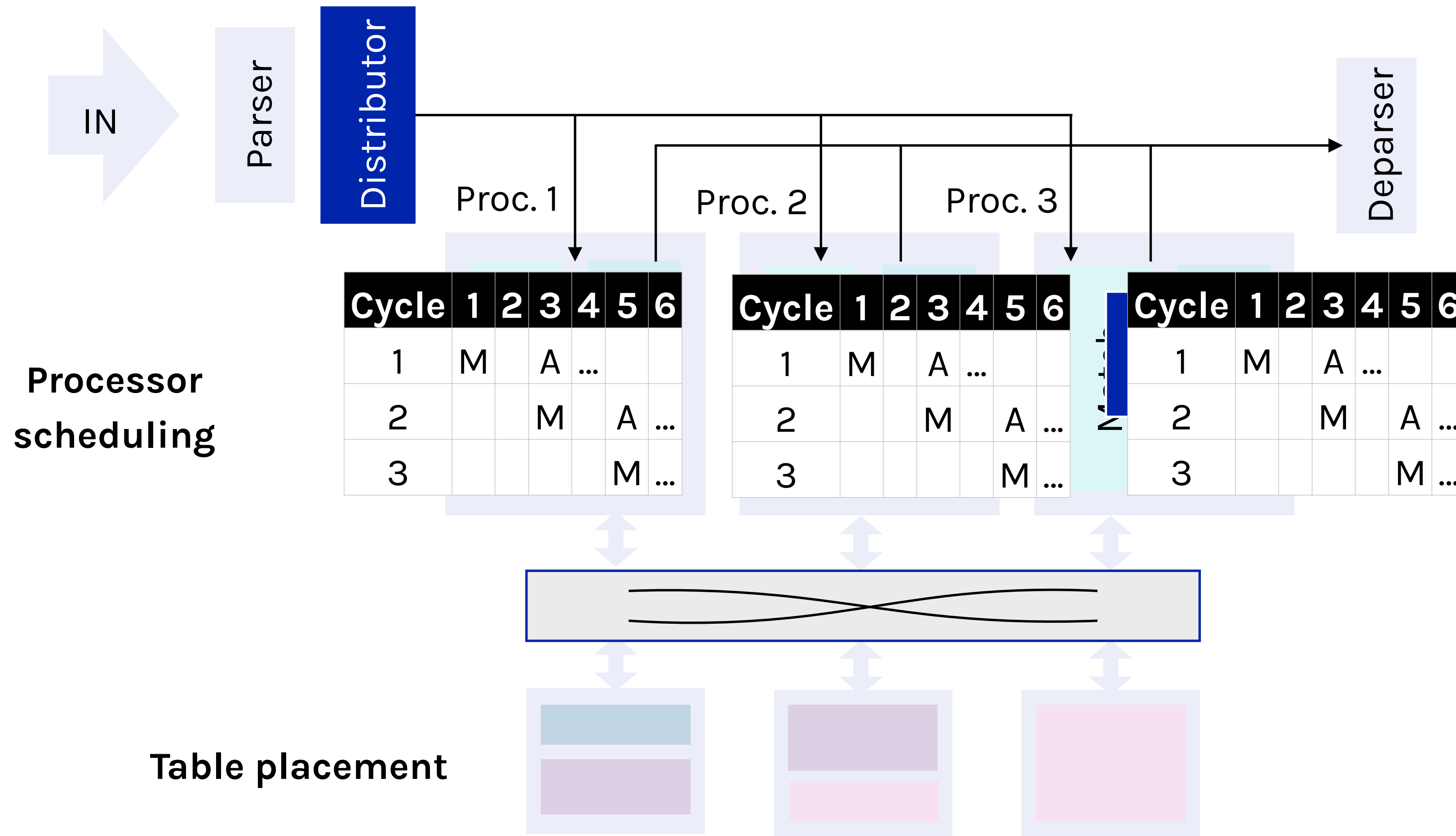


Wiring complexity similar to the unit crossbar and is equivalent to the full crossbar if tables are not split across memory clusters

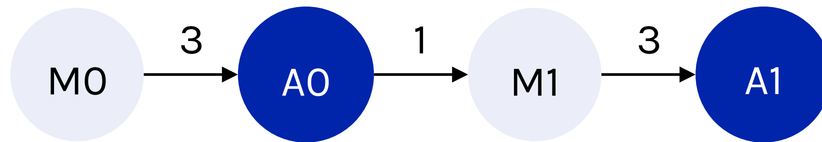
dRMT complexity

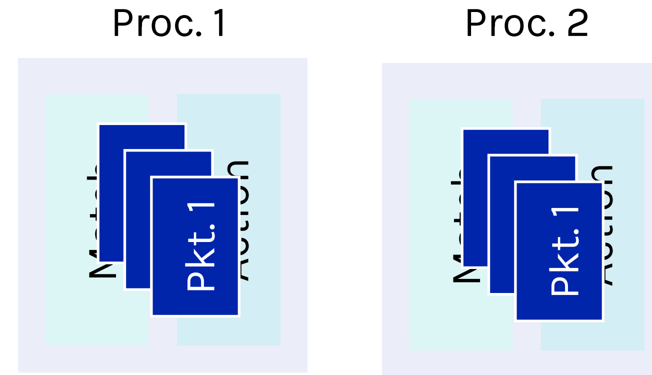


dRMT complexity



Processor scheduling example





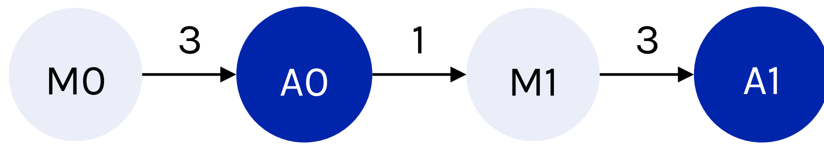
3 cycles to complete a match
1 cycle to complete an action

Two factors to consider:

Program dependencies

Resource constraints

Processor scheduling example



2 processors handle 1 packet per cycle

Packet arrives every 2 cycles per processor

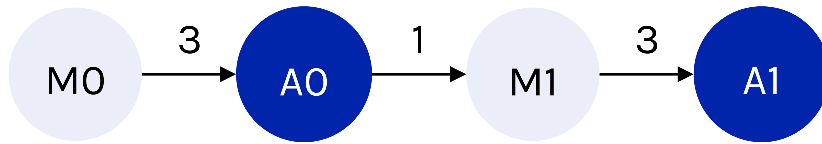
Cycles

Packets

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

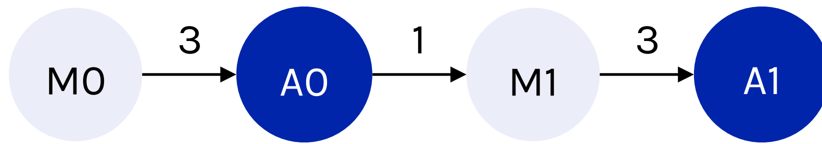
Cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	M1			A1											

Packets

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

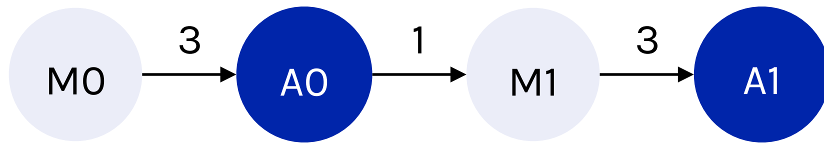
Cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	M1			A1											
2			M0			A0	M1			A1									

Packets

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

Cycles

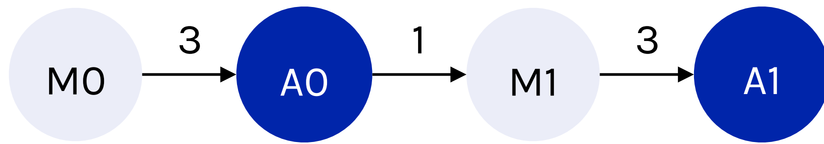
Packets

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	M1			A1											
2			M0			A0	M1			A1									
3					M0			A0	M1			A1							

Is this schedule feasible?

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

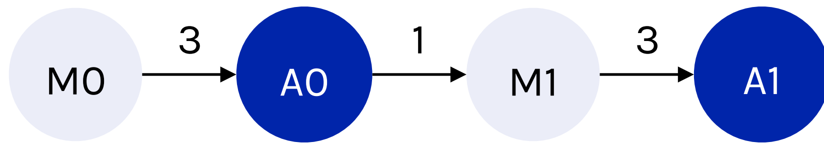
Cycles

Packets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	M1			A1											
2			M0			A0	M1			A1									
3					M0			A0	M1			A1							

Every processor can only do 1 match per cycle

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

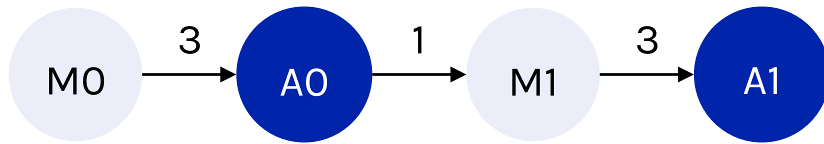
Cycles

Packets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	X	M1			A1										

Delay M1 by inserting a no-op

Schedule per processor

Processor scheduling example



2 processors handle 1 packet per cycle
Packet arrives every 2 cycles per processor

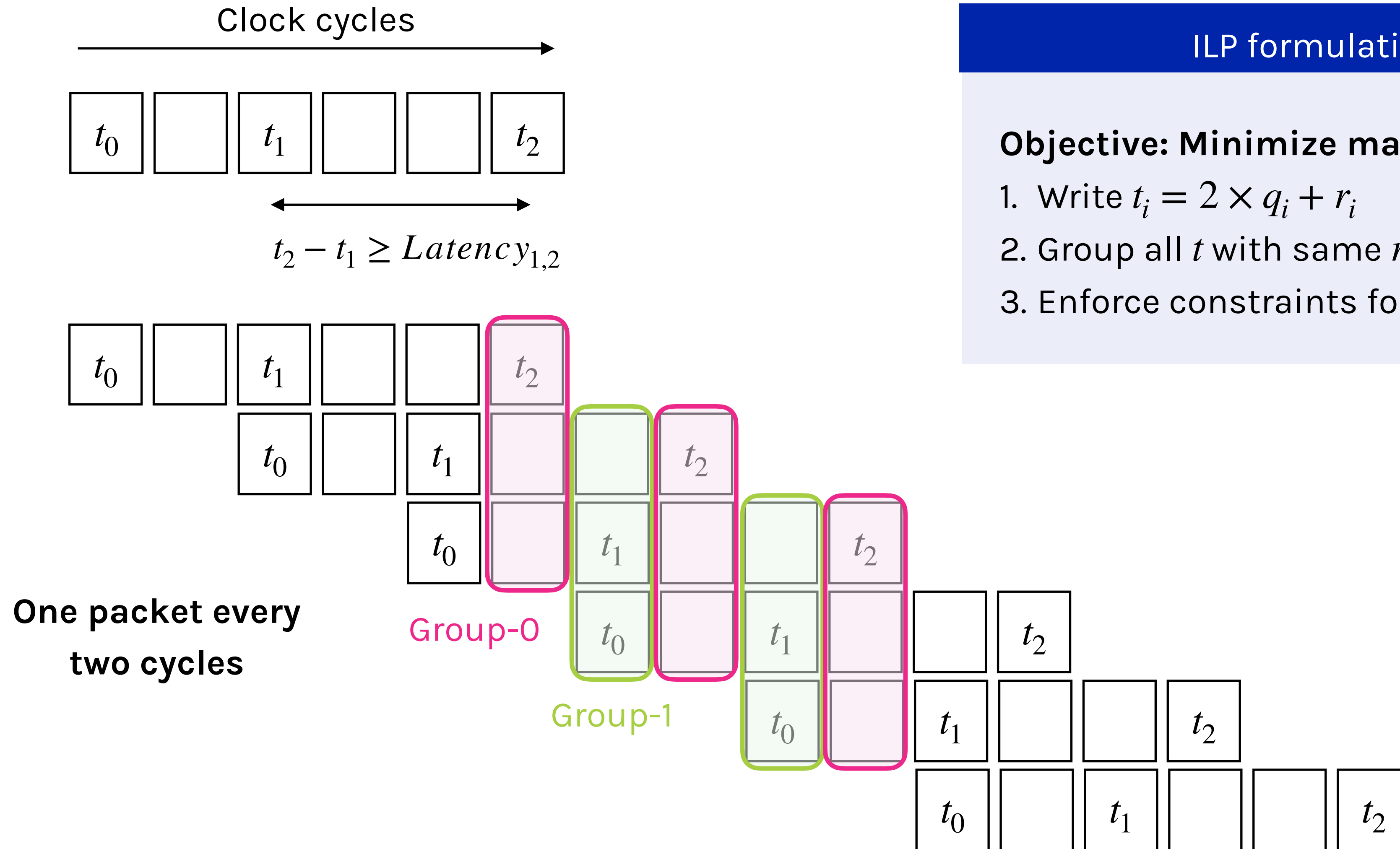
Cycles

Packets

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	M0			A0	X	M1			A1										
2			M0			A0	X	M1			A1								
3					M0			A0	X	M1			A1						
4							M0			A0	X	M1			A1				
5									M0			A0	X	M1			A1		
6											M0			A0	X	M1			A1

Schedule per processor

Minimizing delays



ILP formulation

Objective: Minimize max t_i

1. Write $t_i = 2 \times q_i + r_i$
2. Group all t with same r
3. Enforce constraints for each group

dRMT summary

Can dRMT provide deterministic performance guarantees?

- Yes, **compiler schedules programs** using an Integer Linear Program (ILP) to eliminate memory and processor contention

How does dRMT compare with RMT on real programs

- Needs (4.5% - 50%) **fewer processors** on real and synthetic P4 programs for achieving line rate

Is dRMT feasible in hardware?

- Yes, dRMT takes up some more chip area than RMT, but the additional area is **modest** relative to a switching chip

dRMT: Disaggregated Programmable Switching

Sharad Chole[1], Andy Fingerhut[1], Sha Ma[1], Anirudh Sivaraman[2], Shay Vargaftik[3], Alon Berger[3], Gal Mendelson[3], Mohammad Alizadeh[2], Shang-Tse Chuang[1], Isaac Keslassy[3,4], Ariel Orda[3], Tom Edsall[1]

[1] *Cisco Systems, Inc.* [2] *MIT* [3] *Technion* [4] *VMware, Inc.*

ABSTRACT

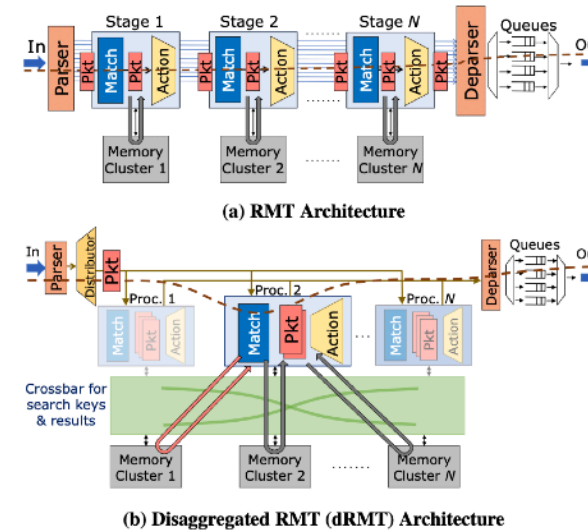
We present dRMT (disaggregated Reconfigurable Match-Action Table), a new architecture for programmable switches. dRMT overcomes two important restrictions of RMT, the predominant pipeline-based architecture for programmable switches: (1) table memory is local to an RMT pipeline stage, implying that memory not used by one stage cannot be reclaimed by another, and (2) RMT is hard-wired to always sequentially execute matches followed by actions as packets traverse pipeline stages. We show that these restrictions make it difficult to execute programs efficiently on RMT.

dRMT resolves both issues by disaggregating the memory and compute resources of a programmable switch. Specifically, dRMT moves table memories out of pipeline stages and into a centralized pool that is accessible through a crossbar. In addition, dRMT replaces RMT's pipeline stages with a cluster of processors that can execute match and action operations in any order.

We show how to schedule a P4 program on dRMT at compile time to guarantee deterministic throughput and latency. We also present a hardware design for dRMT and analyze its feasibility and chip area. Our results show that dRMT can run programs at line rate with fewer processors compared to RMT, and avoids performance cliffs when there are not enough processors to run a program at line rate. dRMT's hardware design incurs a modest increase in chip area relative to RMT, mainly due to the crossbar.

CCS CONCEPTS

• **Networks** → **Programmable networks**; *Routers*;



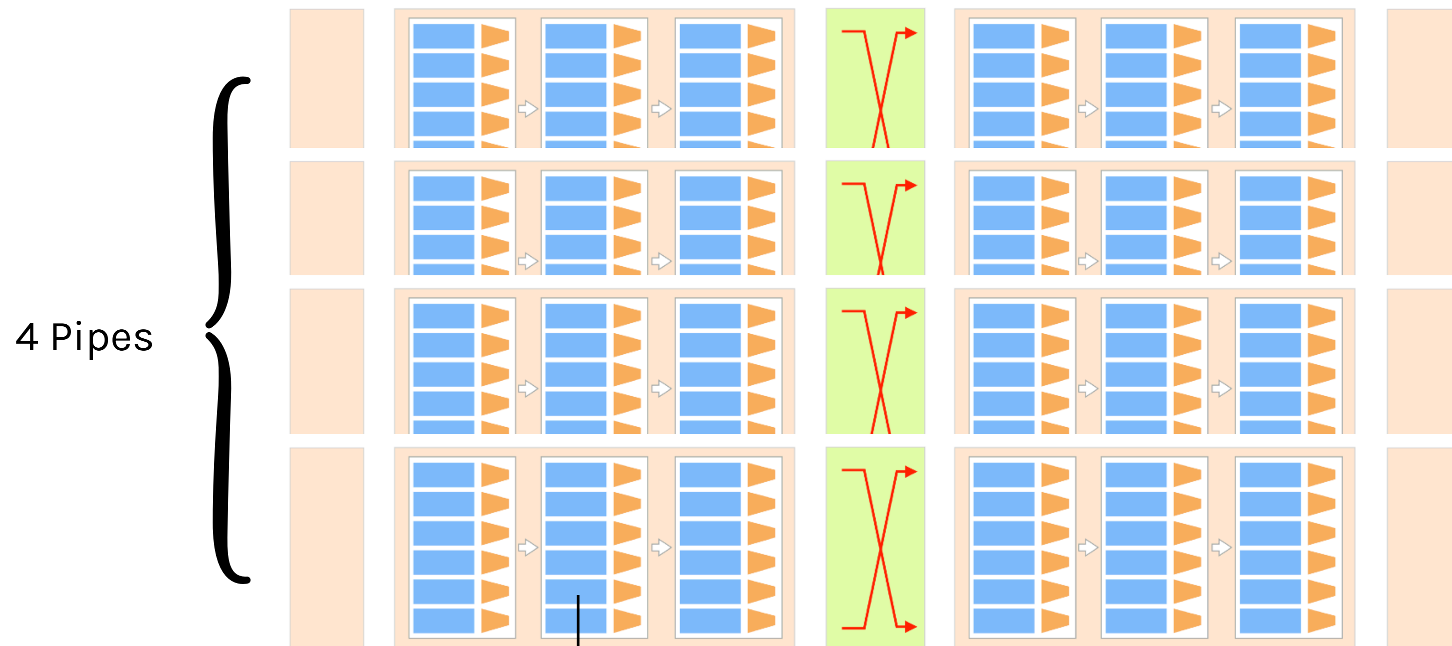
(a) RMT Architecture

(b) Disaggregated RMT (dRMT) Architecture

Figure 1: Comparison of the RMT [16] and dRMT architectures. dRMT replaces RMT's pipeline stages with run-to-completion match-action processors, and separates the memory clusters from the processors via a crossbar. The dashed arrows show the flow of a packet through each architecture.

Open Tofino Native Architecture (TNA)

Lab5 requires you to respect these constraints

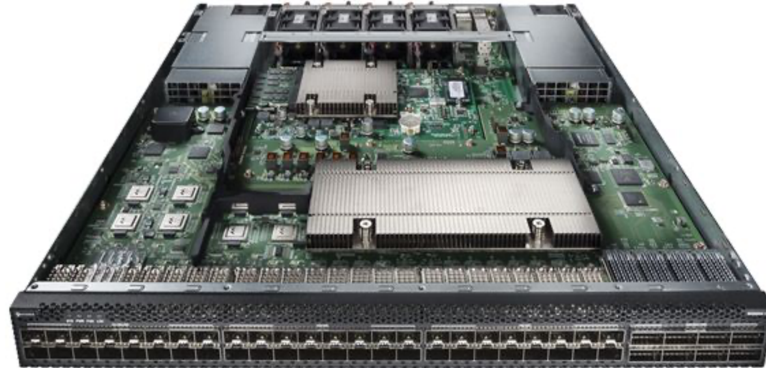


Registers as externs for persistent memory (local to a stage and only read-modify-write once allowed)

```
control example<M>(M meta) {
    Register<bit<32>, _>(4096) counters;
    RegisterAction<_, _, void> increment_counter = {
        void apply(inout bit<32> value) {
            value = value + meta.increment_amount;
        }
    };
    action trigger_counter() {
        increment_counter.execute(meta.index);
    }
}
```

https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf

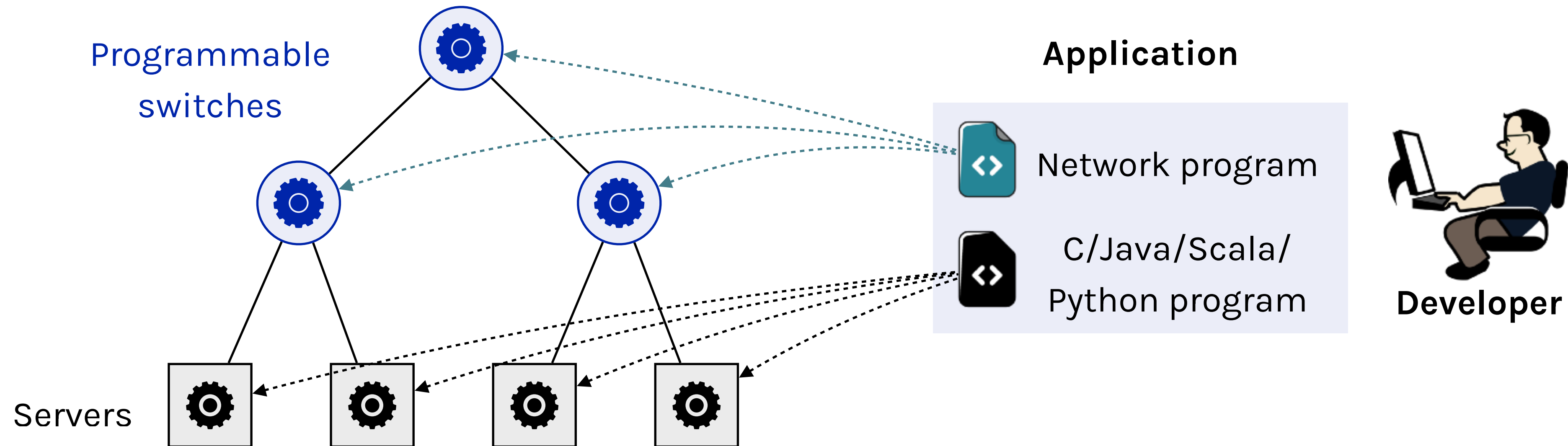
Summary



RMT for implementing programmable data planes

dRMT for improving resource efficiency of programmable data planes via resource disaggregation

Next time: in-network computing applications



What **innovative ways** of using programmable data planes do we have?