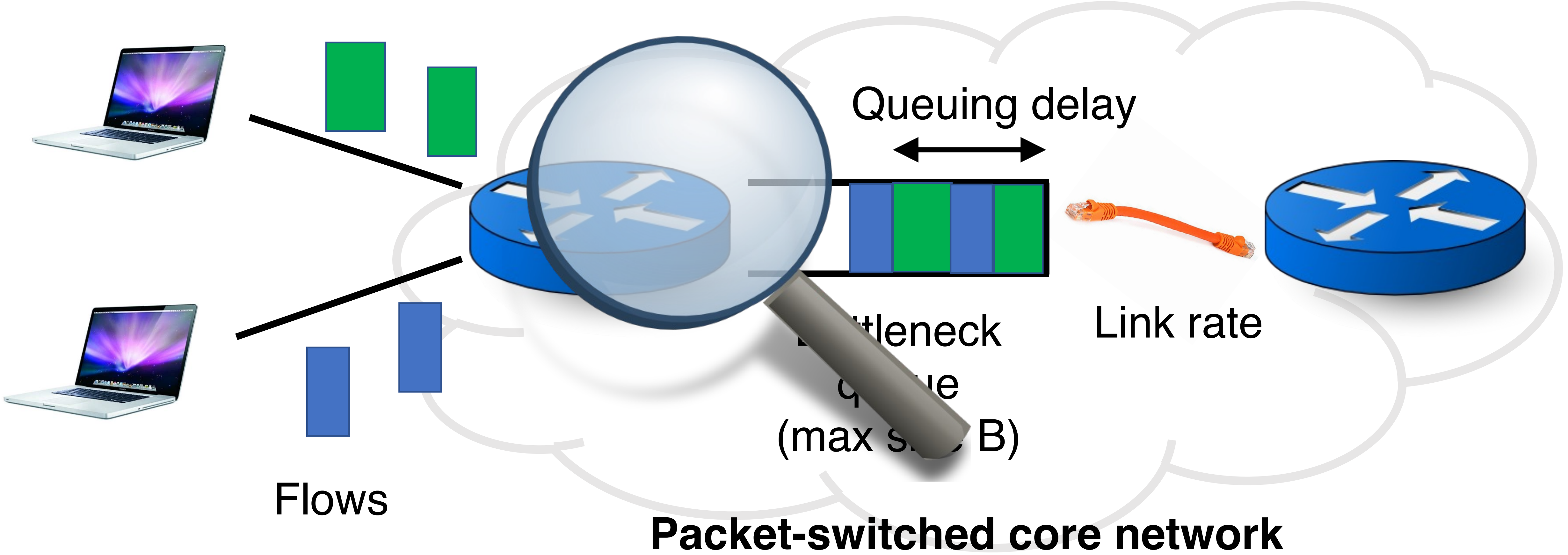
Network

RUTGERS
UNIVERSITY | NEW BRUNSWICK

What’s in a router?



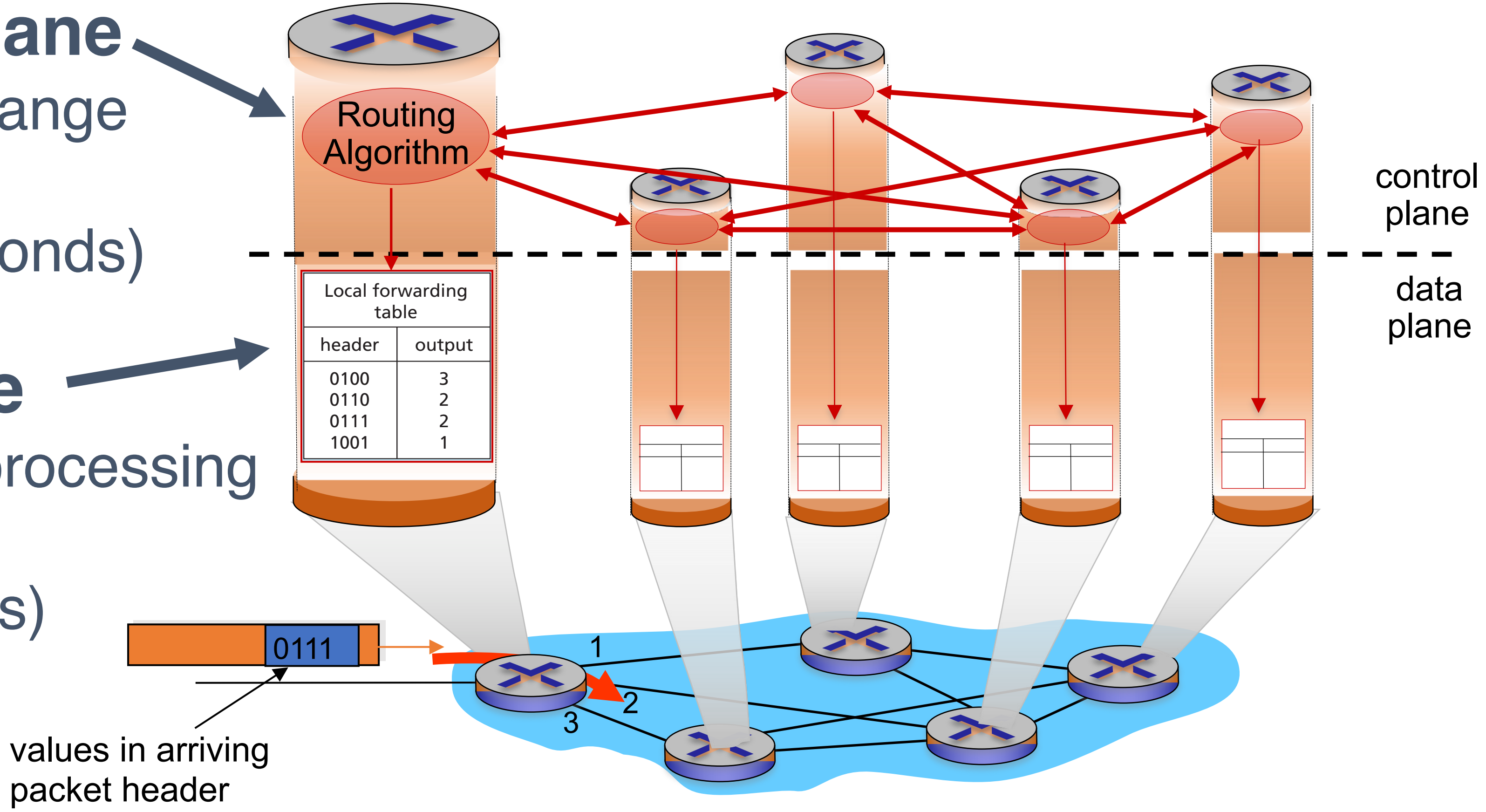
Control & Data Planes inside a router

Control plane

per route-change processing (~ a few seconds)

Data plane

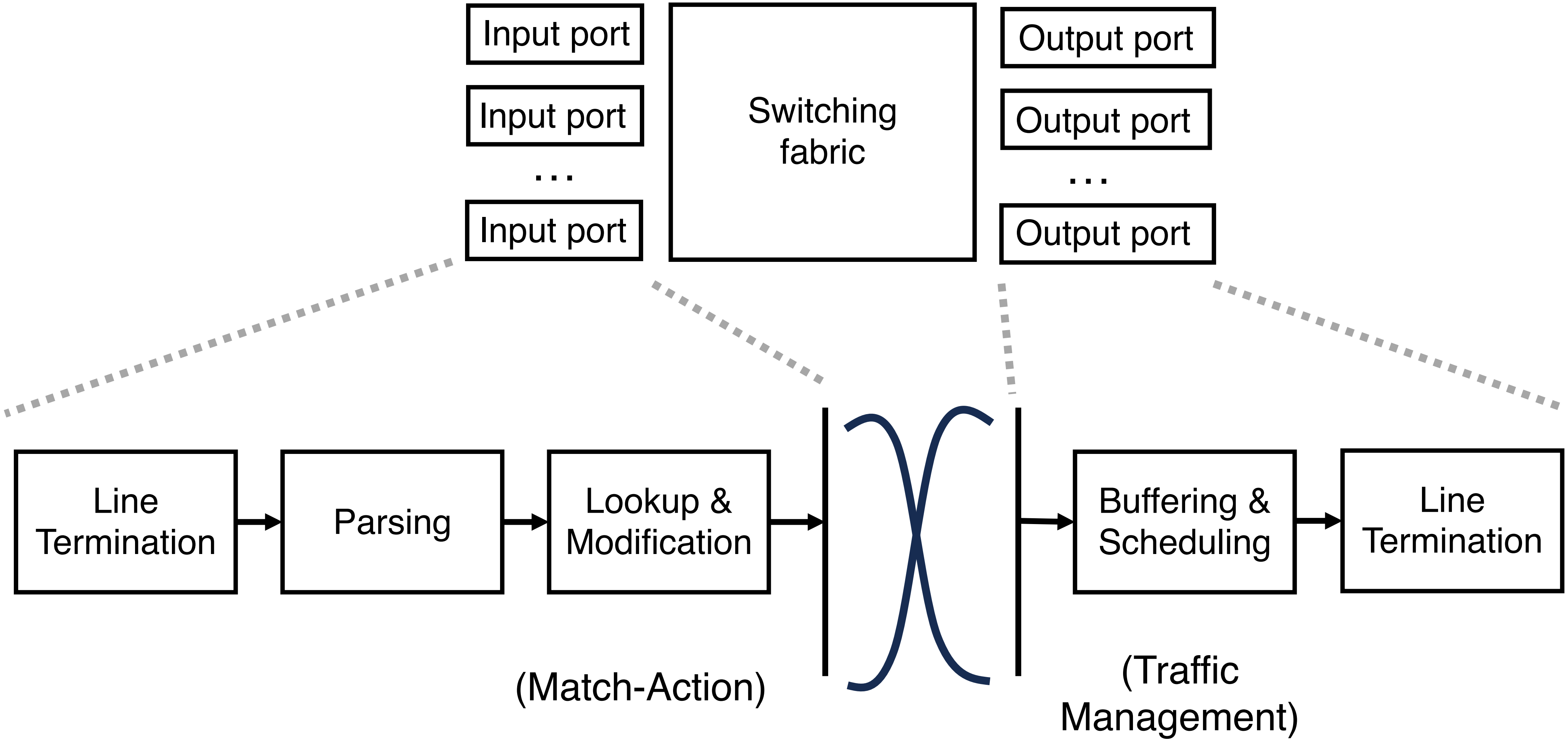
per-packet processing (~ tens of nanoseconds)



Local forwarding table

header	output
0100	3
0110	2
0111	2
1001	1

Hardware Router overview



Study 2 designs

- Historically evolving, multiple concurrent router designs

- 2 exemplars:
  - MGR: router from the late 1990s (50Gbit/s router, Partridge et al)
  - RMT: router from the late 2010s

Life of a Packet

(2) Packet parsing

- Dividing a sentence into its grammatical parts:
  - "I ate an apple"
  - Sentence := Subject (I) Verb (ate) Object (an apple)
  - Object:= Article (an) Noun (apple)
- Packet parsing: dividing sequence of bits into header fields
  - 1100100101011
  - ➔ ethernet destination (1100) | source (1001) | ethertype (01) | …
  - Unlike parsing English, parsing packets is quite mechanical
  - Series of extractions and branches to assign fields to packet bits
- Parsing could be implemented in software (MGR) or hardware (RMT)

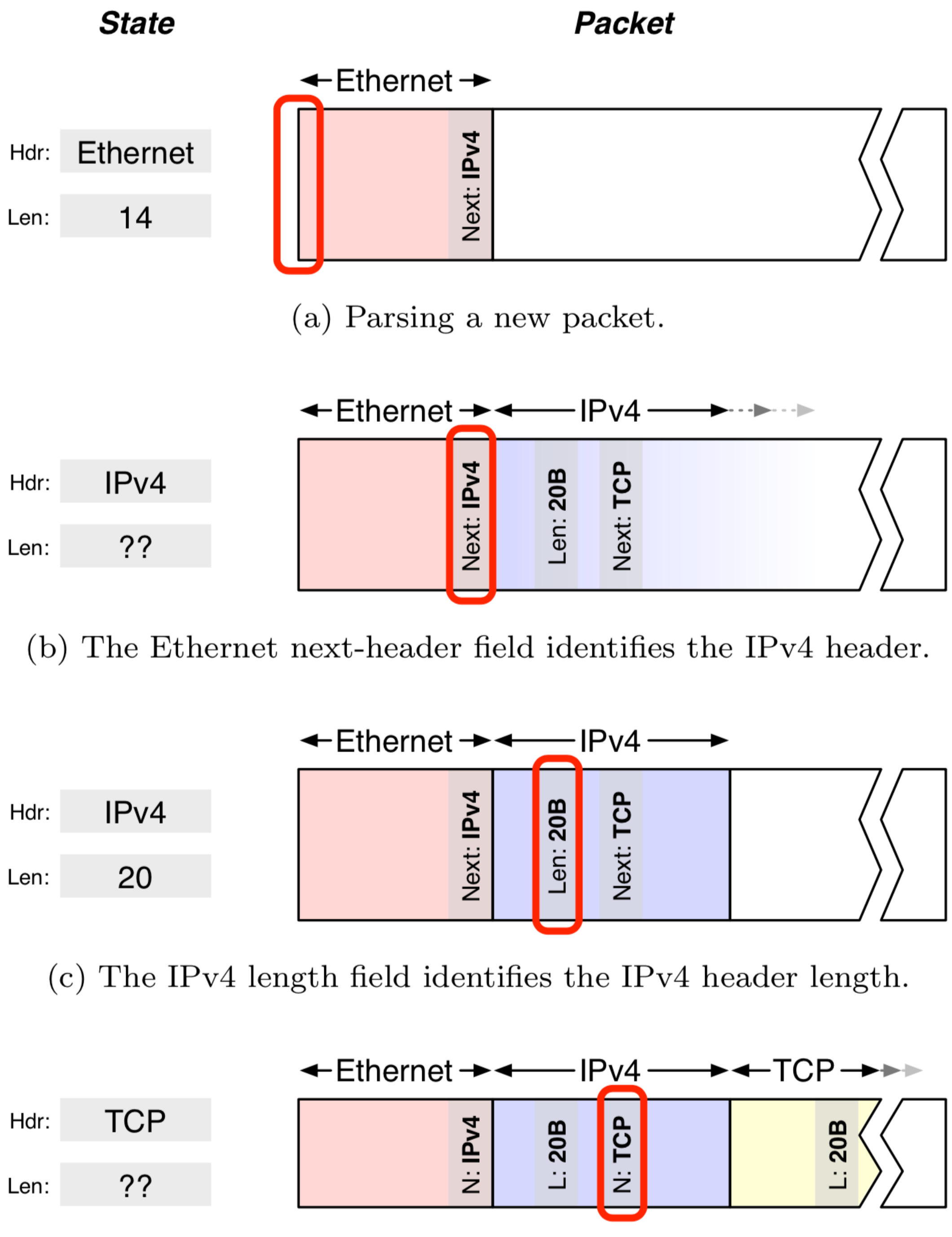
(2) Packet parsing

- A key principle: Separate header and payload data paths
  - Router functionality is “header-heavy” but “payload-light”
  - Don’t move the payload around too much
  - Conserve bandwidth & resources for data moved inside the router
- Header goes on as input to route lookup/packet modification
- Payload sits on a buffer until router knows what to do with pkt
  - Buffer could be on the ingress line card (MGR) or a buffer shared between line cards (RMT)

Parsing state machine

- Parsing is an inherently sequential process
- Previous header determines the next header type
- Current header length determines the start of the next header
- Parser state: tracks the current header and its length
    - Help jump to the next state

Source: Design principles for packet parsers, Gibb et al.



(a) Parsing a new packet.

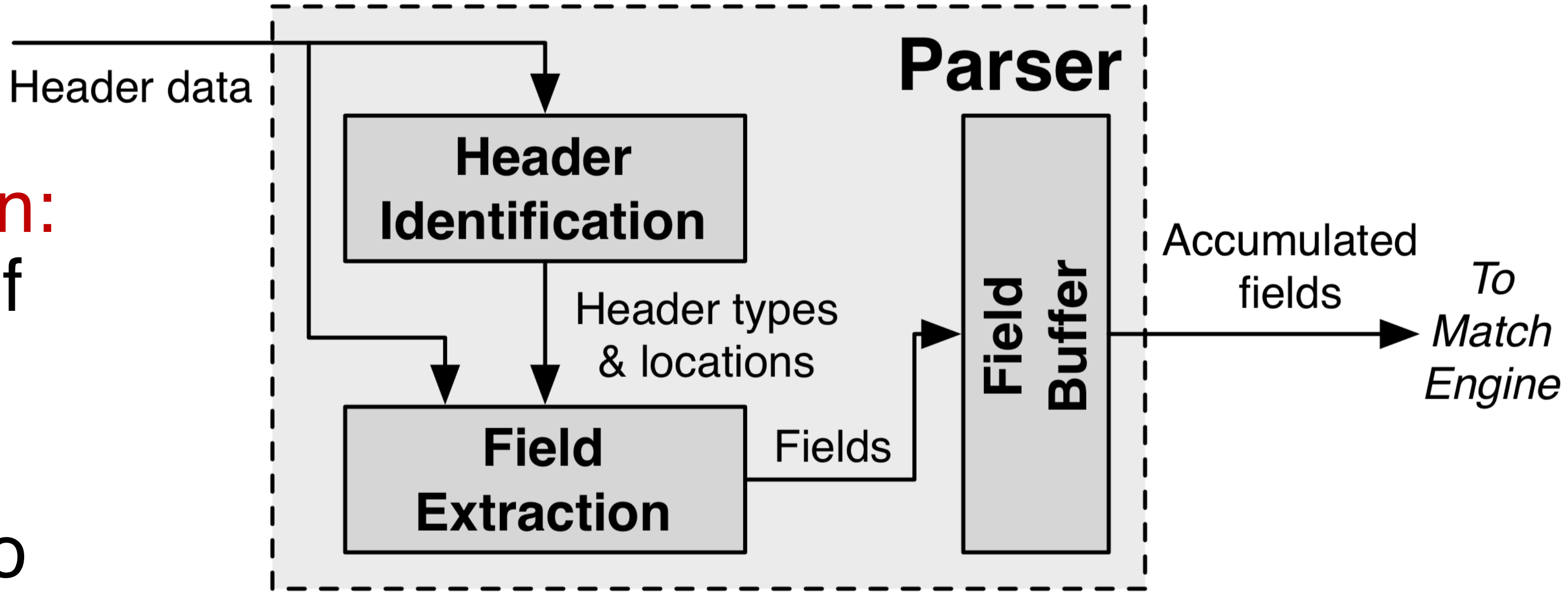
(b) The Ethernet next-header field identifies the IPv4 header.

(c) The IPv4 length field identifies the IPv4 header length.

(d) The IPv4 next-header field identifies the TCP header.

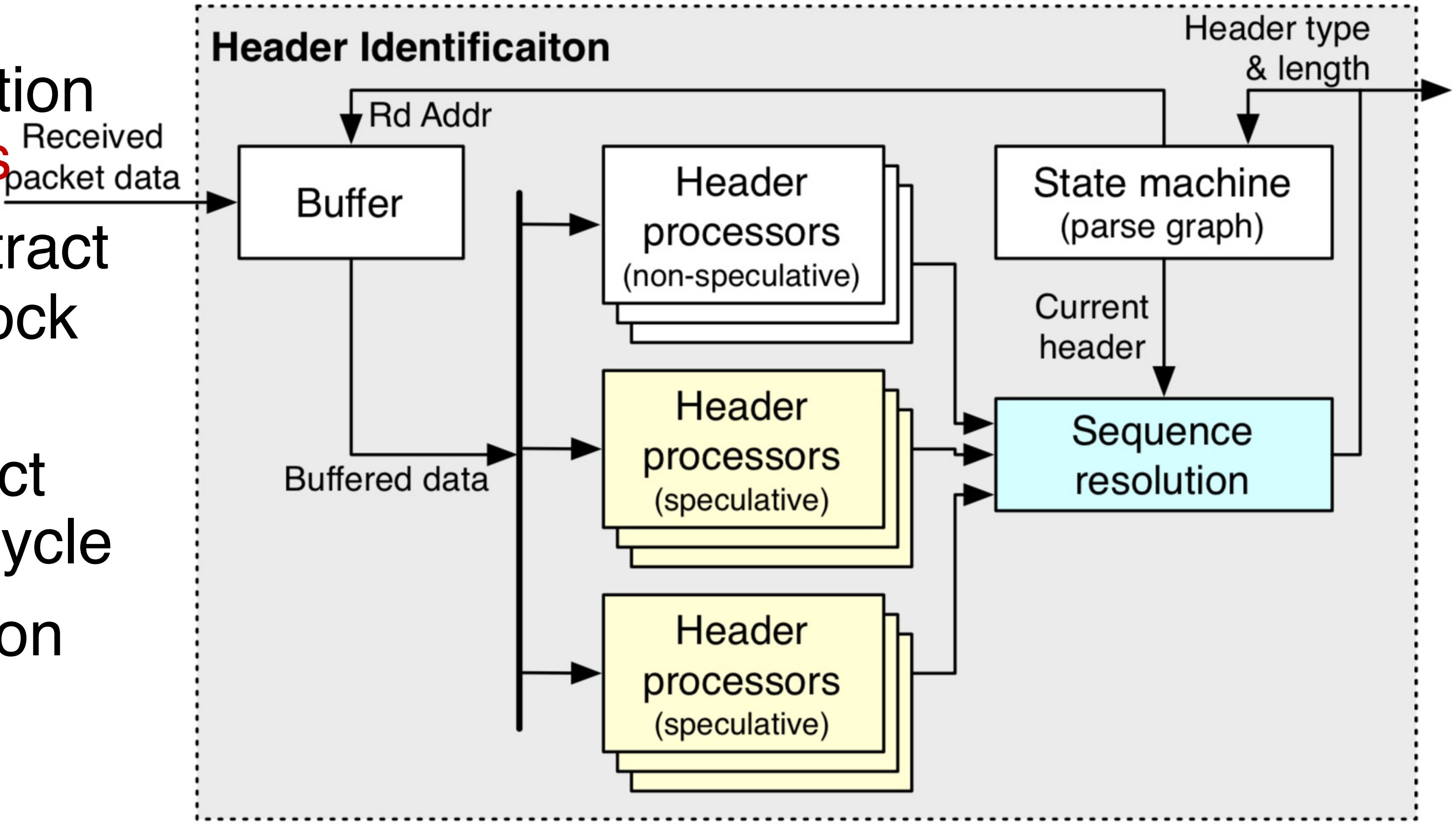
A hardware (fixed function) parser

- Two steps:
- Header identification: Identify sequence of headers
- Extract fields from identified headers to send to matching components of tables
- Design digital circuits with a high clock rate?



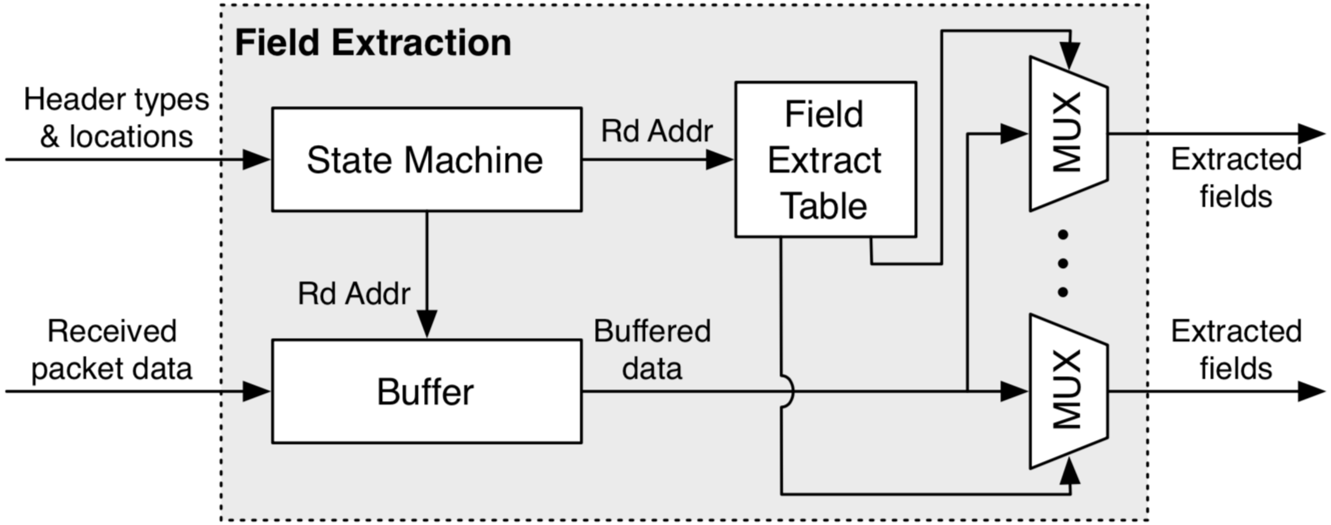
(Fixed function) header identification

- Identify headers through fixed-function header processors
- Simple design: extract one header per clock cycle
- Speculate to extract multiple headers/cycle
- Sequence resolution picks one


Header Identificaiton
Received packet data
Buffer
Rd Addr
Buffered data
Header processors (non-speculative)
Header processors (speculative)
Header processors (speculative)
State machine (parse graph)
Current header
Sequence resolution
Header type & length

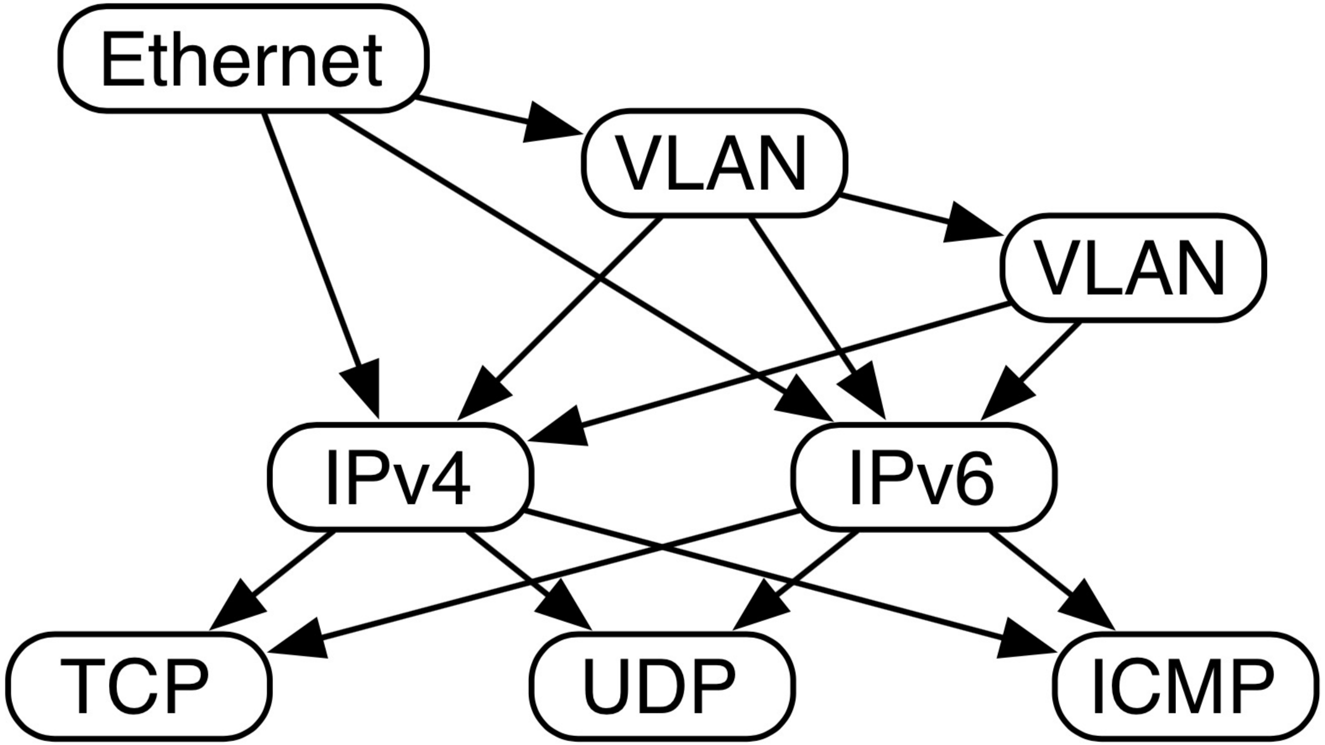
(Fixed function) field extraction

- Extract fields using fixed offsets into the packet, depending on parser state
- Field-extraction table is hard-coded for specific fields and protocols
  - E.g., IPv4 length: always at bytes 3 & 4



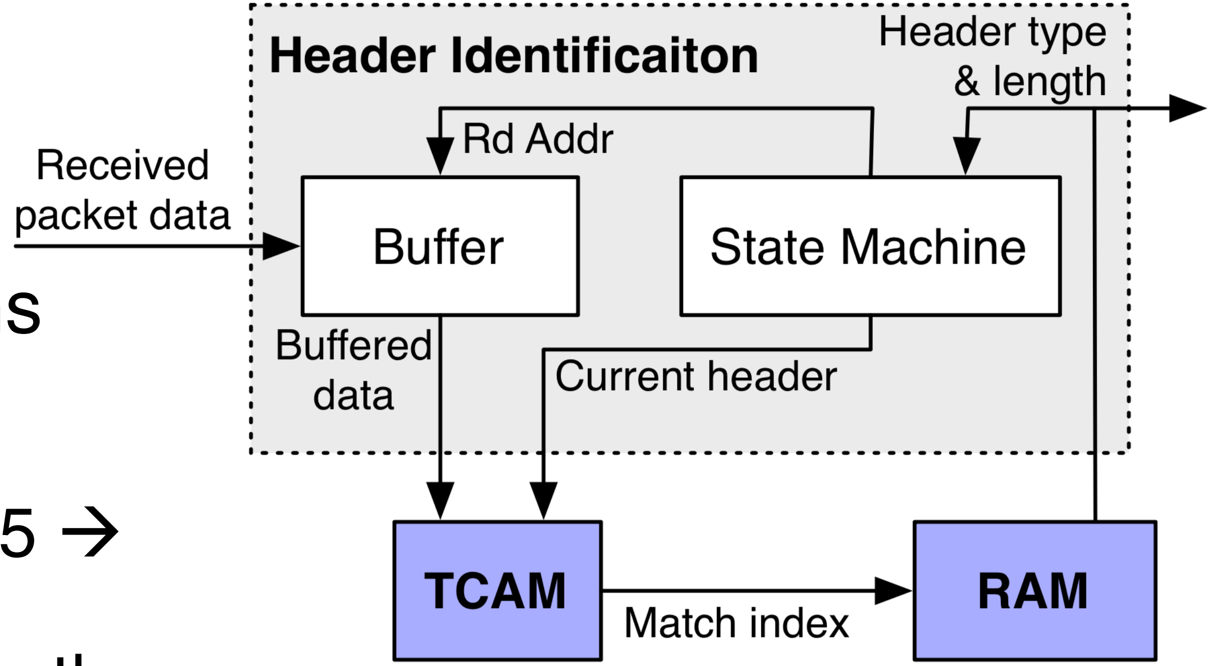
RMT makes parsing programmable

- Parse graph: Representation of all valid header sequences you expect



Programmable parser

- Reprogrammable TCAM allows us to identify headers based on the protocol parse graph
- Reprogrammable SRAM contains the protocol-dependent field extractions:
  - e.g., IPv4, byte 8 → TTL, 12—15 → IPv4 source address, etc.
  - Extracting the next header & length is a special case of generic field extractions for each protocol



Output of programmable parser

- Vector of extracted packet fields and labels. Example:
  - Ethernet -> src: 1010..; dst: 0101…; ethertype: IPv4
  - IPv4 -> version: …; …
  - … *<other headers>*

- Termed the packet header vector (PHV)

- Fed into the packet lookup and modification engines

(3) Packet Lookup & Modification

- The main job of a router is to forward packets to the correct output port(s)
- Typically done by looking up a table of entries that were pre-computed by the control plane
- Look-up tables with a range of sizes (# entries), widths (headers)
- Packets may also need to be modified
  - RFC 1812 TTL decrement, recompute IP checksum, MAC rewrite, …
  - Virtualized public cloud: encap/decap headers
- Outcome: a (set of) output ports + (possibly modified) headers

Typical table structures

- Sequence of tables: L2, L3, ACL
- Ethernet (L2) headers to forward packets within one IP network
- IP (L3) headers to forward packets across IP networks
- Access control lists (ACL) to implement firewalls & other policies
- Different kinds of lookups possible:
  - Exact match
  - Longest prefix match
  - Wildcard match

Packet lookup in MGR

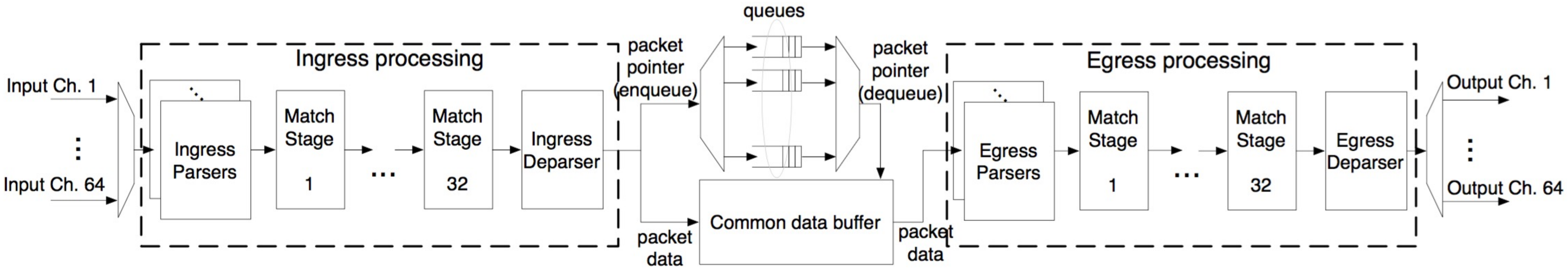
- A forwarding engine card separate from line cards
  - Scale forwarding and interface capacity separately
- Software: Use Alpha 21164 (a 415MHz generic processor)
- Programmed in assembly to do route lookup and other processing.
- Many optimizations to improve performance
  - Need for such optimizations continues today for software

MGR Software lookup performance

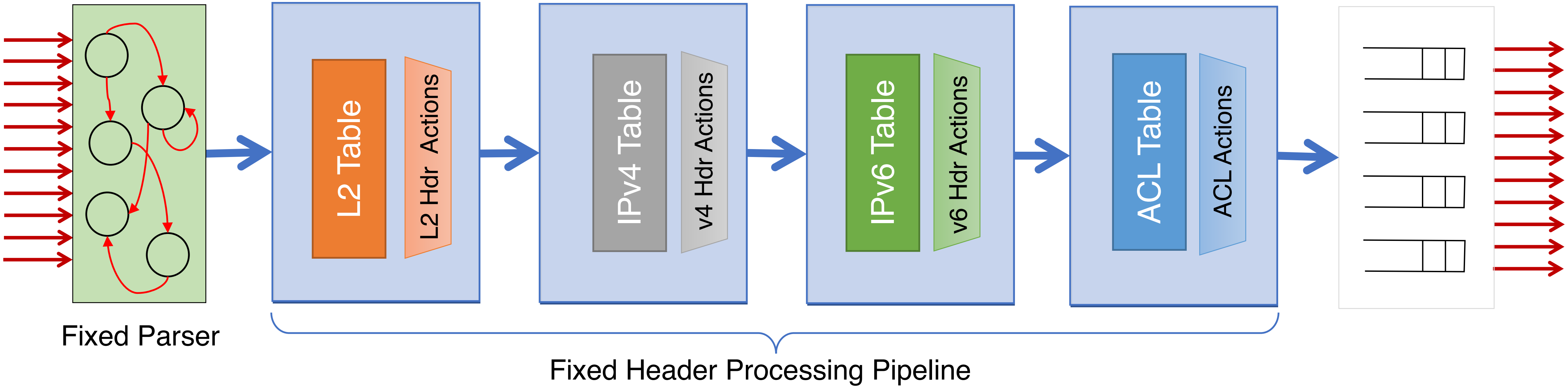
- Separate fast path from slow path (optimize the common case)
    - ARP lookup, fragmentation, error handling
- Try to fit all code into the processor instruction cache
- Heavily use caching for table entries across different memories in a hierarchy
    - Traffic locality: a small fast memory can service “most” traffic
- Two copies of table in external memory to support downtime-less updates to the forwarding table
- However, can’t guarantee deterministic throughput for packets
    - Packet might access slower memories in the memory hierarchy

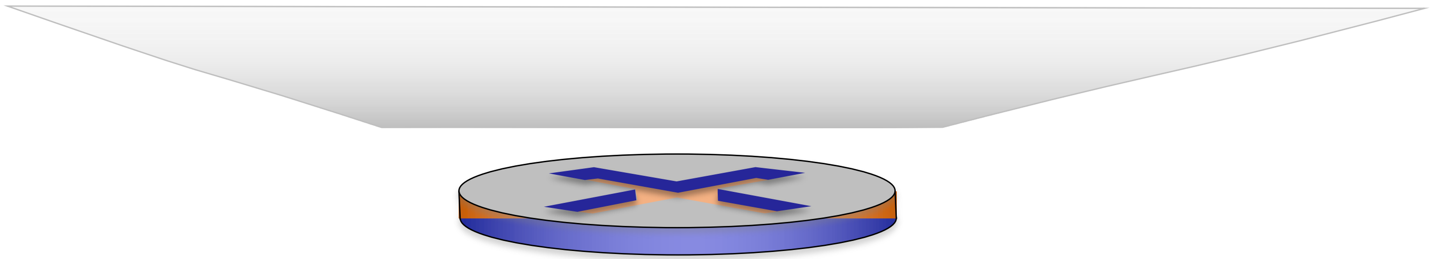
Packet lookup in RMT: *Pipelined parallelism*

- Different functionalities (ex: L2, L3) in different table stages
- Highly parallel over packets: admit 1 packet/cycle
- Pipeline circuitry *clocked* at a high rate: ex: RMT@1 GHz
- Deterministic throughput



Traditional pipelined hardware: fixed-function (Multiple Match-Action Tables)





MMT isn’t enough!

- Operators want new protocols and services
- Virtualization and the cloud
  - VMs have their own address space (e.g., 10.0.x.x)
  - Physical networks route traffic using a different set of addresses (e.g., 128.6.x.x)
  - Keep them separate so you can place VMs wherever you can
  - Need: dedicated new packet headers to use for forwarding within the “core” fabric of the network
    - E.g., VXLAN, NVGRE
- Research experimentation and domain-specific headers
  - E.g., finance; university campuses; network feedback signals

MMT isn’t enough!

- Want to use table memory flexibly
  - Different environments need tables of different shapes and sizes
- Enterprises: ACL-heavy (“students can see information sent to other students from professors, but cannot see info from professors to printers”)
- Tier-1 ISP like ATT: L3-heavy (~1 million Internet IPv4 prefixes)
- Static table sizes don’t work well
  - Can’t use another table’s memory, even if it is empty
  - Heterogenous devices to support different scenarios: complexity
  - Even device for a specific market may have insufficient resources

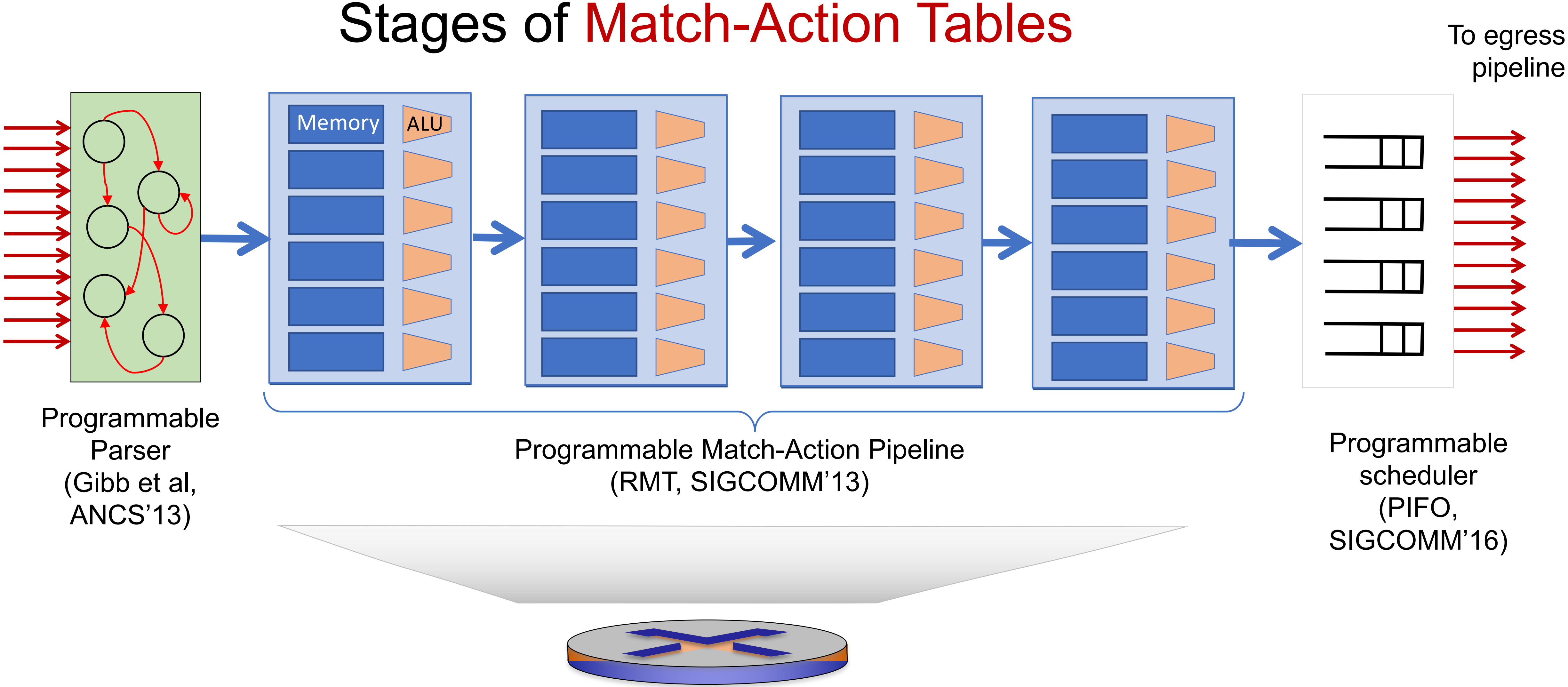
MMT isn’t enough!

- All of this might be supported if switch hardware can be upgraded as often as software in general (e.g., on smartphone)

- Unfortunately, the reality is very far from it:
- Each device generation hardware upgrade: 3--5 years
  - New ASIC design, verification, fabrication, testing
- Even software upgrades take time:
  - Features requested by other customers stand in the way of releasing new feature that one customer wants

RMT: Protocol Independent Switch Arch

Stages of Match-Action Tables



Programmable Parser (Gibb et al, ANCS'13)

Programmable Match-Action Pipeline (RMT, SIGCOMM'13)

Programmable scheduler (PIFO, SIGCOMM'16)

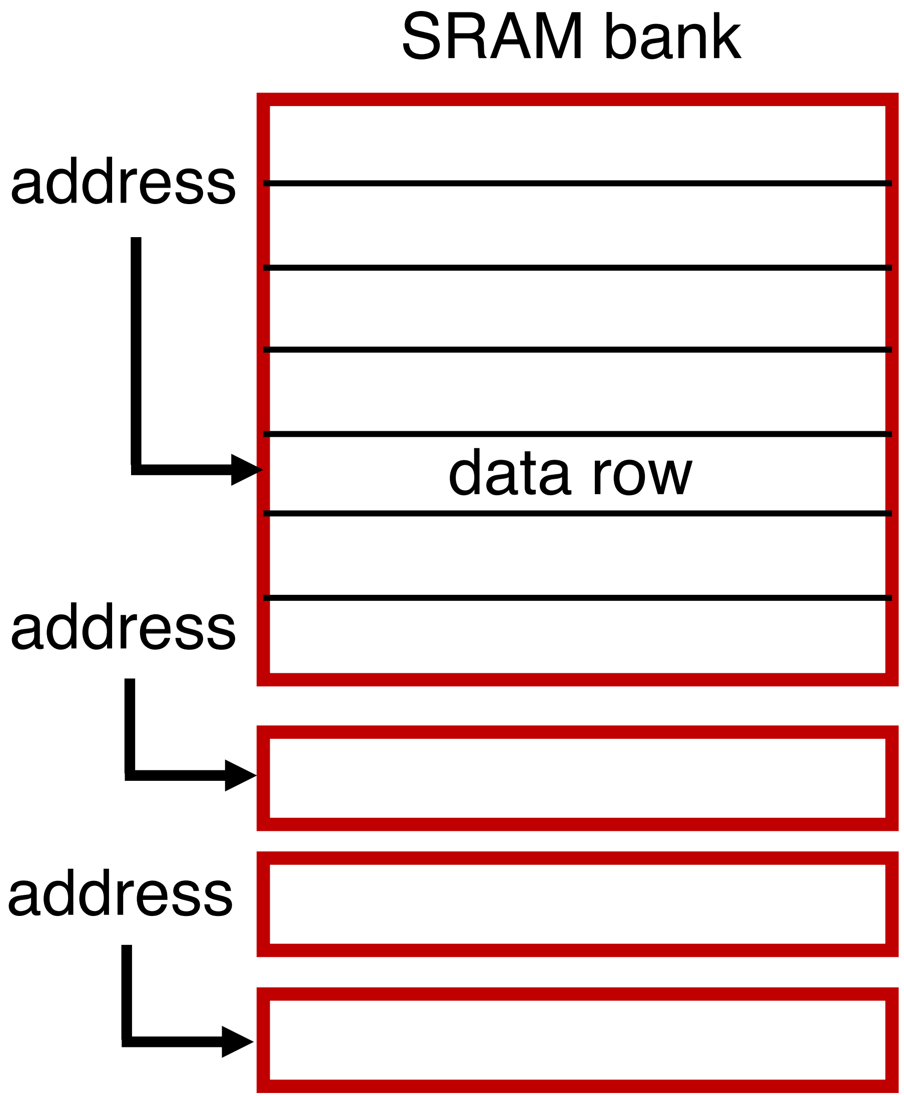
A primer on high-speed memories

- Computers have different kinds of memory
  - Fastest caches (L1, L2, …) made of SRAM
  - Fast main memory made of DRAM
  - Storage (HDD) made of magnetic disks, tape, SSDs, and so on
  - Translation Lookaside Buffer (TLB) with CAM/TCAM

- Today's high-speed routers use SRAM and TCAM
  - Static Random Access Memory for exact match
  - Ternary Content Addressable Memory for wildcard ACL match (also longest prefix match)

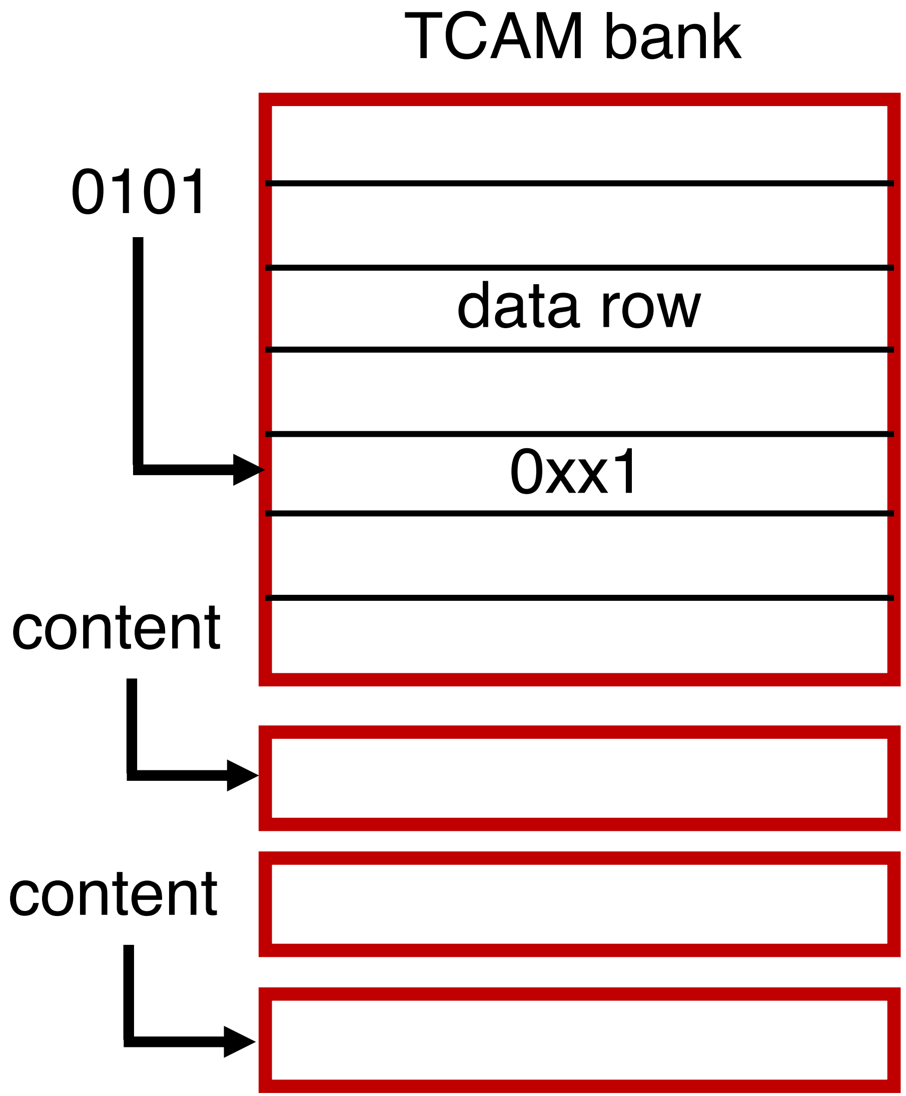
SRAM principles of operation

- Memory is organized into banks
- Each bank can be independently accessed through an address
- Data in the memory row at the address can be read/written each clock cycle
- Banks of larger sizes are denser (fewer wires to run), but you can only read/write one data row per bank per clock cycle (reduced parallelism)
- Looking up (ex: IPv4 dst) involves computing address(es) & reading mem



TCAM principles of operation

- Banks are accessed using content, not addresses (CAM)
- Contents of the memory are ternary: values can be 0, 1, or x (don’t care)
- Incoming keys are matched against TCAM, with any bit accepted at the location of the wildcard bits
- Ternary logic is power- and area-hungry relative to SRAM

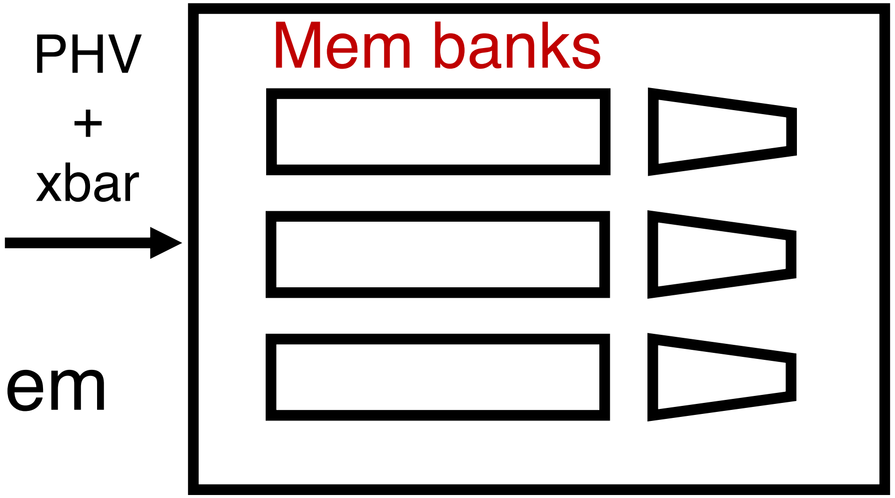


Match-Action Table memory design

- Match and Action units supplied with the Packet Header Vector
- Each stage accesses its own memory containing tables

- RMT uses a crossbar to pick PHV fields for matching against contents of SRAM/TCAM banks
  - Flexible key generation for lookup

- Distinction from fixed-function:
  - User-programmed fields stored in mem
  - Note: PHVs contain fields which are programmatically chosen

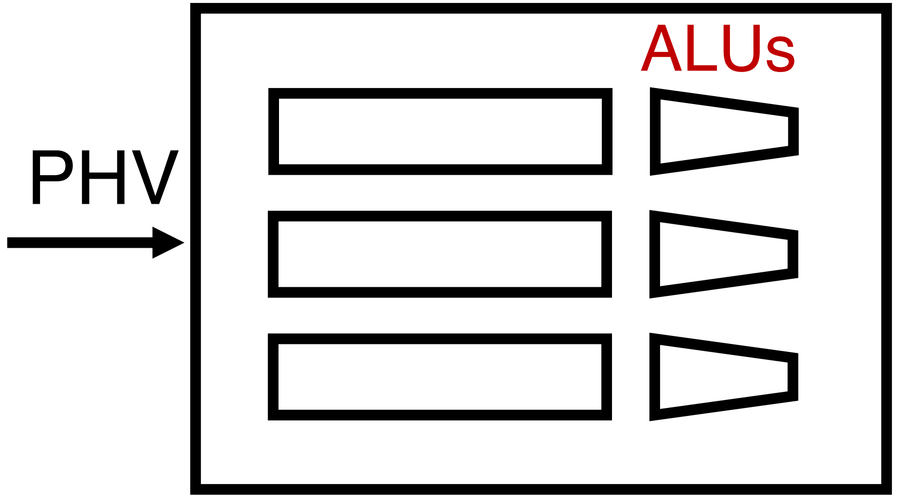


Match-Action Table memory design

- Entry looked up in the memory (SRAM/TCAM) contains a pointer to the instruction (action) for that entry

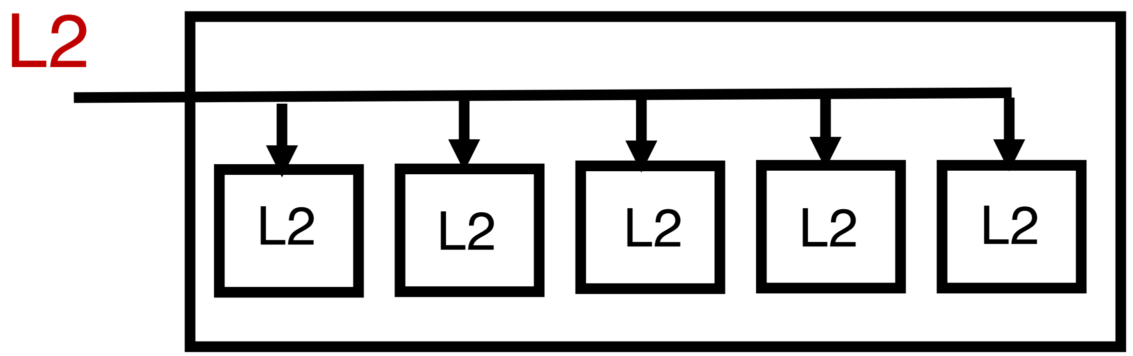
- Instructions implemented through programmable ALUs
  - More general ALUs than fixed-function hardware devices
- Feasible since compute components are extremely cheap
- VLIW: operate on multiple headers simultaneously

- Entry also contains pointers to:
  - Action data (e.g., port),
  - statistics (counters), if needed



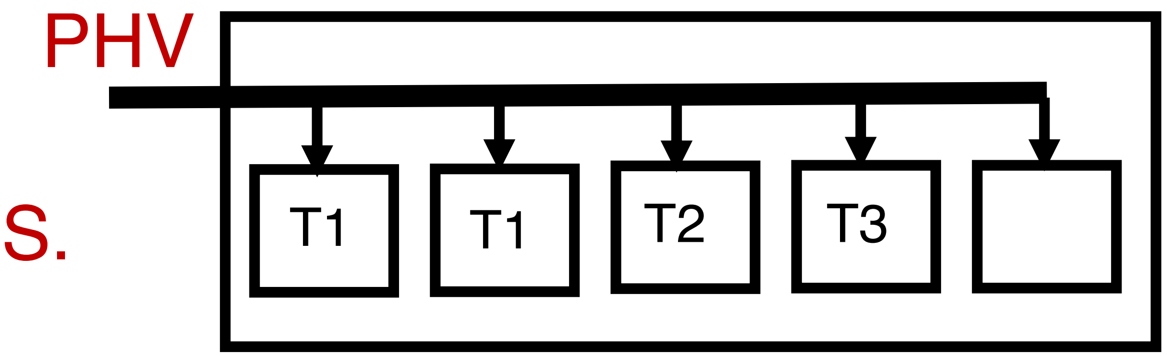
Achieving reconfigurable match-action

Separately configurable and addressable memory banks is the key to using tables flexibly



Fixed-function matching

VS.

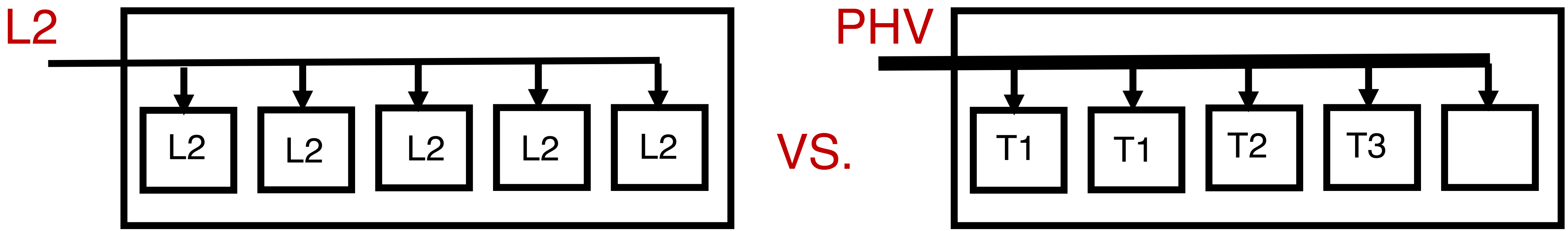


Flexible match function

Achieved by running extra wires (for read/write) to each memory bank + extra crossbars for multiple parallel tables

Achieving reconfigurable match-action

Separately configurable and addressable memory banks is the key to using tables flexibly



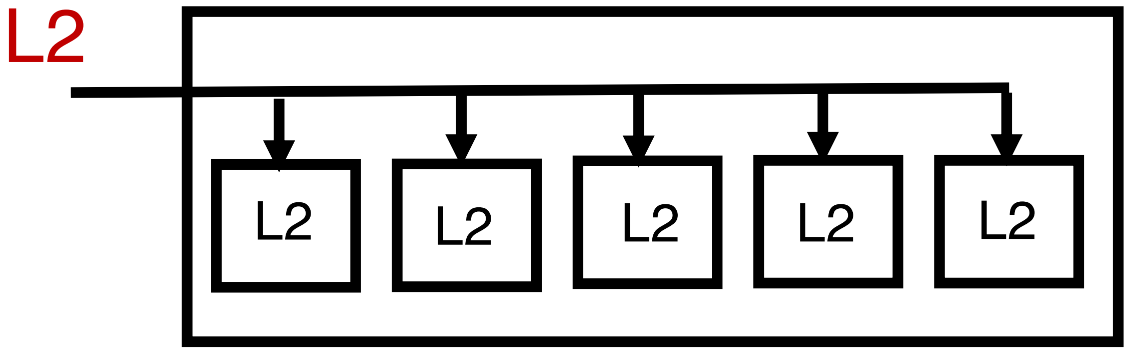
Fixed-function matching

Flexible match function

Different memory banks can contain entries for different tables (e.g., IPv4 matching, ACL, …)

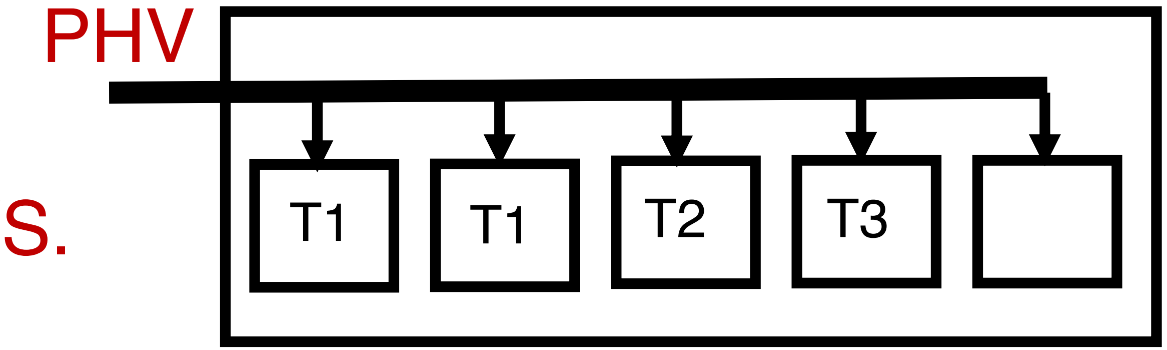
Achieving reconfigurable match-action

Separately configurable and addressable memory banks is the key to using tables flexibly



Fixed-function matching

VS.

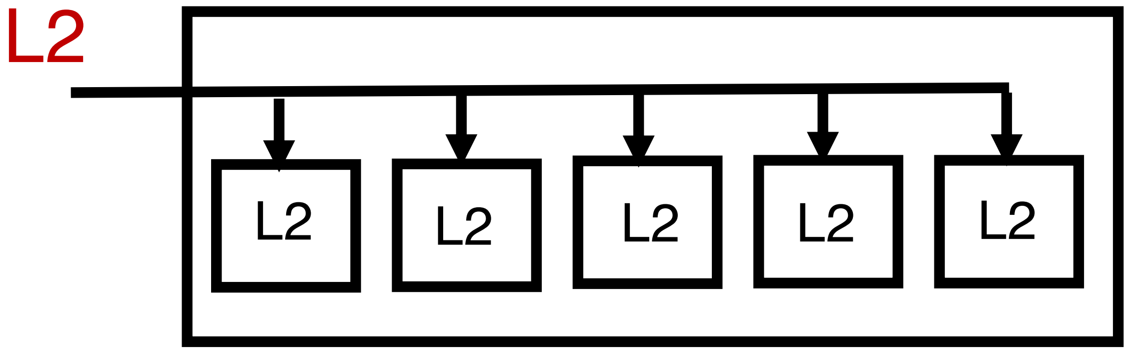


Flexible match function

Each table in a match-action stage may use a different crossbar configuration to extract a different set of fields from the PHV

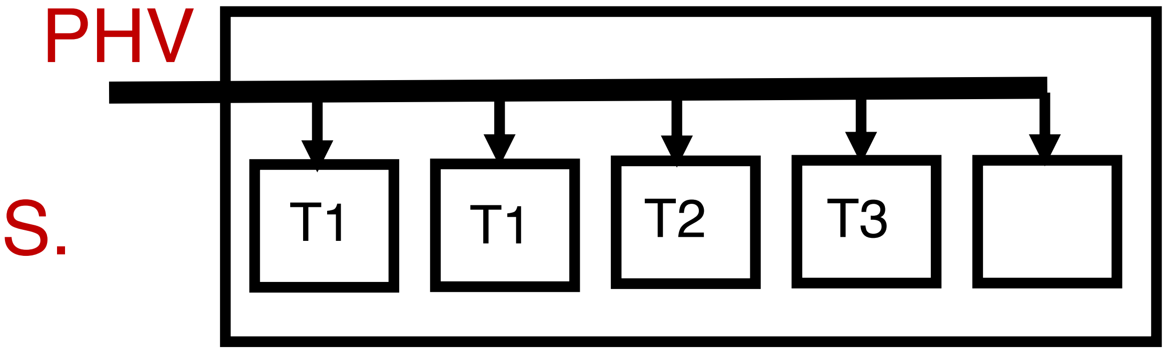
Achieving reconfigurable match-action

Separately configurable and addressable memory banks is the key to using tables flexibly



Fixed-function matching

VS.

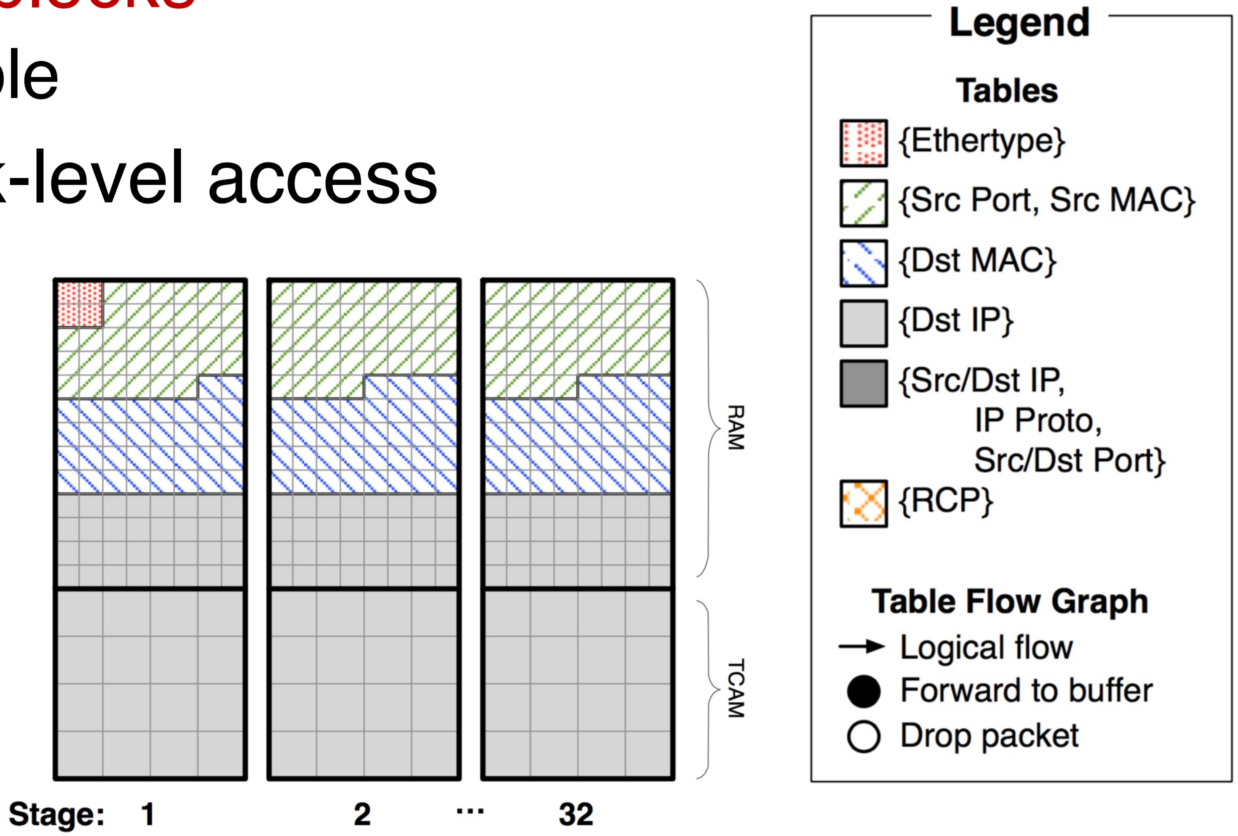


Flexible match function

The cost: 14% extra area (& power) for the fatter wires

Memory layout and use matters

- RMT: flexible partitioning of memory across SRAM and TCAM
- Numerous fixed size memory blocks
  - Memory fragmentation possible
- Circuitry for independent block-level access
- Deterministic access times
  - All of it is SRAM or TCAM
- Interesting compiler issues
  - “Packing” tables



Packet after route lookup

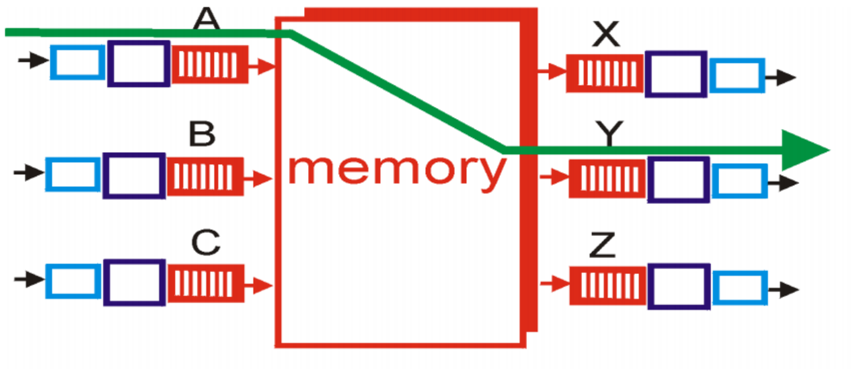
- What we have after lookup and modifications:

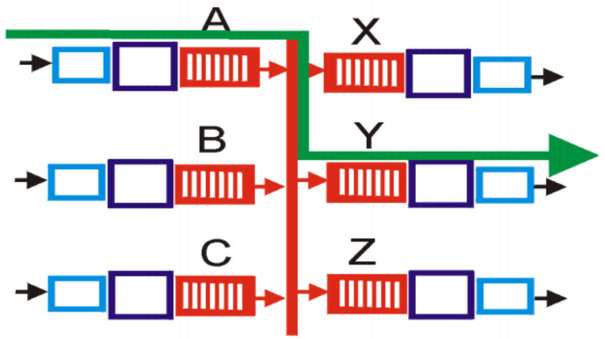
- One or more output ports, or a decision to drop

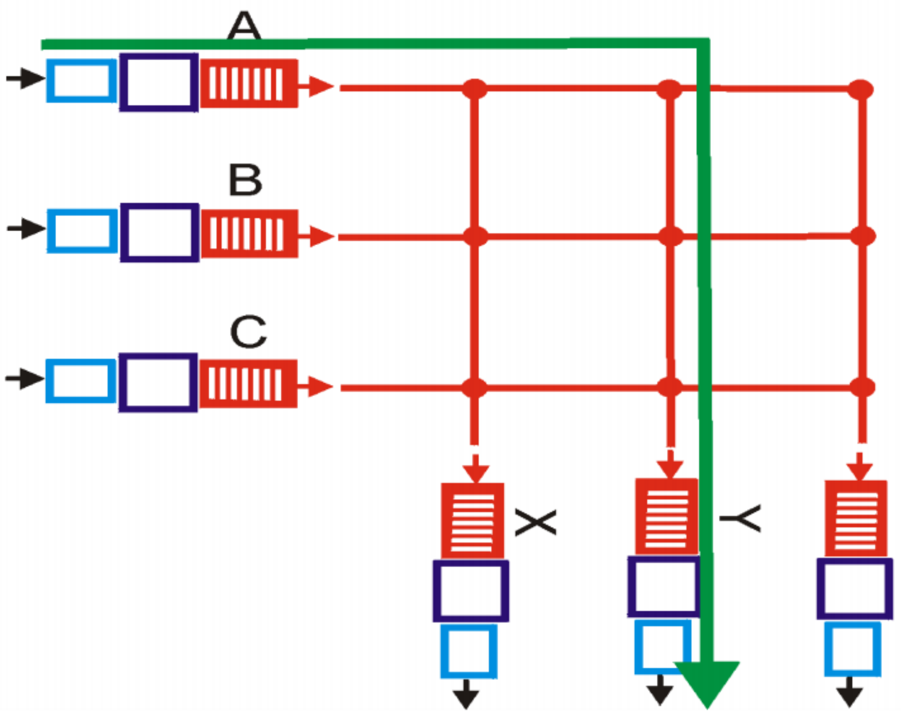
- packet headers, possibly modified from ingress

(4) Interconnect/Switching Fabric

- Move headers and packet from one interface to another
- Kinds of fabrics: memory, bus, crossbar



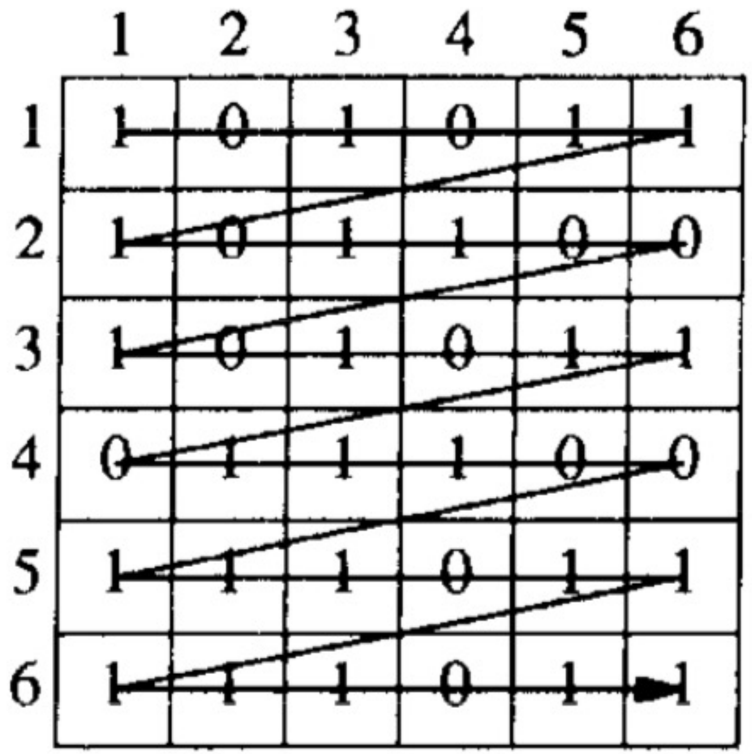




(4) Crossbars: The scheduling problem

- Want to utilize fabric capacity regardless of demand pattern
  - Crossbar is non-blocking

- MGR considers strategies to pair incoming demands & output ports
  - Greedy, wavefront, block wavefront
  - Need to address fairness across ports



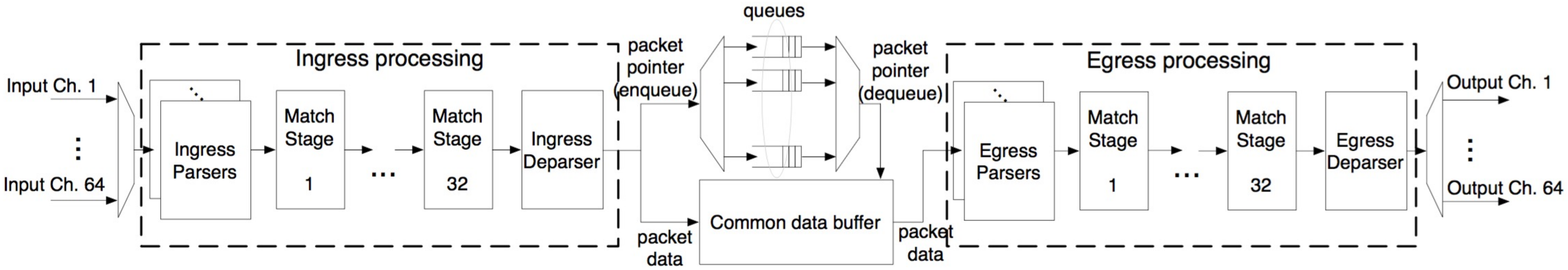
	1	2	3	4	5	6
1	1	0	1	0	1	1
2	1	0	1	1	0	0
3	1	0	1	0	1	1
4	0	1	1	1	0	0
5	1	1	1	0	1	1
6	1	1	1	0	1	1

(4) RMT switching fabric

- RMT uses memory as the fabric to hold packet headers and payloads between any two interfaces
- In the late 90s and early 2000s, there was considerable research on building high-speed packet buffers
- Today: shared memory switches & routers (shared ➔ across ports)
    - Fast memory can be clocked at 1 GHz
- Fundamental tradeoff: faster memories are not very dense
    - Can’t make the memory too large
    - Can’t hold too many packets!
- Workaround: use memory access patterns: e.g., each queue is FIFO
- Traffic manager implements scheduling & buffer management

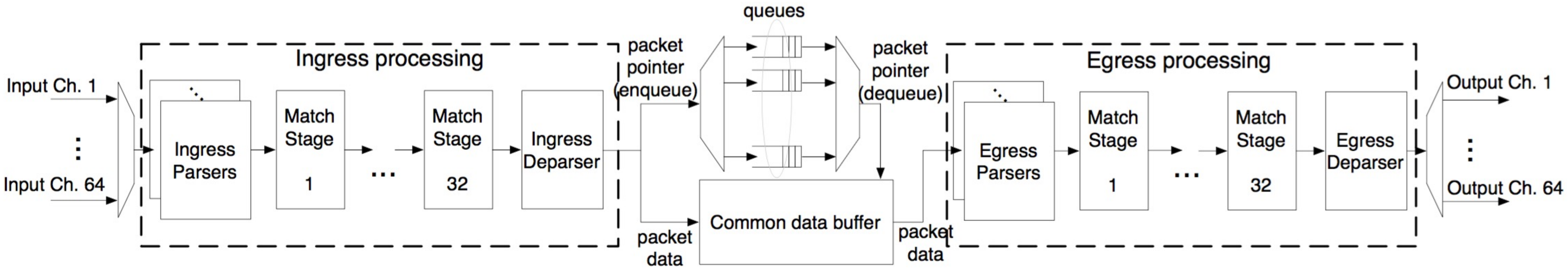
(5) Traffic Manager

- Where should the packets not currently serviced wait?
- Two designs: Input-queued vs. output-queued
- Output queueing avoids HOL blocking exhibited by input queueing.
    - Suppose port 1 wants to send to both 2 and 3 but port 2 busy
    - Packets from p1 towards p3 need not be delayed



(5) Traffic Manager

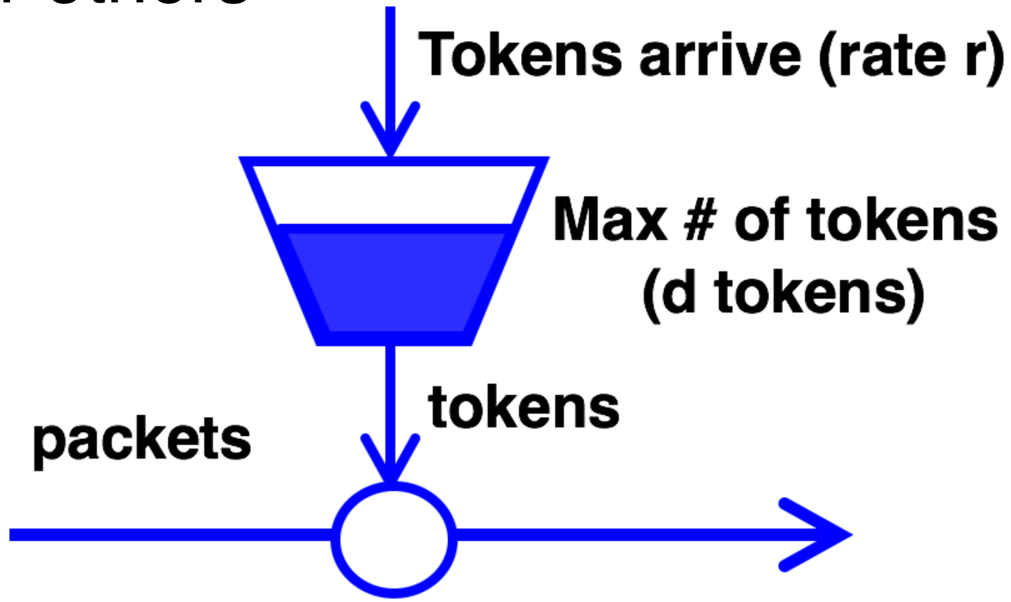
- RMT: Queueing represents output port contention
- A single output port can be represented by multiple queues
  - e.g., to implement weighted fair queueing
- Each queue is just a linked list in the shared memory
  - Maximum flexibility in queue sizes, but pointer overhead
  - Separate memory to maintain per-queue heads and tails



(5) Traffic Manager: Scheduling policies

- Q: how to dequeue packets from the buffer?

- Fair queueing across ports or flows
- Strict prioritization of some ports over others
- Rate limiting per port



(5) Traffic Manager: Buffer Management

- Q: how to enqueue packets into buffer?
  - If buffer is full, which packet should be dropped?

- Typical buffer management: Tail-drop

- Want fairness: if queue 1 has too many buffered pkts, don’t tail-drop q2
  - Share memory by partitioning (carving memory out) across queues

- And efficiency
  - if q1 has no pkts, q2 should be able to use (nearly) all buffer memory
  - Interesting space for dynamic buffer management algorithms
  - e.g., Dynamic queue length thresholds for shared-memory packet switches (Choudhury and Hahne)

(6) Egress line termination

- Combine headers with payload for transmission
    - Must incorporate effect of header modifications
    - Also called deparsing or serialization

- Multicast: egress-specific packet processing
    - Ex: different source MAC address for each output port

- Multicast makes almost everything inside the switch (interconnect, lookups, queueing) more complex

Summary

- Packet buffer memory is a precious resource
    - Modern routers have a few tens of MegaBytes buffer
    - … for all packets across all ports

- Traffic Manager: scheduling and buffer-management algorithms

- Buffer memory costs substantial amount: capital, area, power
    - Data centers: cheap routers with shallow buffers

Three kinds of data plane programmability

- Packet header formats, i.e., the packet parser
  - Example: Go from IPv4 -> IPv6
  - Custom packet format to carry financial info at high speed on a point-to-point link

- Table formats, actions, sizes, i.e., the match-action tables
  - Change which fields in the packet can be processed by a table
  - Control the table sizes, i.e., # entries, and hence the memory resource footprint according to use case.

Three kinds of data plane programmability

- Packet scheduling, i.e., the traffic manager
  - Flexible classification of packets
  - Flexible assignment of ordering and timing of when packets are transmitted from an outgoing link

… distinct from control plane programmability

- The control plane must compute the packet-processing rules put into the memory on the router ASIC
  - Example: packet with IPv4 destination 10.0.0.1 must go out of port 4

- Data plane programmability refers to the flexibility in the allowed set of packet headers, tables, and actions themselves, not the actual rules.
  - Example: There is a table that matches on IPv4 destination addr whose action is to determine the output port