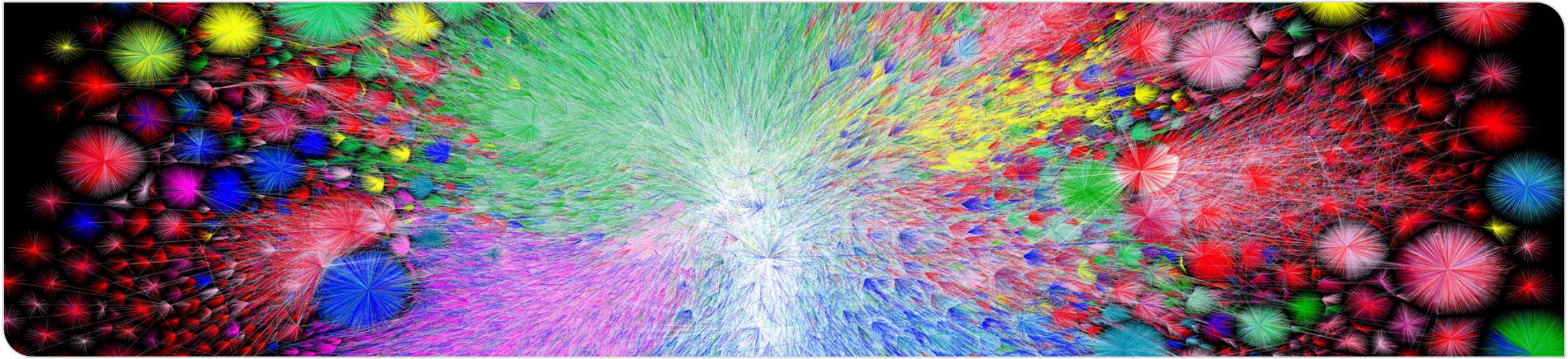
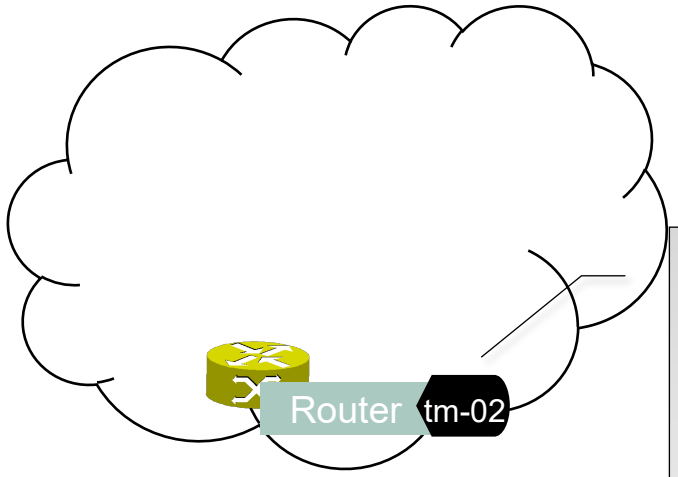
Karlsruher Institut für Technologie

Telematik

2. Router



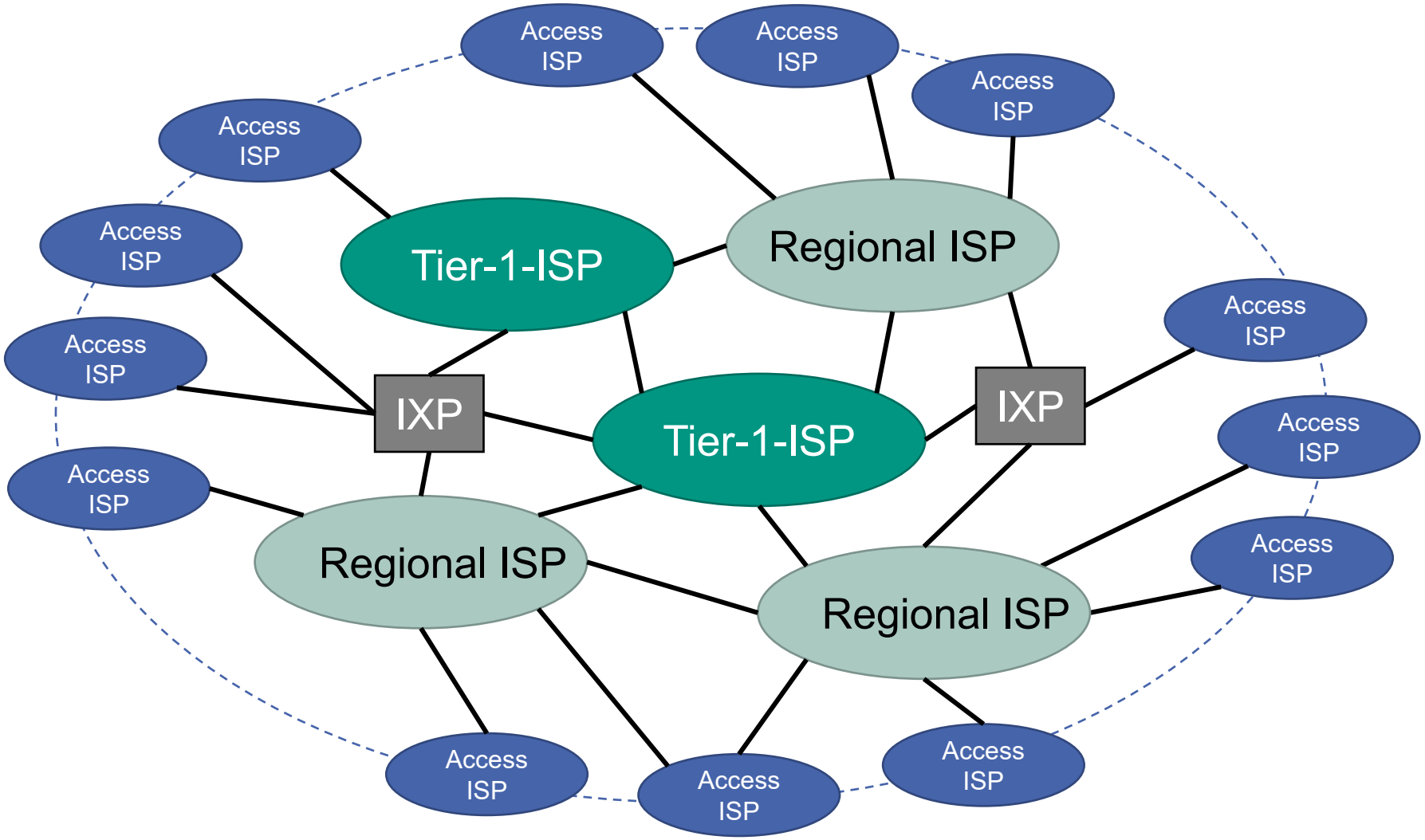


Data path
Router architecture

2 Router

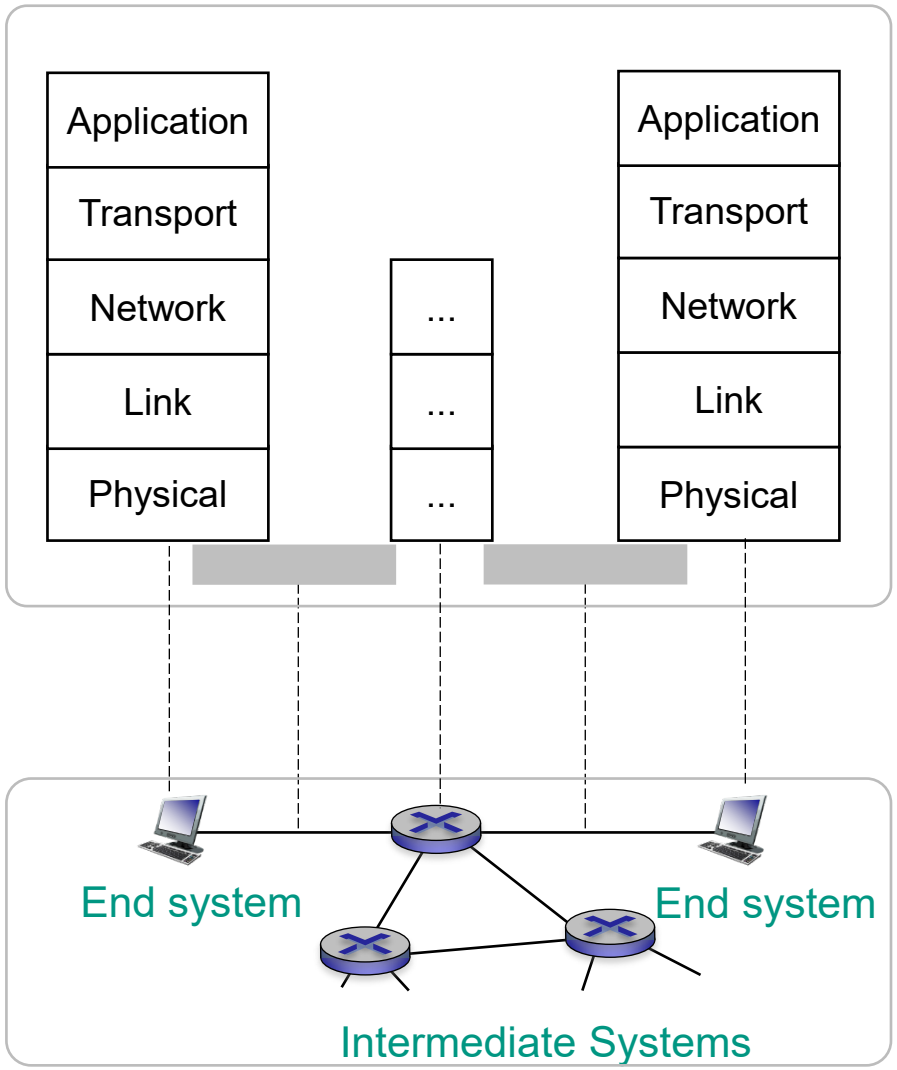
2.1 Basic Functionalities

Internet: Network of Networks



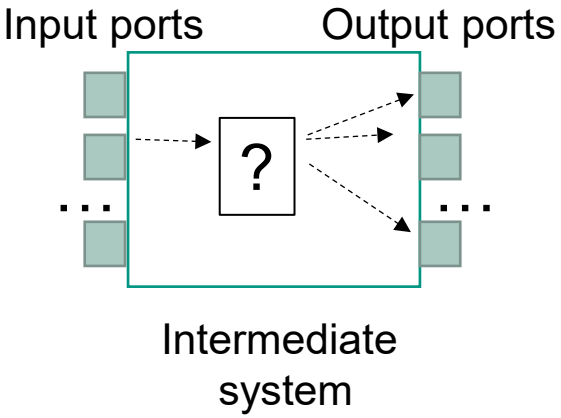
ISP: Internet Service Provider
IXP: Internet Exchange Point

End Systems and Intermediate Systems

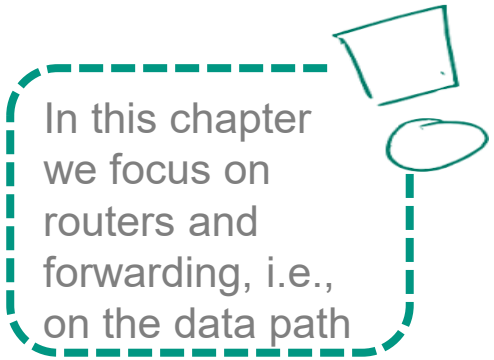


Intermediate Systems

- Forward data from input port(s) to output port(s)
  - Forwarding is a task of the data path
  - Basic scheme

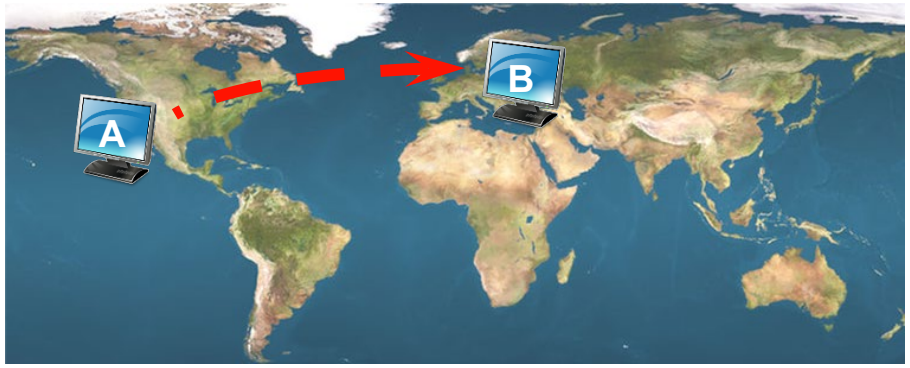


- May operate on different layers
  - Hubs operate on layer 1
  - Bridges operate on layer 2
  - Routers operate on layer 3



Routing

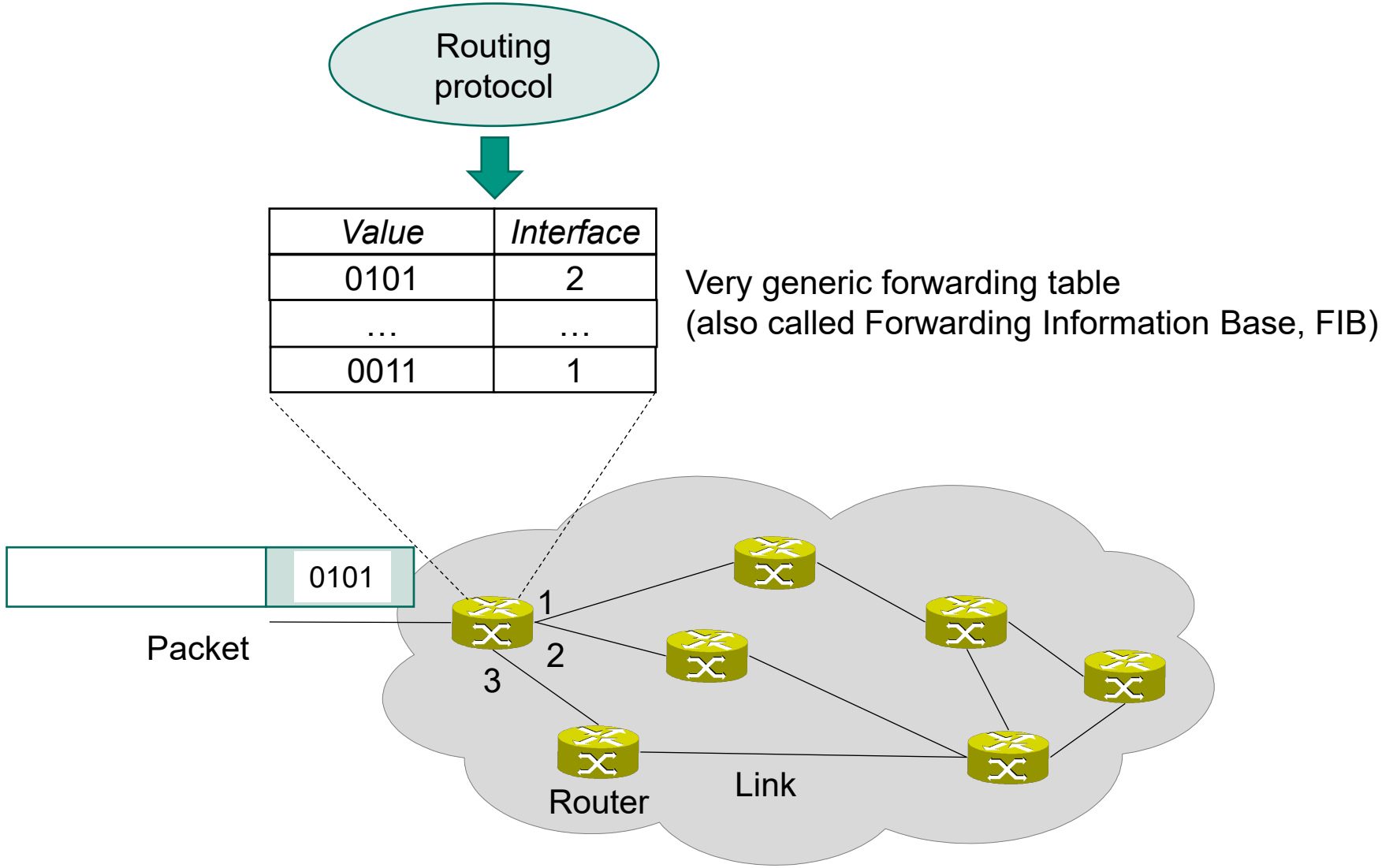
- Determines the path that the packets follow



- Routing is part of the control path
  - Requires routing algorithms and routing protocols
  - Examples of routing protocols
    - RIP, OSPF, BGP

Next chapter focuses on routing, i.e., on the control path

Recap: Basic Components



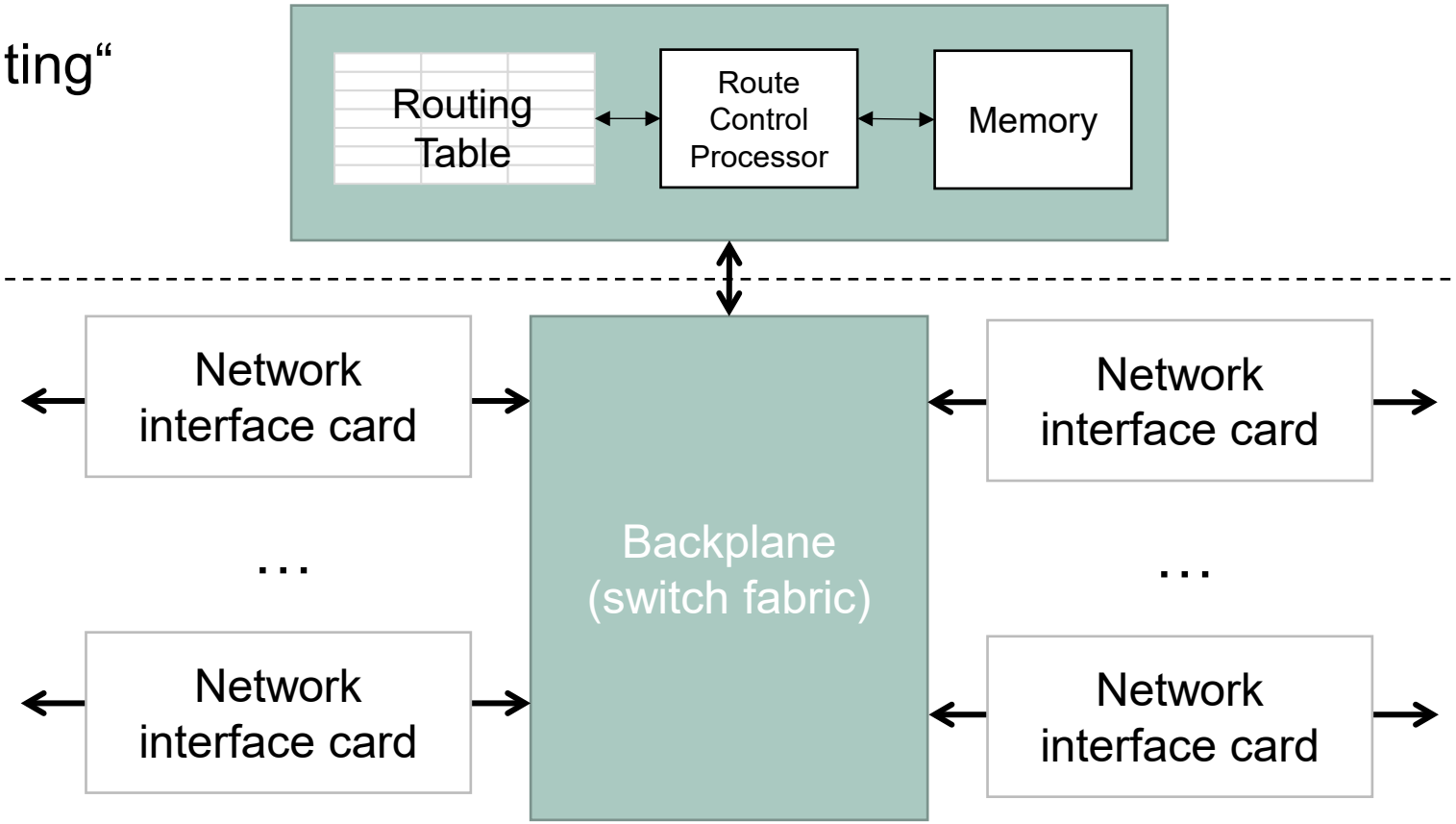
Forwarding within a Router

- Main task
  - Lookup in forwarding table
  - Forward data from input port to output port(s)

- Goals
  - High performance and low resource consumption
    - Forwarding in line speed (high data rate)
    - Short queues (low latencies)
    - Small tables (low resource consumption)

Generic Router Architecture

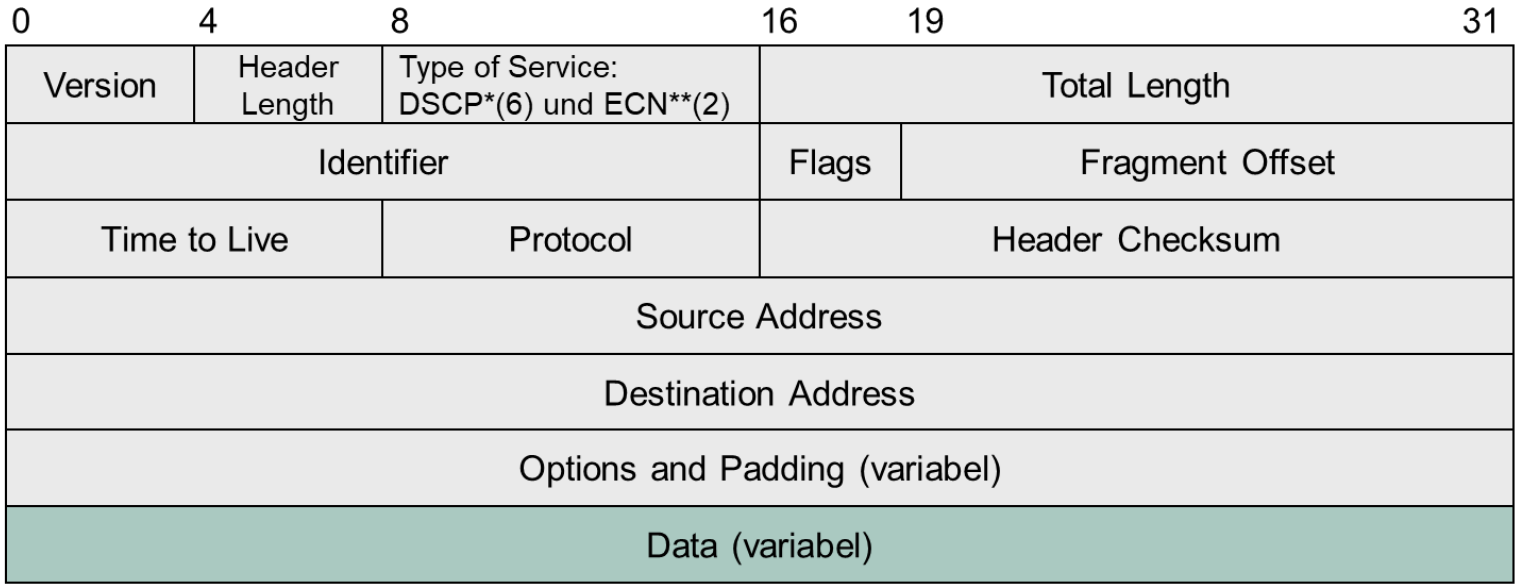
- Control path
  - … in chapter „Internet Routing“
- Data path
  - … focus of this chapter



NIC: network interface card

Forwarding Functionality

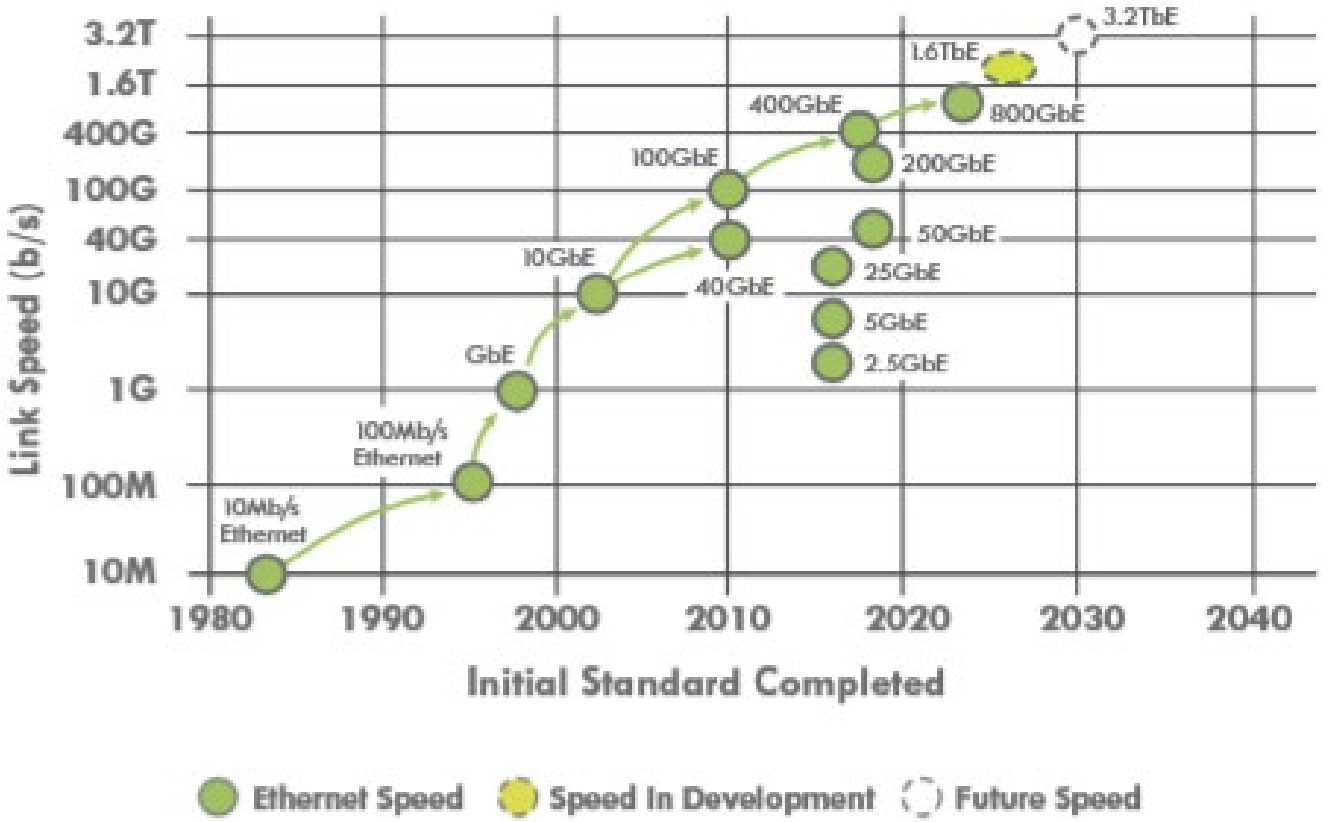
- Basic functions related to IP
  - Check header of IP packet
    - Version number
    - Valid header length
    - Valid checksum
  - Check time to live
    - Decrement TTL field by one
    - → recalculate checksum
  - Lookup
    - Determine output port for packet
  - Fragmentation
  - Handle IP options



- Additional functionality
  - Queueing
  - Classification
  - Prioritization
  - Traffic shaping

2.2 Challenge: Line Speed

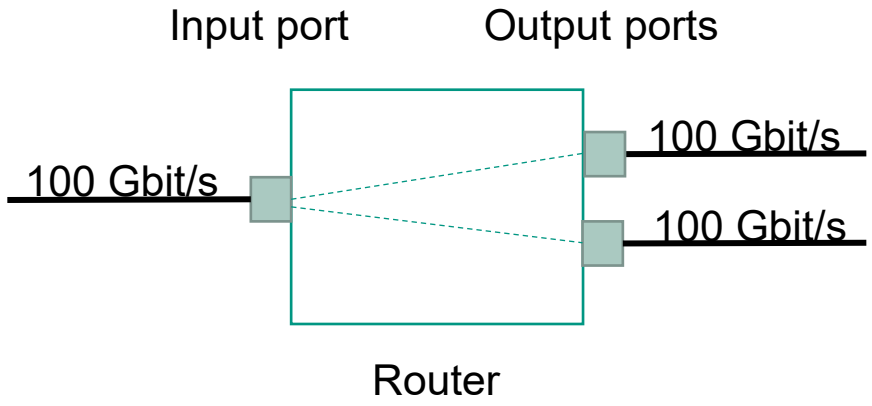
Growing Link Capacities



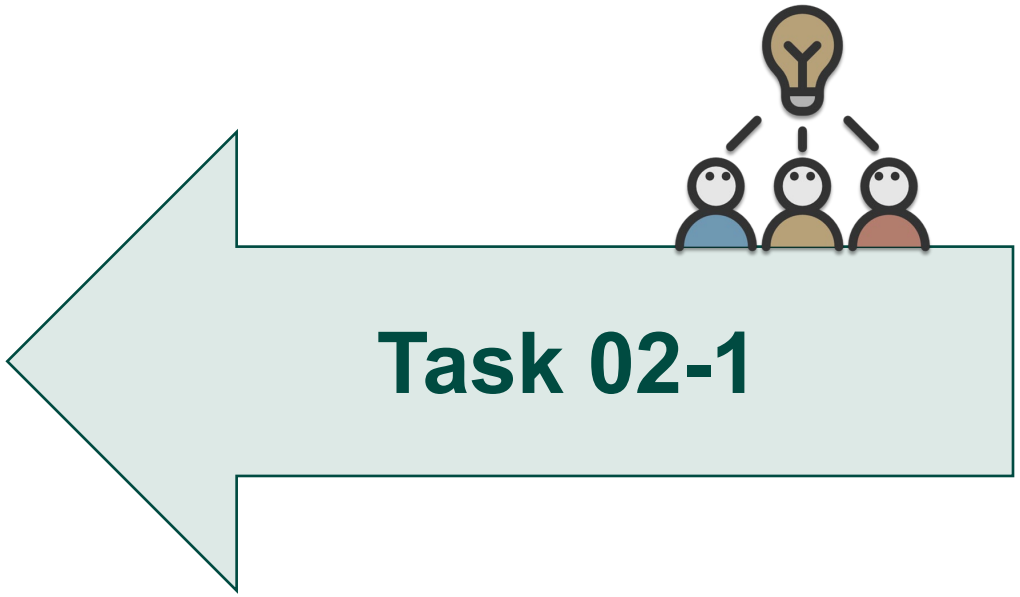
[https://ethernetalliance.org/the-2025-ethernet-roadmap/]

Challenge: Line Speed

- Router needs to keep up with line-speed




Task 02-1
Router
Forwarding
Performance


Task 02-1

Challenge: Line Speed

- Router needs to keep up with line-speed for all packet sizes

- Example
  - 1 Gbit/s line speed
  - 1500 byte packets

$$t_s = \frac{12000\ bit}{10^9 \frac{bit}{s}} = 12\ \mu s$$

- Time budget for forwarding task: ~ t_s
  - Smaller t_s for smaller packets, i.e., less time for forwarding decision
  - Forwarding task is overhead for every single packet!

Challenge: Line Speed

- Regarding TCP: two types of segments
  - TCP segments that transport data
    - E.g., 576 bytes (due to RFC 879, outdated)
    - E.g., 1500 bytes (due to Ethernet MTU)
  - TCP segments that are purely acknowledgements
    - → minimum-size packets (40 byte)
    - … this calculates to $t_s = 320\,ns$ in case of a line speed of $1\,Gbit/s$

$$t_s = \frac{40\,byte \cdot 8\,bit/byte}{10^9\,bit/s} = 320\,ns$$

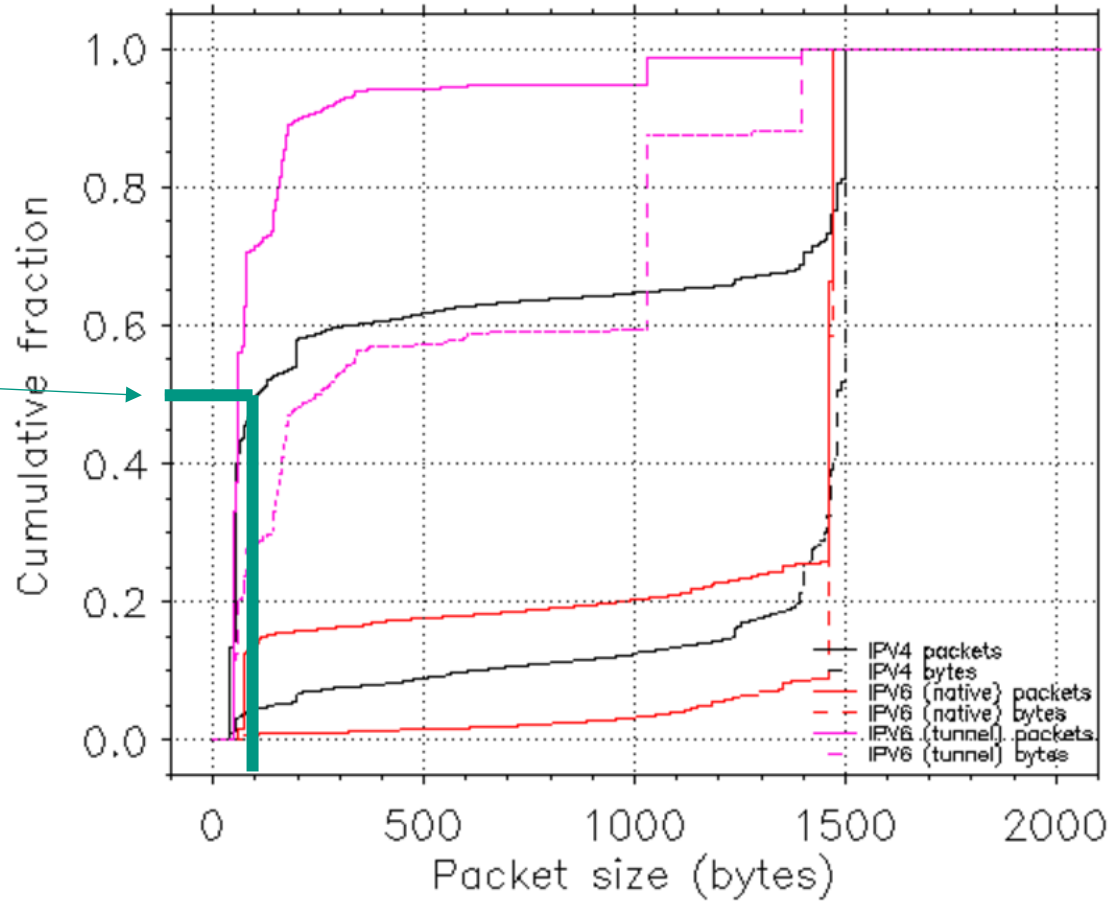
- For comparison
  - single cycle on common CPU: 0,5 ns
  - single value read from L1-Cache: ~1,5 ns

Line-speed	Time budget for 40 byte packets
1 Gbit/s	320 ns
10 Gbit/s	32 ns
40 Gbit/s	8 ns
100 Gbit/s	3,2 ns
400 Gbit/s	0,8 ns

MTU: Maximum Transfer Unit

Small Packets in the Internet?

- Example: Cumulative distribution function of packet sizes from a trace captured at the Internet Exchange Point equinix-chicago
- Trace from 2016-04-06
- 50% of the IPv4 packets are smaller than 100 byte
- Similar for other Internet traces!



Small packet sizes are quite common in the Internet

[Caida16]

Types of Routers

- Core router
  - Used by service providers
  - Need to handle large amounts of aggregated traffic
  - High speed and reliability essential
    - Fast lookup and forwarding needed
    - Redundancy to increase reliability (dual power supply …)
  - Cost secondary issue

- Enterprise router
  - Connect end systems in companies, universities …
  - Provide connectivity to large number of end systems
  - Support of VLANs, firewalls …
  - Low cost per port, large number of ports, ease of maintenance

- Edge router (access router)
  - At edge of service provider
  - Provide connectivity to customer from home, small businesses
  - Support for PPTP, IPsec, VPNs …

Example Router

- Cisco ASR 9922
  (Aggregation Services Router)
  - Edge/Core Router for carrier networks
    - 1 / 10 / 25 / 40 / 100 / 400 Gbit/s interfaces
  - Total backplane capacity: 160 Tbit/s
  - Up to…
    - 44 rack units, $350.000
    - 20 line cards, up to $1.417.000 each
    - E.g., 32x100 Gbit/s per line card

      … often substantial discounts for large carrier networks
  - Black-box router
    - Cisco IOS XR network operating system preinstalled
    - Software is proprietary and cannot be modified

Example Router

- UfiSpace S9610
  - Edge/Core Router for carrier networks
  - Backplane capacity: 14,4 Tbit/s per unit
  - E.g., data center line card with 36 x 400 Gbit/s Ethernet
  - Multiple cards can be interconnected
    - Scales to n x 36 x 400 Gbit/s
  - Price: 38.000€ for each line card

- White-box router
  - Software can be chosen and changed by end user
  - Open-source network operating systems available, e.g.:
  - Development of own software for new applications

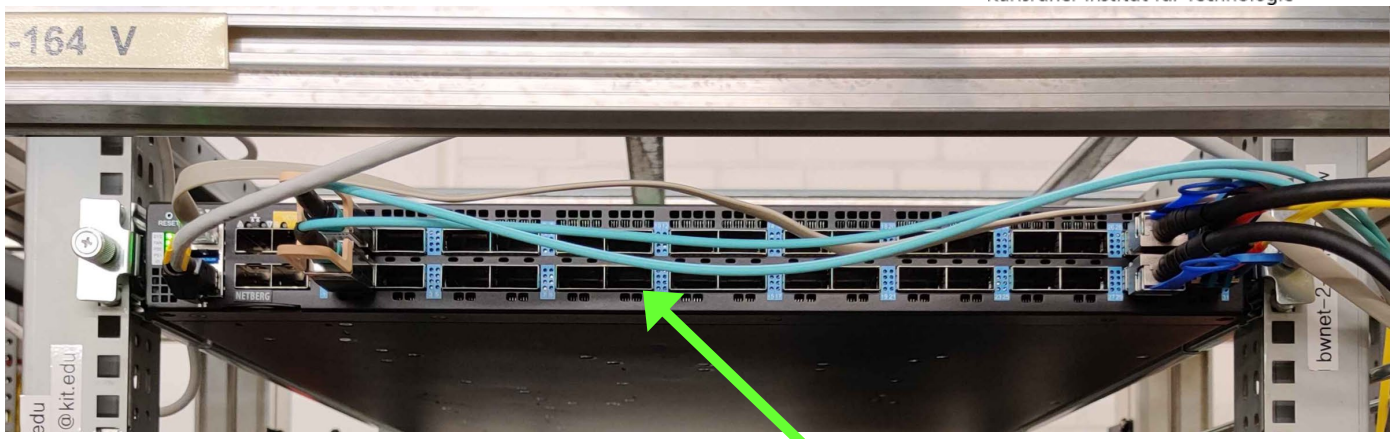
STRATUM

SONiC

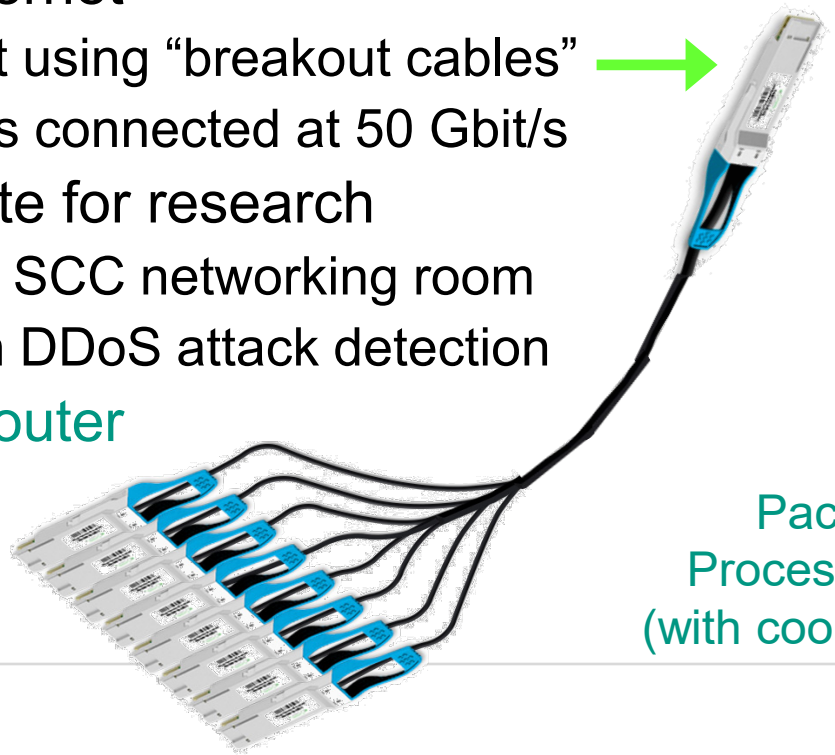
Open Network Linux

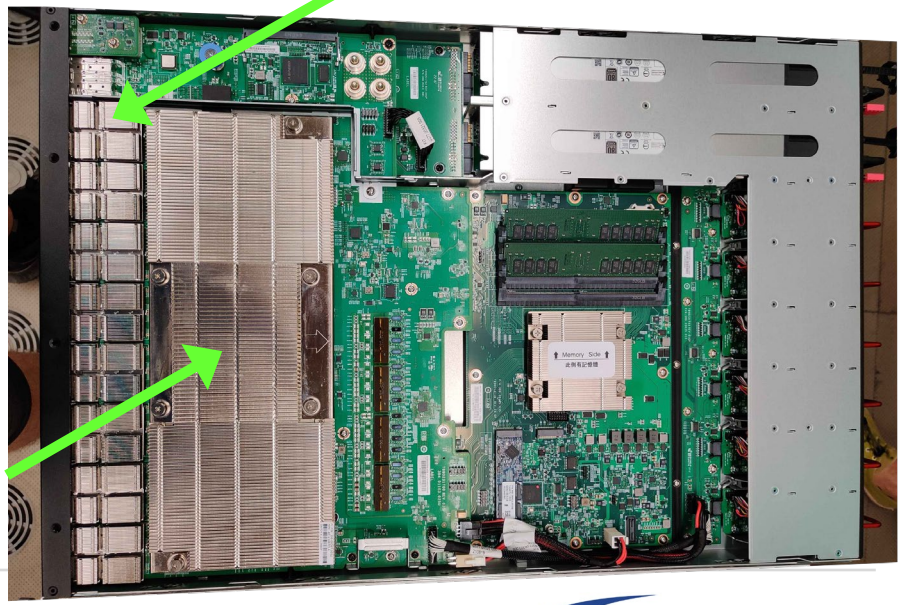
Example Router

- Netberg Aurora 830
  - Datacenter and Cloud Computing
  - Smaller buffer- and table sizes
    - Compared to UfiSpace S9610
  - But “only” 22.000€
  - 32x 400Gbit/s Ethernet
    - Ports can be split using “breakout cables”
    - Up to 256 servers connected at 50 Gbit/s
  - Used at our institute for research
    - Pictures made at SCC networking room
    - Experiments with DDoS attack detection
  - Also a white-box router



400 Gbit/s Interfaces



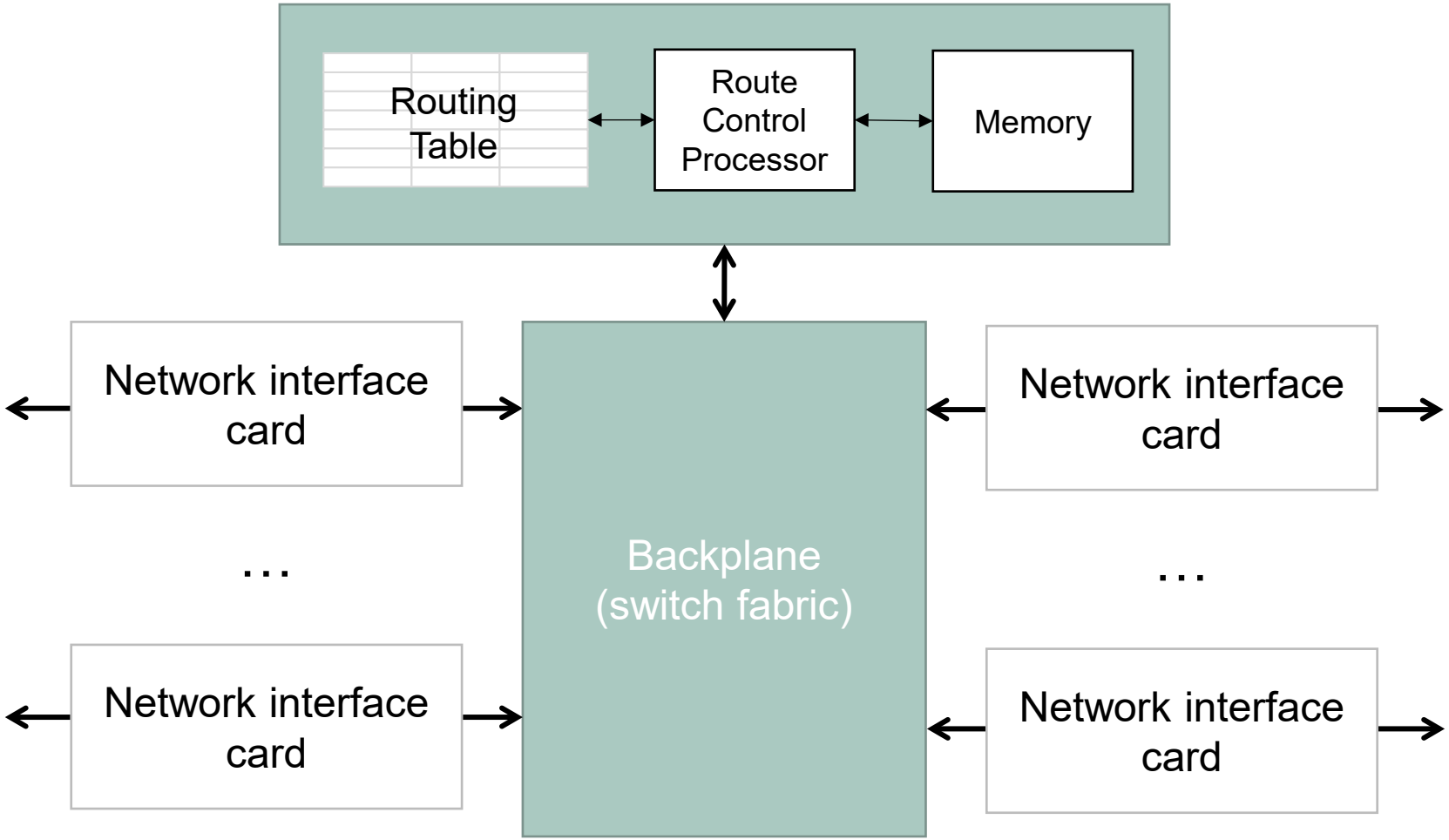


Packet Processor (with cooler)

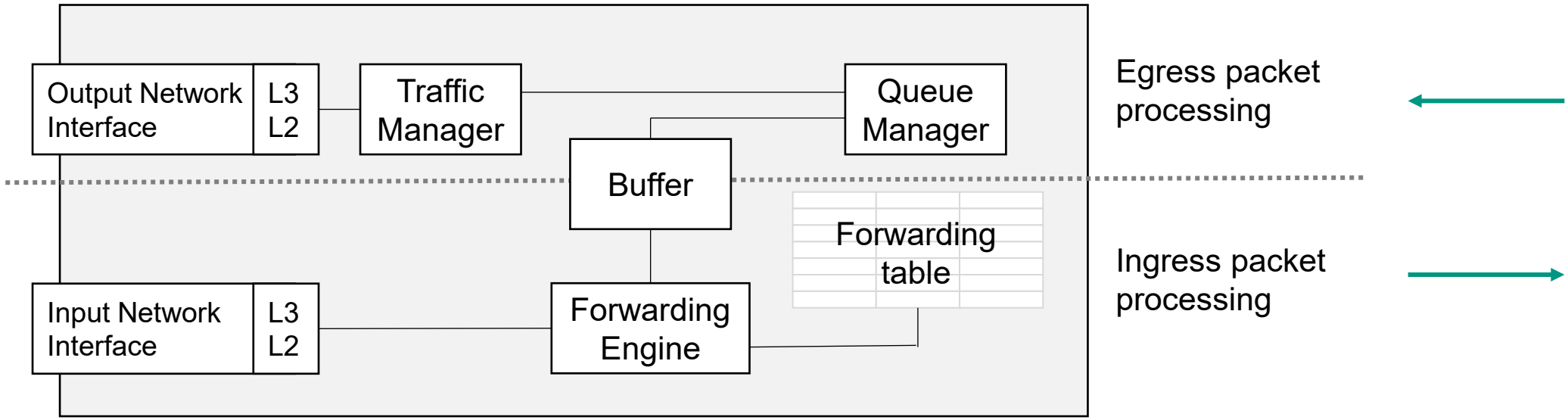
2.3 Generic Router Architecture

[Medh18]

Generic Router Architecture



Network Interface Card



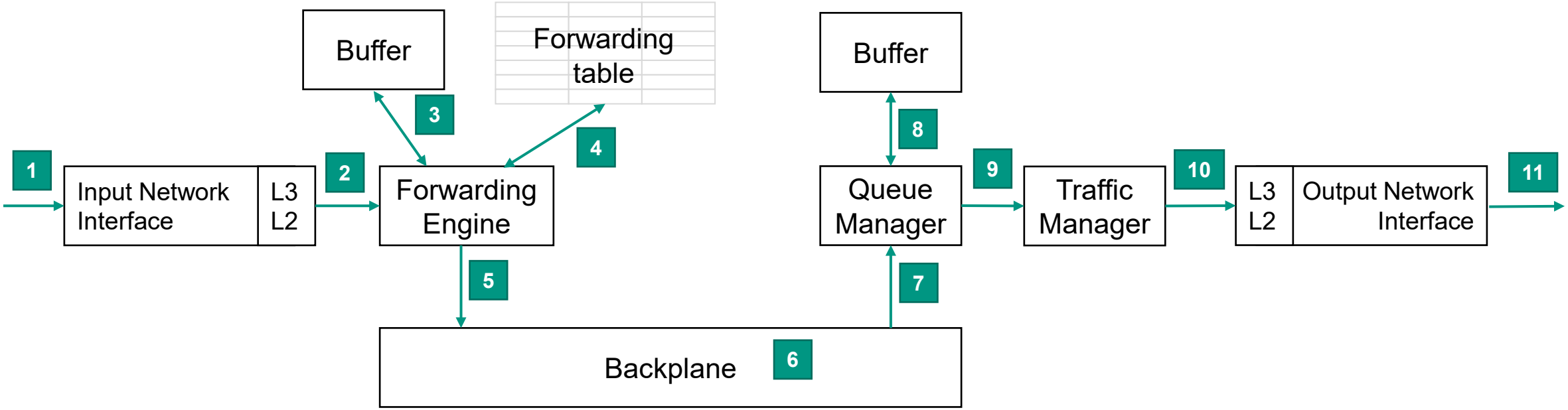
L3: layer 3
L2: layer 2

Major Functional Modules

- Network interface
  - Provides ports with connectivity to physical link (e.g., Ethernet port)
  - Decapsulate layer 2 header
  - Forward IP packet (or layer 3 header only) to forwarding engine
  - Entire packet is buffered
- Forwarding engine
  - Decides which network interface the incoming packet should be forwarded to
  - Accesses forwarding table
  - For QoS support classifying of packets is needed
- Queue manager
  - Manage occupancy of queue
  - Implement policy to drop packets when needed
- Traffic manager
  - Prioritizing traffic
  - Shapes traffic according to service level agreement
- Backplane
  - Provides connectivity for network interfaces
    - Shared or switched
  - Realizes internal forwarding of packets from input to output port
- Routing processor
  - Implements routing protocols
  - Maintains routing table
  - Configuration and management of router

Packet Flow in a Router

- Data plane



Fast Path vs. Slow Path

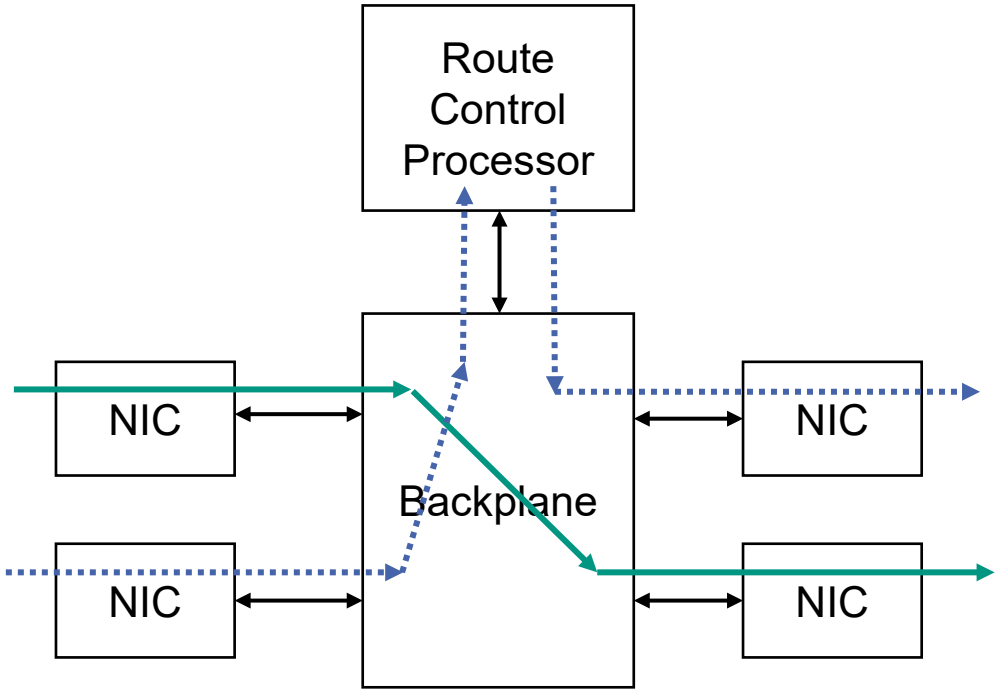
- Router tasks comprise

Fast Path

  - Time critical tasks and
    - Header processing, forwarding, packet classification and scheduling
    - For high-performance purposes implemented in HW
      - Processing in line-speed

Slow Path

  - Non-time critical tasks
    - Packets for maintenance, management, error handling, e.g.,
      - Generation of ICMP packets
      - Packets related to routing protocols
      - SNMP packets
      - IP packet fragmentation
      - IP options



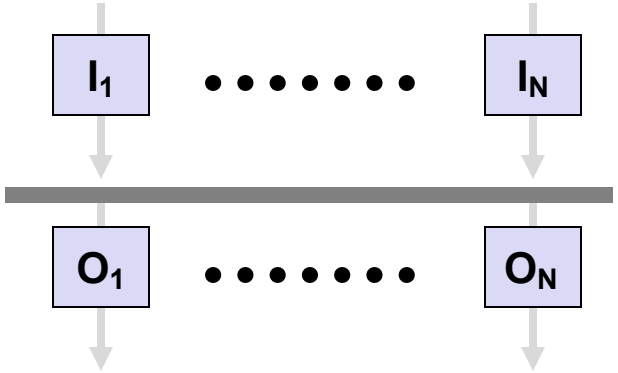
Fast path
Slow path

Structure of the Switch Fabric

- Four typical basic structures
  - Shared memory
  - Bus / ring structure
  - Crossbar
  - Multi-level switching network
- Evaluation
  - Internal blocking behavior
    - Blocking / non-blocking
  - Presence of buffers
    - Buffered / unbuffered
  - Topology and number of levels of the switching network and number of possible routes
  - Control principle for packet routing
    - Self-controlling / table-controlled
  - Internal connection concept
    - Connection oriented / connectionless

Bus or Ring Structures

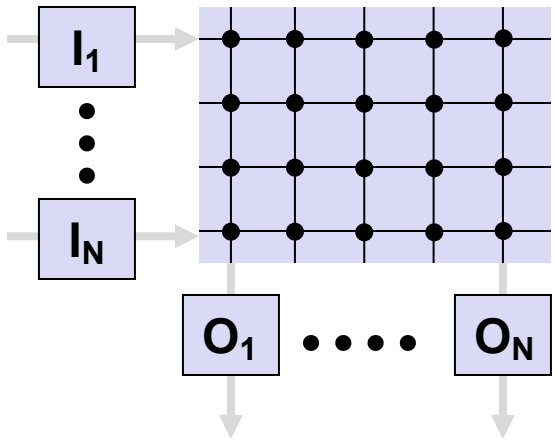
- Idea
  - Conflict-free access through time-division multiplexing
  - Transmission capacity bus / ring
    - At least the sum of the transmission capacities of all input ports



- Characteristics
  - Easy support for multicast and broadcast
  - Spatial extension of bus system is limited

Crossbar

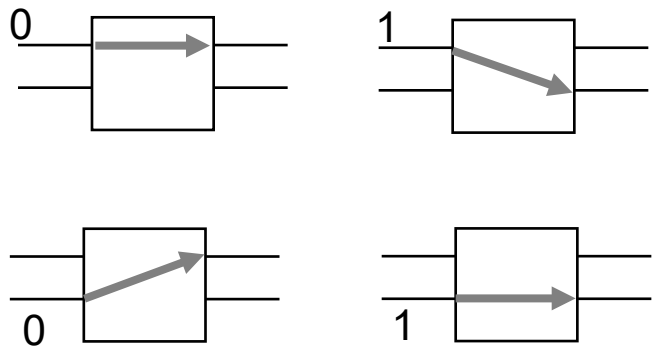
- Idea
  - Each input connected to each output via crossbar
    - N inputs, N outputs, N^2 crosspoints



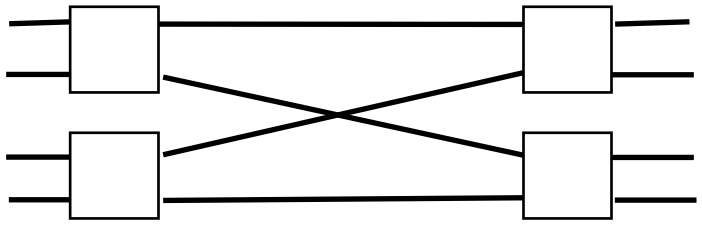
- Characteristics
  - Partial parallel switching of packets possible
  - Multiple packets for the same output → Blocking → Buffering required
  - High wiring costs with a large number of inputs and outputs
    - Mostly limited to 2x2 or 16x16 matrices
  - Especially efficient with packets of the same size

Multi-level Switching Networks

- Build from switching states of an elementary switching matrix



- Example



4x4 Banyan network

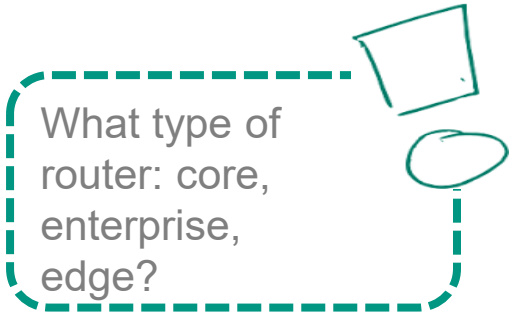
- Characteristics
  - Less wiring effort than crossbar
  - Each input can be connected to each output
  - Not all connections possible at the same time
    - internal blocking possible

2.4 On Router Architectures

[Medh18]

Routing Architectures

- Conflicting design goals
  - High efficiency
    - Line speed
    - Low delay
  - vs. low cost
    - Type and amount of required memory
    - Type of switch fabric



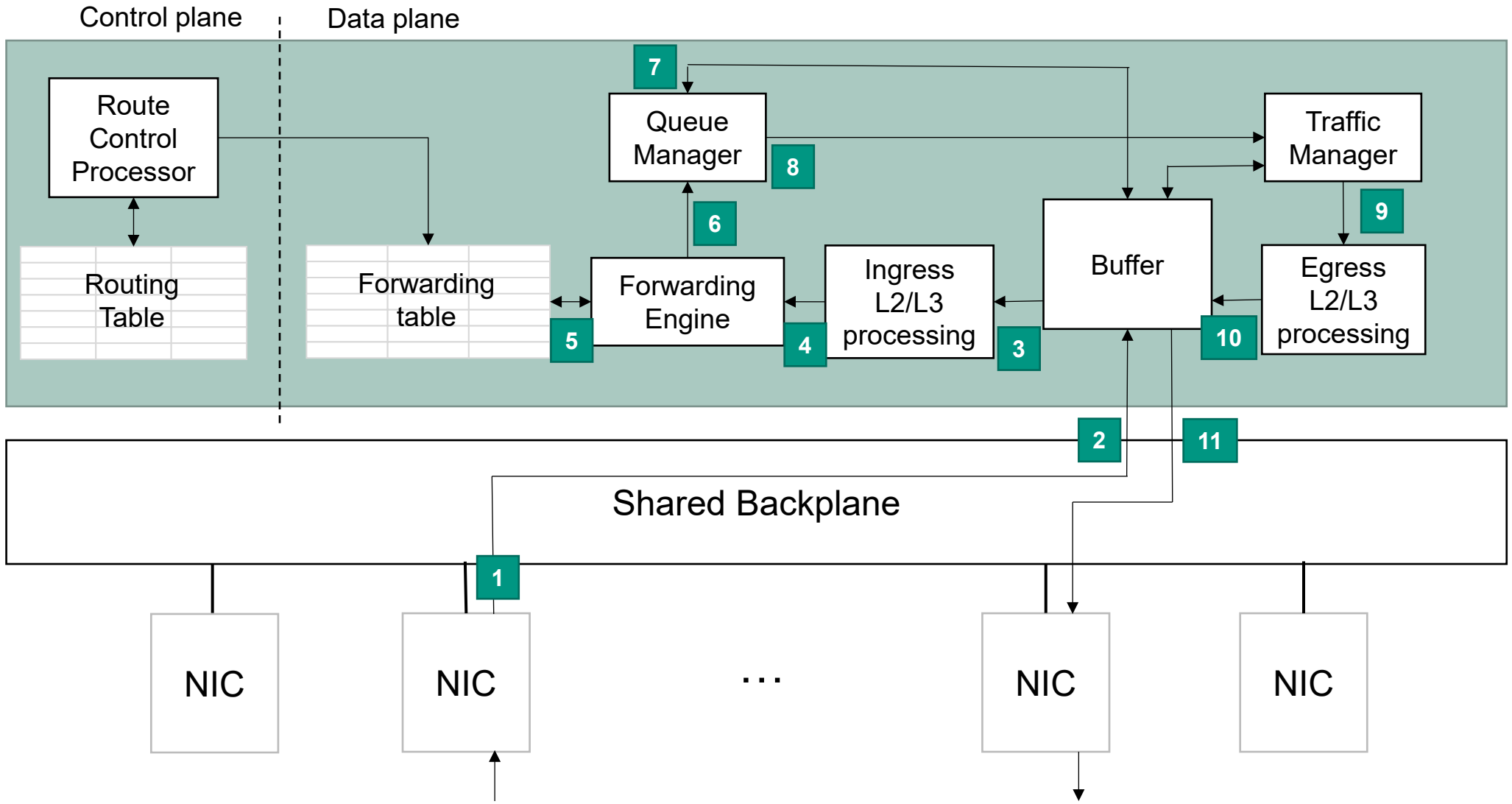
- Problem: how are the functional modules mapped to architectural components of the router ? Examples:
  - Shared CPU architecture
  - Shared nothing architecture

Example: Shared CPU Architecture

- CPU
  - Runs commodity operating system
  - Implements all functional modules of data plane
  - Implements functionality of route control processor

- Network interface cards
  - Each NIC implements a network interface to provide connectivity
  - Share same CPU for forwarding function

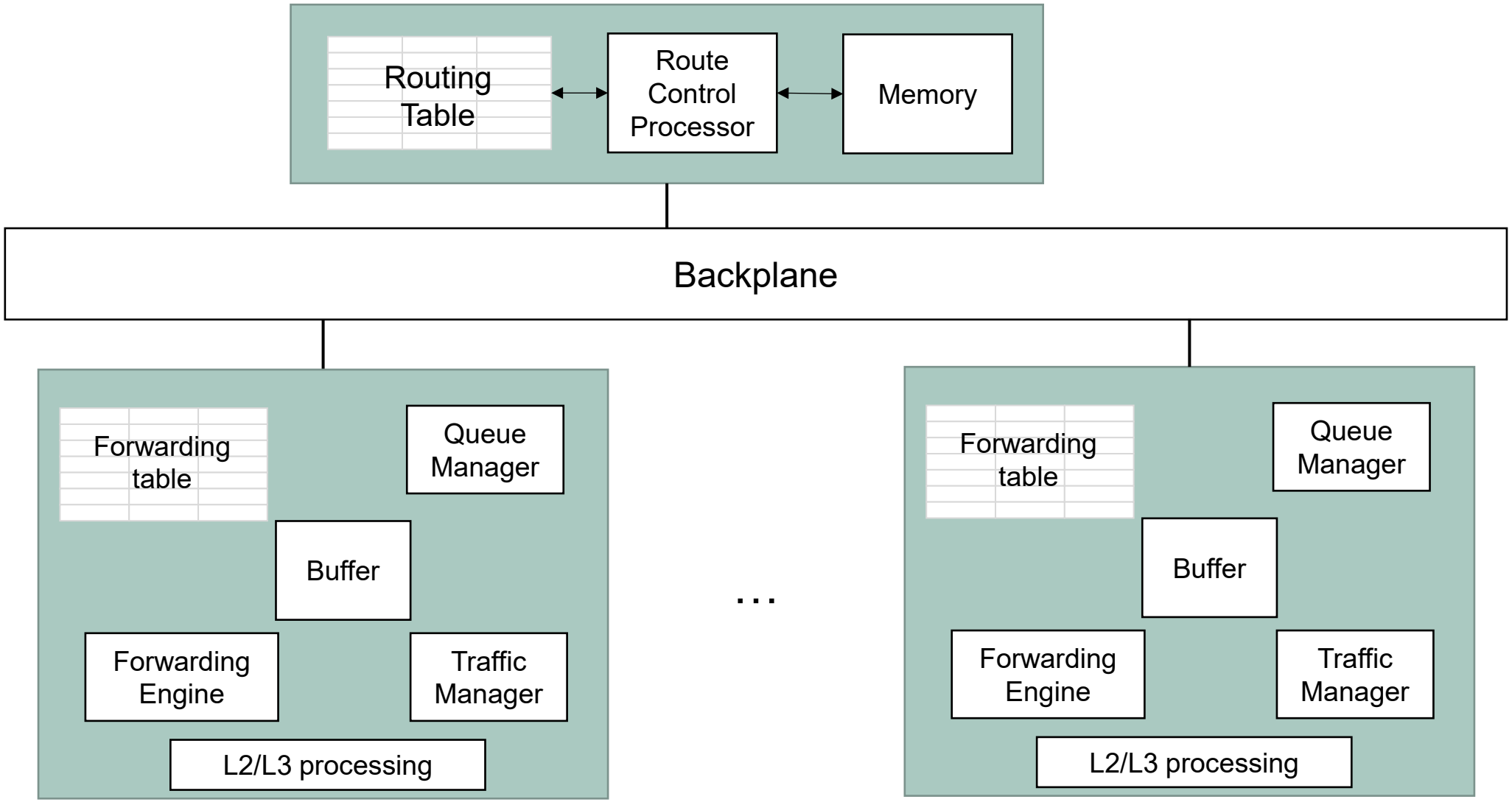
Shared CPU Architecture



Shared CPU Architecture

- Data flow
  - Packet arrives at line card. Interrupt to CPU. Packet is transferred to buffer
  - Ingress L2/L3 processing takes place. Exctraction of packet headers.
  - Forwarding engine determines egress port
  - Queue and traffic manager prioritize packet and shape traffic
  - Egress L2/L3 processing
  - Transfer of packet from buffer to egress line card
- CPU cycles used for
  - Packet forwarding
  - Processing of routing packets
  - Updates of routing and forwarding tables
  - Processing of management functions for configuring and administrating the router
  - → used for control path and for data path
- Shared backplane
  - Each packet traverses this backplane twice

Example: Shared Nothing Architecture

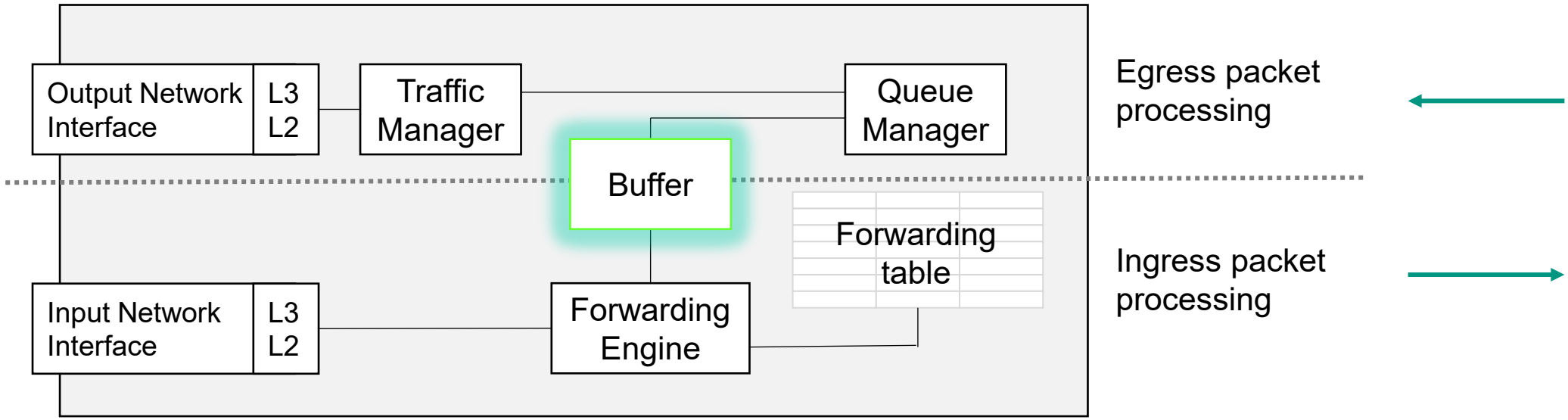


Shared Nothing Architecture

- All forwarding functions are offloaded to line cards
  - Packet has to traverse backplane only once
- Implementation of forwarding functions
  - Typically on FPGAs or ASICs for high performance routers
  - → Networking chips

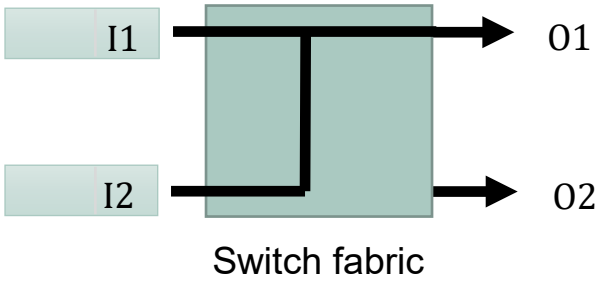
2.5 Buffers

Network Interface Card



Placement of Buffers

- Problem
  - Simultaneous arrival of multiple packets for an output port
    - Sequential processing required, since packets can not be sent in parallel
    - Packets have to be buffered
- Example
  - Packets arrive at input ports I1 and I2 at the same time, both must be forwarded to output O1
  - One out of the two packets requires buffering



→ Where to place the memory elements for buffering?
- Input buffer
- Output buffer
- Distributed buffer
- Central buffer

Evaluation of Alternatives

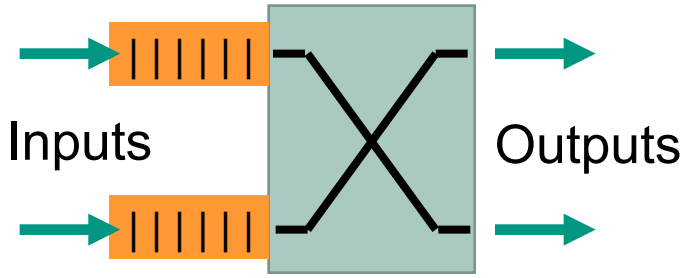
- Parameters of switch fabric
  - Number N of input and output ports
  - Total storage capacity M that is required
  - Speedup factor S of the switch fabric
    - According to the speed of the input and output ports
  - Cycle time Z of memory accesses
    - According to the transmission time of a packet at input and output ports
  - Delay und jitter (=variance of delay)

- Important
  - Additional mechanisms are required, e.g. flow control
  - Organization of memories, e.g. FIFO or RAM

- In the following: simplifying assumptions
  - All ports operate at same data rate
  - All packets have same length

Input Buffer

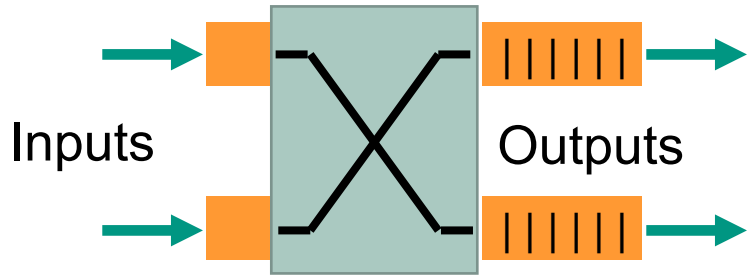
- Idea: conflict resolution at input of switch fabric
  - FIFO buffer per input port
  - Scheduling of inputs, e.g.
    - Round robin, priority controlled, depending on buffer levels, ...
    - Jitter varies
  - Switch fabric internally non-blocking, i.e., no internal conflicts



- Requirements
  - Internal exchange with speed of ports, i.e., $S = 1$
  - Cycle time $Z = {}^{1}/_{2}$
- Characteristics
  - Head-of-Line blocking
    - Waiting packet at head of the buffer blocks packet behind it that could be serviced
  - Maximum throughput: 75% for $N = 2$ and $58{,}58\%$ for $N \to \infty$

Output Buffer

- Idea: conflict resolution at output of switch fabric
  - FIFO buffer per output port
  - Switch fabric internally non-blocking, i.e., no internal conflicts

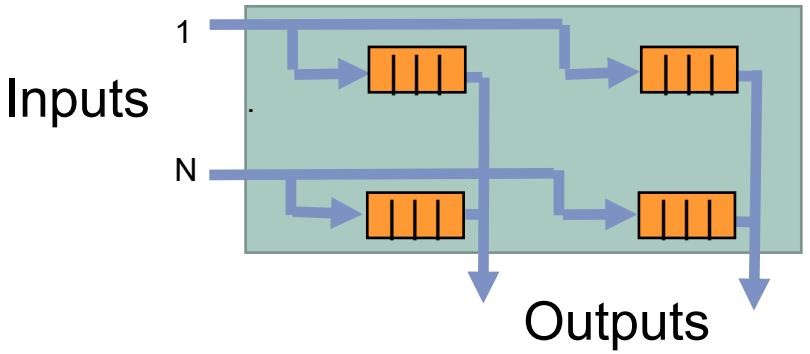


- Requirements
  - Internal switching of packets at N times the speed of the input ports: $S = N$
  - Switching of N packets during one cycle possible. This results in $Z = 1 / (N + 1)$
  - Output buffer must be able to accept packets at N times the speed
  - Input buffer necessary to accept a packet

- Characteristics
  - Maximum throughput near 100%, usually at approx. 80-85%
  - Good behavior with respect to delay and jitter

Distributed Buffer

- Idea: conflict resolution inside switch fabric
  - Switch fabric as matrix
  - FIFO buffer per crosspoint

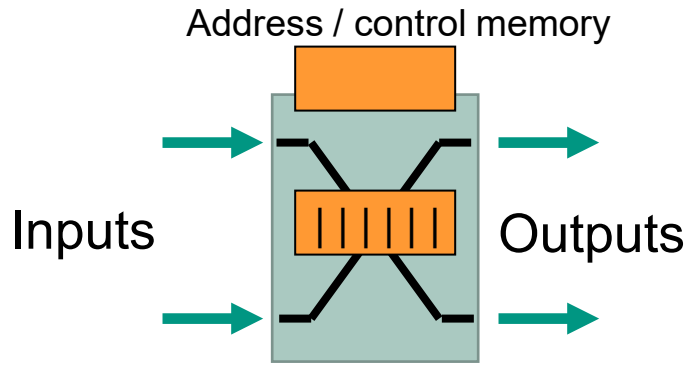


- Requirements
  - Matrix structure
  - Internal exchange with speed of ports: $S = 1$
  - Cycle time: $Z = 1/2$

- Characteristics
  - No Head-of-Line blocking
  - Higher memory requirement M than input or output buffering

Central Buffer

- Idea: conflict resolution with shared buffer
  - All input and output ports
    are connected to a shared buffer
    (organization: RAM)

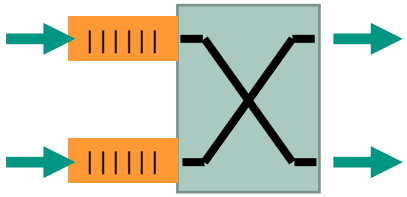


- Requirements
  - Cycle time $Z = 1 / 2N$
  - Address and control memory
    for address information of packets and control of parallel memory accesses

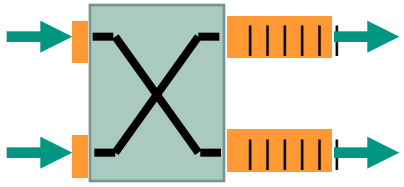
- Characteristics
  - Significantly lower memory requirements than other alternatives (low waste)
  - But: requirements with respect to memory access time are higher

Buffer Placement: Summary

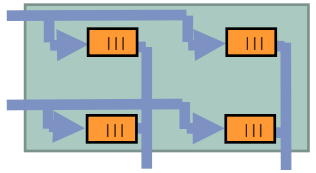
Simple implementation
Head-of-Line Blocking
Throughput 60% - 75%



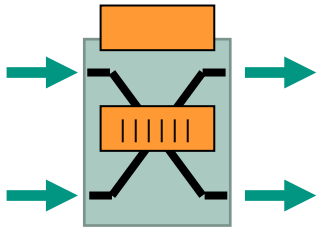
Throughput 80%- 85%
Fast memory access required
Additional input buffer required



Throughput optimal
High memory requirements

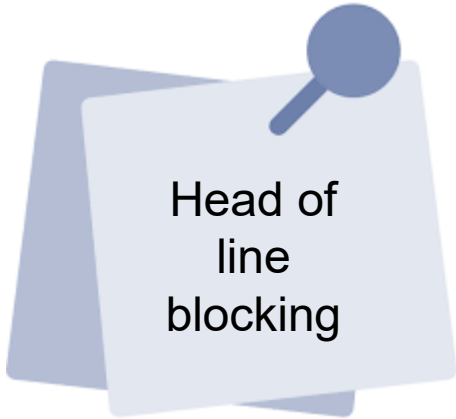


Low memory requirements
Fast memory access required



Homework

Homework 02-01



On Router Buffer Size

Current Research

- Observations
  - Moore´s law
    - Product of speed (clock rate) and complexity (device count) doubles every 18 months
  - Network link rates double every 2 years
    - → buffer speed has to double every 2 years
    - → buffer size has to double every 2 years
    - → buffer speed-complexity product needs to quadruple every 2 years!!
    - Practically not feasible !

[Hous20, McAK19]

On Router Buffer Size

Current Research

- What is the right size of router buffers?
  - A debate that goes on for decades already
  - We refer to the delay bandwidth buffer here
- Many influencing factors, e.g.,
  - Mix of traffic in the network
  - Queueing disciplines
  - Congestion control algorithms

Again, the interplay of different mechanisms is important to the overall desgin

We espacially come back on the issue of congestion control in a later chapter

- Interesting current contributions in this context

[G. Appenzeller; I. Keslassy, Nick Mc Keown; Sizing Router Buffers; ACM SIGCOMM 2004] — Peer reviewed

[N. Mc Keown, G. Appenzeller, I. Keslassy; Sizing Router Buffers (redux); ACM SIGCOMM Computer Communication Review, Vol. 49, Isue 5, October 2019] — Not peer reviewed

[G. Houston; What´s the Right Network Buffer Size; The Internet Protocol Journal, May 2020] — Not peer reviewed

[B. Sprang, S. Arslan, N. Mc Keown; Updateing the Theory of Buffer Sizing; Performane Evaluation Revies, Vol. 49, No. 3, December 2021] — Peer reviewed

[S. Yeluri; Sizing Router Buffers Small is the New Big; February 2023] — Blog

On Router Buffer Size

Current Research

- Buffer size has implications on router design
  - Small buffer: may fit on switch ASIC
    - Router is smaller, simpler, cheaper, consumes less power
    - But: may cause packet losses
  - Big buffer: needs off-chip memory which is slower
    - Typically causes less packet losses
    - But: possibly large queueing delay
    - But: costly router

[Hous20, McAK19]

On Router Buffer Size

Current Research

Karlsruher Institut für Technologie

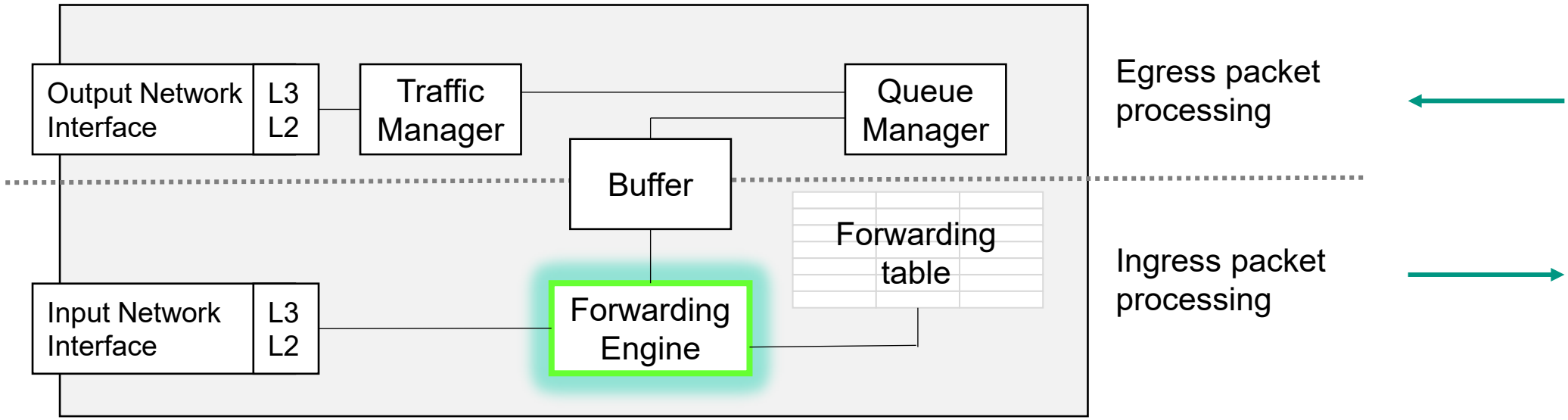
- Typical answer for many years
  - Buffer size equal to one bandwidth-delay product
    - $buffersize = data\ rate \cdot RTT$
  - Example calculations
    - Link with 100 Gbit/s and RTT of 200 ms → 2,4 GByte of buffer needed
      → queueing delay of 200 ms possible
    - Link with 1 Tbit/s and RTT of 100 ms → 12,5 GByte of buffer needed

- Smaller buffer sizes
  - Flows in a high-end router
    - Thousands of flows, very different RTTs, rarely synchronized
  - For N long-lived desynchronized flows
    - $buffersize = (data\ rate \cdot RTT_{min})/\sqrt{N}$

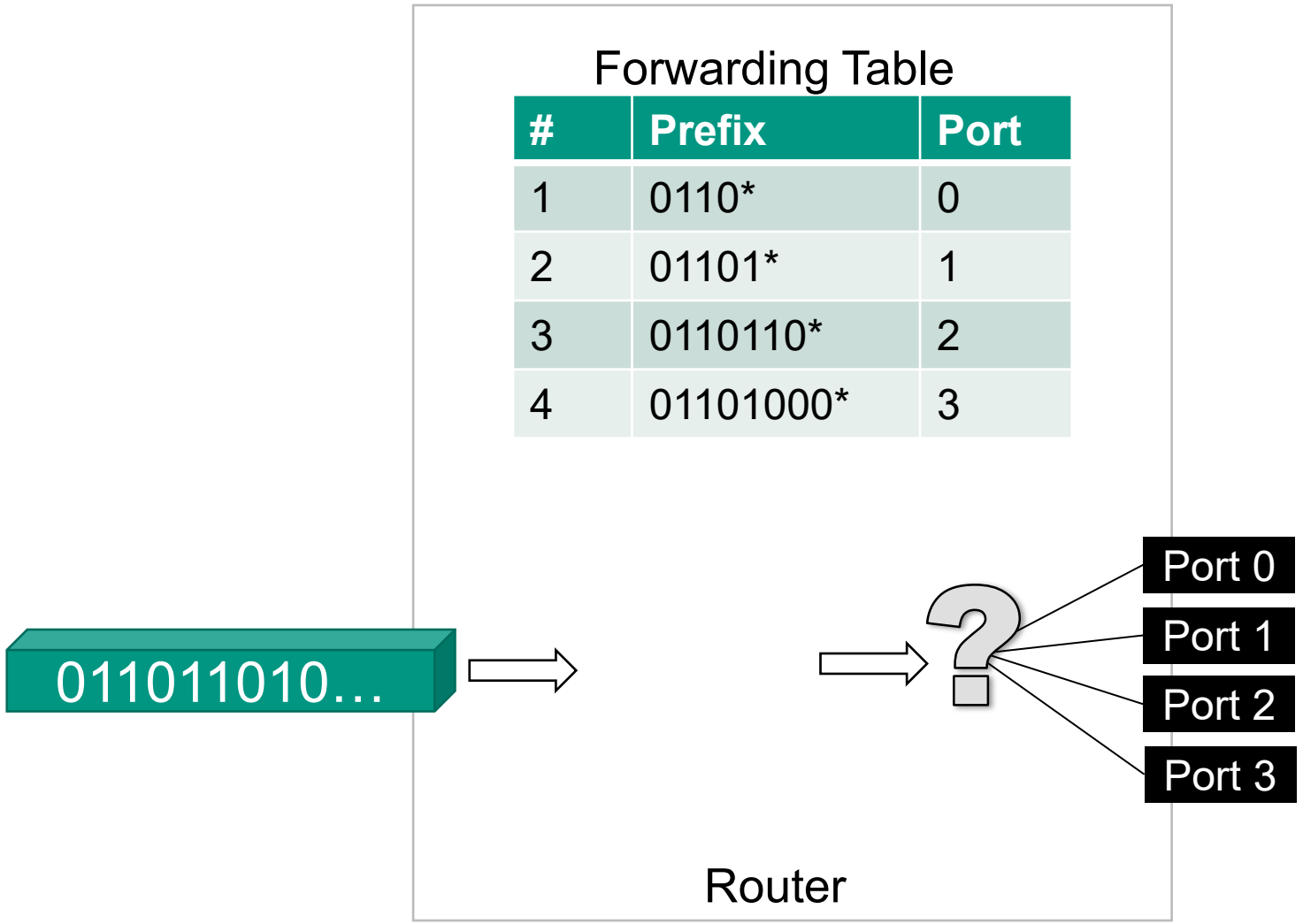
[ApKM04]

2.6 Forwarding Table Lookup

Network Interface Card



Packet Forwarding



Lookup in Forwarding Table

- Example of a forwarding table

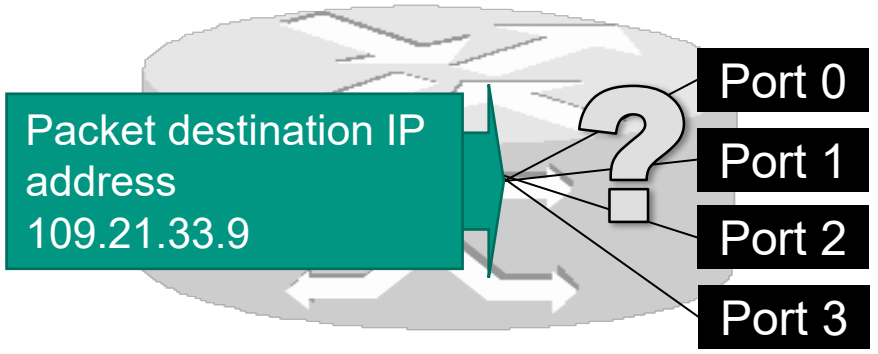
Prefix	Port
11001100 01010101 000010*	3
11001100 01010101 00001100*	1
11001100 01010101 000011*	4
default	2

- Prefix
  - Identifies a block of addresses
    - In the following, we use IPv4-addresses (for simplicity)
  - Continuous blocks of addresses per output port are beneficial
    - … does not require a separate entry for each IP address

    → Scalability

Longest Prefix Matching

- Problem
  - What if multiple prefixes in the forwarding table match on a given destination address?
- Select most specific prefix
  - Highest number of matching bits, i.e., the longest prefix
    - → Longest prefix matching



all three prefixes match

#	Prefix	Port
1	0110*	0
2	01101*	1
3	0110110*	2
4	01101000*	3

109.21.33.9 → 0110 1101 . 0001 0101 . 0010 0001 . 0000 1001

Efficient Prefix Search

- Challenge
  - Lookup in line speed

- Data structures for longest prefix matching
  - Binary trie
  - Path-compressed trie
  - Multibit tries
  - Bloomfilters
  - Hashtables
  - …

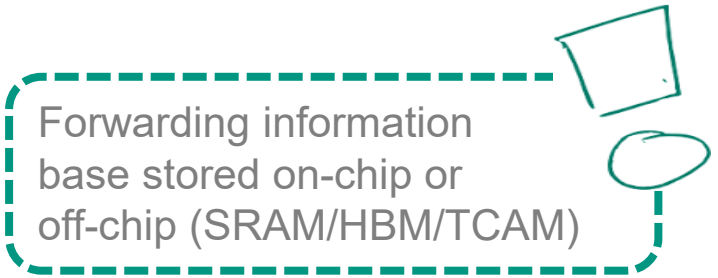
2.3.1 Efficient data structures for longest prefix matching

Requirements

- Fast lookup
  - Very low time budget for processing small packets at high data rates
  - Small packets: e.g. TCP acknowledgements

Line speed	Time budget for 40 byte packets
1 Gbit/s	320 ns
40 Gbit/s	8 ns
100 Gbit/s	3,2 ns

- Low memory
  - High-end routers can have millions of forwarding entries
  - Memory required per forwarding entry is important scalability parameter



- Fast updates
  - Route changes can occur frequently
  - Data structures should support >> 100 updates / second

Simple Array

- Some variables
  - N = number of prefixes
  - W = length of a prefix (e.g., W=32 for full IPv4 addresses)
  - k = length of a stride (only for multibit tries)

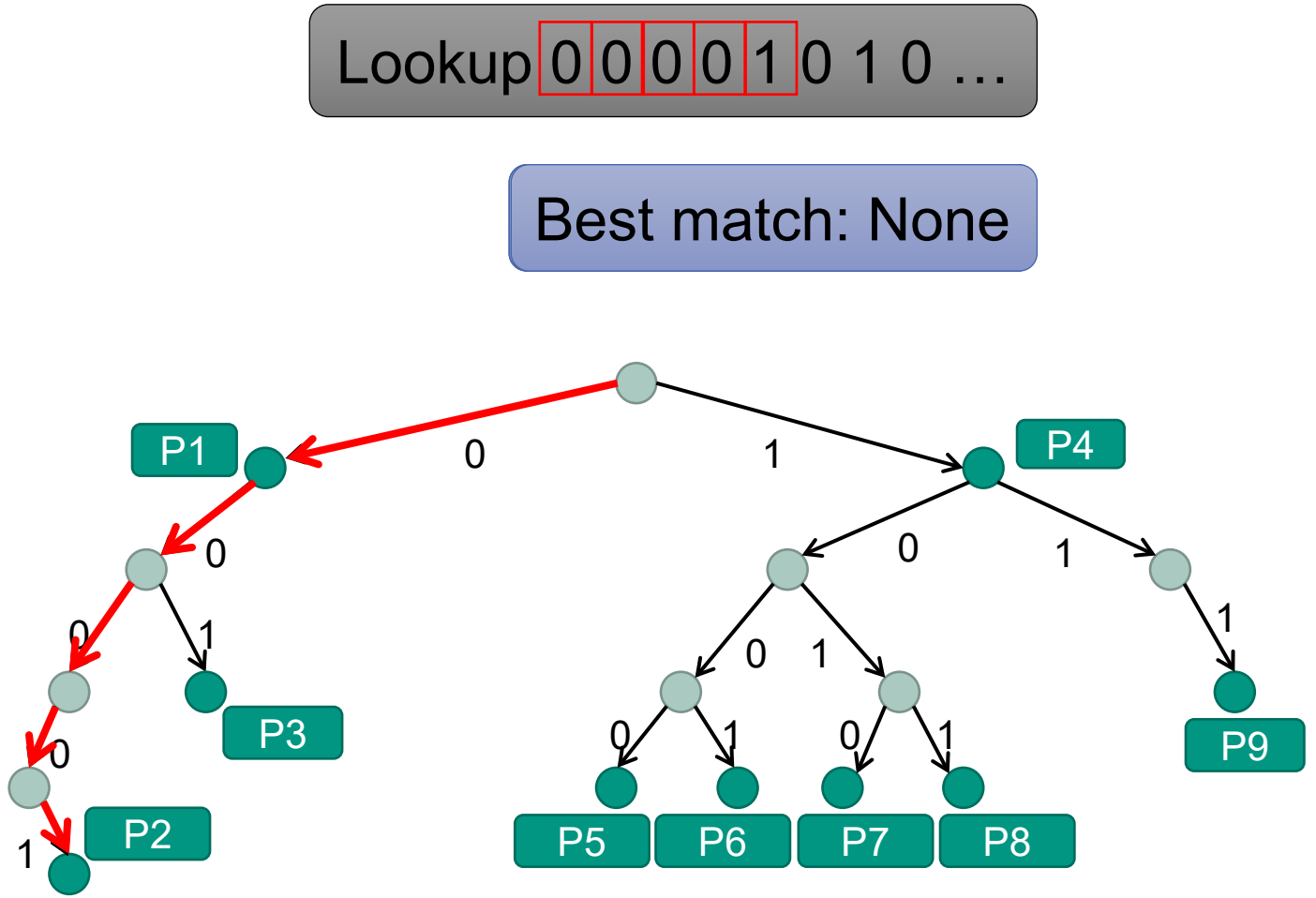
- Naïve approach
  - Store prefixes in a simple array (unordered)
  - Linear search
  - Remember best match while walking through array
  - Evaluation
    - Worst case lookup speed: O(N)
    - Memory requirement: O(N*W)
    - Updates: O(1)

} *pretty bad…*

- How can we do better?
  - Tries → tree-based data structures to store and search prefix information
    - From „retrieval“ (find something)
  - General idea: Bits in the prefix tell the algorithms what branch to take

Binary Trie

Prefixes	
P1	0*
P2	00001*
P3	001*
P4	1*
P5	1000*
P6	1001*
P7	1010*
P8	1011*
P9	111*



[Medh18]

Binary Trie - Evaluation

- Worst case lookup speed: O(W)
  - Maximum of one node per bit in the prefix
  - Much better than naïve approach (W << N),
- Memory requirement: O(N*W)
  - Assumption: prefixes stored as linked list starting from root node
  - Every prefix (out of N) can have up to W nodes
  - Maximum of N*W entries
  - No improvement (compared with naïve approach)
- Updates: O(W)
  - Similar to lookup procedure, a maximum of W nodes has to be inserted or deleted

Binary Trie

- Some thoughts about performance
  - Can find prefix in W steps → binary search in address space of 2^W
  - W = number of bits in address ($W = 32$ for IPv4, $W = 128$ for IPv6)
  - Assumption: separate memory access required for each step
    - Memory access time $t_{access} = 10ns = 10^{-8}s$
    - Maximum L lookups per second?
    - Worst case:
      $t_{lookup} = 32 \cdot t_{access} = 320ns \rightarrow L = \frac{1}{t_{lookup}} = 3.125.000\ lookups/s$

  → For 100 byte packets, this results in only $2{,}5\ Gbit/s$

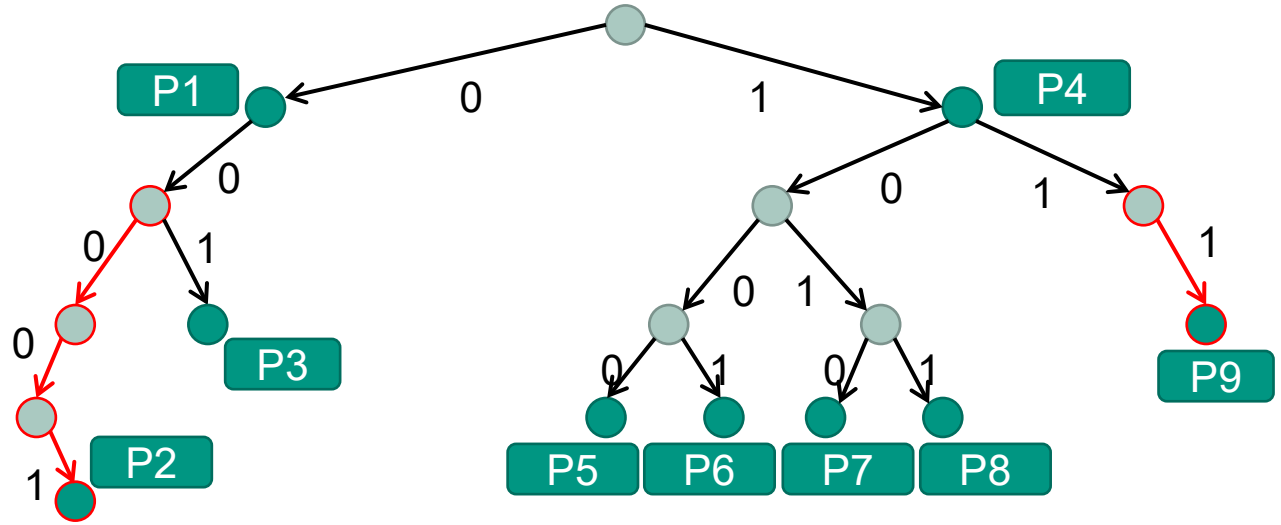
- Optimizations? E.g.,
  - Path compression
  - Multibit-Tries

See [Medh18] *for additional optimizations*

Path Compression

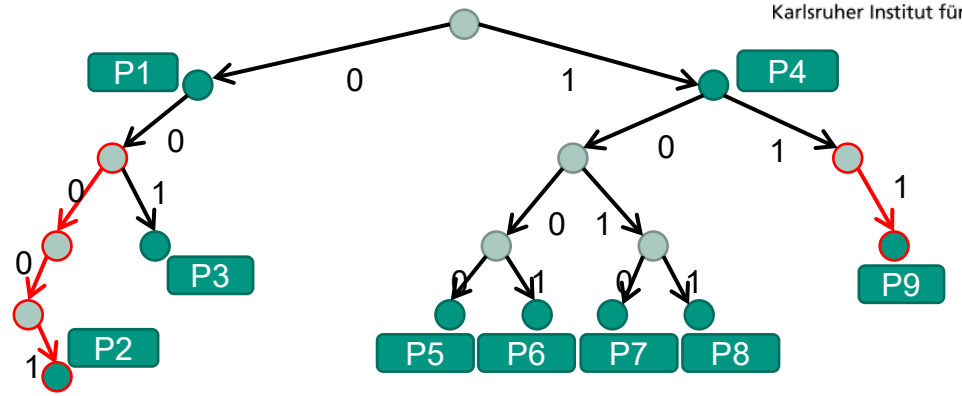
- Long sequences of one-child nodes waste memory
  - See highlighted (red) search paths in example trie
  - Not required for branching decision!

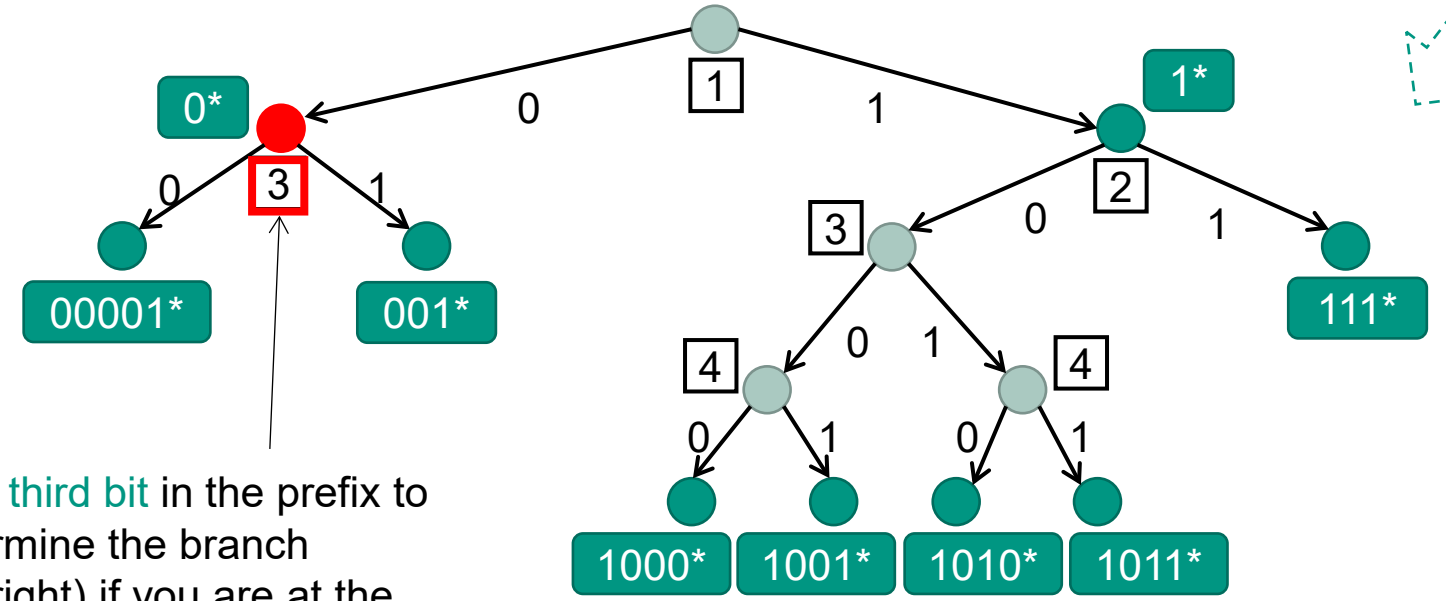
- Idea
  - Eliminate those sequences from trie



Path Compression

- Lookup operation
  - Additional information required
  - Store bit index
    that has to be examined next (X)

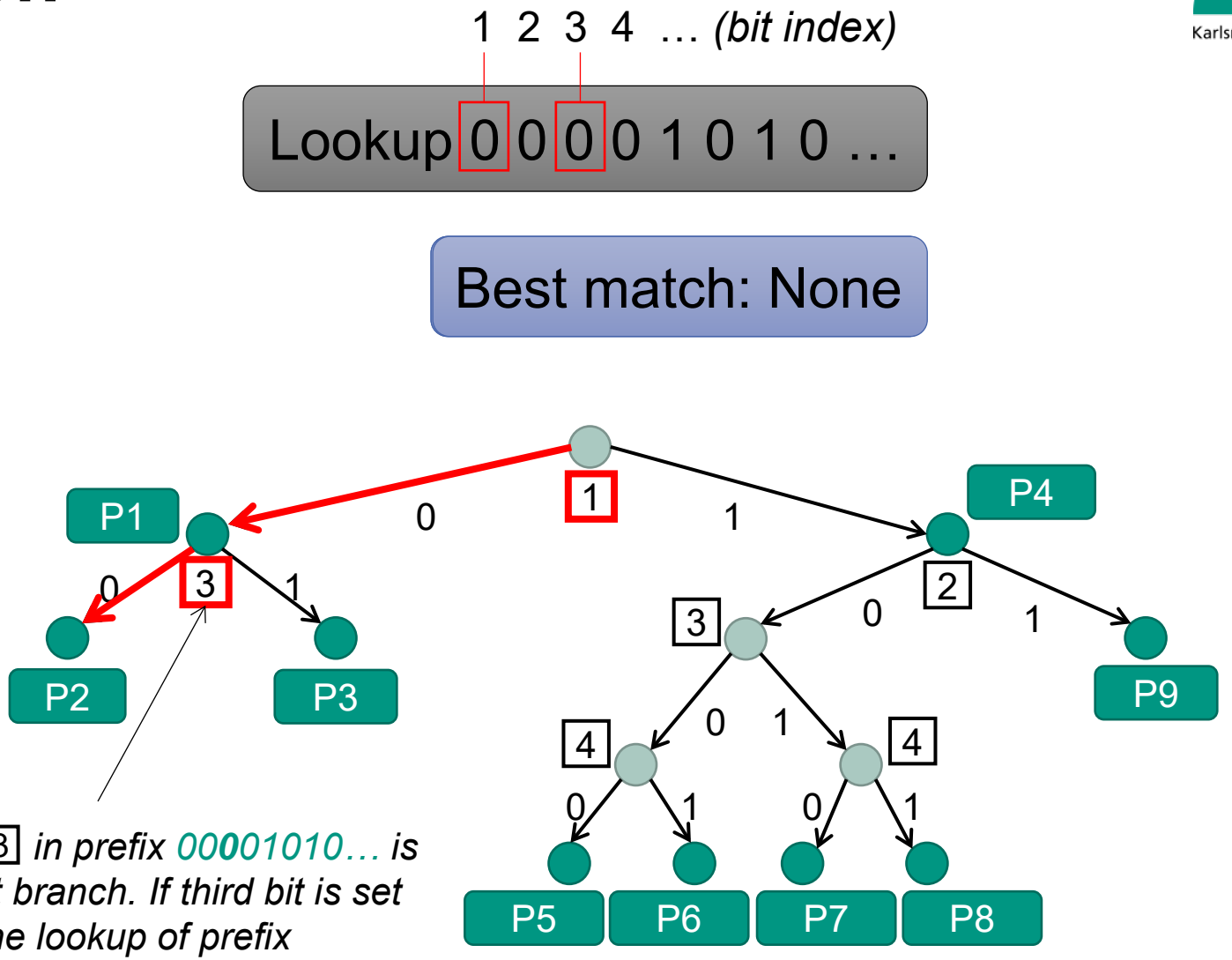




"Use third bit in the prefix to determine the branch (left/right) if you are at the red node"

Path Compression

Prefixes	
P1	0*
P2	00001*
P3	001*
P4	1*
P5	1000*
P6	1001*
P7	1010*
P8	1011*
P9	111*



Note: bit index 3 in prefix 00001010… is 0 so we take left branch. If third bit is set to 1 – e.g., for the lookup of prefix 001001… –, we take the right branch and P3 is selected as best match.

Path Compression - Evaluation

- Worst case lookup speed: O(W)
  - If there are no one-child nodes on a path,
    number of nodes to search is equal to length of prefix
  - → Only effective for sparse tries
- Memory requirement: O(N)
  - Maximum of N leaf nodes, N-1 for the internal nodes
    - Maximum of 2N-1 entries
  - → improvement against binary trie
- Updates: O(W)
  - Same argument as worst case lookup speed

Homework

Homework 02-02

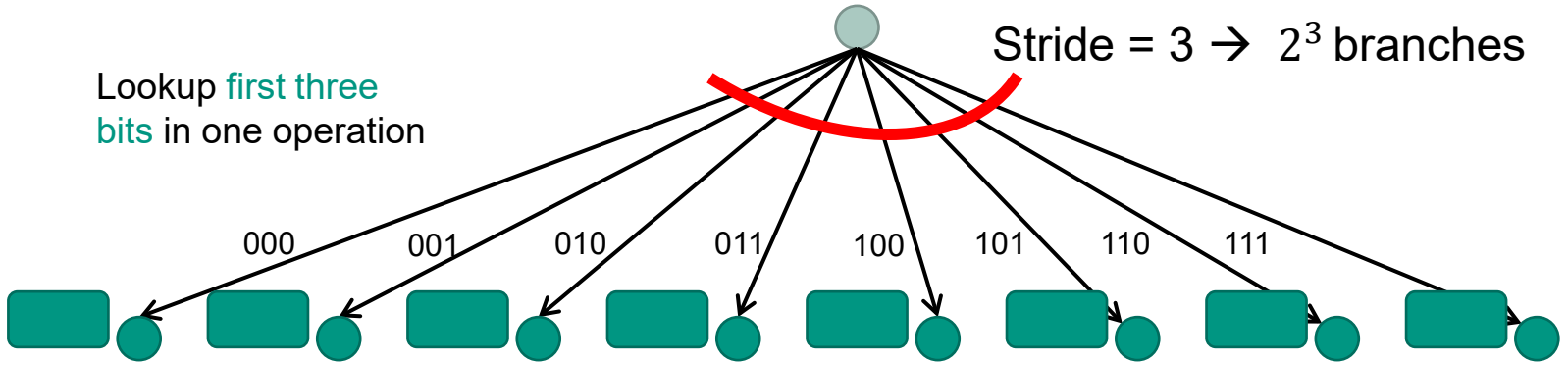
Homework

Homework 02-03

Multibit Trie

- How can we further improve lookup speed?

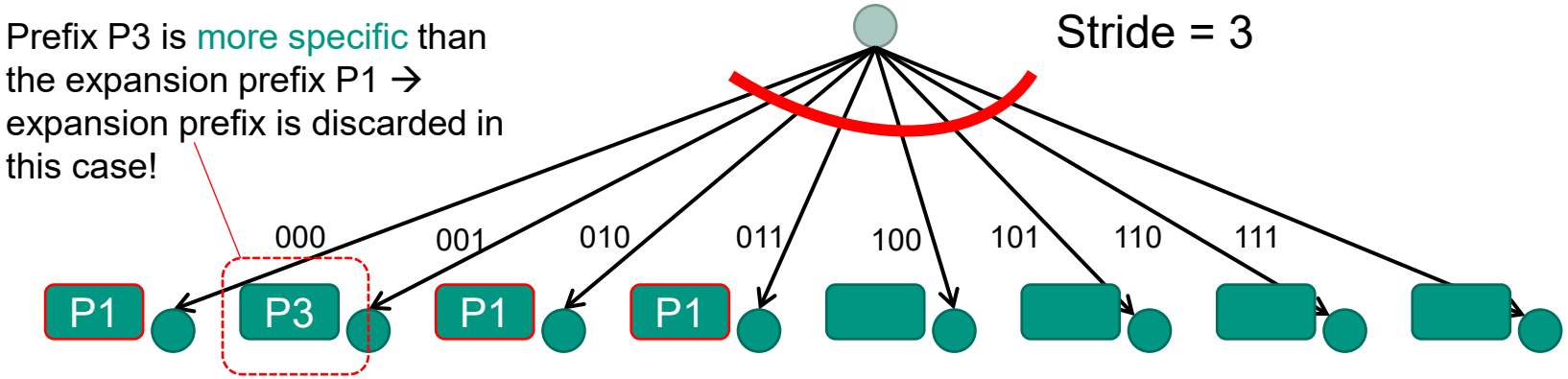
- Idea: Match multiple bits at a time instead of only a single bit
  - Reduced number of memory accesses
  - Example: Match in strides of 4 bits →
    only 8 instead of 32 memory accesses for an IPv4 lookup (speedup of factor 4)
  - Fixed stride multibit tries
    - Same stride on one level of the trie
    - Different levels can have different strides



Multibit Trie

- Problem: Prefixes usually do not match with strides
  - Prefix transformation required
  - One option: expand prefixes to the next available stride

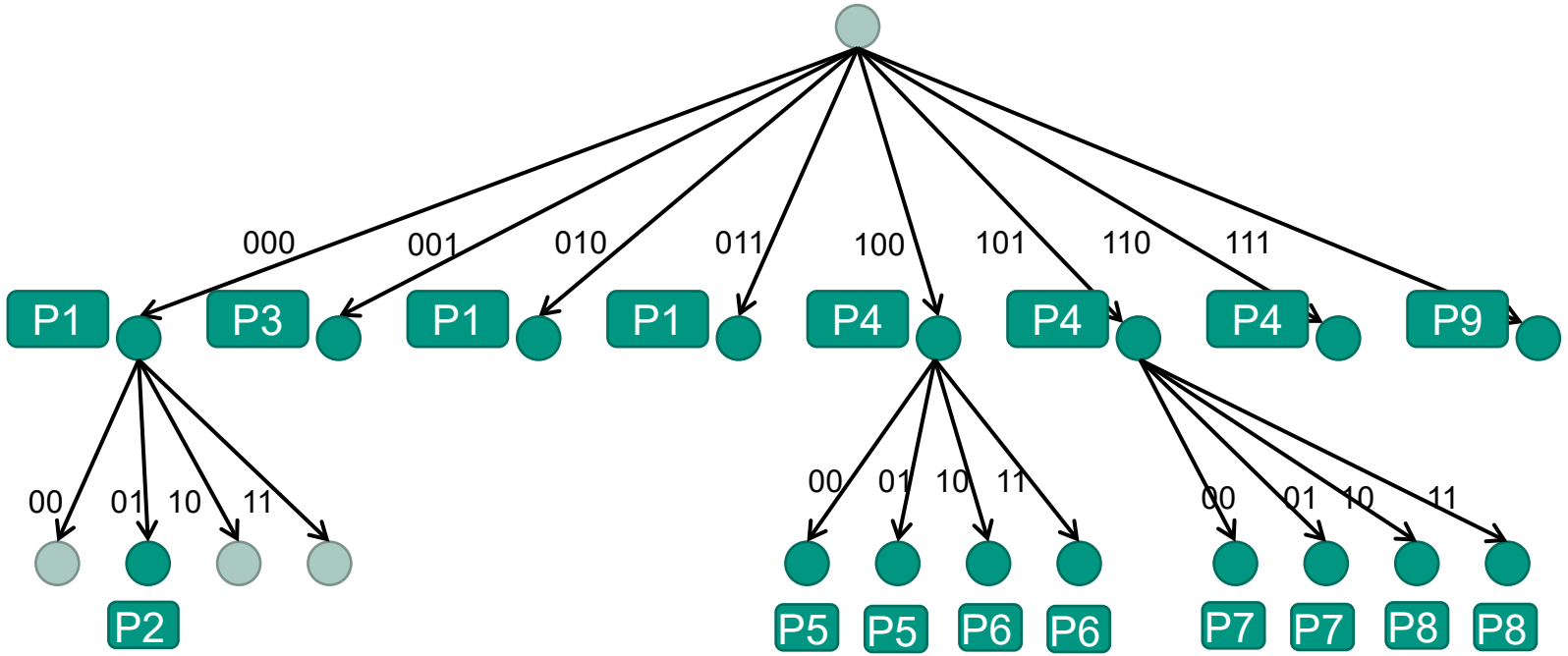
- Prefix Expansion
  - Single prefix 0* with length 1 is equivalent to
    - two prefixes 01* and 00* with length 2
    - four prefixes 000*, 001*, 010*, and 011* with length 3



Prefixes	
P1	0*
P2	00001*
P3	001*
P4	1*
P5	1000*
P6	1001*
P7	1010*
P8	1011*
P9	111*

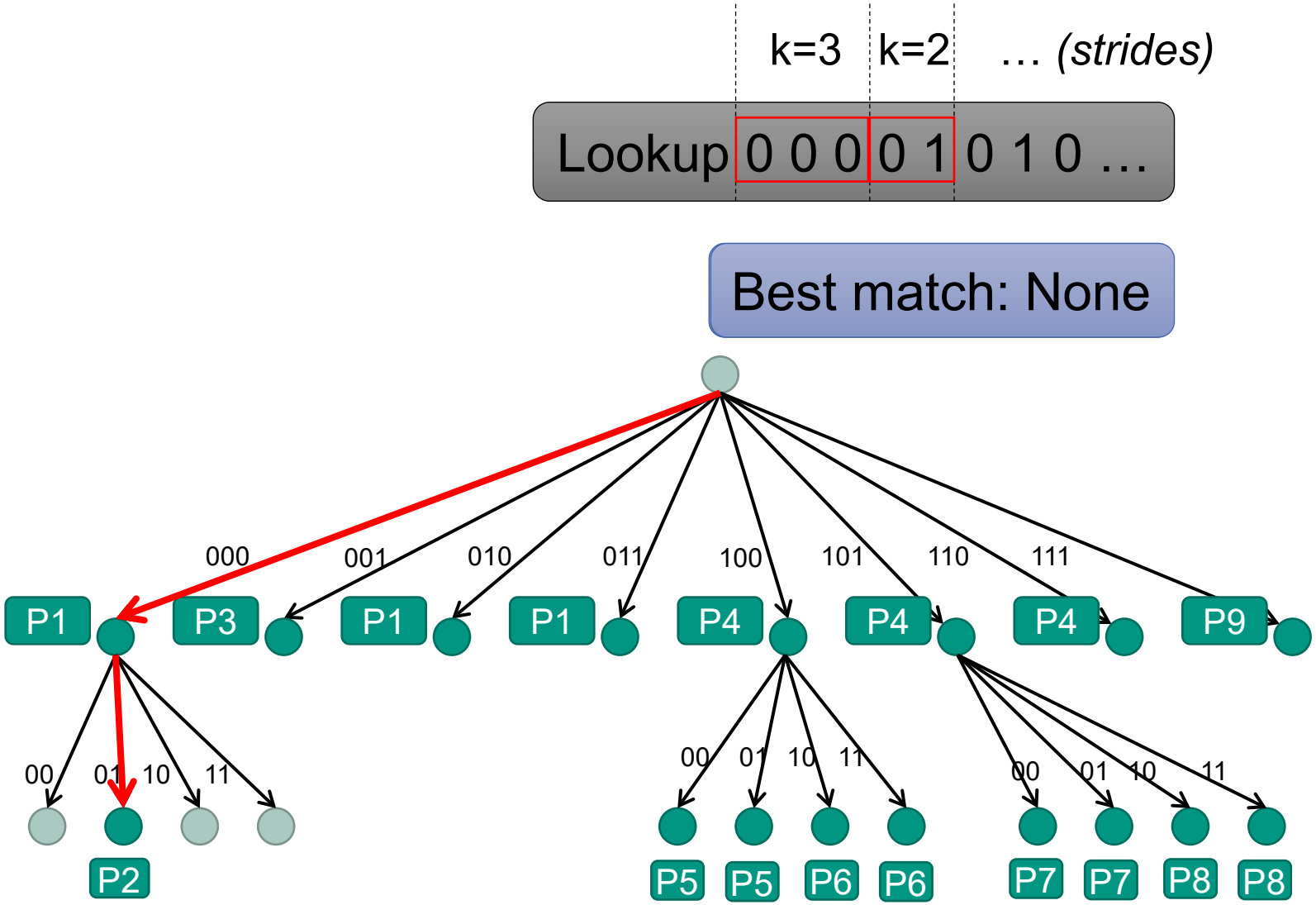
Multibit Trie

- Example of a fixed stride multibit trie with prefix expansion
  - Two different strides (we use variable k for the stride)
    - k=3 on first level, k=2 on second level
    - i.e., all prefixes are expanded to either length=3 or length=5 (3+2)

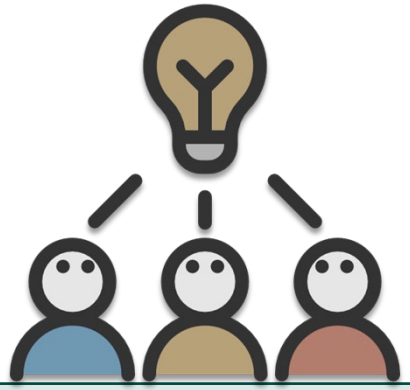


P1	0*
P2	00001*
P3	001*
P4	1*
P5	1000*
P6	1001*
P7	1010*
P8	1011*
P9	111*

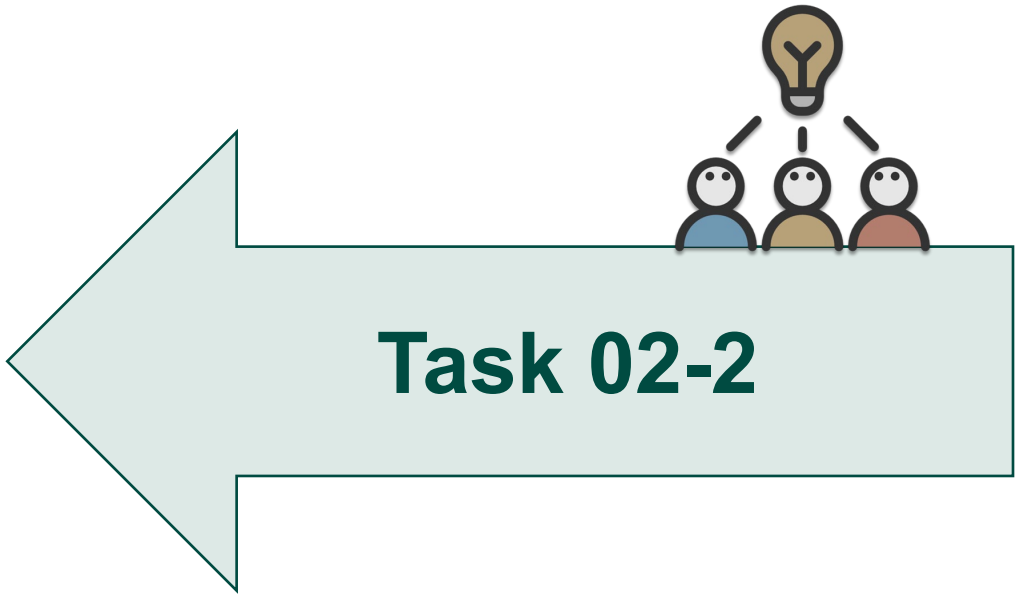
Multibit Trie



P1	0*
P2	00001*
P3	001*
P4	1*
P5	1000*
P6	1001*
P7	1010*
P8	1011*
P9	111*

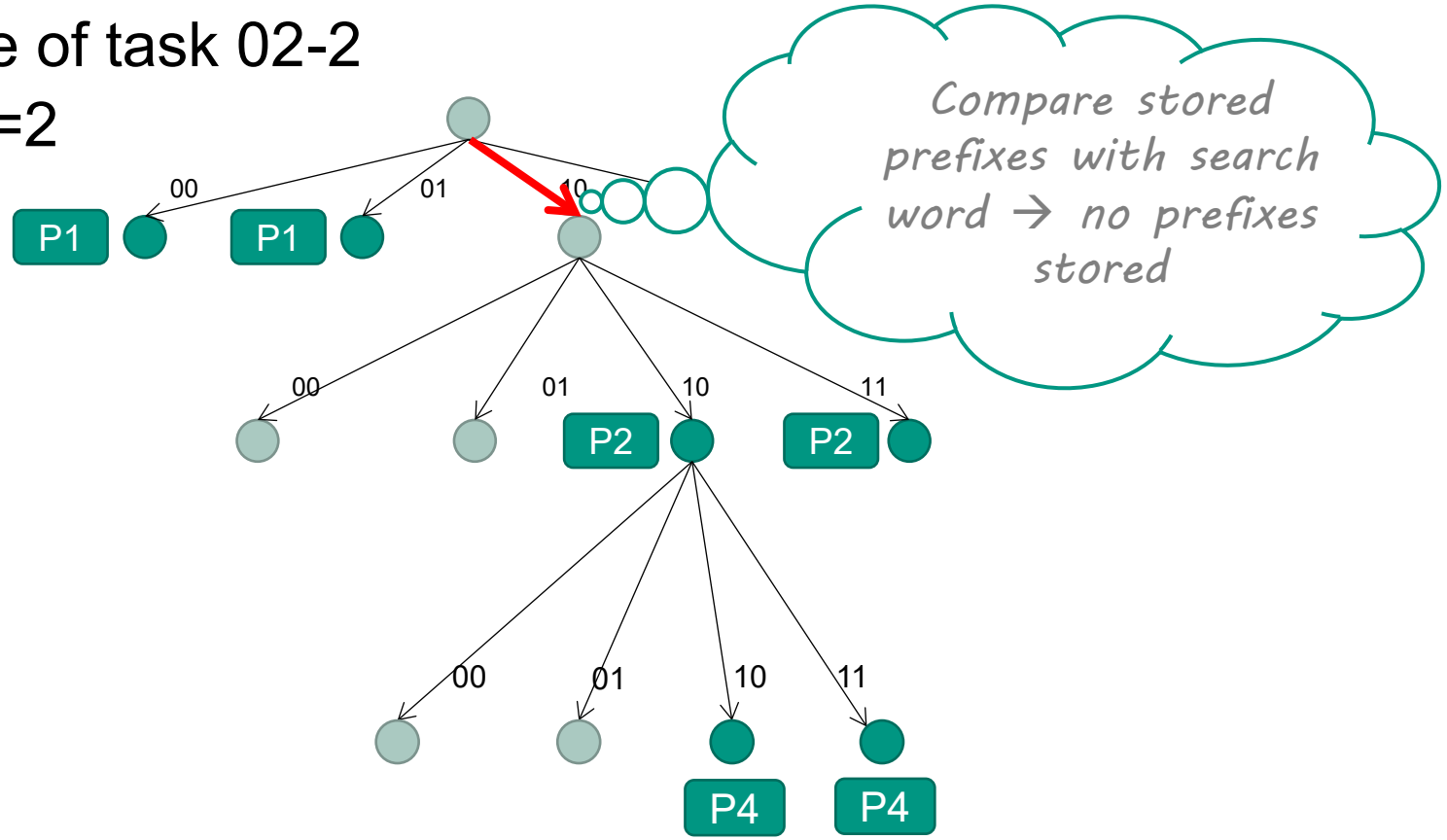


Task 02-2


Task 02-2

Multibit Trie Address Lookup

- Example of task 02-2
- Stride k=2



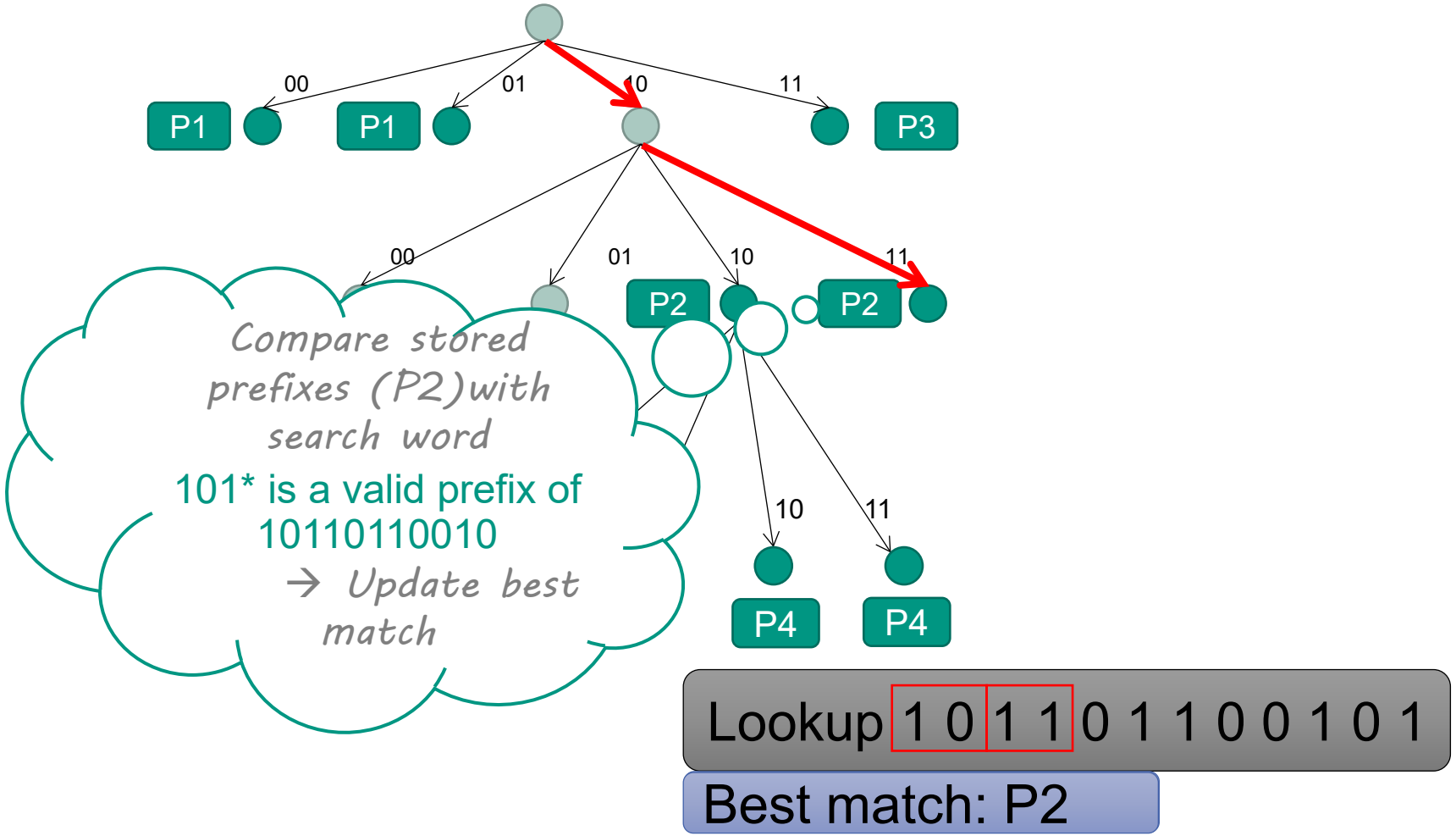
Prefixes		Egress Port
P1	00* 01*	11
P2	1010* 1011*	5
P3	11*	22
P4	101010* 101011*	9

Lookup 1 0 1 1 0 1 1 0 0 1 0 1

Best match: None

Multibit Trie Address Lookup

- Stride k=2

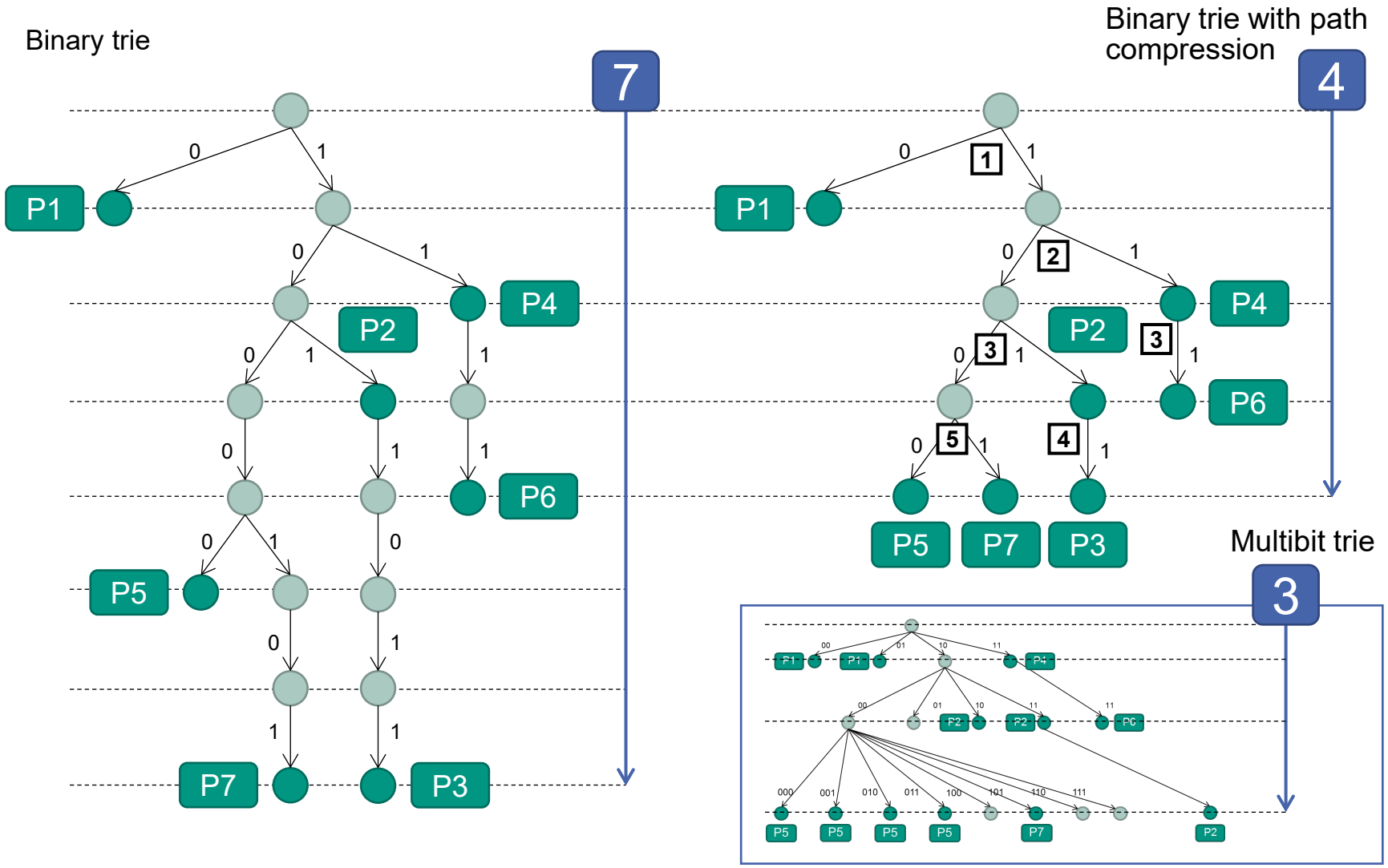


Prefixes		Egress Port
P1	00* 01*	11
P2	1010* 1011*	5
P3	11*	22
P4	101010* 101011*	9

Lookup 1 0 1 1 0 1 1 0 0 1 0 1

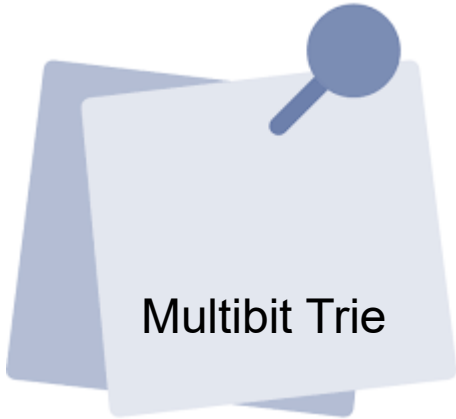
Best match: P2

Maximum Depth Compared to Binary Trie



Homework

Homework 02-04



2.3.2 Hash Tables

Hash Tables

- Objective
  - Improve lookup speed
  - Hash tables can perform lookup in O(1)
  - However: longest prefix match only with hash table doesn‘t work
- Instead: use an additional hash table
  - Hash table stores results of trie lookups
    - E.g., destination IP address 109.21.33.9 → output port 2
  - Significant improvement for large forwarding tables
- For each received IP packet
  - Does an entry for destination IP address exist in hash table?
    - Yes → no trie lookup
    - No → trie lookup
  - Works well if addresses show „locality“ characteristics
    - I.e., most IP packets are covered by a small set of prefixes
    - Not applicable in the Internet backbone

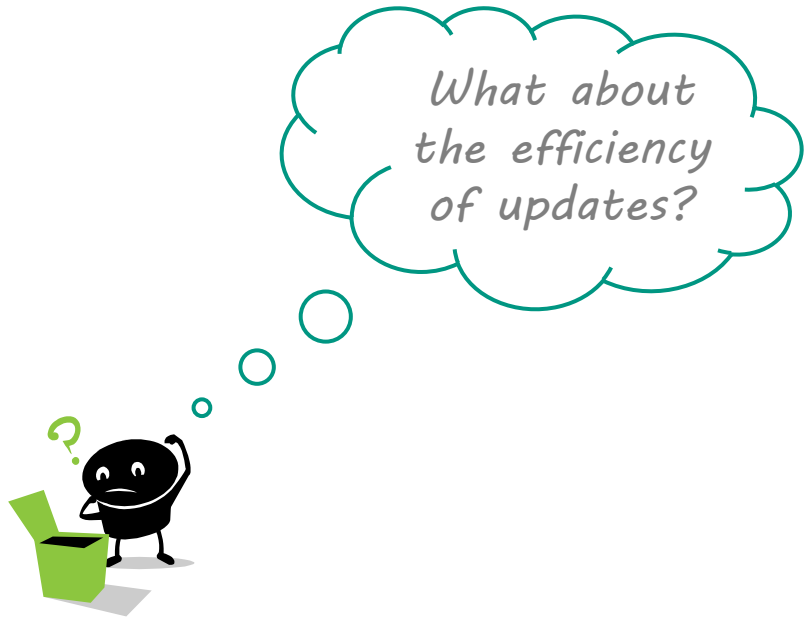
2.3.3 Longest Prefix Matching in TCAM

RAM-based Access

- Basic idea
  - Read information with a single memory access
  - Use destination IP address as RAM address

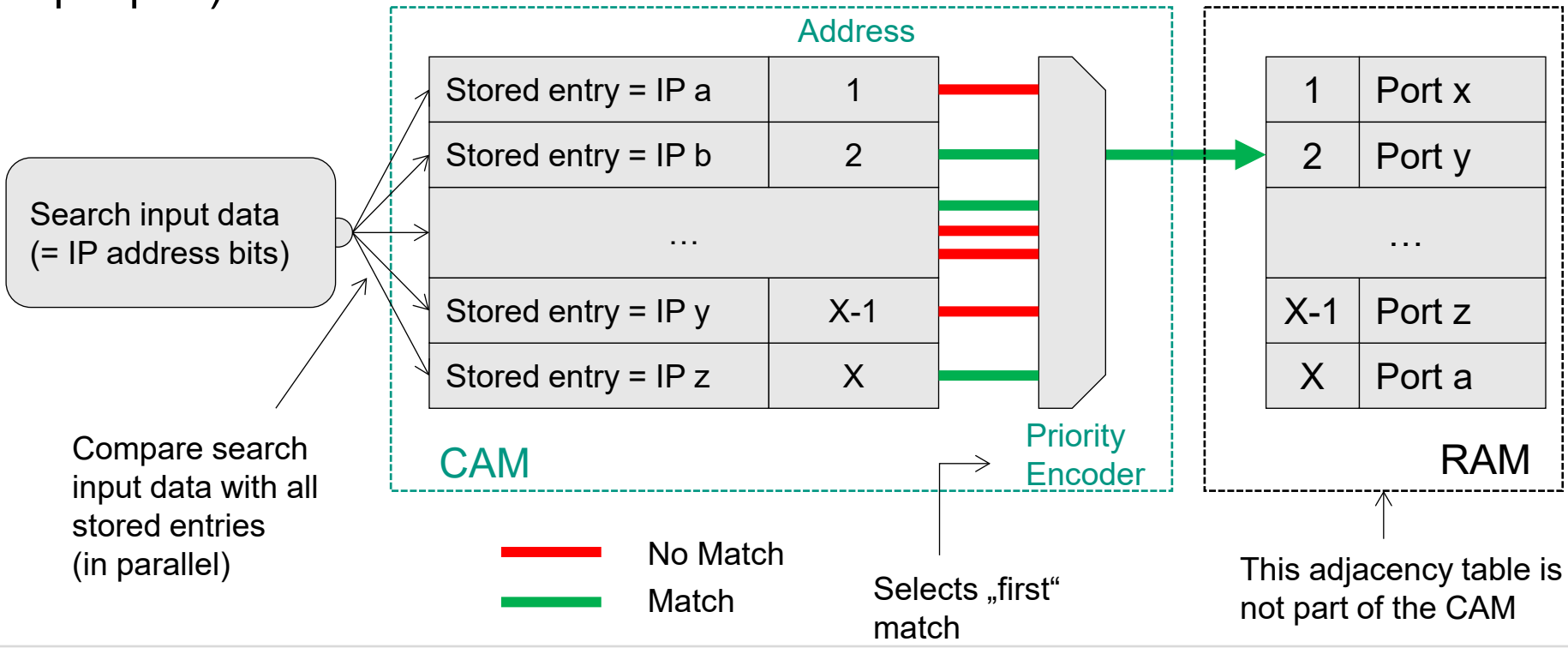
- Observation
  - IPv4 addresses with length of 32 bit
    → requires $4\ GByte\ (2^{32} - 1)$ RAM
  - IPv6 addresses with length of 128 bit
    → requires $\sim 3{,}4 \times 10^{29}\ GByte$ RAM

  - Independent of number of prefixes in use
  - Waste of memory
    - Required memory size grows exponentially with size of address

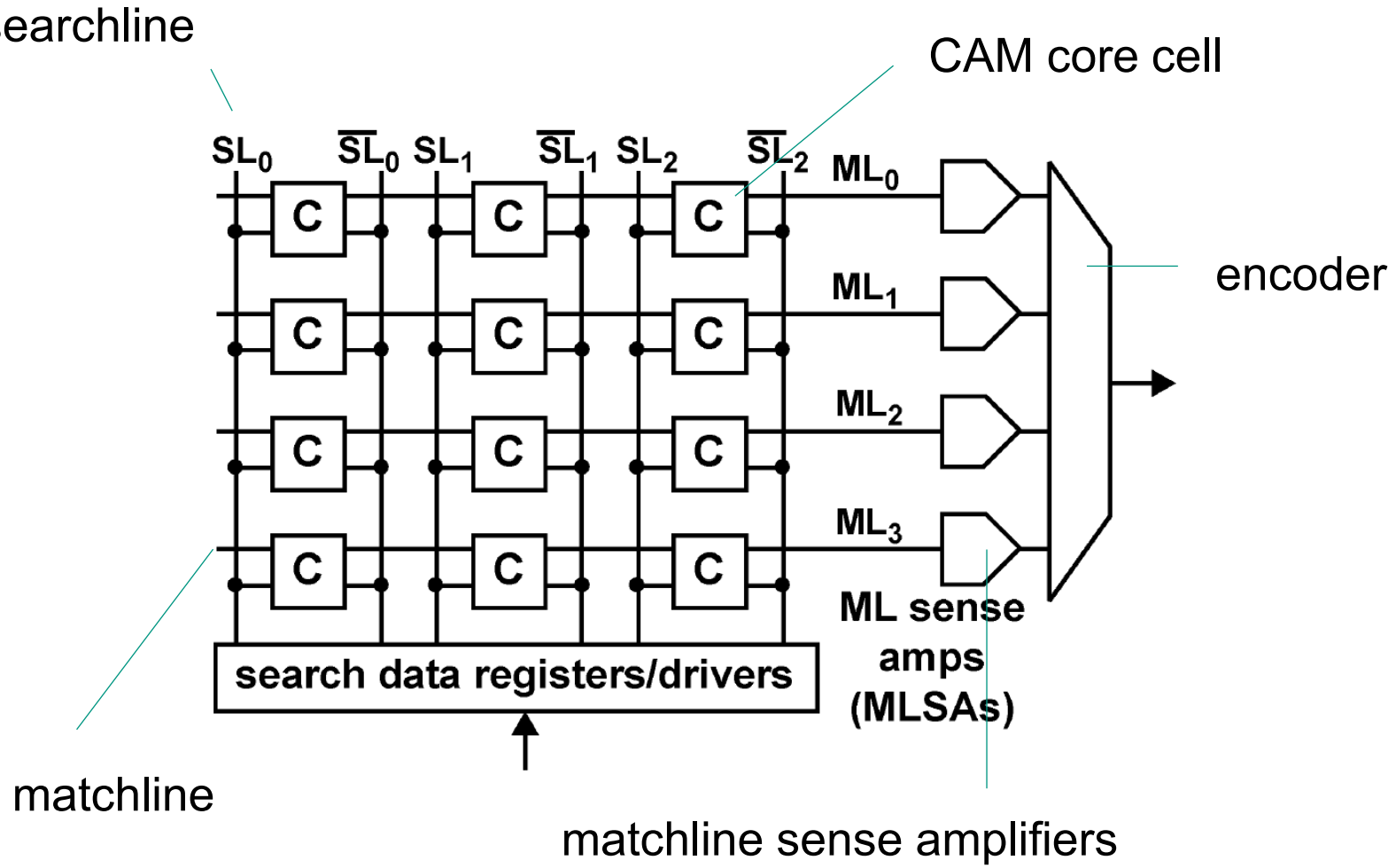


Content-Addressable Memory (CAM)

- RAM: takes address and returns data
- CAM: takes data and returns address
  - CAM can search all stored entries in a single clock cycle
  - Application for networking: use addresses as search input to perform very fast address lookups (IP → output port)



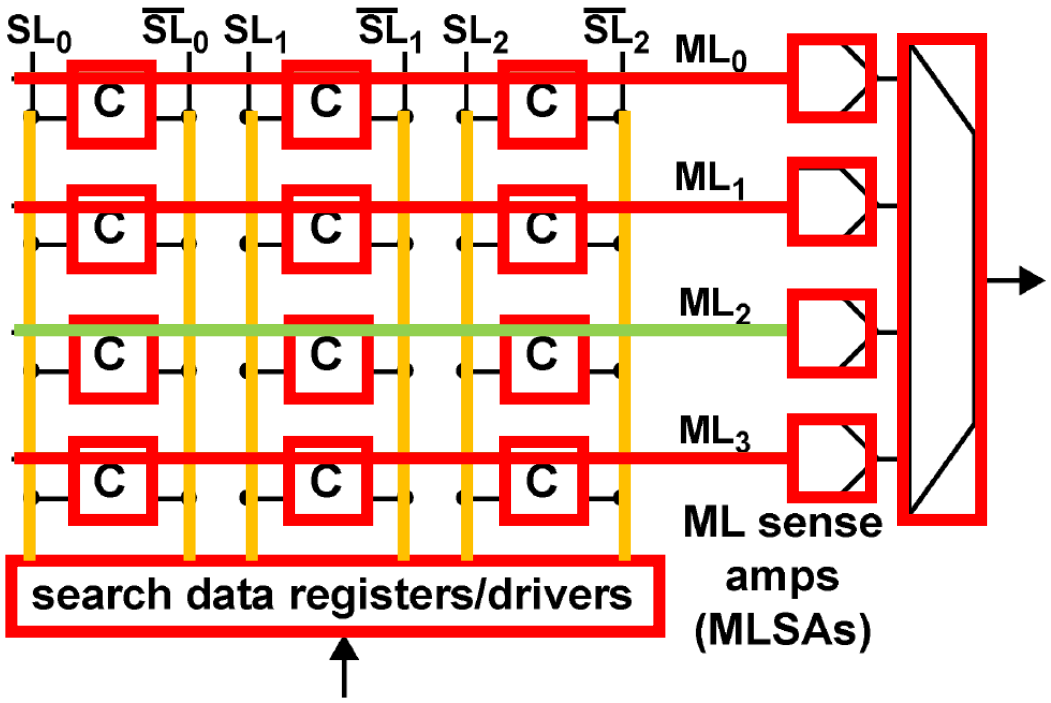
Content-Addressable Memory (CAM)



[PaSh06]

Content-Addressable Memory (CAM)

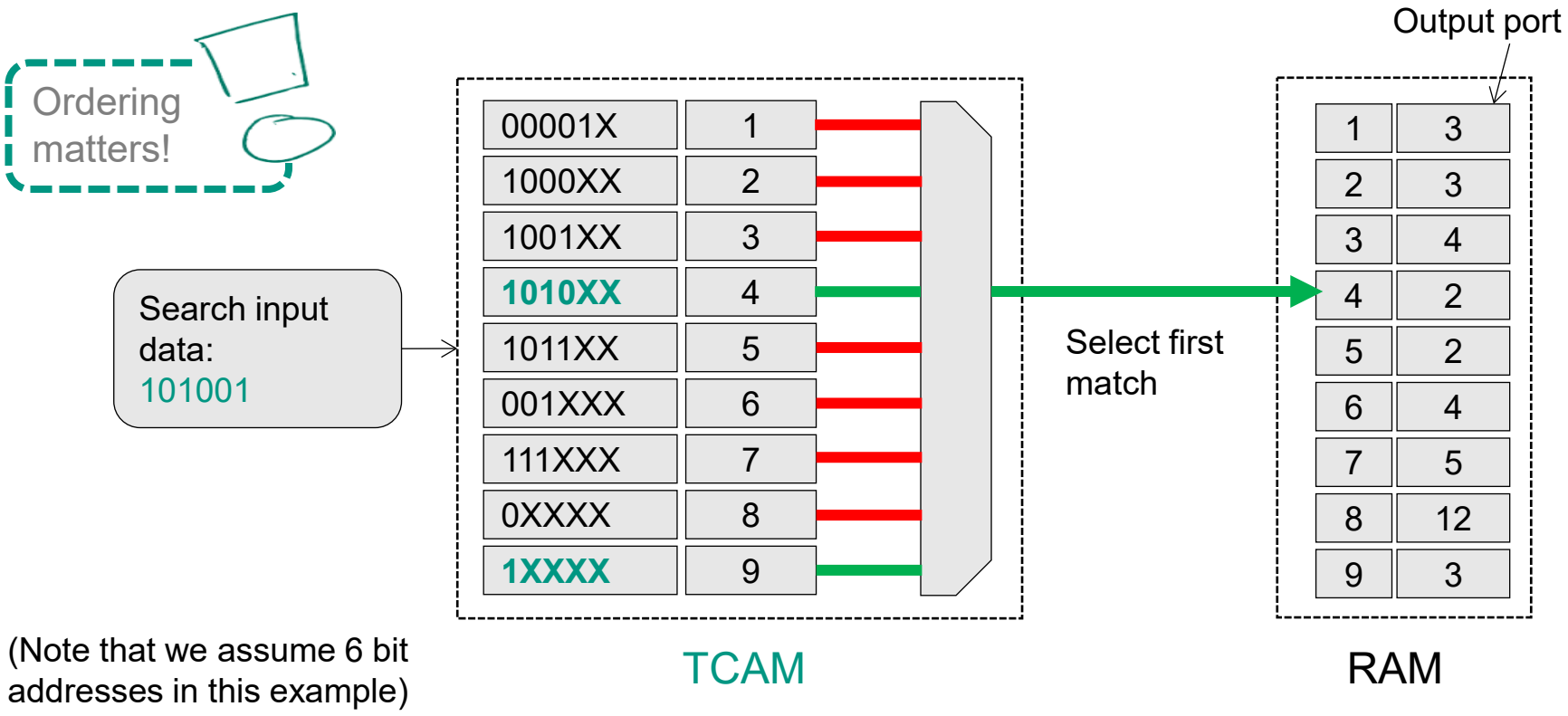
1. Load search data to registers
2. Precharge all matchlines (set to match state)
3. Broadcast search word to all searchlines
4. Core cells compare stored bit with search bit
5. All matchlines with mismatch are set to miss state
6. Matchline sense amplifier detects conditions
7. Encoder selects final matchline



[PaSh06]

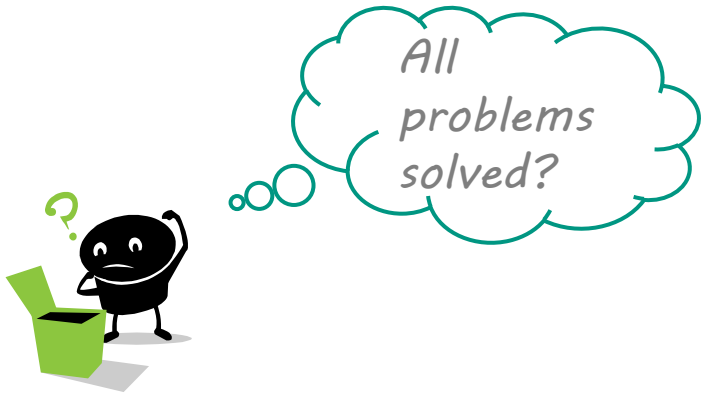
Ternary CAM (TCAM)

- Binary CAM (support for 0 and 1) useful for complete address lookups
- Ternary CAM is an extension that supports a „Don‘t Care“ State (x)
  - Allows longest prefix matching
  - Prefixes are stored in the CAM sorted by prefix length



Ternary CAM (TCAM)

- Evaluation
  - Very fast lookups (1 clock cycle)
  - So the perfect solution for longest prefix matching?

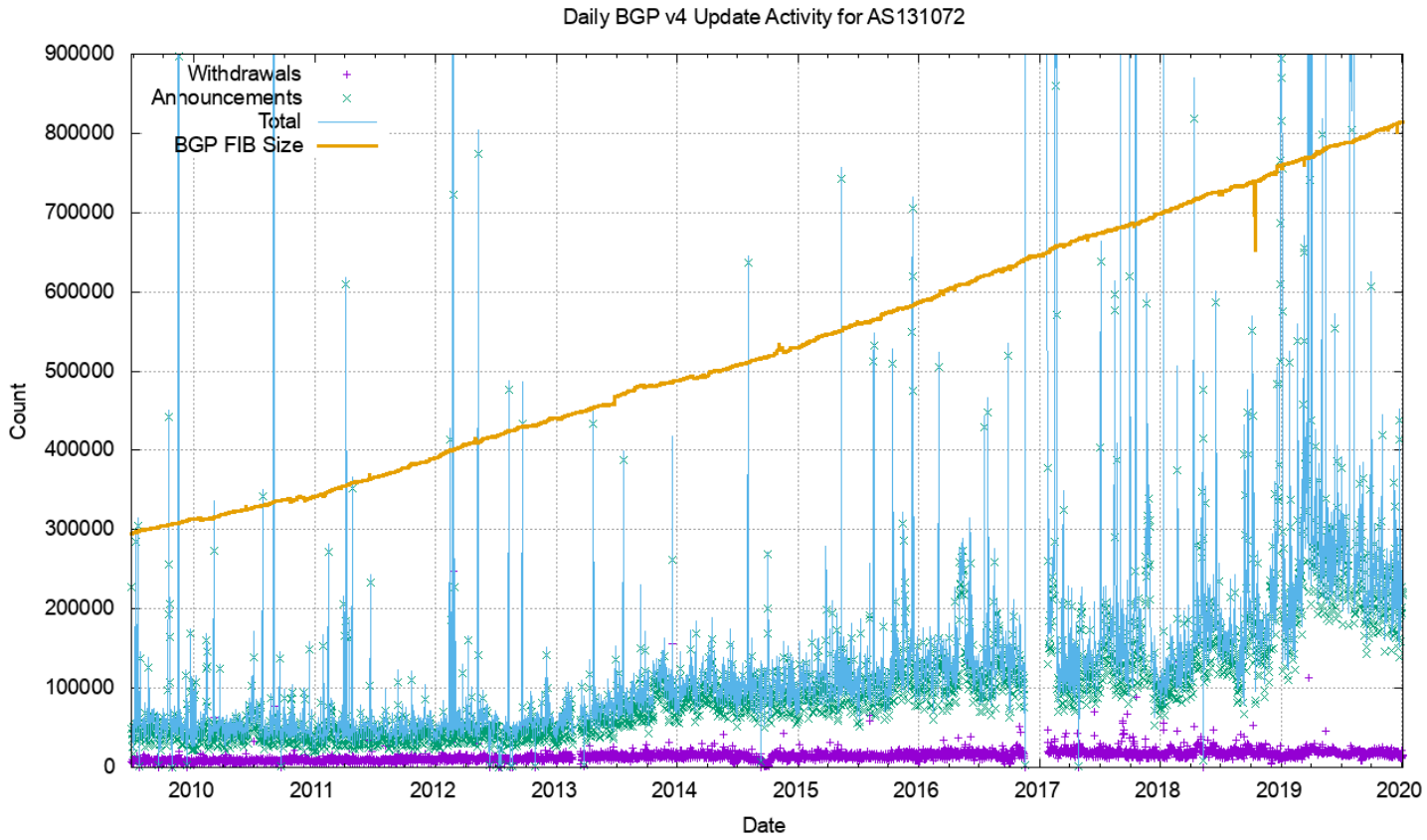


- Problems with TCAM
  - High energy demand
    - All search words are looked up in parallel
    - Every core cell is required for every lookup
  - High cost / low density
    - TCAM requires 2-3 times the transistors compared to SRAM
  - Longest matching prefix requires strict ordering of prefixes in the TCAM
    - New entries can require the TCAM to be „re-ordered“
    - This can take a significant amount of time

→ Severe scalability limitations

Dynamic Changes

- Prefixes stored in forwarding table will change over time
  - Topology changes
  - Broken links
  - Policy updates
  - Traffic engineering
  - ...
- Example
  - 100.000+ updates per day for an Internet core router
  - Can lead to thousands of updates per second in the forwarding table (in peak hours)



Source: https://blog.apnic.net/2020/01/15/bgp-in-2019-bgp-churn/

[Hous20]

TCAM Updates

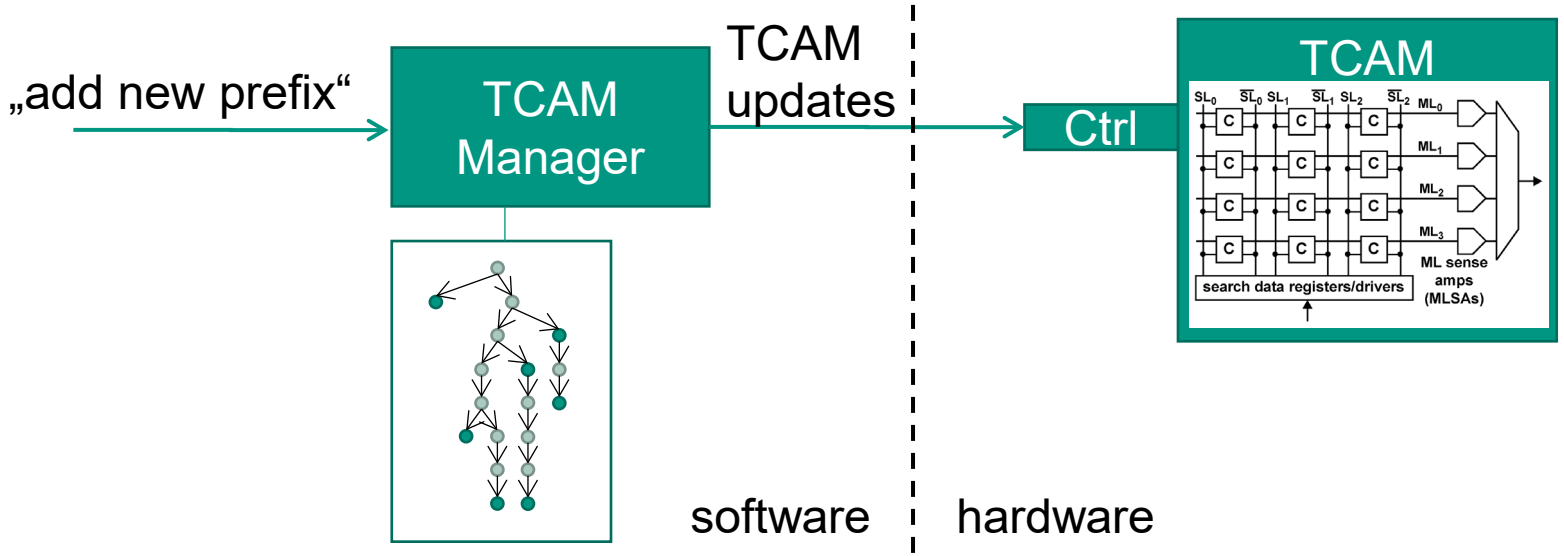
- Entry is added or removed
  - More precisely: CAM core cells in a match line are flushed (hardware operation)

- Need to be efficient and consistent
  - Efficient
    - Single insertion (i.e., new prefix) can lead to multiple TCAM updates
    - Goal: keep #updates as small as possible
    - Different update strategies can be deployed
  - Consistent
    - All packets are forwarded correctly during and after update
    - Important when multiple new prefixes are added in a batch
    - Not further discussed here

- Updates are handled by TCAM manager component

TCAM Manager

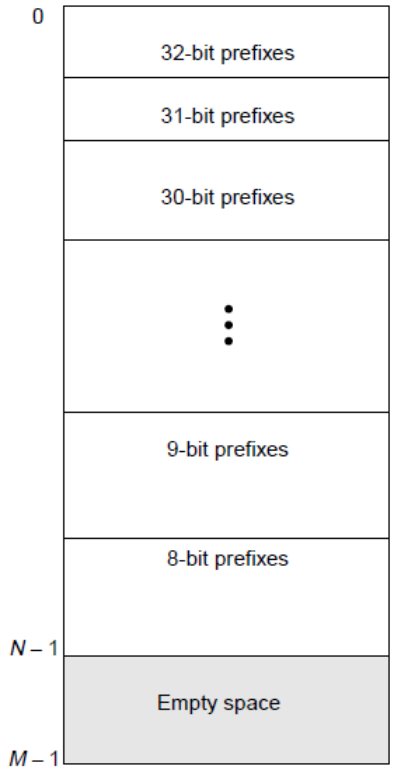
- Piece of software, usually running on switch CPU
- Communicates with hardware controller that manages CAM core cells
- Stores supplementary data structures about current content of TCAM
  - E.g., based on trie
  - Required to calculate TCAM updates
  - Performance depends on update strategy



TCAM Update Strategy

- Simple update strategy
  - Assume all „empty space“ is at bottom of TCAM
  - Shift down all entries with shorter prefix than new prefix
  - Insert new prefix

- Evaluation
  - Time complexity in number of TCAM updates is O(M) where M denotes size of TCAM



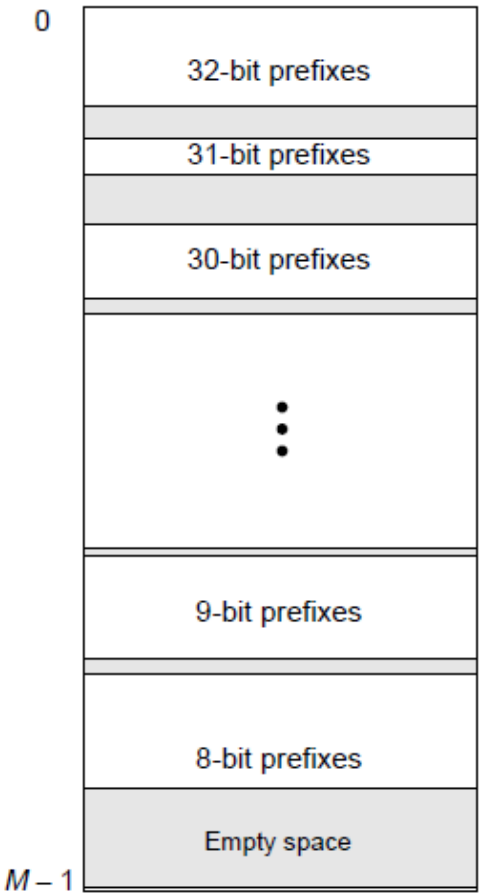
- Is O(M) a problem here?
  - 400 MHz TCAM with 10K entries
    → Update takes 2.5 nanoseconds
  - Worst case: 25 microseconds if all 10k entries must be moved
    - Equals forwarding of nearly 500 64-byte packets on a 10 Gbit/s link
    - Need to be buffered!

[ShGu01, ZLZW20]

TCAM Update Strategy

- Block-based update strategy
  - Organize TCAM in X blocks
    - E.g., one block per prefix length
  - Keep X-1 empty spaces between blocks
  - If block is not fully occupied
    - Insertion can be handled with free space after block

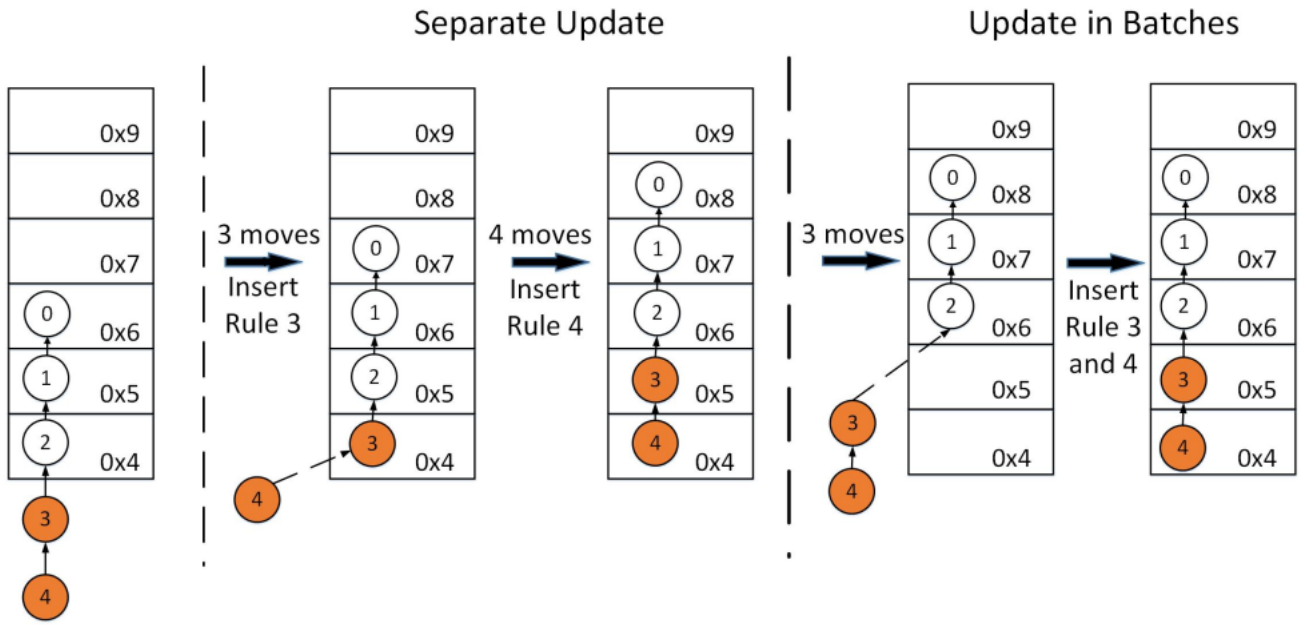
- Evaluation
  - Time complexity in number of TCAM updates is O(M/X)
  - Degenerates to O(M) if empty spaces are filled up



[ShGu01]

More Advanced TCAM Update Strategies

- Update in batches
  - Collect multiple insert operations and update TCAM in one batch
  - Example on the right assumes empty space at the top
  - Only 3 instead of 7 moves required



- And many other approaches...

[ZLZW20]

Homework

Homework 02-05

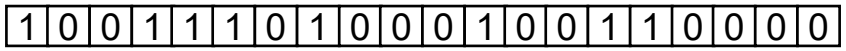
Hybrid Approach (TCAM and Tries)

- Goal
  - Balance power, chip area and performance for large forwarding tables
    - Recall: TCAMs consume large amount of power per lookup
    - Embedded TCAMs are limited in size (~ 4K entries)

- Combine advantages of TCAM and tries
  - Apply large TCAM at root node of the trie
    - Reduces the required depth of the trie, i.e., memory accesses
    - Could, for example, contain 16 most significant bits of IPv4 address
    - Matching TCAM lookup provides pointer to child node
  - Combine with multibit trie for further lookup
    - E.g., multibit trie with variable strides
    - Nodes of upper levels located in on-chip SRAM
    - Nodes of lower levels located in external (off-chip) memory

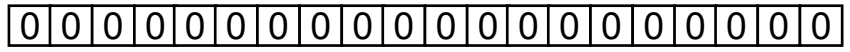
Bloom Filters

- Bloom filter
  - Probabilistic data structure used to test whether an element is member of a set or not
    - False positives are possible
    - False negatives are not possible
- Consists of
  - M-bit array
    - Values either 0 or 1

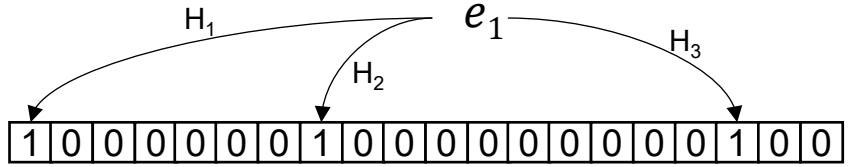


      - Initially all values are set to 0
  - K different hash functions
    - Elements are hashed with the K hash functions
    - The resulting fields in the array are set to 1

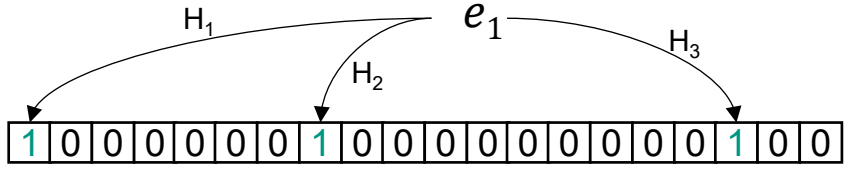
- Example for M=20 and K=3

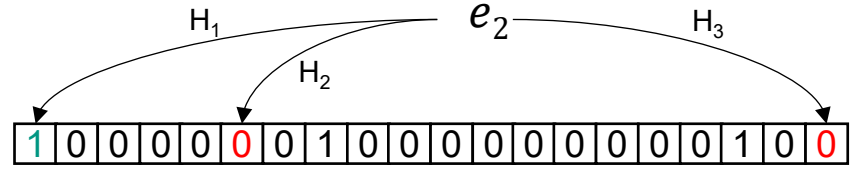


  - Insert element e_1



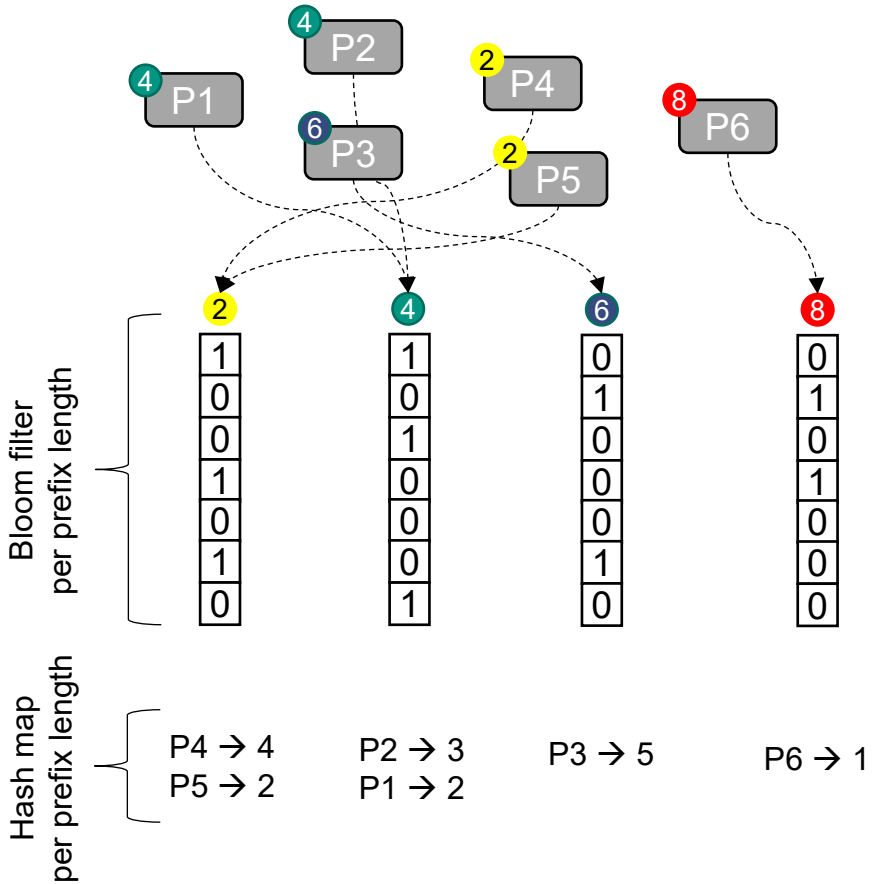
  - Check elements





Longest Prefix Matching with Bloom Filters

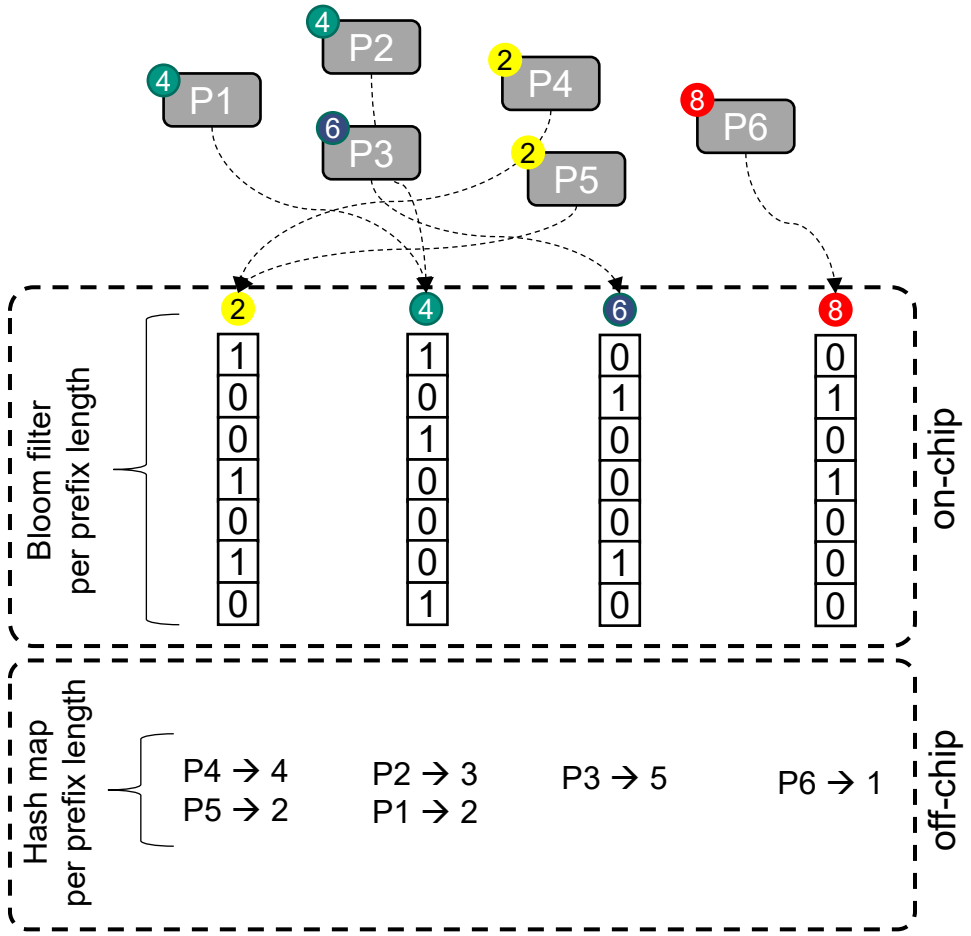
- Given a set of prefixes
  - Sort them into subsets according to prefix length
- For each subset defined by prefix length
  - Create bloom filter
    - Add elements of subset
  - Create a *prefix to next hop* hash map



[Sara03]

Longest Prefix Matching with Bloom Filters

- Given a set of prefixes
  - Sort them into subsets according to prefix length
- For each subset defined by prefix length
  - Create bloom filter
    - Add elements of subset
  - Create a *prefix to next hop* hash map
- Bloom filters implemented in on-chip memory
  - Possible due to memory efficiency of bloom filters
  - Lookup very fast in parallel
- Hash maps implemented in off-chip memory
  - Slow memory access
  - Look-ups expensive regarding duration

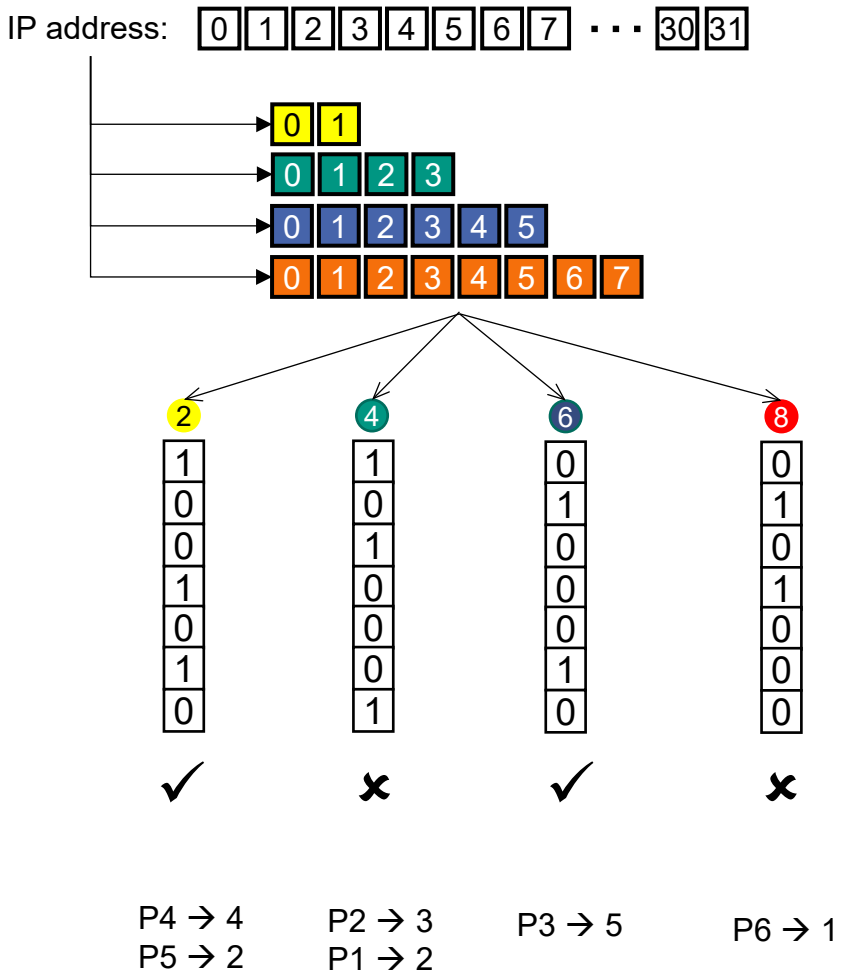


[Sara03]

Longest Prefix Matching with Bloom Filters

- Given an IP address to look up
  - Derive prefixes according to stored prefixes‘ length
- For every prefix, check according bloom filter
- For all successful prefix checks
  - Iterate over prefixes in descending (length) order
    - If prefix is in hash map (necessary due to false positives!)
      - Return next hop
    - Else
      - Check next smaller prefix length
- 2MB of embedded RAM achieves
  - One hash probe on average
  - Two hash probes in worst case

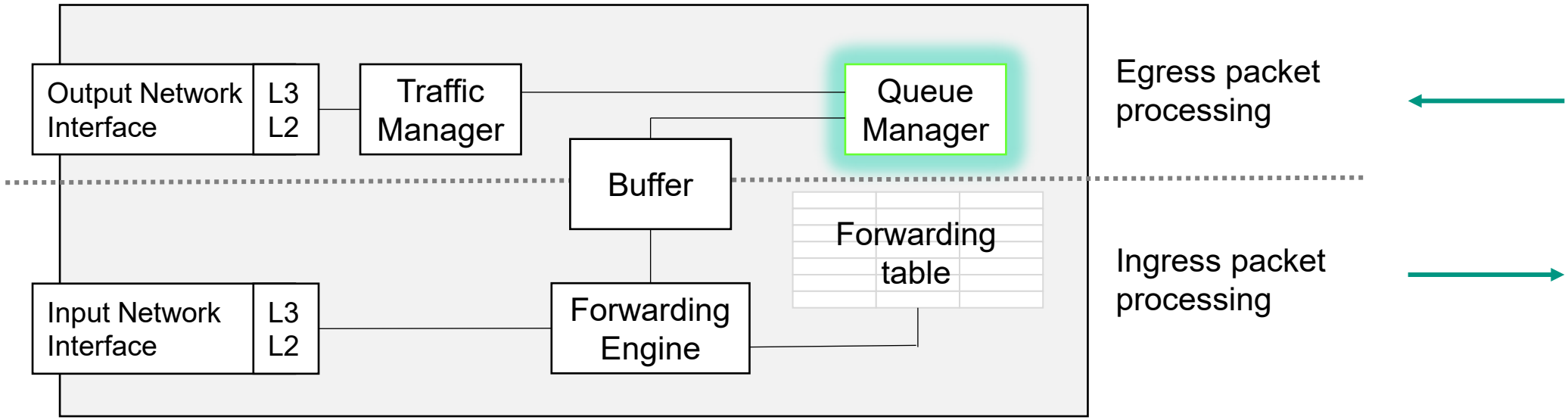
Depends on false positive rate of bloom filters



[Sara03]

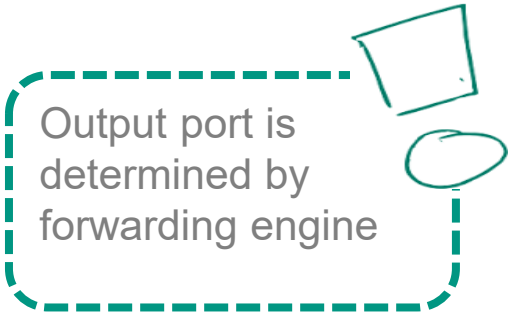
2.7 Packet Scheduling

Network Interface Card

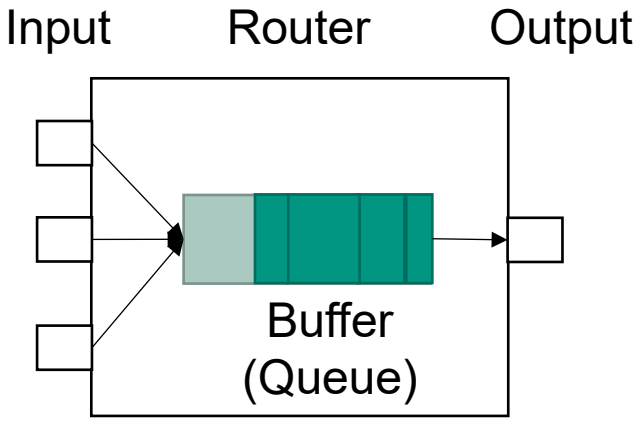


Packet Scheduling

- Basic situation
  - Router receives more traffic from input ports than it can transmit across the desired output port



- Possible decisions
  - Drop all received overload packets
    - Leads to high packet loss rate
    - May have sever consequences, e.g., regarding TCP traffic
  - Temporarily buffer packets
    - Packets are not (immediately) dropped
    - Packets are delayed (waiting time in buffer/queue)
    - But: when to send which packet?
      → Task of packet scheduling



Packet Scheduling

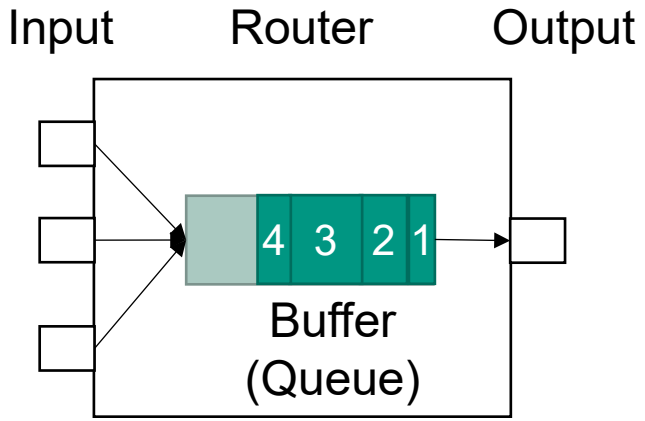
- Two important aspects
  - How does packet scheduling perform in terms of bounded delay?
    - → it affects packets of both data path and control path
  - Can packet scheduling be implemented efficiently?
    - → ideally packets are forwarded in line speed

FIFO Queueing

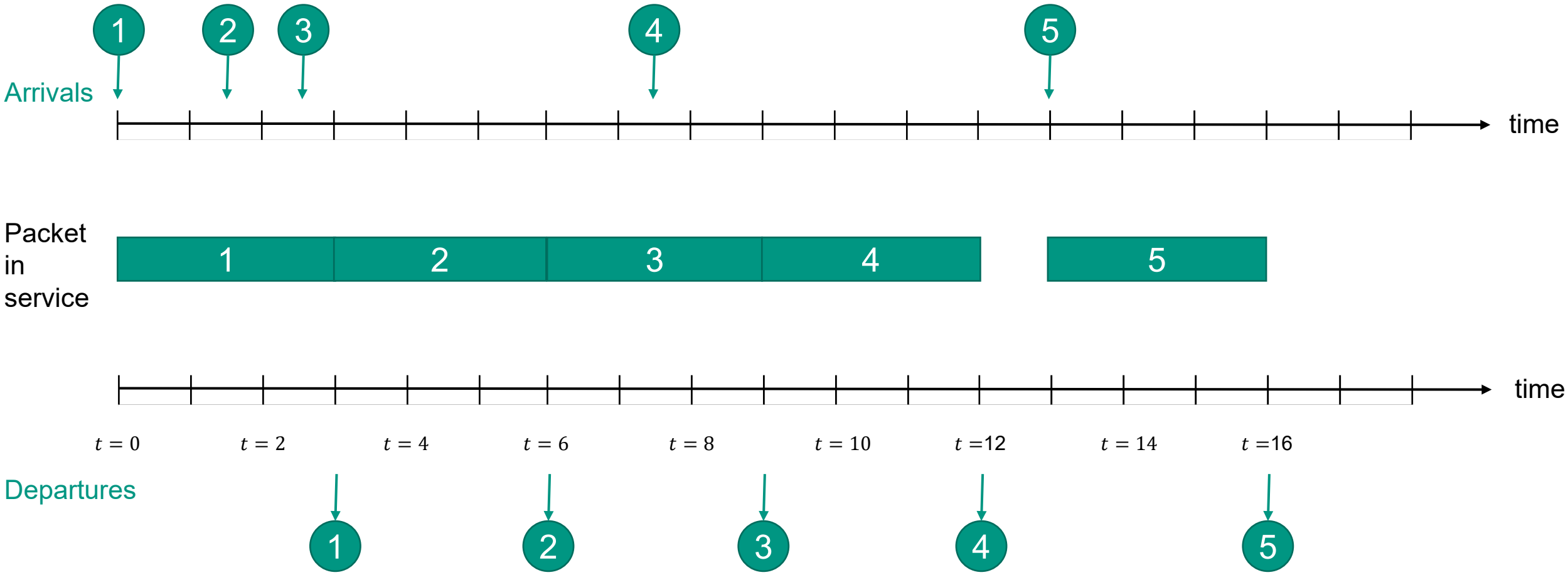
- FIFO: First-in, first-out
  - Equivalent to FCFS: first-come, first-served
  - Packets are transmitted in order of their arrival at the router

- Advantages
  - Easy to implement
  - Low overhead
  - Predictable upper bound D_{max} on delay
    - d: link data rate, b: maximum buffer size
    - $D_{max} \leq \frac{b}{d}$

- Disadvantage
  - No discrimination of packets possible



FIFO Queueing: Example

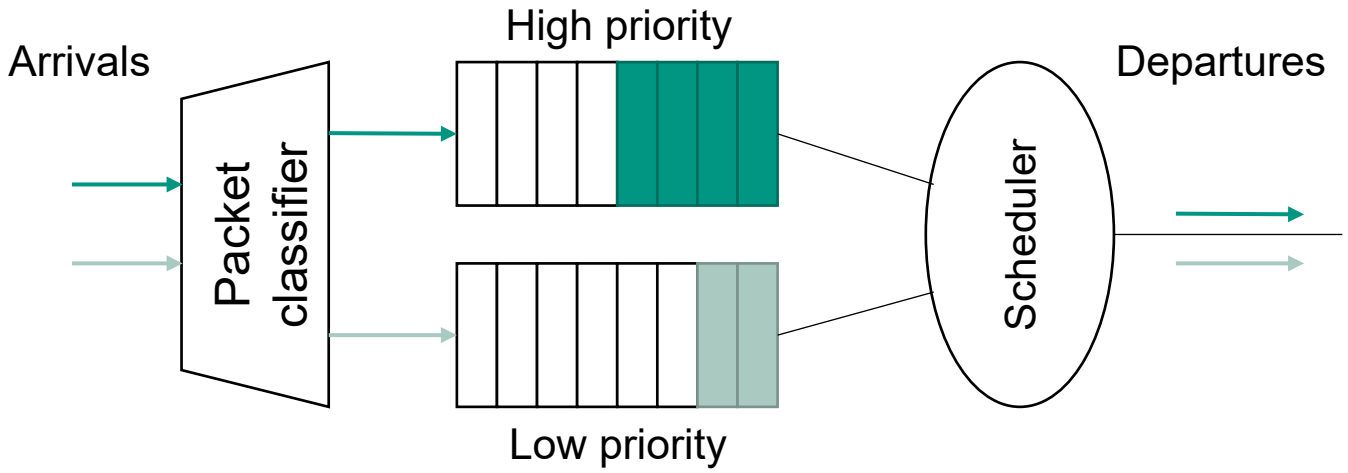


Assumption: all packets need three units of time to be transmitted

[KuRo22]

Priority Queueing

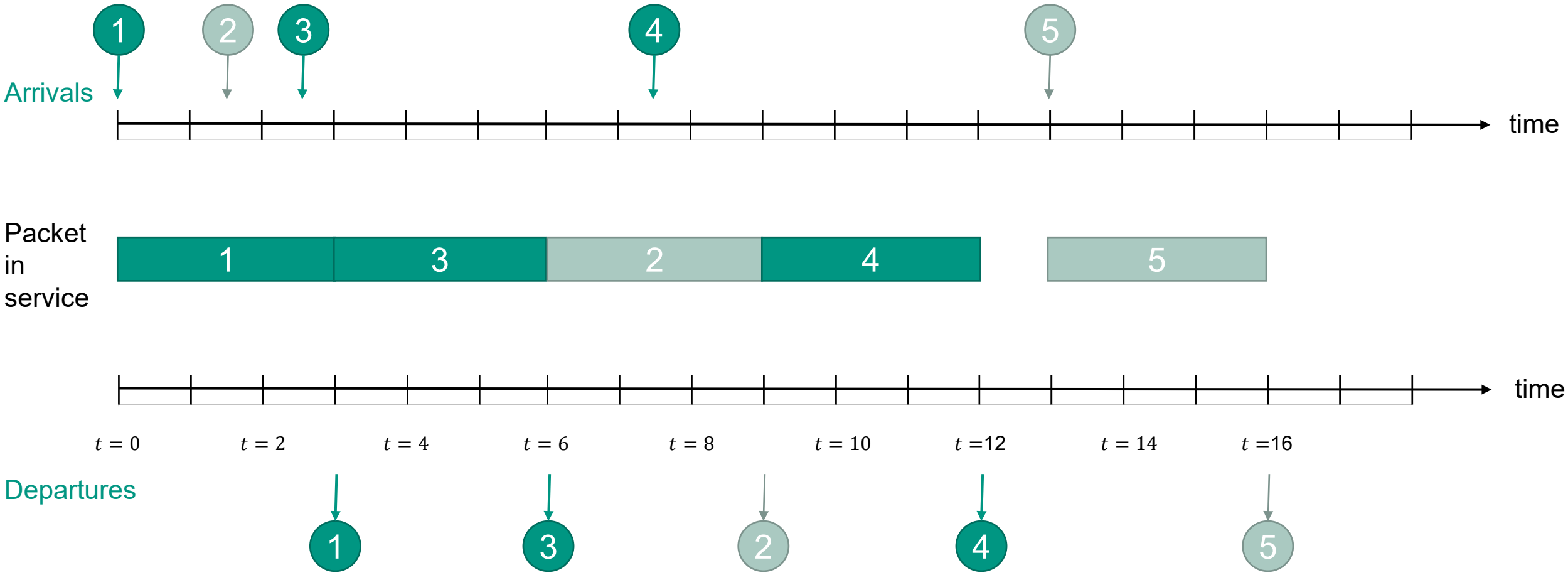
- Basic idea
  - Classify arriving packets into priority classes
  - Own queue for each priority class
    - Within queue typically FIFO scheduling
- Enables packet differentiation
  - Preferential treatment of real-time traffic
  - Preferential treatment of routing packets
  - …



- Strict priority queueing
  - Selection of packet to be transmitted
    - Non-empty queue with highest priority
  - → Starvation of lower priority traffic possible
  - → No guarantees regarding packet delay
- Rate-controlled priority queueing
  - Limits amount of high-priority traffic
    - Example: 25% of outgoing link capacity can be used by high-priority traffic

Sometimes strict priority queueing desirable. For example prioritize packet that carry routing information during periods of congestion

Priority Queueing: Example



Assumption: all packets need three units of time to be transmitted
Assumption: non-preemptive priority queueing, i.e., packet is not interrupted when higher priority packet arrives

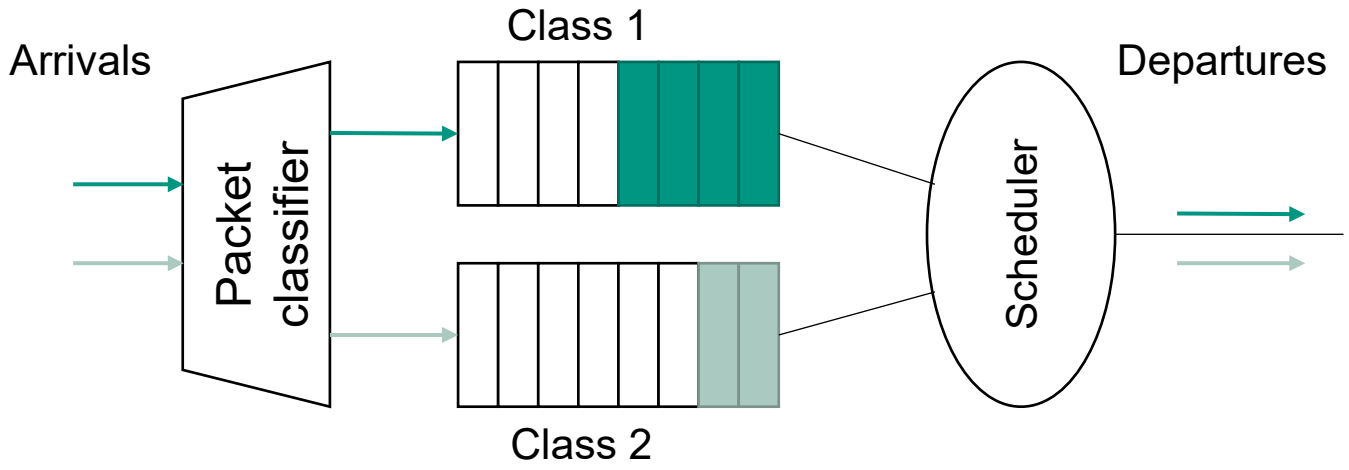
[KuRo22]

Packet Classification

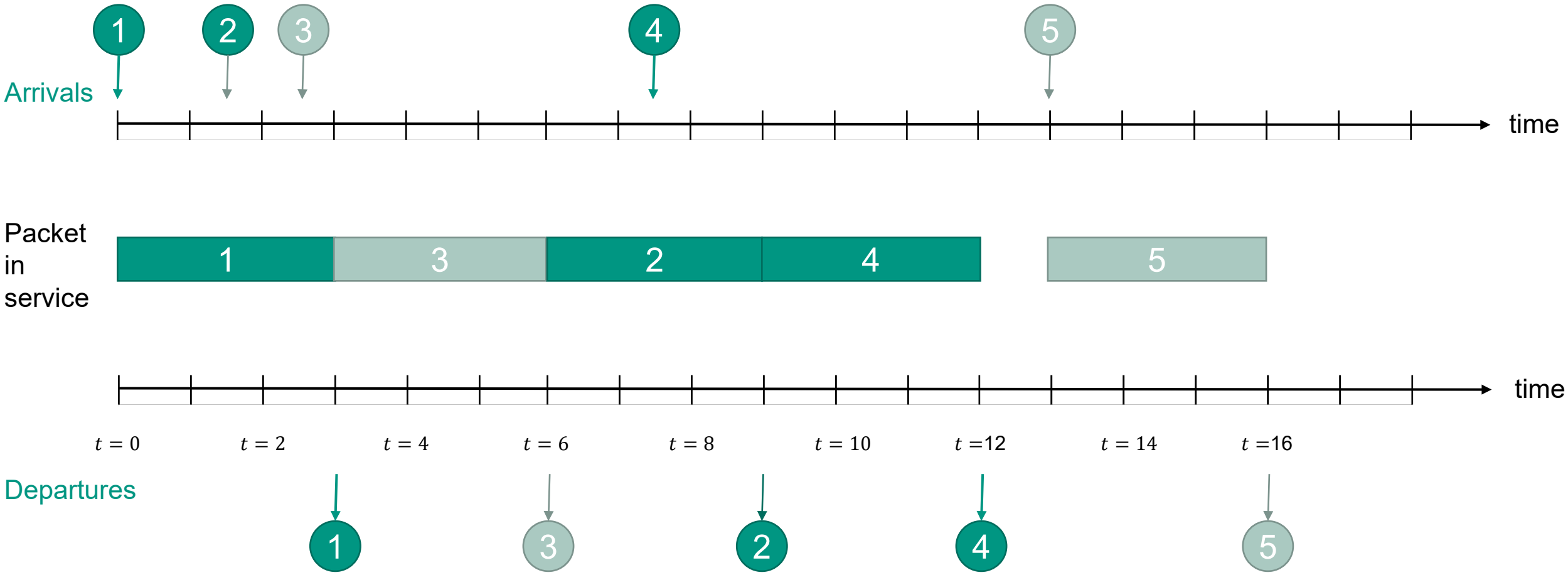
- Goal of packet classification
  - Differentiating packets in order to treat them differently
- Examples that require packet classification
  - Provide service guarantees for different types of traffic
    - E.g., different paths through network for real-time traffic and other data traffic
  - Different traffic can be charged different prices
    - E.g., Voice over IP charged higher price compared to regular data traffic
  - Preventing malicious attacks
    - Malicious users can overload the network affecting other customers

Round Robin

- Basic idea
  - Arriving packets are classified into different classes
  - Classes are served in a circular manner



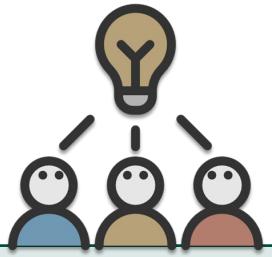
Round Robin: Example



Assumption: all packets need three units of time to be transmitted
Assumption: work-conserving, i.e., as long as packets are available for transmission the link will not be idle

[KuRo22]


Task 02-3
Scheduling

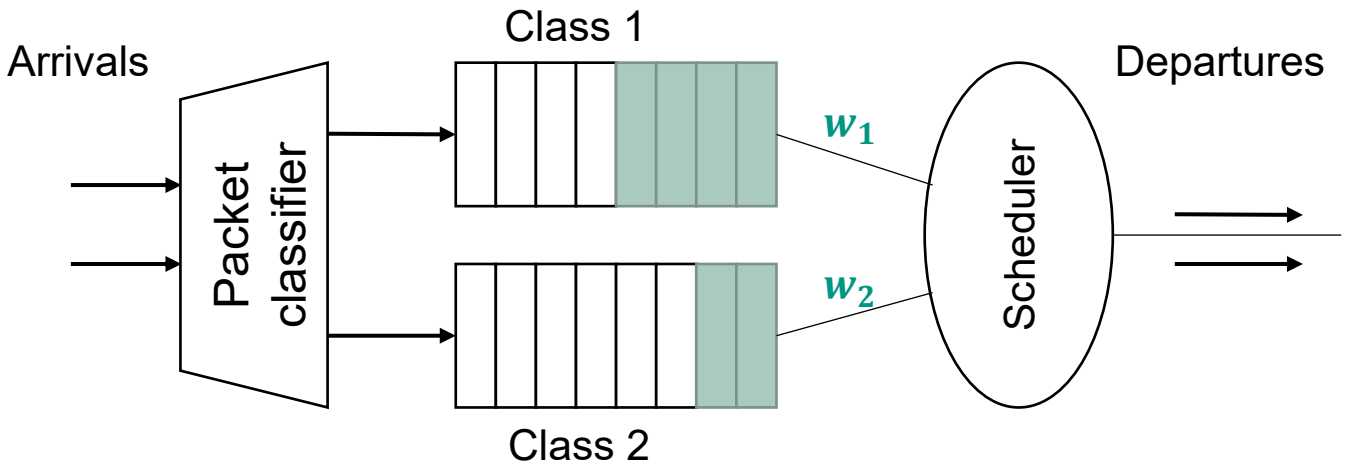


Task 02-3

Weighted Fair Queueing

- Difference to Round Robin
  - Each class c_i is assigned a weight w_i
  - Each time class c_i is served it receives a fraction of service equal to $w_i / \sum w_i$
    - Sum is taken over all classes that also have packets to be transmitted

- Advantage
  - Guaranteed minimum data rate
    - Let c be data rate of link,
      minimum data rate for class c_i achieved by WFQ: $c \cdot (w_i / \sum w_i)$

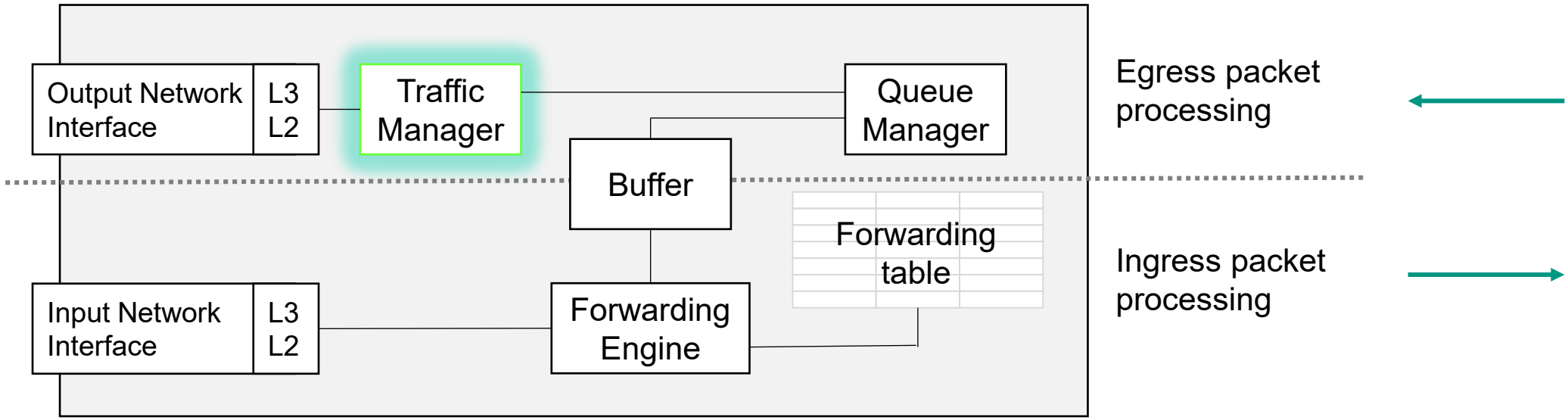


This is an idealized view. Not considered: packets are discrete, packet transmission will not be interrupted by other packet

[KuRo22]

2.8 Traffic Shaping

Network Interface Card



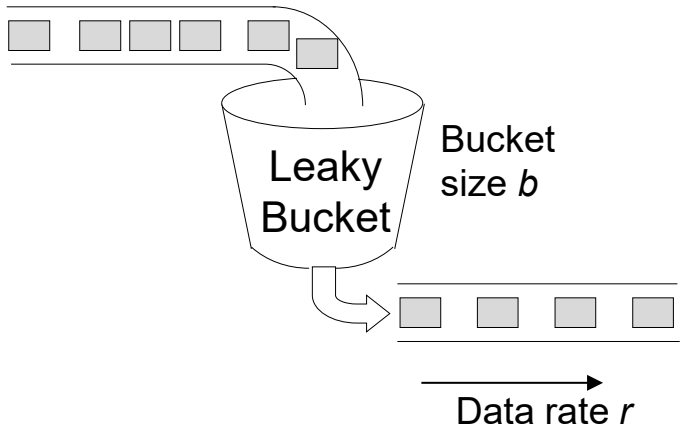
Traffic Shaping

- Basic characteristics
  - Takes place at output interface of a router
  - Regulates rate and volume of traffic admitted to the network
    - Ensures that traffic is in accordance with service level agreements (SLAs)

- Two predominant methods
  - Traffic smoothing
    - Eliminates bursts
    - Presents traffic with constant transmission rate to the network
    - Implementation: leaky bucket algorithm
  - Traffic burst shaping
    - Shapes bursts of predetermined size
    - Averages over a time window
    - Implementation: token bucket algorithm

Leaky Bucket

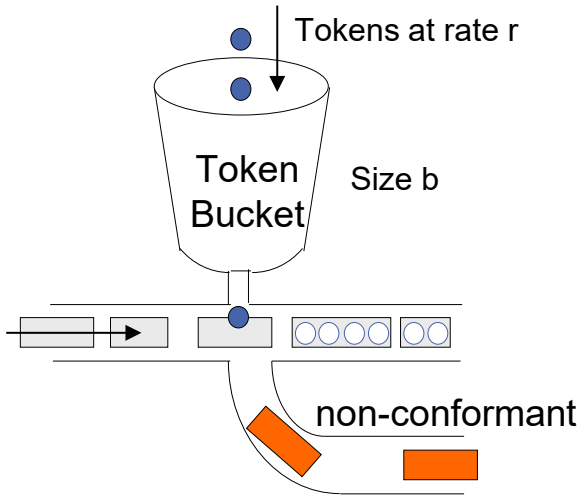
- Enforces constant transmission rate
  - Independent of burstiness of received traffic
- Concept
  - Packets are injected into bucket
    - Bucket size per flow: b
    - FIFO queueing within bucket
    - If bucket is full, new packets are discarded
  - Bucket has hole at the bottom
    - Packets are drained through hole with constant data rate r



- Implementation
  - Bounded FIFO queue
  - Timer
  - Counter X
- Example calculation
  - First packet in queue has size p
  - Timer expires every t seconds
    - Counter is incremented by ${}^{r}/_{t}$ ($X = X + {}^{r}/_{t}$)
    - If $X \geq p$
      - Packet is transmitted
      - Counter updated: $X = X - p$
      - … subsequent packets may be transmitted immediately if X is still large enough
      - … otherwise wait until bucket is full enough

Token Bucket

- Basic characteristics
  - Allows desired level of burstiness
    - In a short time interval
  - Limits maximum burst size b_{max}
  - Limits average data rate
    - In the long term
- Concept
  - Fixed size bucket of tokens
    - Tokens are added into bucket at constant rate
    - Token represents predefined unit of bytes
    - If bucket is full, new tokens are discarded
  - Newly received packet
    - Transmitted if sufficient tokens are available
    - Otherwise
      - Dropped,
      - Enqueued for subsequent transmissions, or
      - Transmitted, but marked as non-conformant
- Implementation
  - Per flow: counter C and timer
  - Timer expires every token interval $T = 1/r$
    - $C = C + b_{max}/T$, always with $C \leq b_{max}$
  - Packet arrives (packet size = p)
    - $C \geq p$: packet transmitted, $C = C - p$

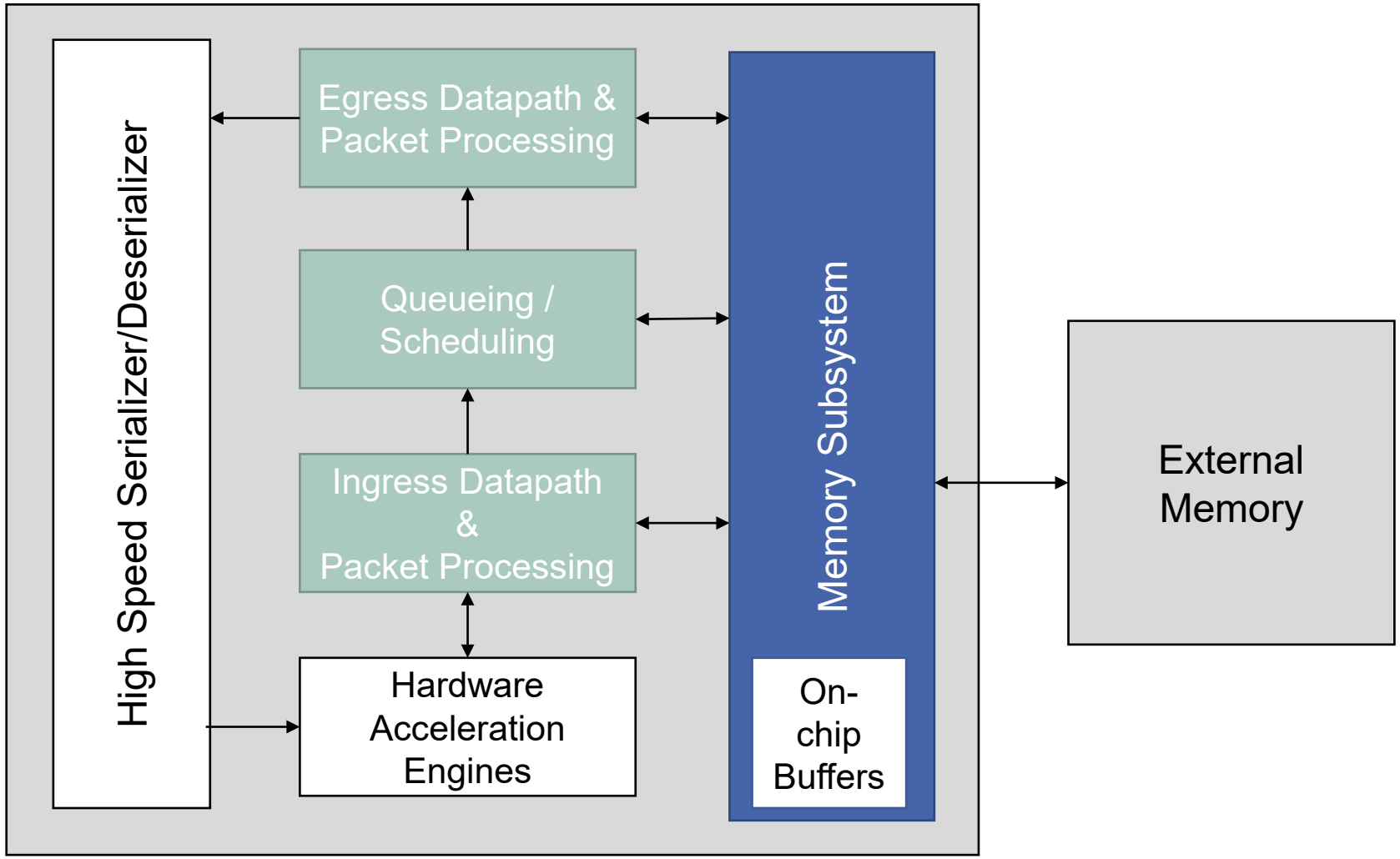


2.9 Networking Chips/Smart NICs

Networking Chips

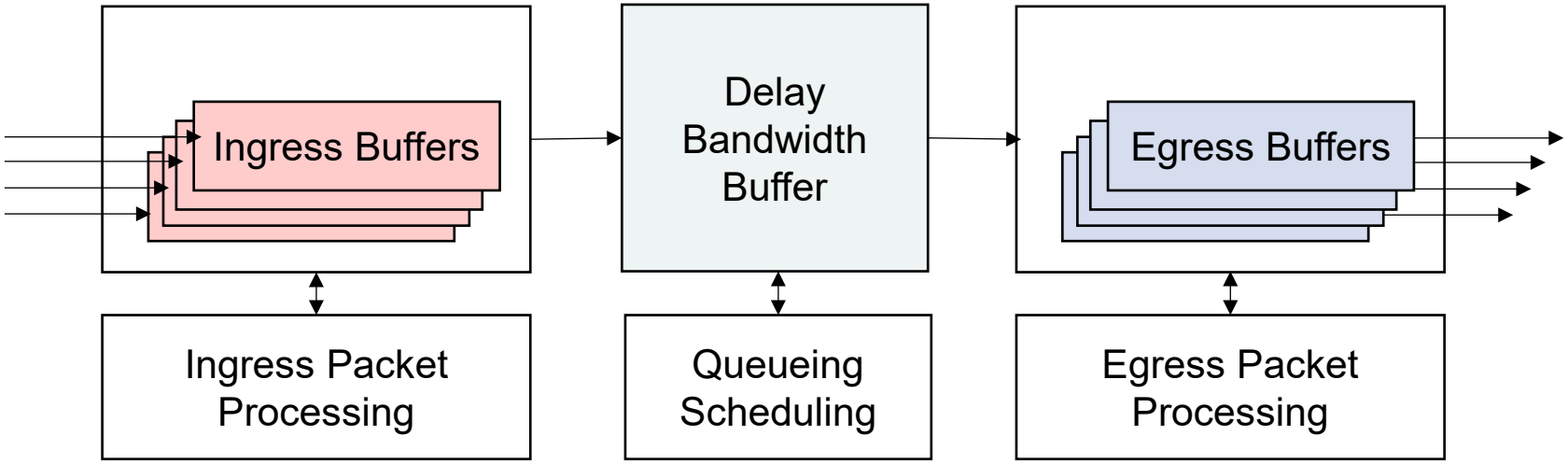
- Description
  - Can receive network traffic (i.e., packets) from multiple input ports
  - Can process some or all L2/L3/L4 headers in the packets
    - Decide on next steps to be taken, e.g., queueing, scheduling
- Performance measured as
  - Gbit/s of traffic it can process at line rate without dropping packets
  - Minimum packet size for meeting this rate
- Different memory technologies
  - Off-chip memory (external)
    - SRAM and HBM (High-Bandwidth Memory)
    - E.g., to buffer packets
  - On-chip memory (internal)
    - SRAM
    - Caches
- Complex subsystems for
  - Longest prefix matching
  - Queueing and scheduling
- High speed serializer/deserializer

Networking Chips: Typical Example



[https://community.juniper.net/blogs/sharada-yeluri/2022/07/01/networking-chips-vs-gpuscpus]

Buffers in Networking Chips

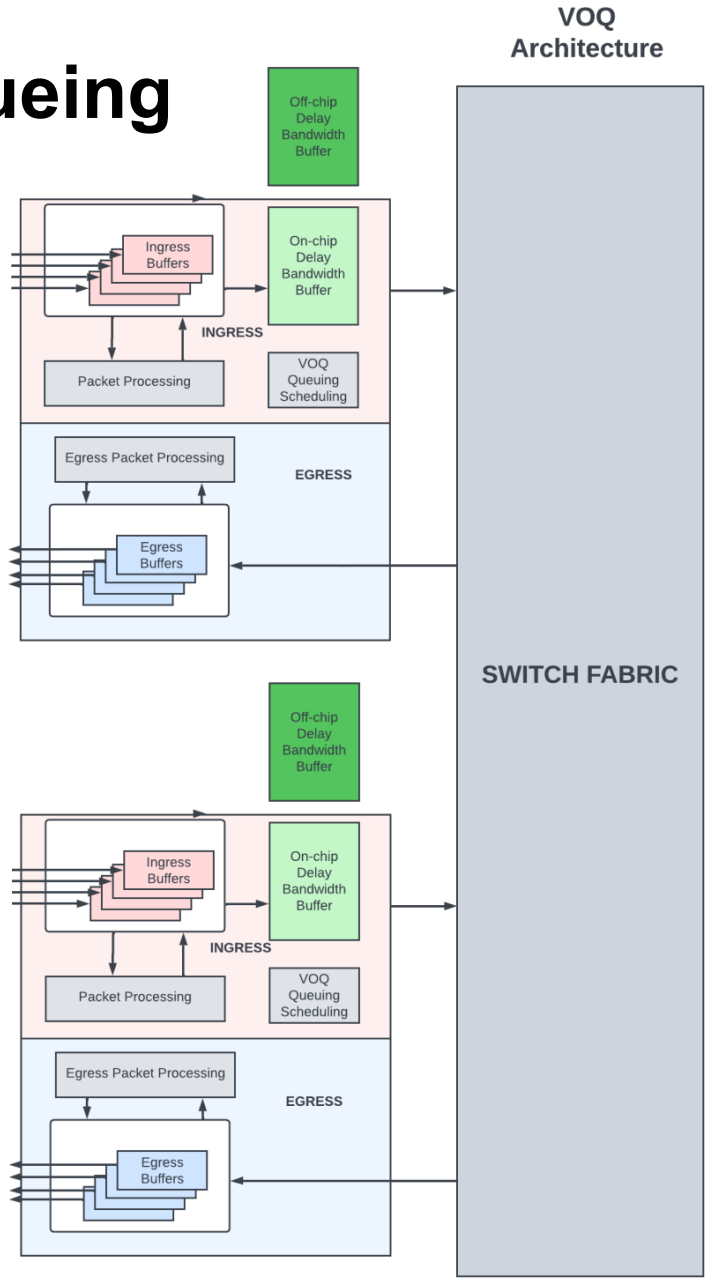


- Input buffers
  - Incoming packets from input ports while packet headers are processed by packet processing
    - Processing latencies are around a few microseconds
  - Priority-aware drops when buffer becomes full
    - Protect critical control packets from being dropped
- Delay bandwidth buffer
  - Deep buffer to absorbe transient congestions
  - Maintain different queues for different flows
- Egress buffers
  - Shallow buffers
  - Input rate should not exceed link data rate of this interface

[https://community.juniper.net/blogs/sharada-yeluri/2023/02/22/sizing-router-buffers?CommunityKey=44efd17a-81a6-4306-b5f3-e5f82402d8d3]

Architecture with Virtual Output Queueing

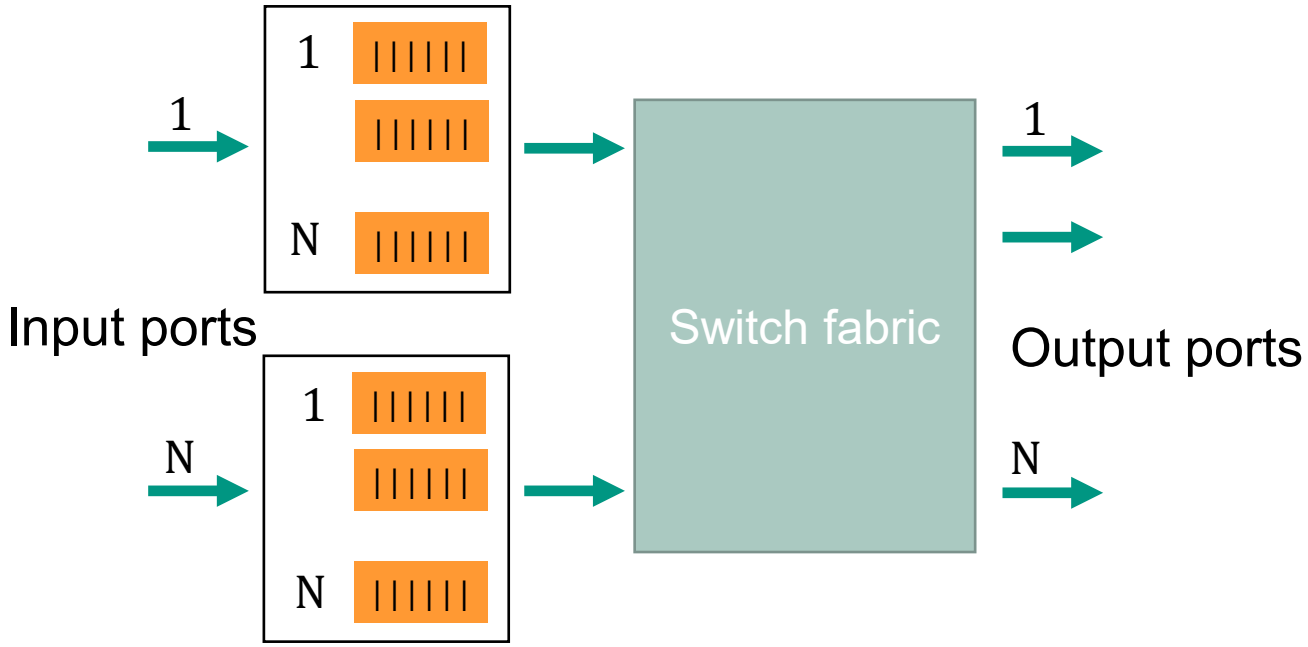
- Delay bandwidth buffering on ingress packet forwarding entity
  - Consists of different types of buffers
    - On-chip buffer
    - Off-chip buffer
  - Packets are queued in virtual output queues



[https://community.juniper.net/blogs/sharada-yeluri/2023/02/22/sizing-router-buffers]

Virtual Output Queuing (VOQ)

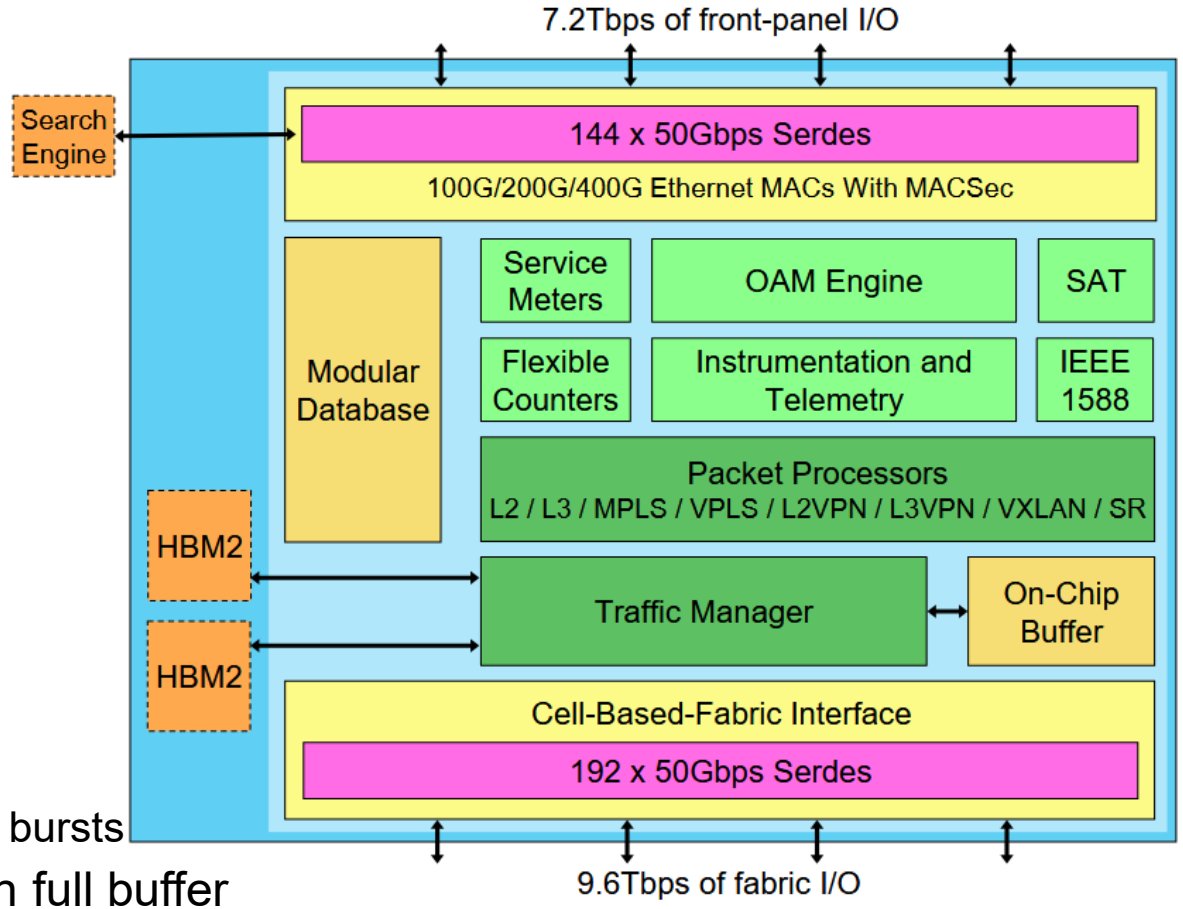
- Goal
  - Overcome Head-of-Line-blocking of input buffering
- Basic concept
  - Split input queue of each input port into N queues
    - With N = number of output ports
  - → virtual output queues at input port
  - … Scheduling becomes more complex



[Medh18]

Networking Chips: Example

- Broadcom Jericho2c+
  - E.g., used in UfiSpace switch (earlier slide)
  - Line rate encryption for all ports
  - Large on-chip tables
    - Up to four million IPv4 routes
  - Buffer
    - On-chip packet buffer: 64 MByte
    - Off-chip packet buffer: 8 GByte (HBM2)
      - 4.8 Tbit/s memory bandwidth
    - Buffers shared between ports of NIC
      - Packets are buffered at ingress NIC
      - Larger burst on single port can be buffered
      - … as long as other ports do not also experience bursts
    - Large buffers increase latency: up to 13ms with full buffer

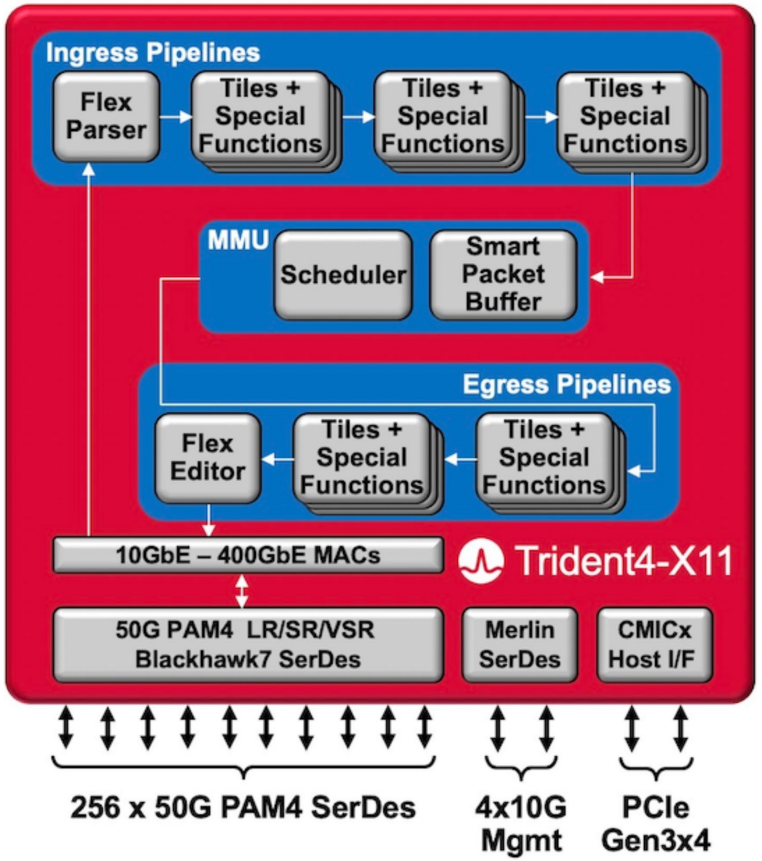


[https://docs.broadcom.com/doc/88850-Report1-PUB]

SAT: Service Activation Testing
OAM: Operations and Maintenance

Networking Chips: Example

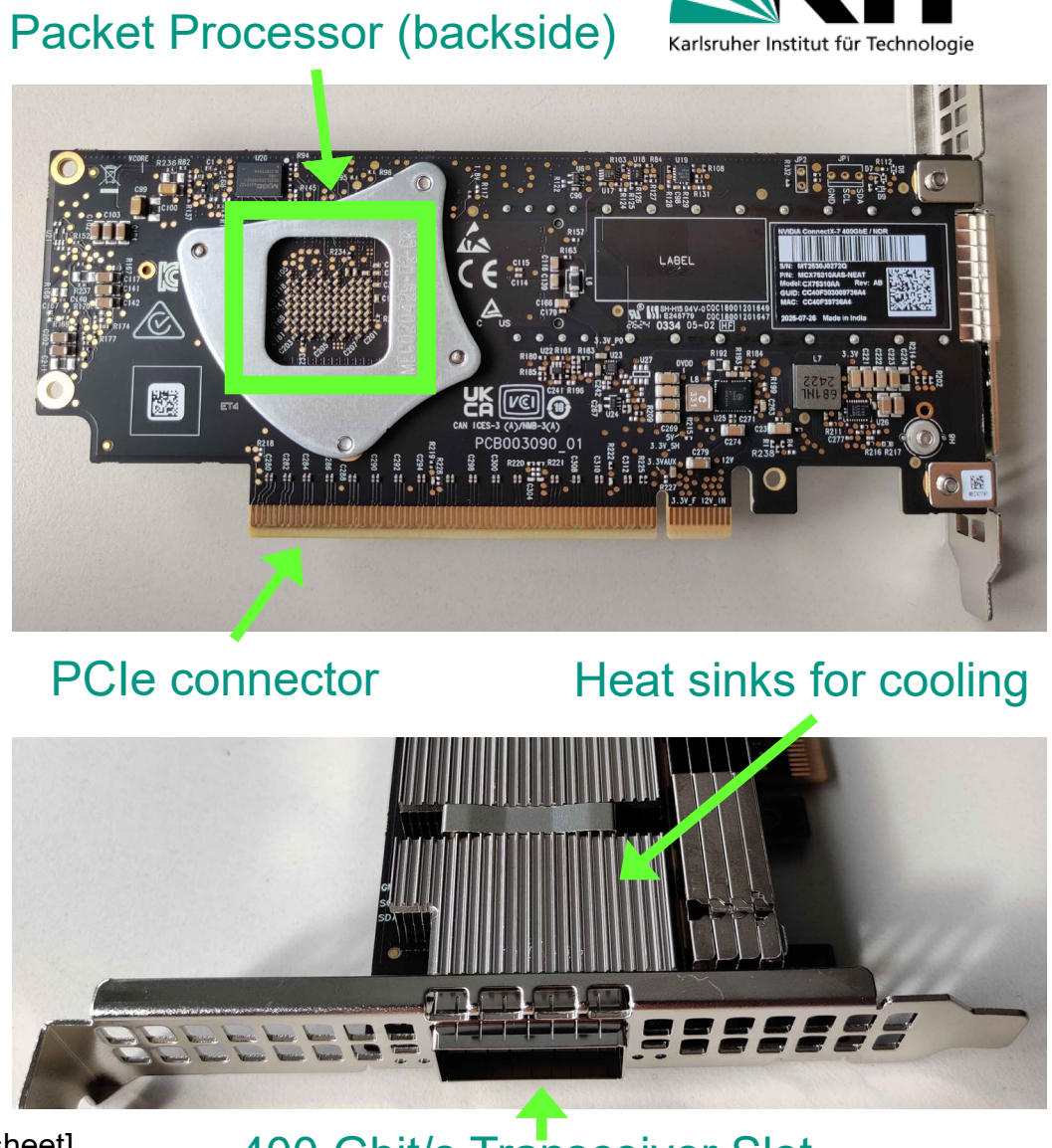
- Broadcom Trident4-X11
  - E.g., used in our Netberg switch (earlier slide)
  - On-chip packet buffer: 132 MByte
  - Off-chip packet buffer: None
    - Trident4 targets datacenters and cloud computing
    - Large congestions much less common within datacenter
      - E.g.: can use DataCenter-TCP (more in Chapter 08)
    - Smaller packet buffers are sufficient
  - Smaller table sizes compared to Jericho2c+:
    - TCAM: 27.000 IPv4 entries
    - SRAM: ~100.000 IPv4 entries
  - Optimized for ultra low latency
    - 800 ns (0.0000008 seconds) from packet entering to packet leaving



[https://bm-switch.com/product/32x400g-netberg-aurora-830-broadcom-trident4-x11-sonic-ready]

Smart NICs: Example

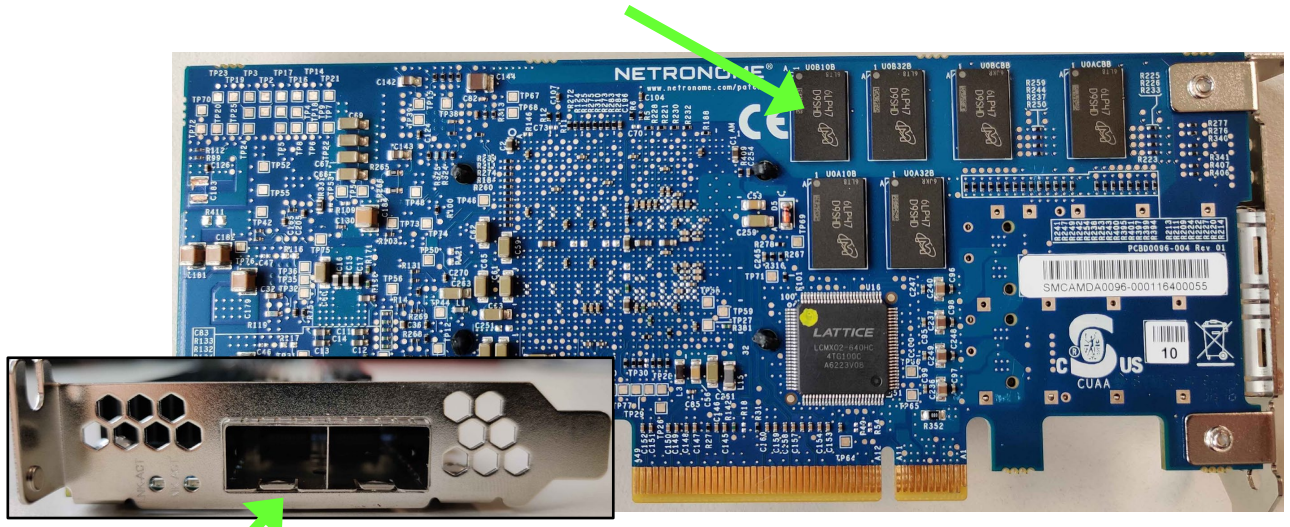
- Nvidia ConnectX-7
  - Used in servers
  - Attached via PCIe slot for high transfer speed
  - Equipped with packet processor
- Packet processing offloaded to NIC
  - Normal CPU too slow to handle data rates of 400 Gbit/s
- Example tasks performed at line rate
  - Flow tracking and exporting
  - IPsec + TLS encryption and decryption
  - Firewalls and packet header inspection
  - …

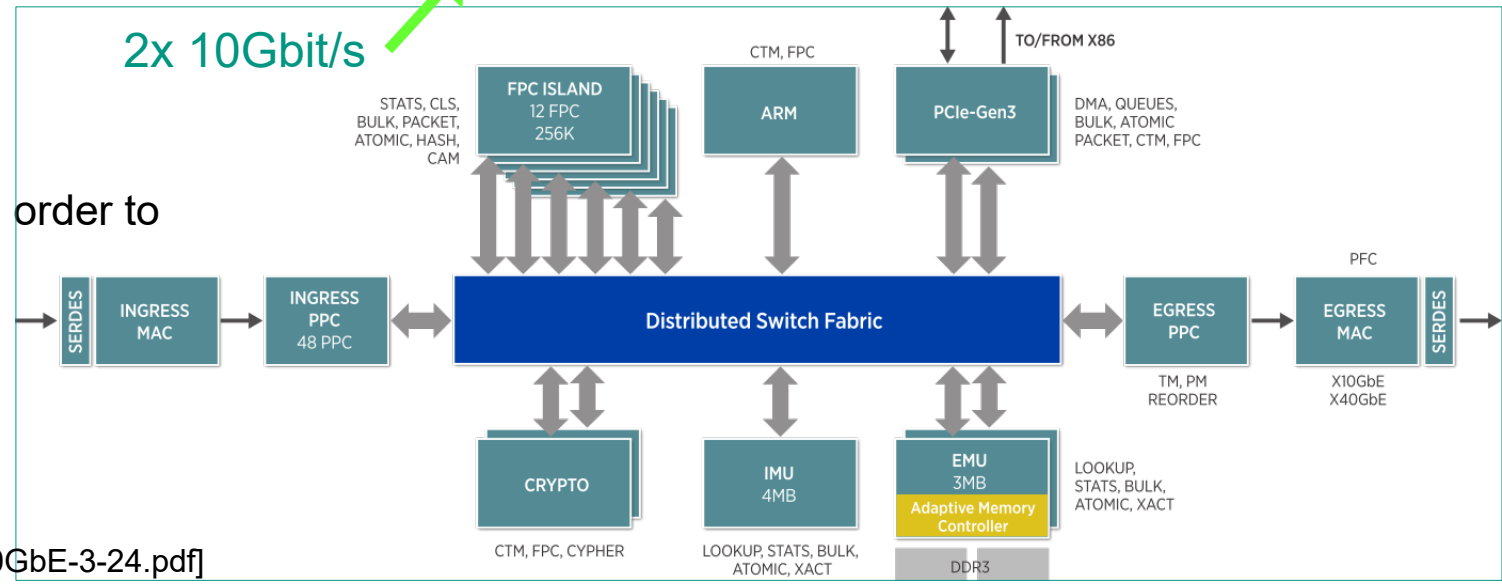


[https://resources.nvidia.com/en-us-accelerated-networking-resource-library/connectx-7-datasheet]

Smart NICs: Example

- Netronome Agilio-CX
  - Used in servers
  - 2x 10 Gbit/s Interfaces
  - Programmable using eBPF Offload
    - eBPF is a kernel script language
    - Normally executed on server CPU
    - Here: runs directly on SmartNIC
    - Code is executed for each packet
      - At line rate!
      - Number of instructions is limited in order to achieve line rate
  - 2GB of On-Board memory
    - Available to eBPF programs





[https://netronome.com/wp-content/uploads/PB_Agilio_CX_2x10GbE-3-24.pdf]

PROBLEMS

1) What are important responsibilities of the network layer?
2) Which basic operations are usually performed by an IP router in order to forward a packet to its destination?
3) Why are high link-speeds such a big problem for modern forwarding hardware?
4) What are possible structures of a switch fabric?
5) How does longest prefix matching work in general?
6) What are efficient (software) data structures for handling longest prefix matching and how do they work?
7) In what way can hash tables support a trie-based address lookup?
8) What are bloom filters and how can they be used for longest prefix matching with hash tables?
9) What is a TCAM? What needs to be considered when updating TCAM rules?
10) What are the main benefits and problems of the TCAM technology?
11) How does the introduced generic router architecture look like?
12) Where can buffer elements be placed inside a switch? What are the associated benefits and drawbacks?
13) Why is queueing necessary? Explain different queueing strategies and their interplay with scheduling.
14) What is traffic shaping and what is its purpose?
15) Explain occurring trade-offs when choosing the size of a router's buffer.

LITERATURE

Literature

[ApKM04] G. Appenzeller; I. Keslassy, Nick Mc Keown; Sizing Router Buffers; ACM SIGCOMM 2004

[Caida16] Center for Applied Internet Data Analysis; Packet size distribution function for equinix-chicago.dirA.20160406-130000.UTC ; 2016; https://www.caida.org/data/passive/trace_stats/chicago-A/2016/equinix-chicago.dirA.20160406-130000.UTC.df.xml

[EA18] ethernet alliance; The 2018 Ethernet Roadmap; https://ethernetalliance.org/the-2018-ethernet-roadmap/

[ITU21] International Telecommunication Union (ITU); ITC Facts and Figures 2021; 2021; http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2021.pdf

[Hous20] G. Houston; What´s the Right Network Buffer Size?; The Internet Protocol Journal; May 2020

[KuRo22] J.F. Kurose, K.W. Ross; Computer Networking – A Top-Down Approach; Addison Wesley, 8th Edition, 2022

[McAK19] N. McKeown, G. Appenzeller, I. Keslassy; Sizing Router Buffers (Redux); ACM SIGCOMM Computer Communication Review, Vol. 49, Issue 5, Oct. 2019

[Medh18] Deepankar Medhi, Karthikeyan Ramasamy; Network Routing: Algorithms, Protocols, and Architectures; Morgan Kaufmann, 2nd edition, 2018

[Sara03] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor.; Longest prefix matching using bloom filters; ACM SIGCOMM 2003

[PaSh06] K. Pagiamtzis and A. Sheikholeslami; Content-addressable memory (CAM) circuits and architectures: a tutorial and survey; IEEE Journal of Solid-State Circuits, vol. 41, no. 3, pp. 712-727, March 2006

[ShGu01] D. Shah and P. Gupta, "Fast updating algorithms for TCAM," in IEEE Micro, vol. 21, no. 1, pp. 36-47, Jan. -Feb. 2001, doi: 10.1109/40.903060.

[ZLZW20] B. Zhao, R. Li, J. Zhao and T. Wolf, "Efficient and Consistent TCAM Updates," IEEE INFOCOM 2020 – IEEE Conference on Computer Communications, Toronto, ON, Canada, 2020, pp. 1241-1250, doi: 10.1109/INFOCOM41043.2020.9155281.