The Internet Ecosystem and Evolution

Contents

- IP router architectures
  - general router architectures, linecard/ backplane/control, IP packet forwarding, router generations
- FIB lookup
  - longest prefix matching
  - hardware and software realizations for LPM: TCAMs, prefix trees
  - FIB aggregation

IP router architectures

IP packet forwarding

- **IP header check:** format, version, header length, options, header checksum
- **FIB lookup:** find the most specific FIB entry for the destination IP address in a packet
- **TTL handling:** if TTL=0 then drop packet and send an ICMP message, otherwise update TTL: TTL←TTL – 1
- **Recompute header checksum**
- Optionally: fragmentation, source routing, etc.

High-performance routers

Cisco GSR 12416

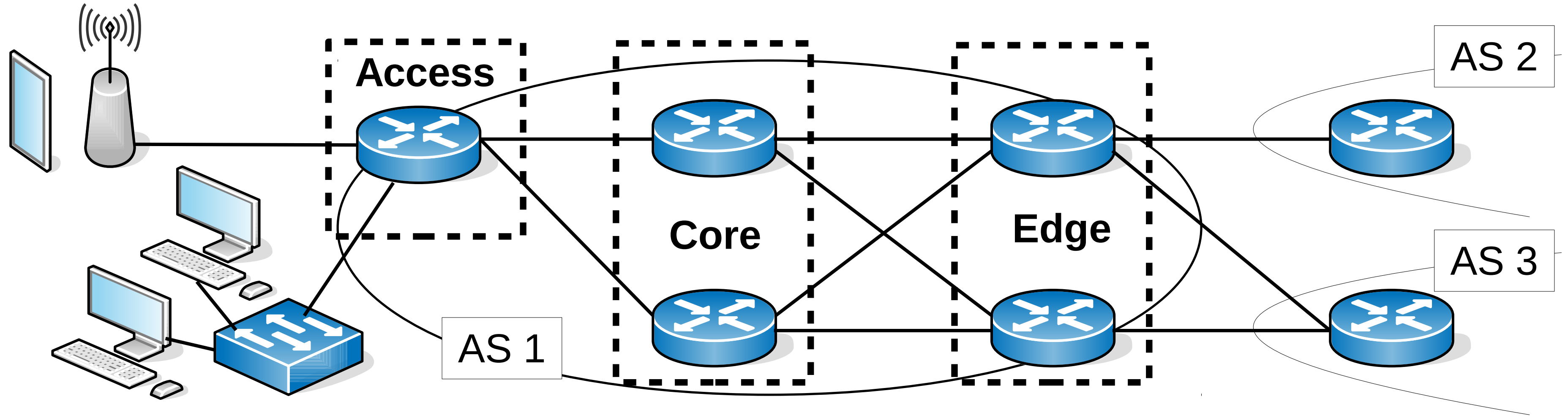
Capacity: 160 Gb/s
Power consumption: 4.2kW

Capacity: 80 Gb/s
Power consumption: 2.6kW

Juniper M160

Router categories (RFC4098)

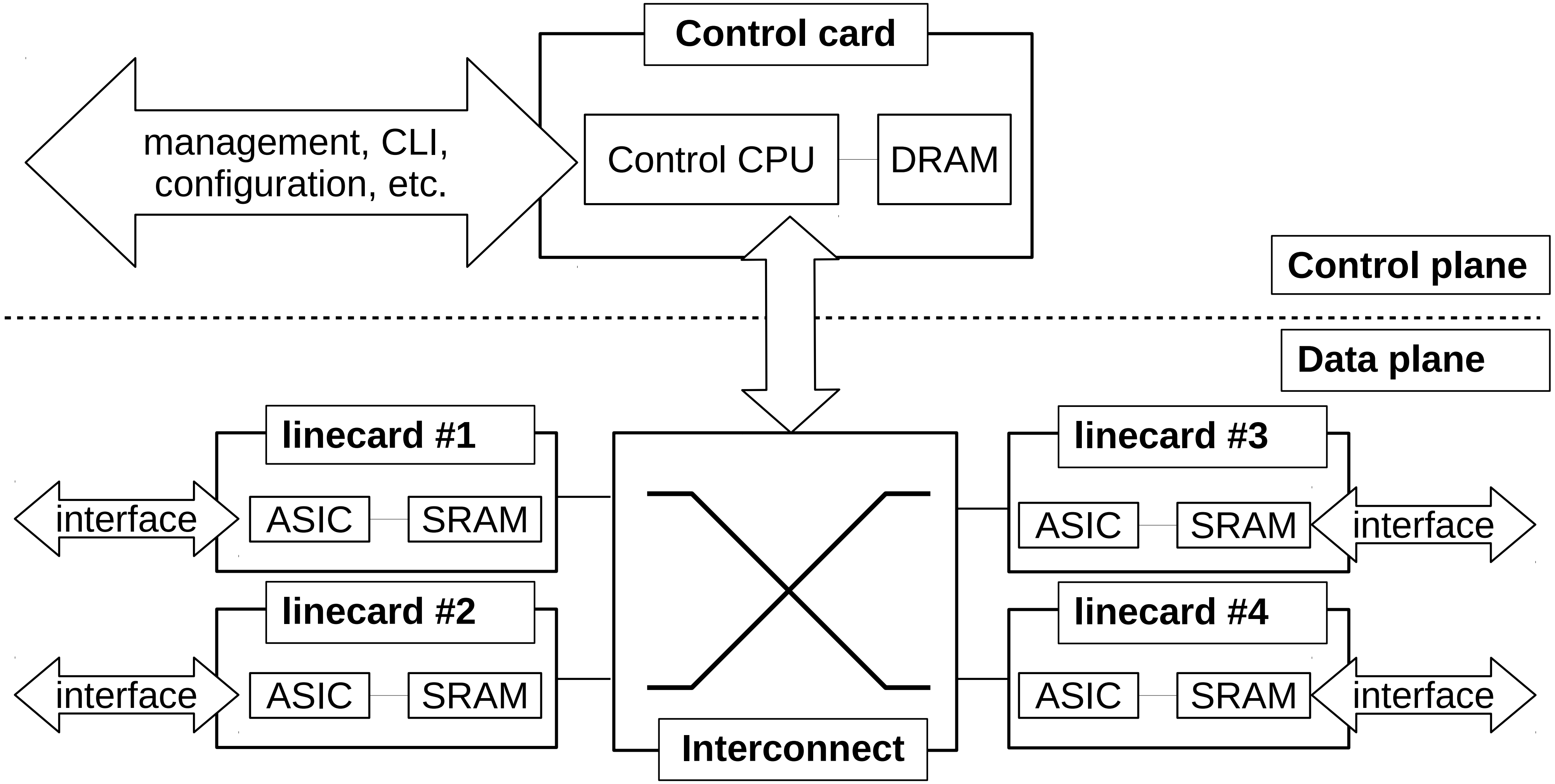
- **Edge/border router:** inter-AS traffic forwarding (iBGP+eBGP+IGP)
- **Core router:** handling intra-AS traffic between different POPs of an ISP (IGP+iBGP)
- **Access router:** concentrating traffic from the Internet edge to the core



Router types

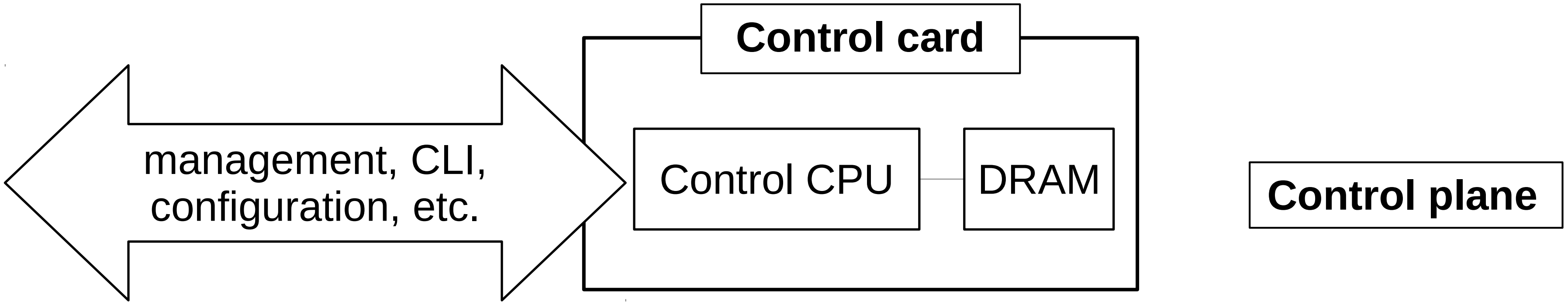
- **Soft(ware) router:** IP router implemented in software and running on general purpose CPUs
  - PC+Linux+Quagga (e.g., our OpenWRT image)
  - smaller performance
  - for smaller ISPs, IXPs, BGP monitors
  - cloud hypervisors(!)
- **Hard(ware) router:** high-performance router using special purpose hardware ASICs
  - edge/core routers of large ISPs
  - access routers concentrating lots of subscribers

General IP router architecture



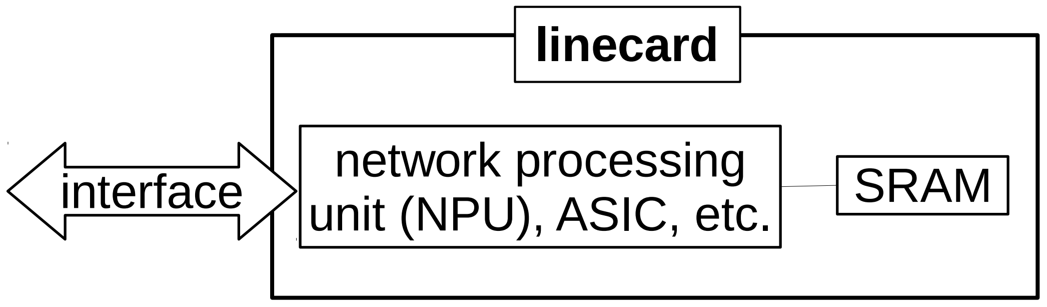
General IP router architecture

- **Control card:** router logics
  - running the routing protocols, management access (CLI: Command Line Interface, SNMP, etc.), monitoring, extra services
  - manage the interconnect, set the FIB
  - general purpose CPU/DRAM, even general purpose OS (like, Linux!)



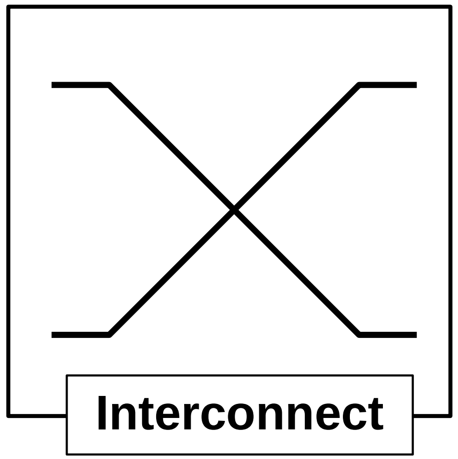
General IP router architecture

- **Interface card (linecard):** packet input/output
  - one or more physical port (interface) for links (Serial/FastEthernet/GigabitEthernet)
  - basic header parsing/processing functions
  - special purpose HW and fast static memory
  - most routers extendible with new cards



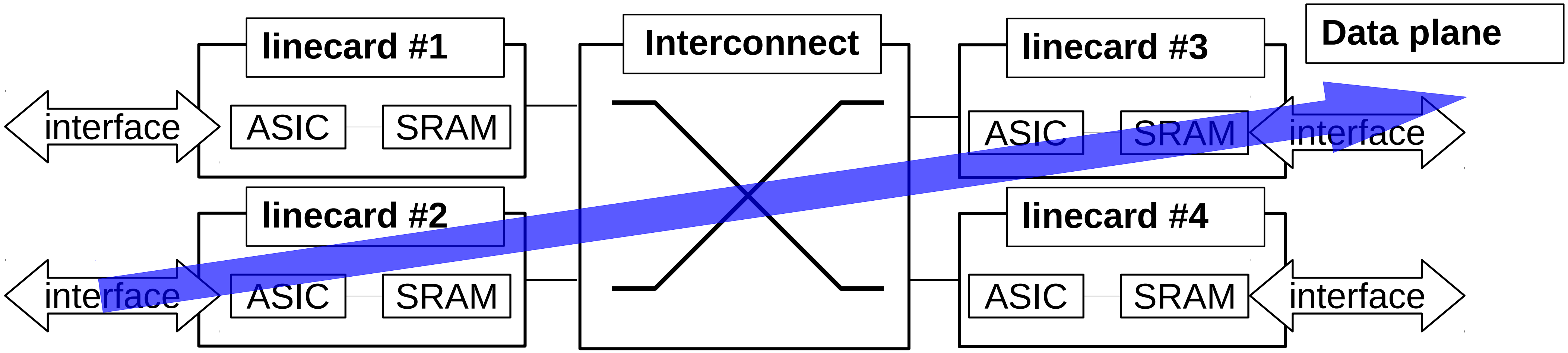
General IP router architecture

- **Interconnect/backplane:** switching matrix
  - communication between linecards and the control CPU + buffering
  - shared bus/internal switch (even Ethernet)
  - input buffer (head of line blocking!)/output buffer/shared memory



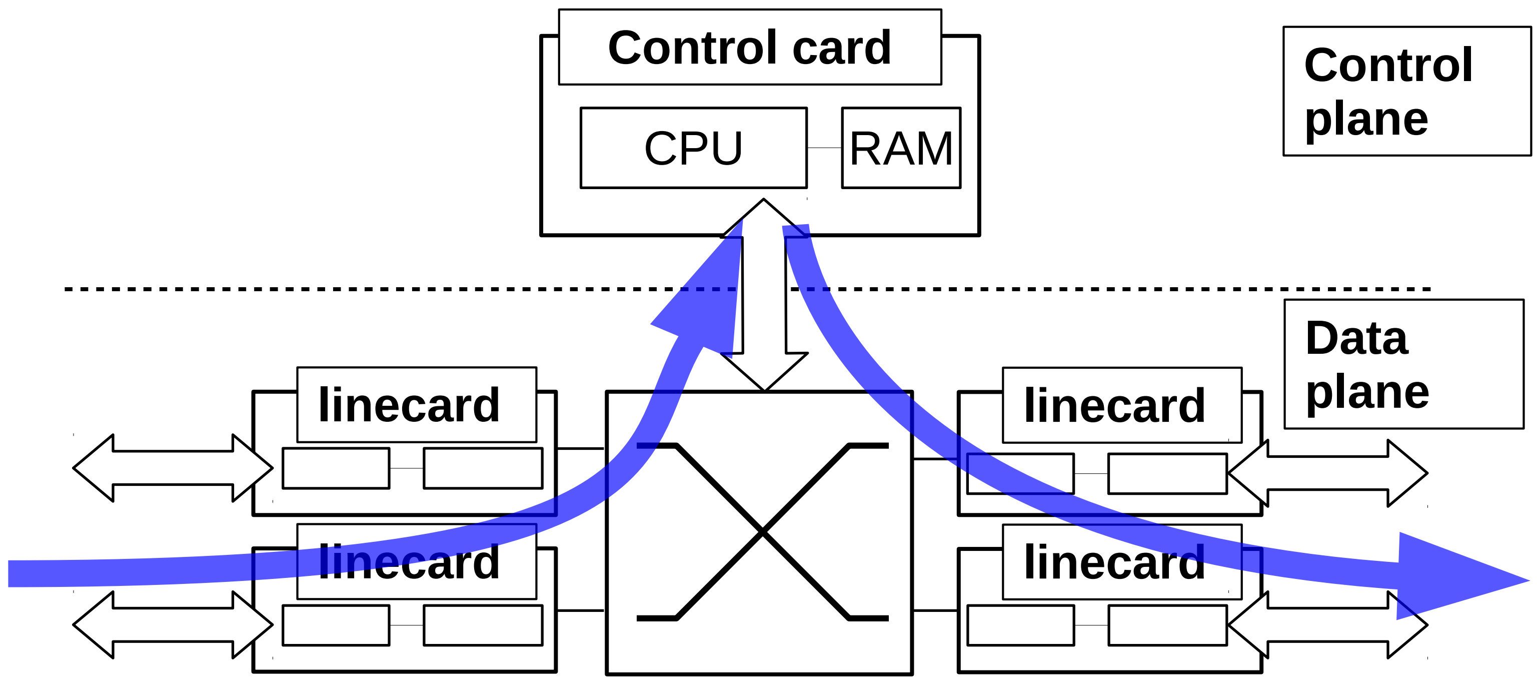
Fast path

- **Fast path:** steps of packet forwarding that are implemented inside the **Data plane** (high speed)
  - operations that are easy to realize in HW
  - header parsing/header checksum computation
  - often FIB lookup too, but this requires the FIB to be downloaded to the linecards!



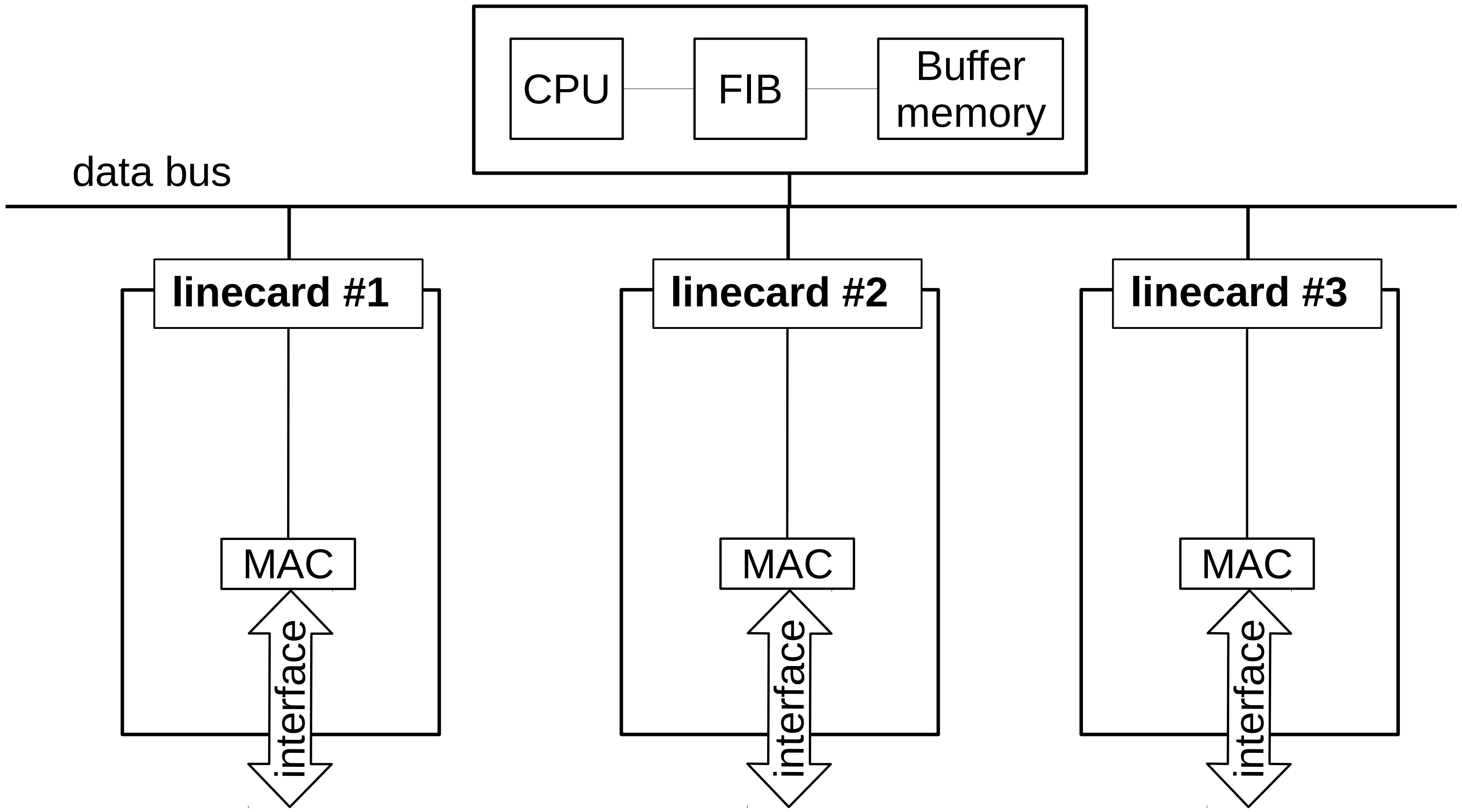
Slow path

- **Slow path:** complex operations that require the intervention of the control CPU (slower!)
- IP options, fragmentation, protocol message handling, ARP, ICMP packet generation, etc.



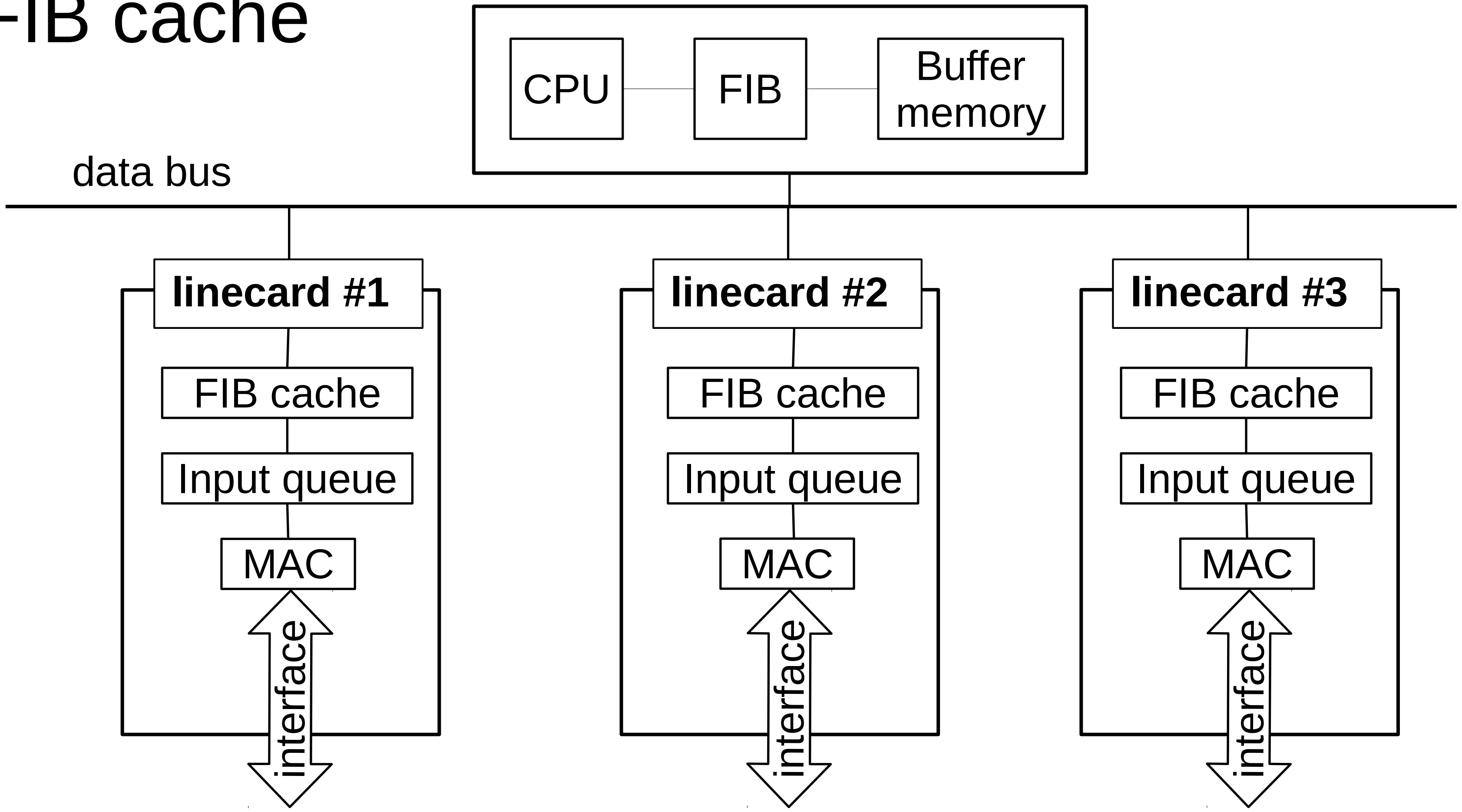
1st generation routers

- Every packet goes through the slow path
- Only basic interface functions on the linecards



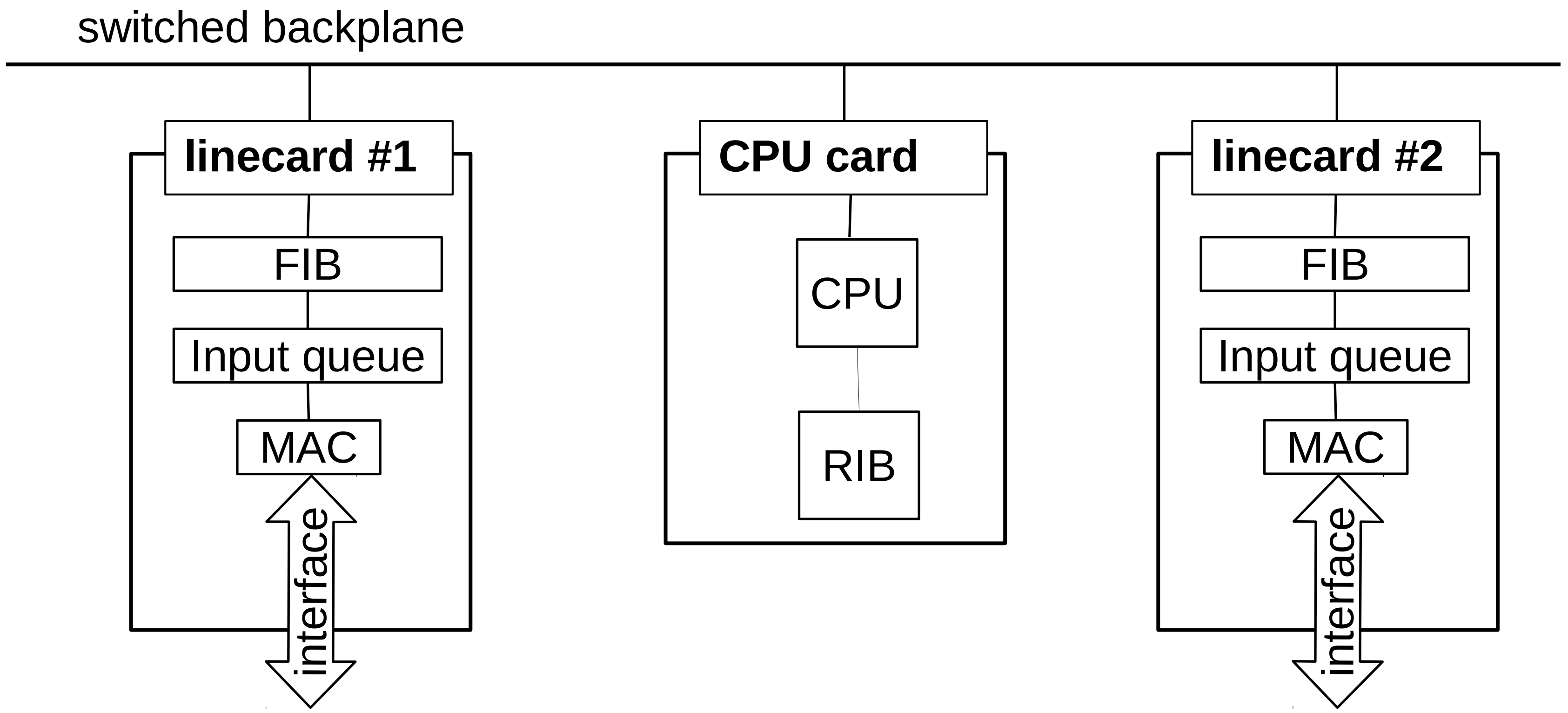
2ng generation routers

- Linecards implement input buffering+FIB cache
- Fast path forwarding if destination address is in FIB cache



3rd generation routers

- Whole FIB downloaded to the linecard, normal packet forwarding fully inside the fast path
- The CPU is just another card in the chassis



Router generations: the future?

- Today's CPUs are as fast as, and cheaper than, yesterday's special purpose hardware ASICs
  - general purpose CPUs improve by Moore's law
  - forwarding is a massively parallel process
  - we may dedicate a separate CPU per packet
- Router virtualization rules out special purpose HW
- Today's routers are proprietary “black boxes”
- Future routers will adopt an open, programmable SW/HW design

FIB lookup

FIB lookup

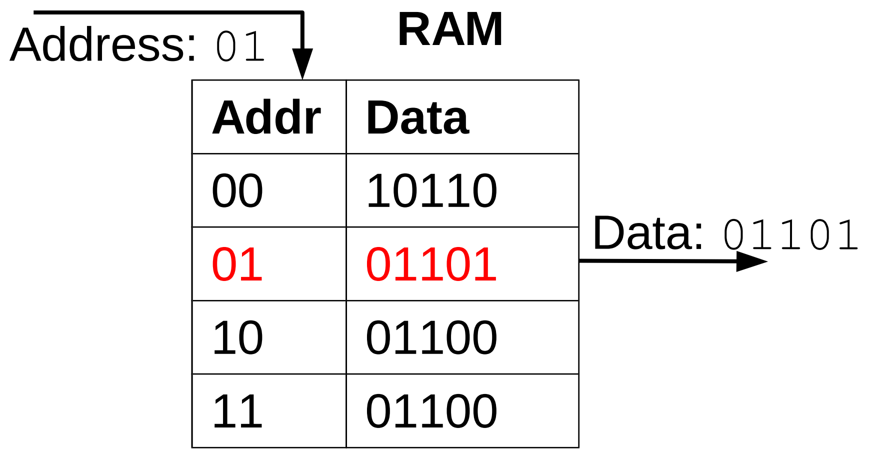
- The most expensive (by a large margin) operation of IP packet forwarding: **find the most specific FIB entry for the destination IP address in the packet header** (LPM)
- **FIB (Forwarding Information Base):** the collection of all forwarding rules for a router
- Built by the routing protocols, downloaded by the control CPU to the linecards (3rd generation)
- This gives fast-path packet forwarding

FIB lookup: LPM

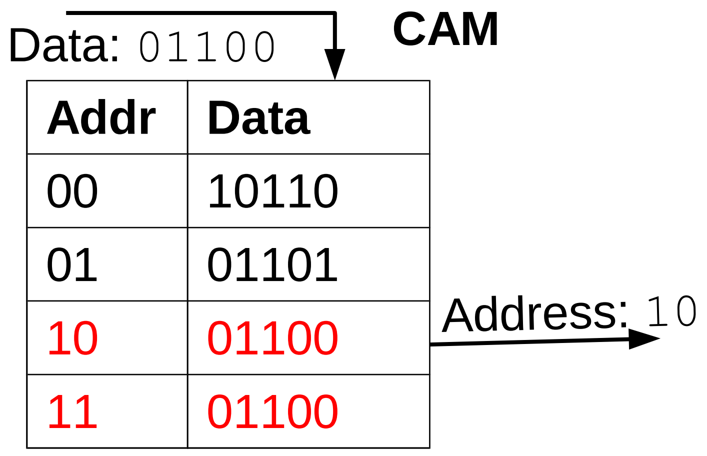
- Implementing LPM is non-trivial (at best)
- A naïve approach would be to search through all FIB entries linearly to find the one matching on the most bits (counted from the MSB)
- Complexity *O(N)*, if number of FIB entries is *N*
- In practice, $N \approx 10^5 – 10^6$, the time budget is roughly 400 CPU cycles (at line rate 10Gbit/s, 500 byte packet size, 1GHz CPU clock rate)
- The naïve approach is hardly usable

Content addressable memories

- A **Content Addressable Memory (CAM)** is the opposite of a typical memory (RAM):
  - RAM: find data based on memory address
  - CAM: find memory address for data
  - if more entries match, find the first match



Addr	Data
00	10110
01	01101
10	01100
11	01100



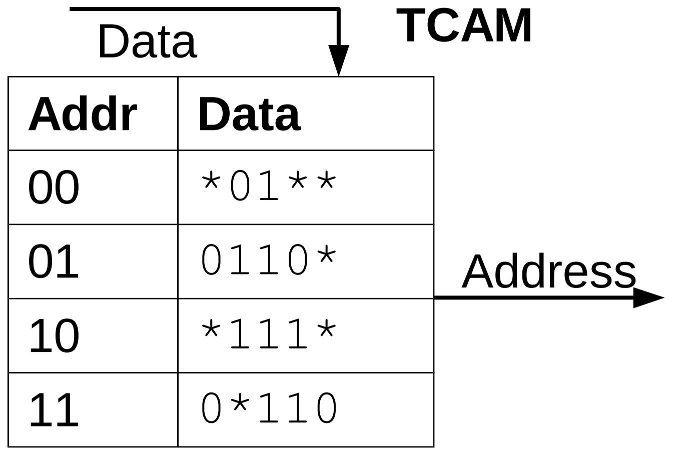
Addr	Data
00	10110
01	01101
10	01100
11	01100

Ternary CAM: TCAM

- A CAM can match fully specified data (contain bits valued `0` or `1`) only, while a **Ternary Content Addressable Memory (TCAM)** can take as input patters that contain "Don't care" bits (`*`) too
- For instance, the input TCAM pattern `101**` matches each of the possible TCAM entries `10100`, `10101`, `10110` and `10111`
- `*` can appear anywhere in pattern (not just the end)
- Entries at lower addresses matched first and hence override entries at higher addresses: **priority**
- Output the address of the first matching entry found

Ternary CAM: TCAM

- For input `00110` the entries at address `00` and `11` both match, since the first address is smaller the TCAM search result is address `00`
- For pattern `11111` result is `10`
- For pattern `01110`, enties at address `10` and `11` both match, output is: `10`



Addr	**Data**
00	`*01**`
01	`0110*`
10	`*111*`
11	`0*110`

TCAM: Implementation

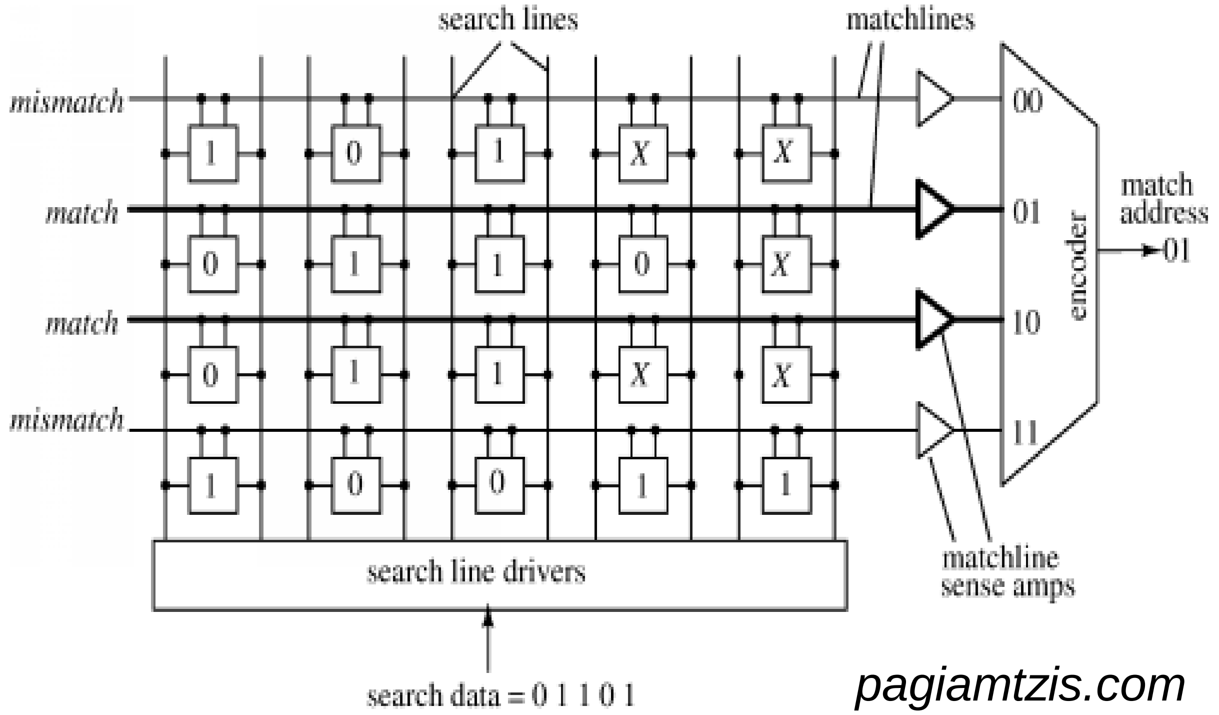
- *(#entries * bit_width)* TCAM cells, each can compare against a stored pattern bit `0/1/*`
- Cells do the comparison in parallel
- Output logics picks the smallest active address

TCAM

Data: `01101`

Addr	**Data**
00	`101**`
01	`0110*`
10	`011**`
11	`10011`

Addr: `01`



pagiamtzis.com

Implementing FIBs in a TCAM

- Consider the below FIB

IP prefix	Binary prefix	Next-hop
`160.0.0.0/3`	`101`	`10.0.0.1`
`96.0.0.0/4`	`0110`	`10.0.0.2`
`96.0.0.0/3`	`011`	`10.0.0.3`
`184.0.0.0/5`	`10111`	`10.0.0.2`

- LPM: find the FIB entry matching the destination IP address on the most bits
- For instance, address `96.128.59.12` matches entries 2 and 3, the former on more bits
- LPM result: next-hop for entry 2 (`10.0.0.2`)

Implementing FIBs in a TCAM

- TCAMs are a natural way to implement FIBs
- **Fully specified subnet prefix** in the entries
- The TCAM matches the prefix bit-by-bit
- Let the **host identifier bits as “dont care”** (*)
- These bits do not count in LPM lookup
- “Don't care” bits (*) appear at the end of entries

Addr	**IP prefix**	**TCAM pattern**	**Next-hop**
00	160.0.0.0/3	101***** ******** ******** ********	10.0.0.1
01	96.0.0.0/4	0110**** ******** ******** ********	10.0.0.2
10	96.0.0.0/3	011***** ******** ******** ********	10.0.0.3
11	184.0.0.0/5	10111*** ******** ******** ********	10.0.0.2

Implementing FIBs in a TCAM

- **Problem:** if we write entries into the TCAM in this order, a less specific entry may override a more specific one
- E.g., for address `184.1.1.1` the result would be the first entry in the TCAM, even though the real LPM result would be entry 4 (matches on more bits)
- **Solution:** order FIB entries into the **decreasing order of prefix length**
  - longer (more specific) prefixes go to lower addresses
  - shorter prefixes (less specifics) go high addresses
  - entries at lower addresses preferred by TCAM = match on a long prefix overrides match on a short prefix=LPM

Implementing FIBs in a TCAM

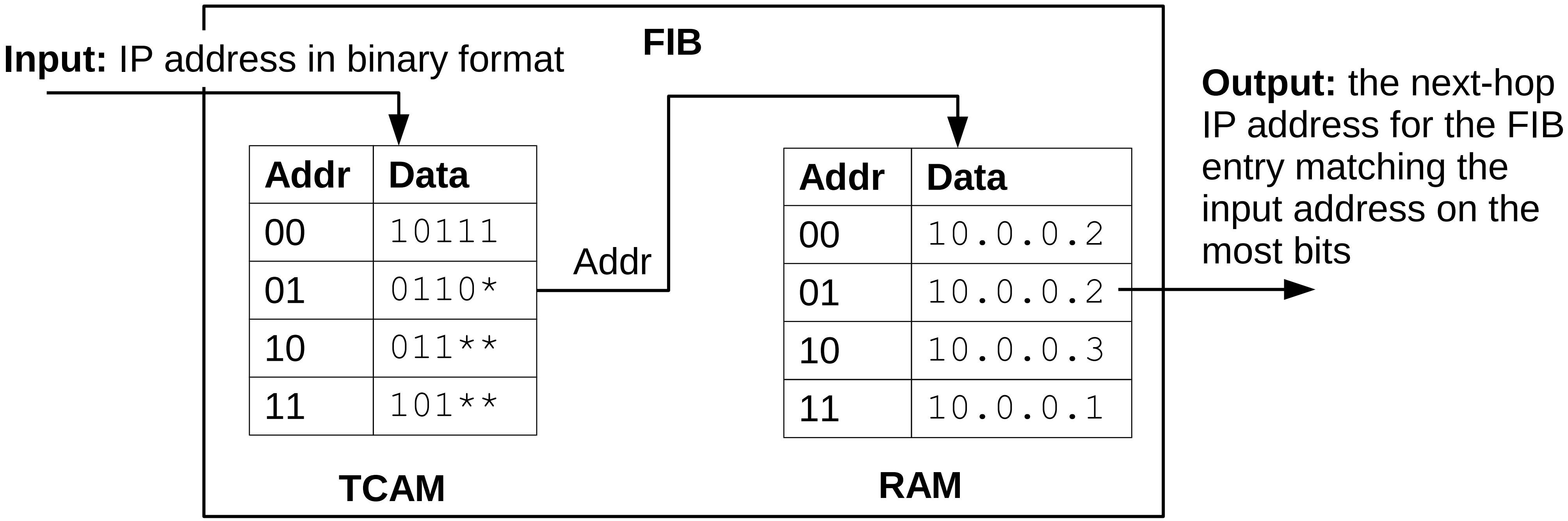
- FIB entries can be freely reordered, as the priority is set firmly by the prefix length
- The FIB after ordering the entries into the decreasing order of prefix length

Addr	IP prefix	TCAM pattern	Next-hop
00	184.0.0.0/5	10111*** ******** ******** ********	10.0.0.2
01	96.0.0.0/4	0110**** ******** ******** ********	10.0.0.2
10	96.0.0.0/3	011***** ******** ******** ********	10.0.0.3
11	160.0.0.0/3	101***** ******** ******** ********	10.0.0.1

- Red columns go into the TCAM verbatim
- Rest are stored in a separate RAM module

Implementing FIBs in a TCAM

- **HW FIB:** a TCAM connected to a RAM
- Result of the TCAM search is used as address into the RAM to read the next-hop address
- 5-bit wide TCAM is enough (max. prefix length)



Input: IP address in binary format
FIB
Addr
TCAM
RAM
Output: the next-hop IP address for the FIB entry matching the input address on the most bits

TCAM:

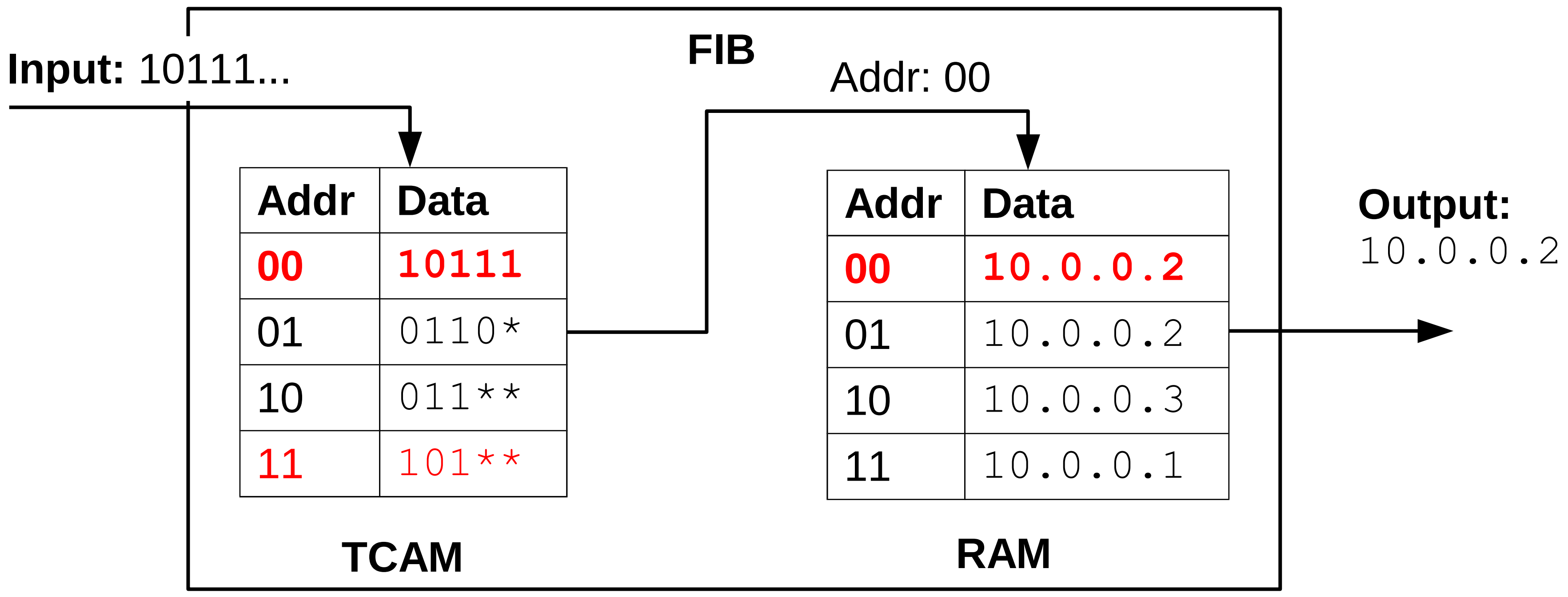
Addr	Data
00	`10111`
01	`0110*`
10	`011**`
11	`101**`

RAM:

Addr	Data
00	`10.0.0.2`
01	`10.0.0.2`
10	`10.0.0.3`
11	`10.0.0.1`

Implementing FIBs in a TCAM

- For the IP address `184.1.1.1=10111...` the TCAM gives result `00`
- Next-hop is taken from the RAM at address `00`



TCAM

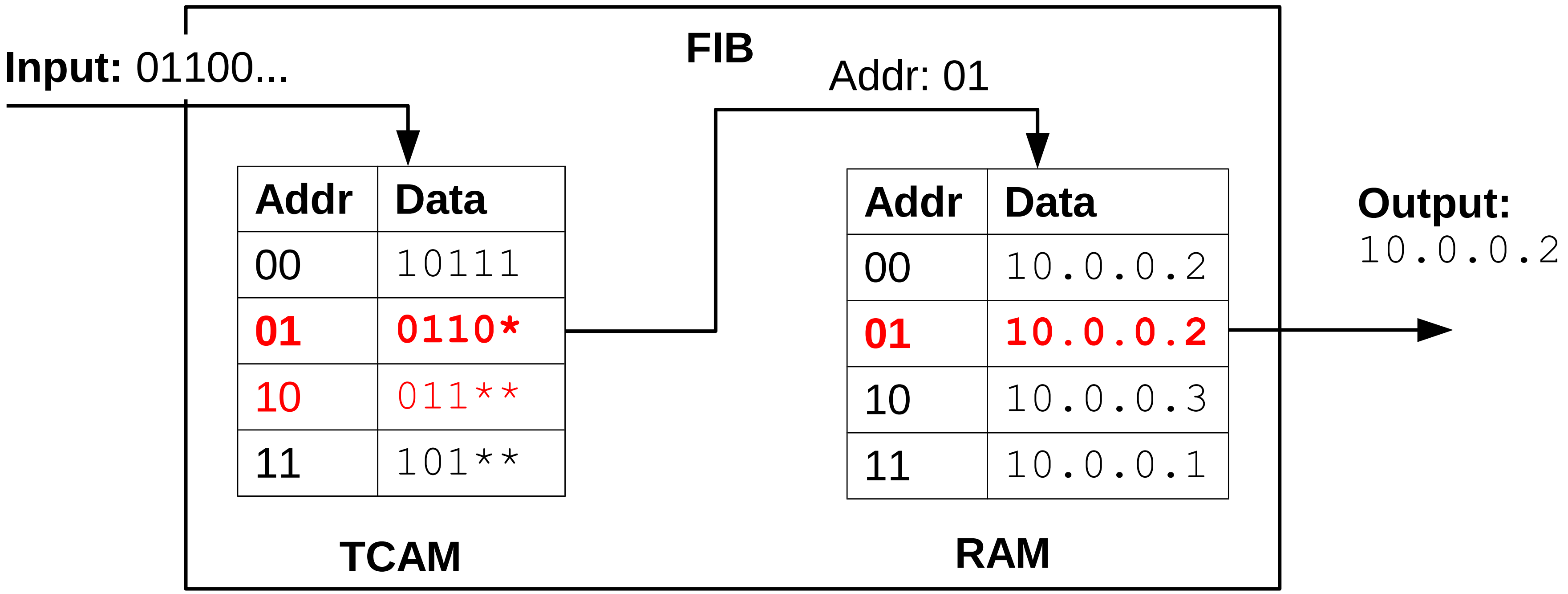
Addr	Data
00	**10111**
01	0110*
10	011**
11	101**

RAM

Addr	Data
00	**10.0.0.2**
01	10.0.0.2
10	10.0.0.3
11	10.0.0.1

Implementing FIBs in a TCAM

- For IP address `97.12.124.45=01100...` the TCAM patterns at address 2 and 3 both match
- Result is address `01`, next-hop is `10.0.0.2`



Addr	Data
00	10111
01	0110*
10	011**
11	101**

Addr	Data
00	10.0.0.2
01	10.0.0.2
10	10.0.0.3
11	10.0.0.1

Implementing FIBs in a TCAM

- Router ASICs (Application Specific IC) usually contain both the TCAM and the RAM
- FIB lookup in a couple of clock cycles: very efficient fast-path IP packet forwarding
- TCAMs commonly used for other purposes: Ethernet MAC learning, firewall/ACL rules, etc.
- But TCAMs are complex: 16 transistor/cell (SRAM: 6, DRAM: 2 transistor/cell): expensive!
- High power consumption (9MB TCAM chip, 100 MHz clock, 10–15W dissipation): cooling!

Implementing LPM in software

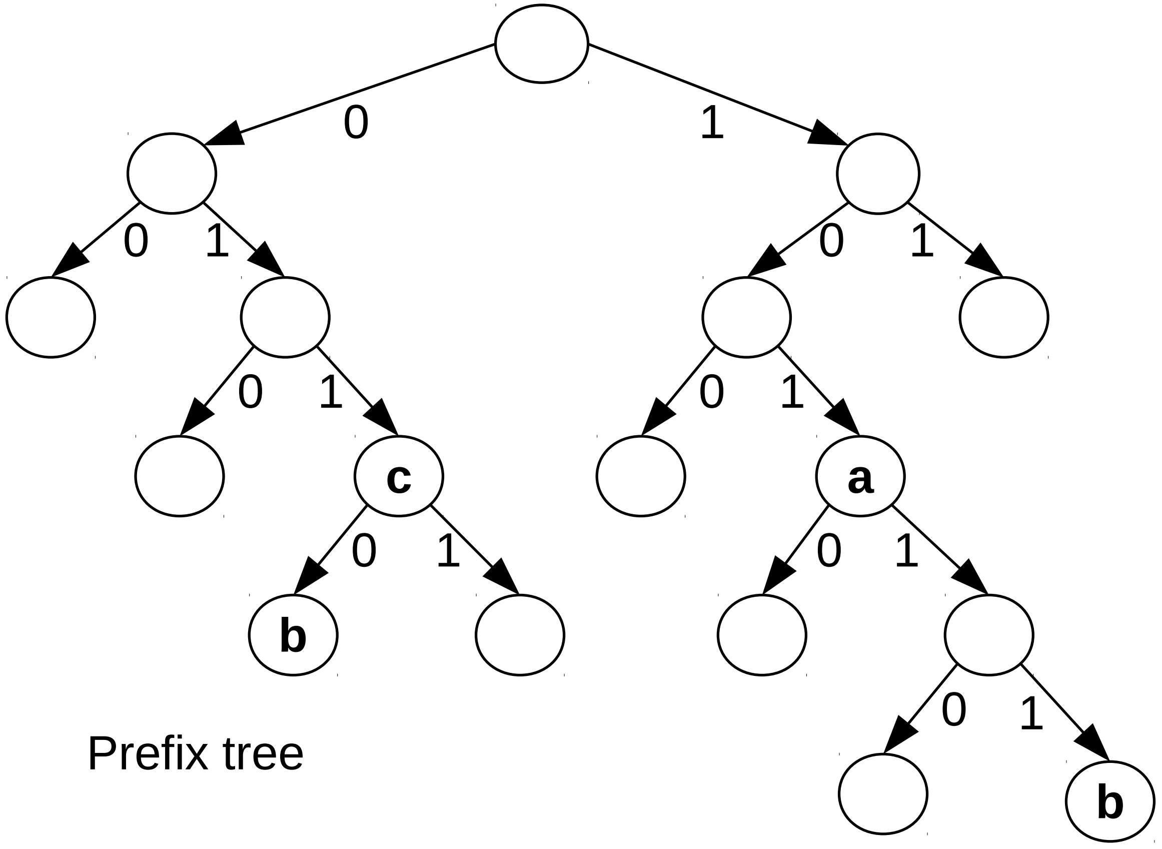
- Often, TCAMs are an overkill: soft routers, simple access routers (e.g., SOHO router), virtual switches/routers (cloud)
- A FIB data structure is needed that supports fast LPM on a general purpose CPU
- In software the most expensive operation is memory accesses (DRAM: ~200 CPU cycles)
- **Goal:** minimize the number of memory reads needed for a longest prefix matching lookup

The binary prefix tree

- Data structure optimized for LPM: a content- (or prefix-)addressable memory
- Storage and search of (prefix→label) pairs
- The **prefix tree** supports these operations:
  - **lookup:** find the longest prefix matching the input and read the corresponding label
  - **insert:** insert a (prefix→label) pair
  - **delete:** remove prefix and the corresponding label from the tree
  - **modify:** modify label at prefix

The binary prefix tree

- Consider the previous FIB divided into two parts
- Identify next-hops with **unique labels** and store them into a separate **next-hop index** table



Prefix tree

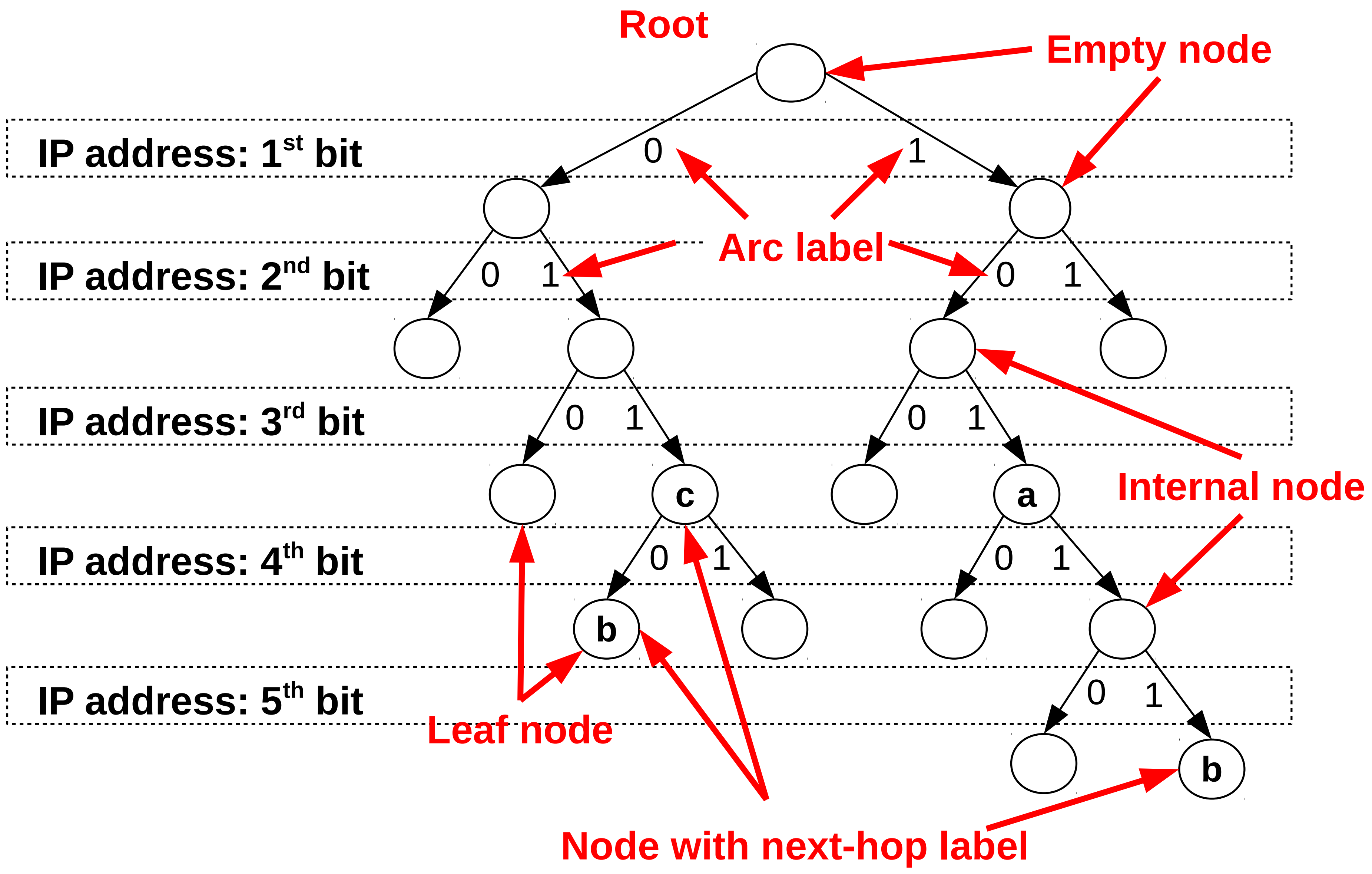
FIB

IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

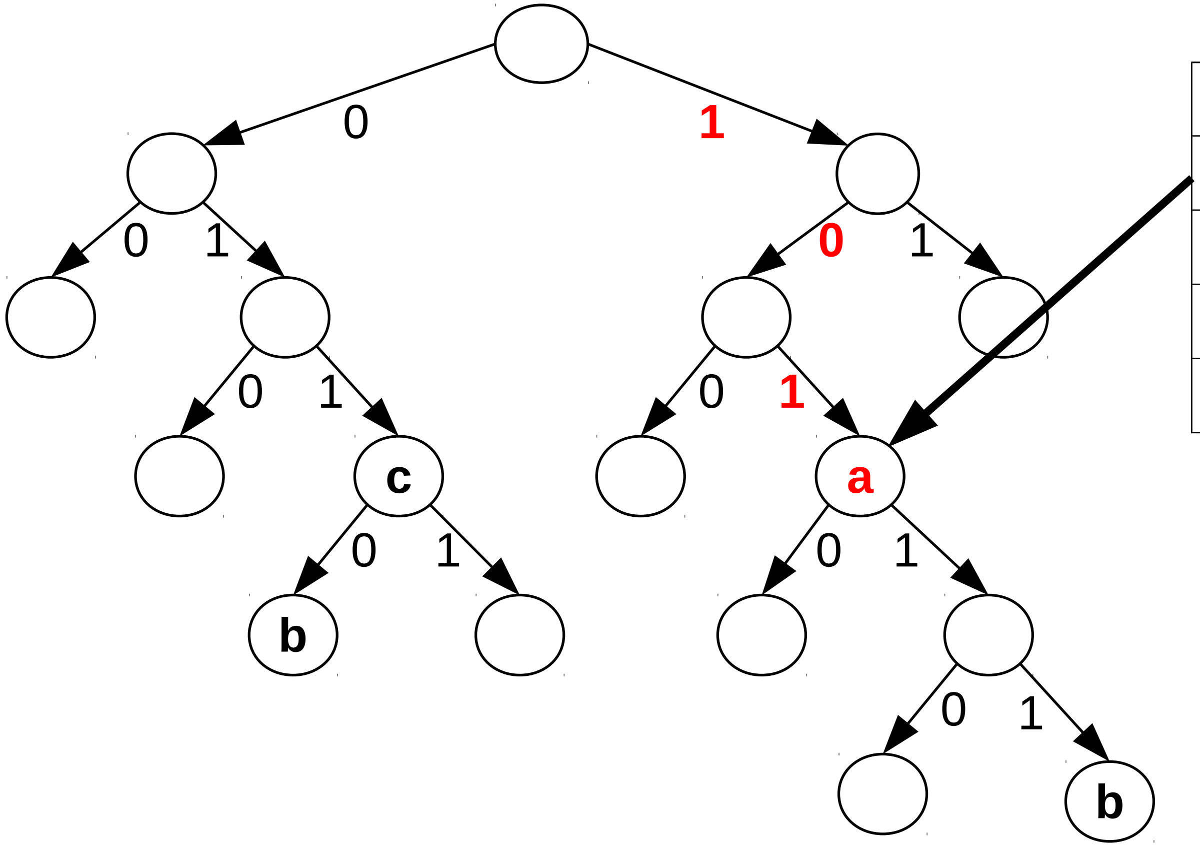
Label	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

The binary prefix tree



The binary prefix tree

- **Prefix=sequence of arc labels along a tree path**
- Mark the tree node that belongs to each prefix in the FIB with the next-hop label for the prefix



FIB

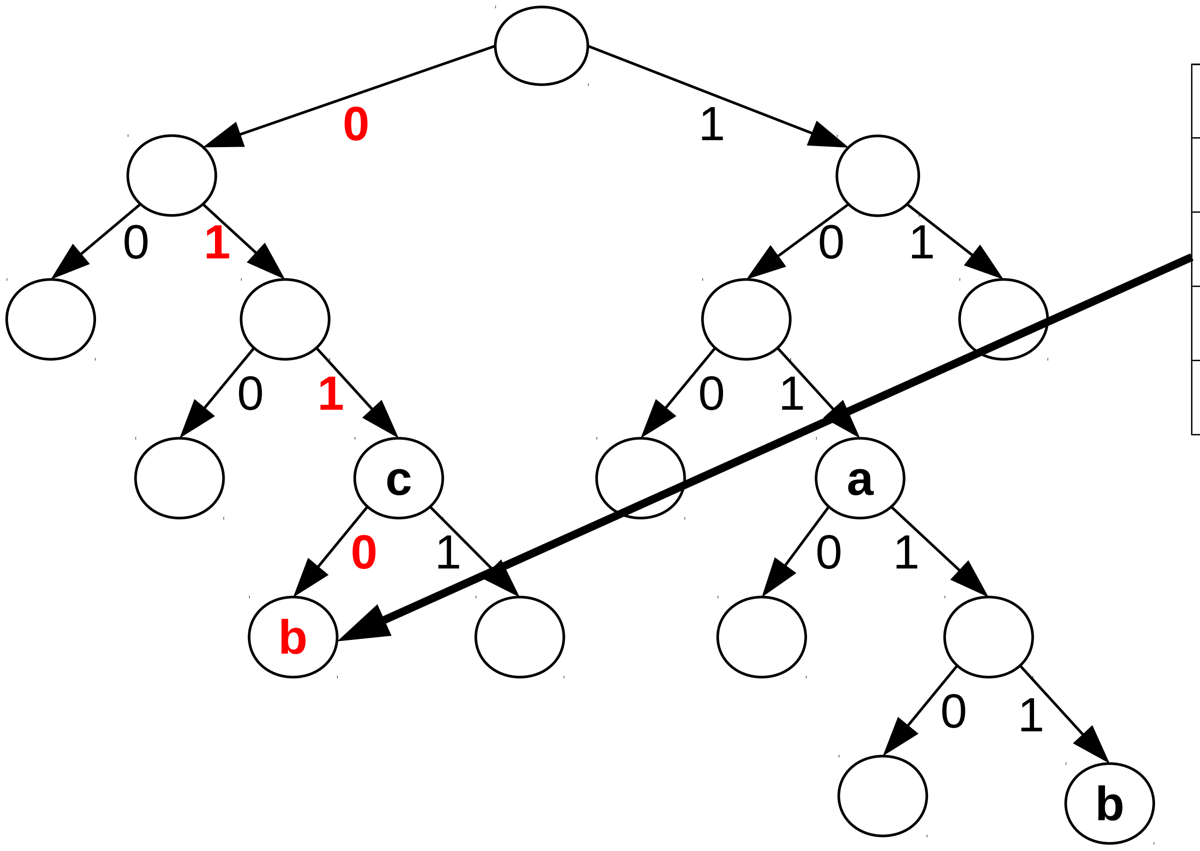
IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Label	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

The binary prefix tree

- **Prefix=sequence of arc labels along a tree path**
- Mark the tree node that belongs to each prefix in the FIB with the next-hop label for the prefix



FIB

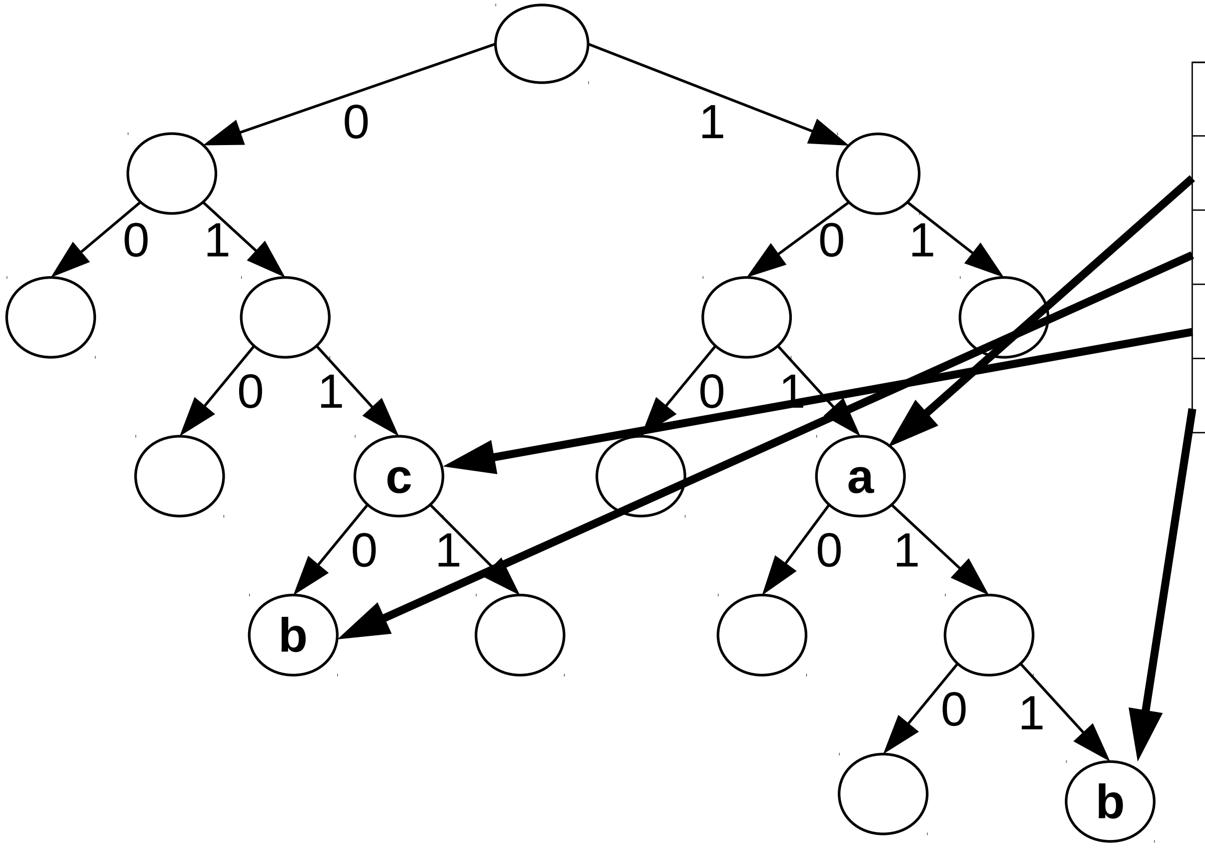
IP prefix	**Prefix**	**Label**
160.0.0.0/3	101	a
96.0.0.0/4	**0110**	**b**
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Label	**Next-hop**
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

The binary prefix tree

- **Prefix=sequence of arc labels along a tree path**
- Mark the tree node that belongs to each prefix in the FIB with the next-hop label for the prefix



FIB

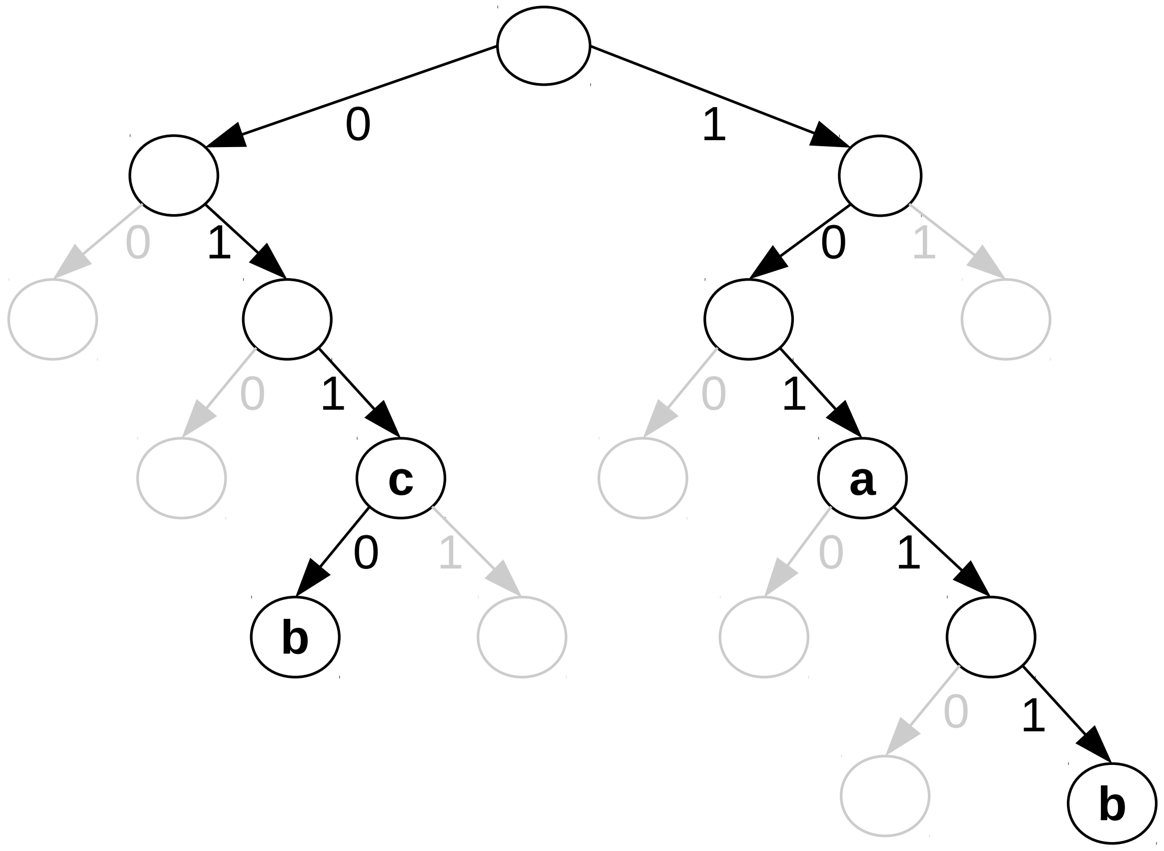
IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Label	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

The binary prefix tree

- Empty leaf nodes can be omitted (arcs to empty nodes will be marked by NULL pointers)
- Smaller tree, less memory



FIB

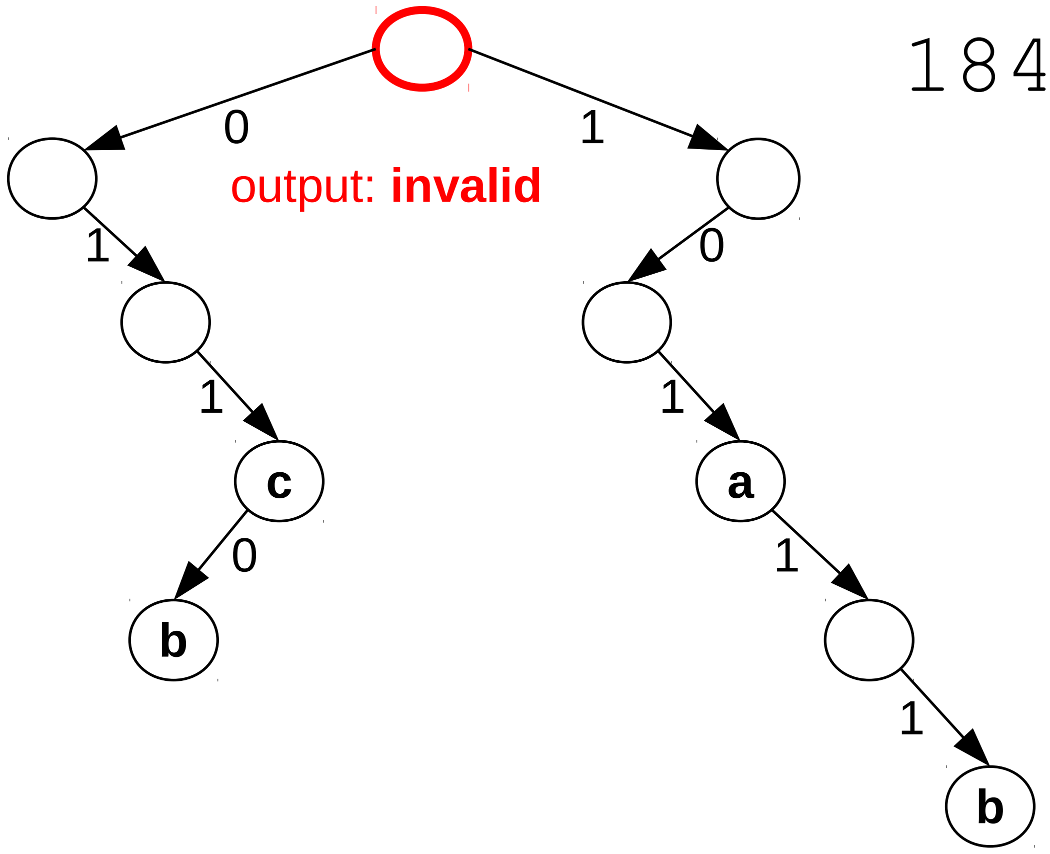
IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Label	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

Prefix tree: Lookup

- Find the most specific entry in the prefix tree for the IP address `184.1.1.1=10111...`
- Start from the root node, **`output←invalid`**



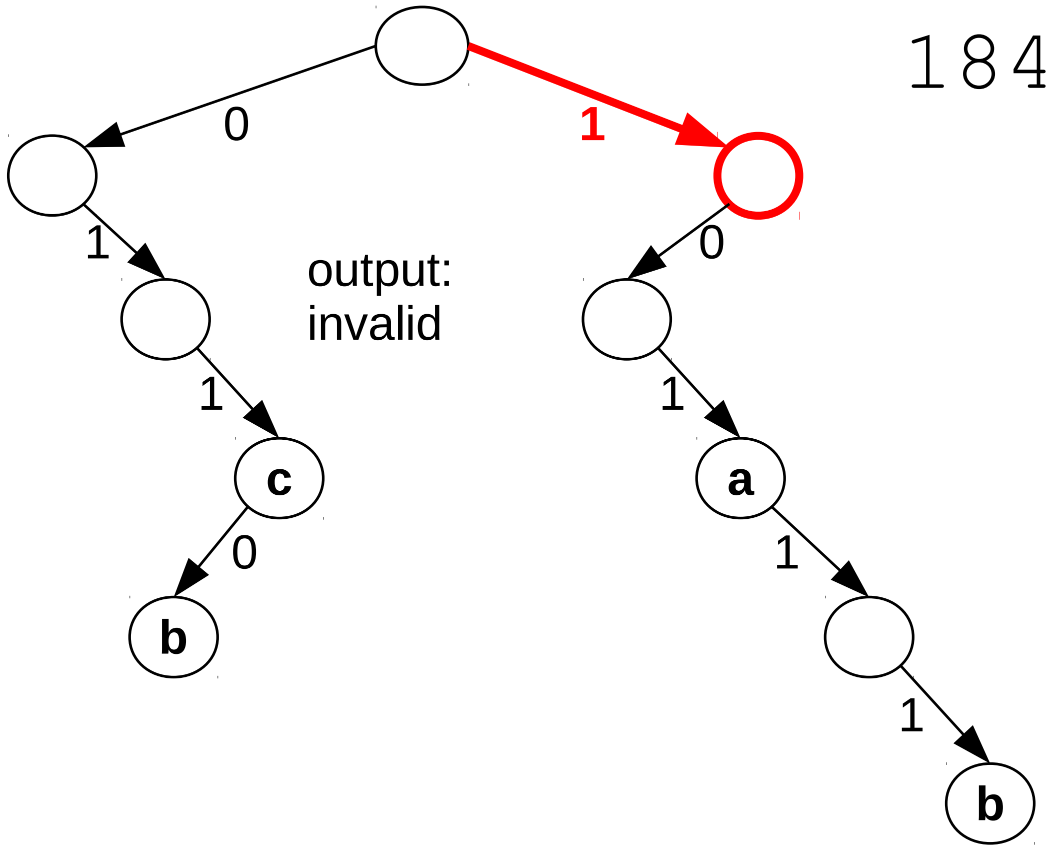
`184.1.1.1=10111...`

FIB

IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Prefix tree: Lookup

- First bit of the address `184.1.1.1=10111...` is set to `1`, so we **proceed from the root along the arc labeled with arc-label `1`** to the next node



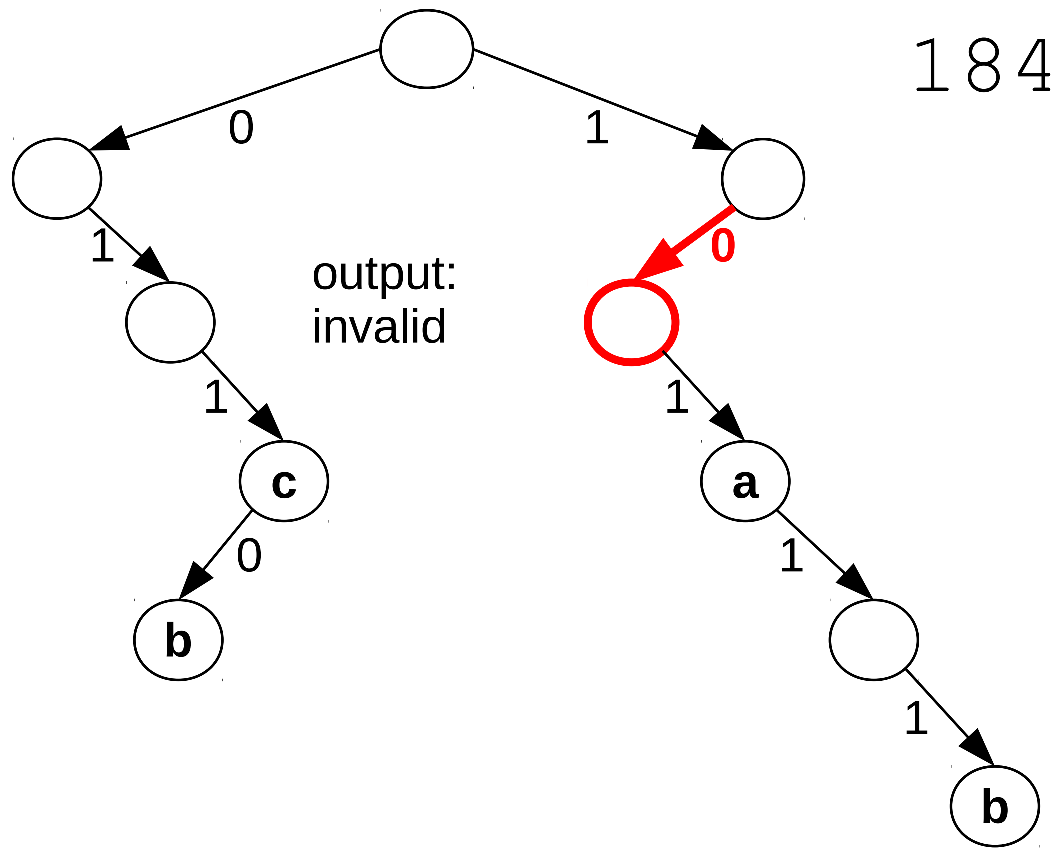
`184.1.1.1=`**`1`**`0111...`

FIB

IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Prefix tree: Lookup

- No label in the current node, `output` is unchanged
- Second bit is `0`, so **move to the next node** along the arc with arc-label `0`



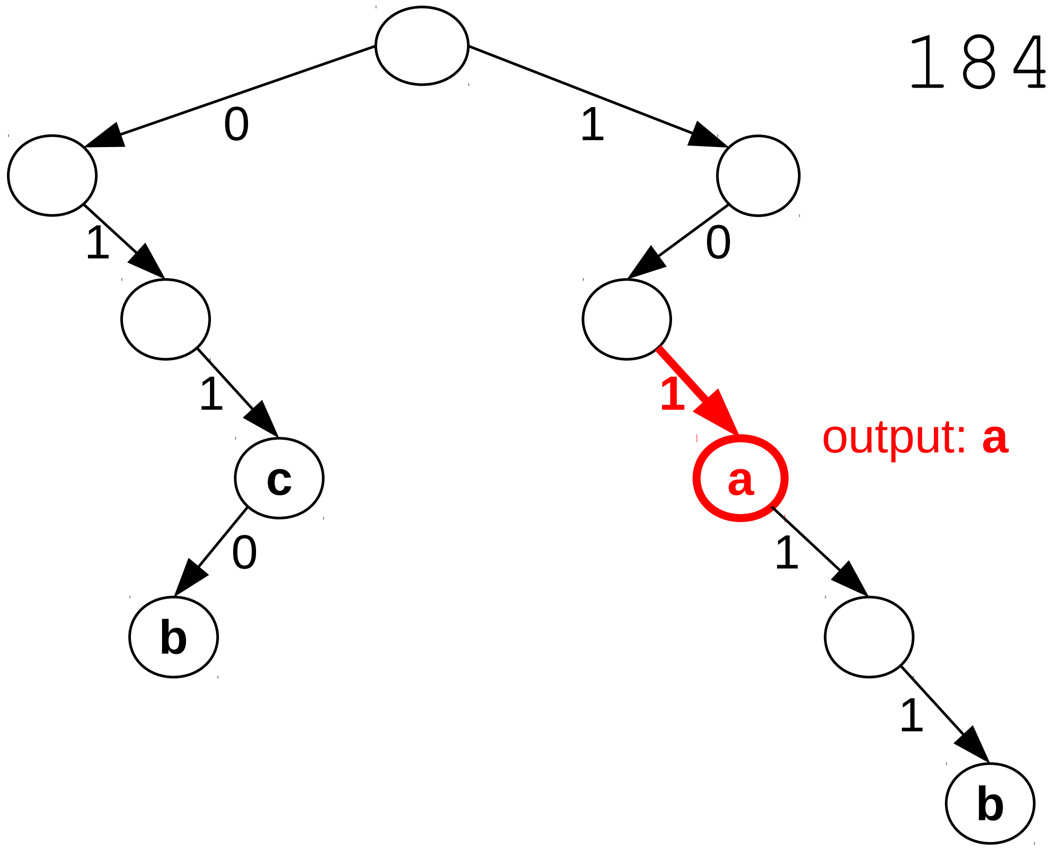
184.1.1.1=1**0**111...

FIB

IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Prefix tree: Lookup

- Third bit is again `1`, so proceed along arc with label `1` to the next-node
- New node has label **a**, therefore: **`output ← a`**



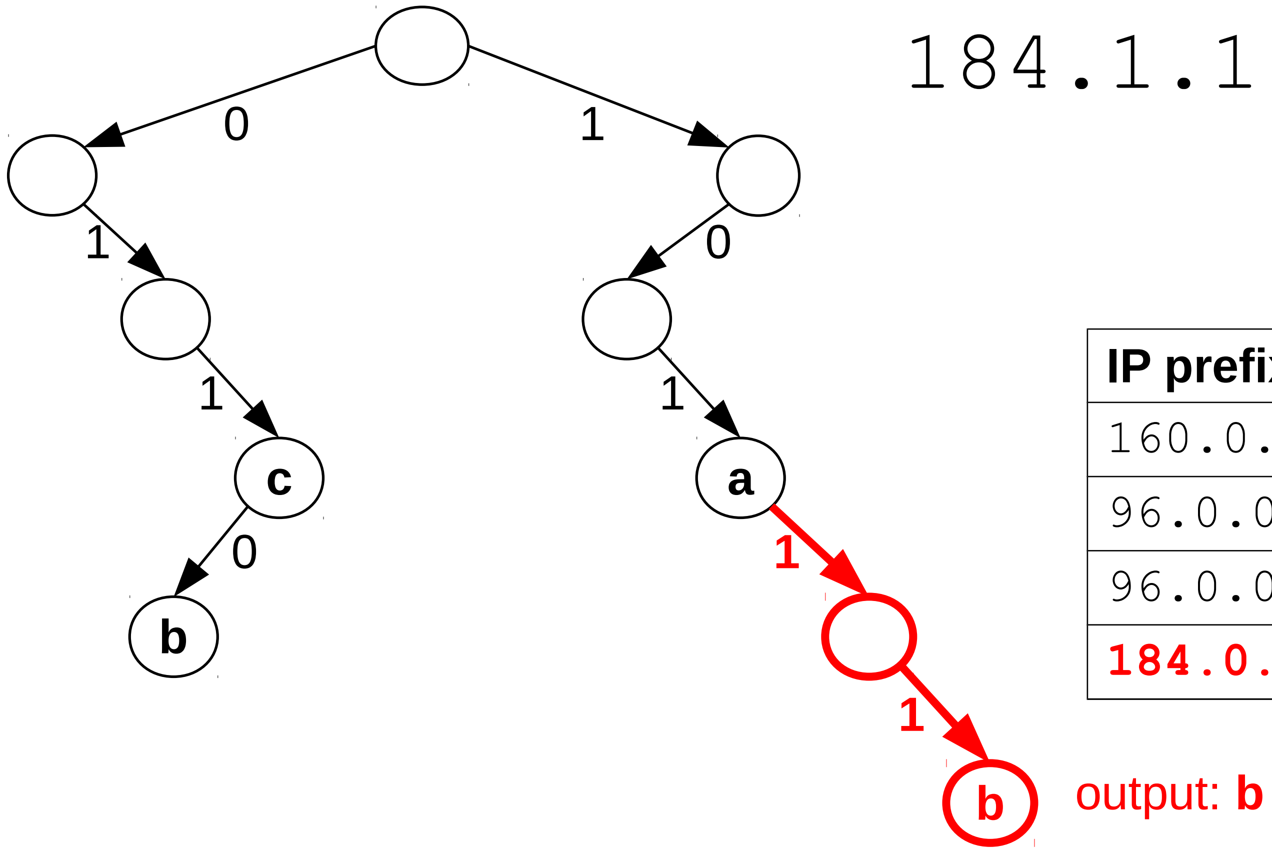
`184.1.1.1=10111...`

FIB

IP prefix	Prefix	Label
`160.0.0.0/3`	`101`	`a`
`96.0.0.0/4`	`0110`	`b`
`96.0.0.0/3`	`011`	`c`
`184.0.0.0/5`	`10111`	`b`

Prefix tree: Lookup

- 4th and 5th bits are `11`, so move twice along arcs of label `1` to a new node with next-hop label **b**: **`output ← b`**
- New node is a leaf: **terminate** with **`output = b`**



184.1.1.1=10111...

FIB

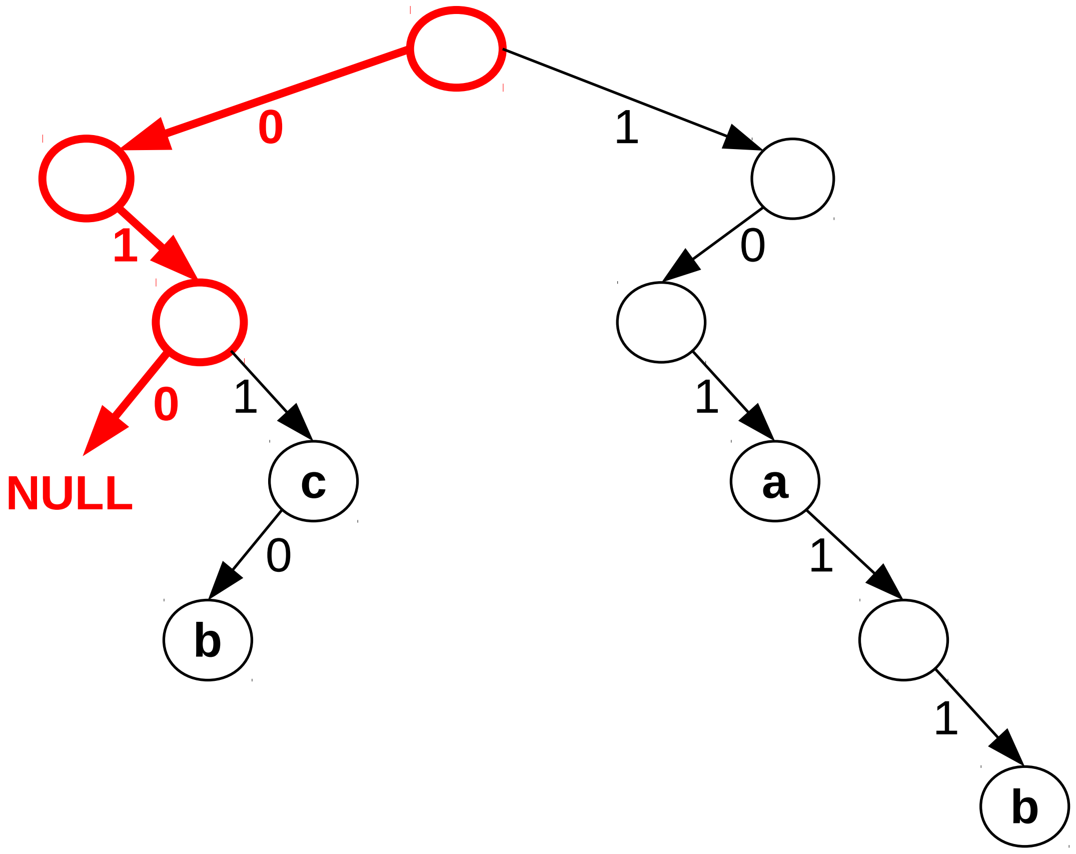
IP prefix	Prefix	Label
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	**10111**	**b**

Prefix tree: Lookup

- **Algorithm:** take all bits of the input IP address
- Proceed to the next node along arc labeled `0` or `1` based on the next-bit of the IP address
- Store the last next-hop label found in a variable `output` (initialized to "invalid" on start)
- **Terminate** if a leaf node or a NULL pointer is encountered and return the current value in `output`
- On exit, read the next-hop from the next-hop index corresponding to the label read from the tree
- In our case the LPM result is: `b→10.0.0.2`

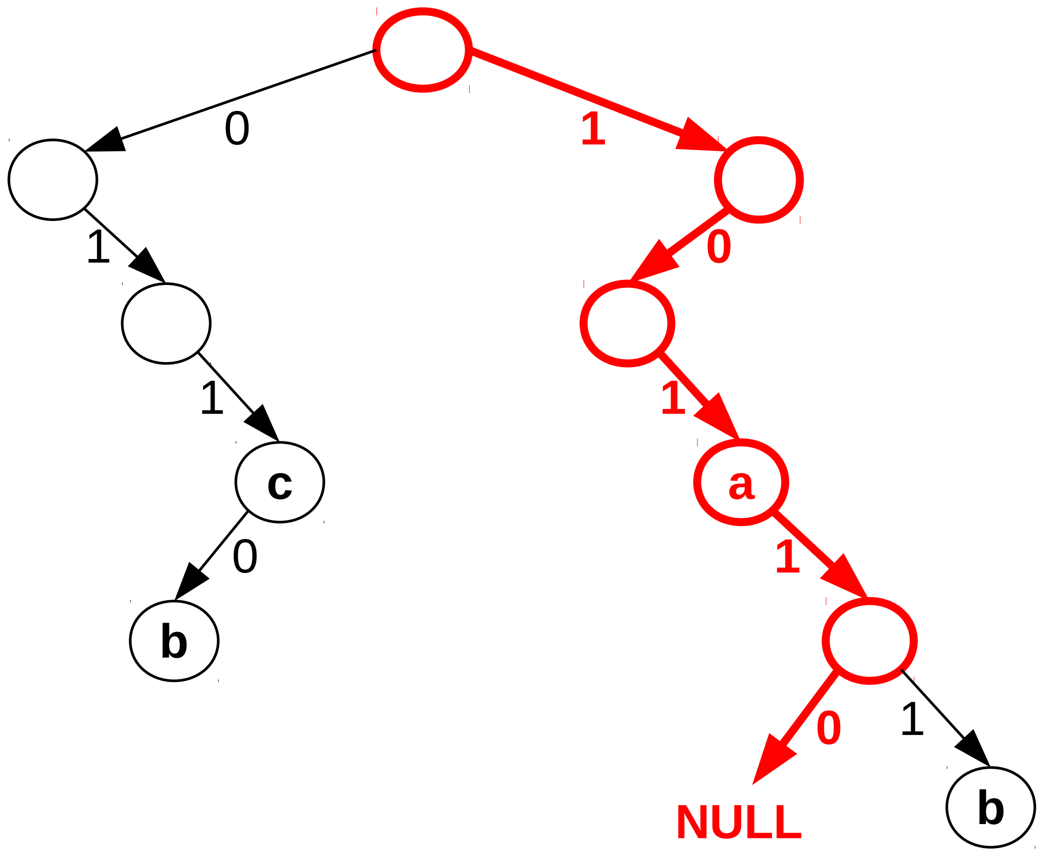
Prefix tree: Lookup

- LPM for the IP address `69.12.75.54=01000...`
- No node with valid label along the path traced out by the input address: **`output = invalid`**



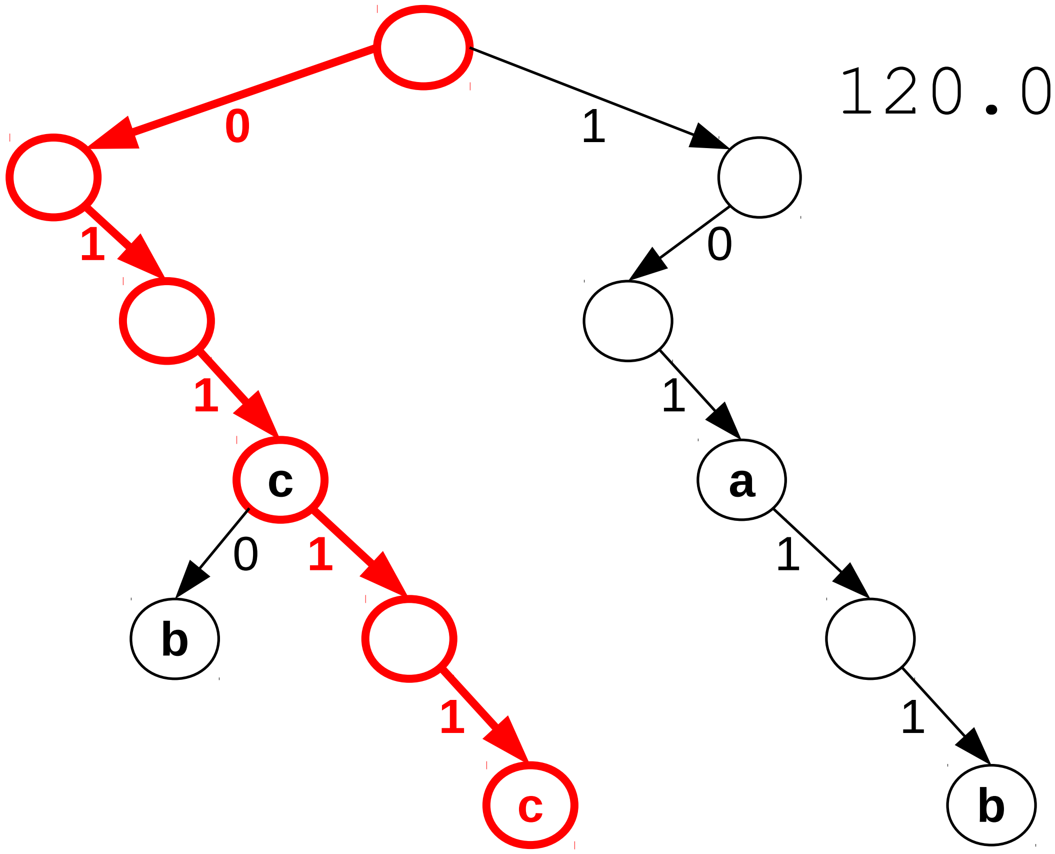
Prefix tree: Lookup

- LPM for IP address `178.4.66.19=10110...`
- Last label encountered is **a**: **`output = a`**



Prefix tree: Insert

- Insert entry `120.0.0.0/5→10.0.0.3` into the FIB
- Follow the path traced out by the prefix, create missing nodes, set the label of the final node to **c**



`120.0.0.0/5=01111...`

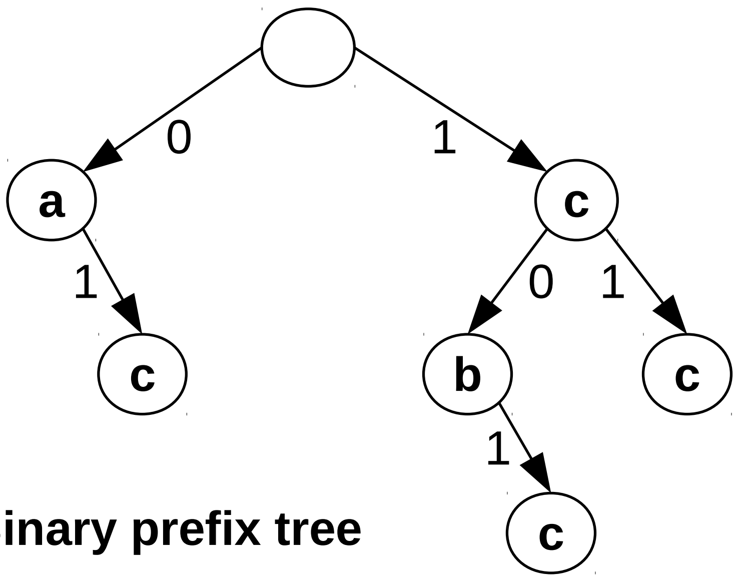
Prefix tree: Other operations

- **Modification** goes along similar vein: follow path traced out by the bits and overwrite label in the resultant node
- **Delete:** similar, but after removing label recursively delete all empty leaves upwards in the tree
- **Complexity:** we terminate in at most as many steps as the number of bits in the input
- **Theorem:** LPM, insert, delete, and modify in a prefix tree terminate in at most *O(W)* steps, where *W* is the width of the address space (IPv4: 32, IPv6: 128)

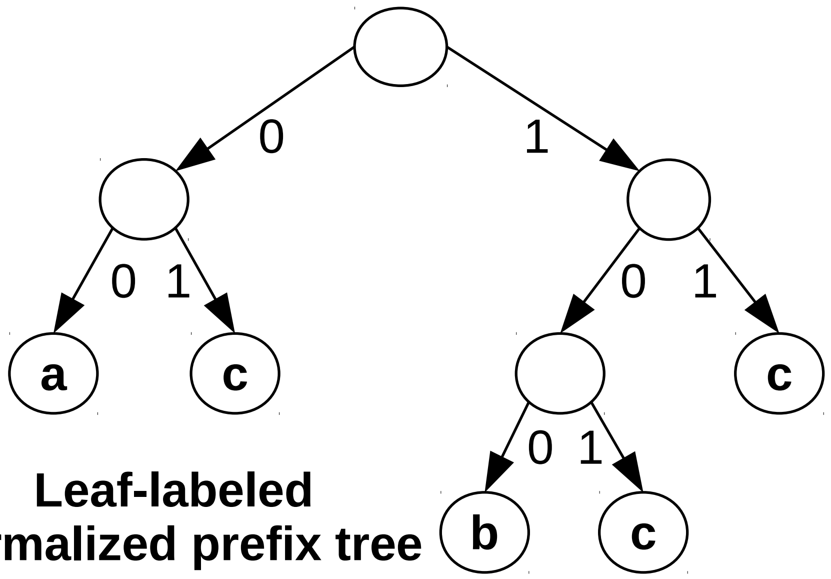
The prefix tree

- In general: in a prefix tree storing *N* prefixes, the complexity of lookup, insert, modify, and delete operations is *O(log N)*
- Recall, that the naïve table-based FIB scheme needed a linear sweep throuh the table: *O(N)* steps
- But 32 RAM accesses (especially if reads do not hit the CPU caches) can still be costly for Gbps line rates
- **FIB aggregation:** convert the prefix tree into a smaller but equivalent (as per LPM) form

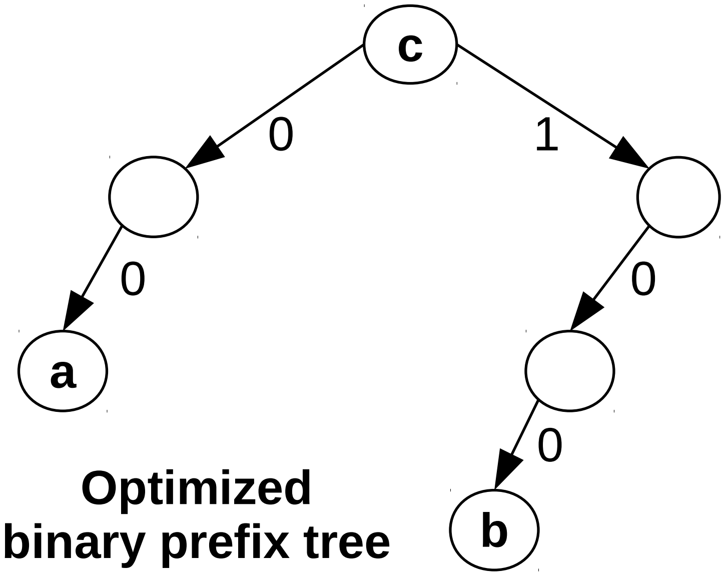
FIB aggregation



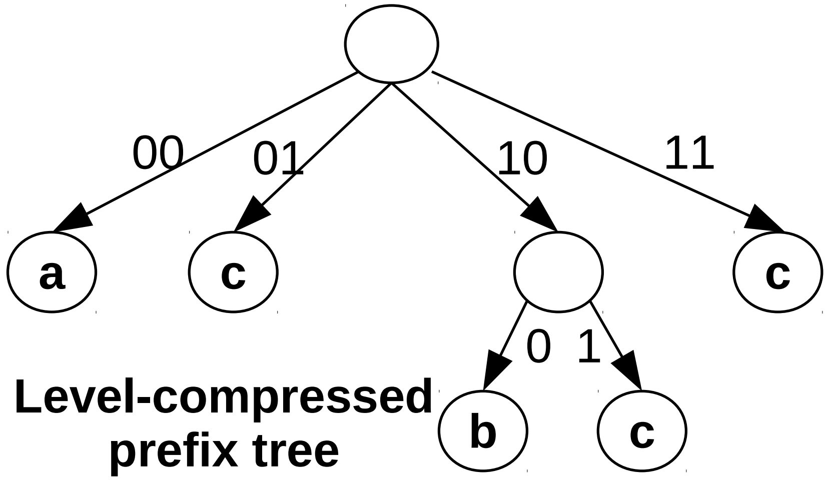
Binary prefix tree



Leaf-labeled normalized prefix tree



Optimized binary prefix tree



Level-compressed prefix tree