

Az internet ökoszisztémája és evolúciója

Tartalom

- Az IP routerek felépítése
 - routerek általános felépítése, linecard/backplane/control, a csomagtovábbítás menete, IP router generációk
- FIB keresés
 - a legspecifikusabb prefix keresési feladat
 - hardver és szoftver megvalósítások: a TCAMek, a prefix fa és FIB aggregáció

Az IP routerek felépítése

Routerek: csomagtovábbítás lépései

- **IP csomag ellenőrzése:** formátum, verzió, headerhossz, opciók, fejrész-ellenőrzőösszeg
- **FIB keresés:** a cél IP címéhez tartozó next-hop keresése a FIBben
- A legspecifikusabb bejegyzést keressük!
- **TTL állítása:** ha $TTL=0$ a csomag eldobása és ICMP üzenet a küldőnek, egyébként $TTL \leftarrow TTL - 1$
- **Fejrész-ellenőrzőösszeg újraszámolása**
- Opcionálisan: fragmentáció, source routing, stb.

Nagyteljesítményű routerek

**Cisco GSR
12416**



Kapacitás: 160 Gb/s
Felvett teljesítmény:
4.2kW

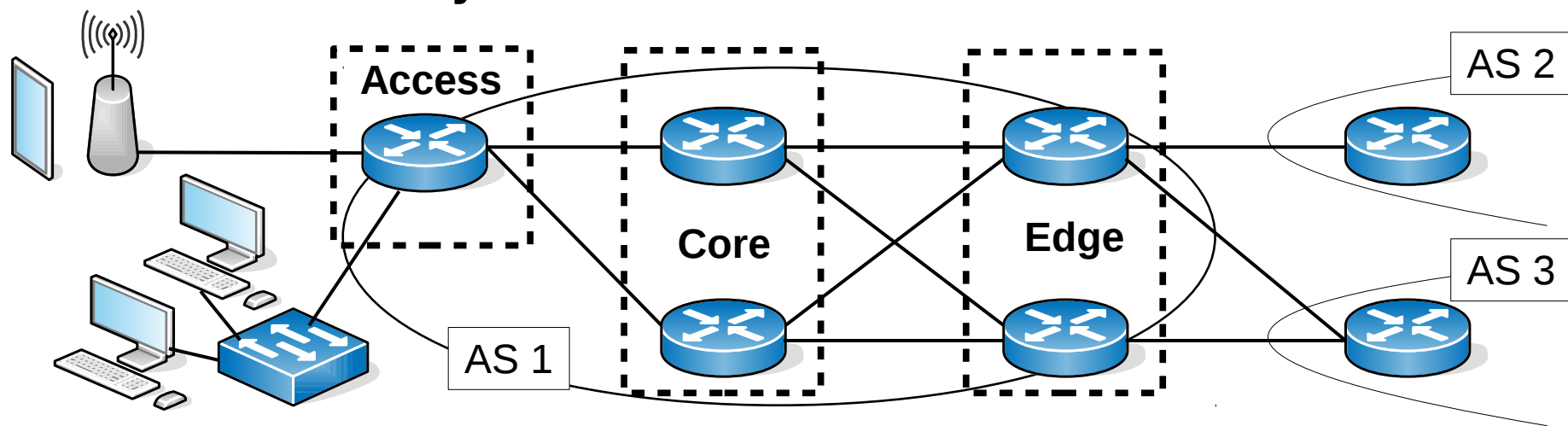
Kapacitás: 80 Gb/s
Felvett teljesítmény:
2.6kW



**Juniper
M160**

Routerek típusai (RFC4098)

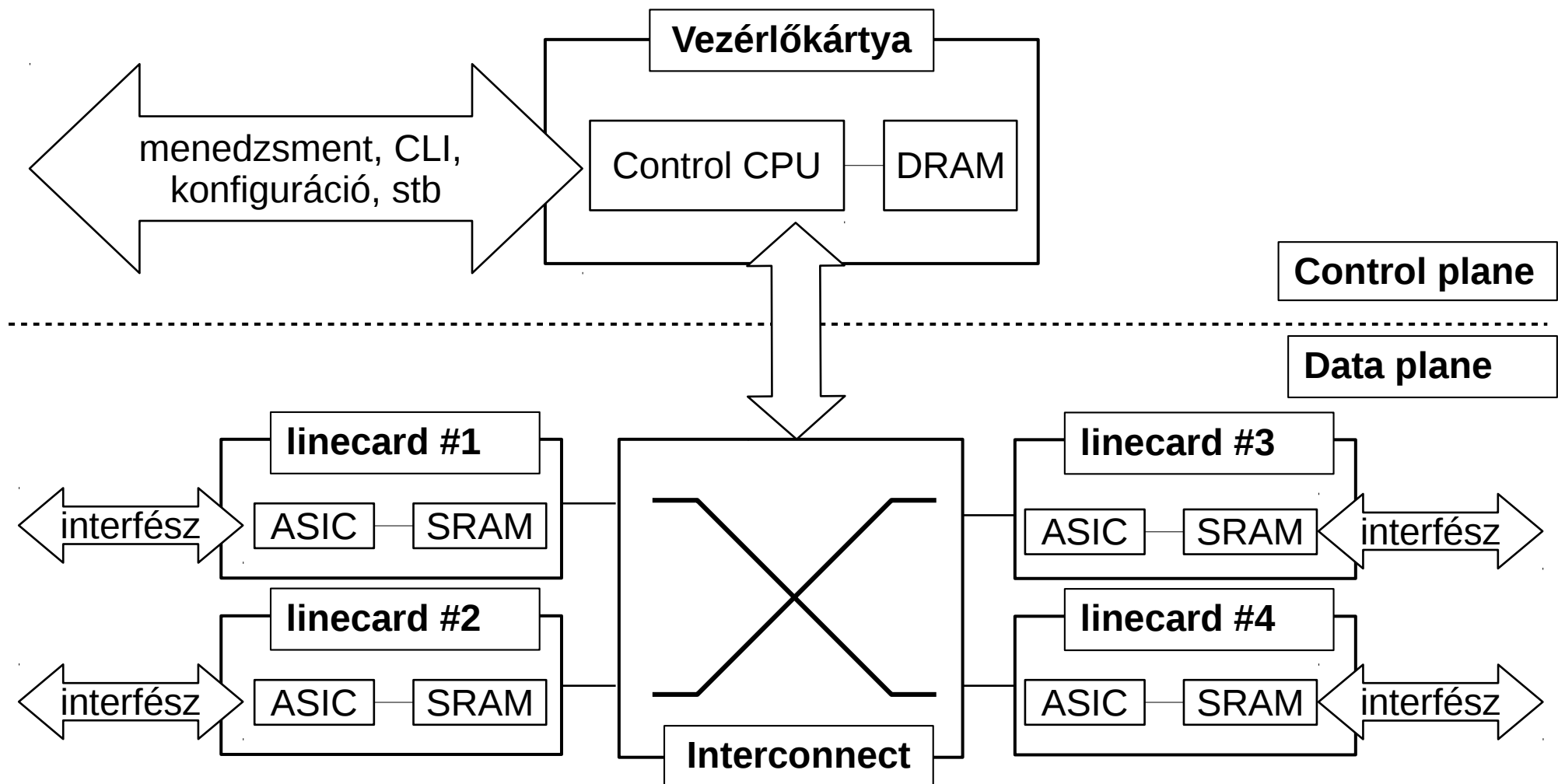
- **Edge/border router:** ASek közti forgalom továbbítása más ASek edge routerei felé (iBGP+eBGP+IGP)
- **Core router:** ASen belüli router POPok közti forgalomátvitelre (IGP+iBGP)
- **Access router:** az Internet edge forgalmának koncentrációja a core felé



Routerek típusai

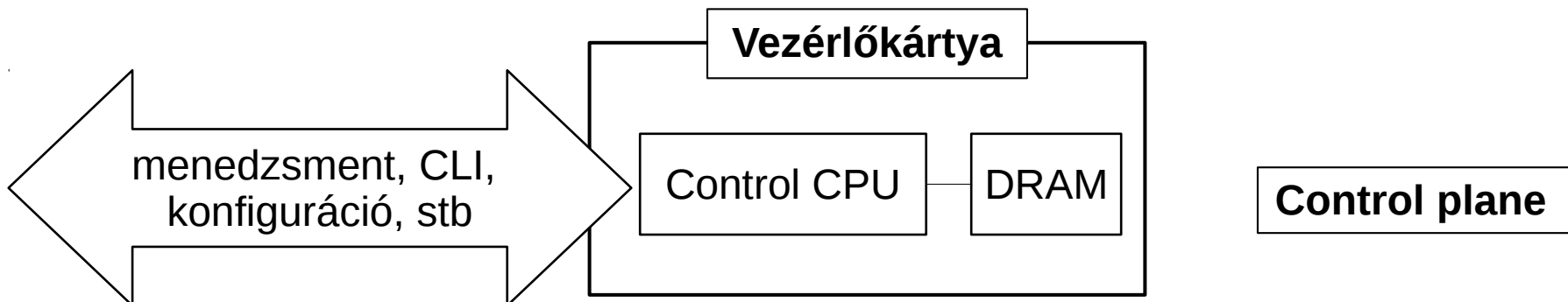
- **Soft router:** szoftverben, általános célú CPU-n megvalósított router
 - például PC + Linux + Quagga
 - kisebb teljesítmény (Intel DPDK!)
 - kis ISP-k, IXP-k(!), BGP monitor, cloud(!)
- **HW router:** speciális célú HW-t tartalmazó nagy teljesítményű router
 - nagy ISP-k core és edge routerei
 - sok előfizetőt kiszolgáló access routerek

Általános felépítés



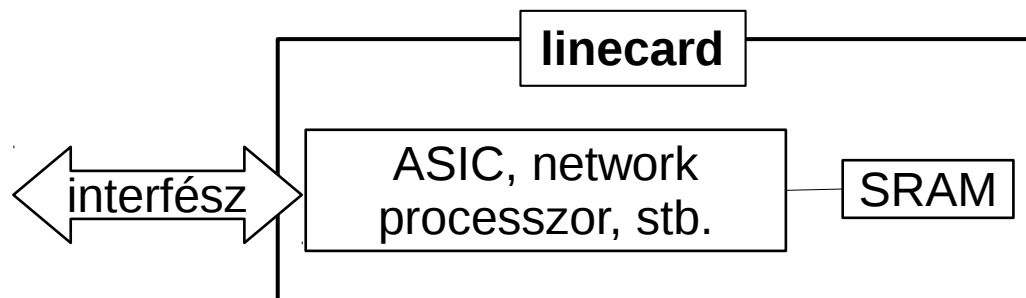
Általános felépítés

- **Vezérlőkártya:** a router logikája
 - routing protokollok futtatása, menedzsment hozzáférés (CLI (Command Line Interface), SNMP, stb.), monitoring, extra szolgáltatások
 - vezérli az interconnect-et, beállítja a FIBet
 - általános célú CPU/DRAM, néha általános célú operációs rendszer (pl. Linux!)



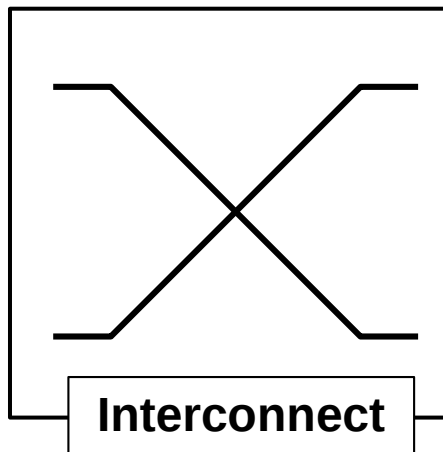
Általános felépítés

- **Interfészkártya (linecard):** csomag adás/vétel
 - egy vagy több fizikai csatoló (interfész) a linkre (Serial/FastEthernet/GigabitEthernet)
 - alapvető csomagtovábbítási funkciók
 - speciális célú HW és gyors statikus memória
 - legtöbb router bővíthető új kártyákkal



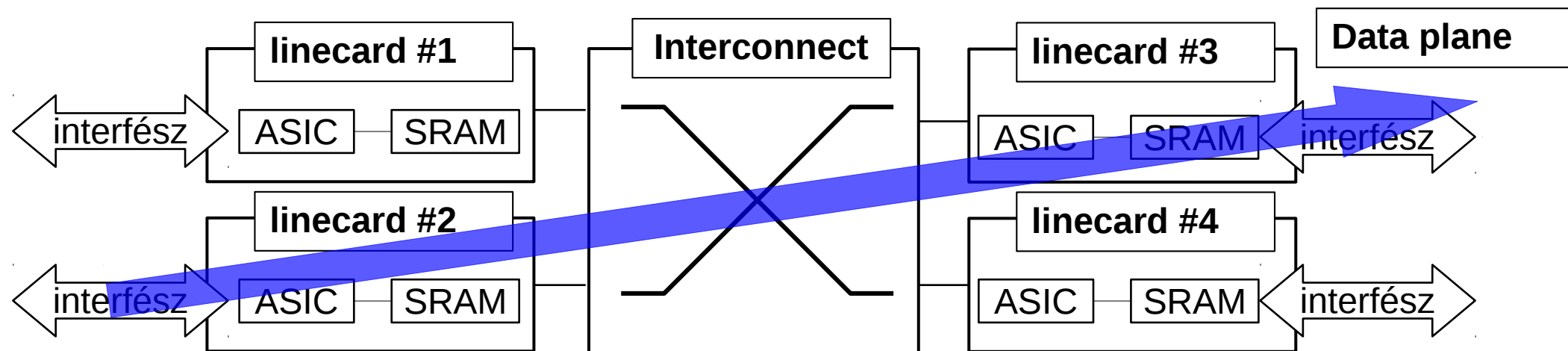
Általános felépítés

- **Interconnect/backplane:** kapcsolólogika
 - interfészkártyák és a control CPU közti kommunikáció + puffereelés
 - osztott busz/belső switch (akár Ethernet)
 - input puffer (head of line blocking!)/output puffer/osztott memória



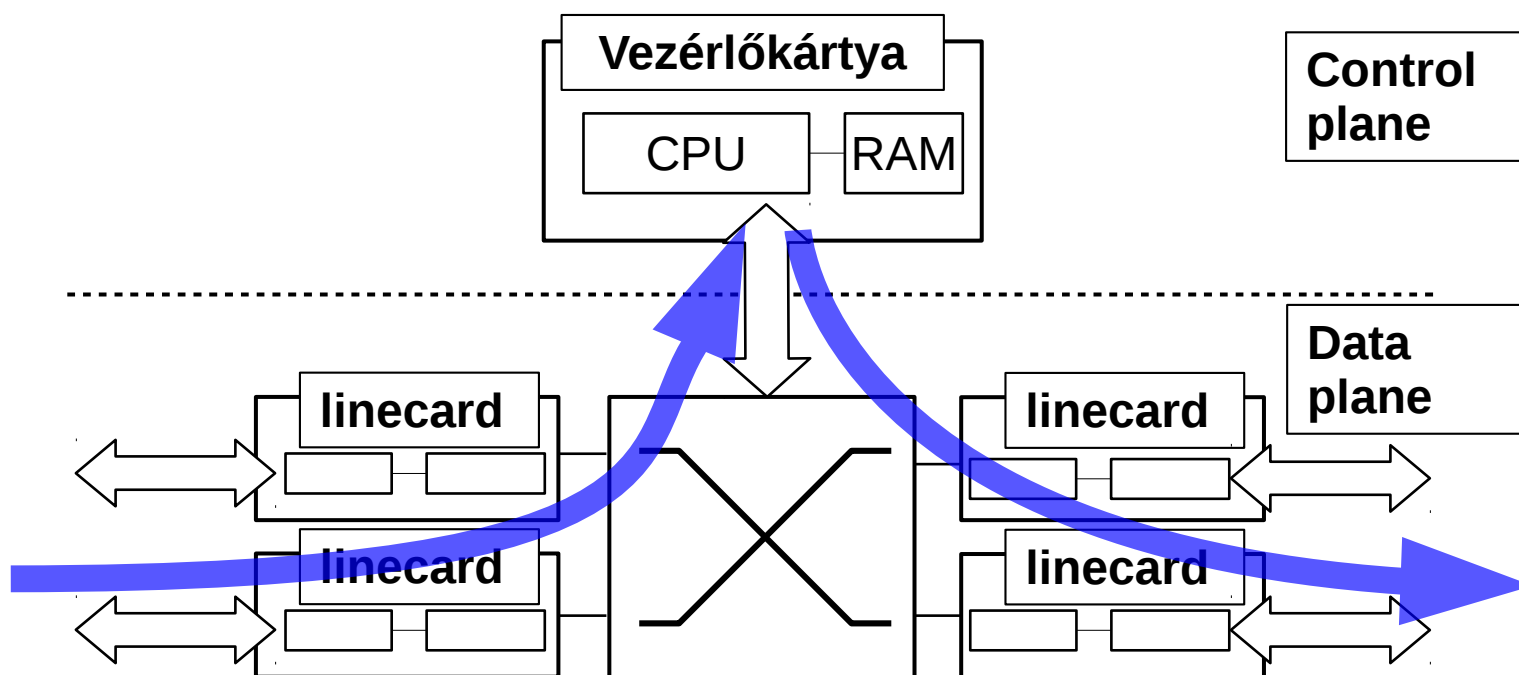
Fast path

- **Fast path:** a csomagtovábbítás **Data plane**-en megvalósított lépései (nagy sebesség!)
 - HWben könnyen megvalósítható műveletek
 - célcím olvasása, ellenőrzőösszeg számítása
 - sokszor a FIB keresés is (de ehhez le kell tölteni a FIBeket az interfészkártyákra!)



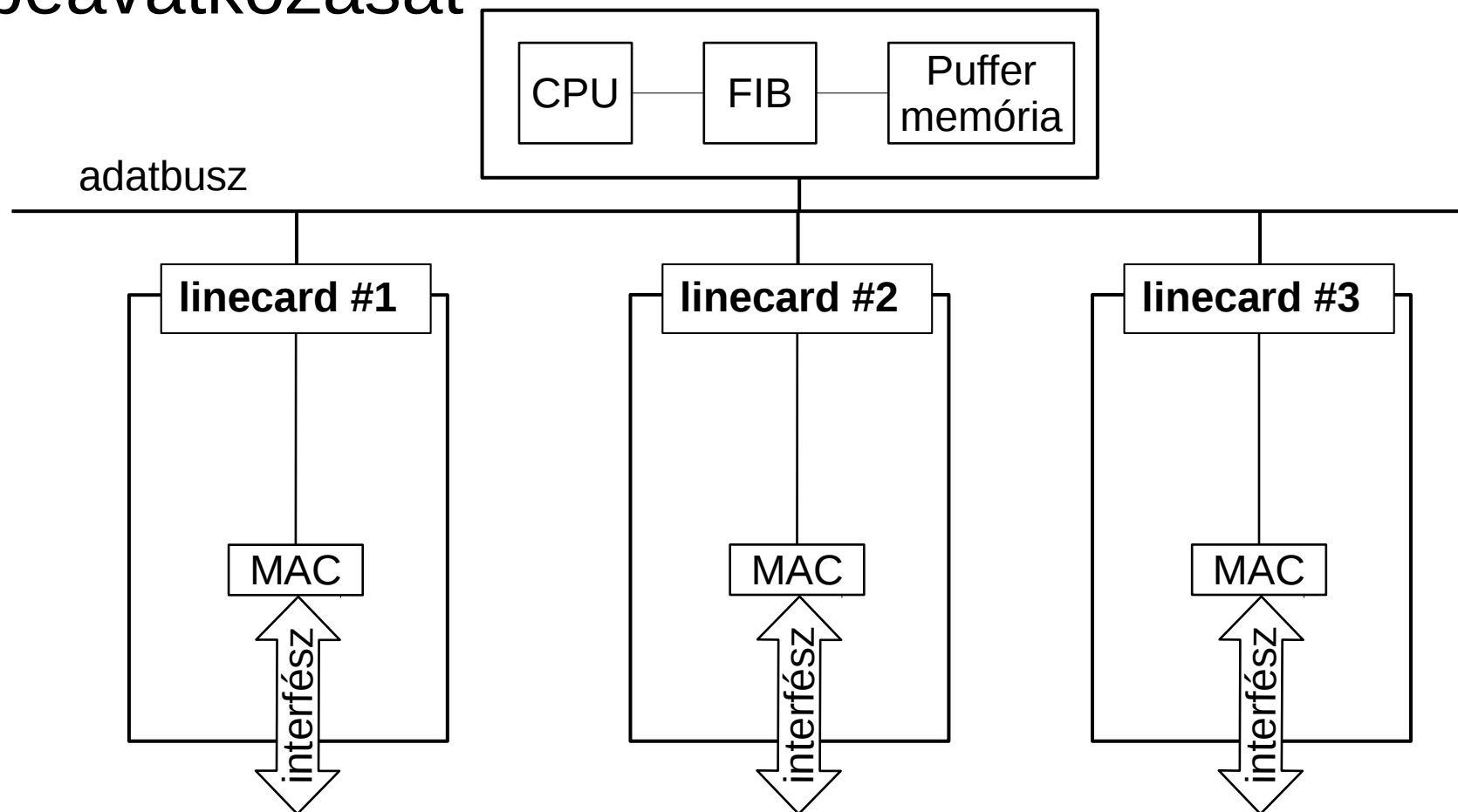
Slow path

- **Slow path:** a vezérlősík beavatkozását igénylő „bonyolultabb” funkciók (csökkent sebesség!)
- IP opciók, fragmentáció, protokollüzenetek kezelése, ARP, ICMP generálás, stb.



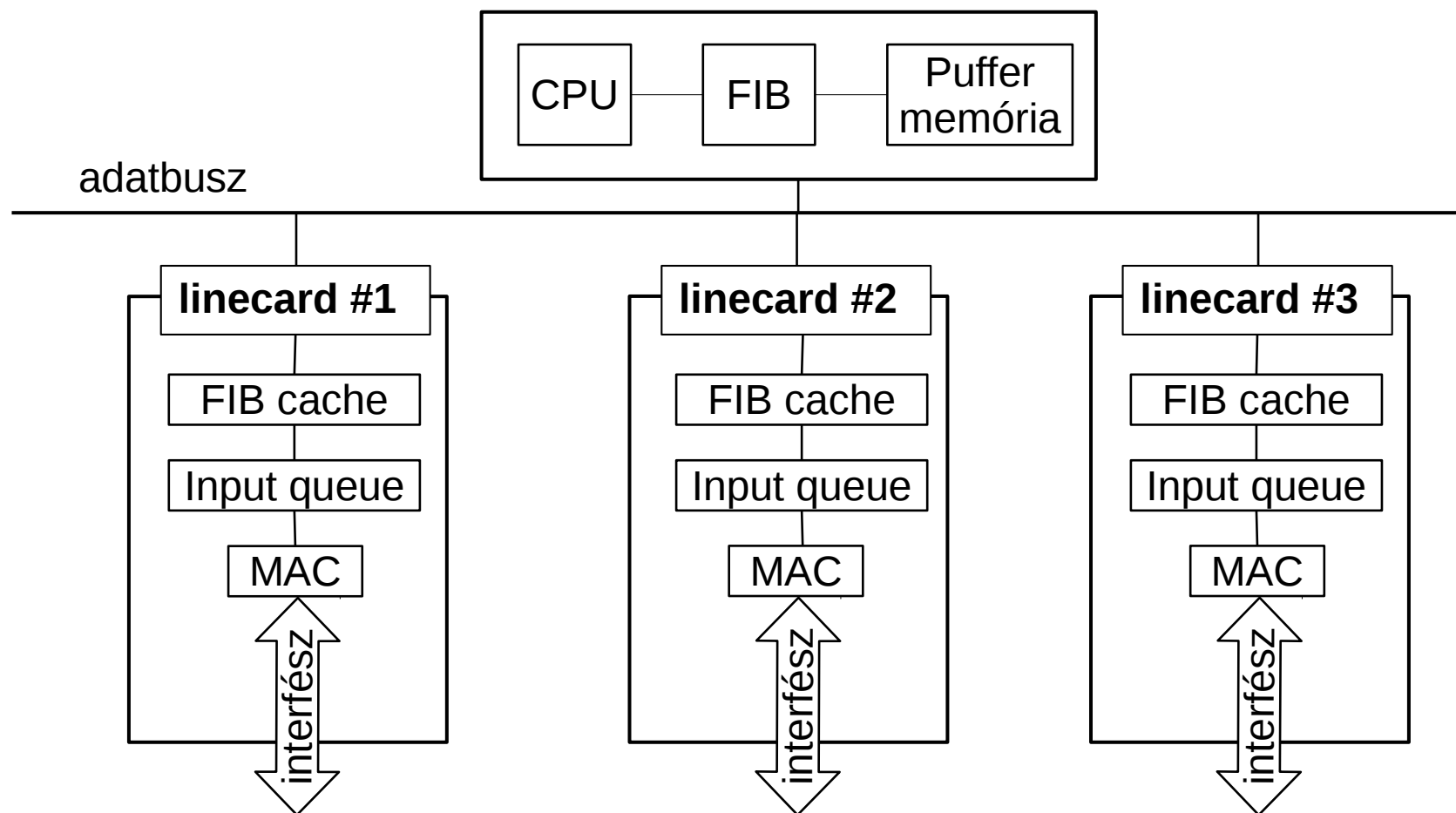
Routerek fejlődése: 1. generáció

- A hálózati csatoló kivételével minden a slow path-en, minden csomag igényli a vezérlőszík beavatkozását



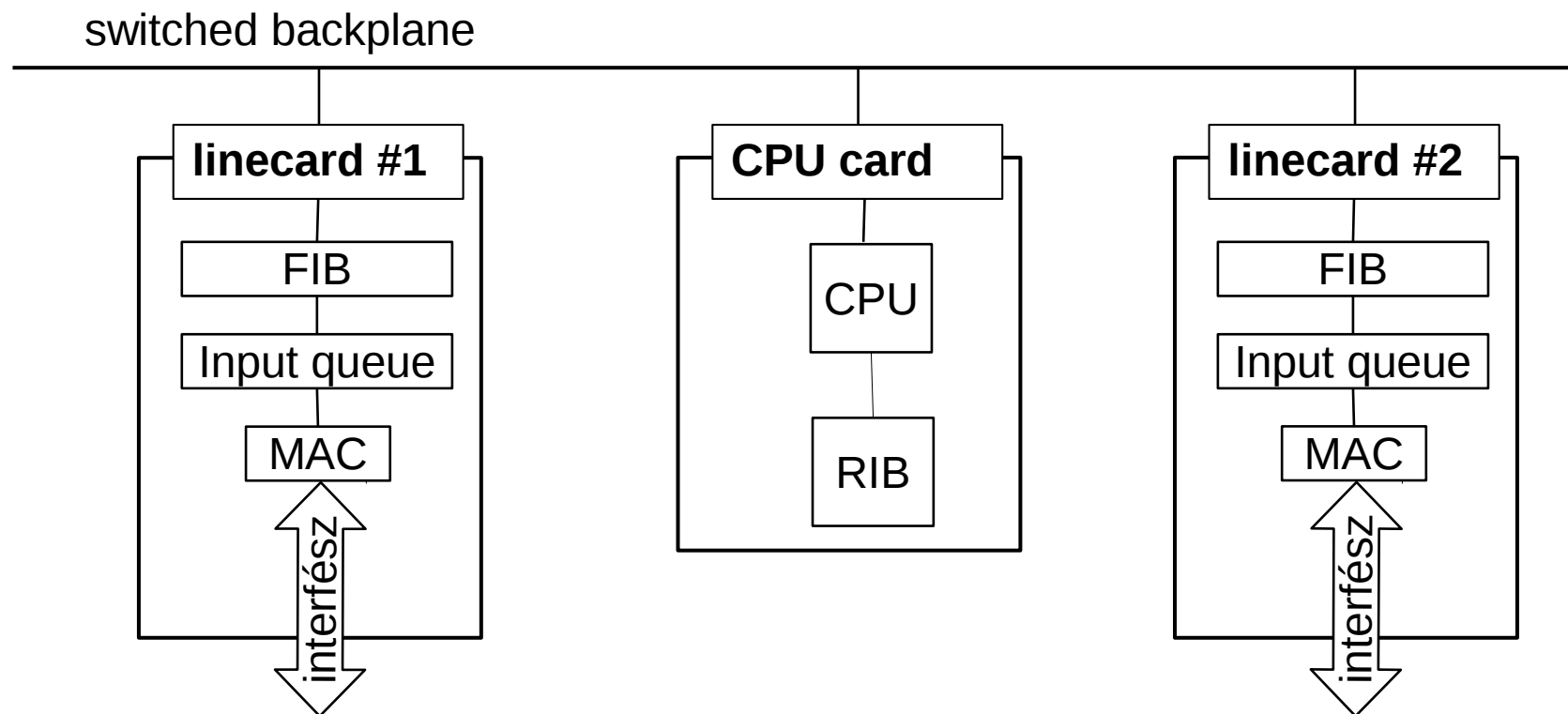
Routerek fejlődése: 2. generáció

- Az interfészkártyán: input puffer + FIB cache
- Fast path továbbítás, ha a célcím a cache-ben



Routerek fejlődése: 3. generáció

- Az interfészkártyán a teljes FIB, szinte minden csomag marad a fast path-en
- A CPU csak egy másik kártya a chassis-ben



Routerek fejlődése: a jövő?

- A mai általános célú CPUk gyakran vannak olyan gyorsak, mint egy ASIC
 - ráadásul sokkal olcsóbbak is
 - Moore-törvénye miatt gyorsan fejlődnek
 - a csomagtovábbítás masszívan párhuzamos feladat: csomagonként akár külön CPU végzi
- A fejlődés az olcsó, masszívan parallel, virtualizált szoftver-routerek felé mutat
- Akár nem is különálló doboz, hanem a cloud-ban fut (Intel DPDK)!

FIB keresés

FIB keresés

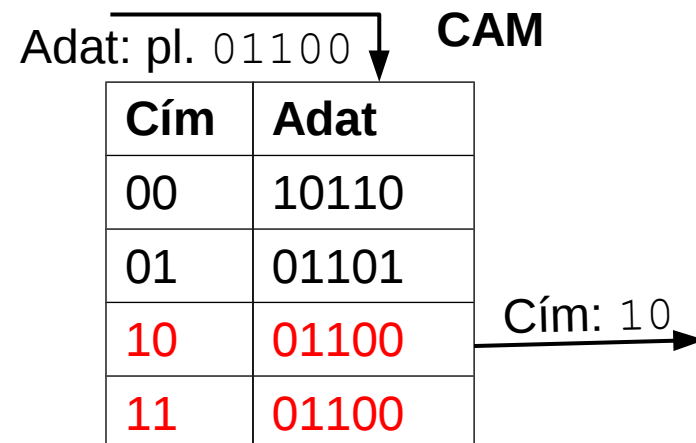
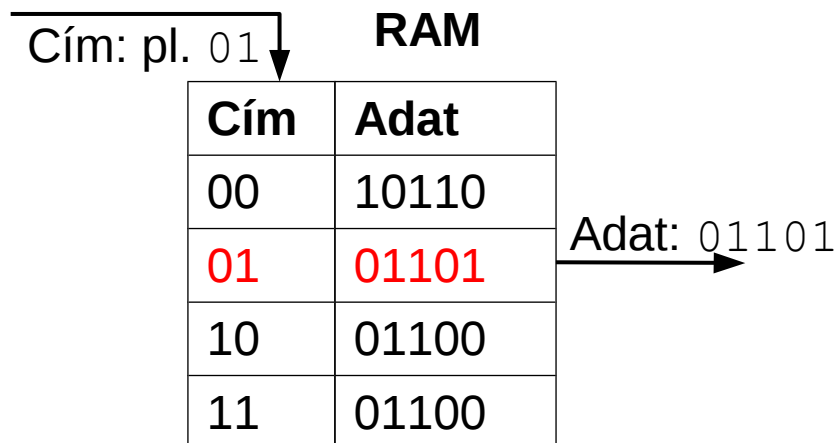
- Az IP csomagtovábbítás legidőigényesebb feladata: a csomagban található célcím alapján meg kell találni a megfelelő next-hopot
- A legspecifikusabb bejegyzés kell (LPM)
- **FIB (Forwarding Information Base):** a csomagtovábbítási szabályok összessége
- Az útvonalválasztó protokollok adatai alapján a control CPU tölti le az interfészkártyákra
- Így azok a control CPU igénybevétele nélkül, fast path-en továbbítják a csomagokat

FIB keresés: LPM

- A LPM megvalósítása nem triviális
- Egy naiv megközelítés végigkeresné az összes bejegyzést a FIBben és megjegyezné a legtöbb biten illeszkedőt
- $O(N)$ komplexitás, ha a bejegyzések száma N
- A gyakorlatban $N \approx 10^6$, a rendelkezésre álló időkeret pedig 500 byte-os csomagokkal számolva 10Gbit/s sebességen 1GHz-es órajelen alig 400 órajel
- A naiv módszer használhatatlan

Tartalomcímezhető memóriák

- **Content Addressable Memory (CAM)**
- Pontosán a klasszikus memóriák ellentéte:
 - RAM: cím alapján megadja a tárolt adatot
 - CAM: az adathoz megkeresi annak címét
 - ha több adat stimmel, az első címét adja meg



Háromállapotú CAM: TCAM

- **Ternary Content Addressable Memory:** a CAM csak 0 és 1 értékű biteket ismer, a TCAM mintákban beállítható "Don't care" bit is (*)
- Például az 101^{**} TCAM minta illeszkedik 10100 , 10101 , 10110 és 10111 lekérdezésekre
- * bárhol szerepelhet, nemcsak a minta végén!
- Alacsonyabb címen levő minták előbb illesztve
- Illeszkedő minta esetén kiszállítás a címmel
- Az alacsonyabb címen levő minták felülbírálják a magasabb címen levő mintákat: **prioritás**

Háromállapotú CAM: TCAM

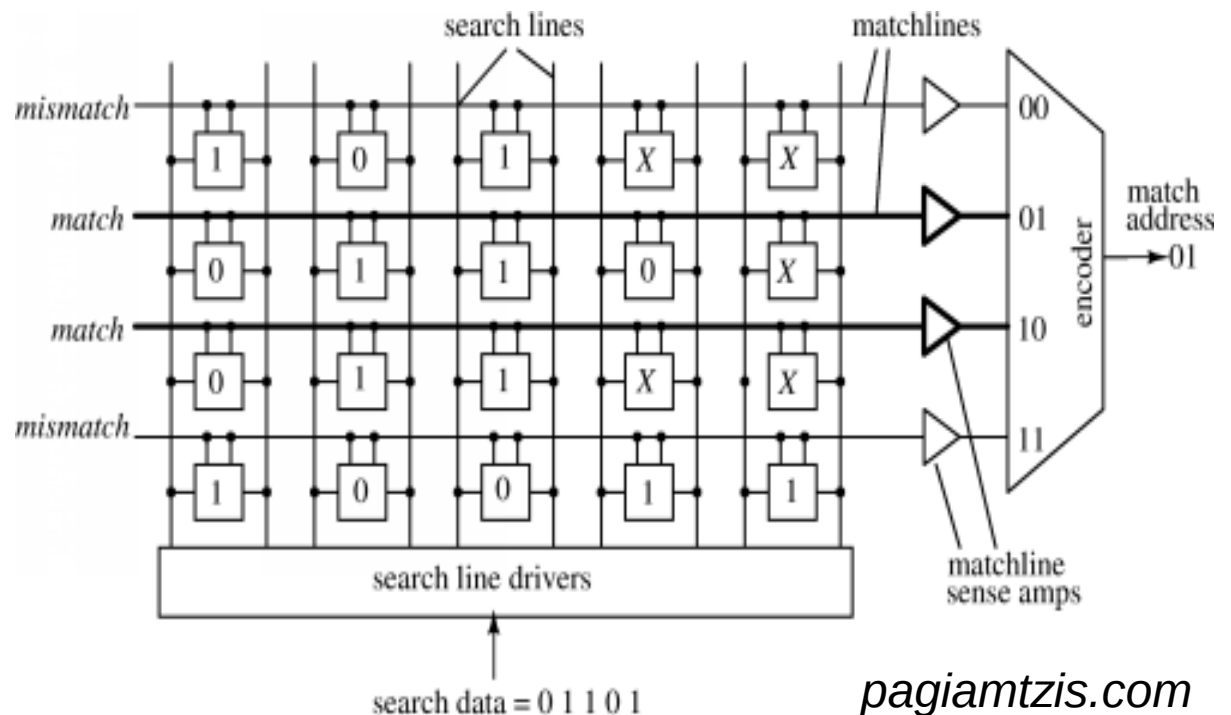
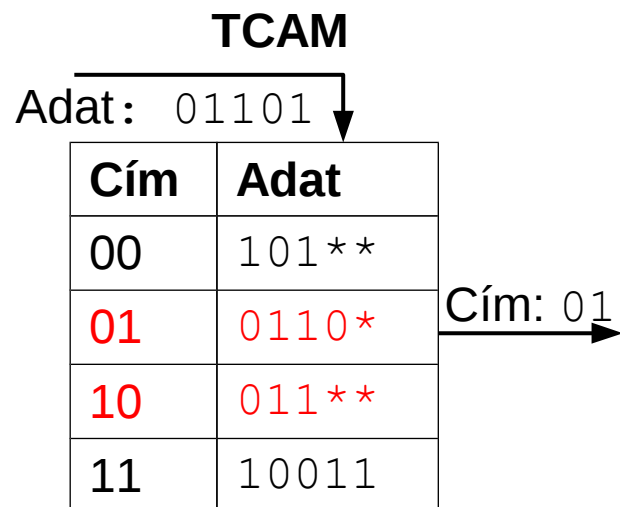
- A 00110 keresésre a 00 és az 11 címen levő minta is illeszkedik, az első preferált, output: 00
- Az 11111 keresésre az eredmény: 10
- 01110 keresésre a 10 és az 11 címen levő minta is illeszkedik, output: 10

Adat		TCAM	
Cím	Adat		
00	*01**		
01	0110*		
10	*111*		
11	0*110		

Cím

TCAM: megvalósítása

- *(Mintaszám * bitszélesség)* számú cella, mindegyikben 0 / 1 / * tárolva és egy komparátor
- A cellák egyszerre végzik az összehasonlítást
- Kimeneti logika választja ki a legkisebb címet



TCAM: a LPM megvalósítása

- Tekintsük az alábbi FIBet

IP prefix	Bináris 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

- Minden csomaghoz a cél IP címére a legtöbb biten illeszkedő FIB-bejegyzést keressük
- A 96.128.59.12 IP címre a 2. és 3. bejegyzés is illeszkedik, de a 2. több biten, így az preferált
- LPM eredménye: a 2. bejegyzés-beli next-hop

TCAM: a LPM megvalósítása

- Használjunk az LPM megvalósítására TCAMet
- Az alhálózati prefix bitjeit pontosan megadjuk
- Ezeket a biteket a TCAM pontosan illeszti
- A hoszt-azonosítót * bitekkel helyettesítjük
- Hisz ezen bitek nem számítanak az LPM-ben
- Így csak az utolsó pozíciókban állhat *

Cím	IP prefix	Illesztendő TCAM minta	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

TCAM: a LPM megvalósítása

- **Probléma:** ha ebben a sorrendben írjuk a TCAMbe a mintákat, egy kevésbé specifikus prefix felülírhatja a specifikusabbat (pl. a 184.1.1.1 címre az 1. minta lenne a kimenet, de a 4. a jó LPM találat, hiszen hosszabb)
- **Megoldás:** rendezzük át a bejegyzéseket a prefix-hossz szerinti csökkenő sorrendbe
 - a legspecifikusabb bejegyzésekhez tartozó minták (vagyis a leghosszabb prefixek) kerülnek alacsony címekre = magas prioritás
 - kevésbé specifikus prefixek: magasabb cím

TCAM: a LPM megvalósítása

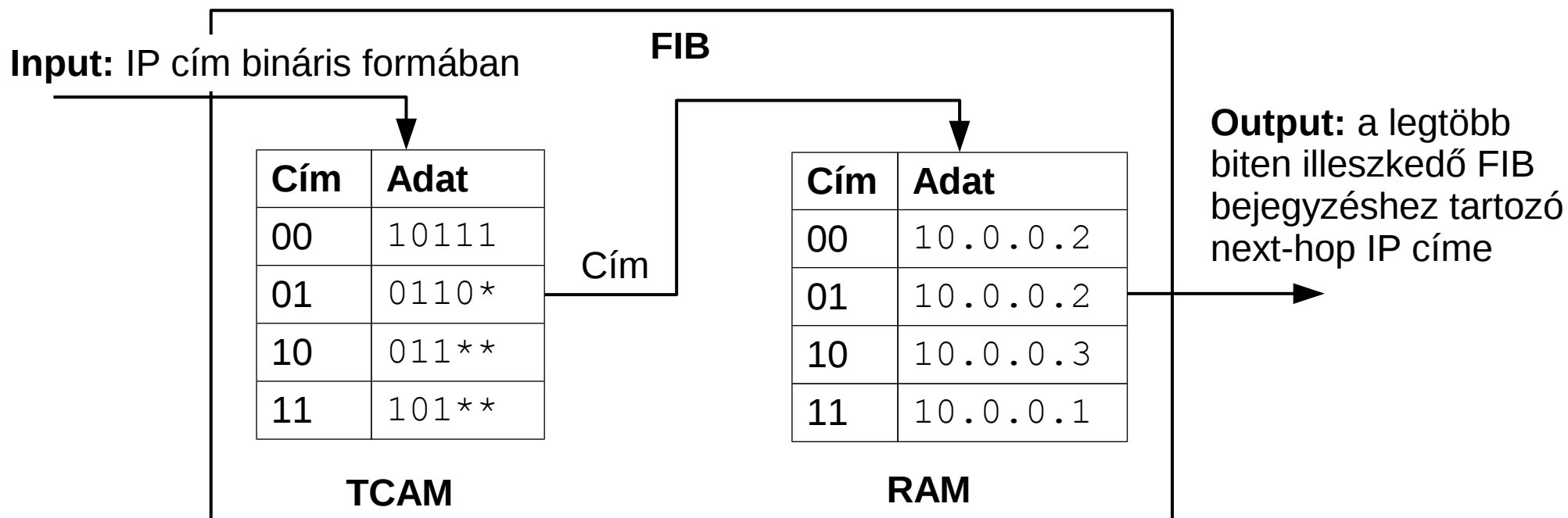
- A FIB bejegyzések szabadon átrendezhetők
- Hiszen pontos prioritást ad a prefix-hossz
- A prefix-hossz szerinti csökkenő sorrendbe rendezés után kapott táblázat

Cím	IP prefix	Illesztendő TCAM minta	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

- A pirossal jelzett oszlopok TCAMbe írhatók
- A maradék oszlopokat RAMban tároljuk

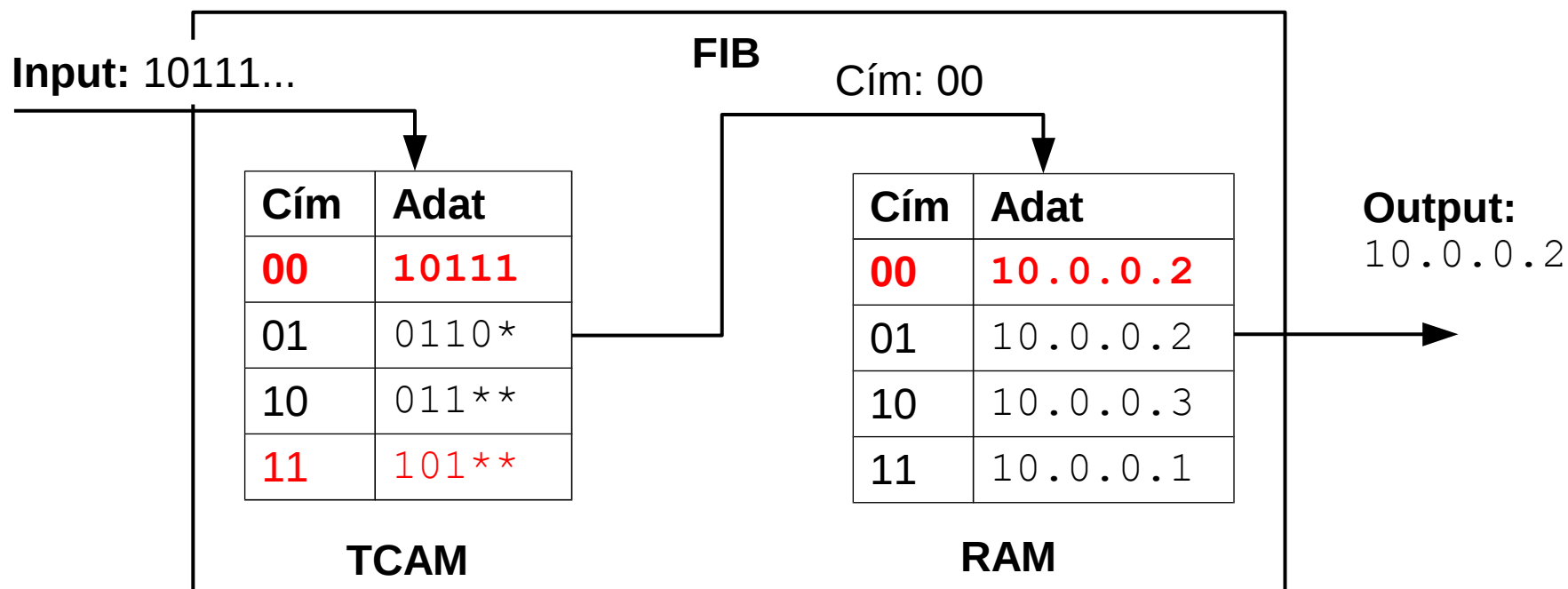
TCAM: a LPM megvalósítása

- **HW FIB:** TCAM és RAM összekapcsolása
- A TCAMból kiolvasott címmel címezzük a RAMot, amelyből kiolvassuk a next-hop címét
- Elég 5 bit széles TCAM (max. prefix-hossz)



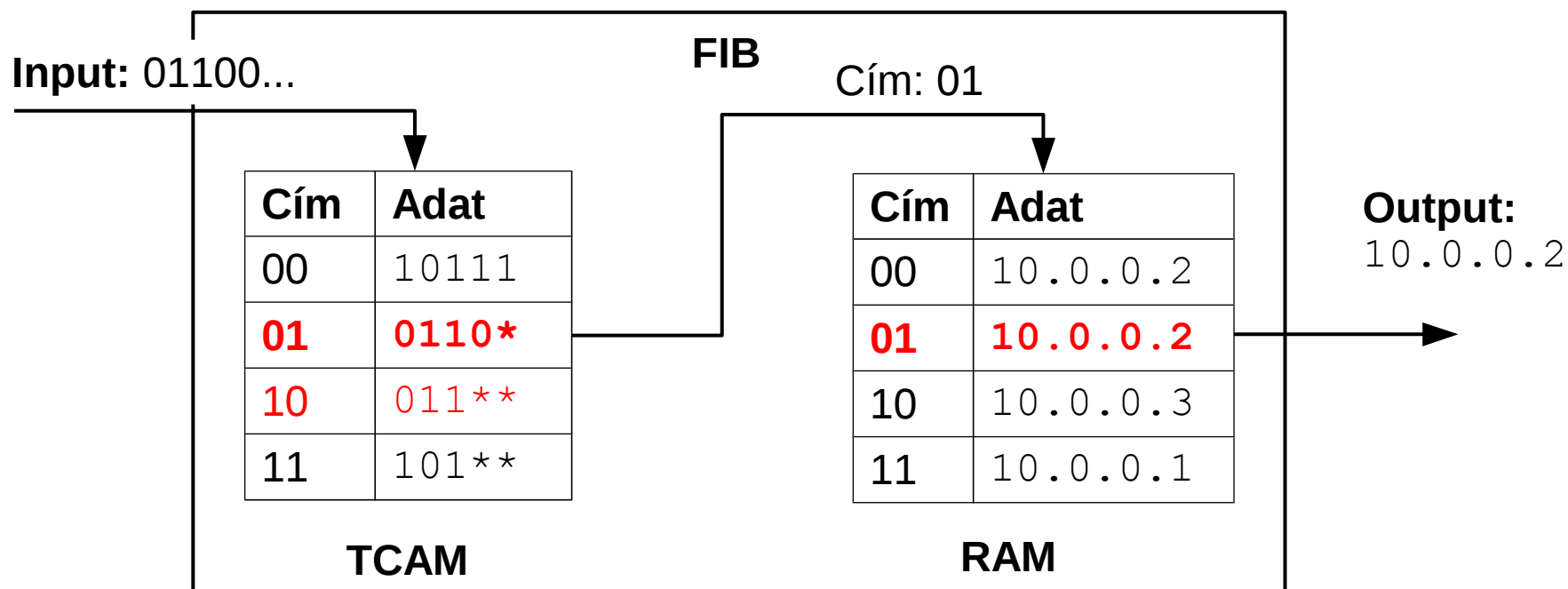
TCAM: a LPM megvalósítása

- A 184.1.1.1=10111... IP címre a TCAM a 00 címet adja
- A RAMból a 00 címen kiolvassuk a next-hopot



TCAM: a LPM megvalósítása

- A 97.12.124.45=01100... IP címre a 2. és a 3. TCAM bejegyzés is illeszkedik
- A TCAM a 01 címet adja, a next-hop: 10.0.0.2



TCAM: a LPM megvalósítása

- A routerekben alkalmazott HW ASICek a TCAMet és a RAMot is tartalmazzák (egyben)
- Pár órajel alatt kész az LPM keresés: nagyon gyors fast-path IP csomagtovábbítás
- Elterjedt egyéb feladatokra is: Ethernet MAC learning tábla, firewall szabályok kódolása, stb.
- De a TCAM bonyolult: 16 tranzisztor/cella (SRAM: 6, DRAM: 2 tranzisztor/cella): drága!
- Magas a fogyasztása (9MB TCAM csip, 100 MHz órajel, 10–15W disszipáció): melegszik!

A LPM megvalósítása SWben

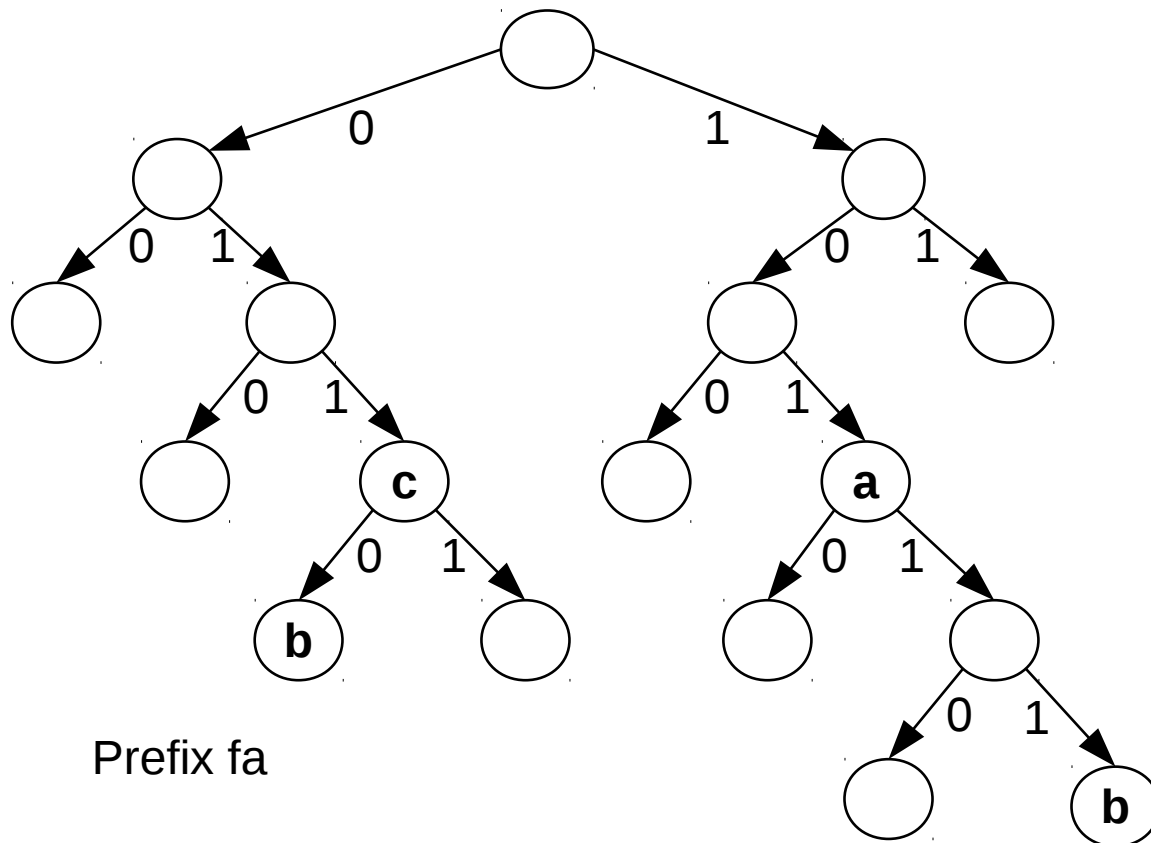
- Nem minden felhasználásra célszerű a TCAM: soft routerek, egyszerű access routerek (pl. SOHO router), virtualizált routerek (cloud)
- Hasznos lenne egy olyan FIB adatstruktúra, amely gyors keresést tesz lehetővé általános célú CPU-n implementálva
- Egy általános CPU-n a legköltségesebb művelet a memóriáhozáférés
- **Cél:** a LPM megvalósításához szükséges RAM olvasások számának minimalizálása

A (bináris) prefix fa

- LPM keresésre optimalizált adatstruktúra
- A **prefix fa** a következő műveleteket támogatja:
 - **keresés:** adott mintára legtöbb biten illeszkedő prefix megtalálása és a hozzá tárolt címke kiolvasása
 - **beszúrás:** (prefix → címke) páros beillesztése
 - **törlés:** prefix és hozzá tartozó címke törlése
 - **módosítás:** prefix címkéjének módosítása

A prefix fa

- Tekintsük a korábbi FIBet és bontsuk két részre
- A next-hopokat azonosítsuk **egyedi címkékkel** és tároljuk külön egy **next-hop index** táblában



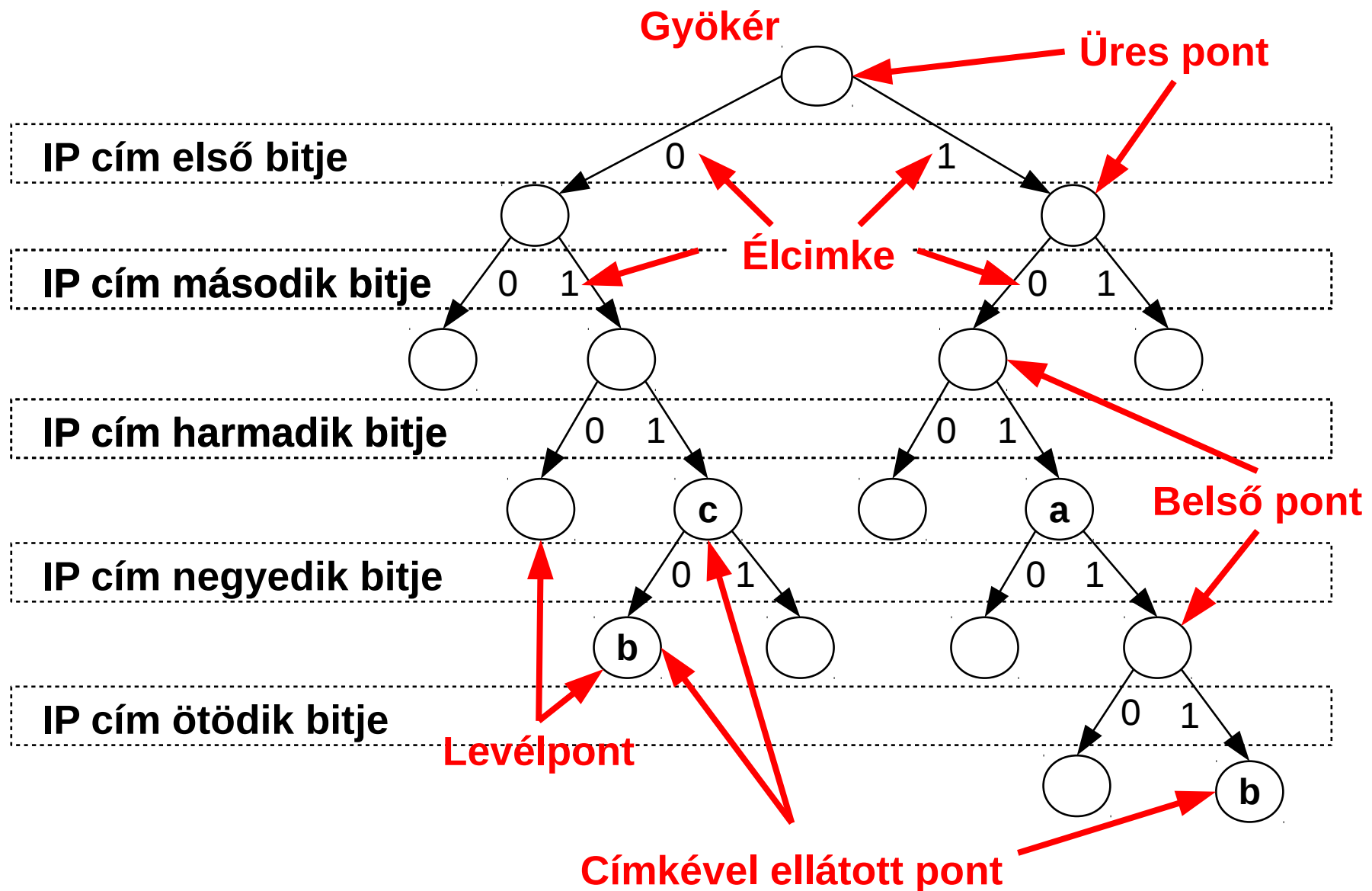
FIB

IP prefix	Prefix	Címke
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

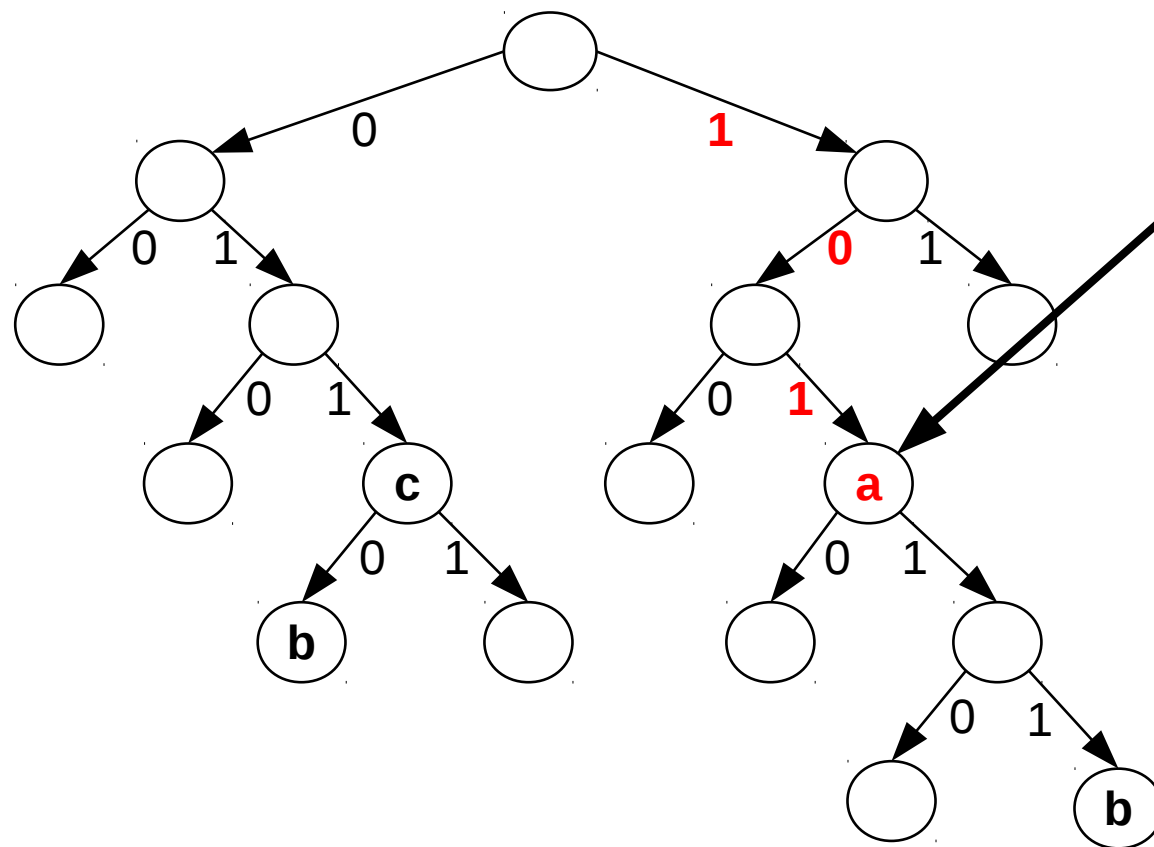
Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa



A prefix fa

- **Prefix=pontba vezető út élcímkeinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük

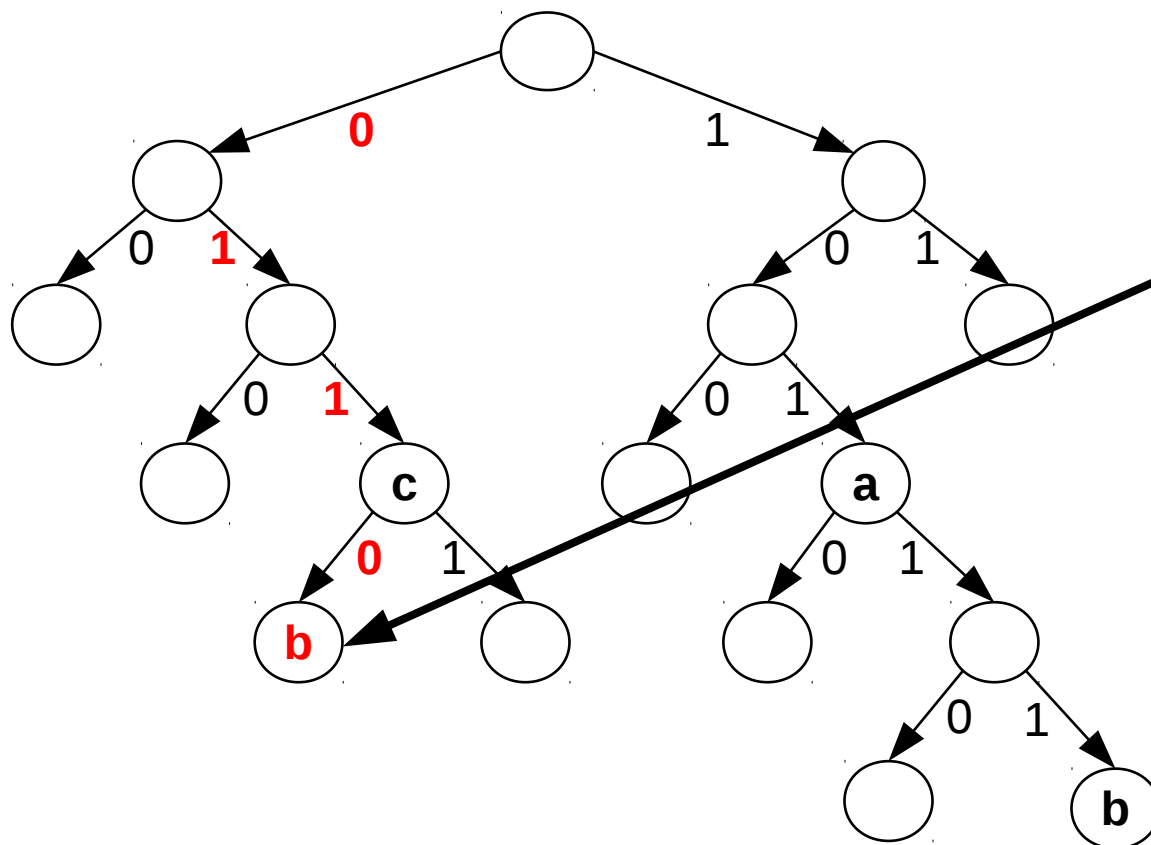


IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index	
Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa

- **Prefix=pontba vezető út élcímkeinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük



FIB

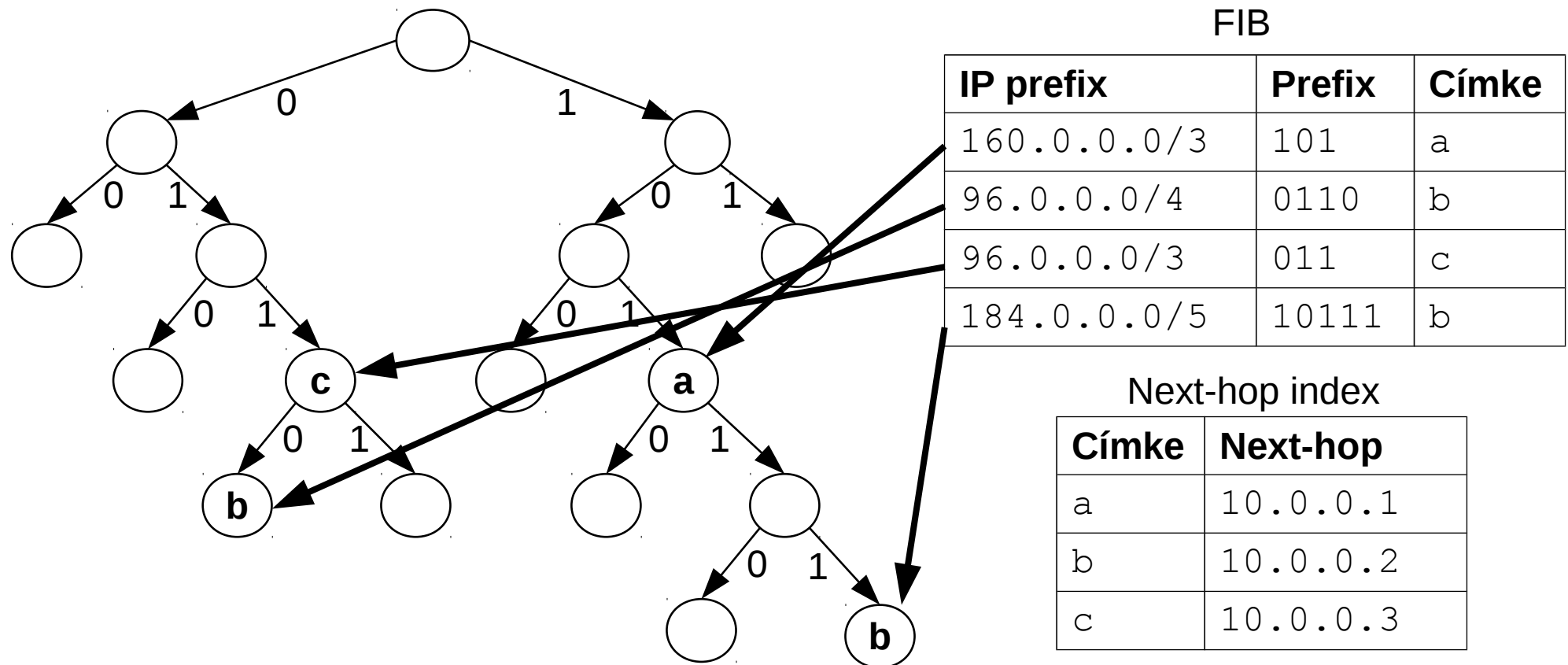
IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

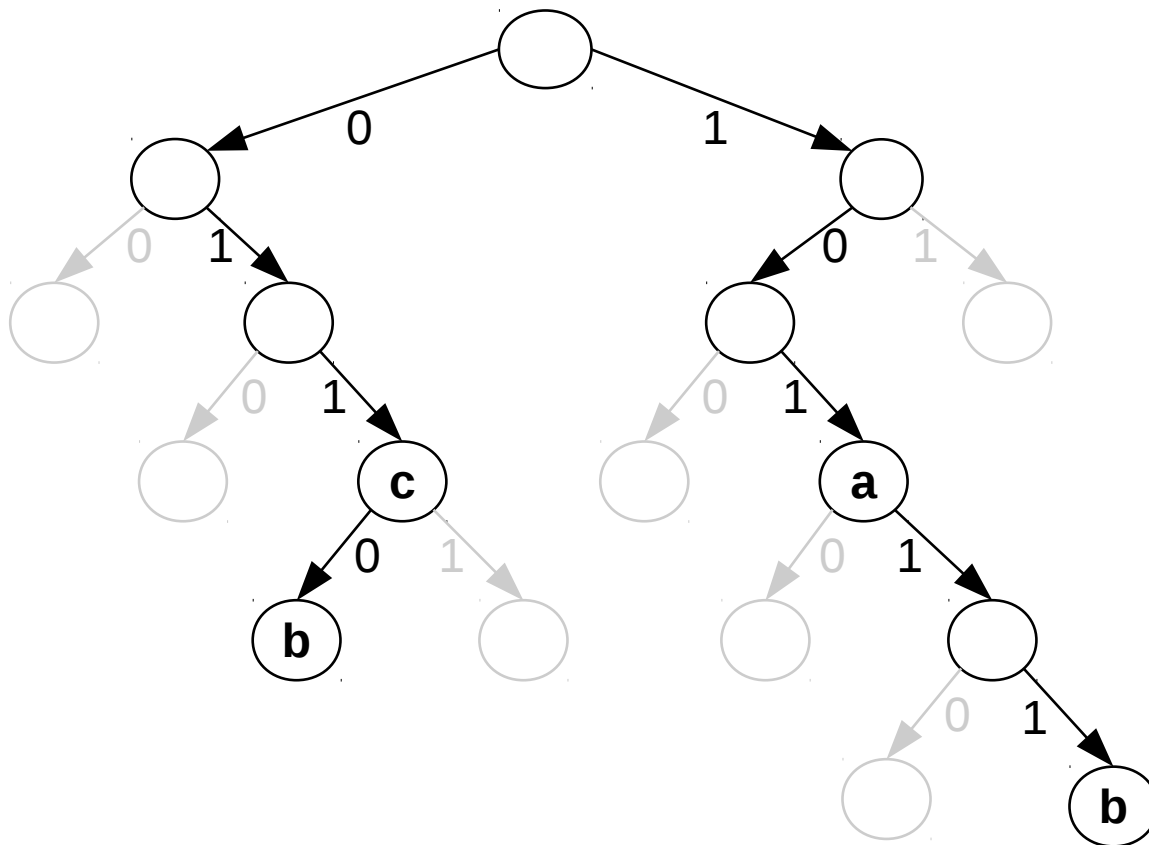
A prefix fa

- **Prefix=pontba vezető út élcímkeinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük



A prefix fa

- Az üres levélpontokat elhagyhatjuk, az üres pontokba mutató pointererek: NULL
- Kisebb fa, kevesebb memória



FIB

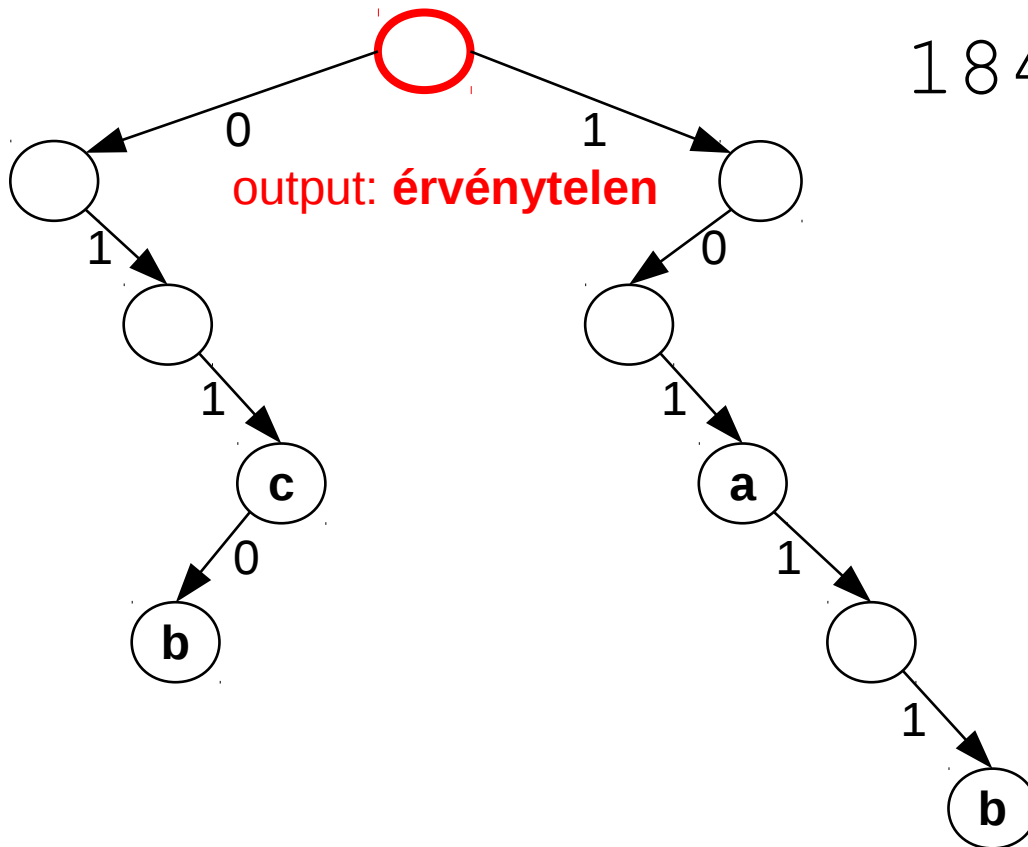
IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa: keresés

- Keressük a $184.1.1.1=10111\dots$ IP címre a legspecifikusabb találatot a prefix fában
- Start a **gyökérpontból**, **output←érvénytelen**

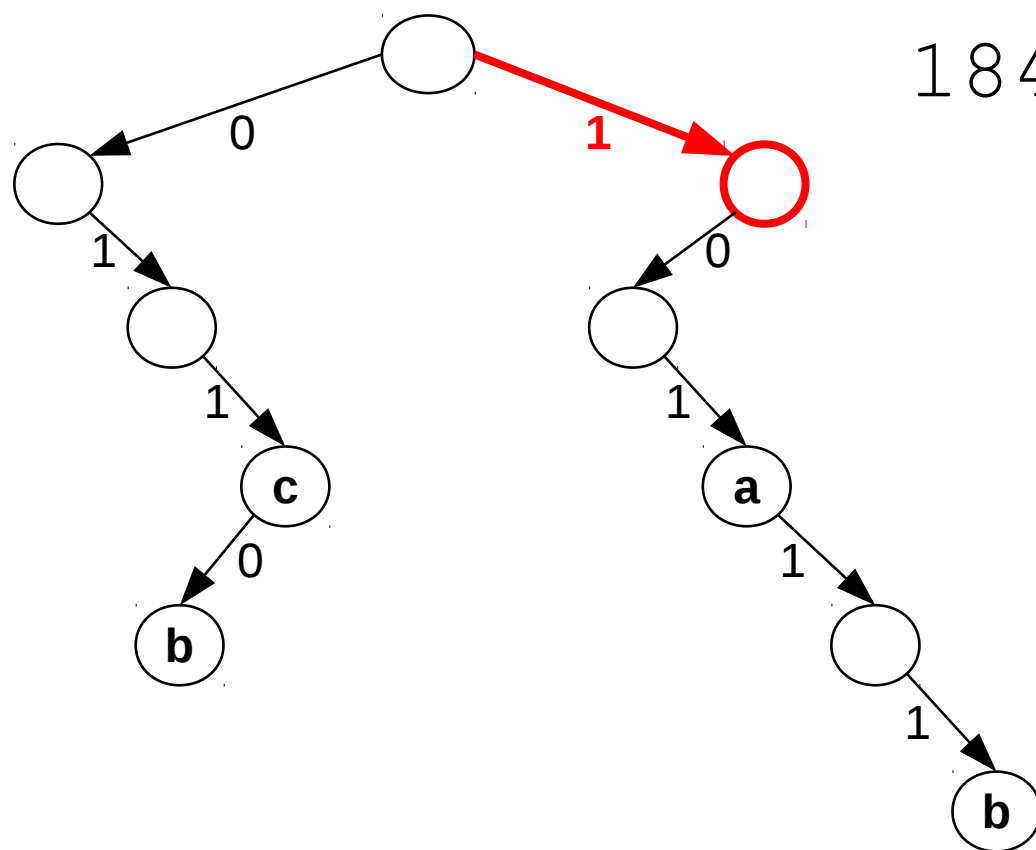
$$184.1.1 = 10111...$$


FIB

IP prefix	Prefix	Címke
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

A prefix fa: keresés

- A $184.1.1.1=10111\dots$ keresési cím első bitje 1 értékű, így a gyökérből az 1-es élcímkevel ellátott élen lépünk a **következő pontba**



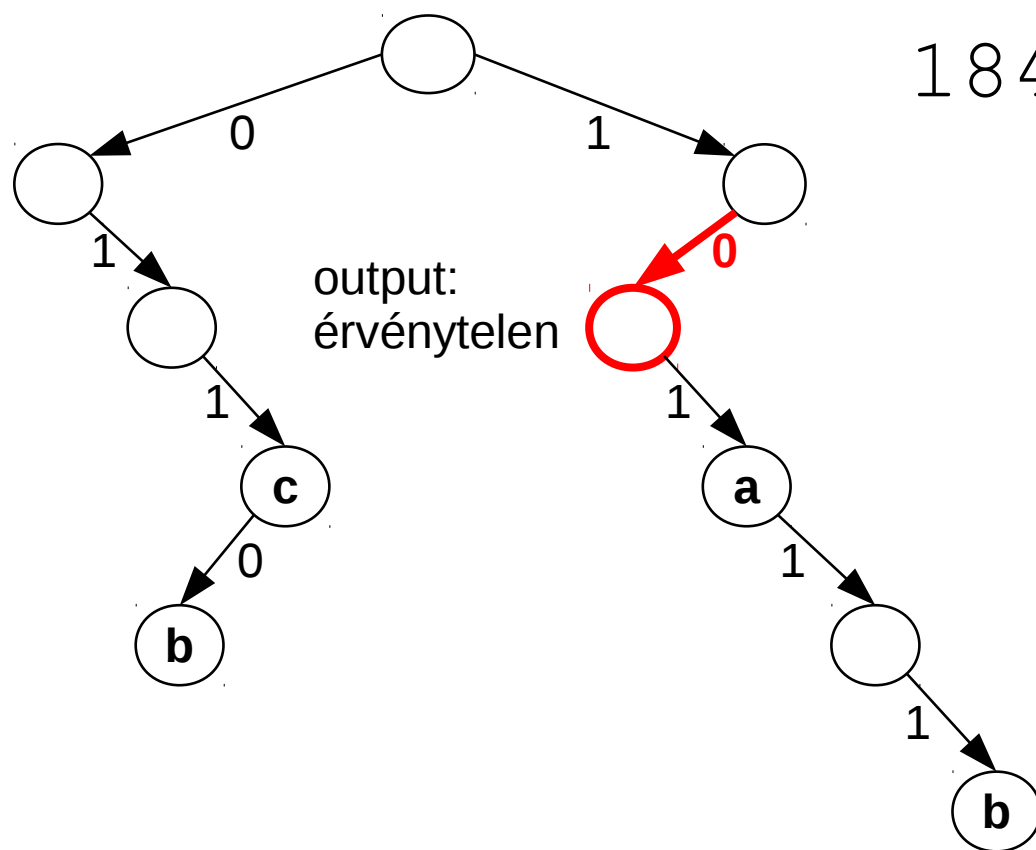
$184.1.1.1=10111\dots$

FIB

IP prefix	Prefix	Címke
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

A prefix fa: keresés

- Az új pontban nincs címke, output változatlan
- A második bit 0, így az 0 élcímken haladunk tovább a **következő** pontba



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

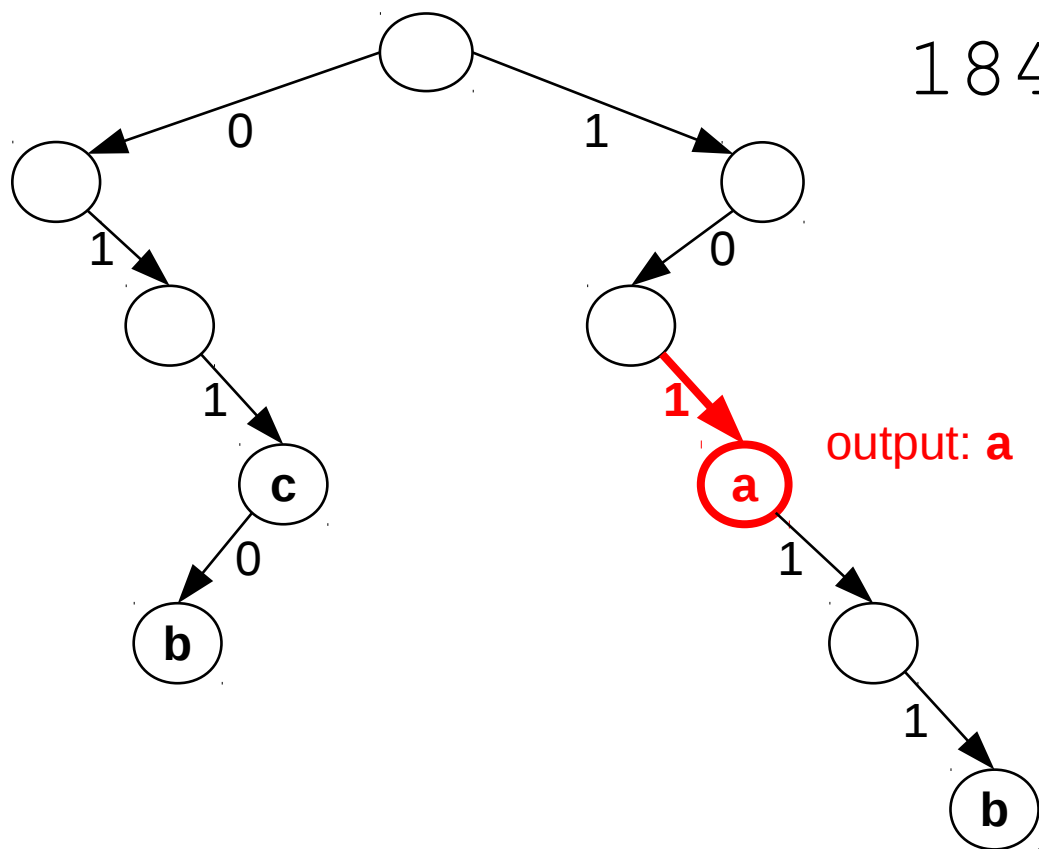
FIB

IP prefix	Prefix	Címke
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

A prefix fa: keresés

- A harmadik bit ismét 1, az 1 élcímkén haladunk tovább a **következő pontba**
- Az új pont címkéje **a**, ezért: **output** $\leftarrow$ **a**

184.1.1.1=10**1**11...



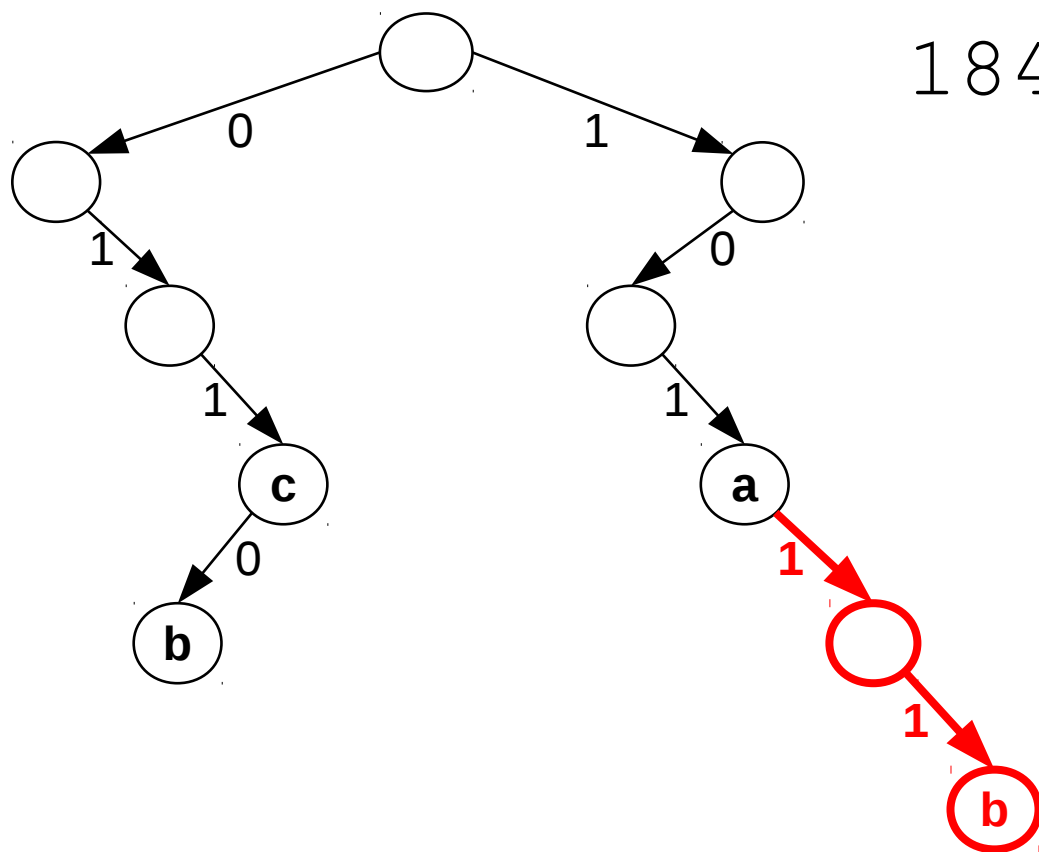
FIB

IP prefix	Prefix	Címke
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

A prefix fa: keresés

- A 4. és 5. bit 11, az 1 élcímkéken egy **b**, címkével ellátott pontba jutunk, ezért: **output** $\leftarrow$ **b**
- A kapott pont levél, így **kilépés**: **output** = **b**

184.1.1.1=101**11**...



FIB

IP prefix	Prefix	Címke
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

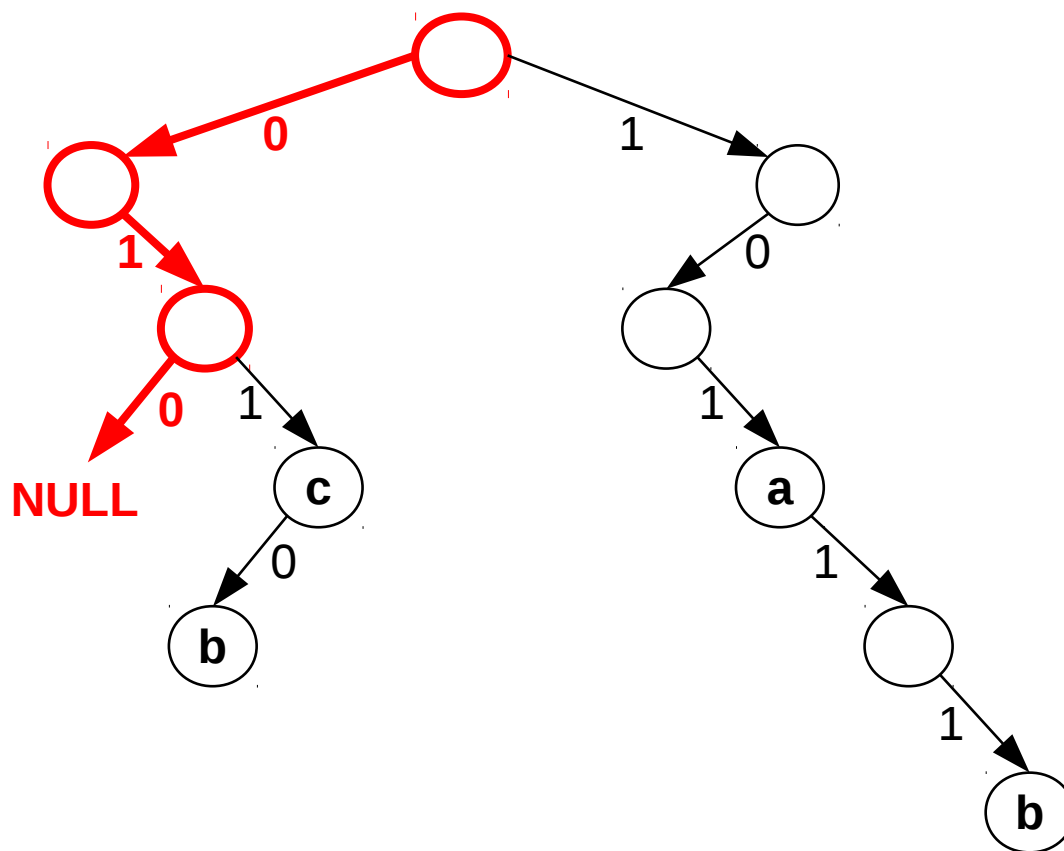
output: b

A prefix fa: keresés

- **Algoritmus:** sorban olvassuk a keresett IP cím bitjeit, és az olvasott bitnek megfelelően mindig a 0 vagy 1 élcímkével ellátott élen lépünk tovább egy következő pontba
- Az iteráció közben az output változóban tároljuk a legutoljára olvasott címkét (kezdetben: érvénytelen)
- Ha levélpontot vagy NULL pointert találunk: kilépés
- A kapott címkéhez (output) kiolvassuk a next-hop indexből a megfelelő next-hop címet és visszaadjuk
- Esetünkben az LPM eredmény: $b \rightarrow 10.0.0.2$

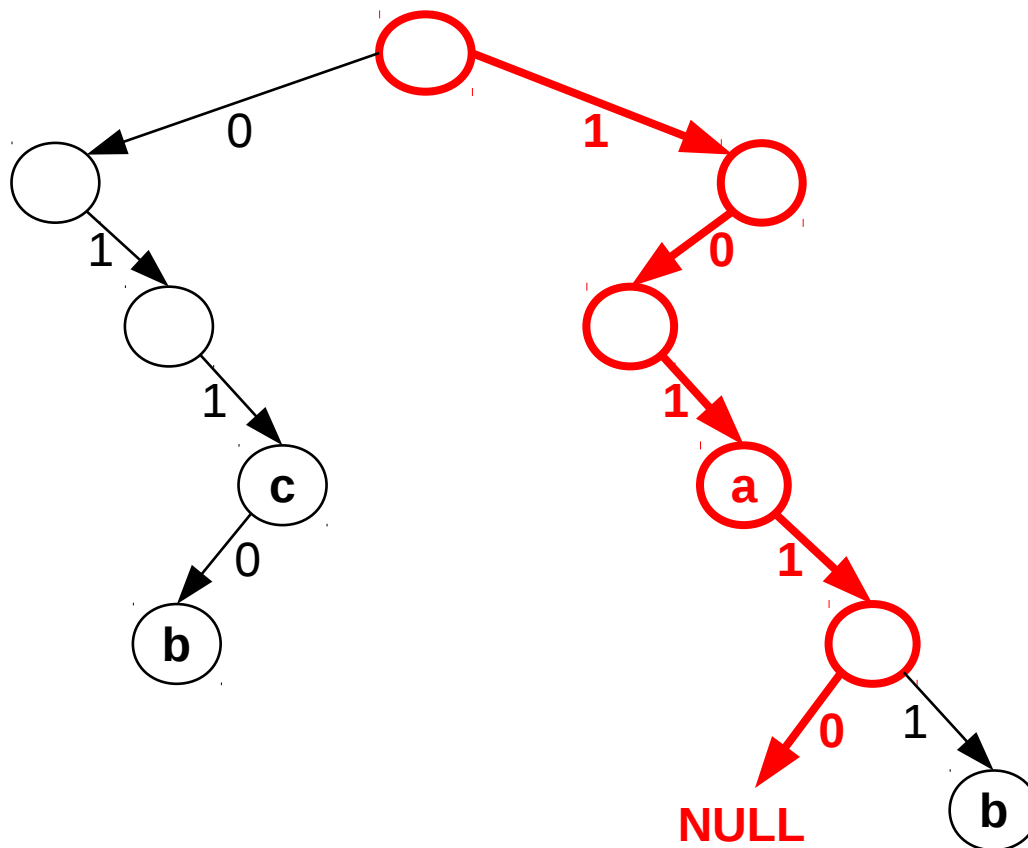
A prefix fa: keresés

- LPM a $69.12.75.54=01000\dots$ IP címre
- Nem találkozunk érvényes címkéjű ponttal:
output = érvénytelen



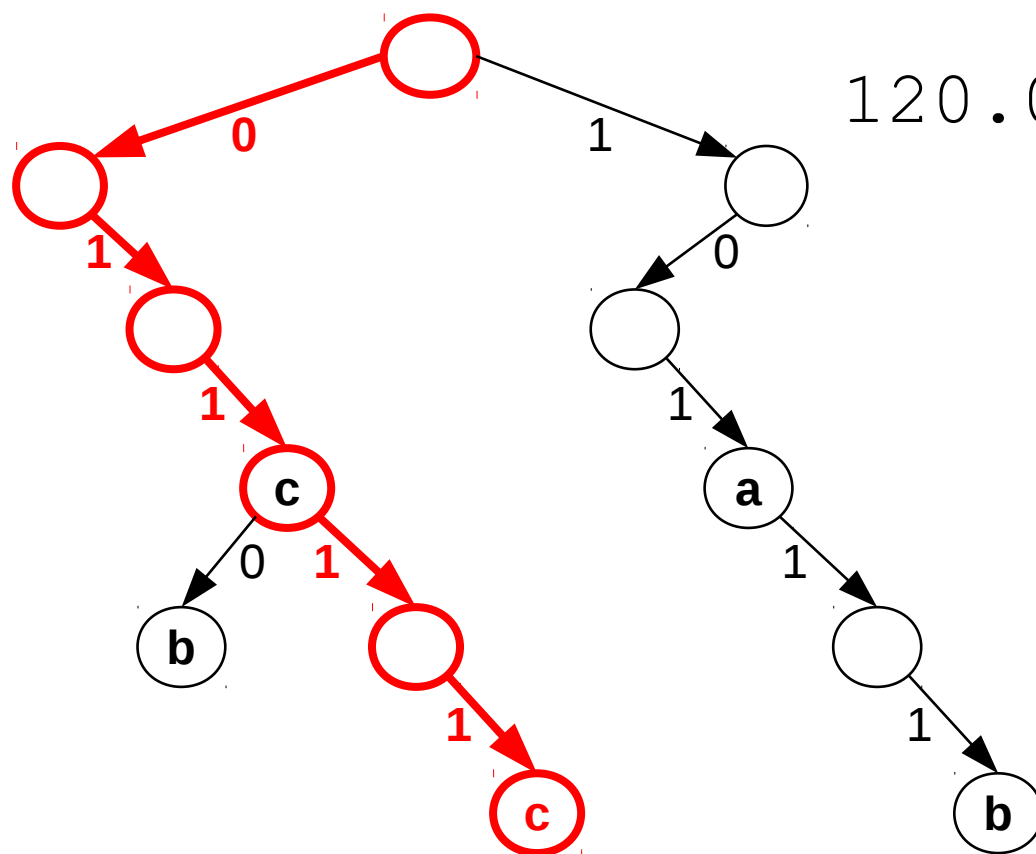
A prefix fa: keresés

- LPM a 178.4.66.19=10110... IP címre
- Utolsó látott címke: **a**, **output = a**



A prefix fa: beillesztés

- Illesszük be a $120.0.0.0/5 \rightarrow 10.0.0.3$ bejegyzést
- Kövessük a bitek által kijelölt utat, hozzuk létre a hiányzó pontokat, és a kapott pontba írjuk **c**-t



$120.0.0.0/5 = 01111\dots$

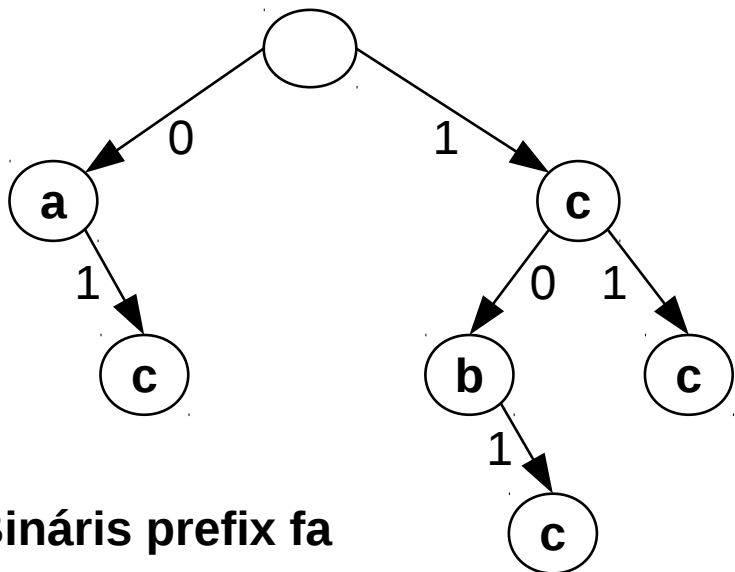
A prefix fa: egyéb műveletek

- A **módosítás** is hasonló: kövessük a bitek által kijelölt utat, írjuk felül a kapott pontban a címkét
- A **törlés** is hasonló, de kitöröljük a címkét a kapott pontból és felfelé haladva a fán rekurzívan kitöröljük az üres levélpontokat
- Legfeljebb annyi lépést kell tennünk a fában, ahány bitből egy IP cím állhat
- **Tétel:** a prefix fán a LPM keresés, beillesztés, törlés és módosítás, legfeljebb $O(W)$ lépésben végez, ahol W a címtér bitszélessége (IPv4: 32, IPv6: 128)

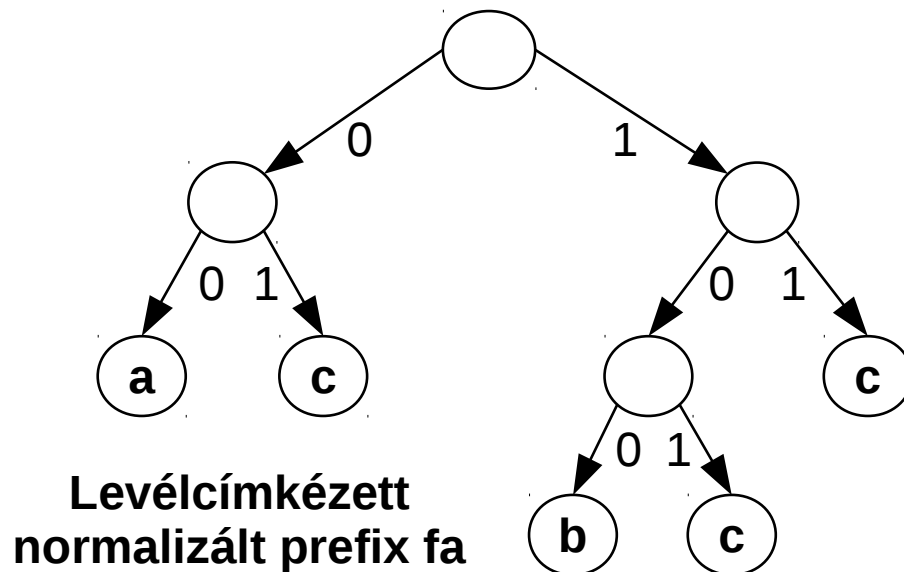
A prefix fa

- Általánosságban: N prefix tárolása esetén a LPM keresés, beillesztés, törlés és módosítás műveletek komplexitása $O(\log N)$
- A táblázatos tárolási módszerrel az összes prefixen végig kellene keresni: $O(N)$ lépés
- De 32 bit RAM olvasás (főleg ha nem cache-ből történik) időigényes: lassú LPM
- **FIB aggregáció:** a prefix fa konverziója kisebb, de az eredetivel keresés szempontjából teljesen ekvivalens formába
- Feltétel: LPM csak teljesen specifikált IP címekre (32 bit), tetszőleges prefixekre nem feltétlenül működik!

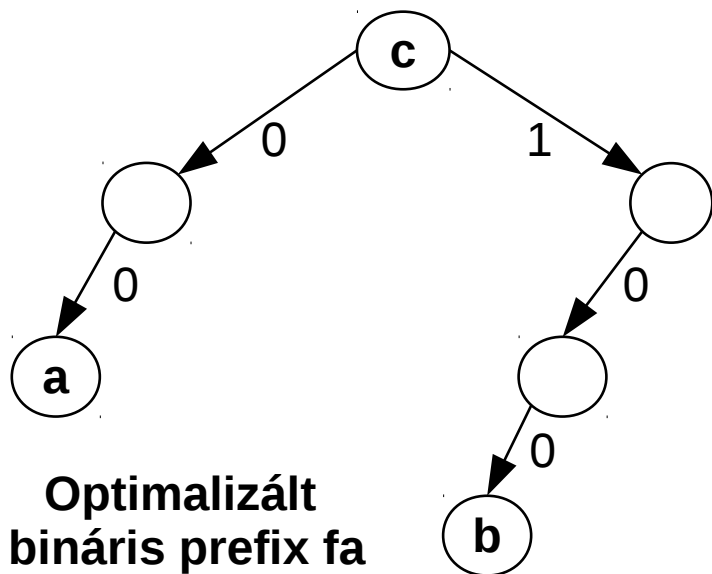
FIB aggregáció



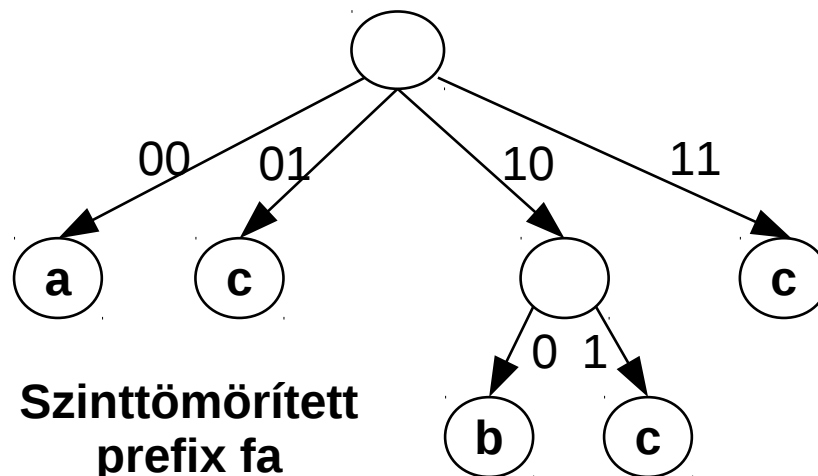
Bináris prefix fa



**Levélcímkézett
normalizált prefix fa**



**Optimalizált
bináris prefix fa**



**Szinttömörített
prefix fa**