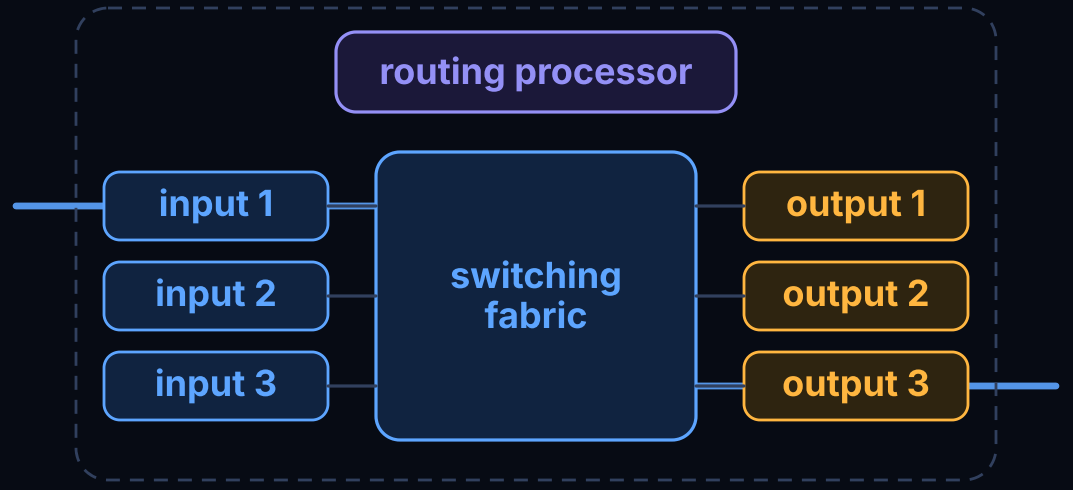
Inside the router Ports, fabric and queues

Two lectures treated the router as a box that reads an address and picks a link. This one opens the box: the hardware a packet passes through, how the lookup is done in nanoseconds, the piece that moves a packet between ports, and the memory where it waits.

Yaxiong Xie

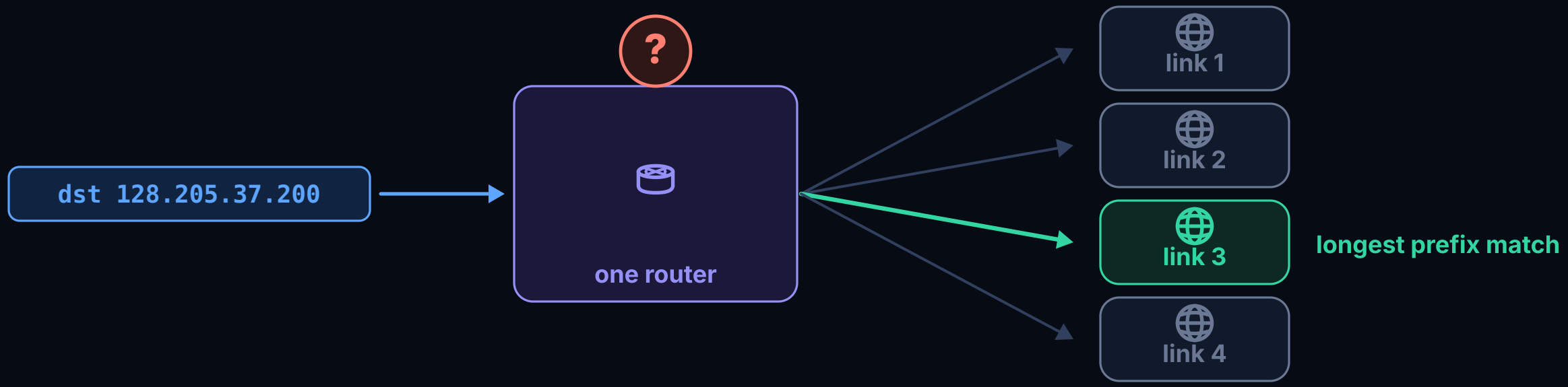
6.7 ns

between two packets on one 100 Gbps port



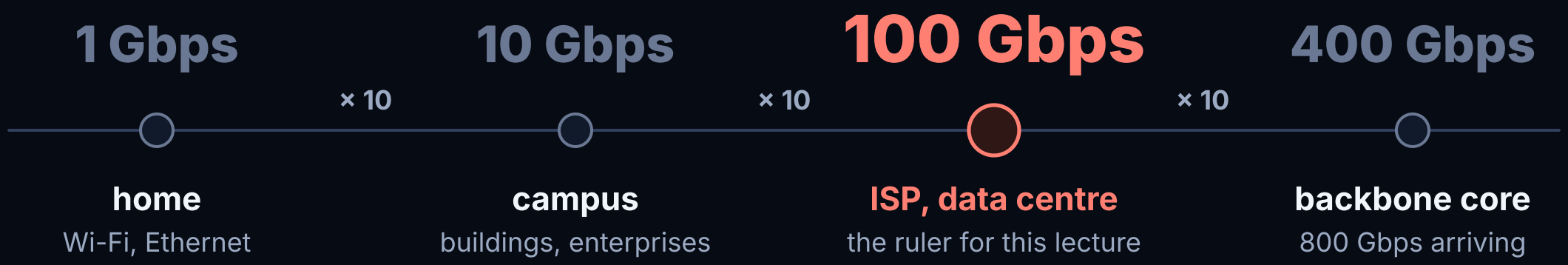
- three parts a packet crosses, one it never sees
- two places inside where a packet waits
- and the one row of slots where it is dropped

Two lectures treated the router as one box



Read one field, pick one link. Nothing yet about the hardware that does it.

How fast are the links a router sits on?



one router port is one of these links; a core router has tens of them

A core router port carries 100 to 400 Gbps. One 100 Gbps port is our ruler.

How many packets is 100 Gbps?

EVERY PACKET 1,500 BYTES



12,304 bits
1,538 bytes on the wire
`100 Gbit/s ÷ 12,304`

→ **8.1 million**
packets a second

× 18

EVERY PACKET 64 BYTES



672 bits
84 bytes on the wire
`100 Gbit/s ÷ 672`

→ **148.8 million**
packets a second

The smallest Ethernet frame is 64 bytes; the preamble and the gap between frames add 20 bytes on the wire.

Same link, same bits per second. Eighteen times more packets when they are small.

Which case must the router be built for?

the easy case

every packet 1,500 bytes
8.1 million a second
123 ns for each one: time to spare

the worst case

every packet 64 bytes, the link full
148.8 million a second
6.7 ns for each one

the router does not choose the packet size TCP ACKs · DNS · VoIP · game traffic

a burst of the smallest packets at line rate is legal; a router that falls behind drops packets

Built for the worst case: the smallest packets, back to back, at line rate.

So how much time does one packet get?

6.7 ns

1 s ÷ 148.8 million = 6.72 ns between two packets on one 100 Gbps port



Under seven nanoseconds per packet, per port. Everything inside runs at that pace.

Four topics, in the order a packet meets them

01



Four parts

two jobs,
two speeds

02



The lookup

a trie, or every
row at once

03



The fabric

from one port
to another

04



Queues, scheduling

who waits,
who goes first

Each from its own problem: why it exists, how it works, what it costs.

What is inside the box?

01

Four parts

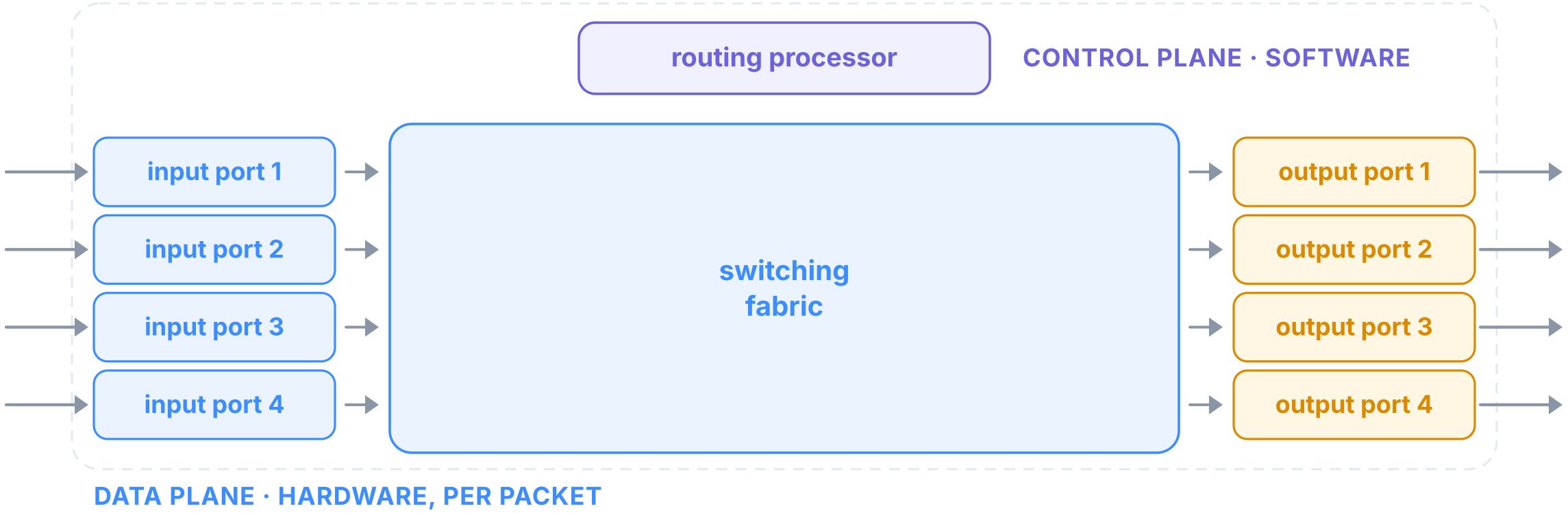
Two jobs run inside a router, at speeds a million apart. They cannot be done by the same machine.

What has to happen to every packet?



Receive, decide, send. For every packet, in under seven nanoseconds.

Inside the box: four parts



Three parts a packet passes through, in hardware. One that writes the table, in software.

Why this split?

per packet

every 6.7 ns: 148.8 million a second, per port
the same fixed steps: receive, look up, send
so: hardware, in every port and the fabric

$\times 10^6$

per route change

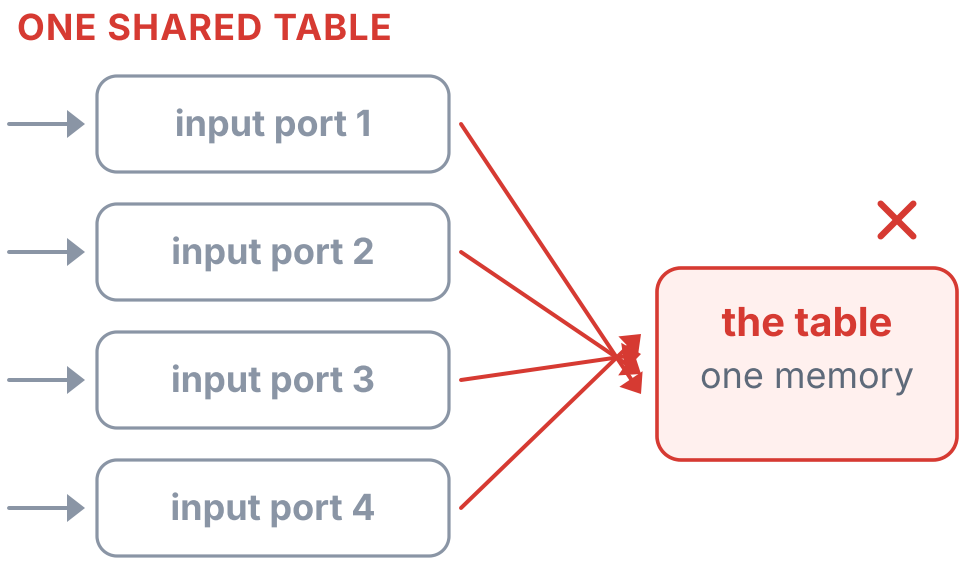
every few milliseconds to seconds
a protocol: messages to neighbours, state, policy
so: a program, on one processor

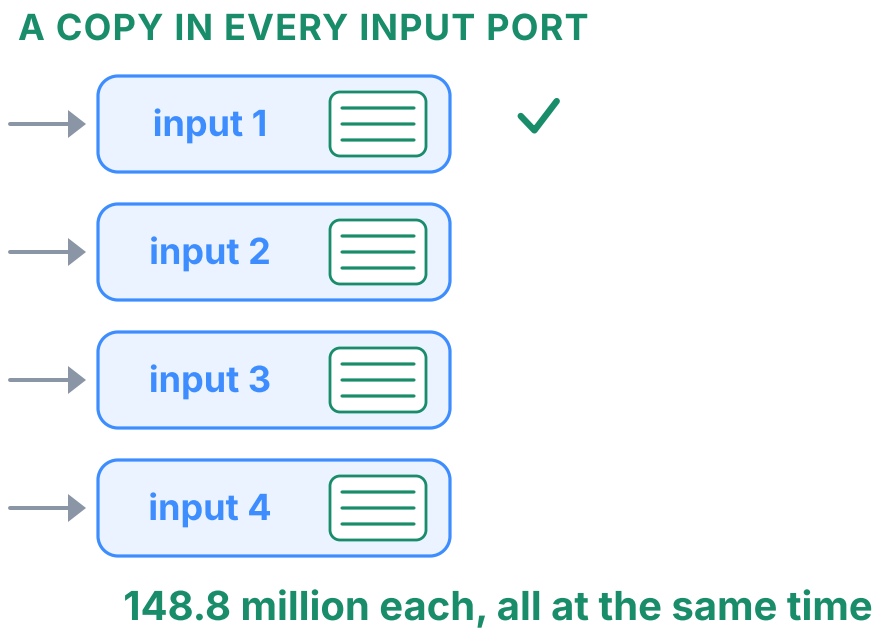
a core forwarding in software manages about a million packets a second

one port delivers 149 million, sixteen ports 2.4 billion: the per-packet job cannot be a program

Two jobs, a million apart in time. Hardware for the packets, software for the routes.

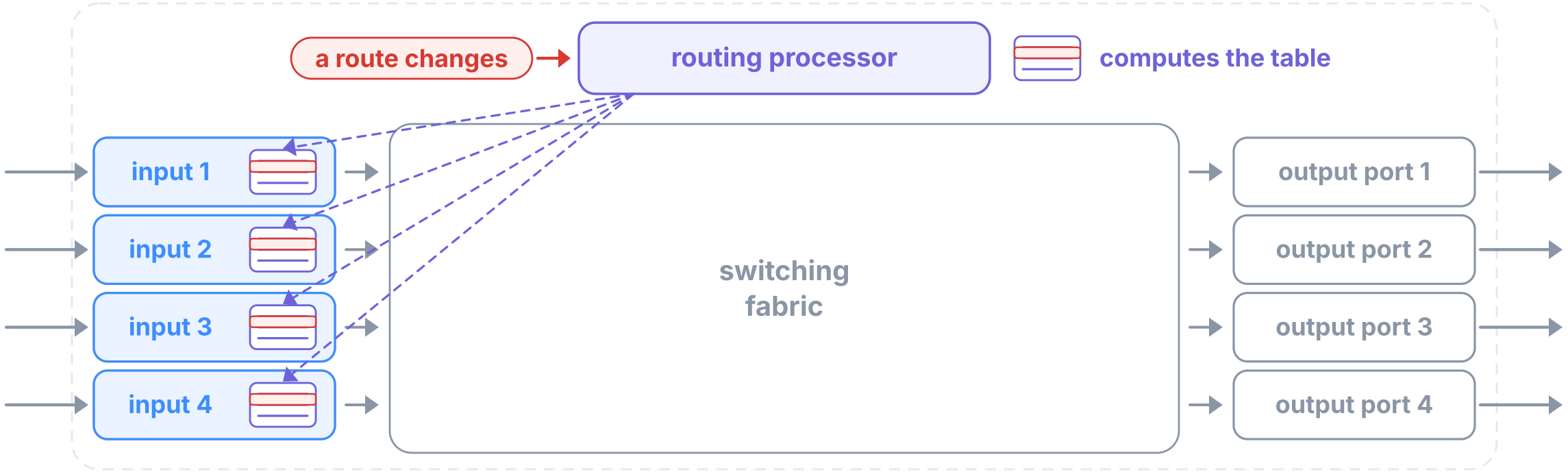
Why a copy in every port, and not one table for all?





Sixteen lookups at once, each in its own port. Only the fabric stays shared.

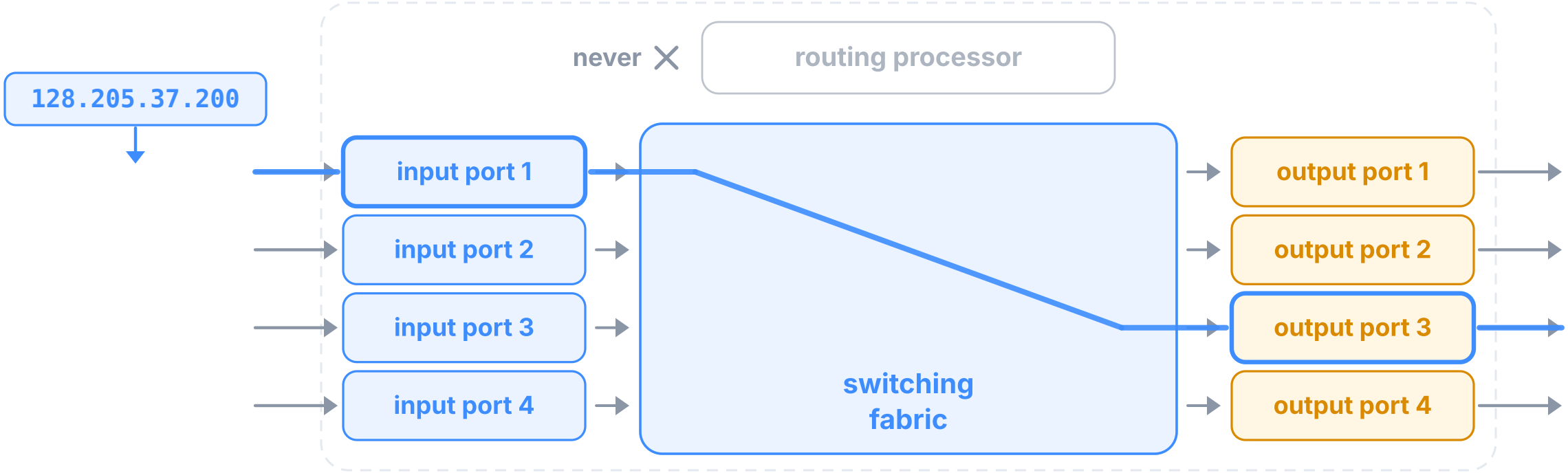
How does the table reach the hardware?



a route changes: one row rewritten in every copy; the ports keep forwarding

Computed once in software, then copied into every input port for hardware to read.

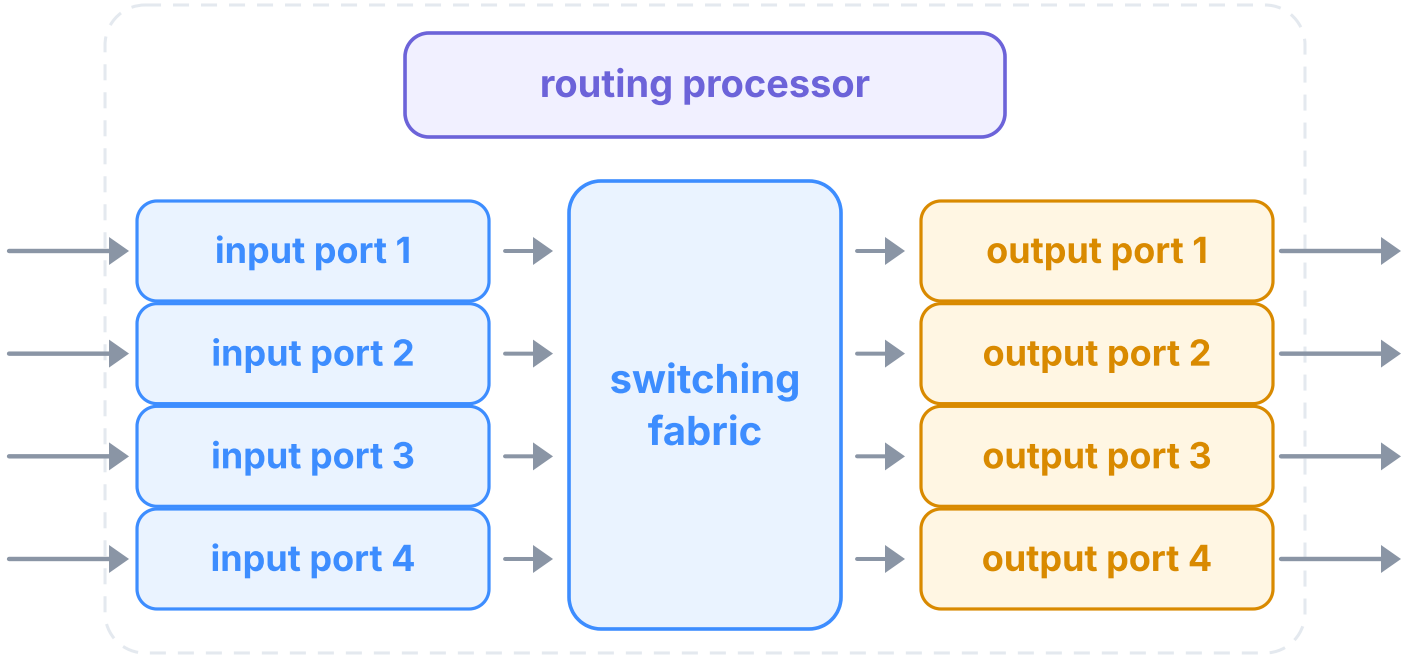
Which parts does a packet touch?



the routing processor never sees the packet

Three. The routing processor never sees a packet.

What does the split demand?



02 · the input ports

the longest matching prefix,
alone, in a few nanoseconds

03 · the fabric

sixteen packets from port to port,
all at the same time

04 · the output ports

two packets, one wire: one waits.
Where, how long, which goes first

What it buys: a packet never waits for software, and a route can change without stopping a packet.

Each hardware part must keep up on its own. The rest of the lecture is how.

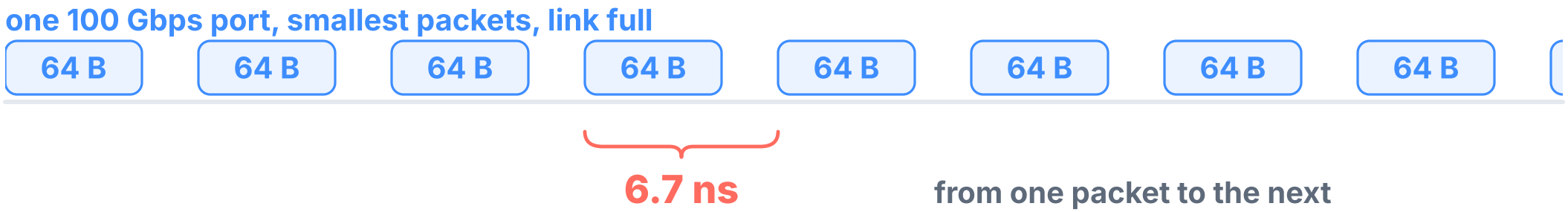
How does a port find the longest matching prefix?

02

The lookup

Five rules, four bits. Scan them one by one, walk a tree one bit at a time, or compare every row at once.

How much time does one lookup get?





one lookup must finish in 6.7 ns, before the next packet arrives

100 Gbps, 64-byte packets (84 bytes on the wire with preamble and gap): 148.8 million packets a second.

We get 6.7 ns per packet: the lookup must finish before the next one arrives.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

Real addresses have 32 bits; our pictures use 4

a real IPv4 address: 32 bits

128.205.37.200

1 0 0 0 0 0 0 0 . 1 1 0 0 1 1 0 1 . 0 0 1 0 0 1 0 1 . 1 1 0 0 1 0 0 0

128 205 37 200

for the pictures

a 4-bit address **1 0 1 1**

16 addresses: a tree that fits on a slide

everything that follows works the same at 32 bits

We use 4-bit addresses in the pictures; everything carries over to 32 bits.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

What must the lookup find in that time?

destination address



prefix		length	port
*	✓	0	A
0*		1	B
10*	✓	2	C
101*		3	D
11*		2	E



three rules match

the longest wins: 3 bits

We find every prefix that matches, then take the longest: longest prefix match.

The simplest way: compare with every row

destination address

1	0	1	1

prefix	port
*	A
0*	B
10*	C
101*	D
11*	E

compare

...

best so far

D

prefix length 3

1011 → port D

We read a row, compare, keep the longest match so far, and go on to the end.

How big is the real table?

prefix	port
*	A
0*	B
10*	C
101*	D
11*	E
...	
...	
...	

1,078,000 rows

a core router today

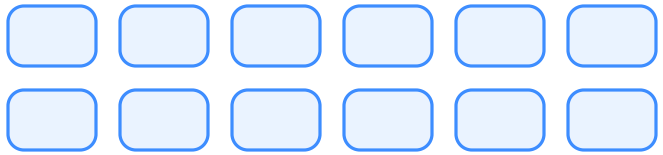
1,078,000 IPv4 prefixes

one scan

1 ns per comparison →
about 1 ms per lookup

meanwhile

about 160,000 packets
arrive, one every 6.7 ns



... and 159,988 more

1,078,000 prefixes × 1 ns = 1.08 ms for one lookup; 1.08 ms / 6.72 ns ≈ 160,000 packets.

We cannot scan a million rows in 6.7 ns: one scan takes 160,000 packet times.

The table is too big. What can we do?

Trade space for time

a pattern many programs use

the index of a book

extra pages at the back;
find a word without
reading the book

a hash table

extra, mostly empty slots;
one step to the key
instead of N

a cache

store results before they
are asked for; read them
instead of recomputing

organize the data so a lookup touches a few entries

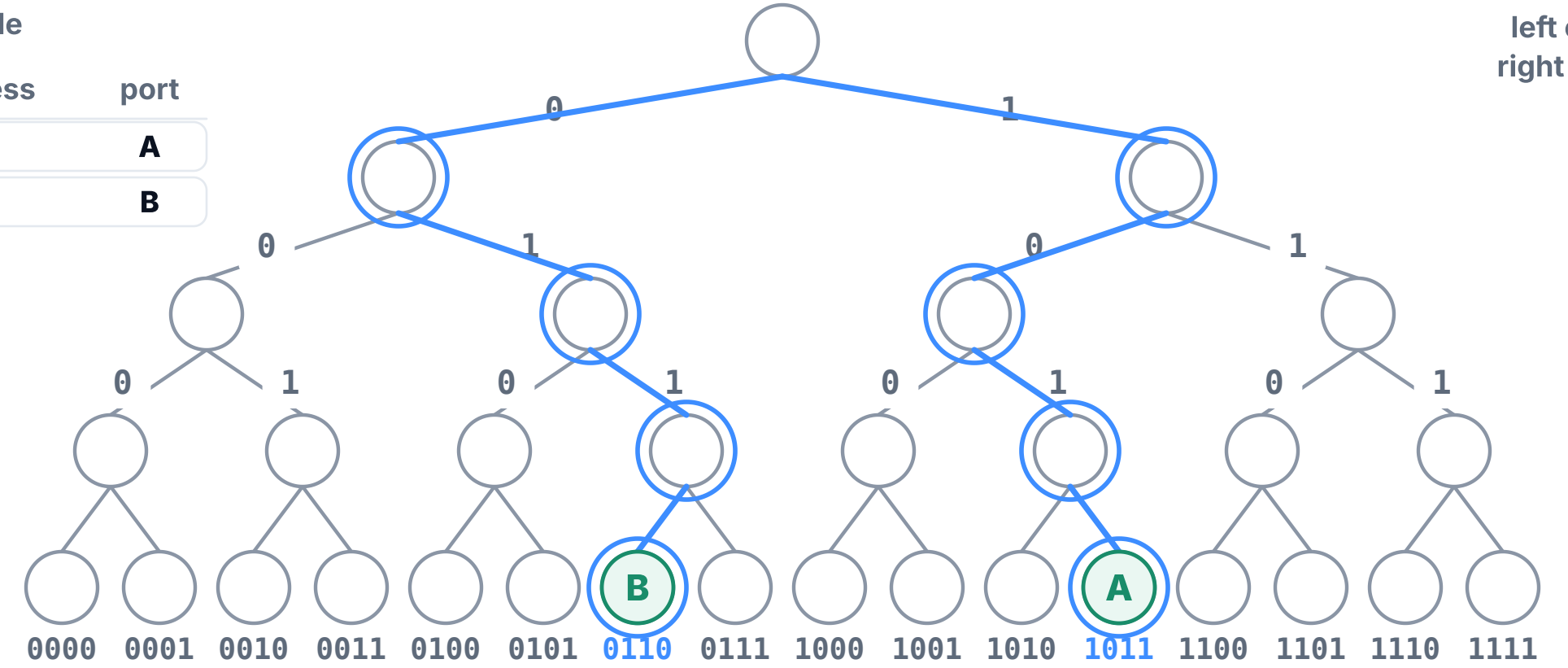
We spend memory on organizing the table so a lookup reads a few rows, not all.

Every 4-bit address is one path in the complete tree

the table

address	port
1011	A
0110	B

left child: 0
right child: 1

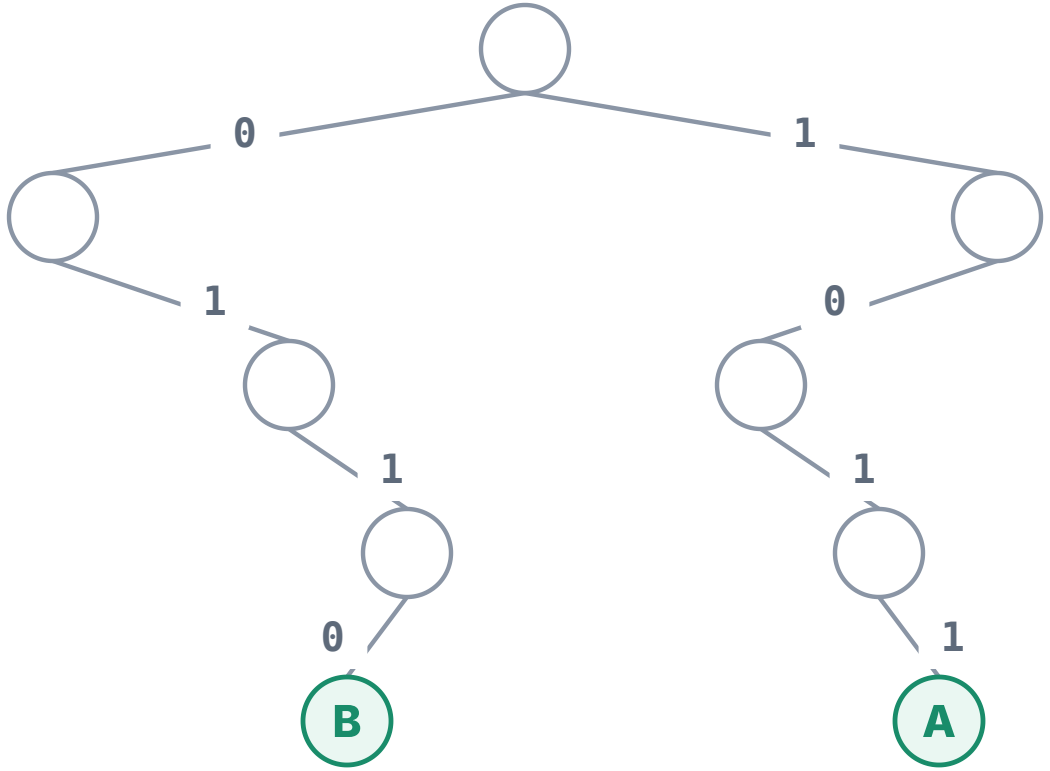


Sixteen addresses, sixteen leaves; our two records are two of the paths.

The table keeps only the paths of its records

the table

address		port
1011	✓	A
0110	✓	B



left child: 0
right child: 1

a record = a path
a bit = left (0) or right (1)

Two records leave two paths; the rest of the tree is never built.

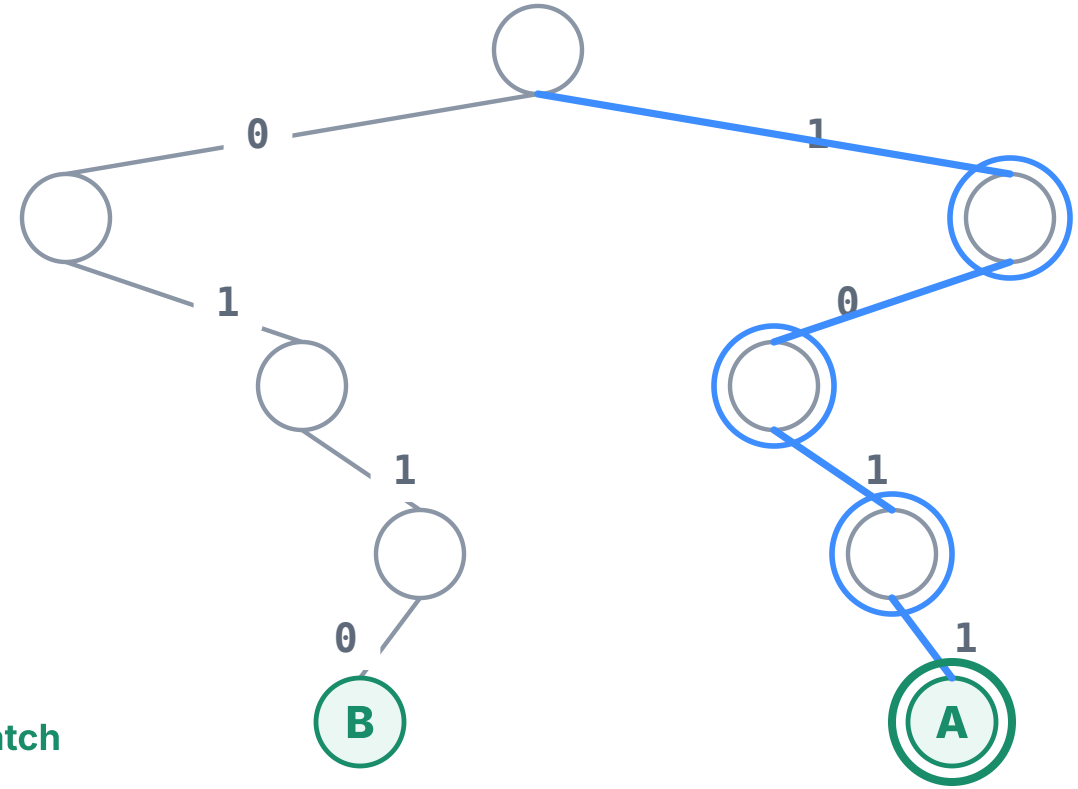
Look up 1011: walk down the tree

destination address

1	0	1	1

bits read so far

1011



1011 → port A

the walk reaches a record: a match

We follow the bits down; the walk ends on a record. 1011 → port A.

Look up 1001: where does the walk end?

destination address

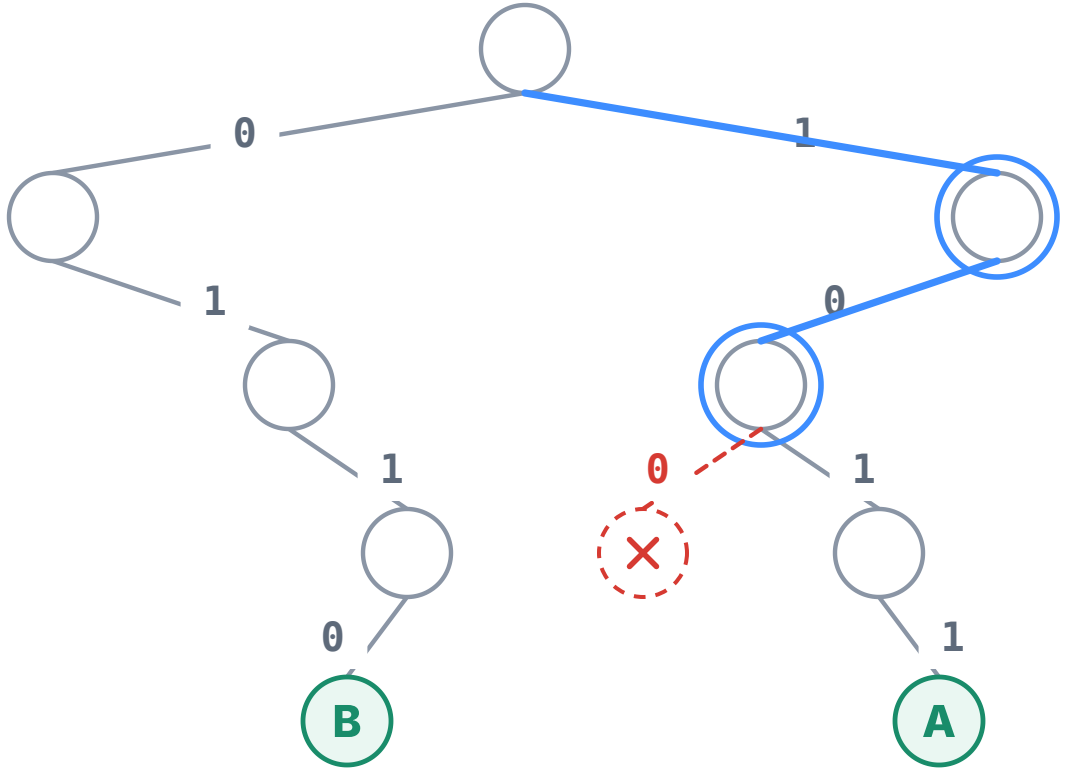
1	0	0	1

bits read so far

100

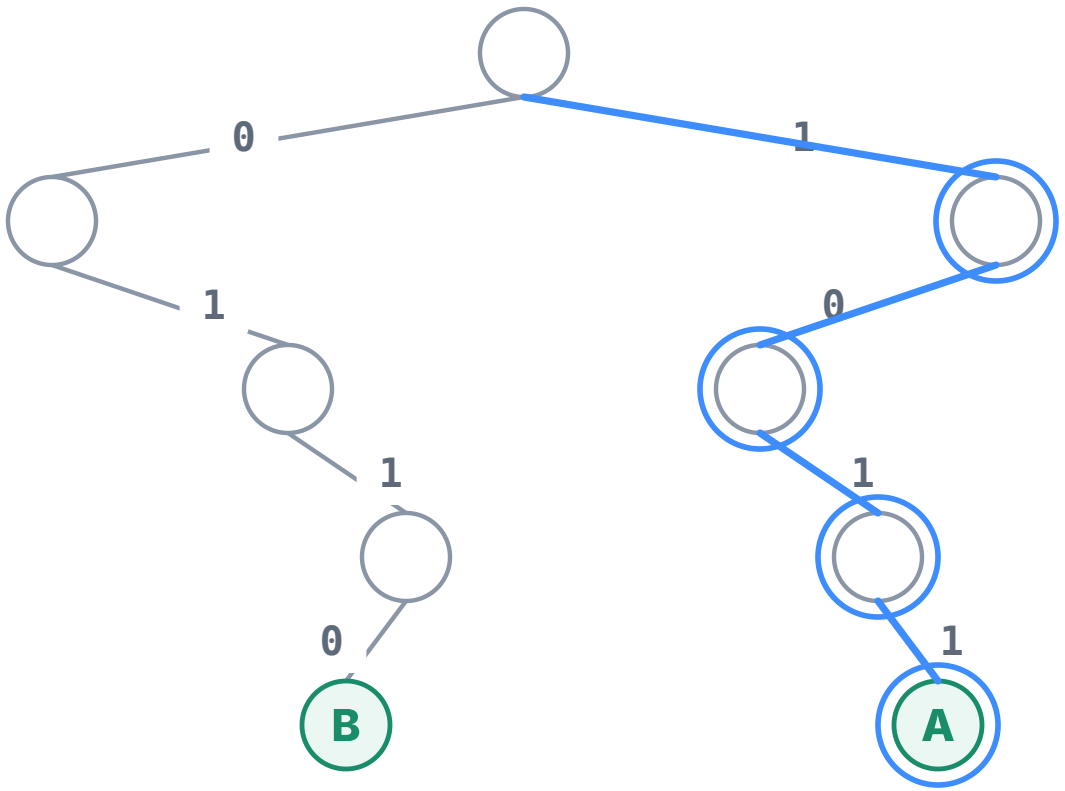
1001 → no match

no branch for the third bit



A missing branch ends the walk: no record matches 1001.

This tree has a name: the binary trie



depth 0

depth 1

depth 2

depth 3

depth 4

binary trie

two children per node

bit 0 goes left,
bit 1 goes right

a path spells the key

the key is not stored;
the path is the key

4 bits → at most 4 steps

however many records there are

Four bits: at most four steps down, however many records the table holds.

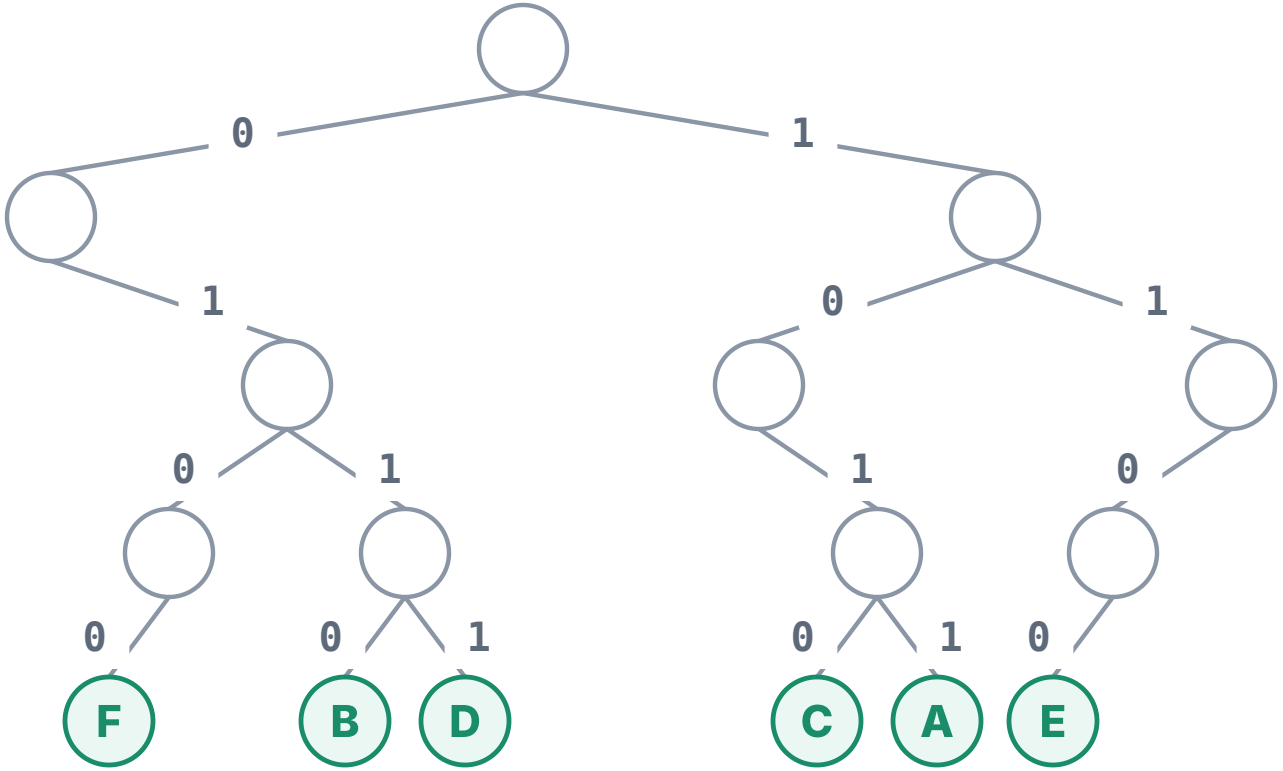
More records: what happens to the tree?

the table grows

address	port
1011	A
0110	B
1010	C
0111	D
1100	E
0100	F

records **6**

depth **4**

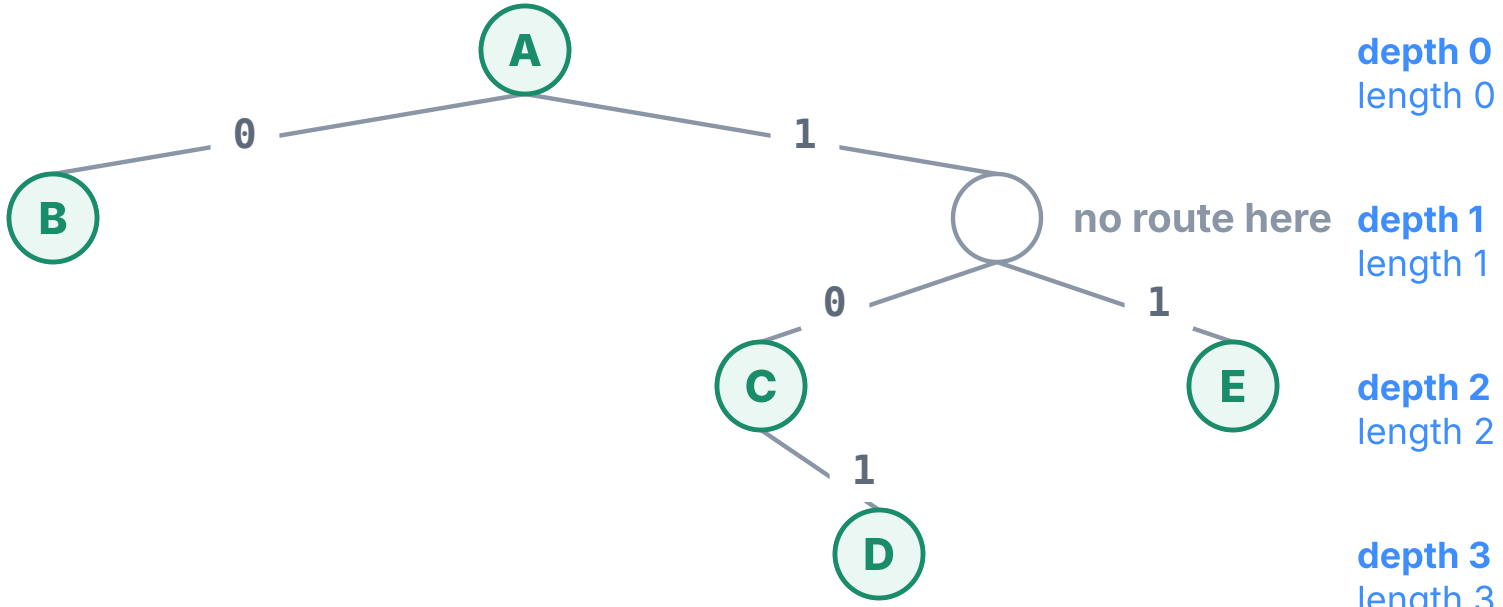


More records make the tree wider and denser; the depth stays at four.

Real tables hold prefixes of different lengths

the table: five prefixes

prefix		port
*	✓	A
0*	✓	B
10*	✓	C
101*	✓	D
11*	✓	E

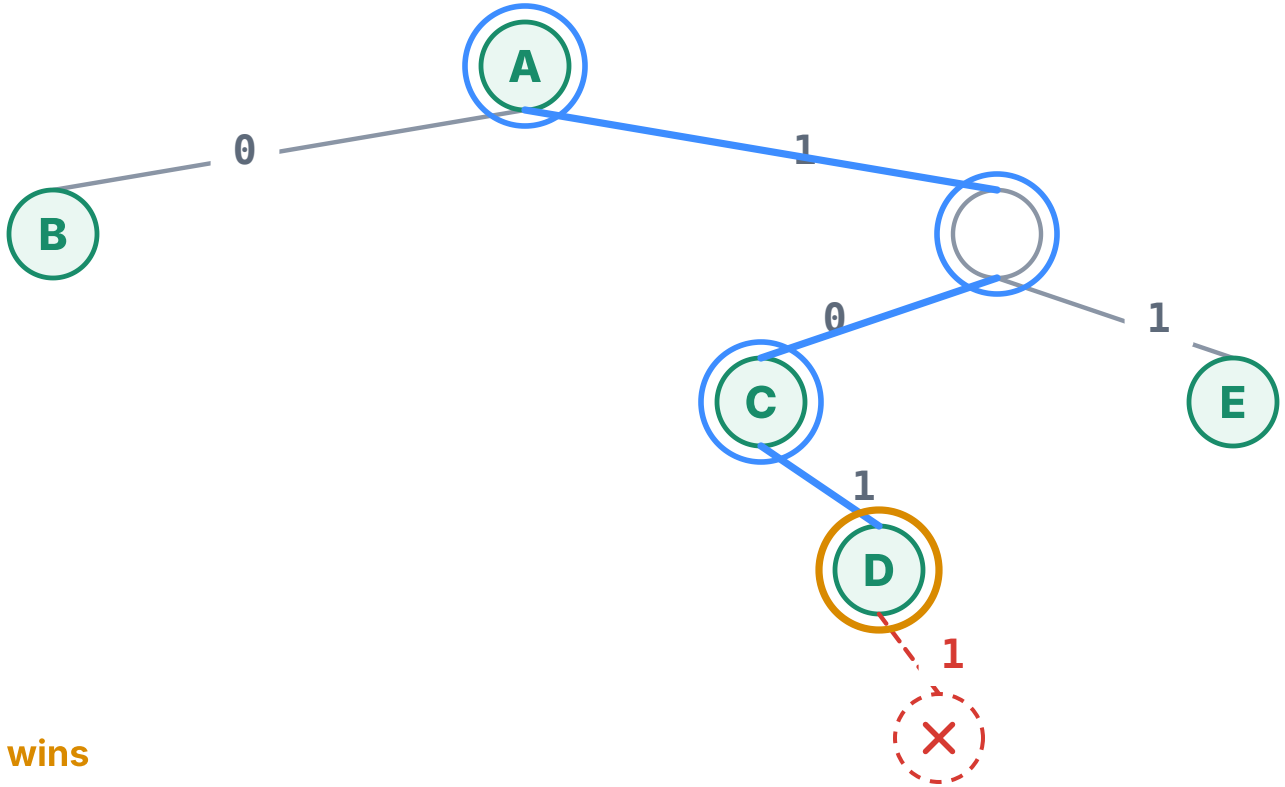


A prefix of length k becomes a route at depth k; routes sit at any level.

Look up 1011: pass a route, reach a longer one

destination address

1	0	1	1



best so far

D

prefix length 3

1011 → port D

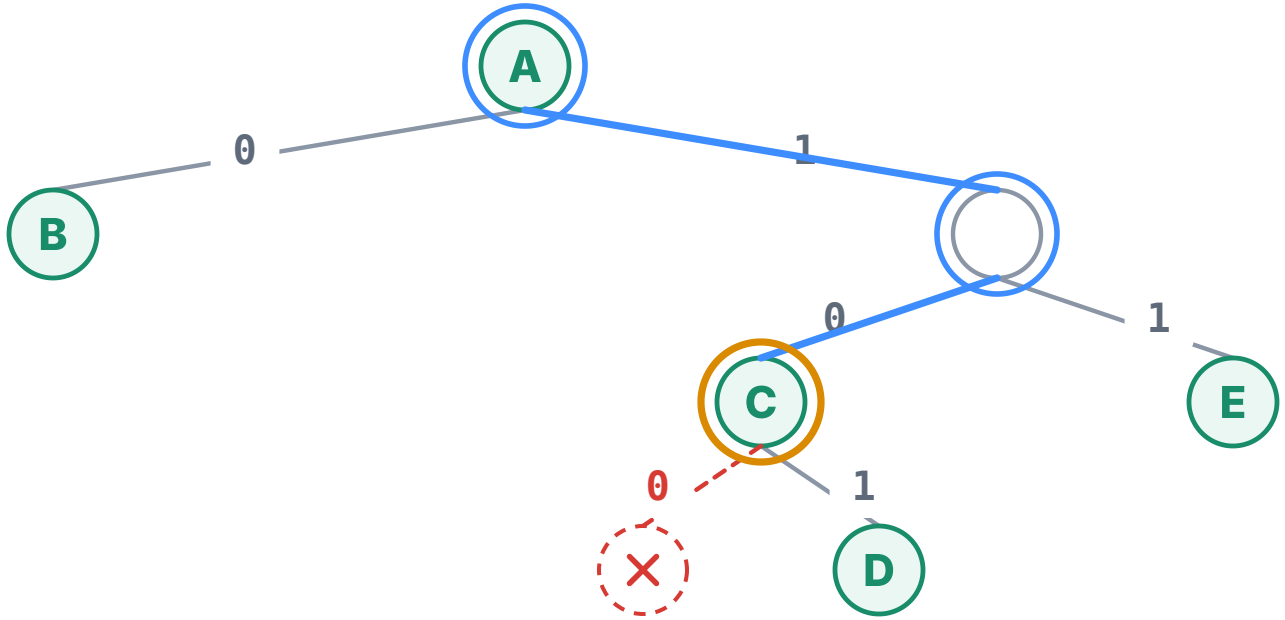
no deeper branch: the last route wins

We keep the last route we pass; the deepest one is the longest match: 1011 → D.

Look up 1001: the branch ends early

destination address

1	0	0	1



best so far

C

prefix length 2

1001 → port C

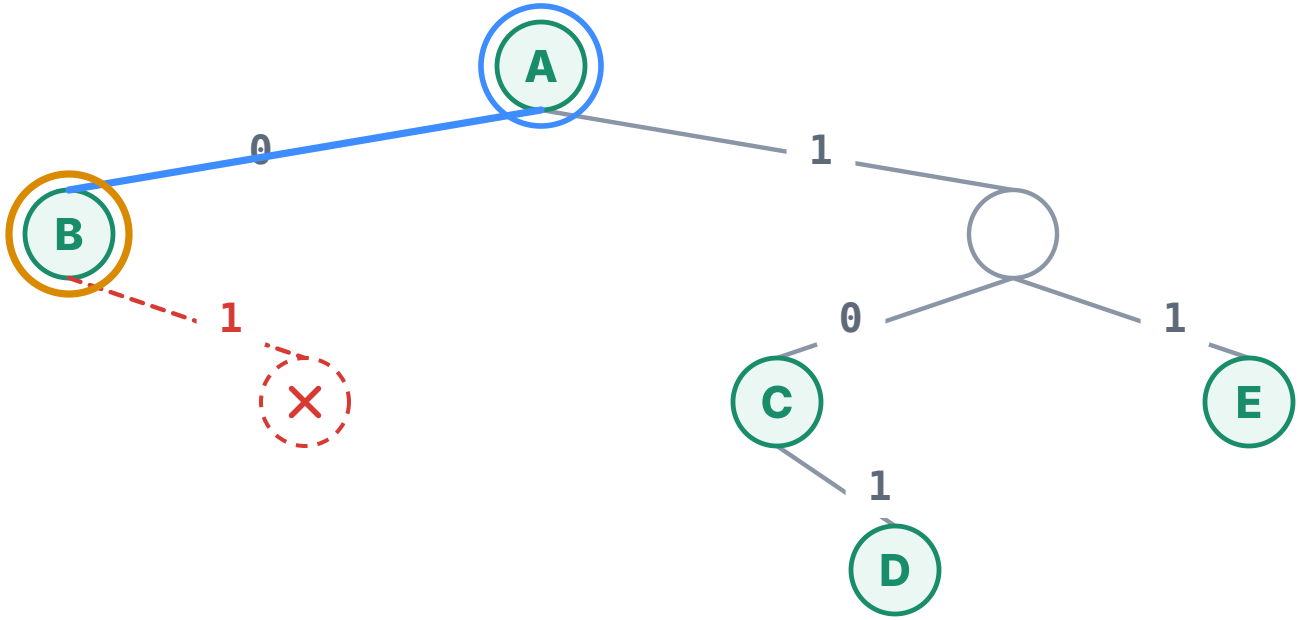
no branch for 0: the best so far stands

A missing branch returns the best so far: 1001 → port C.

Look up 0110: a short prefix answers early

destination address

0	1	1	0



best so far

B

prefix length 1

0110 → port B

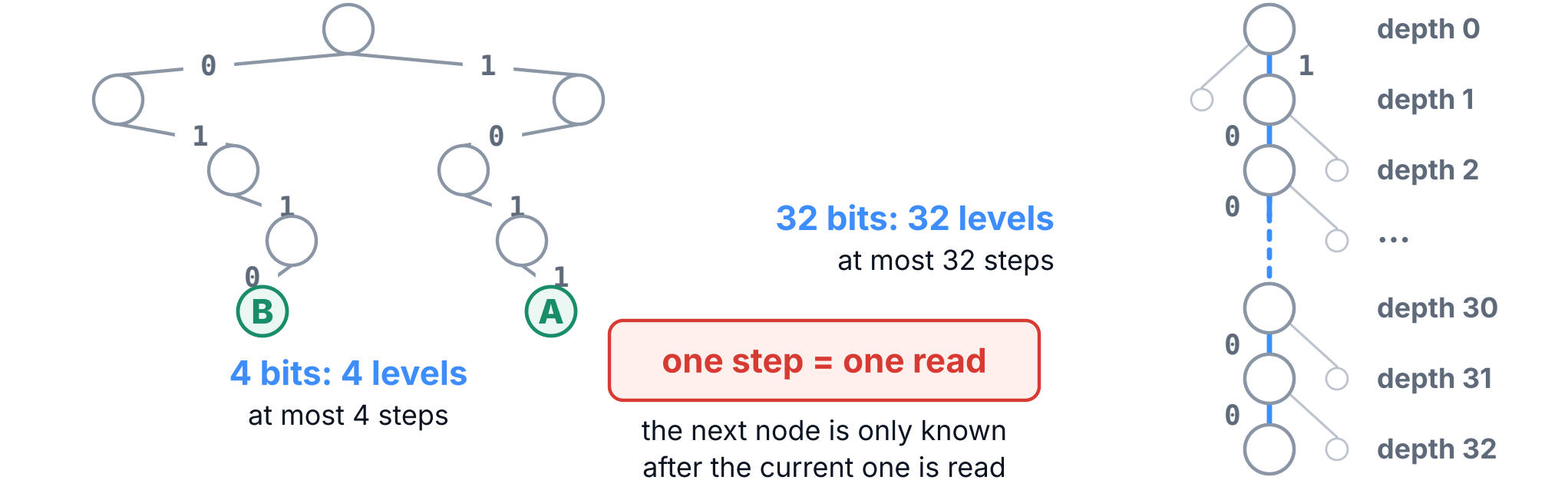
three bits never read

A short prefix can end the walk after one bit: 0110 → port B.

From 4 bits to 32 bits

the same address, all 32 bits

128.205.37.200



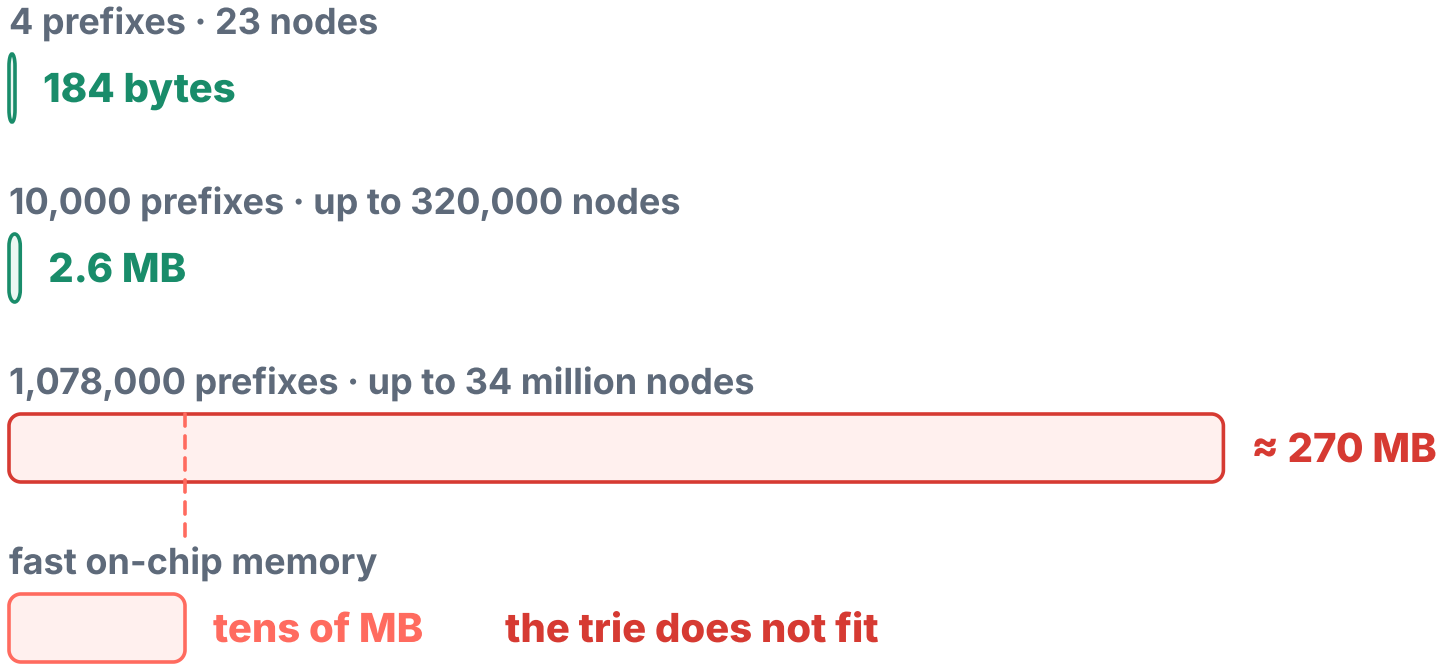
IPv4: 32 bits, a 32-level trie, up to 32 steps per lookup.

The tree grows with the table

prefixes
1,078,000

nodes
up to 34 million

up to 32 new nodes per prefix;
shared prefixes share nodes



8 bytes a node (two child pointers); worst case 32 new nodes per prefix. Shared prefixes share nodes, so real tries are smaller, but still millions of nodes.

The trie grows with the table: millions of nodes, more than fast memory holds.

32 bits

Where does this leave us?

space

a tree of millions of nodes;
it grows with the table

time

up to 32 reads, each waiting
for the one before: 32 ns
against a 6.7 ns budget

Can we make the walk shorter?

read several bits per step · compare every row at once

multi-bit tries · TCAM

Open: fewer steps per lookup, and a smaller tree. We take that up next.

Why is the trie walk slow?



the next read can only start where the previous one pointed

one lookup: 32 reads, one after another

32 ns

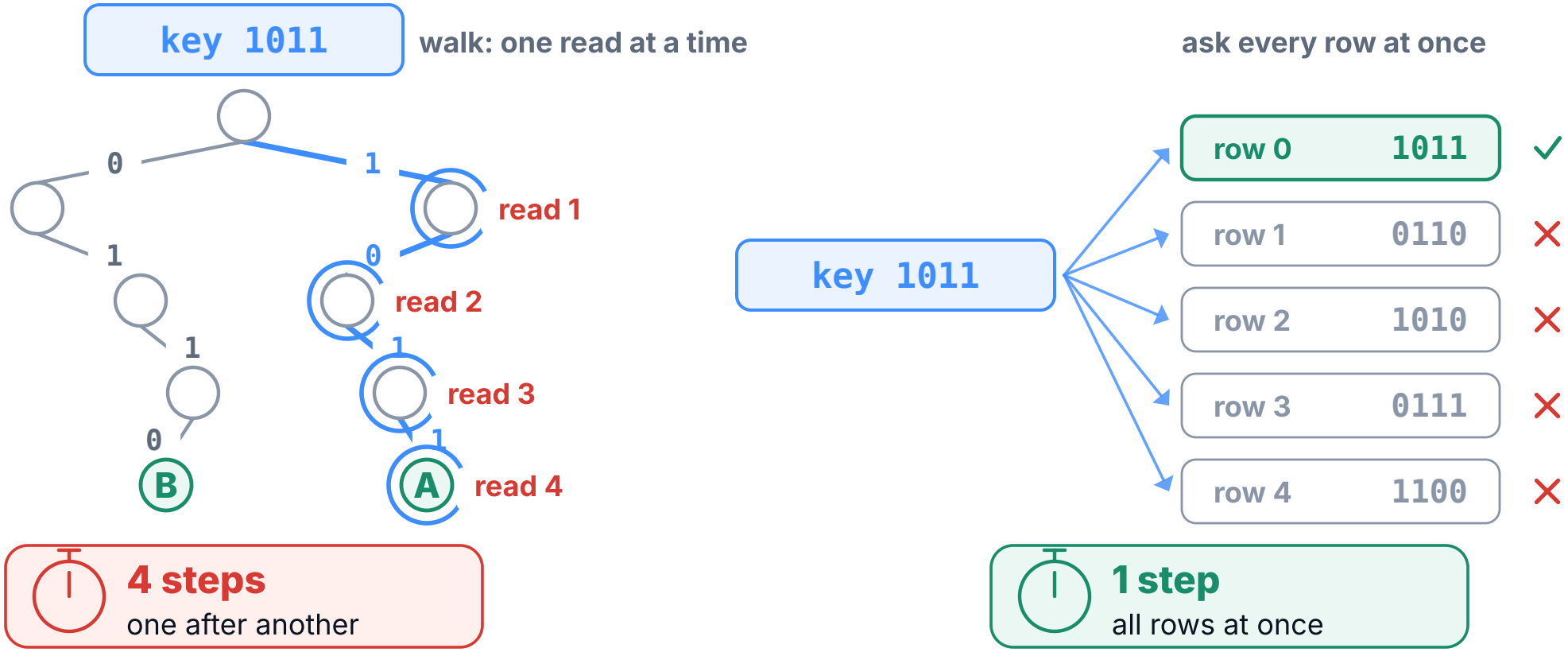
the time one packet gets

6.7 ns

five packets arrive while one lookup runs

Each read waits for the one before: 32 reads, 32 ns, against a 6.7 ns budget.

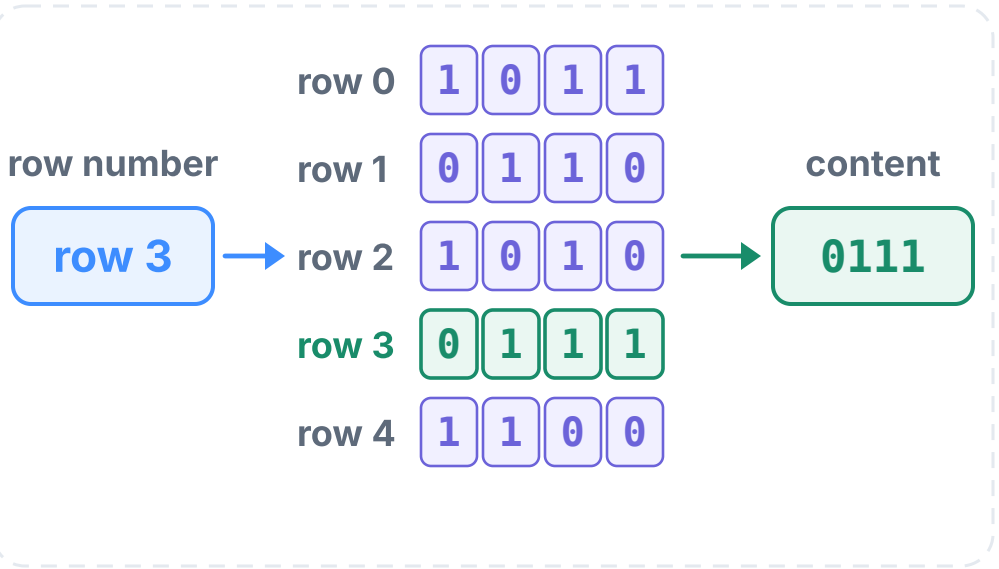
Stop walking: ask every row at the same time



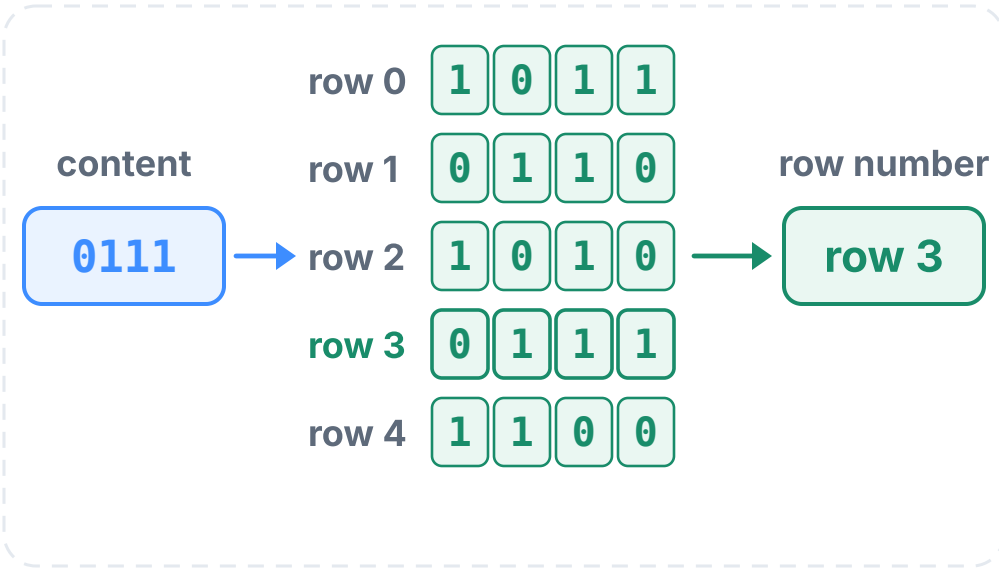
We send the key to every row at once; every row answers in the same step.

RAM and CAM: what is the address?

RAM — **random-access memory**
addressed by position: a row number



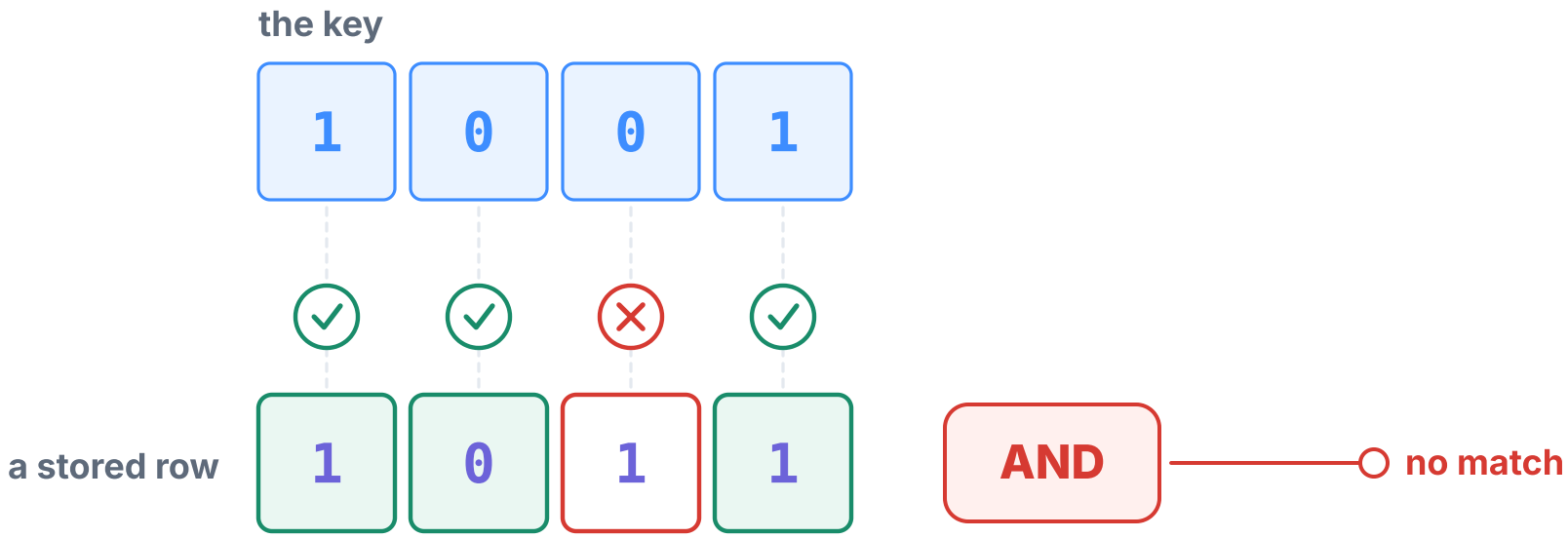
CAM — **content-addressable memory**
addressed by content: the value itself



the same array of rows; what changes is what we hand it, and what it hands back

RAM: a row number in, its content out. CAM: a content in, its row number out.

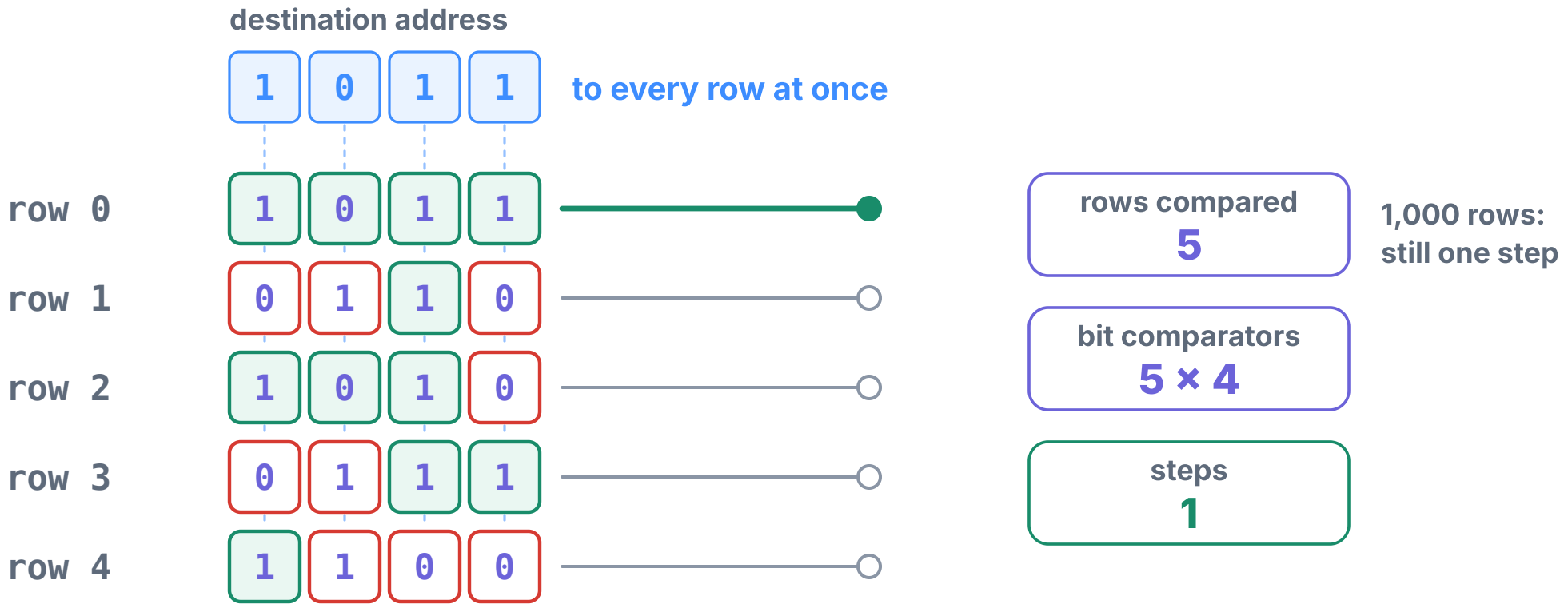
How does one row compare with the key?



every cell compares its own bit; the row matches only if all four agree

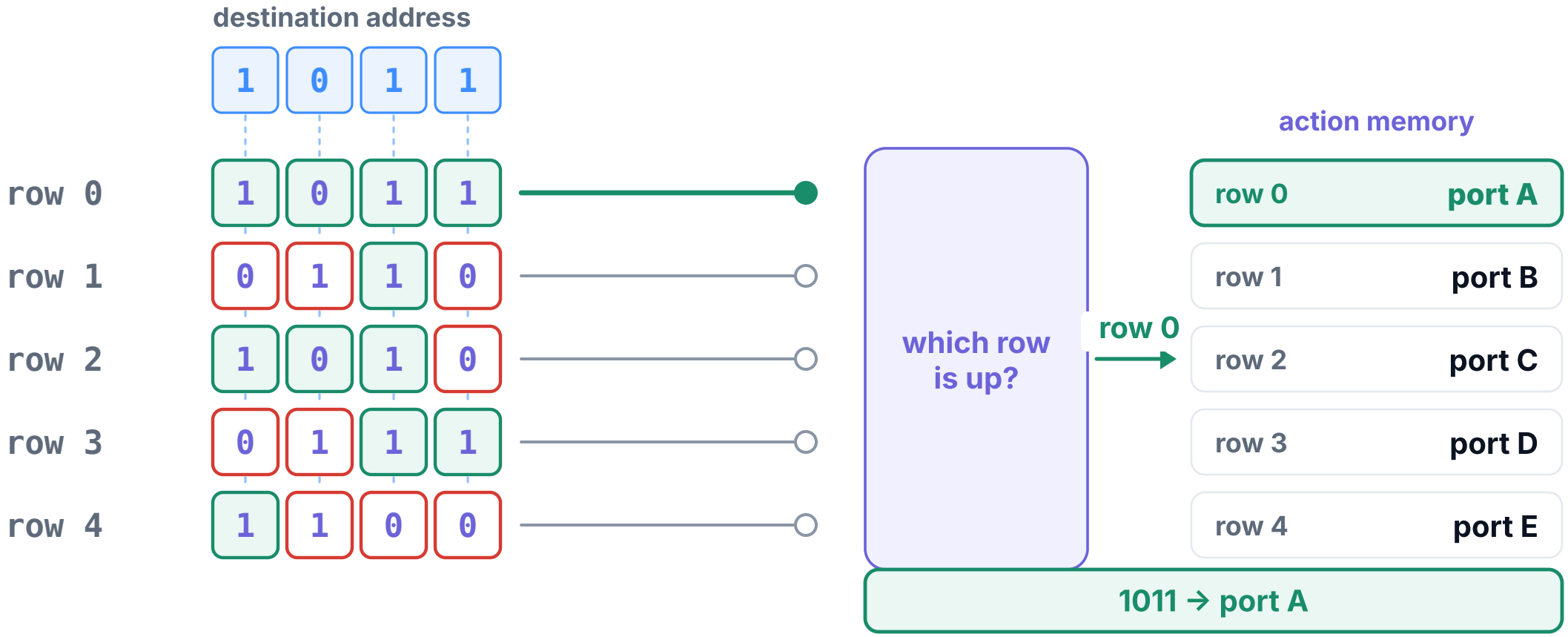
A row matches only when every one of its cells agrees with the key.

Every row has its own comparators



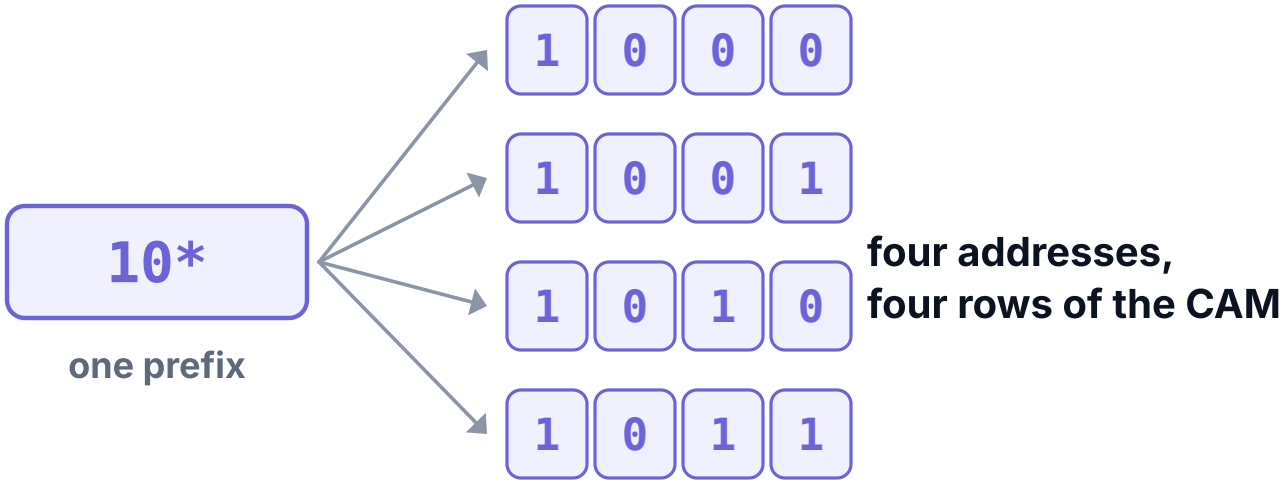
N rows, N × W comparators, one clock cycle: the time does not grow with N.

From a match line to the output port



Encode the row whose line is up, read its port: two short steps, no walk.

A prefix is not one value



a /16 prefix

2^16 = 65,536 addresses

65,536 rows for one prefix

the whole table

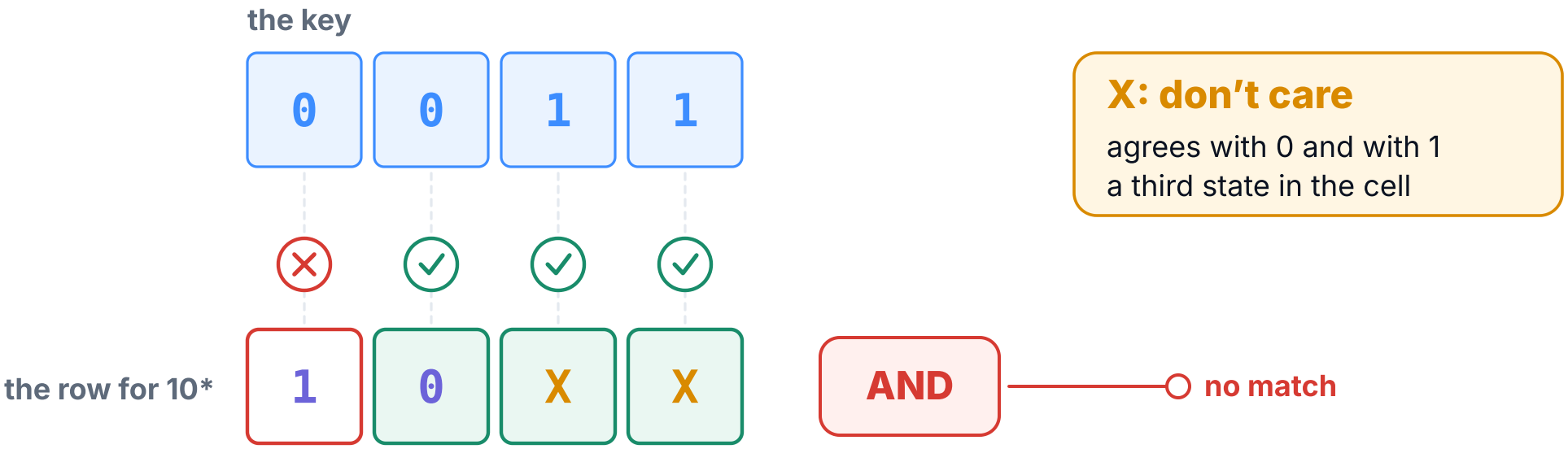
every address it covers:

billions of rows

a CAM row holds one value; a prefix stands for many

A prefix of length k stands for 2^(W−k) addresses: one row each is hopeless.

Add a third cell value: X, don’t care



the X cells never object: the row answers for every address that starts with 10

An X cell always agrees: one row now stands for a whole prefix.

Ternary CAM: five prefixes, five rows

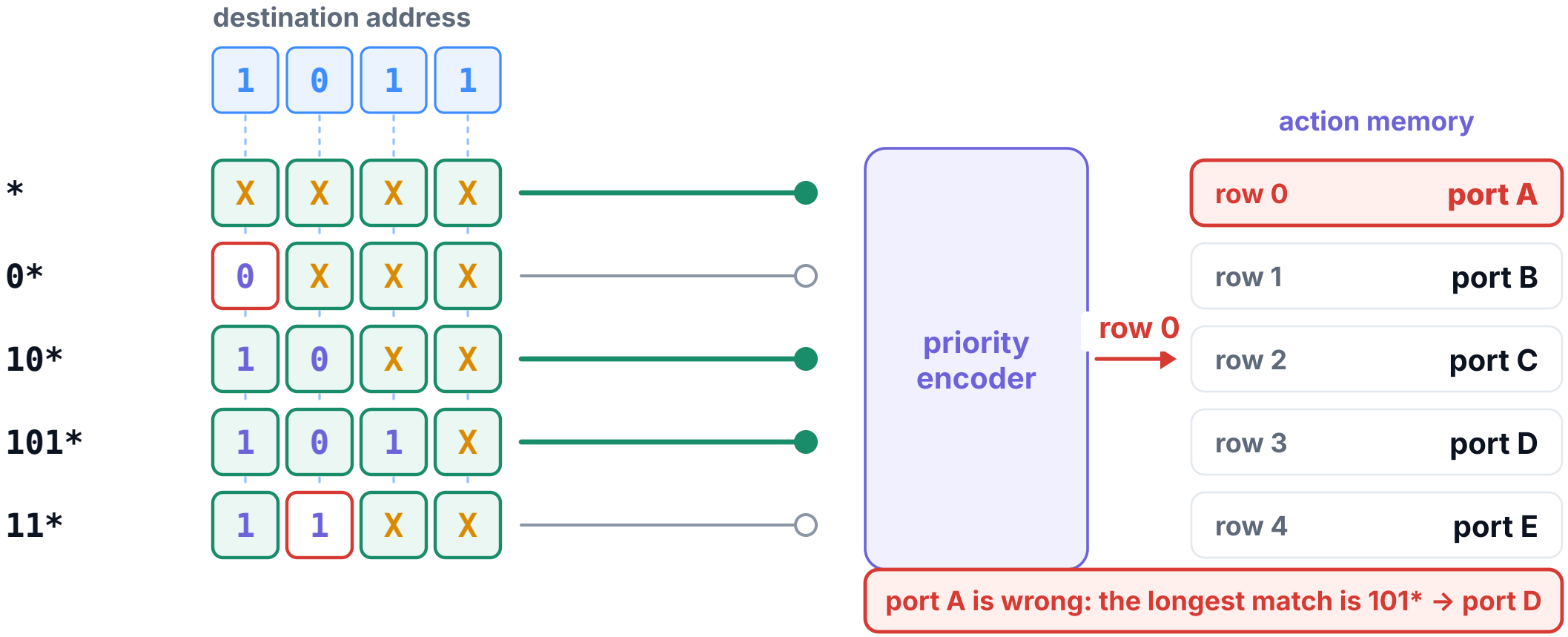


three match lines are up

, 10 and 101* all match 1011,
just as three prefixes match

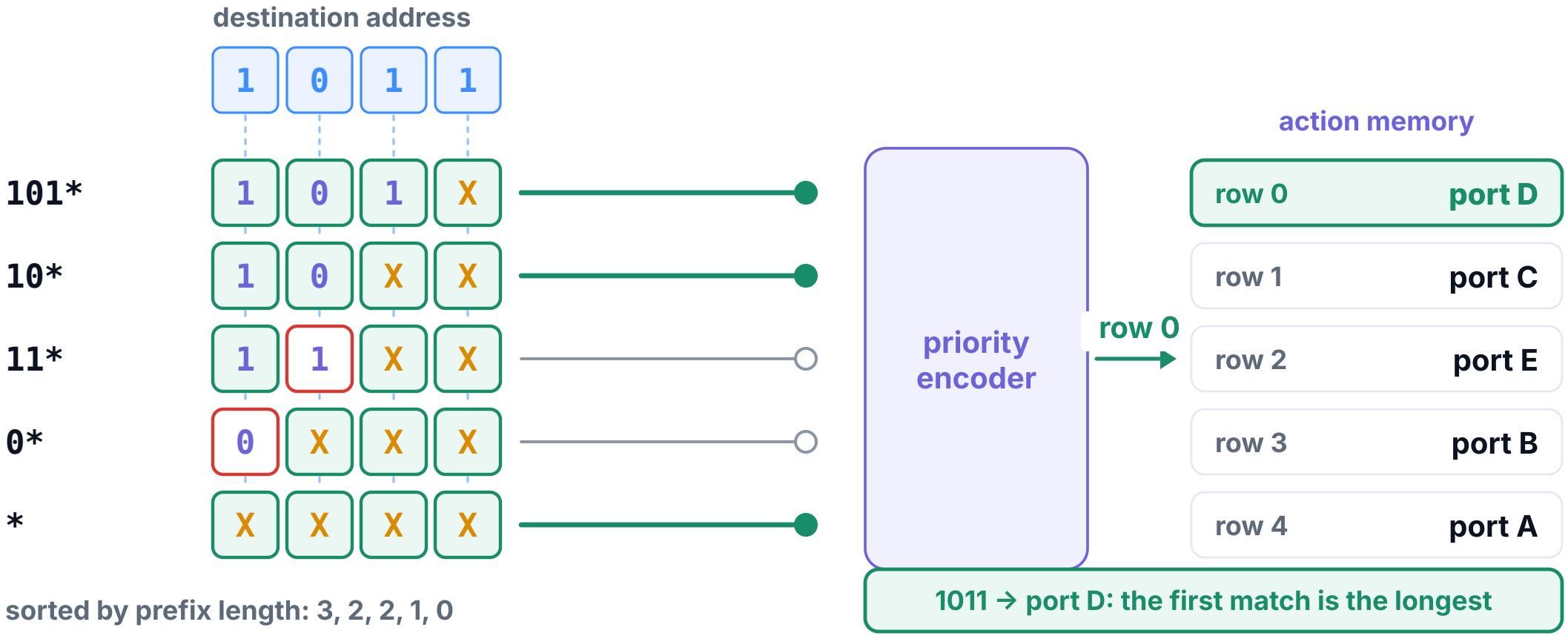
Three rows match at once, just as three prefixes match the address.

Which matching row wins?



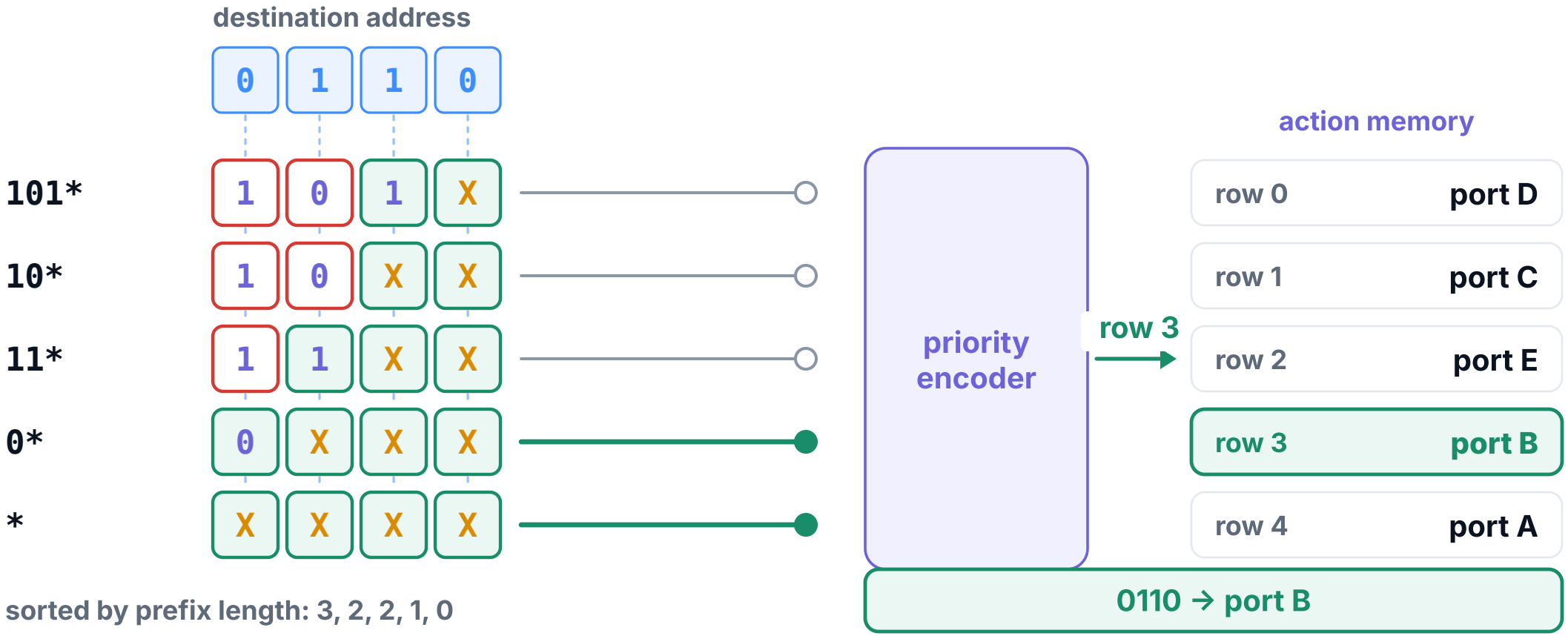
The priority encoder returns the first match; in this order that is the shortest.

Put the longest prefixes first



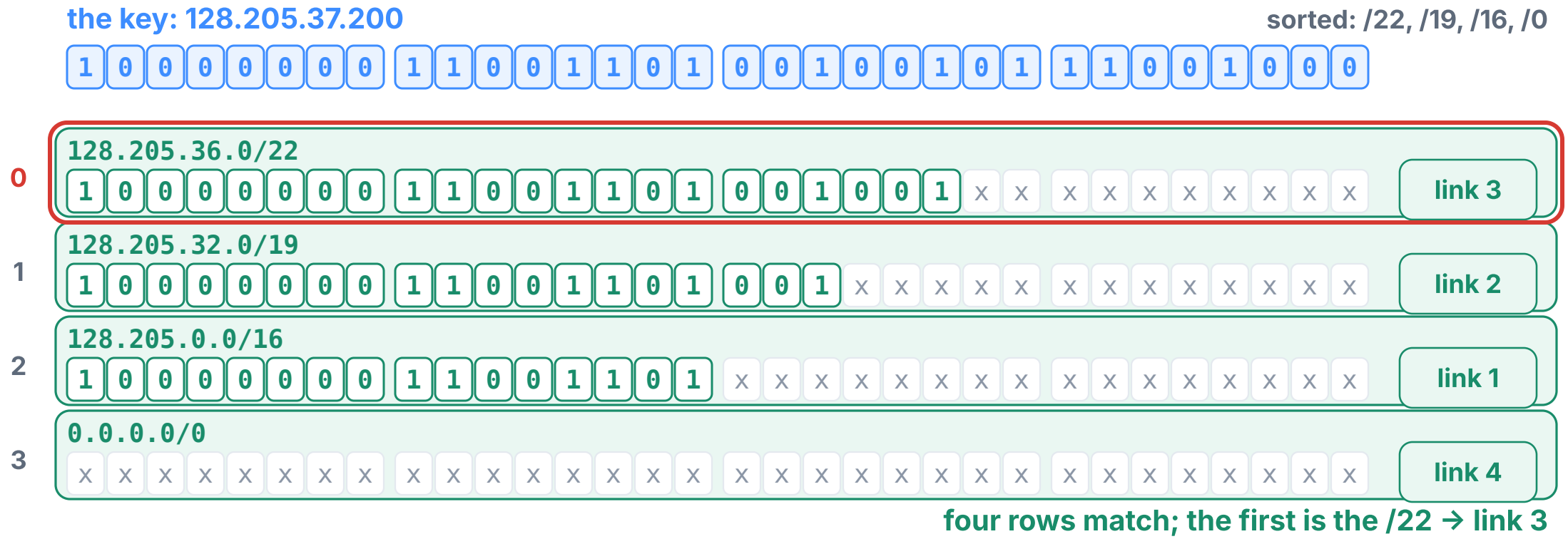
Longest first: the first matching row is the longest matching prefix.

Two more lookups, the same three steps



Compare, encode, read: the same three steps for every address.

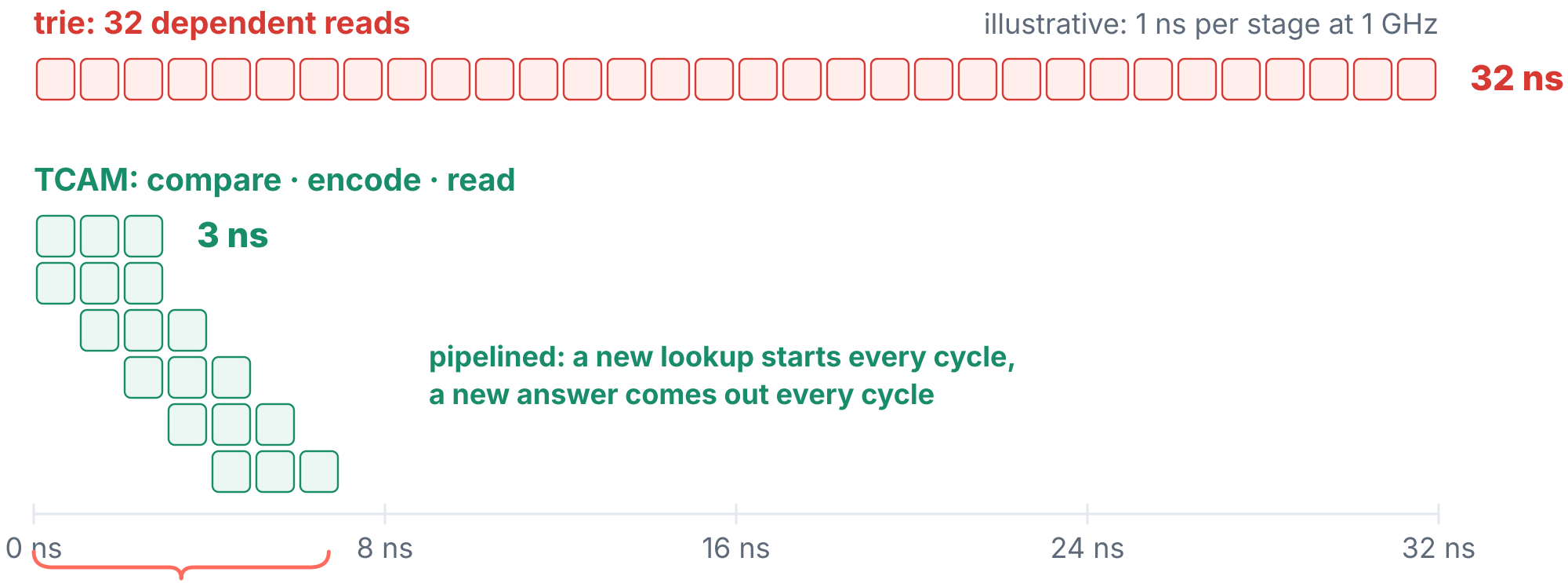
Back to 32 bits



A /19 is 19 fixed cells and 13 X cells; rows sorted by length, longest first.

32 bits

One lookup per clock cycle



One compare, one cycle: the time depends neither on W nor on N.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

What does a TCAM cell cost?

DRAM bit



1 transistor + 1 capacitor
stores; cannot compare

SRAM bit



6 transistors
stores; cannot compare

TCAM bit



about 16 transistors:
two SRAM bits + compare logic

and every cell compares on every lookup: energy grows with rows × bits

A TCAM bit is about 16 transistors, and every bit works on every lookup.

Two ways to find the same route

binary trie

cheap bits: SRAM or DRAM

memory grows with the table

a chain of up to 32 dependent reads

TCAM

one cycle, whatever W and N

about 16 transistors a bit

power on every lookup; limited size

routers use both

TCAM where the table is small or the speed is critical: classification, access lists
SRAM tries, multi-bit and pipelined, for the large forwarding table

Trie: cheap bits, a long walk. TCAM: one step, expensive bits. Routers use both.

Where the speed comes from

01



parallel compare

every row compares with the key at once: one step, not a walk

02



don’t-care cells

X agrees with 0 and 1: one row stands for a whole prefix

03



longest first

rows sorted by length: the first match is the longest match

Parallel compare, don’t-care cells, longest-first order: LPM in one cycle.

How does a packet get from one port to another?

03

The fabric

The output port is known. The packet is still on the wrong side of the router. Three designs, in the order they came.

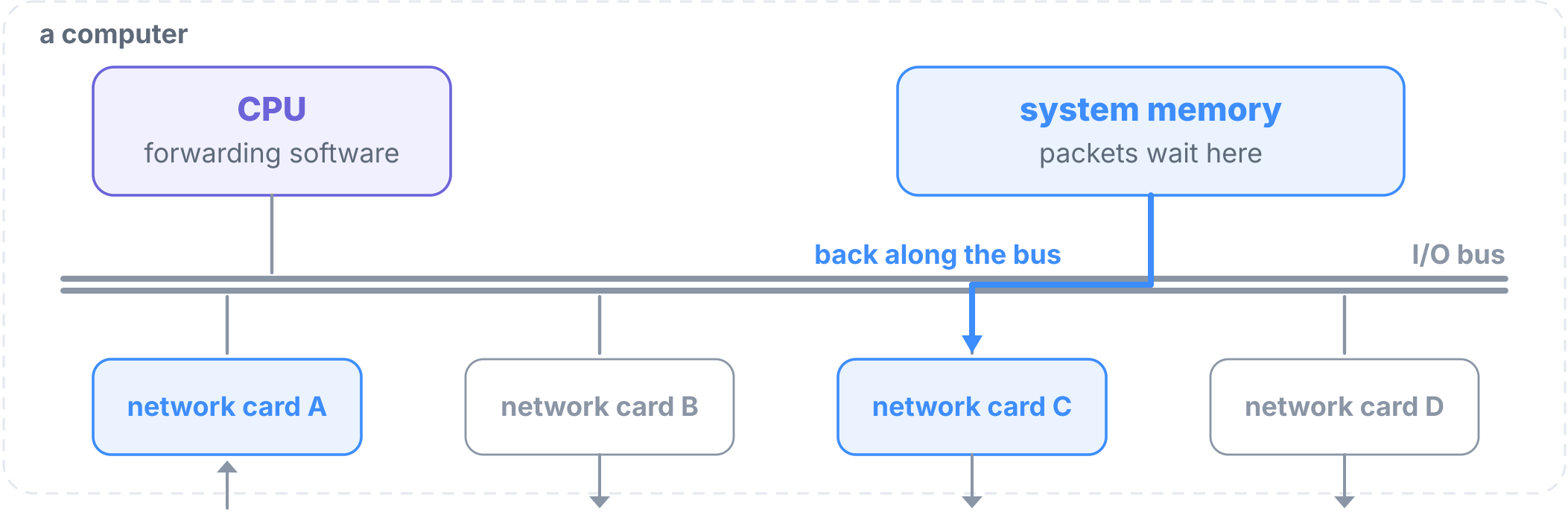
A computer with several network cards

1

The first routers

The system memory and the I/O bus move every packet. The CPU decides where it goes.

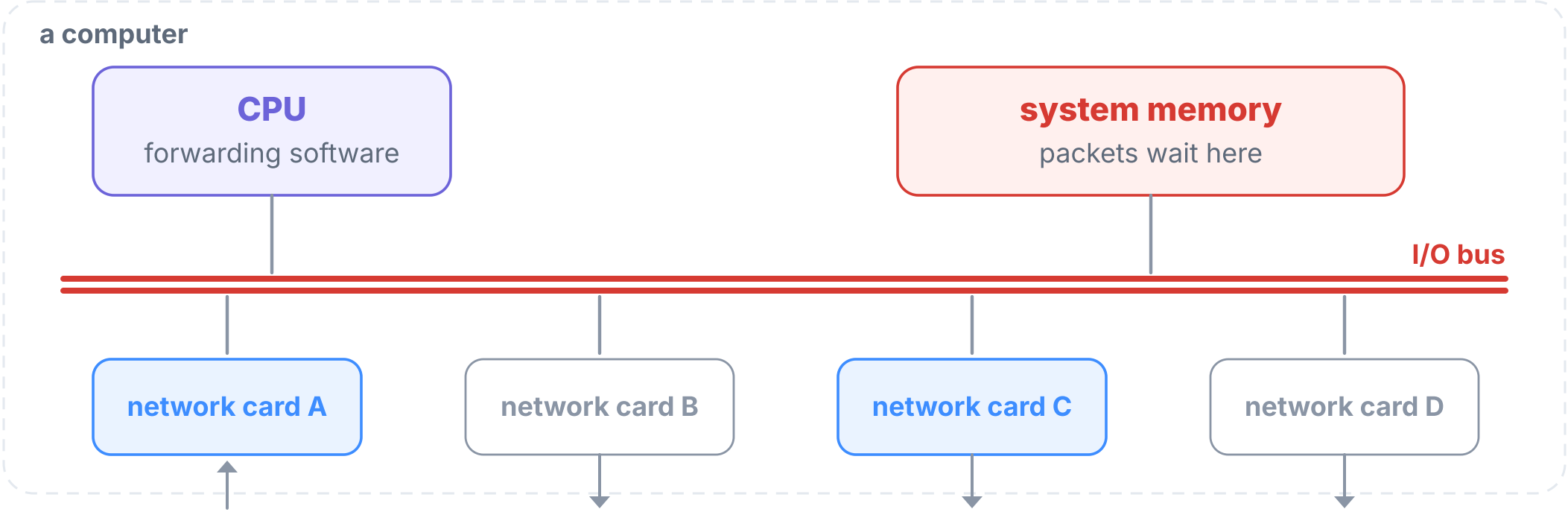
The earliest router: a computer with several network cards



3 · card C reads the packet out of memory, over the bus again, and sends it

Over the bus into memory, the CPU decides, over the bus out again: every packet.

Where is the fabric?



no dedicated fabric: the memory and the I/O interconnect do the moving

There is none. The system memory and the I/O interconnect move every packet.

Two bottlenecks: the CPU and the memory

the CPU

every packet passes through software
about a million packets a second per core
(Part 1): one port delivers 149 million

the memory

every packet is written once and read once
shared by every port
packet data alone: 2 × the input rate

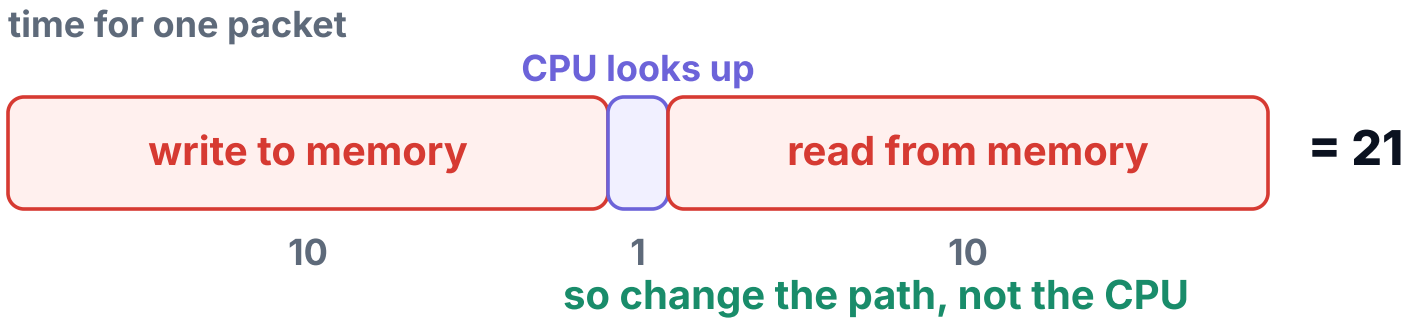
N links at R each: input N R → memory traffic ≈ 2 N R

16 × 100 Gbps = 1.6 Tbps in → 3.2 Tbps of memory bandwidth, before any overhead

no memory of that era came close

Each packet costs CPU time and two memory passes. Both are shared by every port.

A faster lookup does not move the packet



→ move the work to the ports

the lookup on the line card, no central CPU

move the path off the memory

a backplane carries the packet between cards

Partridge et al., A 50-Gb/s IP router, 1998: forwarding engines beside the ports, a switched backplane between them.

An instant lookup still leaves two memory passes. Change the path, not the CPU.

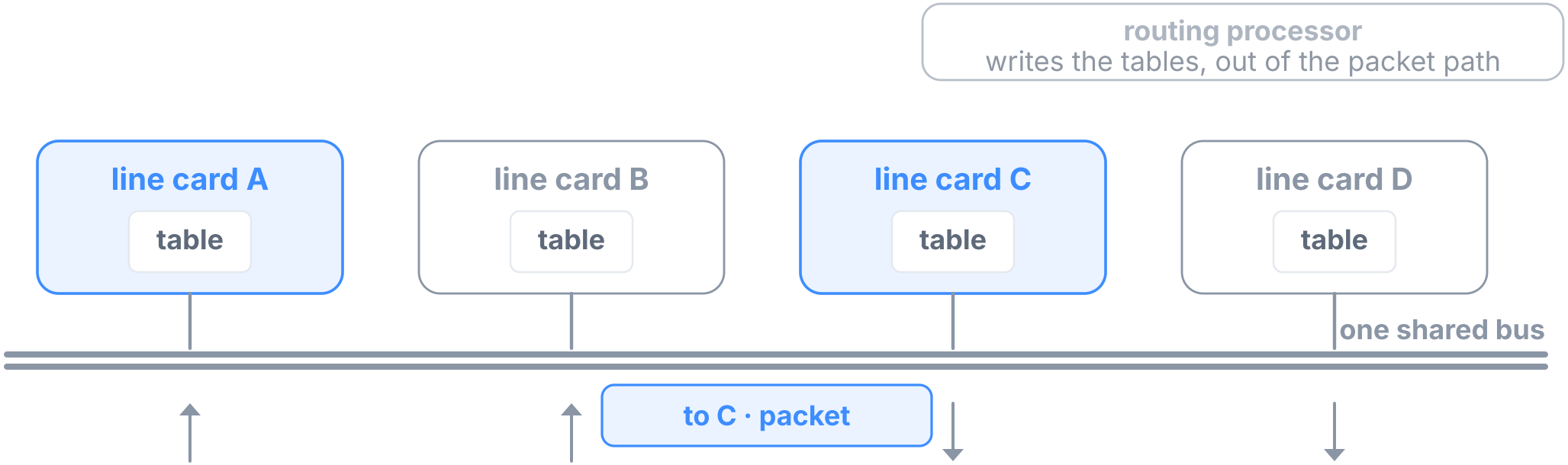
Line cards on a shared bus

2

No CPU in the path

Each line card looks up for itself. Packets cross one shared wire, one at a time.

Put the lookup on the line card

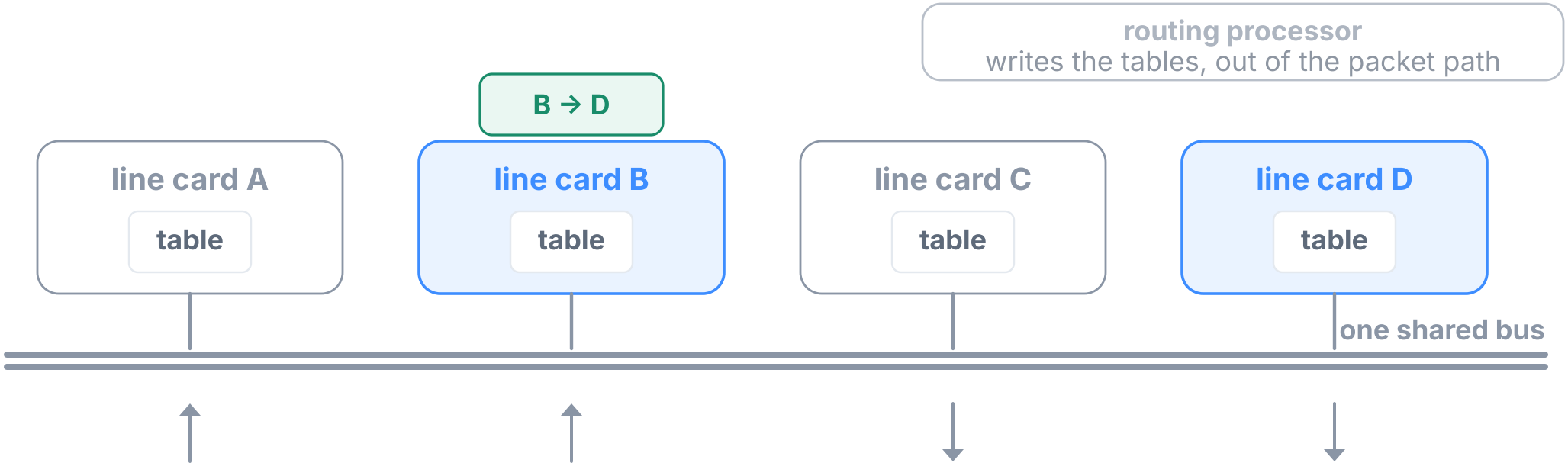


3 · card C takes it; the other cards ignore it. No CPU, no memory copy

The input card decides and tags; the bus carries. The CPU is out of the path.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

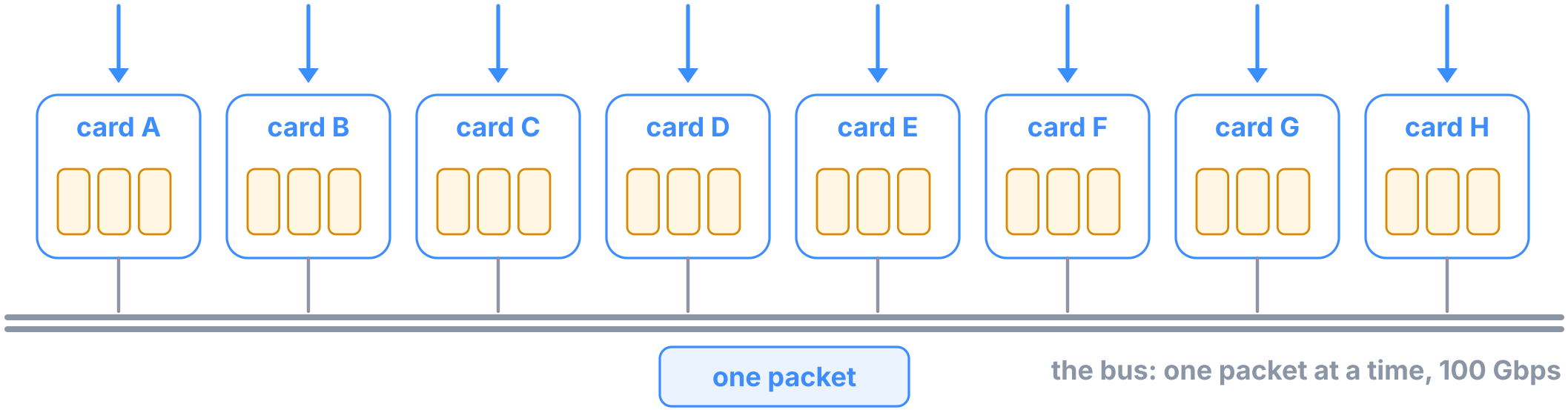
One bus, one transfer at a time



only now does B → D go

Transfers that do not conflict still serialise on the one shared path.

The bus is the ceiling



wanted: 8 × 100 = 800 Gbps carried: still 100 Gbps

add cards: more piles, the same bus

A 100 Gbps bus is 100 Gbps however many cards hang on it.

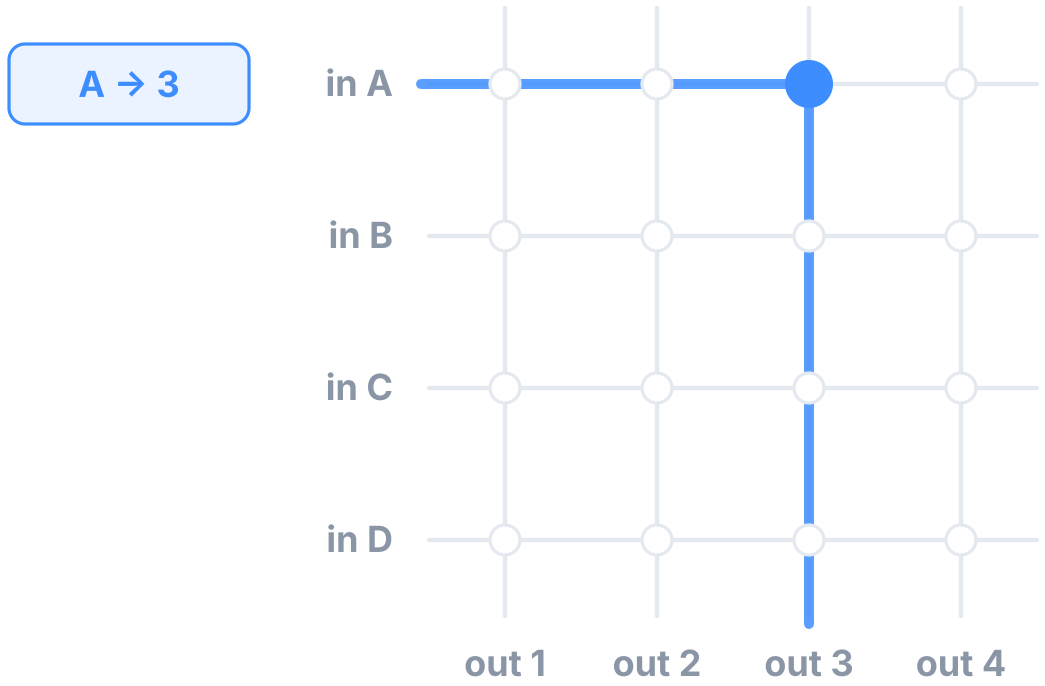
A crossbar

3

Many connections at once

A switch at every input–output crossing. New problems: contention, and scale.

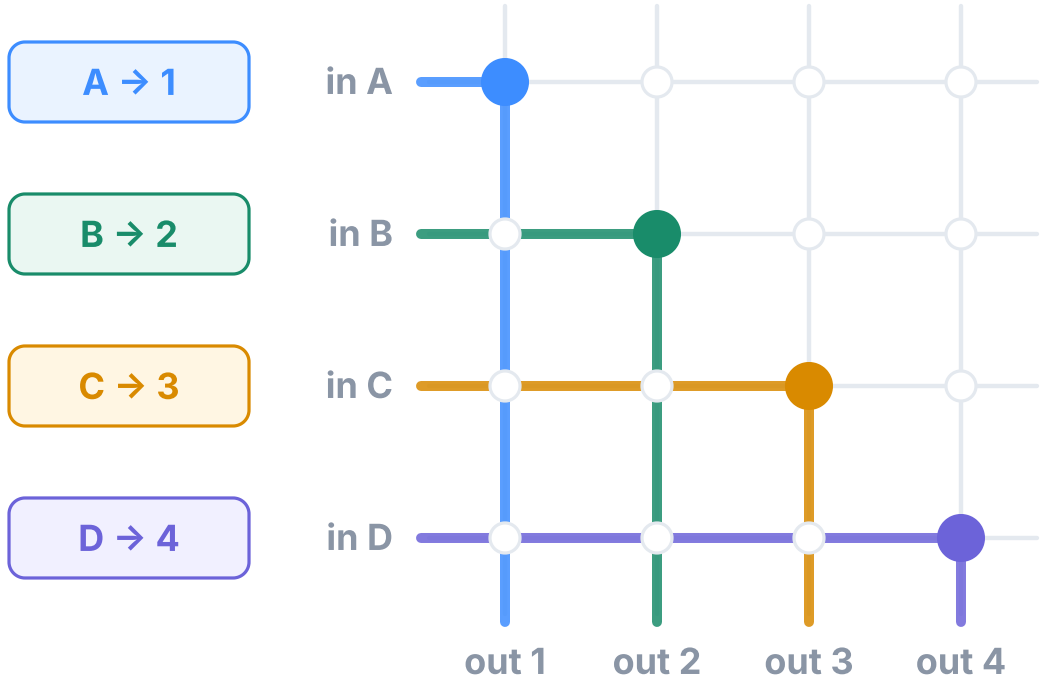
What is a crossbar?



a packet follows the connection

A switch at every crossing. Close one, and that input is wired to that output.

Four connections, four packets at once



four crosspoints closed, four connections

no two share a wire,
so all four can carry
a packet at the same time

four packets, one slot

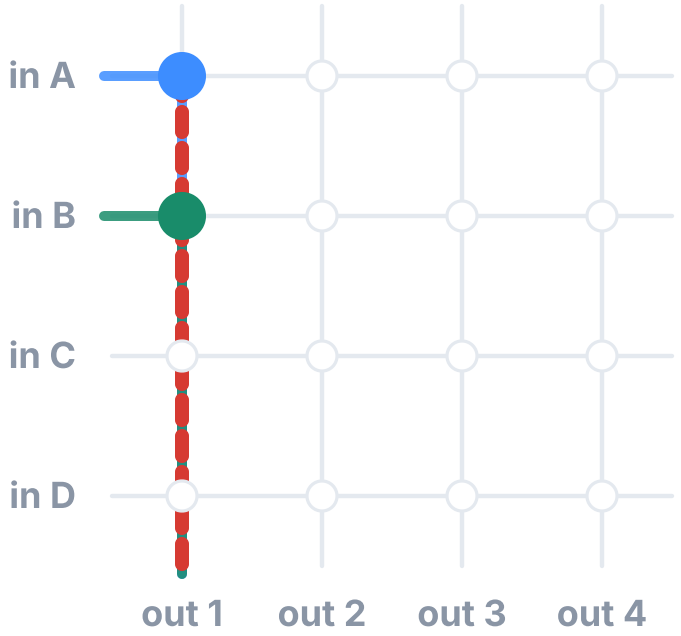
McKeown, A fast switched backplane for a gigabit switched router, 1997.

Different rows, different columns: four packets cross in the same slot.

Which sets can go together?

this slot

A→1, B→2, C→3, D→4 ✓	A → 1
A→3, B→1, C→4, D→2 ✓	B → 1
A→1, B→1 ✗	



two paths on the wire of out 1: conflict

Distinct rows and columns go together. Two inputs for one output: their paths overlap.

Count what crossed in one slot

shared bus

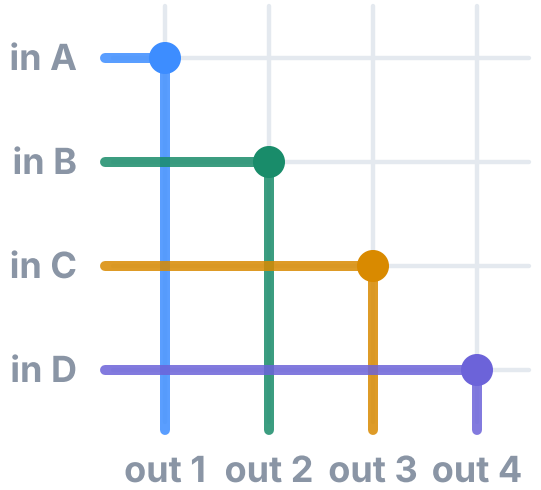


one slot: 1 packet crosses

100 Gbps, whatever N is

16 ports: still 100 Gbps

crossbar



one slot: 4 packets cross

N × 100 Gbps

16 ports: 1.6 Tbps

A bus moves one packet per slot. A crossbar moves up to N.

Two new problems

⚠ contention

several inputs want the same output
in the same slot
who goes, and where do the others wait?

first

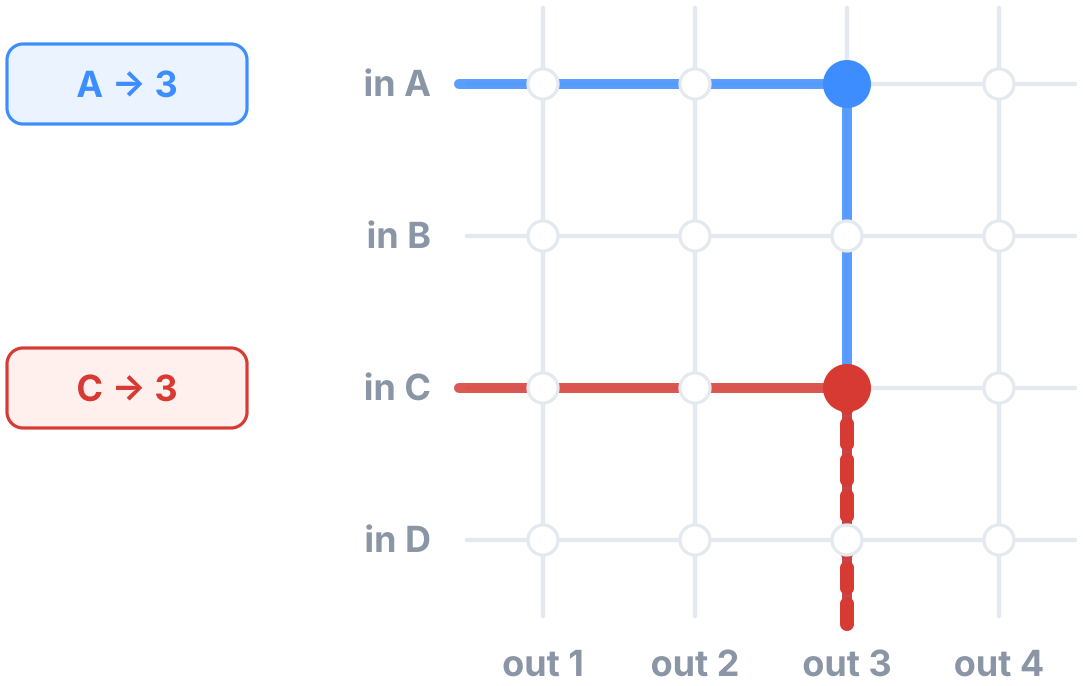
scale

N × N crosspoints: 16 ports, 256 switches
64 ports, 4,096 · 512 ports, 262,144
wiring and arbitration grow with them

later in this part

Installing a crossbar is not enough: contention first, then scale.

Two inputs, one output

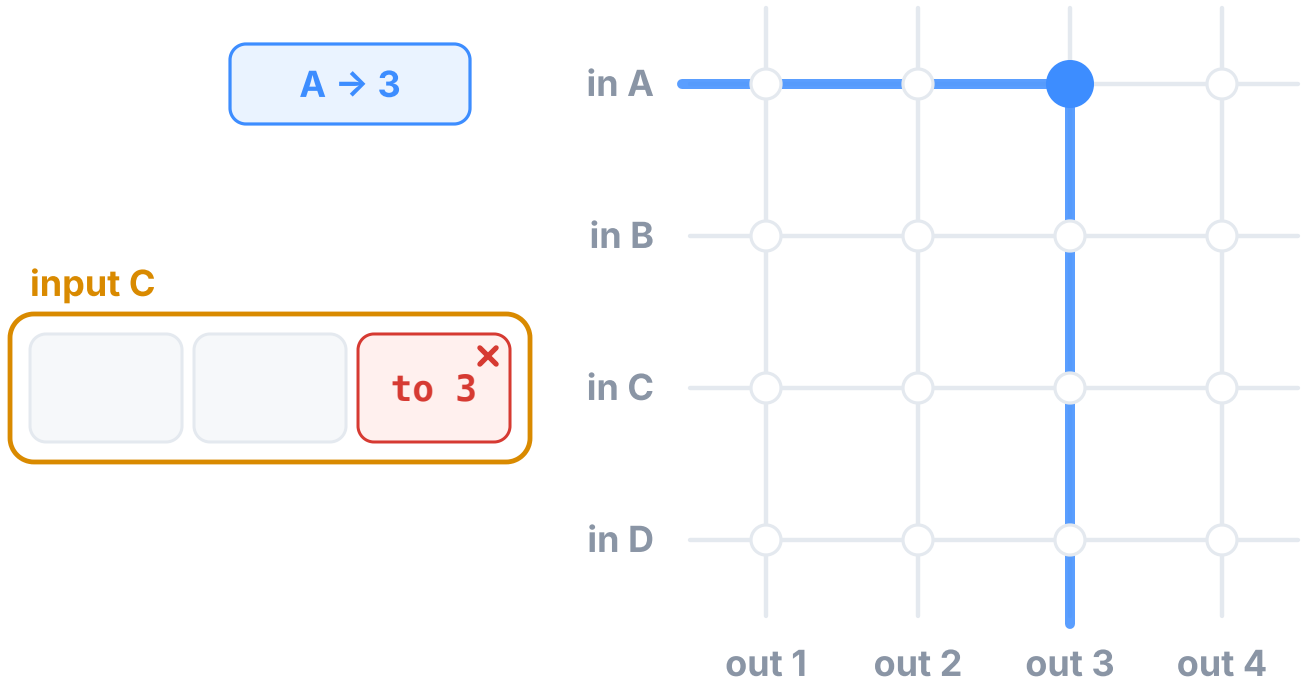


a wire carries one packet per slot

so only one crosspoint in column 3 may close in this slot

A and C both want output 3 in the same slot. Its wire can take one packet.

One goes, one waits



where does it wait?

in a small buffer at its own input port:

an input queue

The arbiter picks A. C's packet is stored at its input port: an input queue.

The next slot

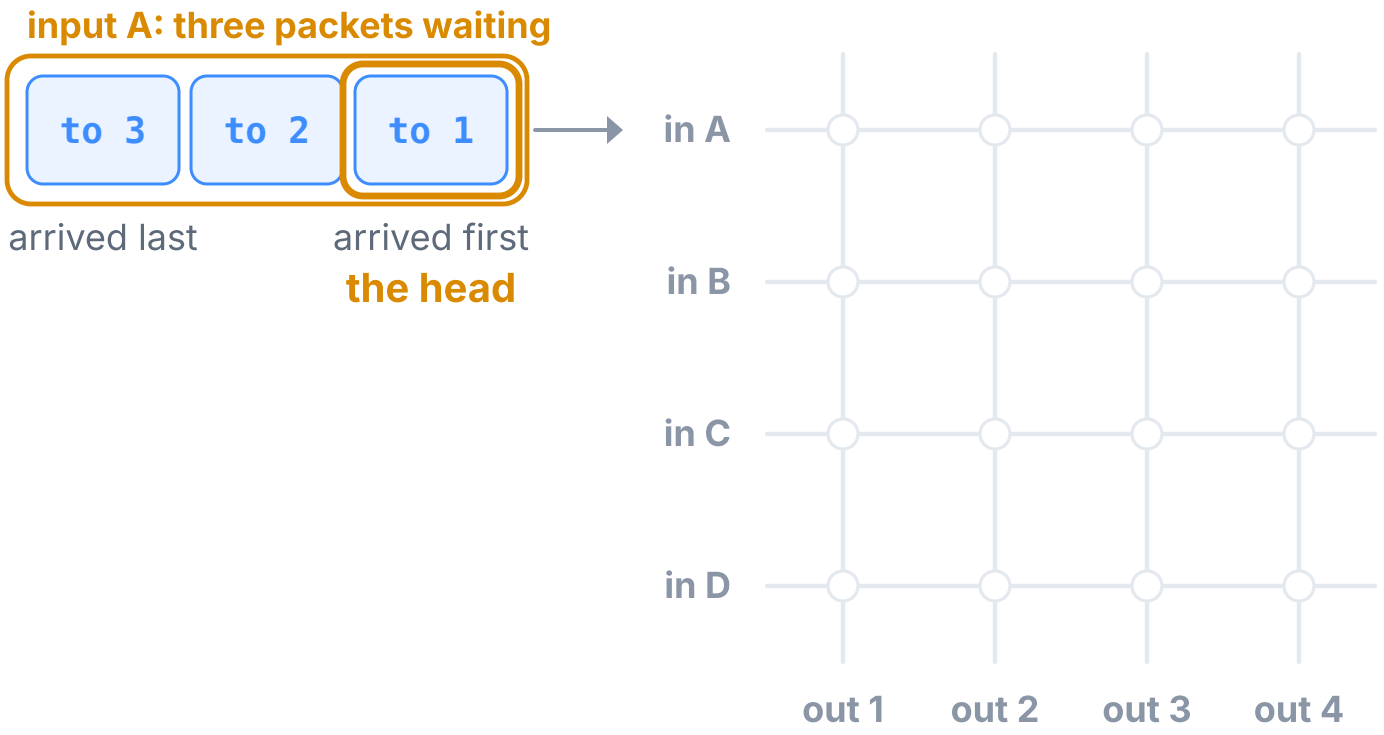


A in slot 1, C in slot 2: the loser is delayed by one slot, not lost

as long as its input buffer has room

Contention costs time, not packets: the loser goes in the next slot.

Packets line up at an input

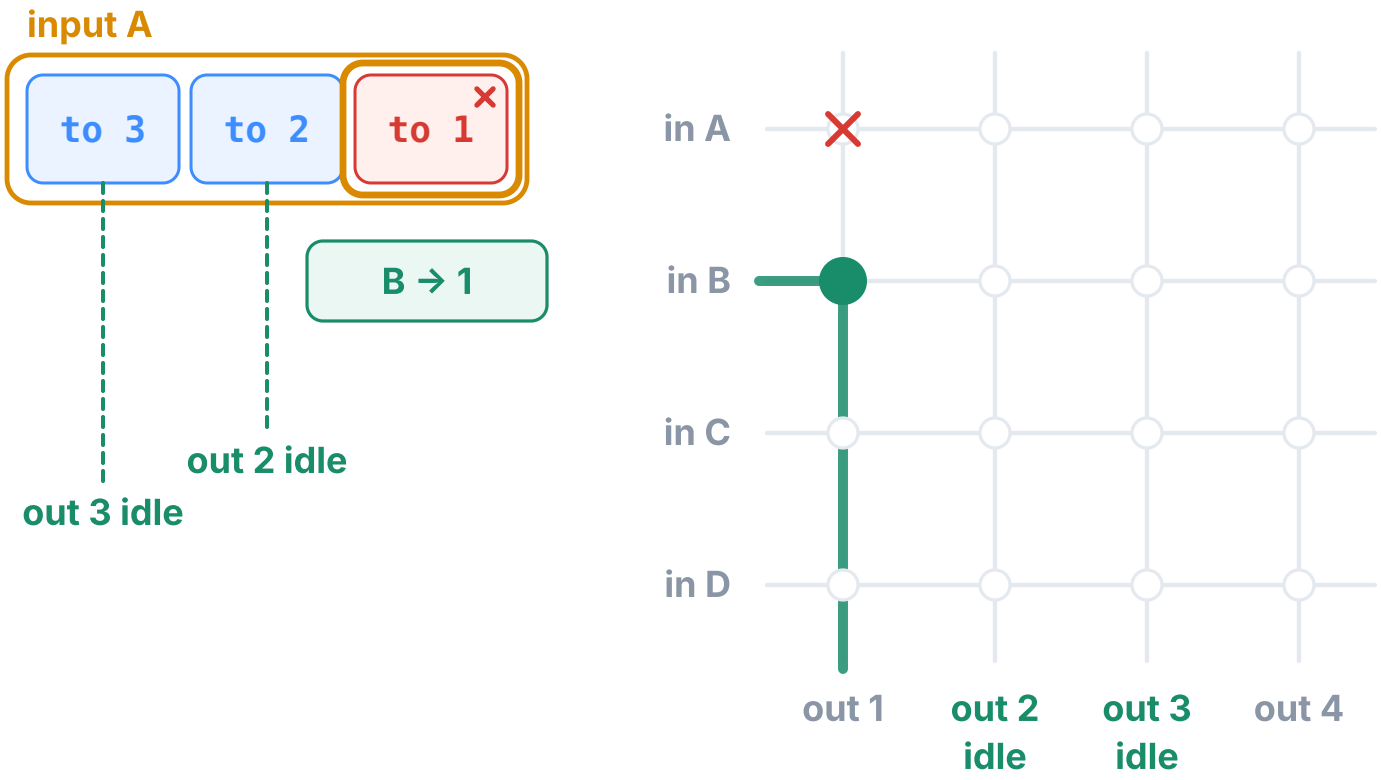


the crossbar is offered one packet: the head

the two behind it are invisible to it

A FIFO shows the crossbar one packet: the one at its head.

Slot 1: the head cannot go, so nothing goes



behind the head:
packets for 2 and 3

both outputs idle,
both packets stuck:

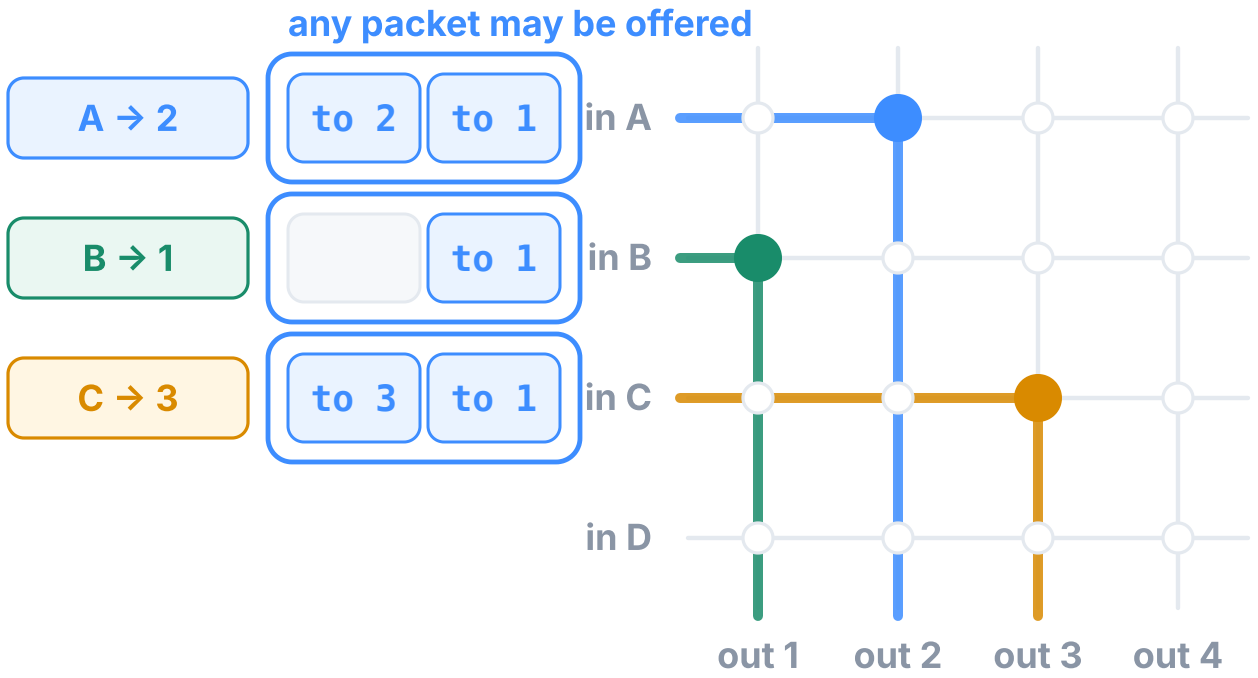
head-of-line blocking

The head waits for out 1. Behind it, packets for the idle outputs 2 and 3 wait too.

The waste, on the crossbar

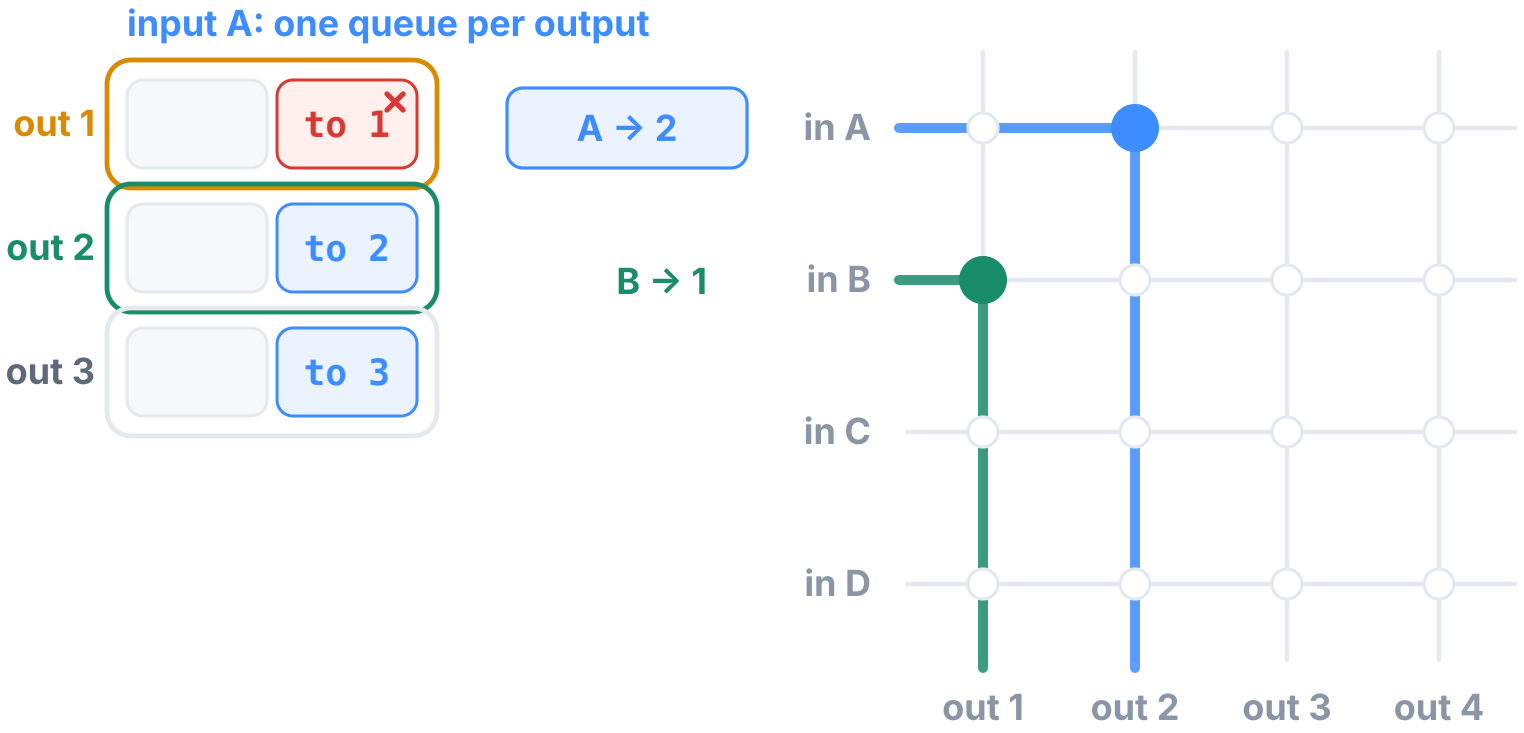
any packet:
A → 2, B → 1, C → 3

3 packets cross, nothing wasted



Heads only: one packet crosses. Any packet: three. The FIFO was the limit.

The fix: one queue per output

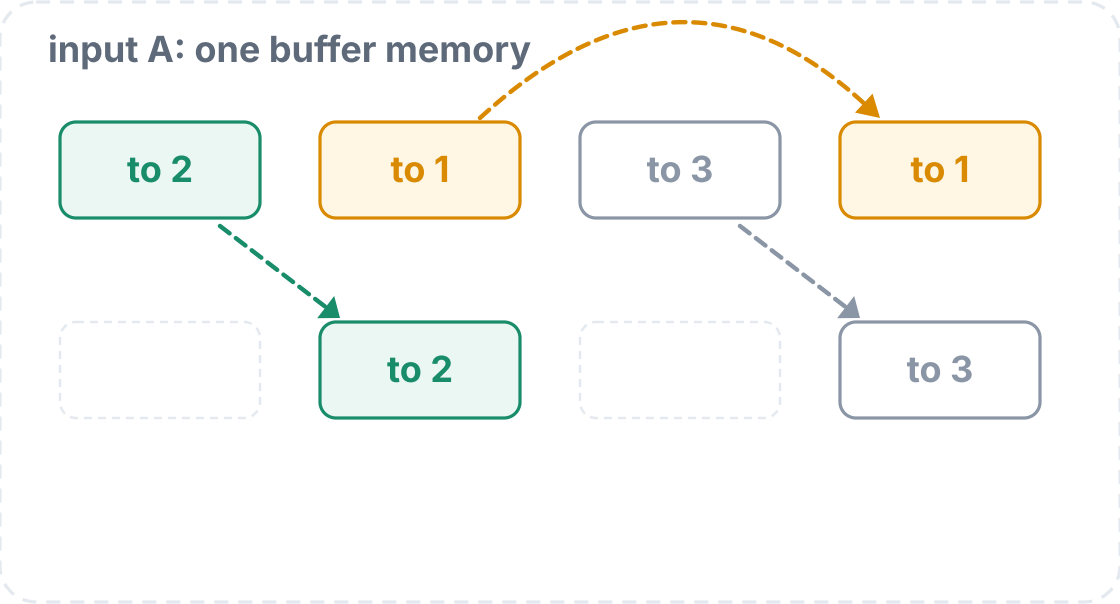


the queue for 2 is offered, and goes

no slot wasted

Three queues, three heads. A packet for a free output is never hidden behind a busy one.

Why “virtual”?



the packets sit anywhere in one memory

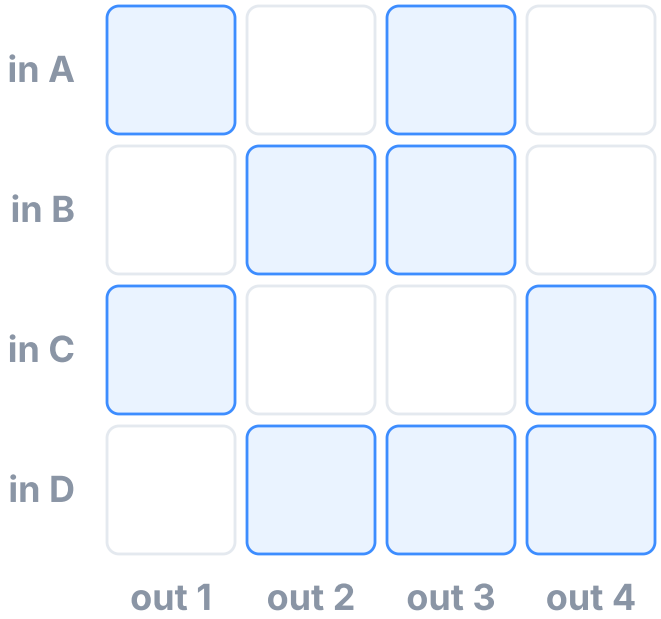
a queue is a list of pointers, one per output

no memory per output;
that is what virtual means

Real systems add a queue per class of service and let credits from the output side gate the sending; Cisco 8000 documents VOQs for this.

Virtual output queues: three lists in one buffer, not three memories.

What the scheduler sees

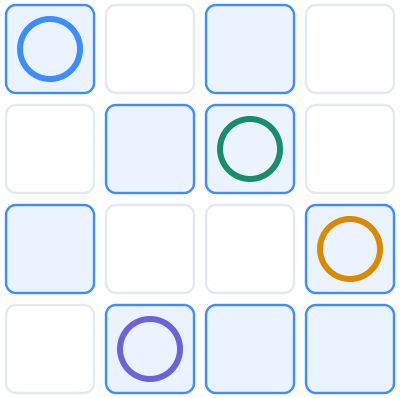


every input reports its non-empty queues

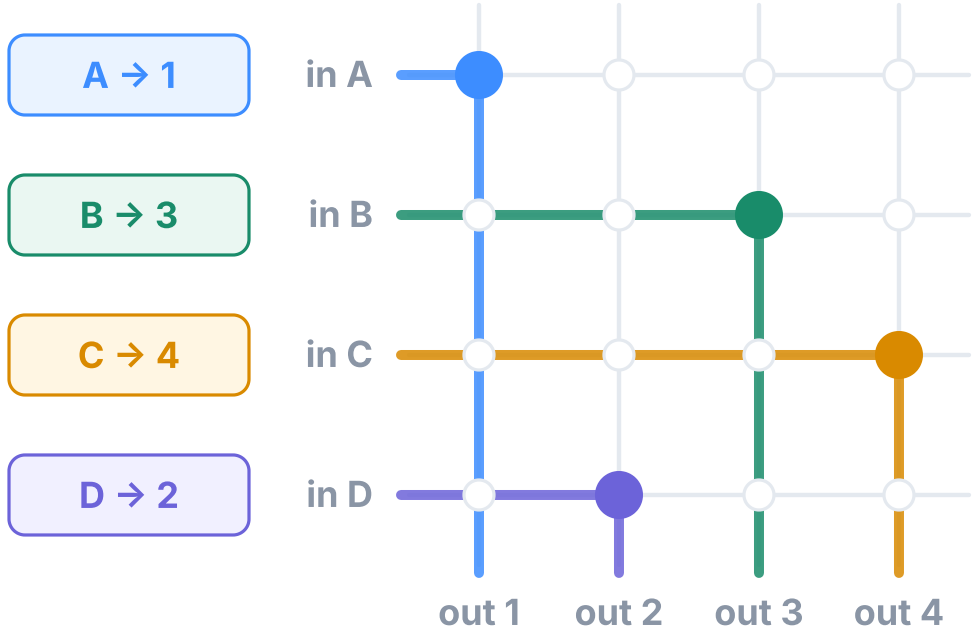
this is what the scheduler sees at the start of a slot

Rows are inputs, columns outputs. A filled cell: this input holds a packet for that output.

The scheduler’s job: as many as possible, no wire shared



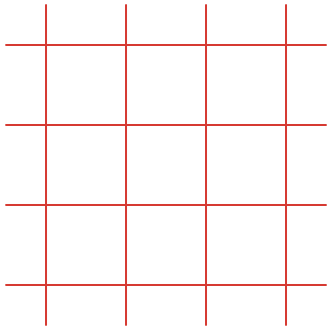
the chosen cells become the closed crosspoints



iSLIP (McKeown, 1999) finds such a set in a few rounds of request, grant and accept; most crossbar schedulers descend from it.

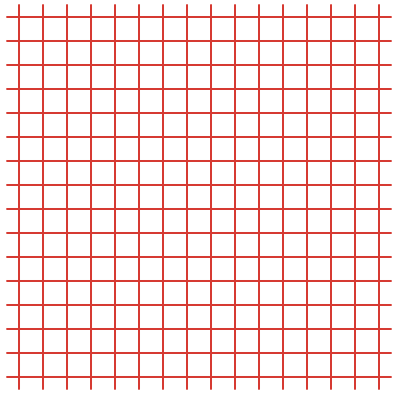
Pick cells with no row or column twice, as many as you can. Every slot, in nanoseconds.

One giant crossbar does not scale



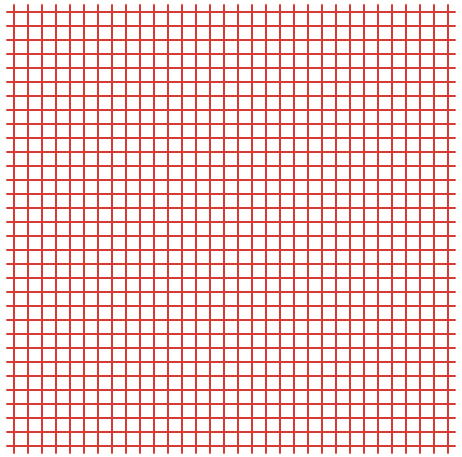
4 × 4

16 crosspoints



64 × 64

4,096



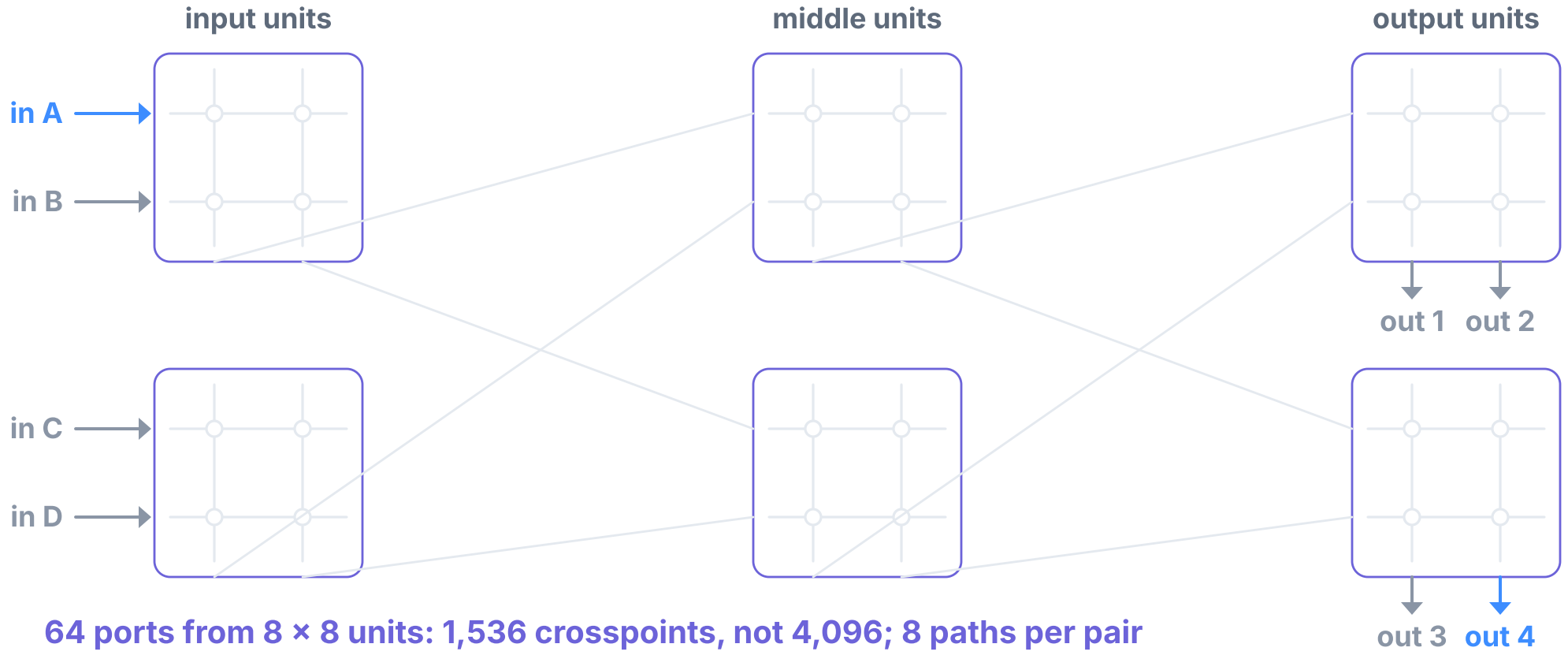
512 × 512

262,144

pins, wiring, arbitration: all grow with N^2

Crosspoints grow as N^2. Past a few hundred ports one crossbar is not buildable.

A network inside the router: small crossbars in stages



64 ports from 8 × 8 units: 1,536 crosspoints, not 4,096; 8 paths per pair

Small crossbars in three stages: fewer crosspoints, more than one path per pair.

What multistage buys and costs

✓ what it buys

repeatable small modules instead of
one ever-larger crossbar
path redundancy

⚠ what it costs

more internal hops
path assignment, load balancing,
buffering, flow control

multistage is not automatically non-blocking

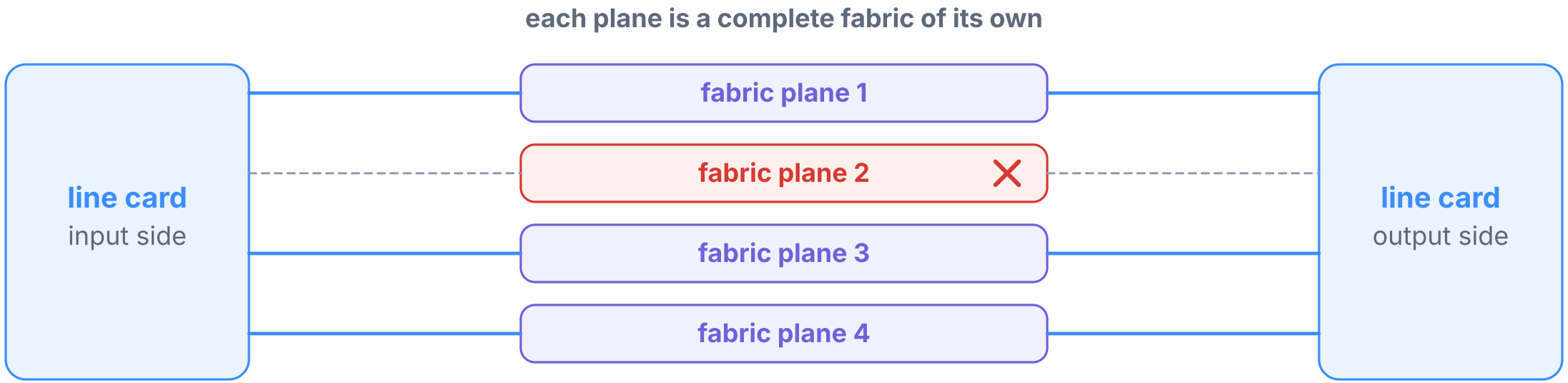
the topology, the internal bandwidth and the scheduling decide

Mature, not new: Cisco CRS used a buffered three-stage Benes fabric from the start.

Modules and redundancy, paid for in hops, balancing and flow control.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

Several fabrics in parallel



plane 2 fails: the other three carry on, at three quarters of the capacity

Not one fabric but several planes side by side: more capacity, and one can fail.

Two packets, one wire: who waits, and who goes first?

04

Queues and scheduling

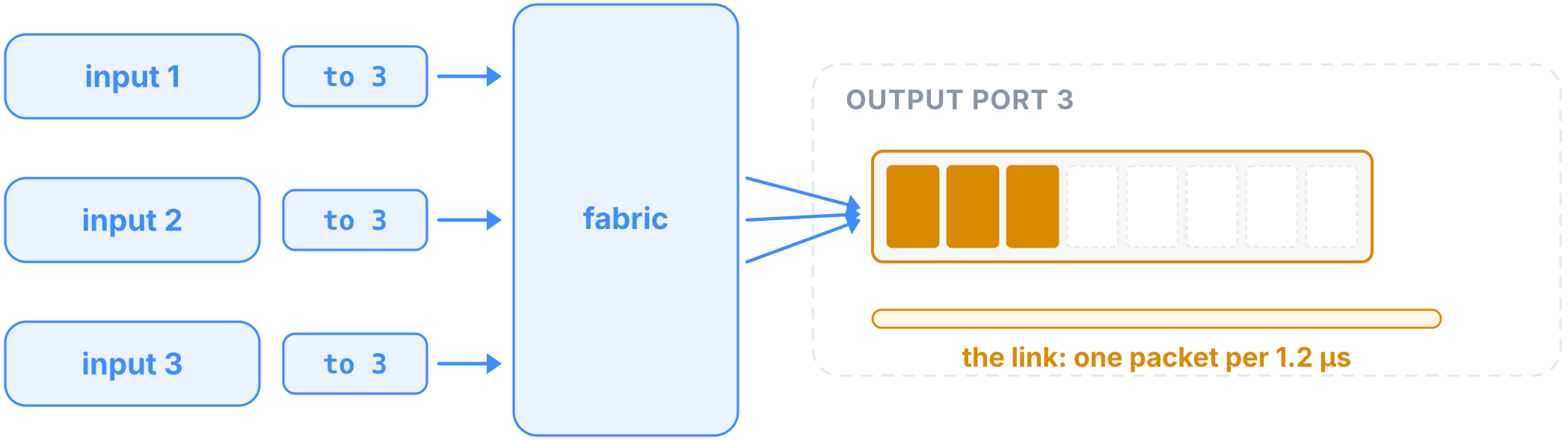
The fabric delivered them both to output port 3. The wire sends one at a time.

What is inside an output port?



A buffer, the framing, and the wire. Packets that arrived too early wait in the buffer.

Why does the queue grow?

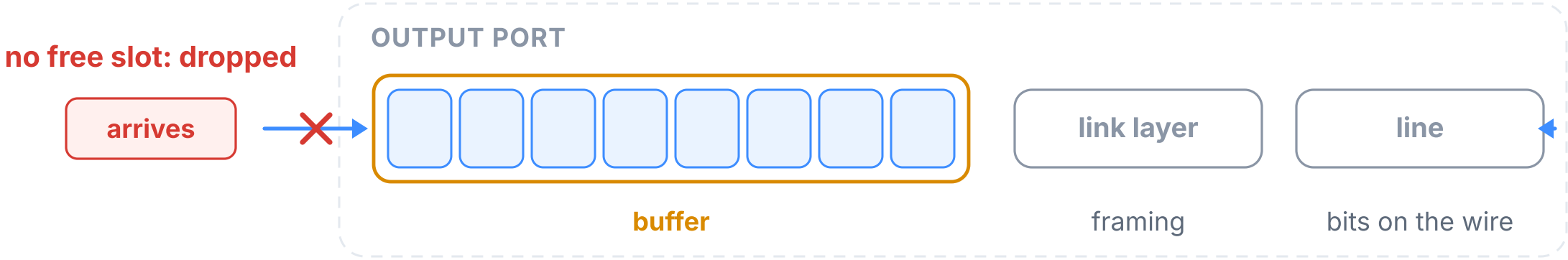


three arrive in one slot; one leaves

Press → three times: one packet from each input, in the same slot.

Packets arriving faster than the link sends them pile up in the buffer.

What happens when the buffer is full?



A packet that arrives at a full buffer is dropped. This is where loss comes from.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

In what order do they leave?



Press → to send the next packet.

FIFO: arrival order is departure order. What a buffer does if nobody tells it anything.

What does one heavy flow do under FIFO?



one download, sending as fast as it can

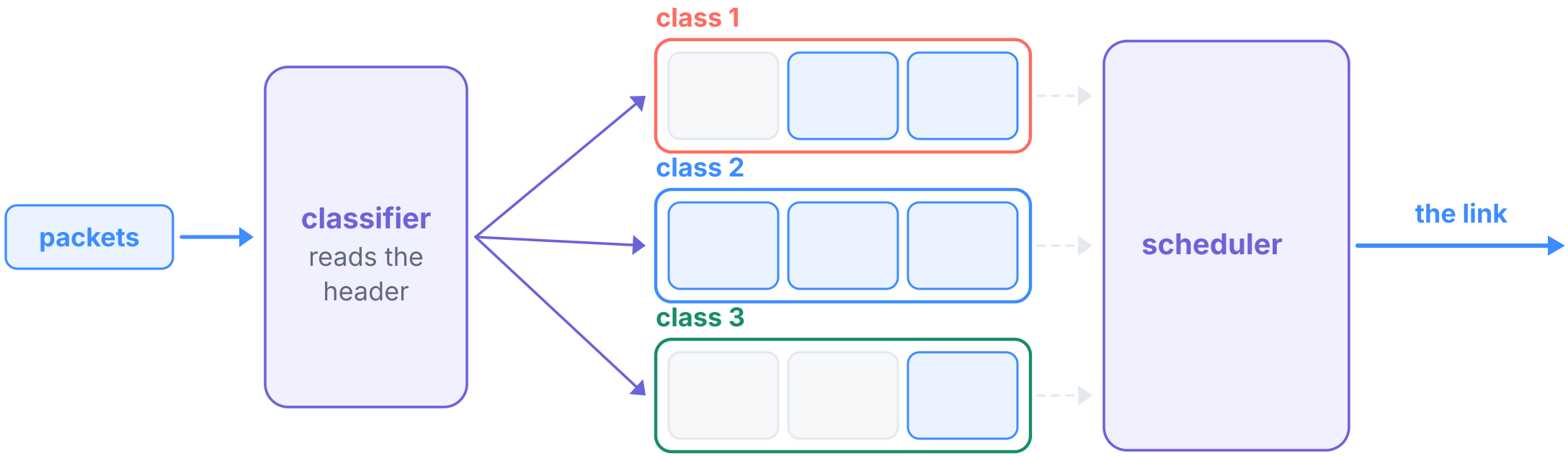
one voice packet, every 20 ms

the voice packet waits behind nine download packets, every time

the download does nothing wrong; FIFO simply serves whoever fills the queue

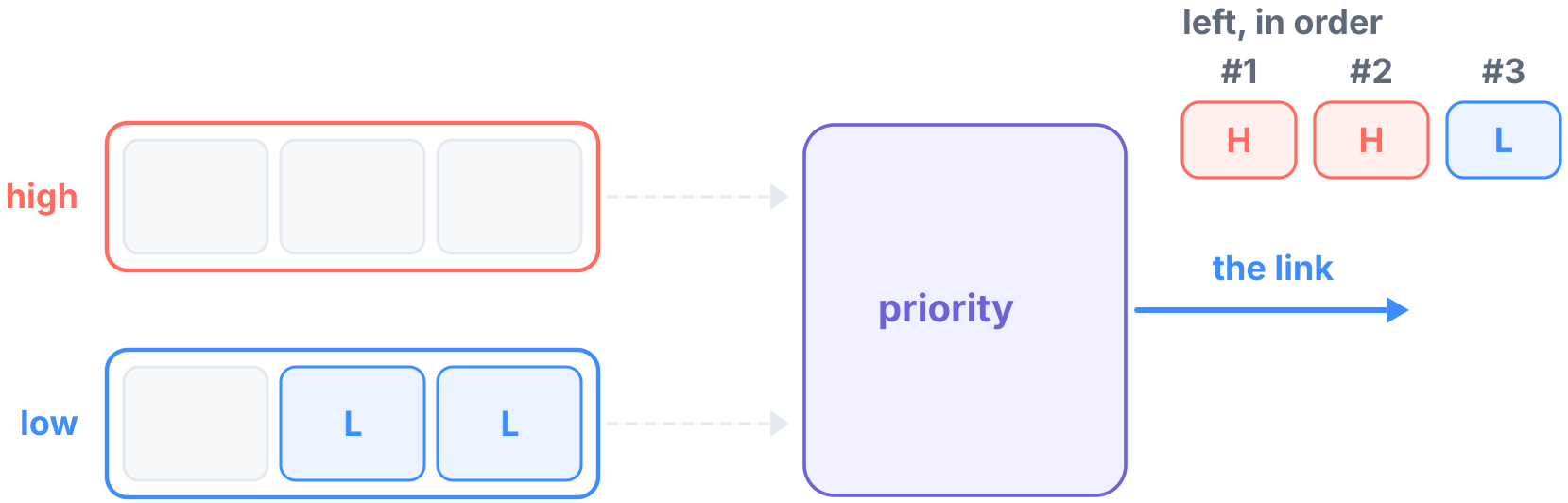
Whoever sends more gets more of the link, and everyone else waits behind them.

What has to happen before any other rule?



Classify first: one queue per class, then a rule for which queue goes next.

Priority: what is the rule?



as long as the high queue holds a packet, the low queue waits

Press → to send the next packet.

The highest non-empty class goes first, every time.

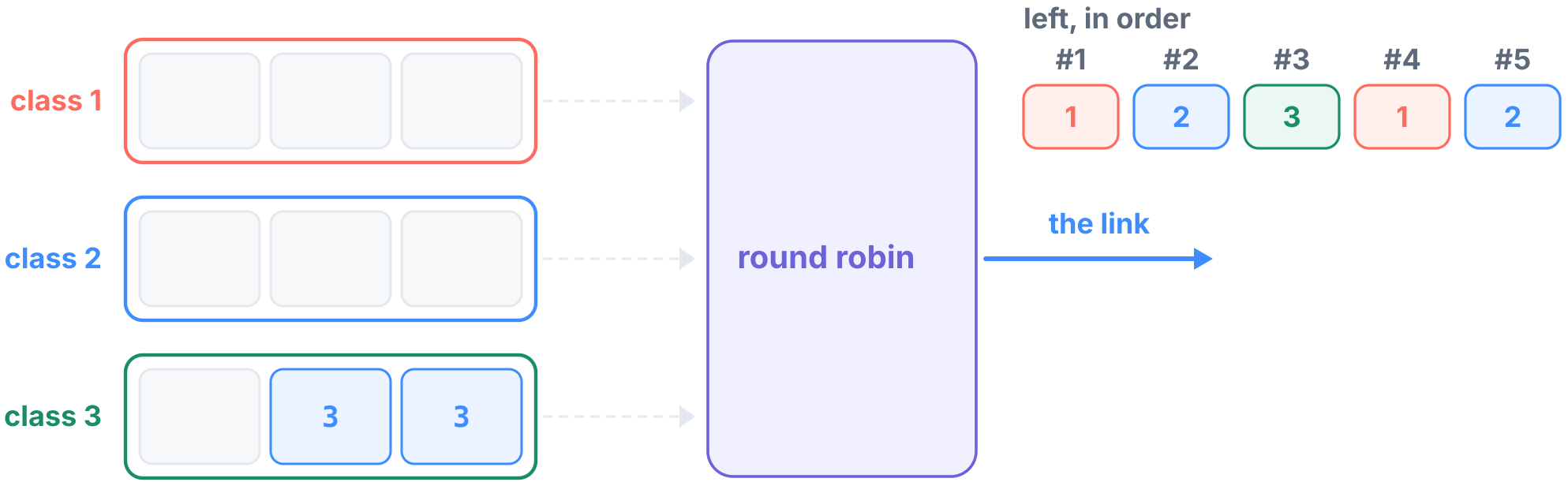
Priority: what does it cost?



the low queue only grows; nothing in the rule ever lets it go

A low class can wait forever while the high class keeps sending. Starvation.

Round robin: what is the rule?



one packet from each non-empty queue, in turn: 1, 2, 3, 1, 2, ...

Press → to send the next packet.

Each non-empty queue sends one packet in turn. Nobody starves.

Is one packet each fair?

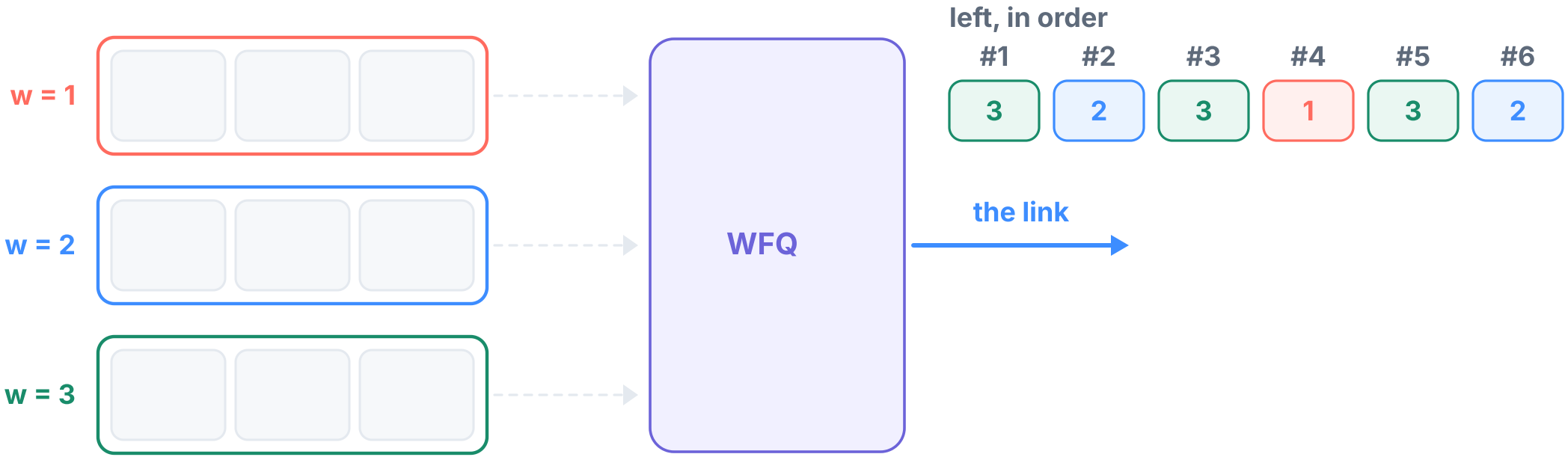


the same number of turns, fifteen times the bandwidth

Turns are not bandwidth. Large packets buy more of the link for the same turns.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

Weighted fair queueing: turns in proportion to the weights



six slots: class 3 gets three, class 2 two, class 1 one; in bytes, not packets

Press → to send the next packet.

Each class gets its weight's share of the link. Idle shares go to the others.

WFQ: weights, in bytes

class	weight	all three active	class 3 idle
class 1	w = 1	100 Mbps	200 Mbps
class 2	w = 2	200 Mbps	400 Mbps
class 3	w = 3	300 Mbps	idle

a 600 Mbps link, shared in bytes: class i gets w_i / Σw of whatever is left

a weight is a promise about bandwidth, not about turns

Weighted fair queueing: Demers, Keshav and Shenker, SIGCOMM 1989; Parekh and Gallager, IEEE/ACM ToN 1993.

Each active class gets its weight's share, in bytes. Idle shares go to the others.

Four schedulers, side by side

scheduler	the rule	can a class starve?	what it promises
FIFO	arrival order	no	nothing
priority	highest class first	yes	the high class goes first
round robin	one packet each, in turn	no	a turn
WFQ	a weighted share, in bytes	no	a share of the link

each row down expresses more, and costs more to run per packet

Each row down expresses more and costs more per packet.

What is this machinery for?

voice, games, video calls

a few kilobits a second,
and every packet must arrive soon

downloads, backups, updates

as many bits as possible,
and nobody minds a wait

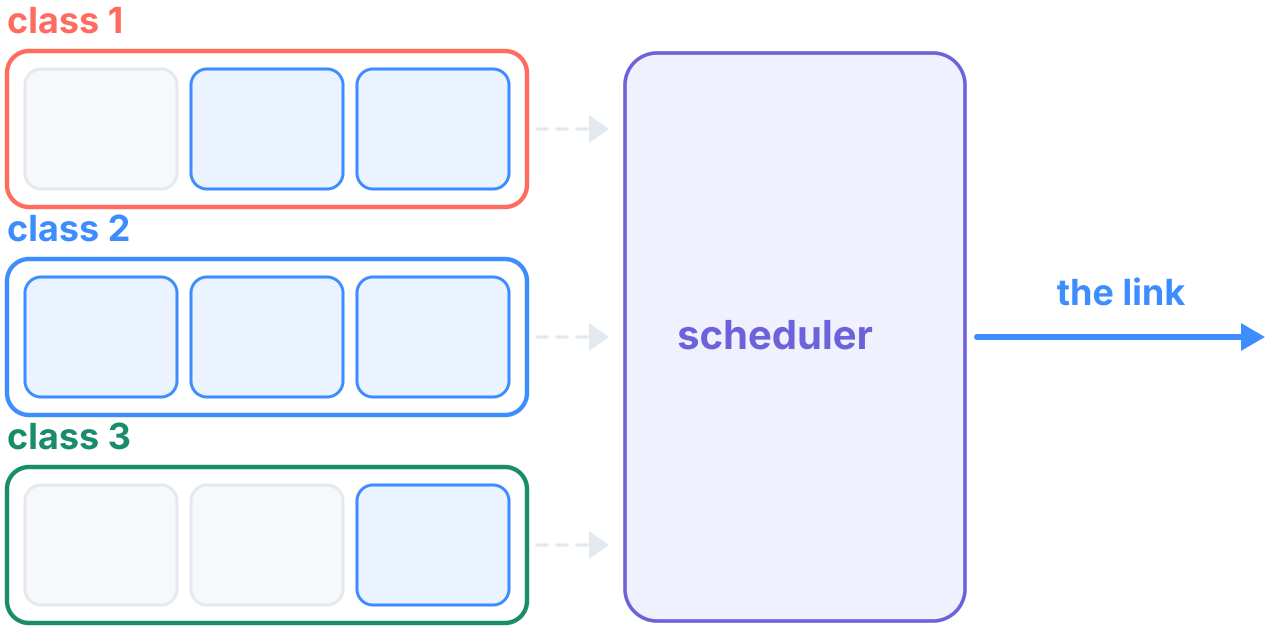
a real router: a handful of classes, a priority queue for the first kind,

weighted shares for the rest; the DSCP bits in the IP header ask for a class

Different traffic wants different things; the scheduler can give them.

1 Four parts | 2 Lookup | 3 Fabric | 4 Queues and scheduling

Net neutrality is a scheduling question



the classifier can sort

by application
voice before downloads

by source
one company's traffic before another's

by payment
a fast lane for those who pay

FIFO and fair queueing treat all packets alike; priority by payer is paid prioritization; a small weight is throttling; a weight of zero is blocking. The same box does all of them.

The mechanism sits in every router. Whether to use it is not an engineering decision.

The argument, both sides

for treating all traffic alike

no gatekeeper between users and services
new services compete on equal terms
the network does not pick winners

for letting networks differentiate

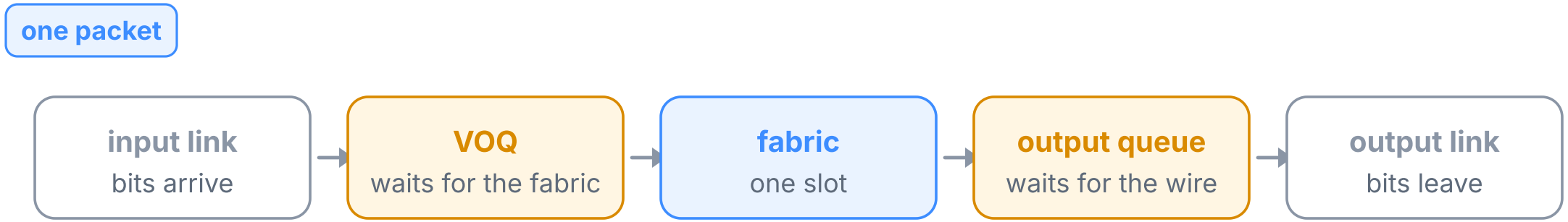
some traffic needs low delay: calls, remote care
network owners want to recover investment
managed services for those who need them

where the line is drawn differs by country: the EU has had an open-internet regulation since 2015; US rules have been adopted, repealed and re-adopted; “reasonable network management” is allowed everywhere

the engineer builds the scheduler; the regulator writes the rule it may enforce

Management or picking winners: regulators draw the line, schedulers enforce it.

Where does a packet wait inside a router?



the lookup happens before the first queue; the scheduler picks from the second

Three parts, two queues, one wire.

Four topics, one line each

01



Four parts

hardware per packet, software per route

02



The lookup

a trie walk, or all rows at once

03



The fabric

crossbar, VOQs, cells, planes; one wire out

04

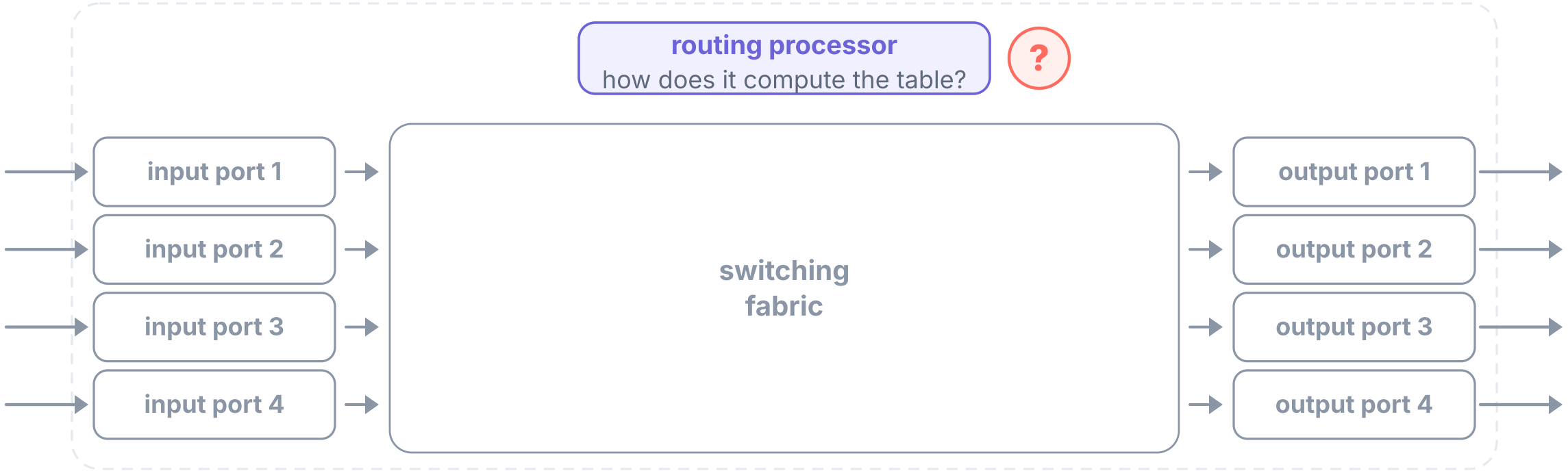


Queues, scheduling

the buffer, then FIFO, priority, round robin, WFQ

The box is open. What is left is where the routing processor gets its table.

What is left



This lecture copied a finished table into the ports. How routers compute it is Chapter 5.