MPLS VPN

l.cittadini, m.cola, g.di battista

motivations

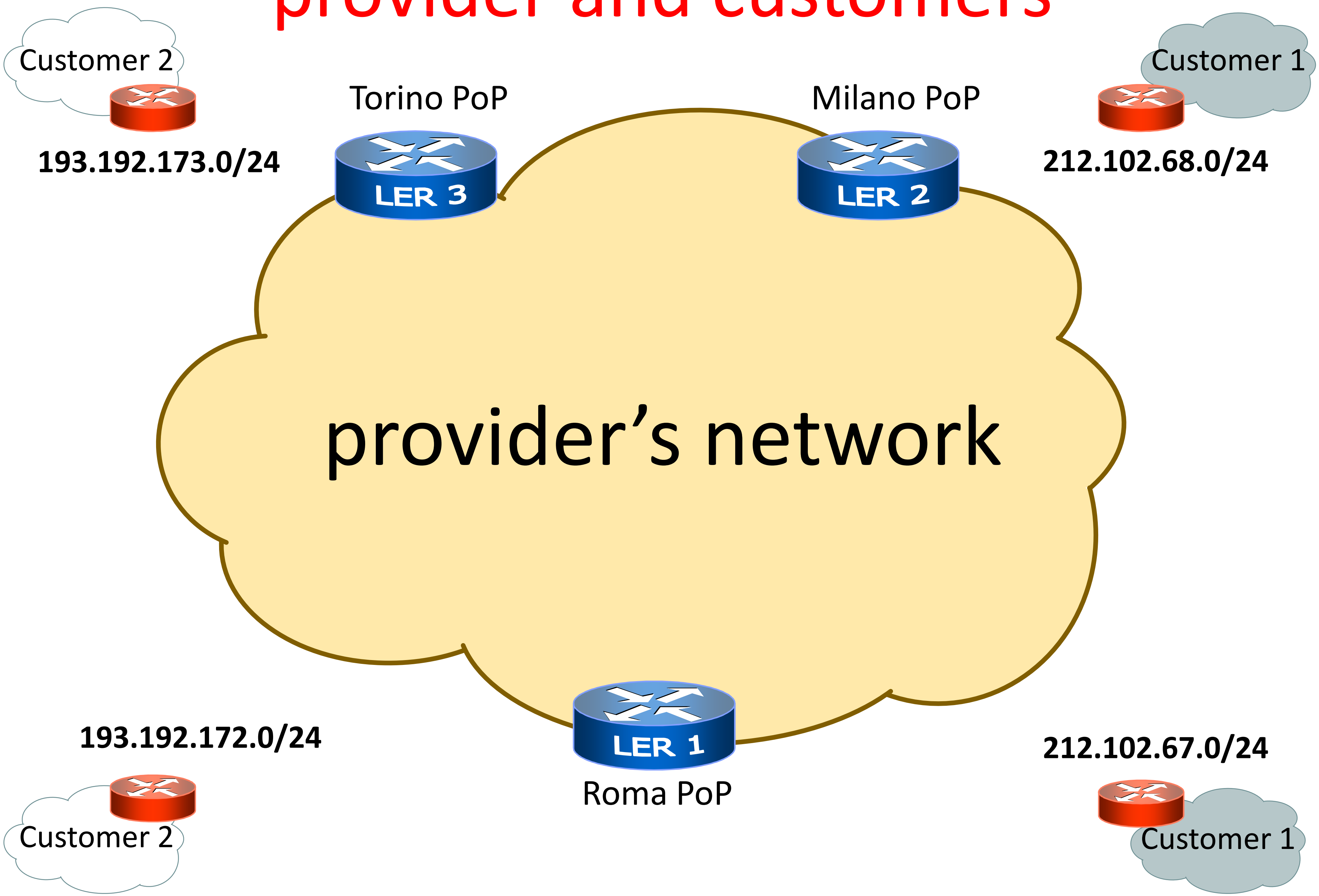
customer’s problem

- a customer (e.g., private company, public administration, etc.) has several geographically distributed sites and would like to connect them into a unique IP network
  - ideally it would like to have “wires” connecting its sites

provider's target

- a provider owns a network infrastructure with many distributed PoPs (Points of Presence) and would like to exploit it to offer IP level connectivity services to its customers
  - it would like to sell virtual wires (where IP packets can flow) to its customers

provider and customers



customer’s constraints

- keep the addressing unchanged
- isolation from different customer's traffic
- quality of service

provider’s constraints

- low configuration and maintenance costs
- no performance penalties
  - performance in the backbone should only depend on traffic, not on the number of supported VPNs or on the number of supported sites

vendor's targets

- sell many routers
  - possibly expensive carrier-grade machines
- move the focus from old (and already over-sold) ATM & ATM-like technologies to new (and with a growing market) technologies

meeting point

between customers, providers, and vendors

- VPN – Virtual Private Network: behaves like a physical private network, but it's virtual

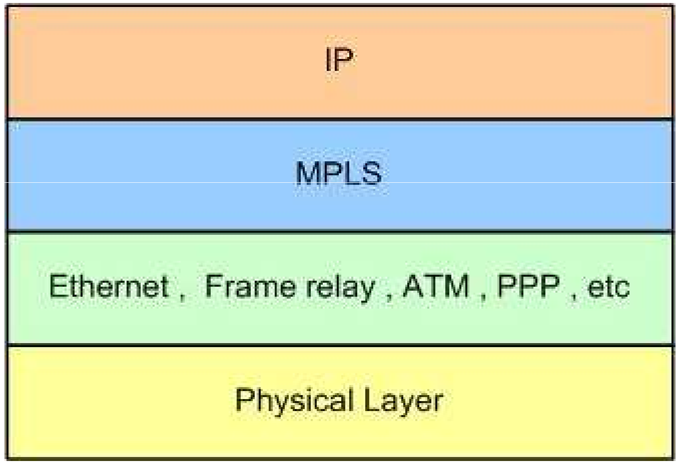
implemented with

- MPLS – Multi Protocol Label Switching (swapping): highly scalable, protocol-agnostic, data-carrying mechanism

MPLS

MPLS in a nutshell 

- MPLS vs OSI

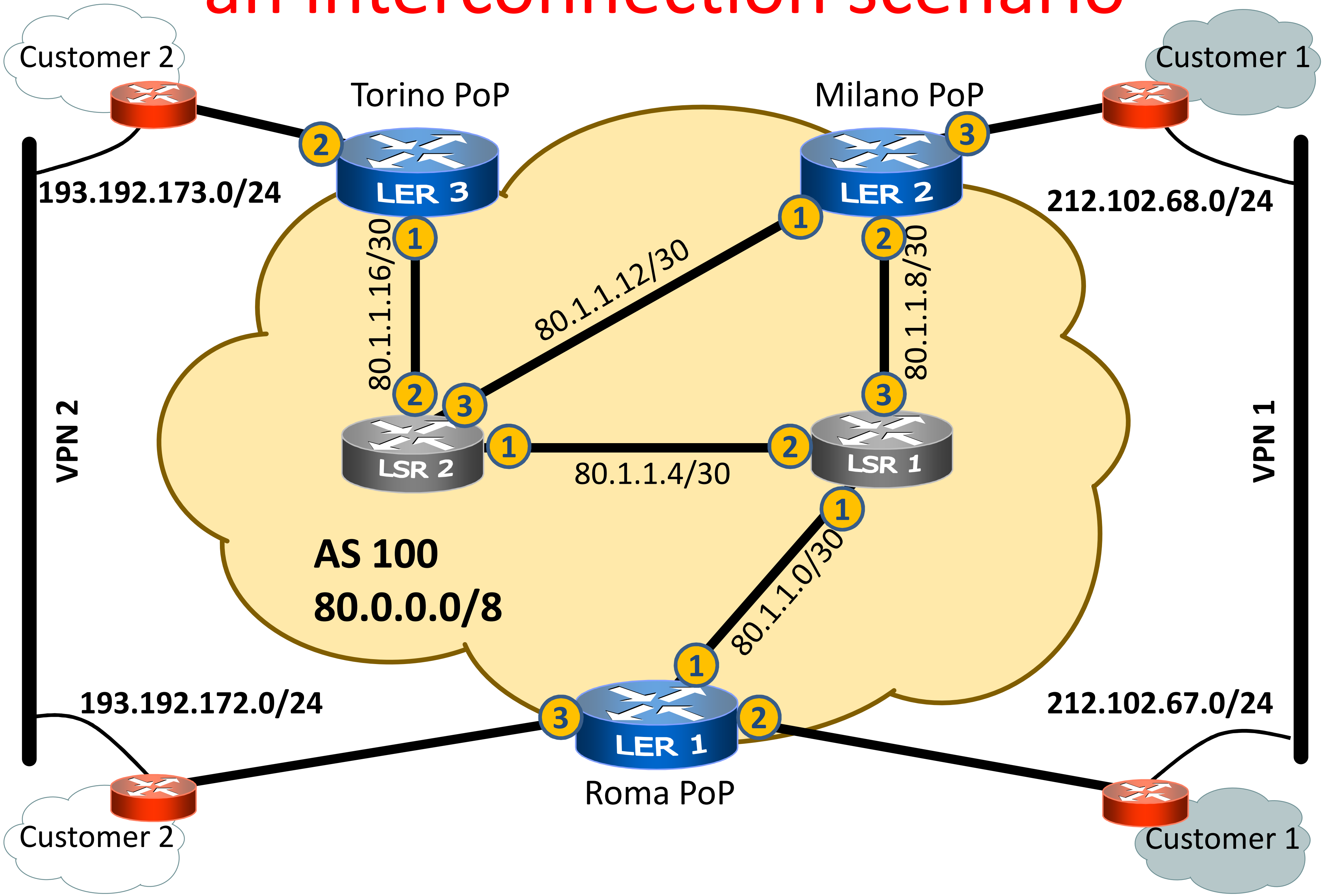


picture from wikipedia

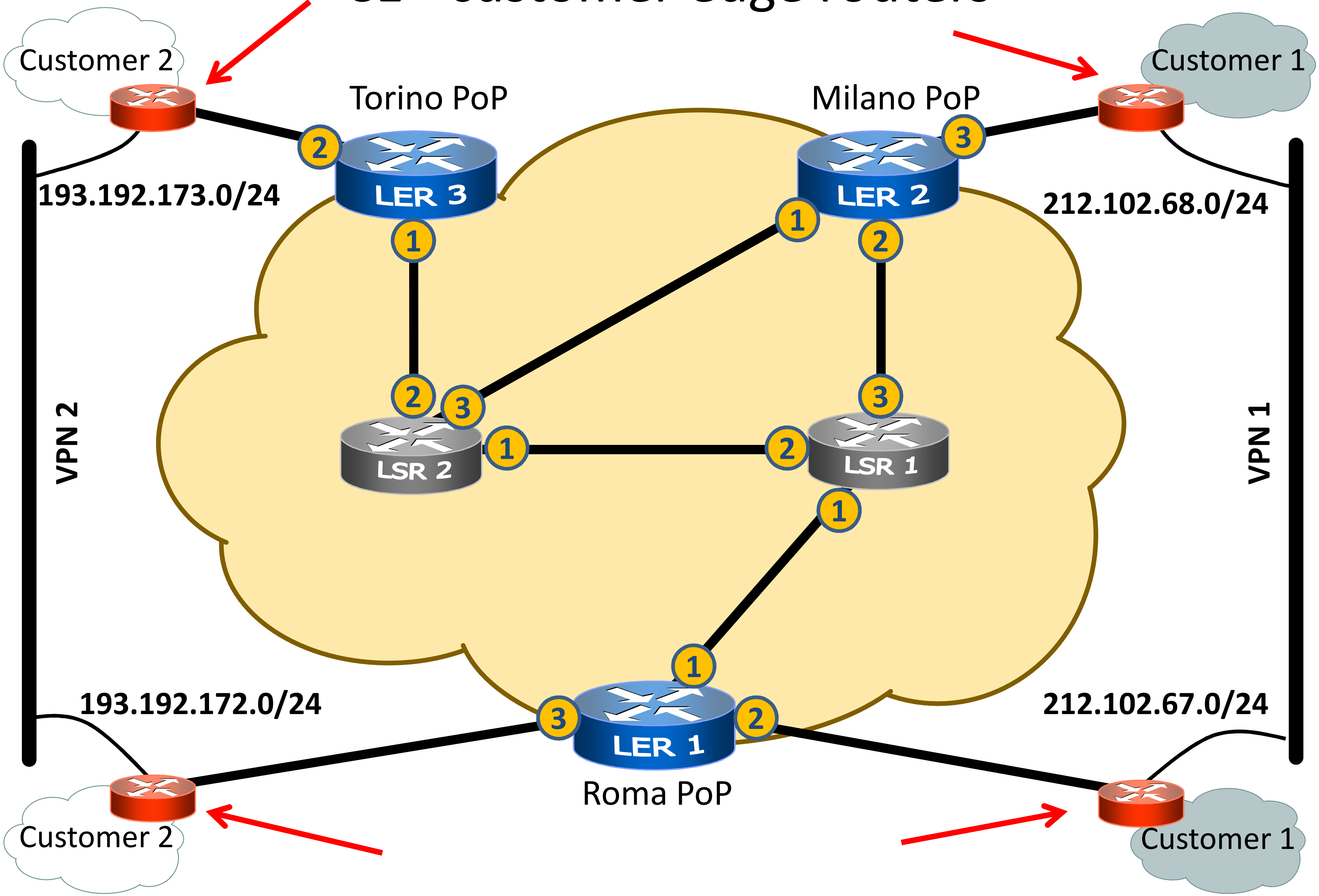
MPLS in a nutshell

- MPLS packet
  - encapsulates transported packets with an MPLS header, containing one or more labels (label stack)
  - each label stack entry contains 4 fields:
    - label value (20 bits)
    - traffic class field for QoS and ECN - Explicit Congestion Notification (3 bits)
    - bottom of stack flag (1 bit)
    - ttl (8 bits)

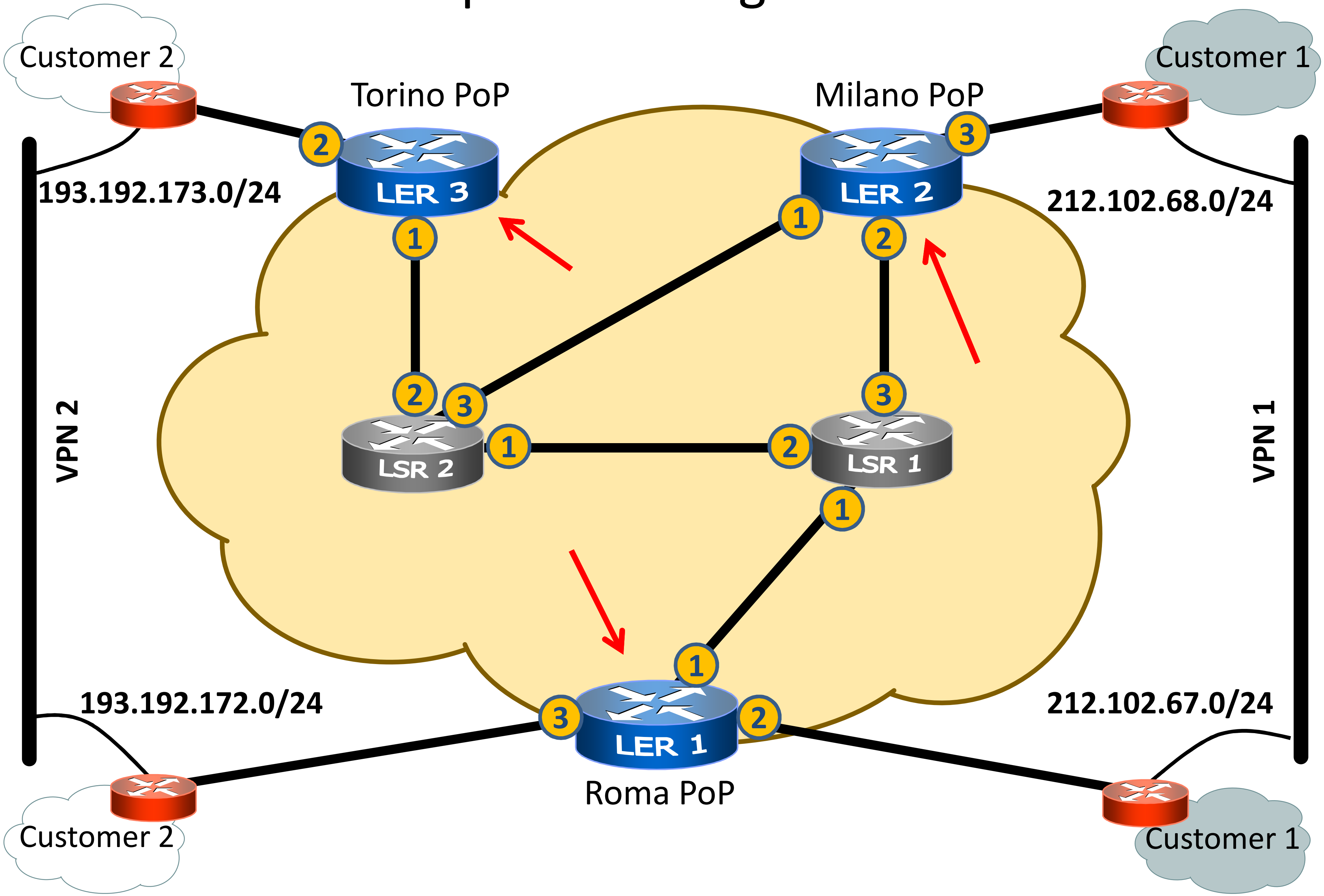
an interconnection scenario



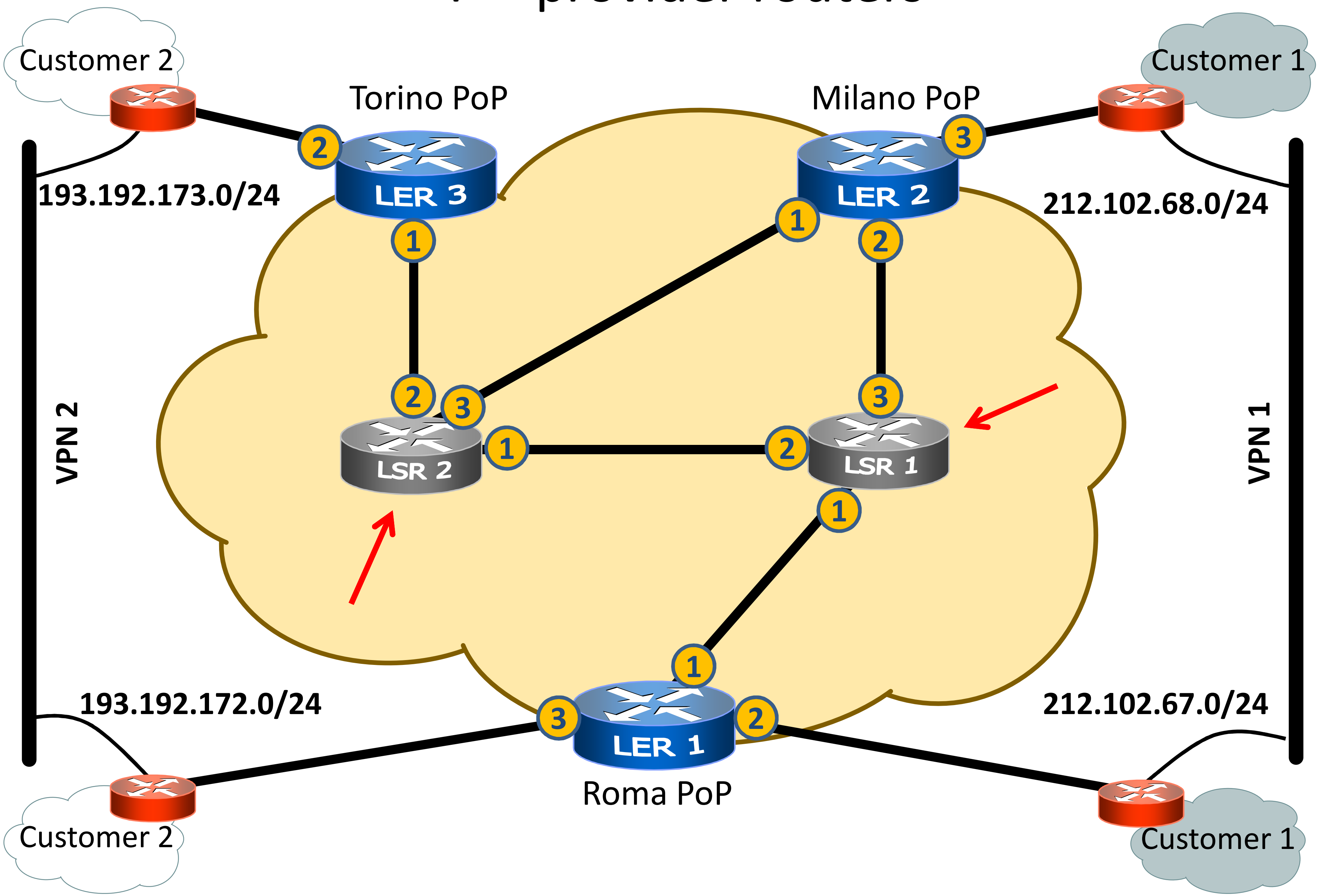
CE - customer edge routers



PE – provider edge routers



P – provider routers



how to 

checkmate VPNs in three moves:

(1) make PEs reachable each other using IP,

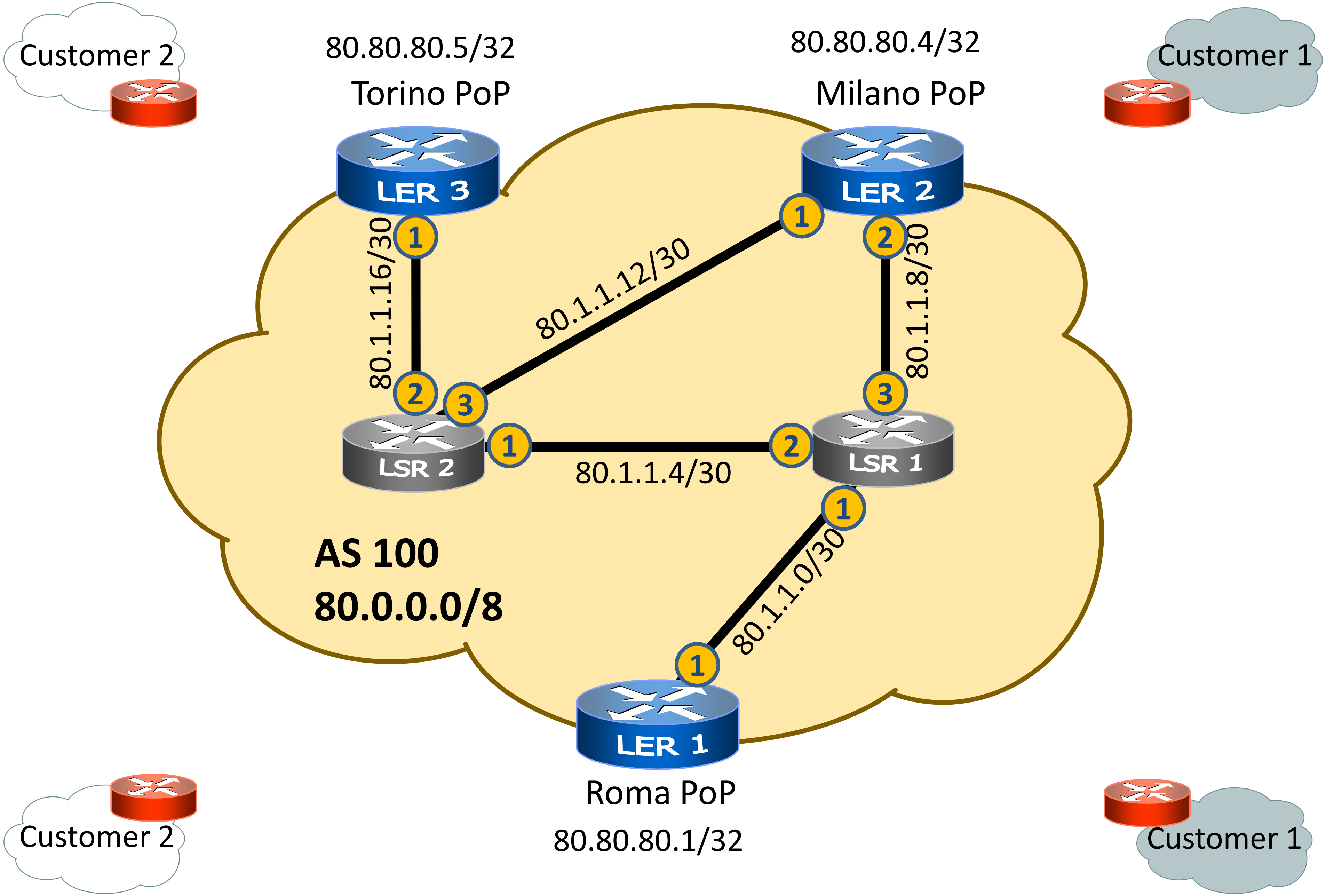
(2) use BGP for announcing customer prefixes, and

(3) use MPLS for tunnels inside the backbone

1st move – IP reachability of the PE’s

- the first thing to do is to assign an IP loopback address to each PE and to ensure IP reachability among PEs in the backbone
  - ignore all the other issues
  - inside backbone use addresses that are not announced outside (e.g. private addresses)
- loopback addresses are propagated by an IGP
  - OSPF, IS-IS, ….

loopbacks of PEs



a tempting solution

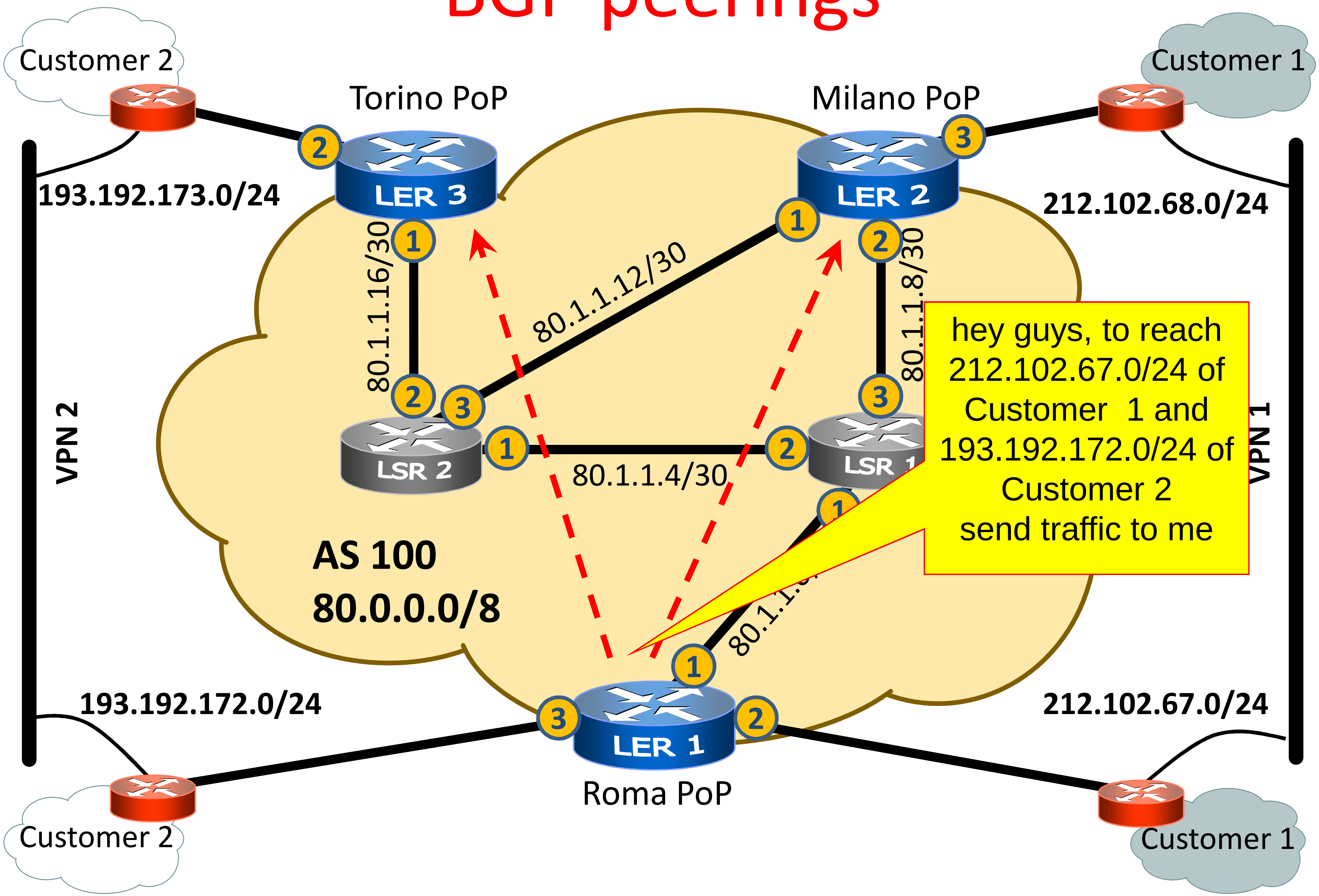
- implement VPNs using IP-in-IP tunnels (or similar technologies) between PEs' loopbacks
- drawbacks
  - difficult to configure
  - quadratic number of configurations

hence, this solution is discarded

2nd move – use BGP to announce customer prefixes

- MP-BGP, a variation of BGP, is used
- each PE establishes an iBGP peering with all other PEs
  - usage of route reflectors for scaling
- customer's networks are announced within the peerings

BGP peerings



3rd move – use MPLS for tunnels

- an IP packet of a customer, coming from a CE, is encapsulated from the PE near to the source into an MPLS packet
- the PE near to the source sends the packet to the PE (loopback) near to the destination
- the IP packet traverses the backbone into an MPLS envelope

MPLS labels and VPN's

- two labels are used
  - the internal one denotes the VPN: it is used in a way similar to the VLAN tags of IEEE 802.1Q
    - remains unchanged for the entire travel of the packet from origin PE to destination PE
  - the external one is used for label swapping
    - is, in general, changed at each hop of the travel from PE to PE
    - a first pop from the stack is done at the penultimate router

how to find a route to loopbacks?

- who constructs the label-swapping data plane of MPLS?
- in other words: in which way a P or a PE knows which is the correct path to the target PE (loopback)?

LDP – Label Distribution Protocol

is in charge of this

LDP

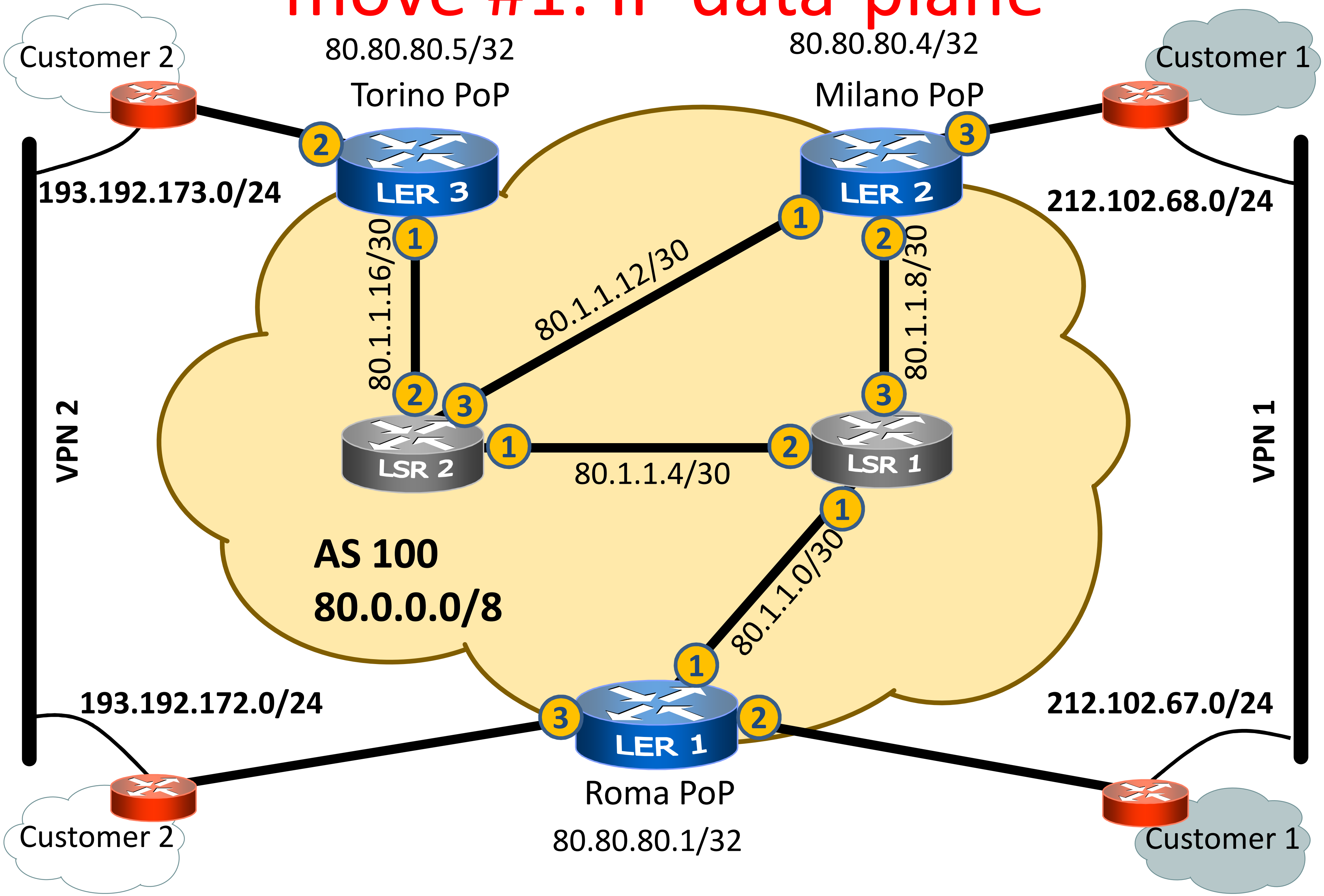
- LDP constructs the label switched paths (LSPs) to reach each PE loopback by simply importing this information from the IP data plane
  – remeber 1st move: the IP data plane knows how to reach loopbacks – it has been setup by an IGP

terminology and details

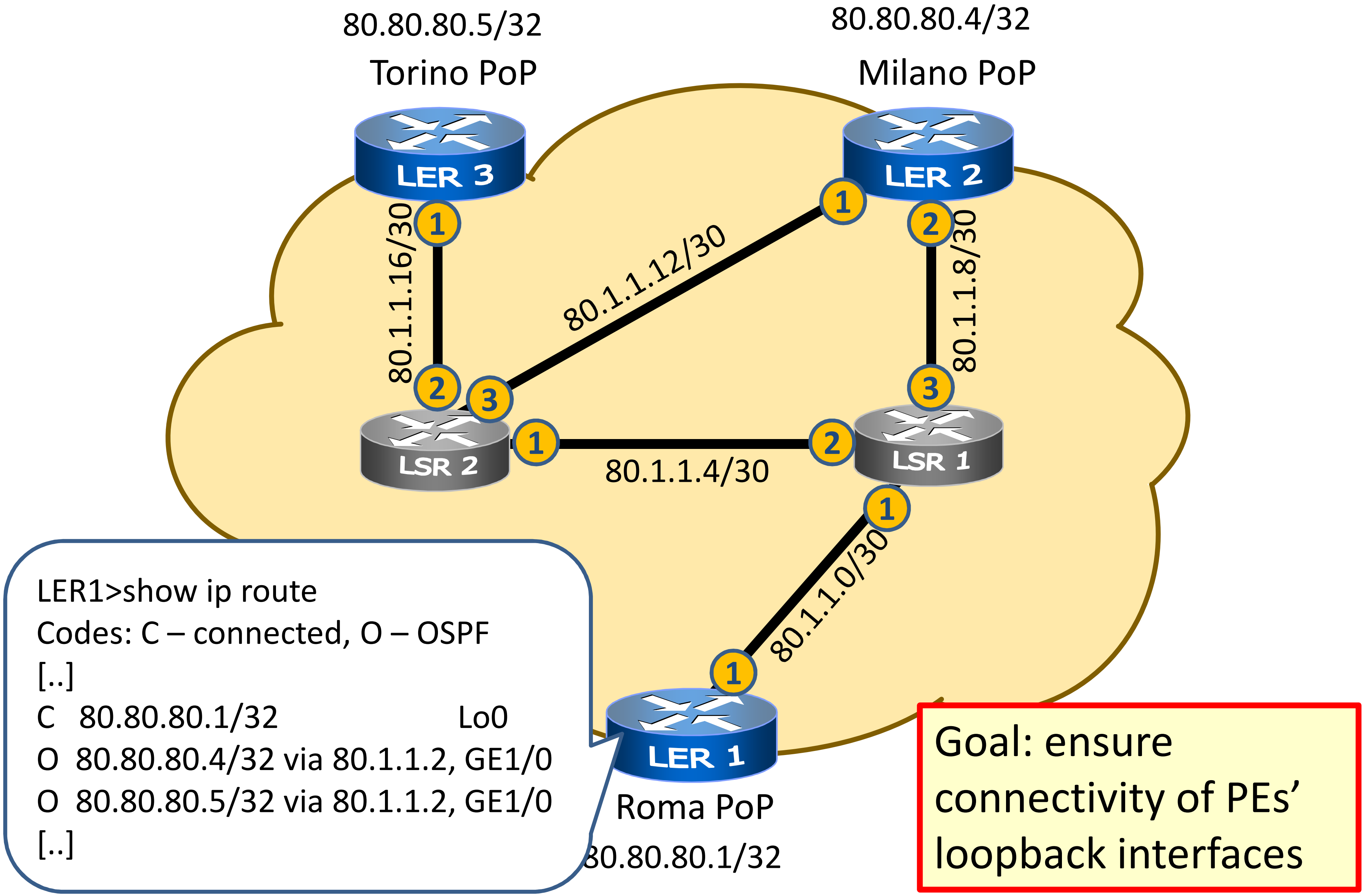
data- and control-planes

- to understand how MPLS operates, let’s review the mate-in-3
- 1st move: IP data-plane
  - build IP routing tables (OSPF, IS-IS, static…)
  - ensure reachability of PEs’ loopback interfaces
- 2nd move: BGP control-plane
  - ensure that VPN prefixes are distributed among PEs
- 3rd move: MPLS data-plane
  - build label switching tables (LFIBs) with LDP
  - ensure availability of LSPs (Label Switched Paths) that connect each pair of PEs

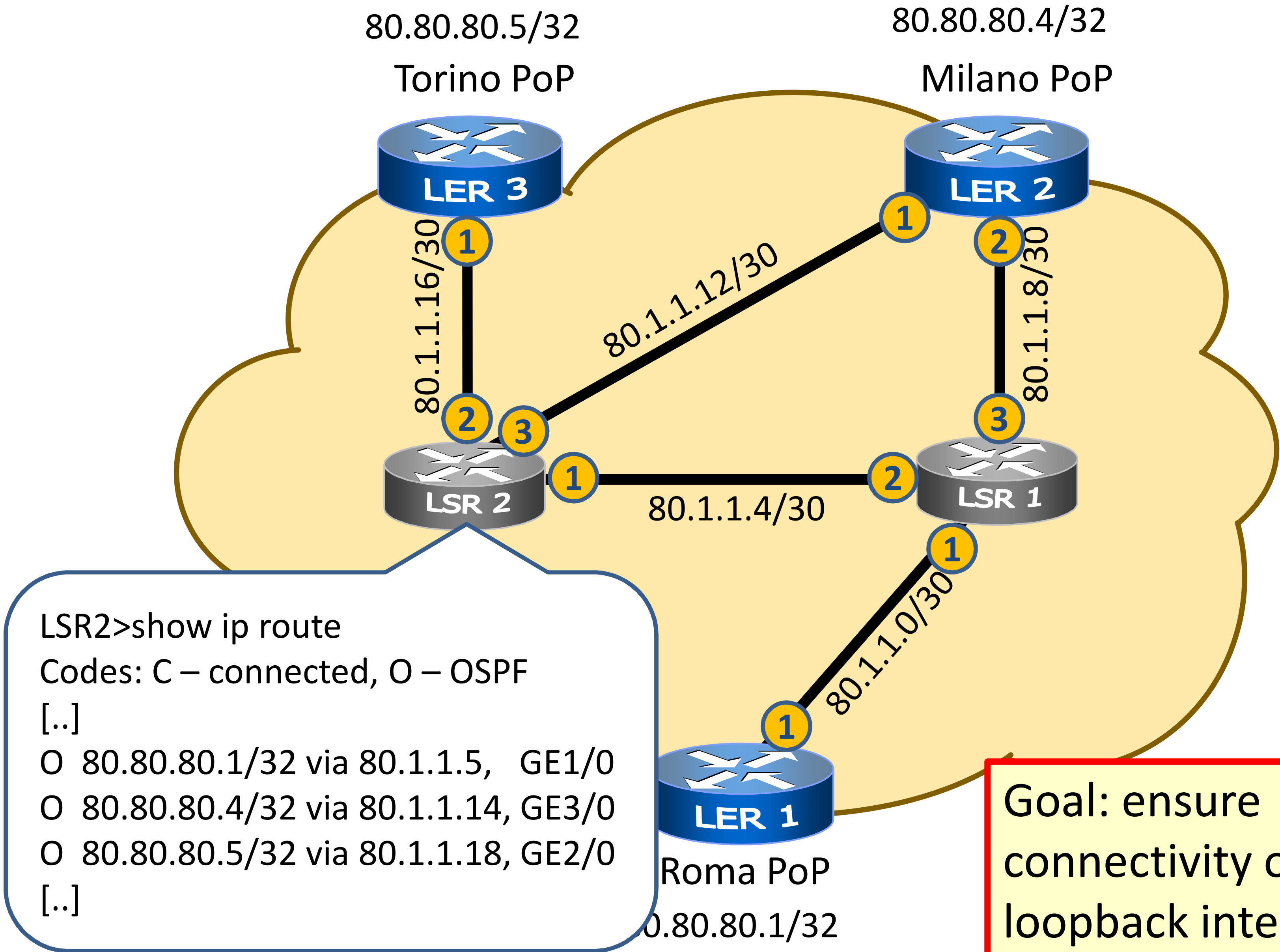
move #1: IP data-plane



move #1: IP data-plane

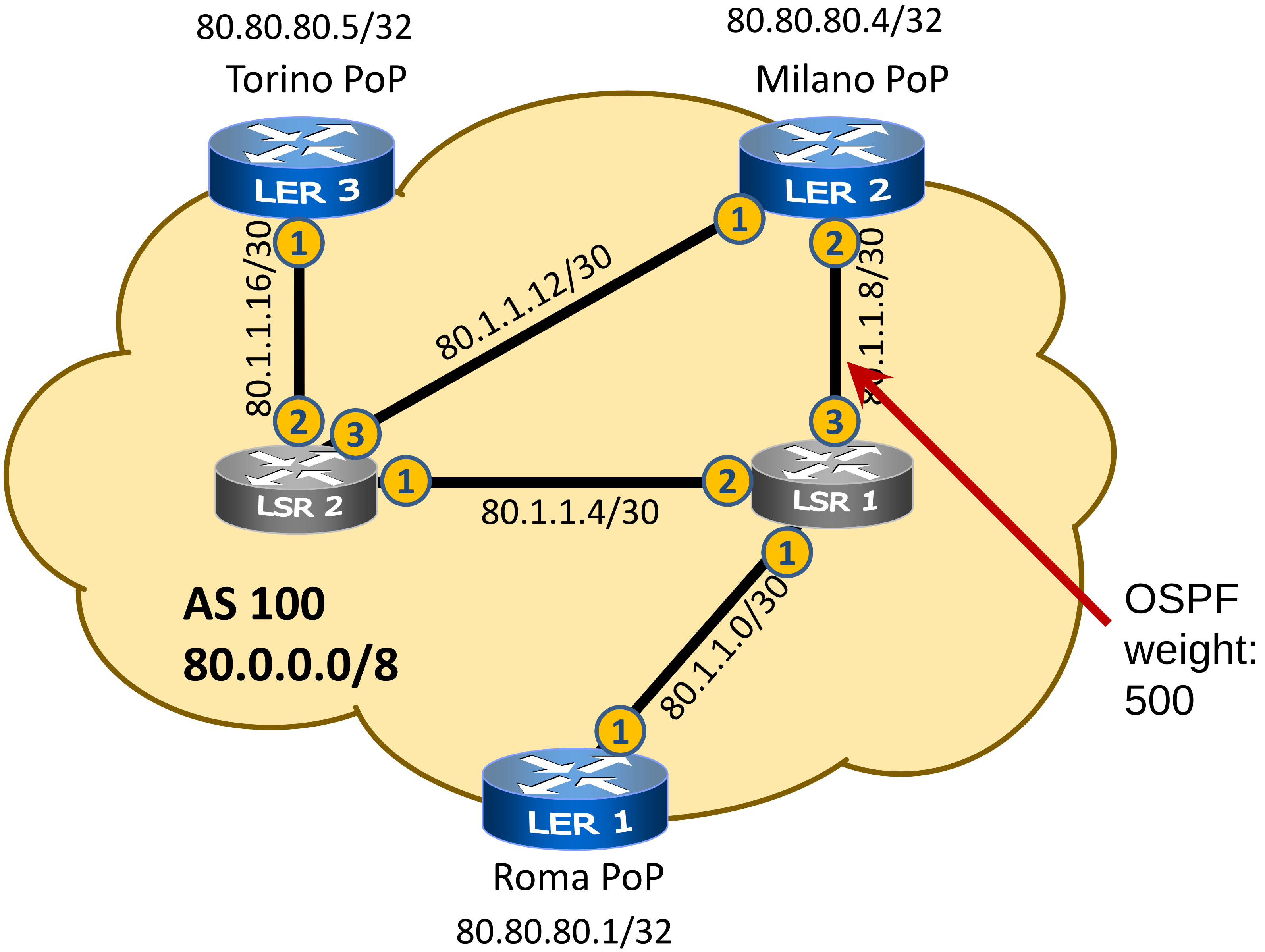


move #1: IP data-plane



Goal: ensure connectivity of PEs' loopback interfaces

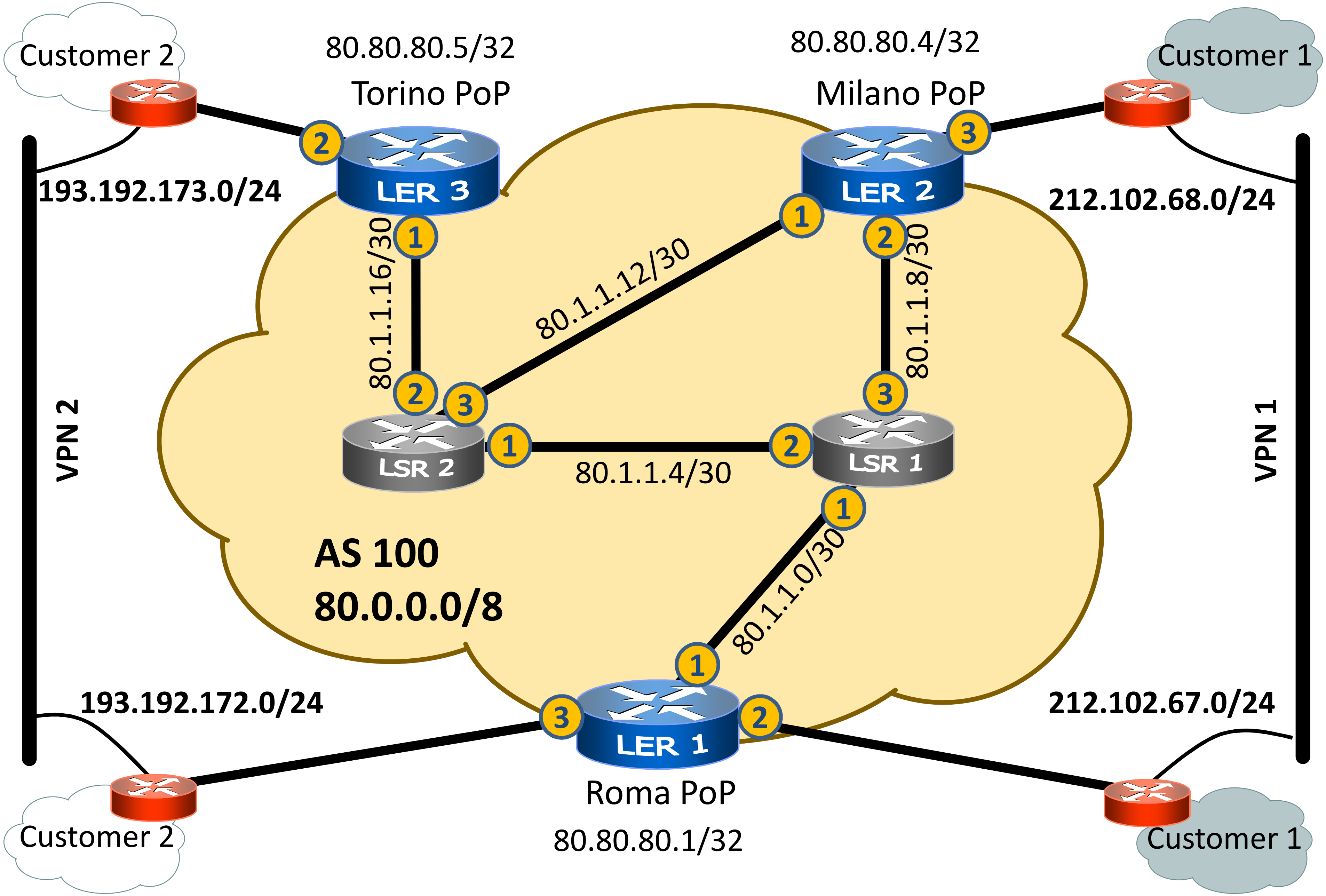
move #1: IP data-plane



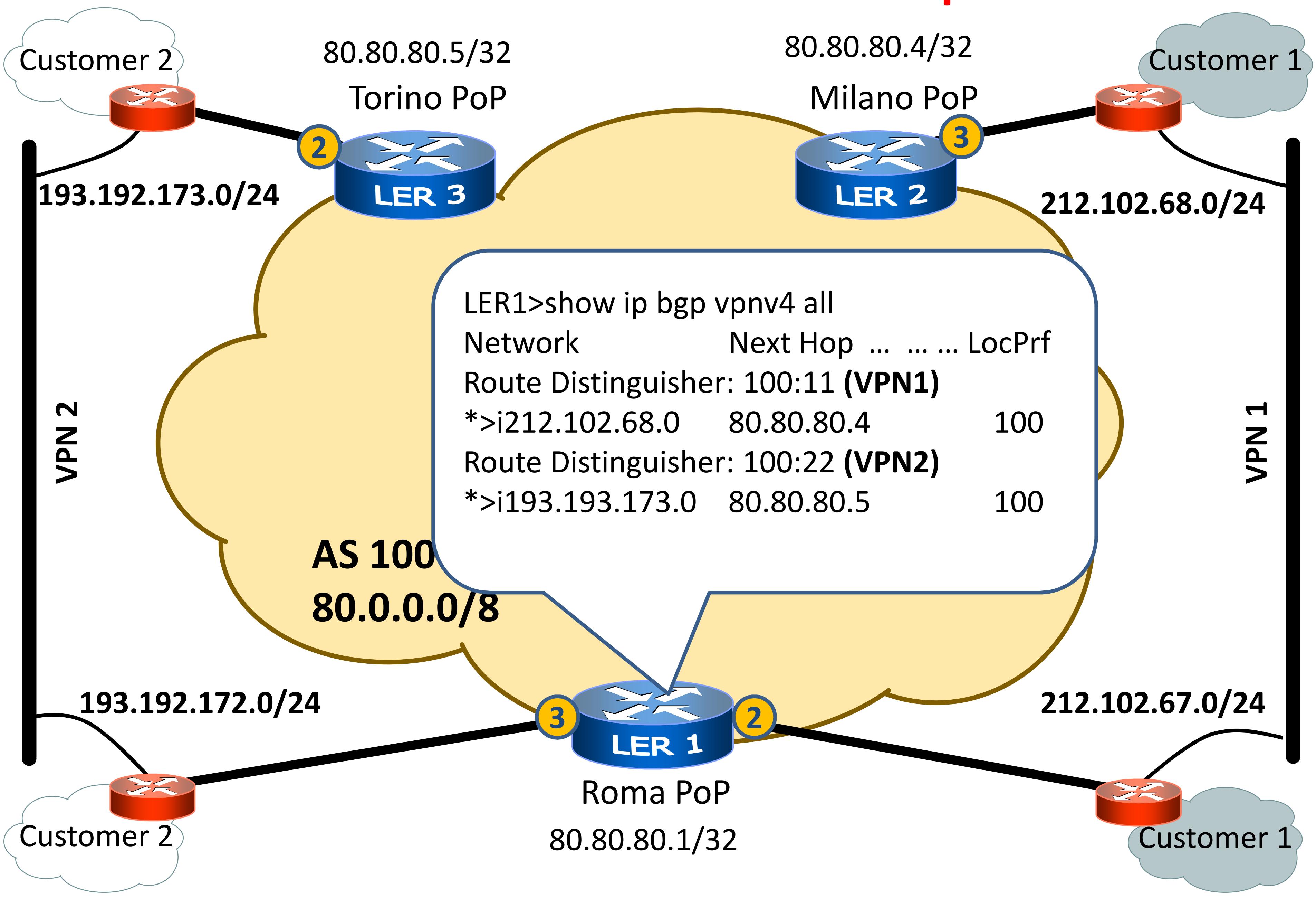
IP data-plane: observations

- IP data-plane must ensure reachability of PEs' loopback interfaces
  - other prefixes (e.g., point-to-point links) are useless for MPLS
    - but they can be used for other purposes, e.g., network management (telnet, ssh, SNMP, …)
- any IGP (OSPF, IS-IS) can do the job
- static routes can do the job too, but they do not handle network dynamics
  - e.g., link failures

move #2: BGP control-plane



move #2: BGP control-plane



multiple routing tables on PEs

- PEs need to distinguish each VPN
  - there might be overlapping address space!
- solution: multiple (virtual) routing tables
- each customer port on PE is associated with a particular routing table
  - at provisioning time
- ports on PE could be logical
  - e.g. VLANs

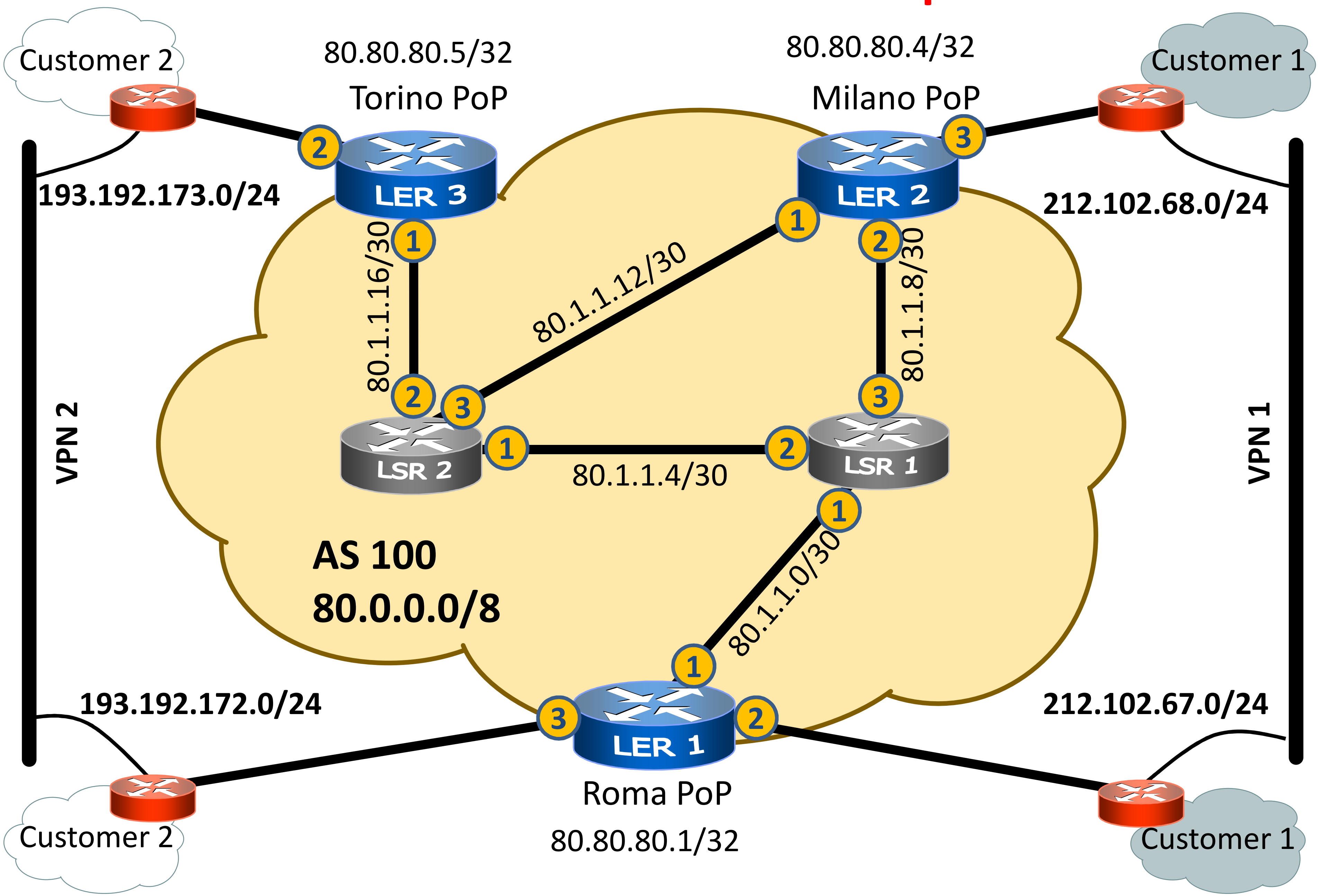
Virtual Routing and Forwarding (VRF)

- VRF - Virtual Routing and Forwarding
  - allows a router to have multiple forwarding tables
  - each table is called a VRF instance
- each PE maintains multiple VRF instances
  - one per set of directly attached sites with common VPN membership
- each VRF instance contains:
  - routes received from directly connected CE's of the sites associated with the VRF instance
  - routes received from other PEs (via BGP)

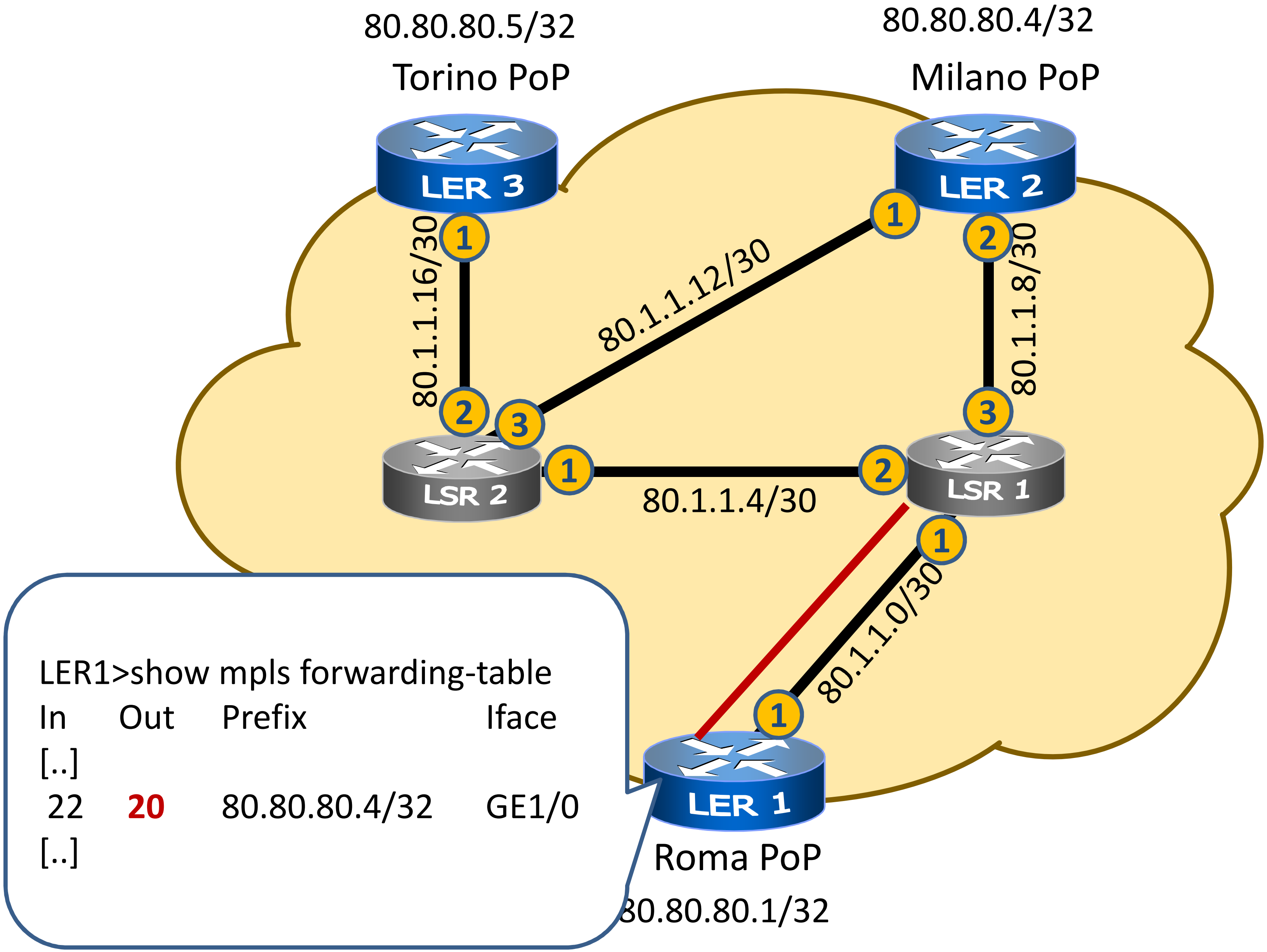
VPN-IP addresses

- VPN-IP address = Route Distinguisher (RD) + IP address
  - RD = Type + Provider's Autonomous System Number + Assigned Number
    - **no two VPNs have the same RD**
- convert non-unique IP addresses into unique VPN-IP addresses
- avoids conflicts if customers have overlapping address spaces

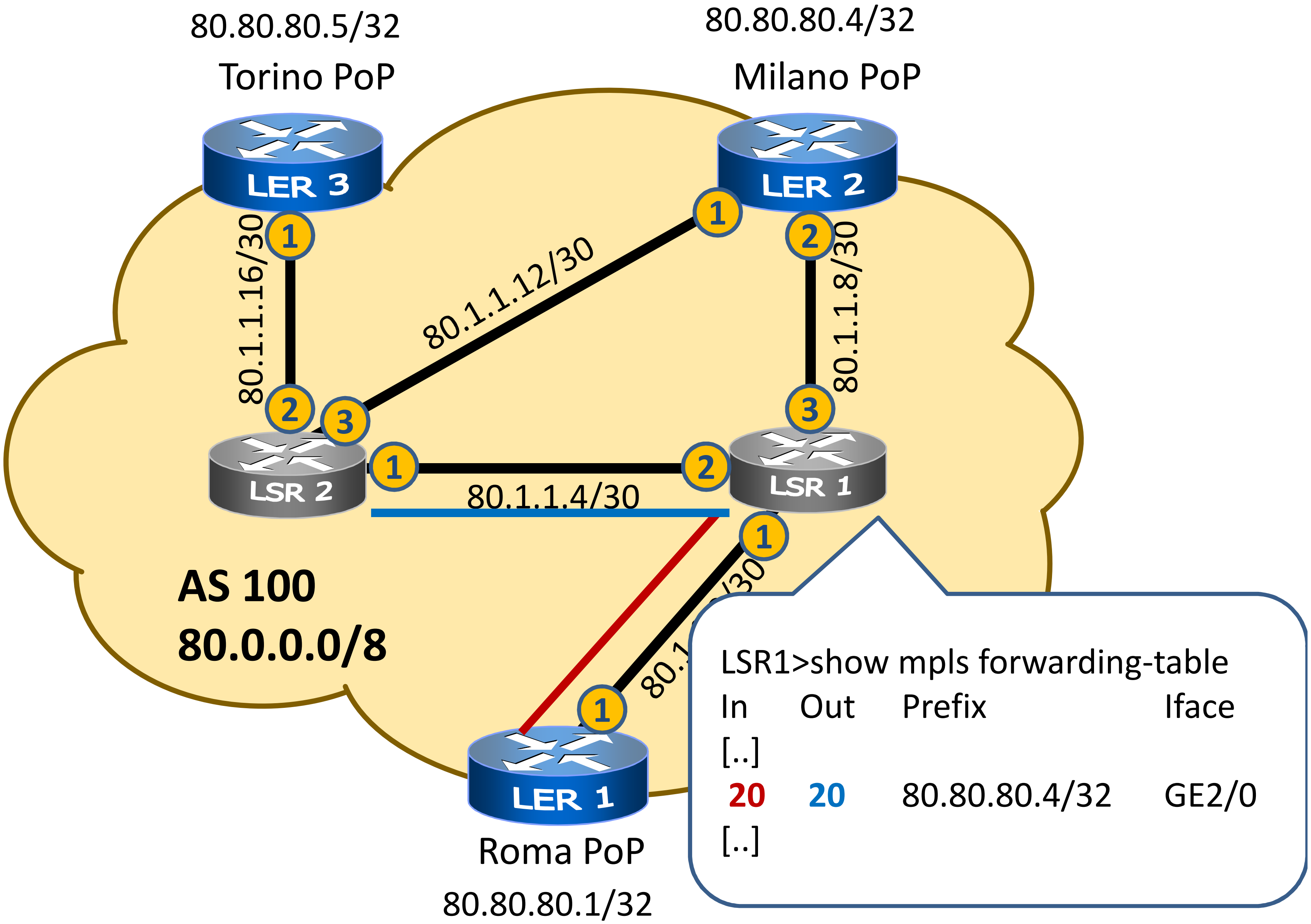
move #3: MPLS data-plane



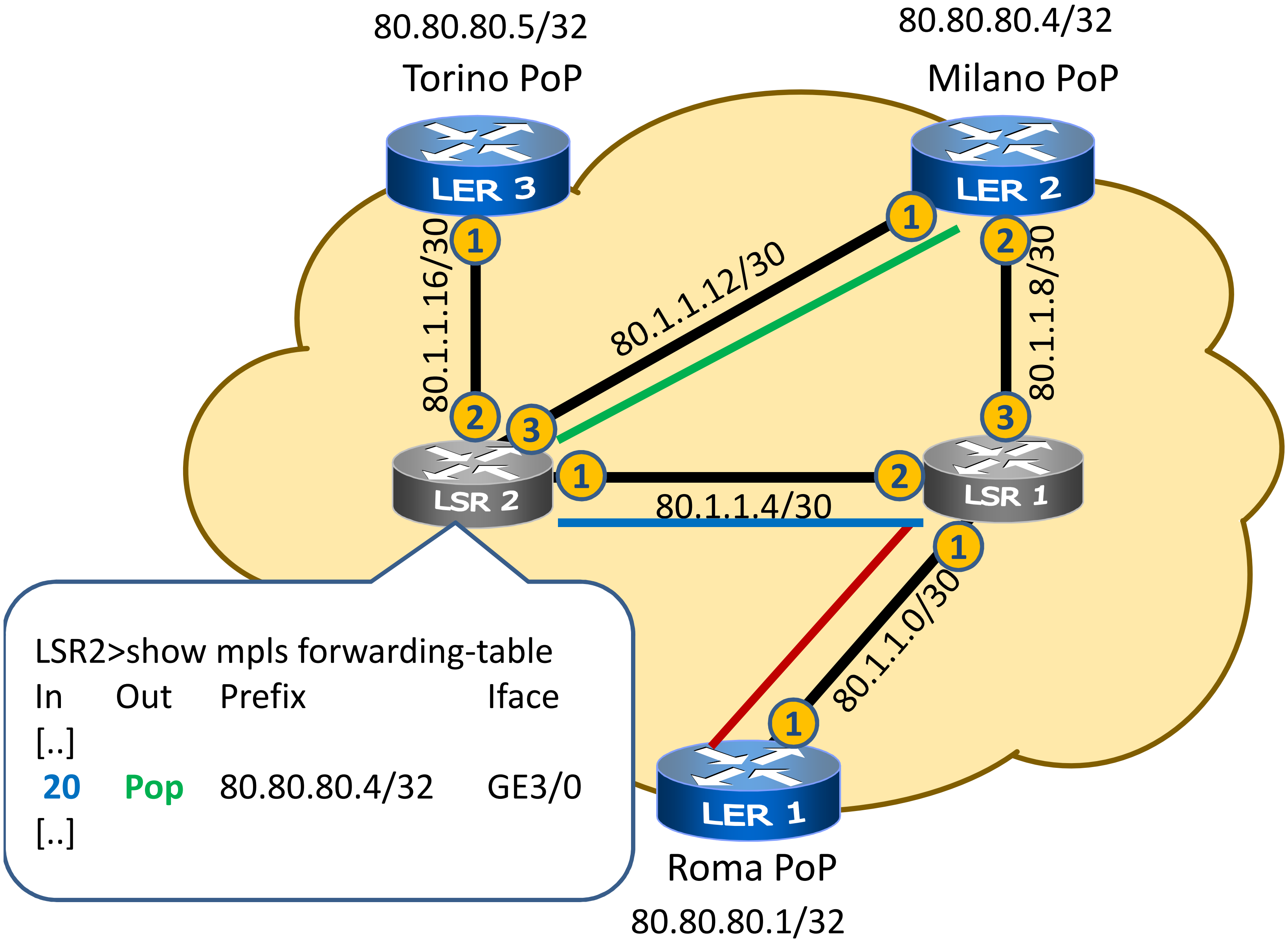
move #3: MPLS data-plane



move #3: MPLS data-plane



move #3: MPLS data-plane



label spaces

- a LSR can receive the same incoming label from multiple incoming interfaces
- what can happen in that case?
  - forwarding is based only on the label
  - labels are unique router-wide
  - (out_iface, out_label) = *f*(in_label)

    per-platform label space
  - forwarding is based on the label and the interface which received the packet
  - labels may be not unique router-wide
  - (out_iface, out_label) = *f*(in_iface, in_label)

    per-interface label space

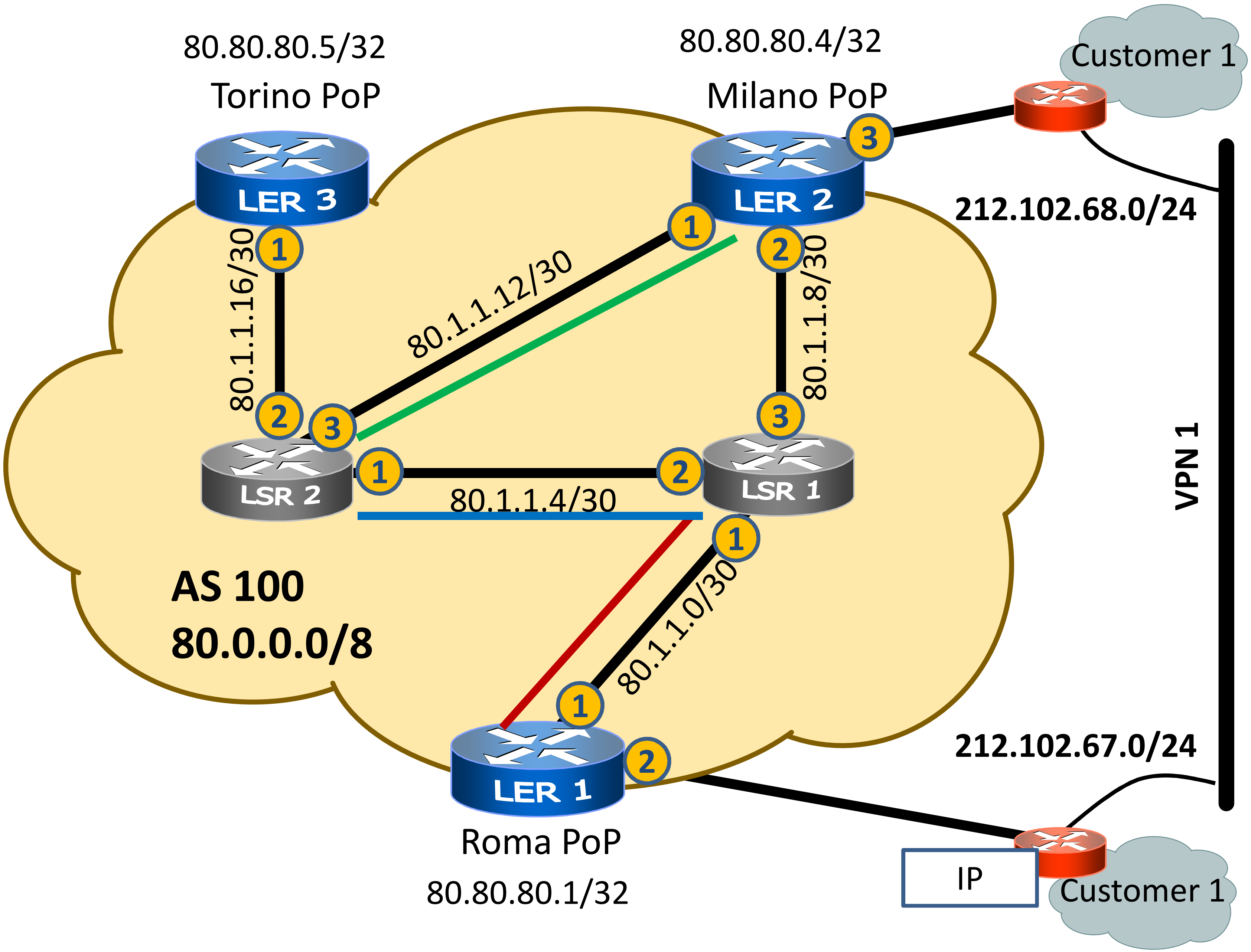
label spaces: which one?

- which label space is used?
  - it depends on the implementation
  - sometimes it also depends on the interface
  - **not** configurable
- example 1: Cisco IOS
  - LC-ATM interfaces use per-interface label space
  - other interfaces use per-platform label space
- example 2: JunOS
  - AAL5 ATM interfaces use per-interface label space
  - other interfaces use per-platform label space

putting it all together

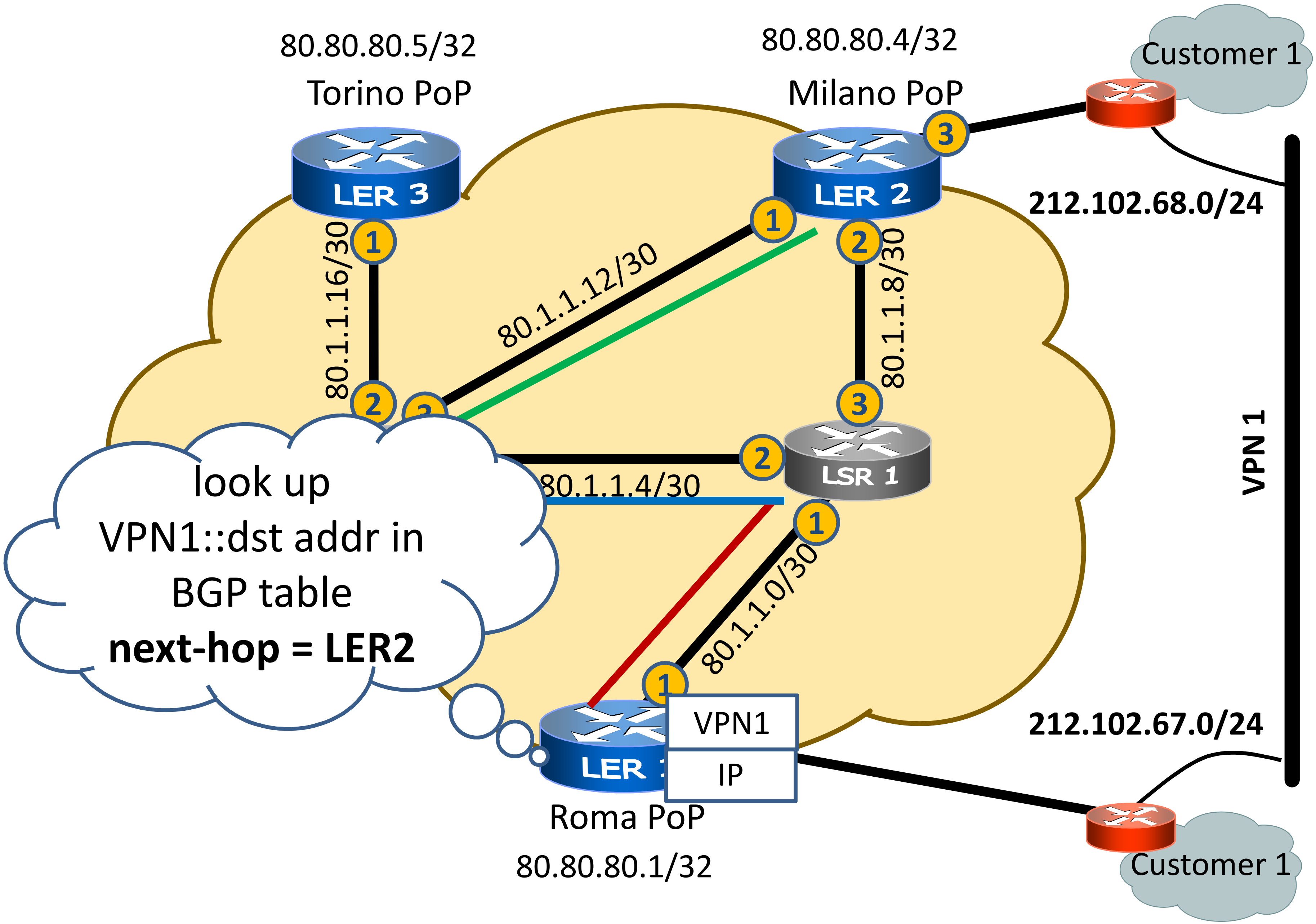
- a host of customer1 in Rome sends a packet, destined to 212.102.68.2 (located in Milan)
- LER1 receives the packet from CE
  - adds a MPLS label to mark it as belonging to VPN1
- LER1 looks in its BGP table for VPN1
  - finds next-hop 80.80.80.4 (LER2's loopback)
- LER1 looks in its LFIB for 80.80.80.4
  - finds label 20, interface GE1/0
- LER1 adds another MPLS label (20) and forwards the packet on GE1/0

Example



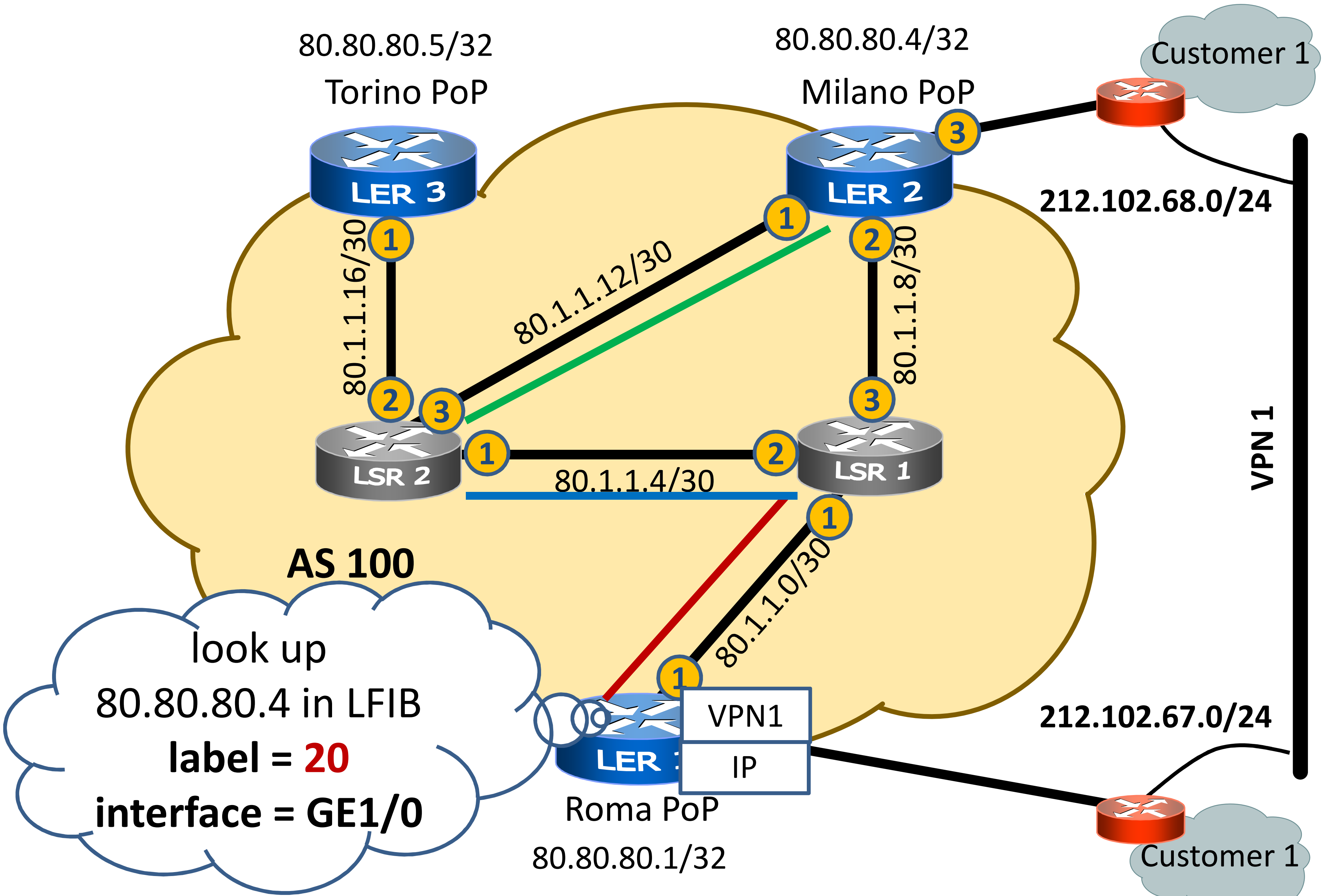
80.80.80.5/32
Torino PoP
80.80.80.4/32
Milano PoP
Customer 1
LER 3
LER 2
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
LSR 2
LSR 1
80.1.1.4/30
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
212.102.67.0/24
LER 1
Roma PoP
80.80.80.1/32
IP
Customer 1
1
2
3

Example



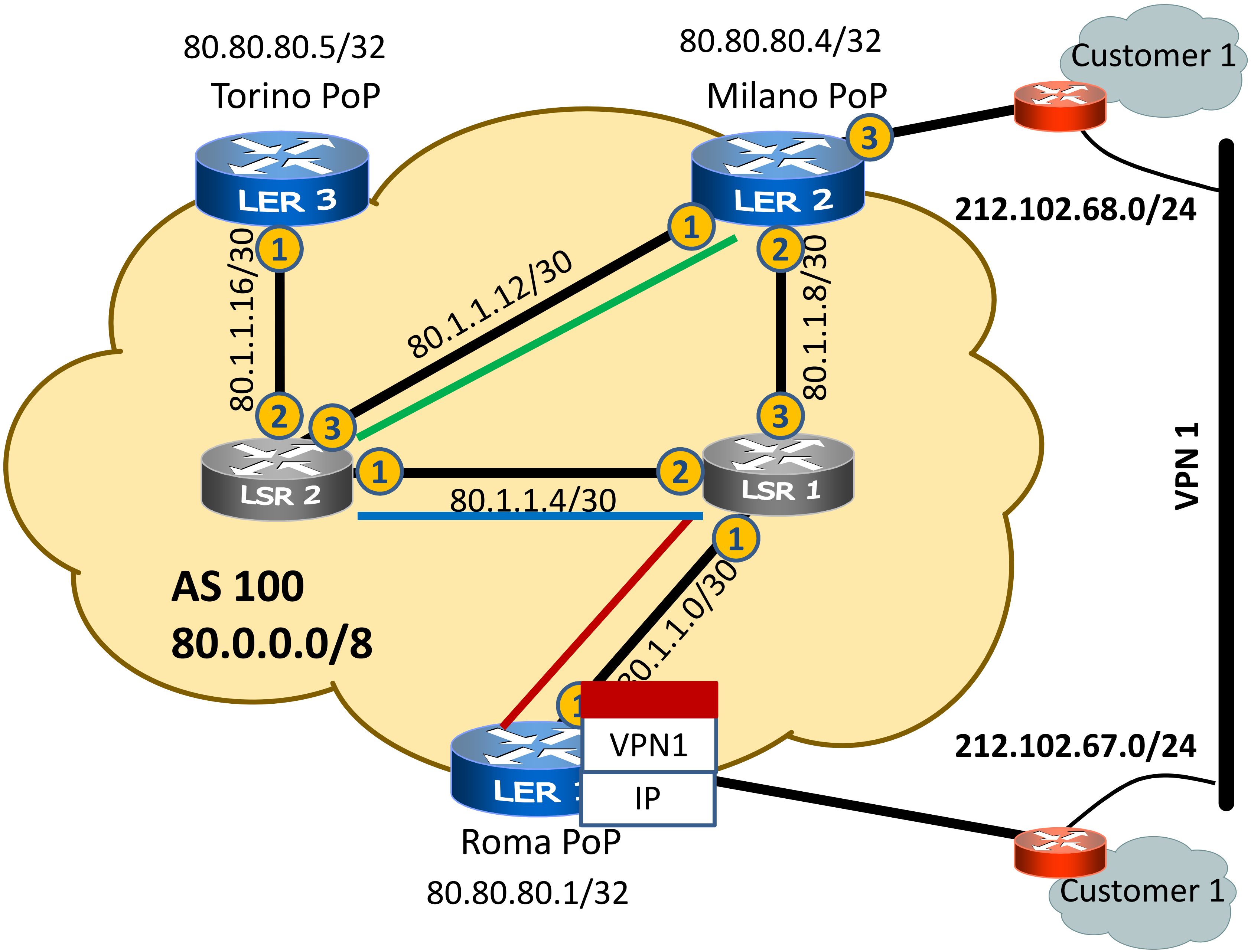
80.80.80.5/32
Torino PoP
LER 3
80.80.80.4/32
Milano PoP
LER 2
Customer 1
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
VPN 1
LSR 1
80.1.1.4/30
80.1.1.0/30
look up
VPN1::dst addr in
BGP table
next-hop = LER2
VPN1
IP
Roma PoP
80.80.80.1/32
212.102.67.0/24
Customer 1

Example



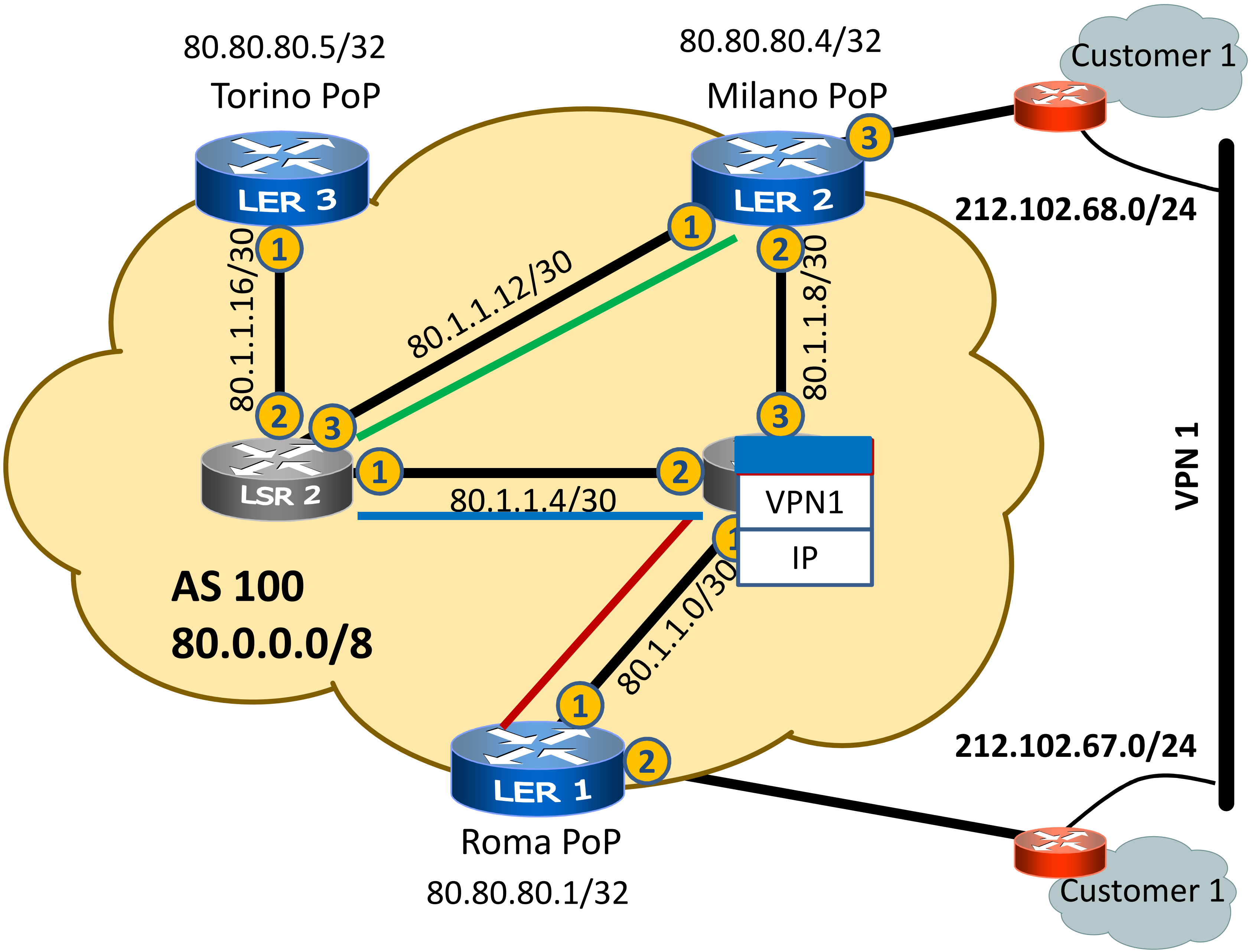
80.80.80.5/32
Torino PoP
80.80.80.4/32
Milano PoP
Customer 1
LER 3
LER 2
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
LSR 2
LSR 1
80.1.1.4/30
VPN 1
AS 100
80.1.1.0/30
look up
80.80.80.4 in LFIB
label = 20
interface = GE1/0
VPN1
IP
212.102.67.0/24
Roma PoP
80.80.80.1/32
Customer 1

Example

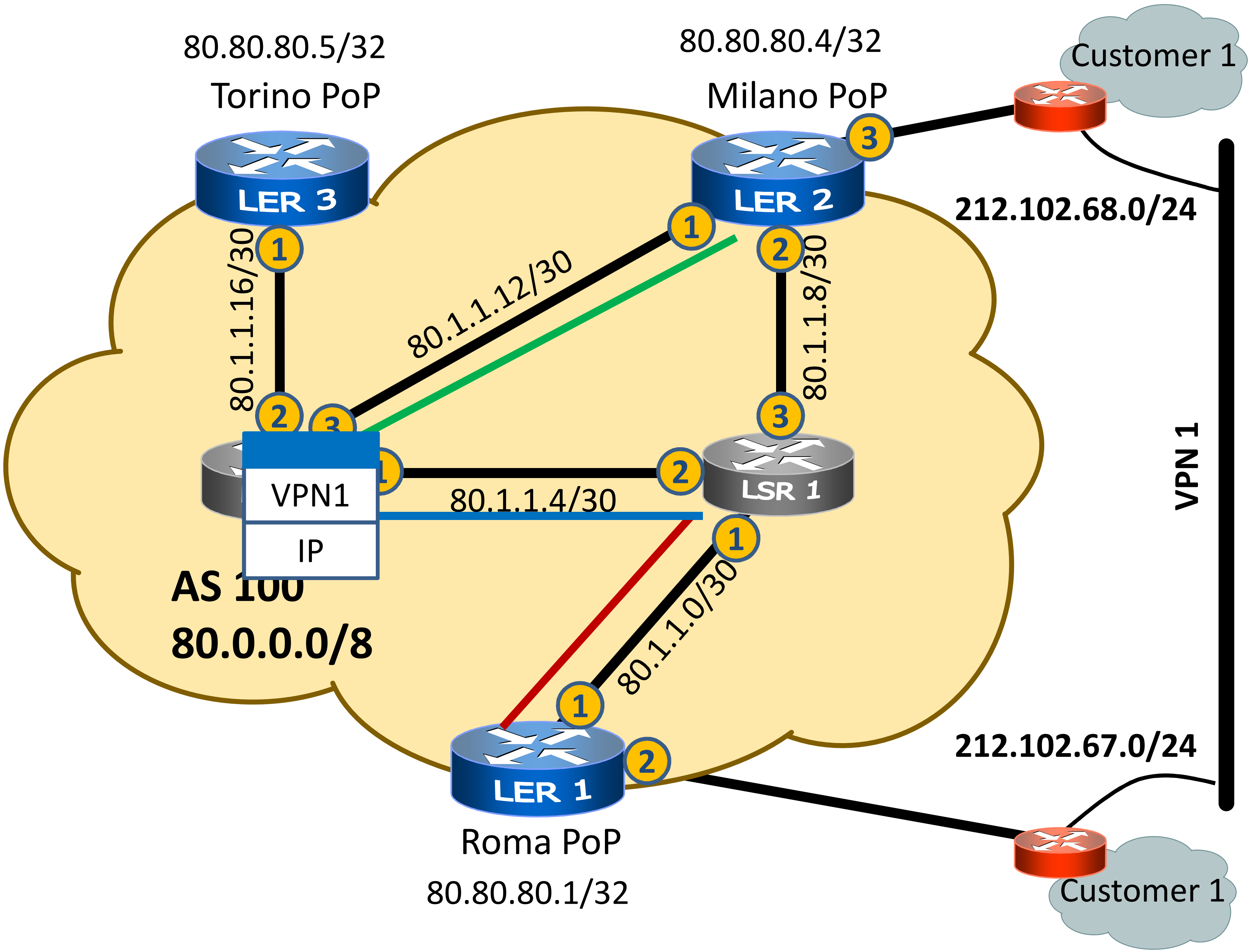


80.80.80.5/32
Torino PoP
80.80.80.4/32
Milano PoP
Customer 1
LER 3
LER 2
1
3
1
2
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
2
3
3
LSR 2
1
2
LSR 1
80.1.1.4/30
1
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
VPN1
IP
LER 1
212.102.67.0/24
Roma PoP
80.80.80.1/32
Customer 1

Example

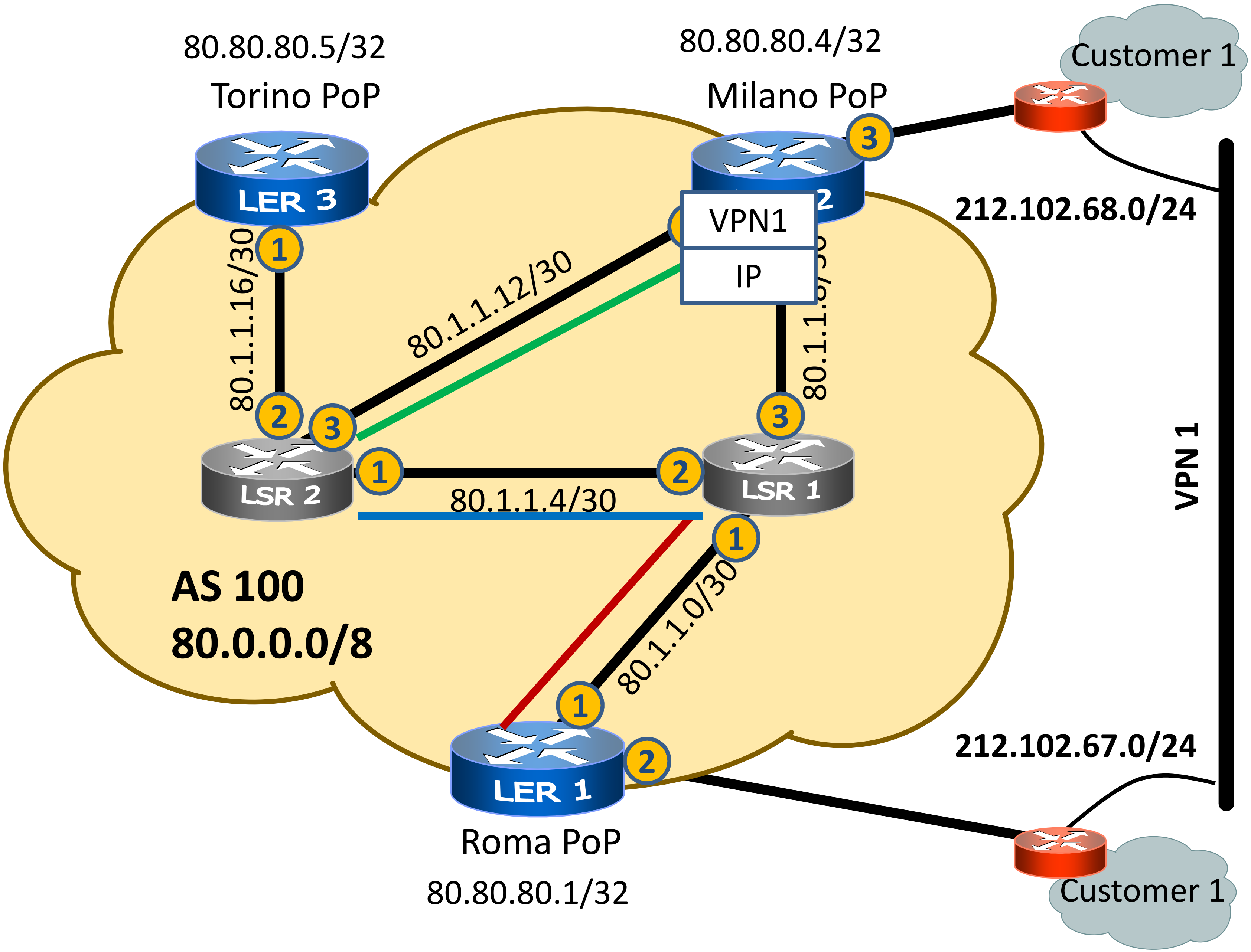

80.80.80.5/32
Torino PoP
80.80.80.4/32
Milano PoP
Customer 1
LER 3
LER 2
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
LSR 2
80.1.1.4/30
VPN1
IP
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
212.102.67.0/24
LER 1
Roma PoP
80.80.80.1/32
Customer 1

Example



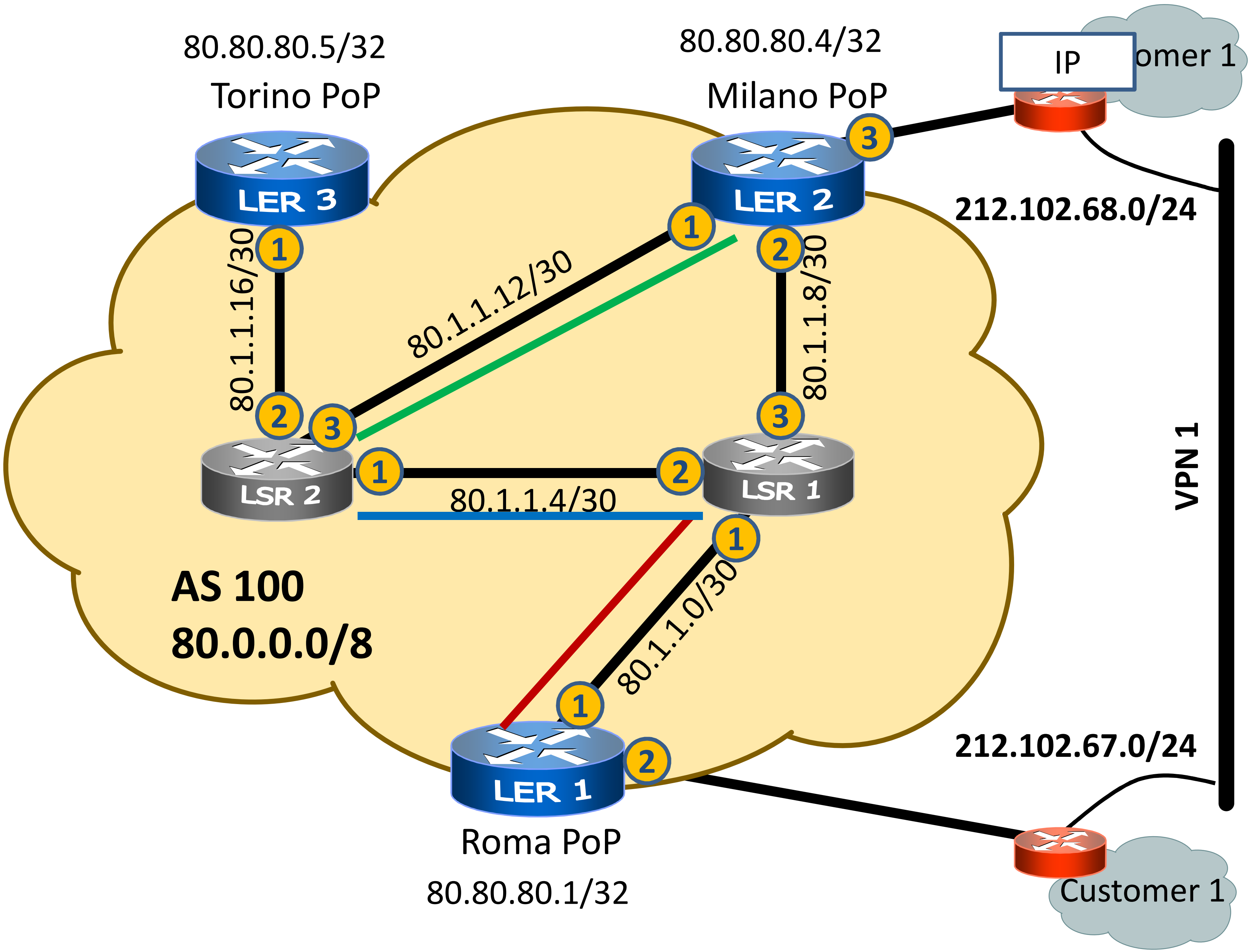
80.80.80.5/32
Torino PoP
80.80.80.4/32
Milano PoP
Customer 1
LER 3
LER 2
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
VPN1
IP
80.1.1.4/30
LSR 1
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
LER 1
212.102.67.0/24
Roma PoP
80.80.80.1/32
Customer 1

Example



80.80.80.5/32
Torino PoP
LER 3
80.80.80.4/32
Milano PoP
Customer 1
VPN1
IP
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
LSR 2
LSR 1
80.1.1.4/30
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
212.102.67.0/24
LER 1
Roma PoP
80.80.80.1/32
Customer 1

Example


80.80.80.5/32
Torino PoP
LER 3
80.80.80.4/32
Milano PoP
LER 2
IP
omer 1
212.102.68.0/24
80.1.1.16/30
80.1.1.12/30
80.1.1.8/30
LSR 2
LSR 1
80.1.1.4/30
VPN 1
AS 100
80.0.0.0/8
80.1.1.0/30
212.102.67.0/24
LER 1
Roma PoP
80.80.80.1/32
Customer 1
1
2
3

MPLS data-plane: observations

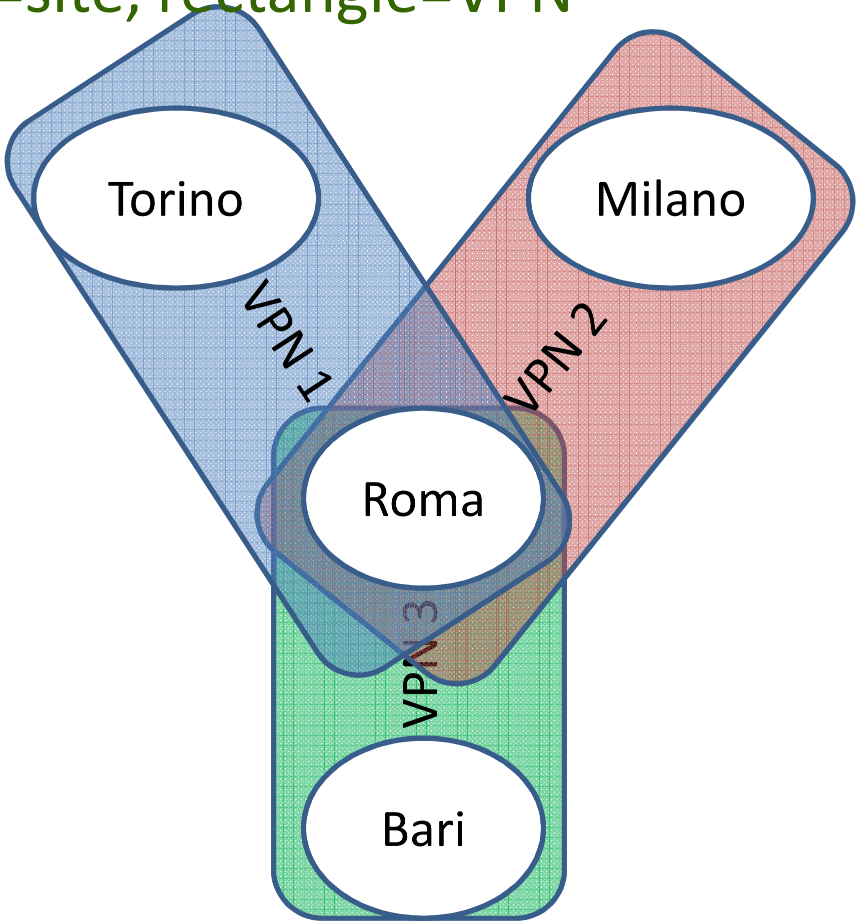
- MPLS data-plane must establish LSPs between PEs' loopback interfaces
- each Label Switching Router (LSR) uses its table to swap the top MPLS label and forward the packet to the next hop
- the penultimate router (LSR2 in the example) pops the tag
  - this way the next hop (PE) will only see the underlying VPN label
  - the PE uses the VPN label to distinguish among different VPNs

does a Route Distinguisher suffice?

- Route Distinguisher (RD) helps with overlapping address spaces
  - it makes every customer prefix unique
  - it **usually** indicates the VPN
- sometimes sites need to be connected with a VPN topology which is not a mesh
- if RDs were just used to indicate the VPN, this would not be possible

complex VPN topologies

- Consider this network
  - ellipse=site, rectangle=VPN

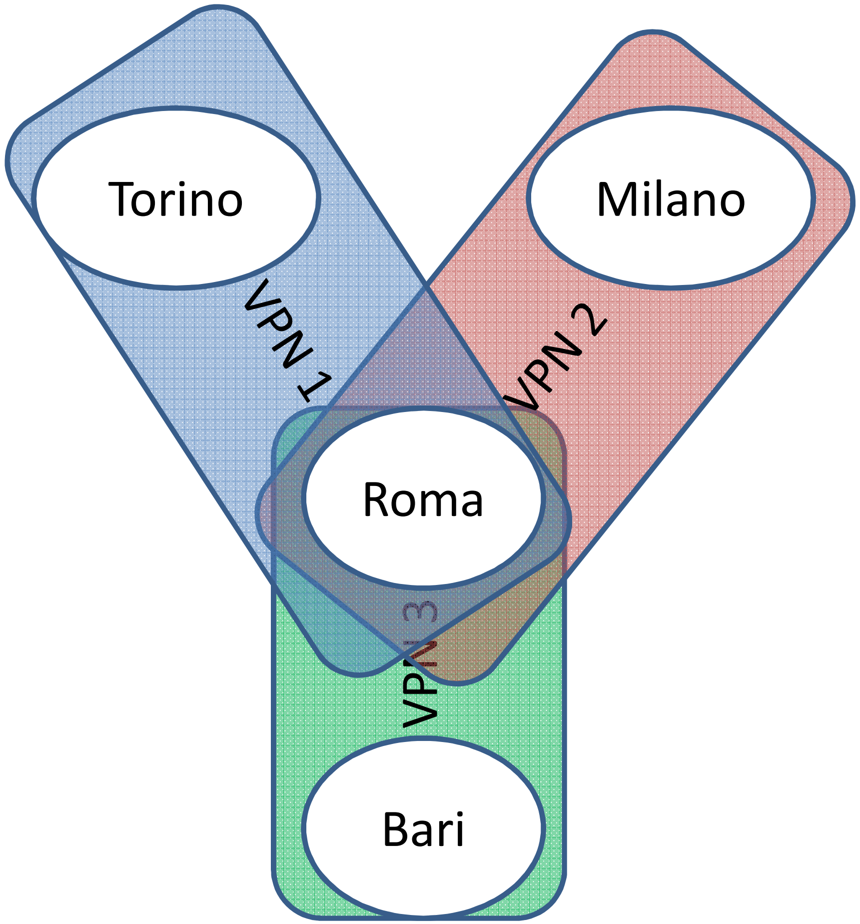


Route Target

- MPLS offers a flexible tool: Route Target (RT)
- RT is an extended community in MP-BGP
  - it can be used to indicate which routes should be imported/exported into which VRF instance
- supports complex VPN topologies
- common strategy
  - assign a RD for each site
  - assign a RT for each VPN
  - configure PE routers to import routes with specific route targets

Route Target - example

- Route Distinguishers:
  - Torino RD 100:1
  - Milano RD 100:2
  - Roma RD 100:3
  - Bari RD 100:4
- Route Targets:
  - VPN1 => 100:1000
  - VPN2 => 100:2000
  - VPN3 => 100:3000



Route Target - example

- PE at Torino

  ip vrf siteTorino

  rd 100:1

  route-target import 100:1000

  route target export 100:1000

- PE at Roma

  ip vrf siteRoma

  rd 100:3

  route-target import 100:1000

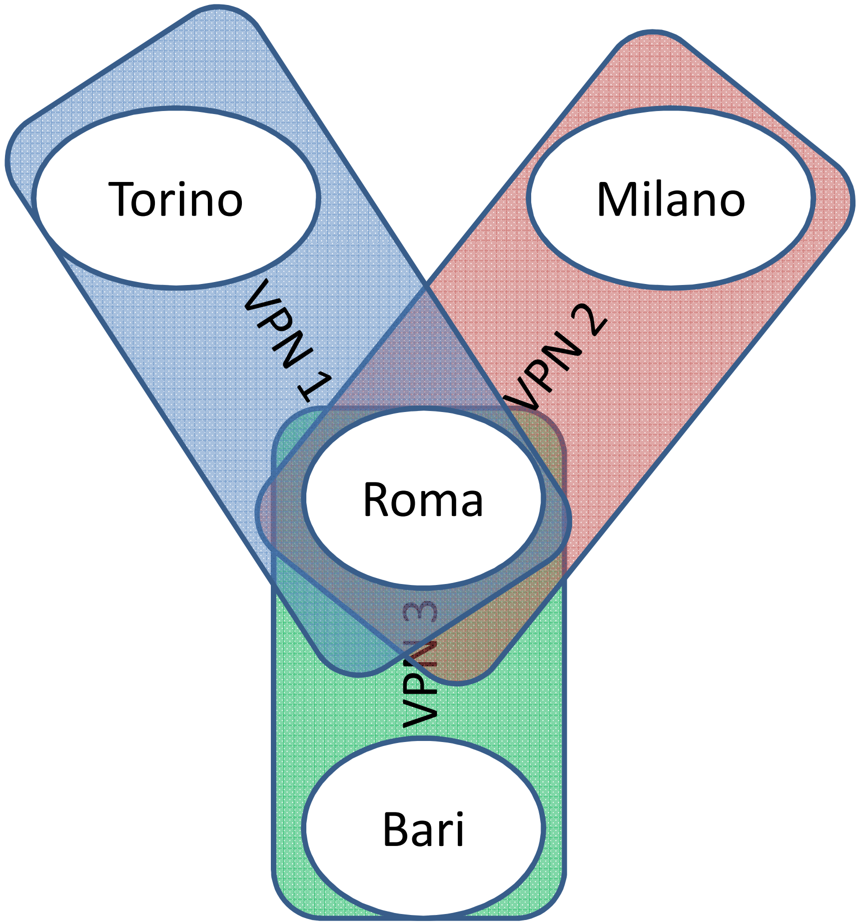
  route target export 100:1000

  route-target import 100:2000

  route target export 100:2000

  route-target import 100:3000

  route target export 100:3000



configuration

- configuring a LSR is straightforward:

```
hostname LSR1
mpls label protocol ldp
interface Loopback0
 ip address 80.80.80.2 255.255.255.255
interface GigabitEthernet1/0
 ip address 80.1.1.2 255.255.255.252
 mpls ip
interface GigabitEthernet2/0
 ip address 80.1.1.5 255.255.255.252
 mpls ip
. . . . .
router ospf 10
 network 80.0.0.0 0.255.255.255 area 10
```

configuration

- configuring a PE router is a bit more tricky
- main building blocks
  - move #1: loopback interface + speak IGP on non-customer ports
  - move #2: speak MPLS and LDP on non-customer ports,
  - move #3: full mesh of iBGP peerings
  - move #4: map customer ports to VRF instances

configuration, step 1

- example: LER1 configuration

```
hostname LER1
```

- speak IGP (in this example, OSPF)

```
 router ospf 10
  network 80.0.0.0 0.255.255.255 area 10
```

- configure loopback interface

```
 interface Loopback0
  ip address 80.80.80.1 255.255.255.255
```

configuration, step 2

- use LDP to distribute MPLS labels

```
mpls label protocol ldp
```

- speak MPLS on non-customer ports

```
interface GigabitEthernet1/0
 ip address 80.1.1.1 255.255.255.252
 mpls ip
```

configuration, step 3

- setup iBGP peerings with other PEs

```
router bgp 100
 neighbor 80.80.80.4 remote-as 100
 neighbor 80.80.80.4 update-source Loopback0
 neighbor 80.80.80.5 remote-as 100
 neighbor 80.80.80.5 update-source Loopback0
 !
 address-family vpnv4
 neighbor 80.80.80.4 activate
 neighbor 80.80.80.4 send-community both
 neighbor 80.80.80.5 activate
 neighbor 80.80.80.5 send-community both
 exit-address-family
```

- `activate` is needed for the vpnv4 address-family, otherwise routes won't be exchanged by default
- `send-community both` is needed to enable standard and extended communities

configuration, step 4

- map customer ports to VRF instances

```
interface GigabitEthernet2/0
  ip vrf forwarding VPN1
  ip address 212.102.67.1 255.255.255.0
interface GigabitEthernet3/0
  ip vrf forwarding VPN2
  ip address 193.193.172.1 255.255.255.0
```

- define RDs and RTs for each VRF instance

```
ip vrf VPN1
  rd 100:11
  route-target export 100:1000
  route-target import 100:1000
ip vrf VPN2
  rd 100:22
  route-target export 100:2000
  route-target import 100:2000
```

configuration, step 4 (continued)

- announce VPN prefixes in BGP

```
router bgp 100
  address-family ipv4 vrf VPN1
  redistribute connected
  exit-address-family
  !
  address-family ipv4 vrf VPN2
  redistribute connected
  exit-address-family
```

MP-BGP in the wild

```
▽ Path attributes
  ▷ ORIGIN: INCOMPLETE (4 bytes)
  ▷ AS_PATH: empty (3 bytes)
  ▷ MULTI_EXIT_DISC: 0 (7 bytes)
  ▷ LOCAL_PREF: 100 (7 bytes)
  ▽ EXTENDED_COMMUNITIES: (11 bytes)
    ▷ Flags: 0xc0 (Optional, Transitive, Complete)
      Type code: EXTENDED_COMMUNITIES (16)
      Length: 8 bytes
    ▷ Carried Extended communities
  ▽ MP_REACH_NLRI (35 bytes)
    ▷ Flags: 0x80 (Optional, Non-transitive, Complete)
      Type code: MP_REACH_NLRI (14)
      Length: 32 bytes
      Address family: IPv4 (1)
      Subsequent address family identifier: Labeled VPN Unicast (128)
    ▷ Next hop network address (12 bytes)
      Subnetwork points of attachment: 0
    ▽ Network layer reachability information (15 bytes)
      ▷ Label Stack=24 (bottom) RD=100:11, IPv4=212.102.67.0/24
```

Route Target

Label Stack: advertises the VPN label

VPN-IP addr

summary

low configuration and maintenance costs

- CE - customer edge routers
  - don't need any special configuration; connected to a PE router via IP (e.g. with a point-to-point connection)
- PE - provider edge routers
  - simple configuration that depends only on the sites of the VPN's that are adjaceny to the PE
- P – provider routers
  - simple configuration that does not depend on the deployed VPN's

forwarding efficiency

- PEs and Ps forward packet only depending on the labels, that, in turn, depend only on the loopbacks of the PEs
- the forwarding tables in the backbone contain only one entry for each loopback
- much less than one entry for each customer prefix

qos

- exploit traffic class field for enforcing QoS
  - tos field of packets is encapsulated inside the MPLS envelope and hence is not accessible while the packet is traversing the backbone
  - tos field is copied in the qos bits of MPLS label (named EXP bits) at the PE

internet

- many customers will also require Internet access as well as VPN access
- more than one way to make this work (rfc4364)
  - CE announces 0/0 to PE
    - works even if the customer has another ISP for Internet
  - Internet packets are forwarded natively (i.e., no MPLS)
  - PEs leak Internet routes in each VRF
- a possible alternative
  - 0/0 is associated with a specific RT, VPNs needing Internet access import it

caveats

MTU

- careful with the MTU inside the backbone
  - each MPLS label takes 4 bytes
  - risk of fragmentation
    - high impact on performance
- IEEE 802.3 standard mandates support for **one** of
  - 1500 bytes MTU
  - 1504 bytes MTU (Q-tag)
  - 1982 bytes MTU (“envelope” frame)
- in practice, it is a matter of
  - implementation
  - hw support
- nowadays most OSes do PMTU discovery, so having an Ethernet MTU of 1492 bytes (allowing 2 MPLS labels) is not a big issue
  - PPPoE also has a MTU of 1492 bytes – did you ever have problems?

TTL

- IP TTL field is encapsulated in an MPLS envelope, hence not accessible from LSRs
  - how do we prevent infinite loops in MPLS?
- recall MPLS label has its own TTL field
  - when the PE encapsulates the IP packet, it copies the TTL value in the newly added MPLS label
  - LSRs decrement the TTL in the label
  - when the label is popped, the TTL is copied onto the next label
  - when the bottom-of-stack label is popped, the TTL is copied onto the IP field