Università di Roma Tor Vergata

University of Rome Tor Vergata
ICT and Internet Engineering

Network and System Defense

Alessandro Pellegrini, Angelo Tulumello

A.A. 2023/2024

Lecture 9: BGP/MPLS VPNs

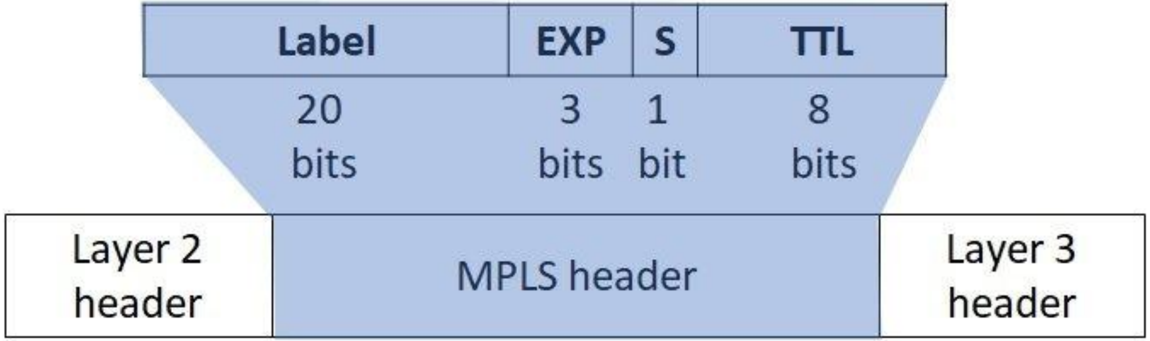
Angelo Tulumello

Slides by Marco Bonola

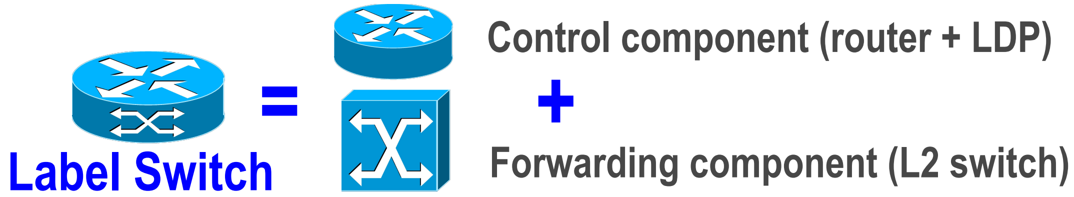
MultiProtocol Label Switching (MPLS)

MPLS: architecture

- The key idea of the MPLS architecture is to associate a brief identifier, namely Label, to every packet.
- Internetworking nodes can then apply fast forwarding mechanisms based on label switching / label swapping
- MPLS is independent both from the transport subnet (Frame Relay, ATM, etc.) both from adopted network protocols



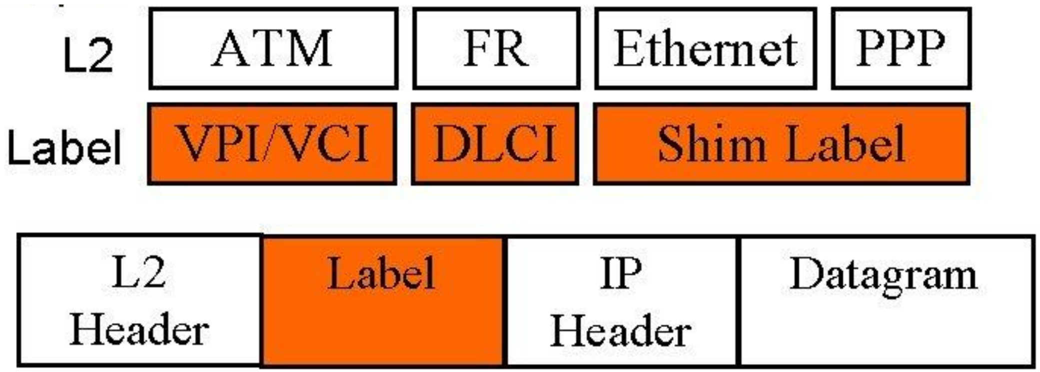
MPLS Network Node



- ***Control Component***
  - A set of modules dealing with Label allocation and binding Labels between adjacent nodes
  - Layer 3 «intelligence» (IP addressing, IP routing)
- ***Forwarding Component***
  - Forwarding based on the label swapping paradigm
- The two components must be independent: they can employ different protocols within every medium
- The Control Component is sometimes realized as a part (SW or HW) of the network node, other times as external controller

Label Encoding

- If data-link layer natively supports a field for the label (ATM does it with VPI/VCI, Frame Relay with DLCI), this can be used to insert the MPLS label
- If data-link layer doesn’t support that field, the MPLS label is embedded in an MPLS header, inserted between layer 2 and layer 3 headers (e.g. Ethernet/MPLS/IP)

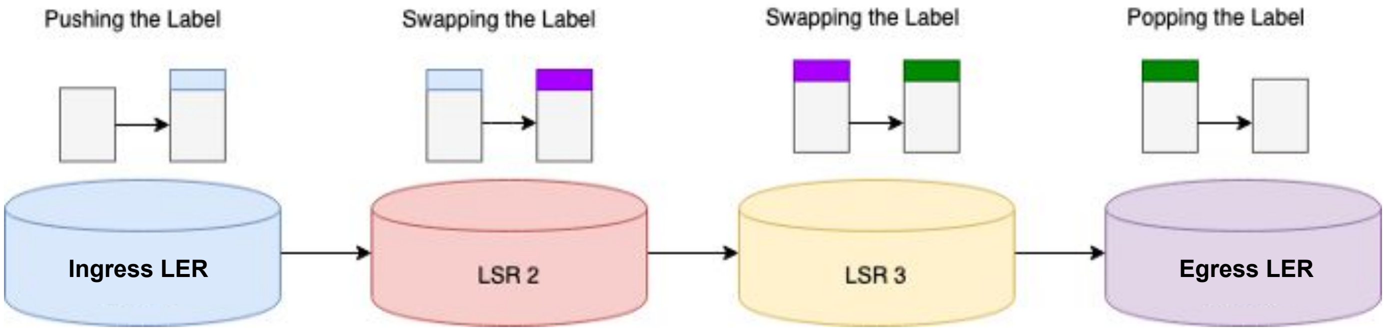


Terminology

- ***Label Edge Router (LER):*** edge routers for an MPLS network: they have forwarding functionalities from and to the outer networks, applying and removing the labels to ingress and egress packets
- ***Label Switching Router (LSR):*** switches operating label swapping inside the MPLS network and supporting forwarding functionalities
- ***Label Distribution Protocol (LDP):*** in conjunction with traditional routing protocols, LDP is used for distributing labels between network devices
- ***Forwarding Equivalence Class (FEC):*** a set of IP packets that are forwarded in the same way (for instance along the same path, with the same treatment)
- ***Label Switched Path (LSP):*** the path through one or more LSRs followed by a packet belonging to a certain FEC

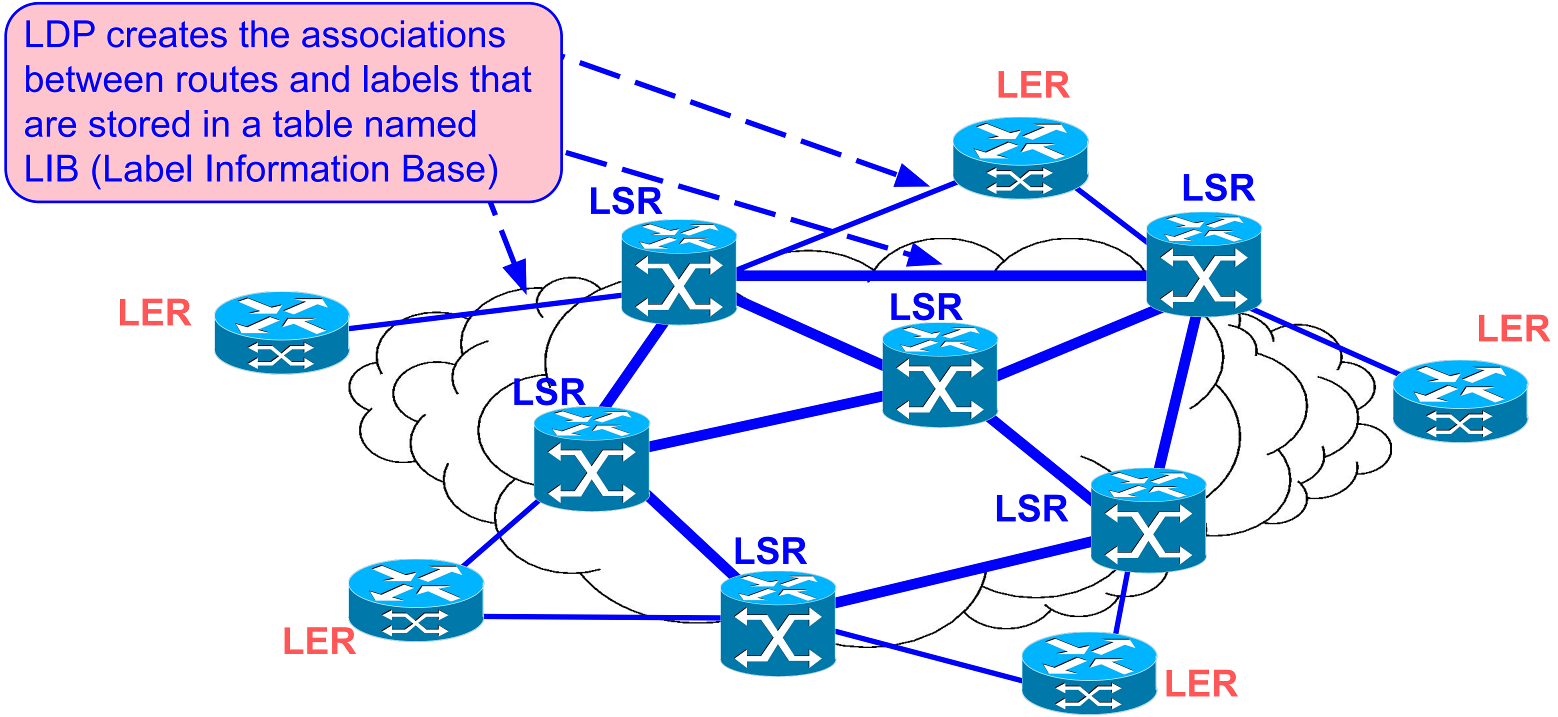
Label Switching Operation: Push, Forwarding and Pop

- The ingress LER of the MPLS backbone analyzes the packet’s IP header, classifies the packet, adds the label and forwards it to the next hop LSR
- In the LSRs cloud the packet is forwarded along the LSP according to the label. At each hop labels are swapped (local label: remote label)
- The egress LER removes the label and the packet is forwarded based on IP destination address



Label Switching Operation: Control

LDP is used for distributing the <label, prefix> associations between MPLS nodes

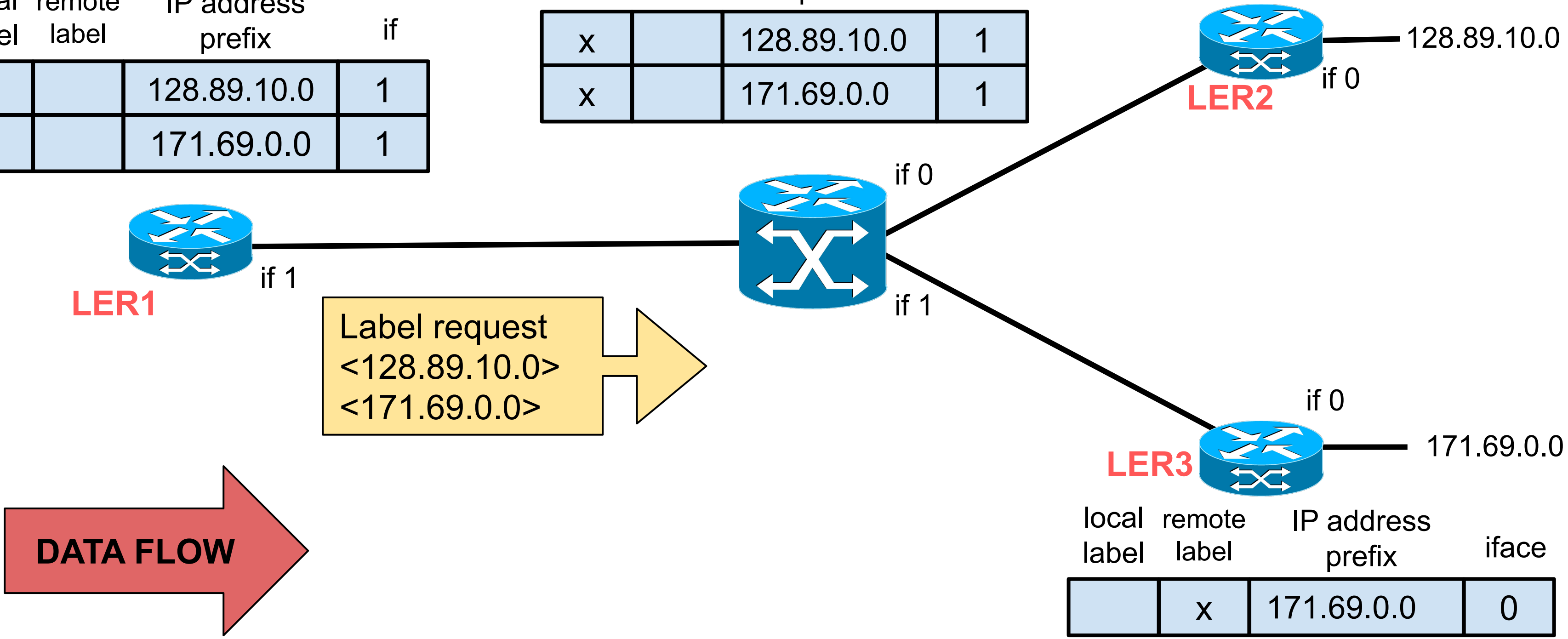


LDP: Downstream on Demand

local label	remote label	IP address prefix	if
x		128.89.10.0	1
x		171.69.0.0	1

local label	remote label	IP address prefix	iface
x		128.89.10.0	1
x		171.69.0.0	1

local label	remote label	IP address prefix	iface
	x	128.89.10.0	0



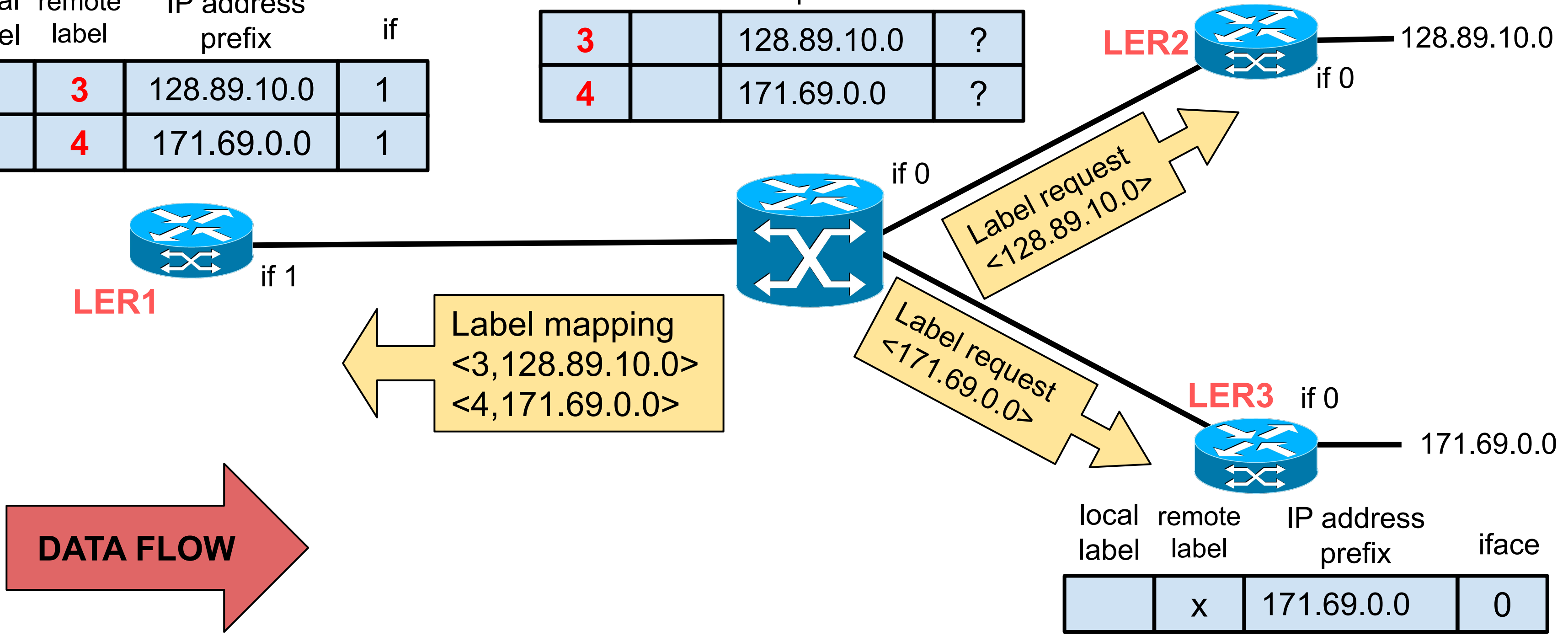
local label	remote label	IP address prefix	iface
	x	171.69.0.0	0

LDP: Downstream on Demand

local label	remote label	IP address prefix	if
x	3	128.89.10.0	1
x	4	171.69.0.0	1

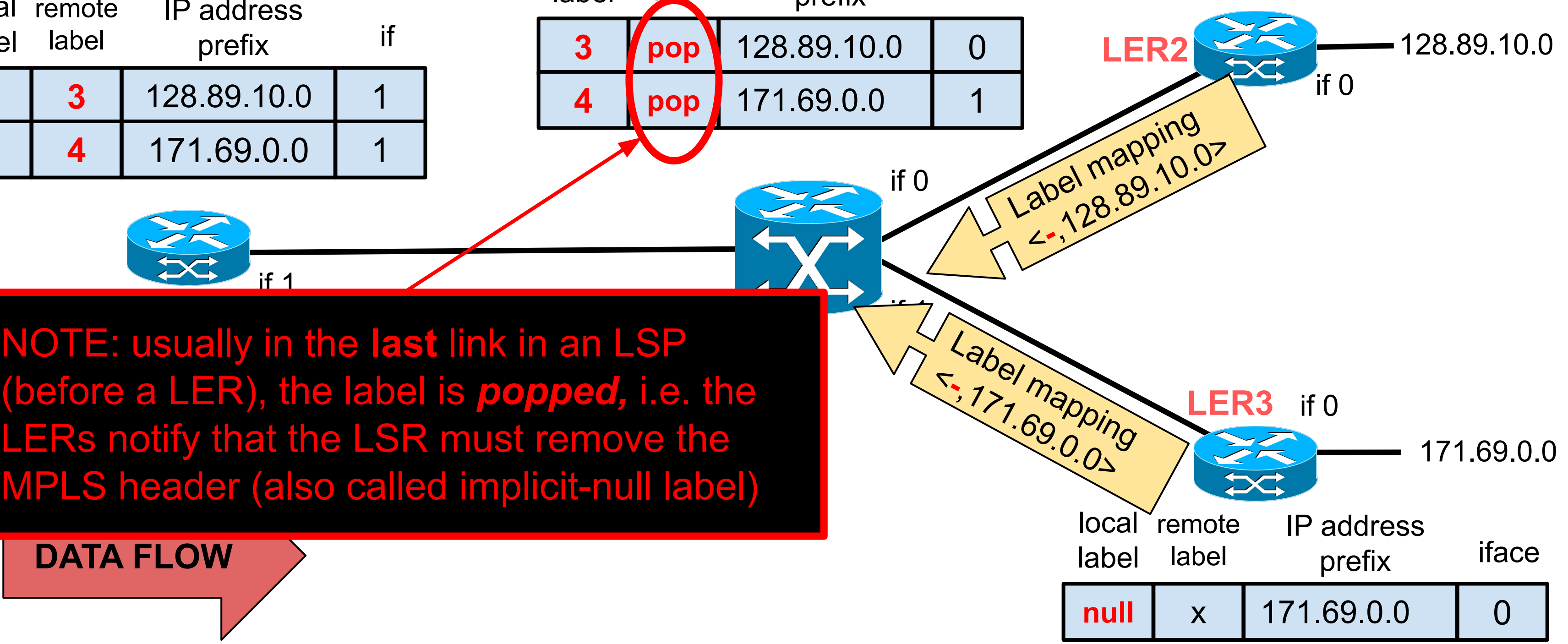
local label	remote label	IP address prefix	iface
3		128.89.10.0	?
4		171.69.0.0	?

local label	remote label	IP address prefix	iface
	x	128.89.10.0	0



local label	remote label	IP address prefix	iface
	x	171.69.0.0	0

LDP: Downstream on Demand



local label	remote label	IP address prefix	if
x	3	128.89.10.0	1
x	4	171.69.0.0	1

local label	remote label	IP address prefix	iface
3	pop	128.89.10.0	0
4	pop	171.69.0.0	1

local label	remote label	IP address prefix	iface
null	x	128.89.10.0	0

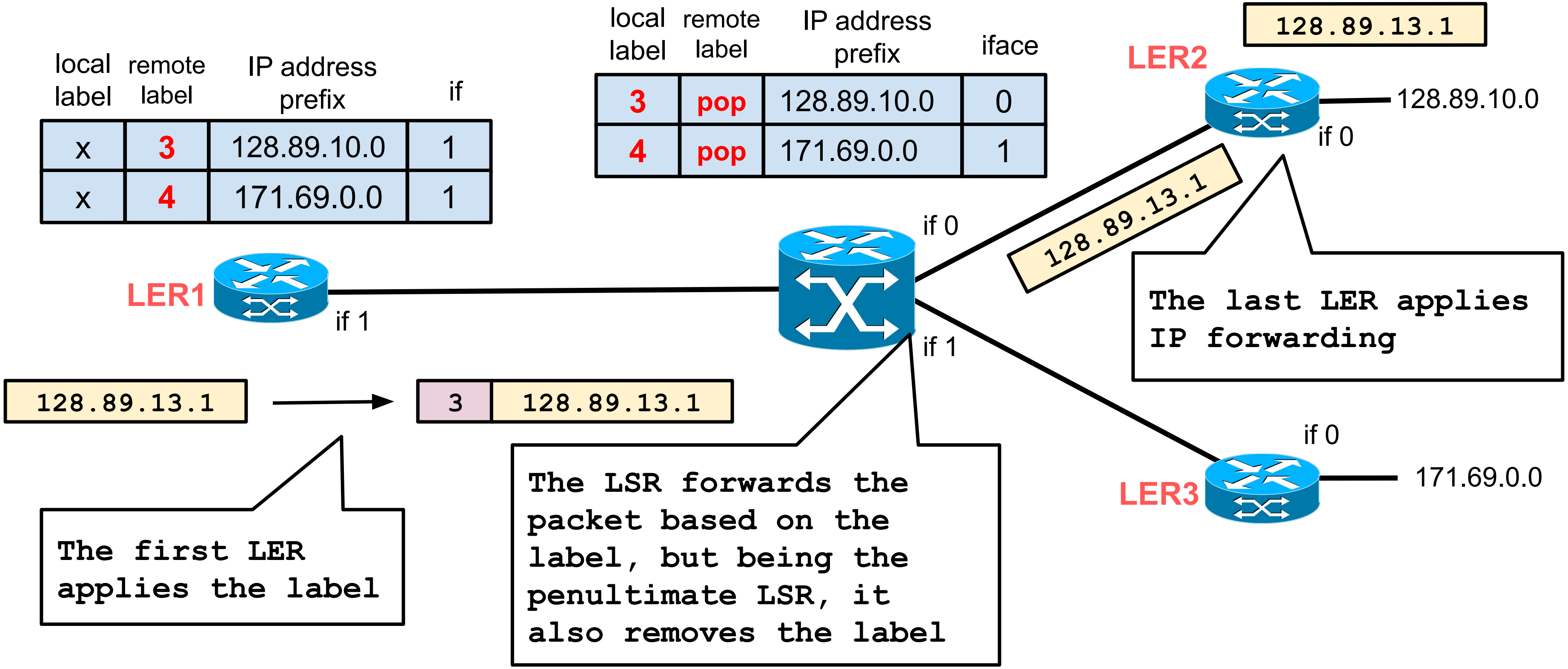
local label	remote label	IP address prefix	iface
null	x	171.69.0.0	0

NOTE: usually in the **last** link in an LSP (before a LER), the label is ***popped,*** i.e. the LERs notify that the LSR must remove the MPLS header (also called implicit-null label)

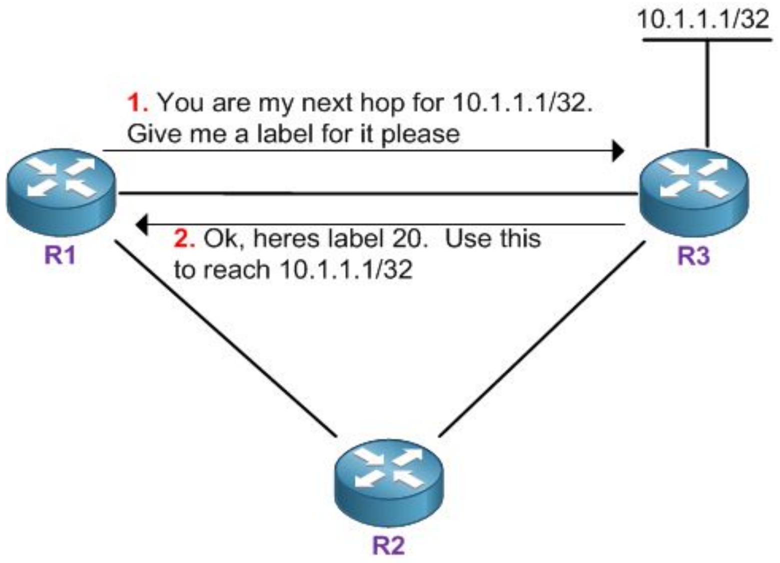
Label Switching Operation: Forwarding

local label	remote label	IP address prefix	if
x	3	128.89.10.0	1
x	4	171.69.0.0	1

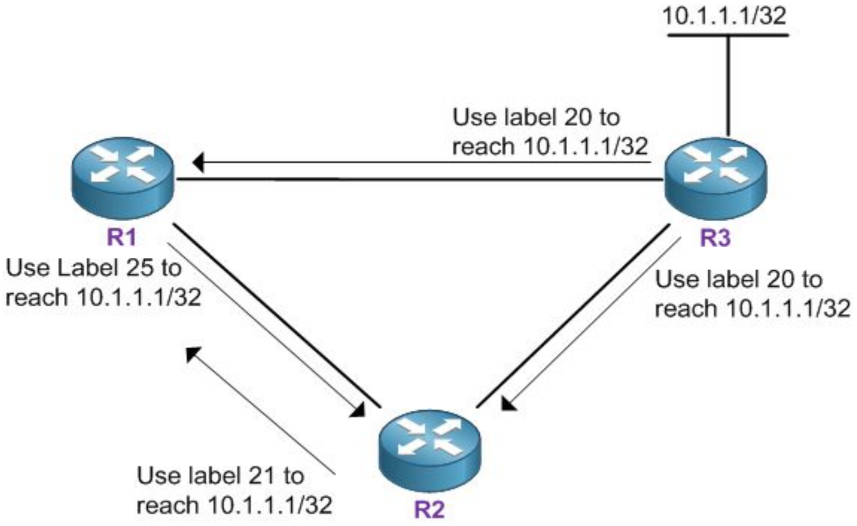
local label	remote label	IP address prefix	iface
3	pop	128.89.10.0	0
4	pop	171.69.0.0	1



LDP: Downstream OnDemand vs Unsolicited



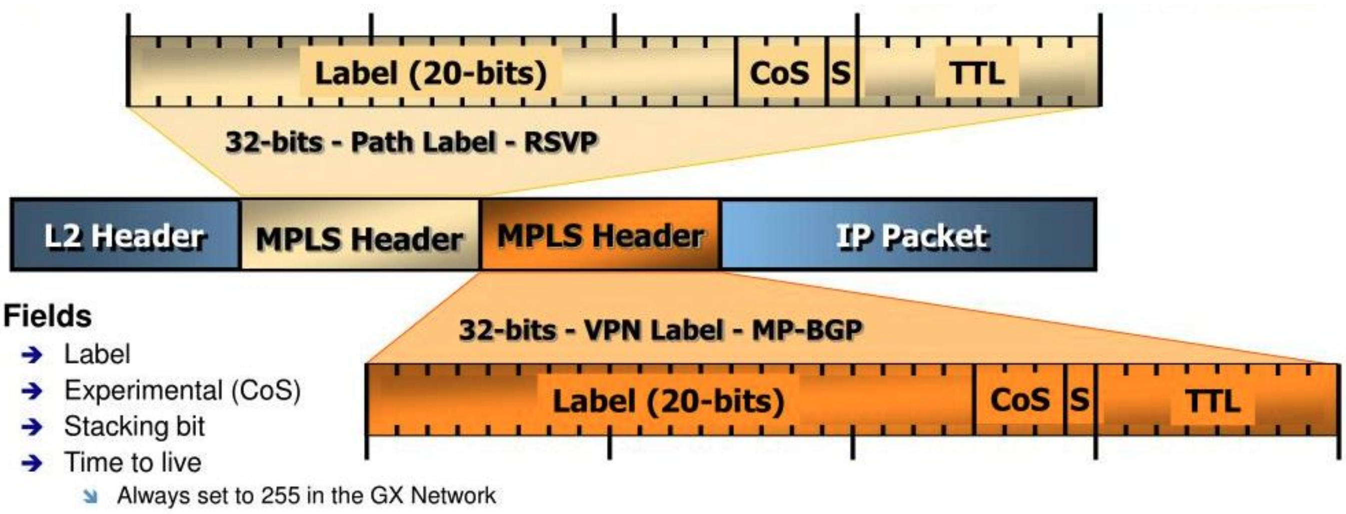
OnDemand



Unsolicited
(Cisco default)

Label Stacking

MPLS label can be stacked to aggregate, in a network section, two or more LSP in a single LSP with higher pecking order (e.g. MPLS VPNs, details in a few slides...)



MPLS and BGP

Problem: how can internal routers (e.g. R2) forward transit packets, i.e. intended to one of the 800k external routes?

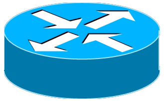
1. Replicate BGP tables also in core routers (costly)
2. Full mesh LSPs between border routers through which only transit traffic is forwarded
   - Internal routers only matters about routing tables to reach internal network nodes

Intra-AS Virtual Private Networks with MPLS/BGP

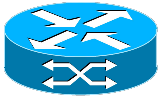
Intra-AS VPNs

- Routing Information exchange between Company and ISP routers
  - Routing happens on a layer composed both by company entities and by ISP entities
- De facto based on BGP/MPLS solution
  - Enterprise’s gateway transfers data to the ISP which handles the forwarding through other Enterprise’s sites
  - Routing (connections topology) is actually in the hands of the ISP
  - Plug & Play, adding a site is a matter of ISP configuration only, the company has to do almost nothing

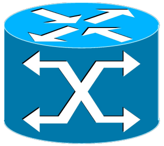
Elements of a VPN BGP/MPLS network



Customer Edge: is the Company side router facing with the ISP which provides the VPN BGP/MPLS service. It has standard routing functionalities; its only peer is the Provider edge with which exchanges info through BGP messages



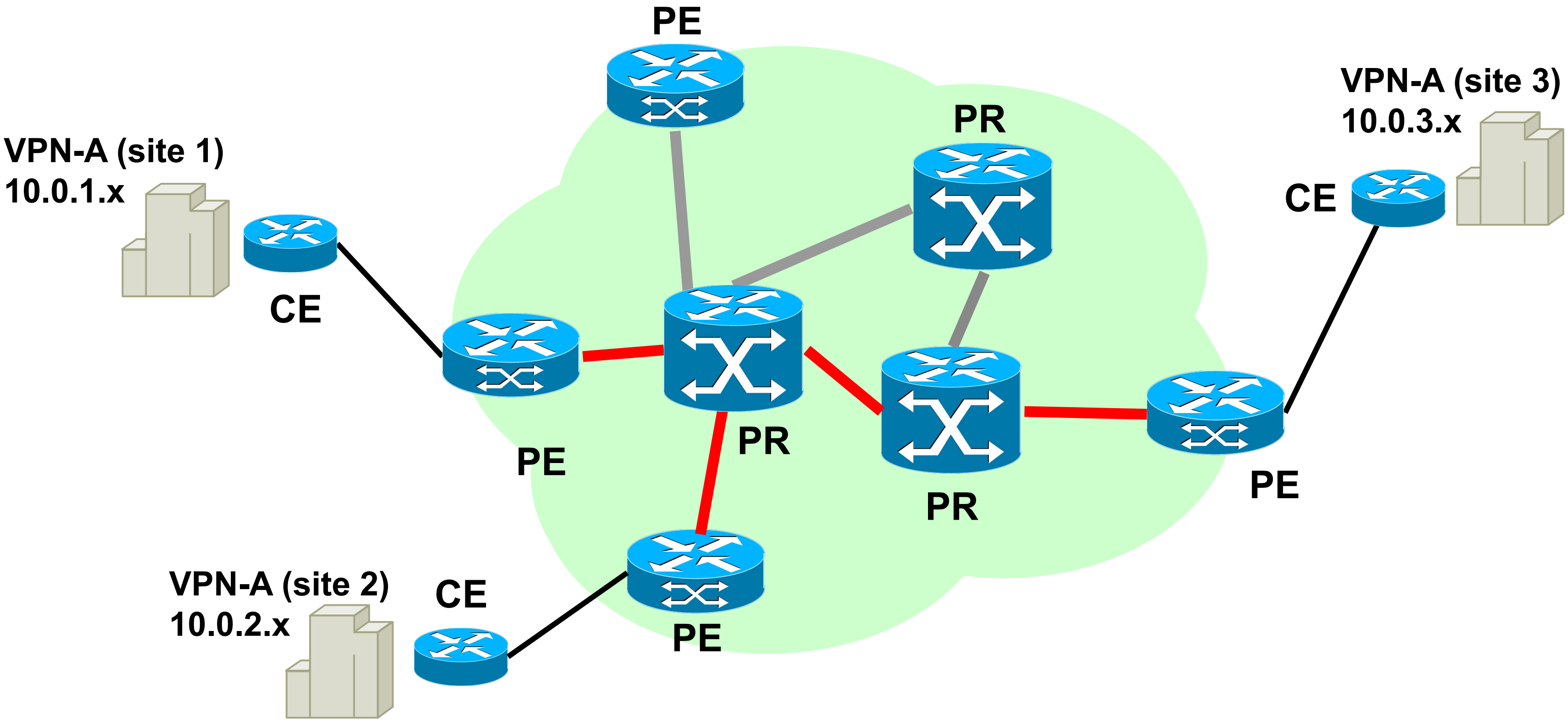
Provider Edge: is the access router on the ISP side in which one or more Customer Edges are connected. Besides IP functionalities, it also handles the MPLS LER role.



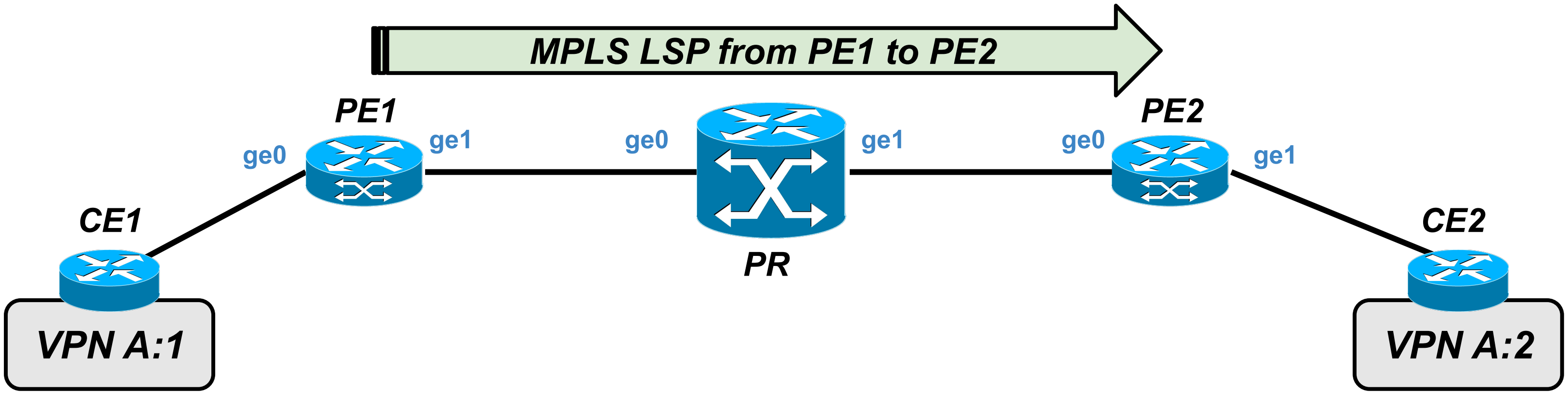
Provider Router: Label Switched Router (LSR) composing the MPLS backbone of the ISP

MPLS/VPN Backbone: MPLS network with properly configures LSPs to interconnect all the Provider Edges.

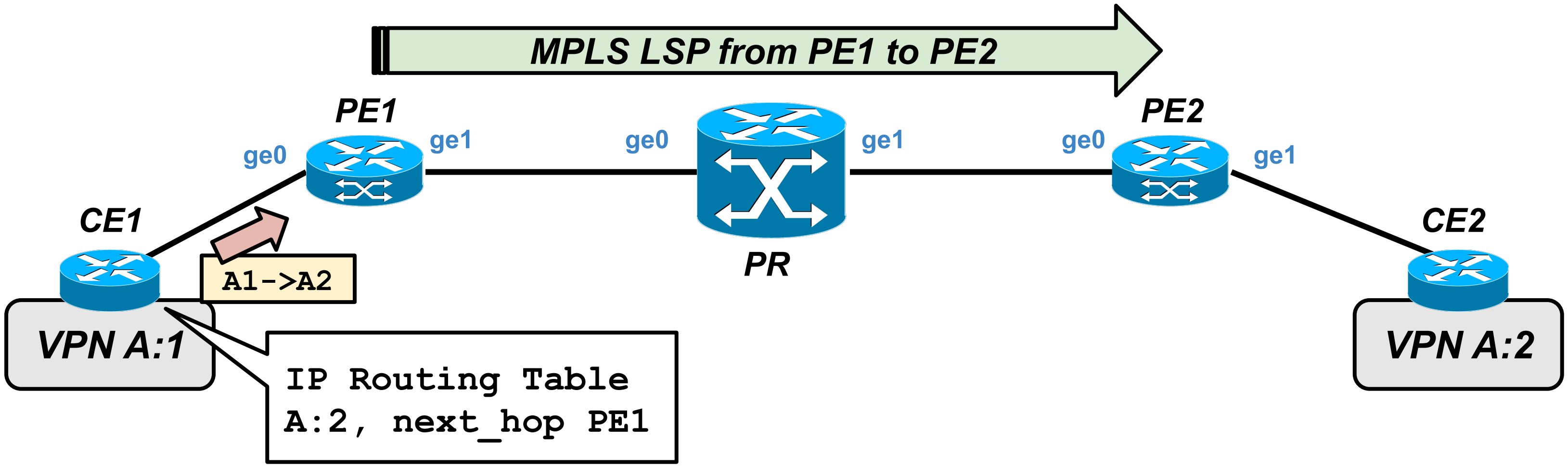
VPN MPLS service architecture



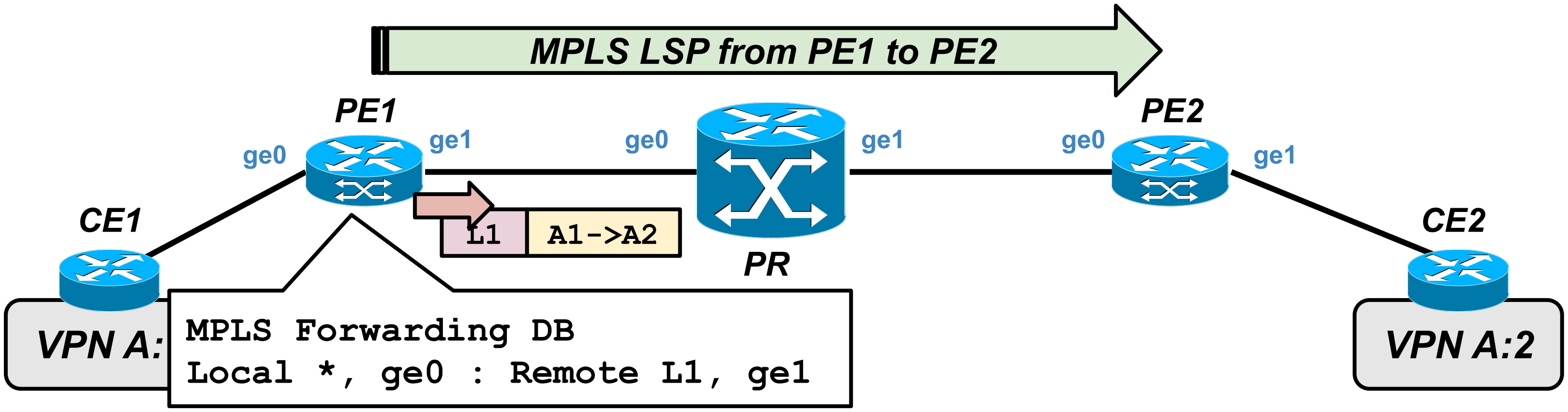
Forwarding mechanism (trivial solution)



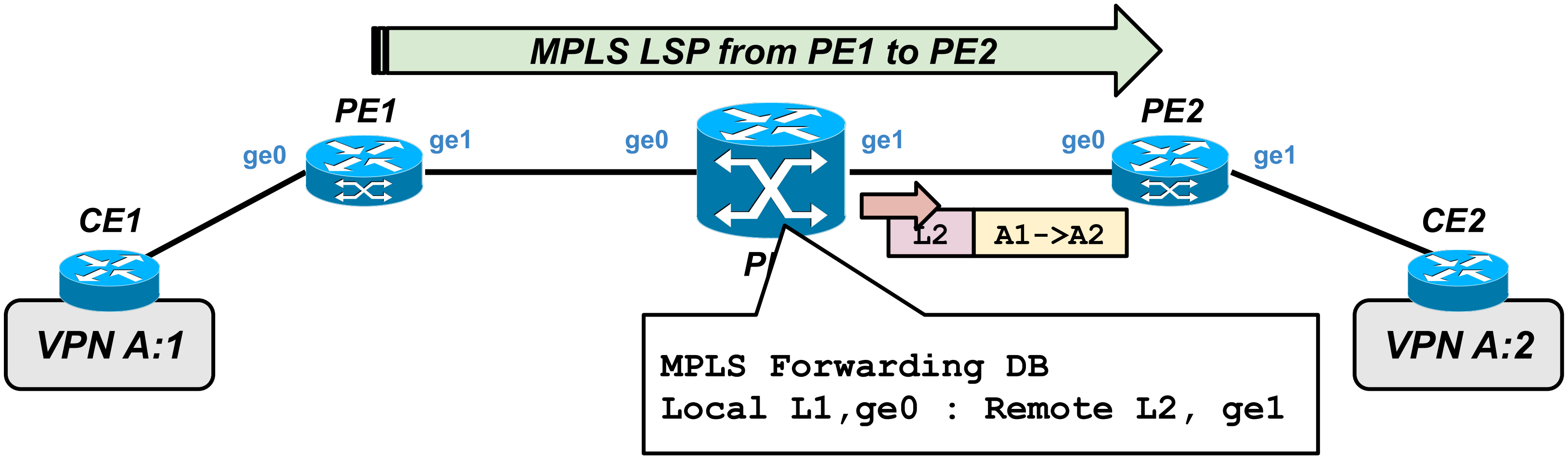
Forwarding mechanism (trivial solution)



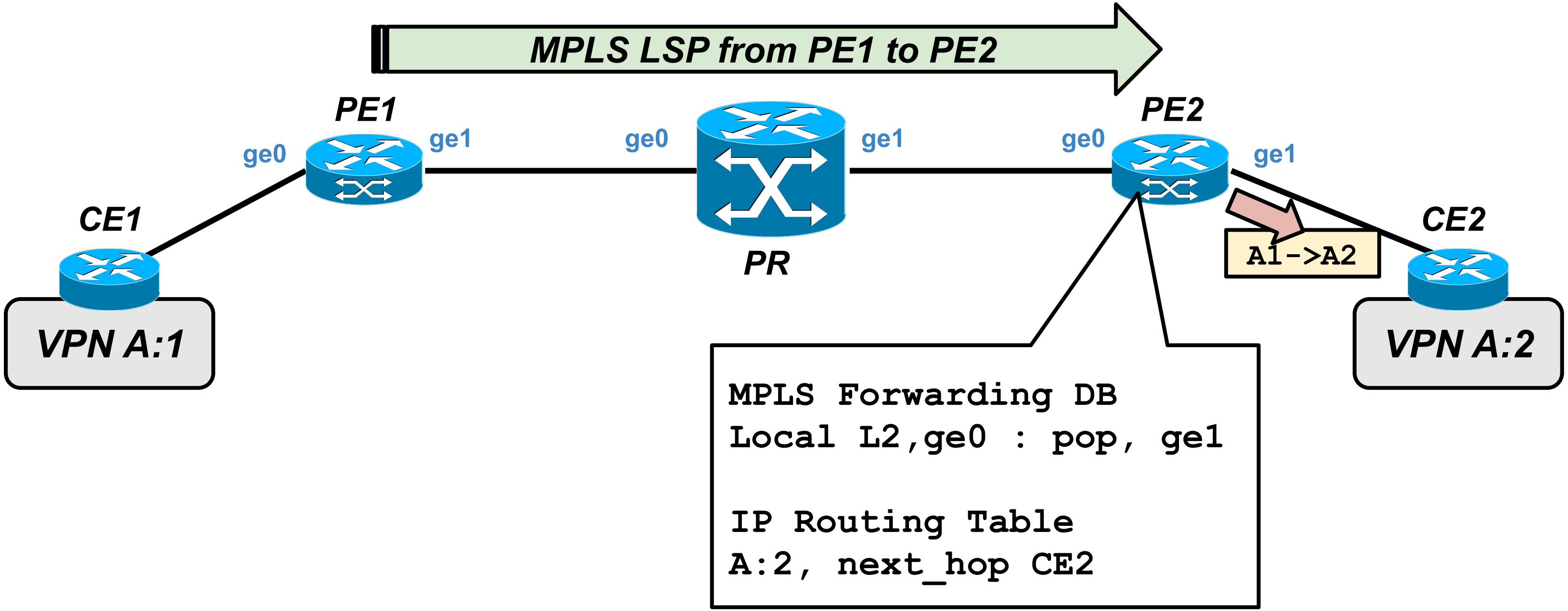
Forwarding mechanism (trivial solution)



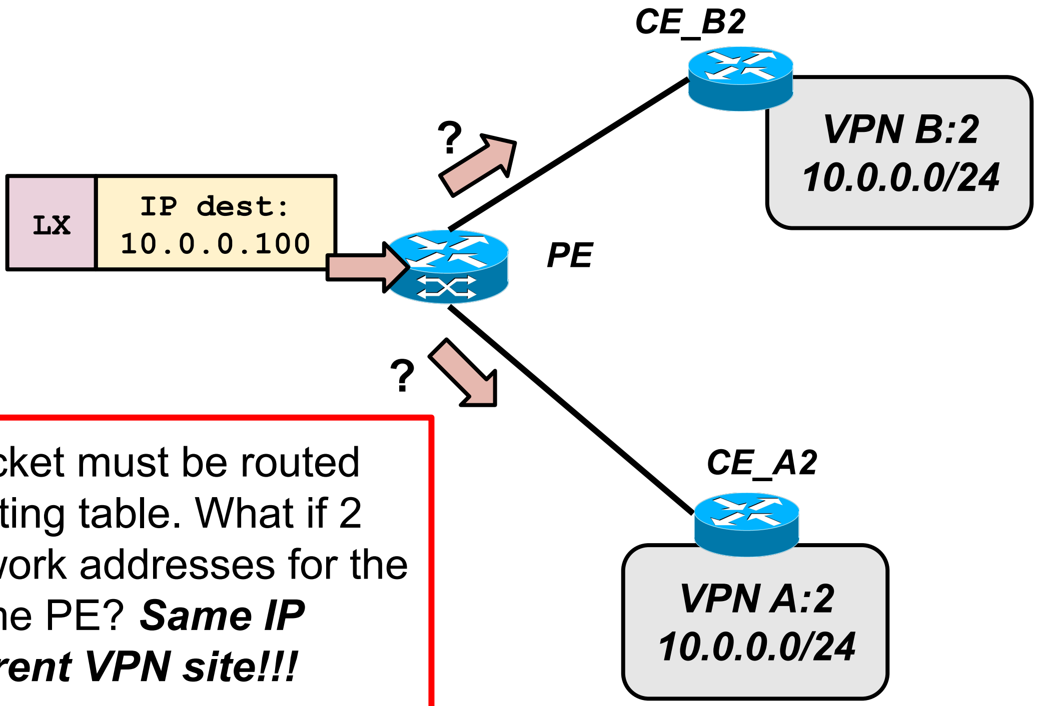
Forwarding mechanism (trivial solution)



Forwarding mechanism (trivial solution)

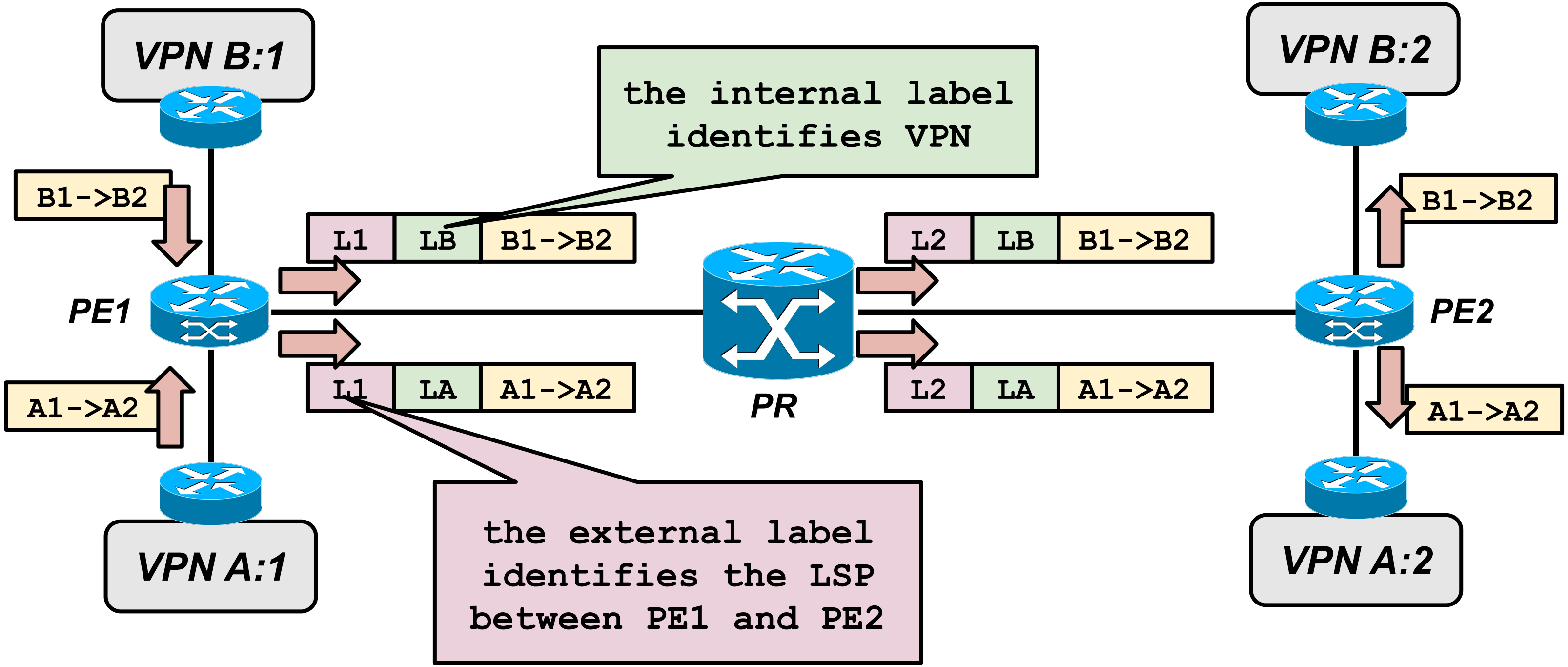


but customer VPN addressing is un-coordinated...



After the MPLS pop(), the packet must be routed according to PE's local IP routing table. What if 2 customers chose the same network addresses for the sites connected to the same PE? ***Same IP destination for two different VPN site!!!***

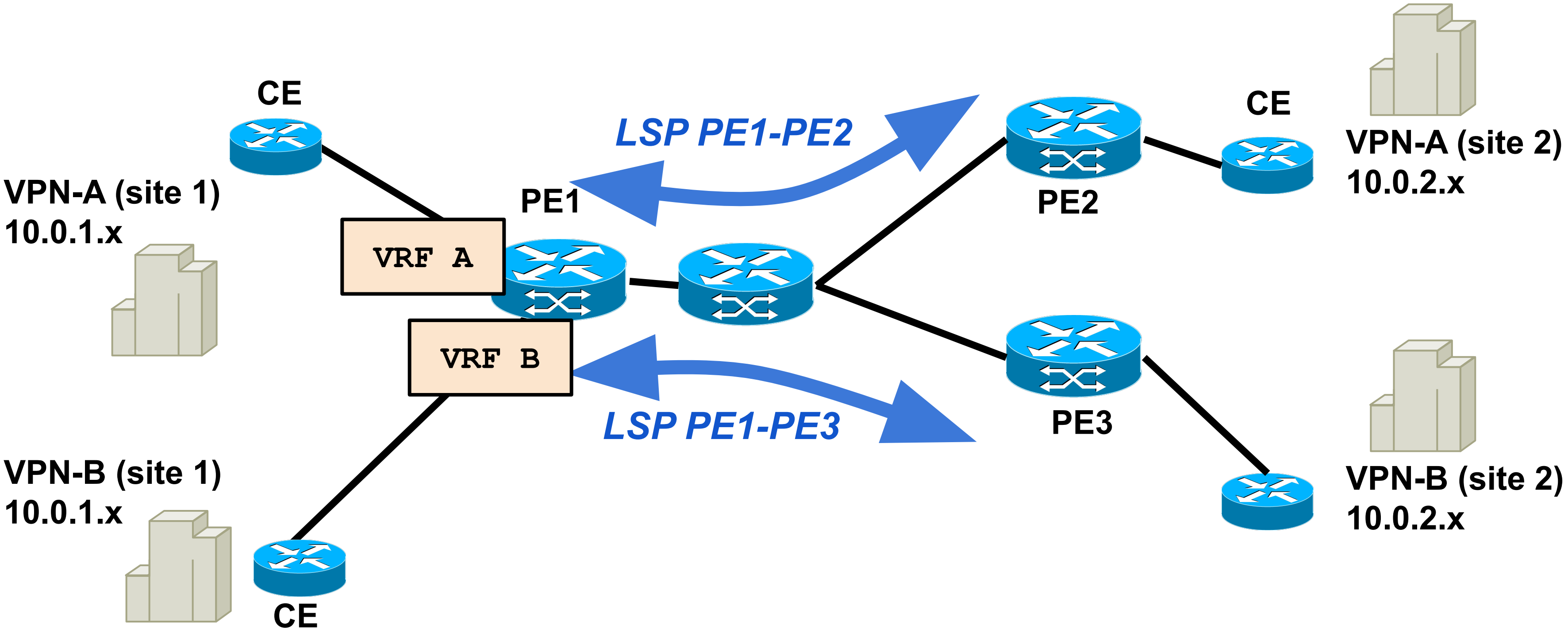
Solution: double MPLS encapsulation

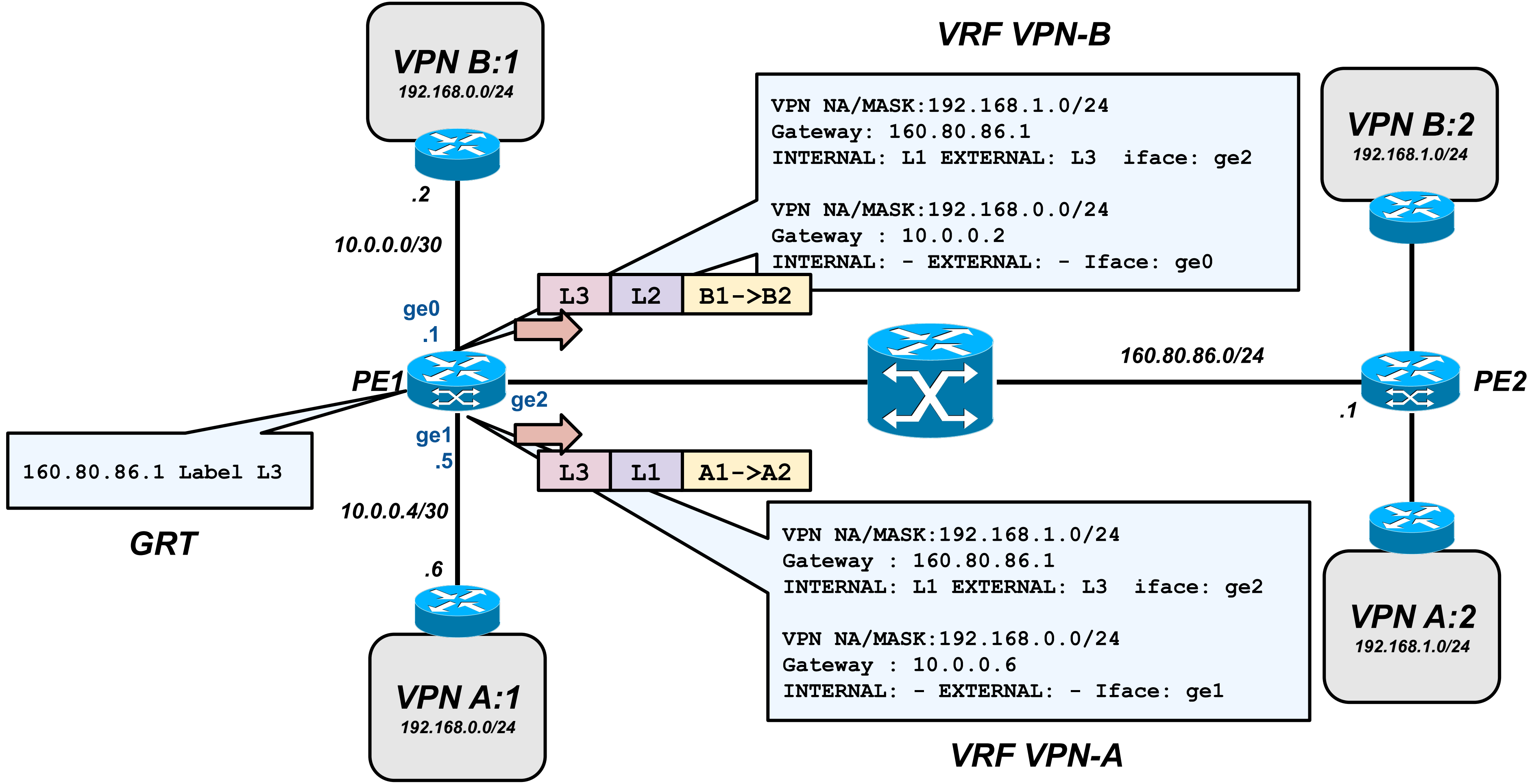


Managing multiple forwarding tables at the PE

- The PE associates the incoming packet to the customer VPN by simply ***matching the ingress interface***
- The MPLS forwarding table changes according to the specific VPN the customers belong to
- The PE must support ***as many forwarding tables as the customers VPNs connected to it***
- Such forwarding tables are called ***VPN Routing and Forwarding (VRF)*** tables
  - A VRF entry contains (logically) the following tuple: <VPN network address, VPN mask, Next PE IP Address, Internal label, Output Interface>
- In addition to the VRF, a PE stores a single ***Global Forwarding Table (GRT)*** which permits to reach a PE from another PE
  - a GRT entry contains the tuple: <PE IP address, external label, Output Interface>

High Level Architecture




VRF VPN-B
VPN B:1
192.168.0.0/24
.2
10.0.0.0/30
VPN NA/MASK:192.168.1.0/24
Gateway: 160.80.86.1
INTERNAL: L1 EXTERNAL: L3 iface: ge2
VPN NA/MASK:192.168.0.0/24
Gateway : 10.0.0.2
INTERNAL: - EXTERNAL: - Iface: ge0
VPN B:2
192.168.1.0/24
L3
L2
B1->B2
ge0
.1
PE1
ge2
160.80.86.0/24
PE2
.1
ge1
.5
160.80.86.1 Label L3
L3
L1
A1->A2
10.0.0.4/30
GRT
.6
VPN NA/MASK:192.168.1.0/24
Gateway : 160.80.86.1
INTERNAL: L1 EXTERNAL: L3 iface: ge2
VPN NA/MASK:192.168.0.0/24
Gateway : 10.0.0.6
INTERNAL: - EXTERNAL: - Iface: ge1
VPN A:2
192.168.1.0/24
VPN A:1
192.168.0.0/24
VRF VPN-A

Populating the GFT and the VRFs

- The Global Forwarding Table is configured by the provider during the set-up or the MPLS/VPN backbone (i.e. LSPs between PEs)
- The GFT can be populated manually (in the case of manual LSPs), or automatically in the case of a set-up with signalling protocols like LDP, RSVP-TE or CR-LDP
- VRFs contain two forwarding categories:
  - Forwarding to LOCAL sites
  - Forwarding to REMOTE sites
- Forwarding to local sites can be:
  - Manually configured
  - Obtained through specific routing protocols (OSPF, RIP, etc.), running the CE-PE link
- Remote routes are obtained through an extension of the BGP-4 protocol, namely Multi-Protocol interior BGP (***MP-iBGP***)

Populating the GFT and the VRFs

- VRFs are synchronized by exchanging the reachability info inside MP-iBGP announces
- An MP-iBGP announce is sent by a PE to all other PEs; an overlay full mesh between PEs must exist
- ***Assumption***: the cost of the direct hop between two PEs is 1, being this an IP level hop (not MPLS hop)
- A same MP-iBGP announce carries reachability information relative to prefixes of more VRFs

Route Distinguisher

- Thanks to MP-iBGP announces, the BGP engine inside the PE calculates the next-hop (and internal label) towards every announced prefix
- VRFs belonging to different VPNs can notify a same private prefix since the addressing spaces can be overlapped.
- To differentiate overlapped prefixes (i.e. make them different to the BGP engine), ***a VRF is identified by an ID named Route Distinguisher (64 bit)***
- Usually, all the VRFs of the same VPN use the same Route Distinguisher, since the prefixes inside a VPN cannot overlap.
- In this way, the Route Distinguisher can be reused

Route Distinguisher

- ❑ The RD is placed before the net_id in the MP-iBGP entries
- ❑ The routes computed by BGP are inserted inside the enabled VRFs (see Route Target next…)

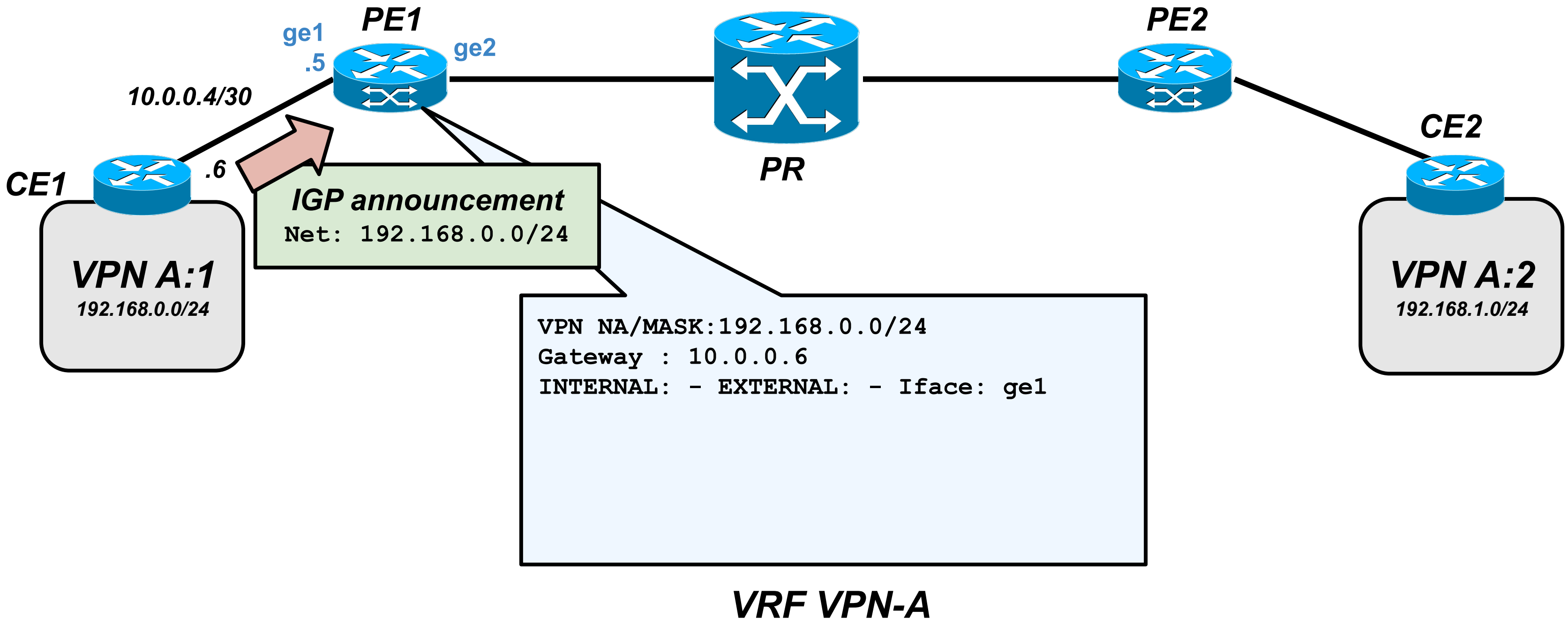
MP-iBGP announcements examples:

```
100:5:192.168.1.0/24 next-hop 160.80.86.1 int label 56 RT 100:1

100:9:192.168.1.0/24 next-hop 160.80.86.15 int label 32 RT 200:1
```

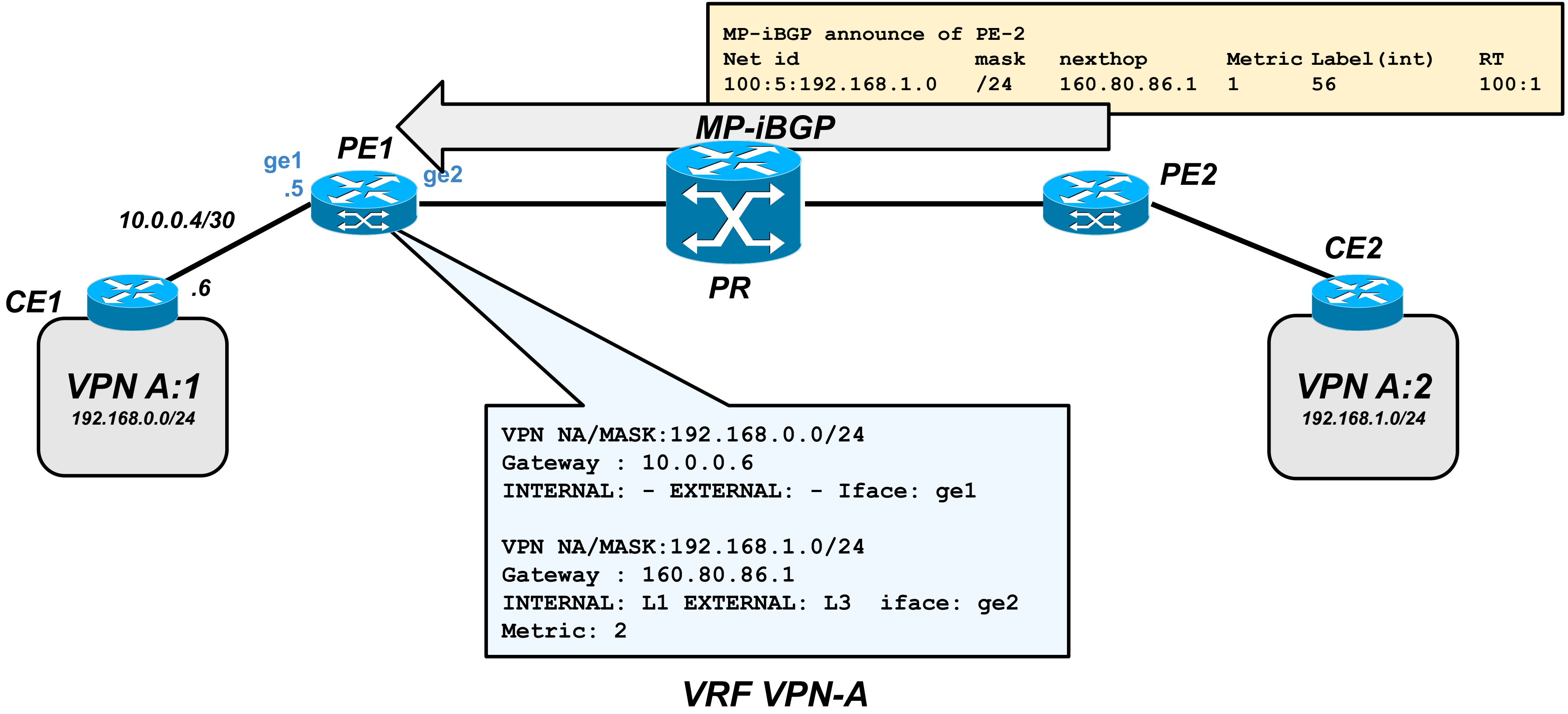
To accept the MP-iBGP announcements:

```
VRF RT import 100:1

VRF RT import 200:1
```

Populating VRFs: example



Populating VRFs: example



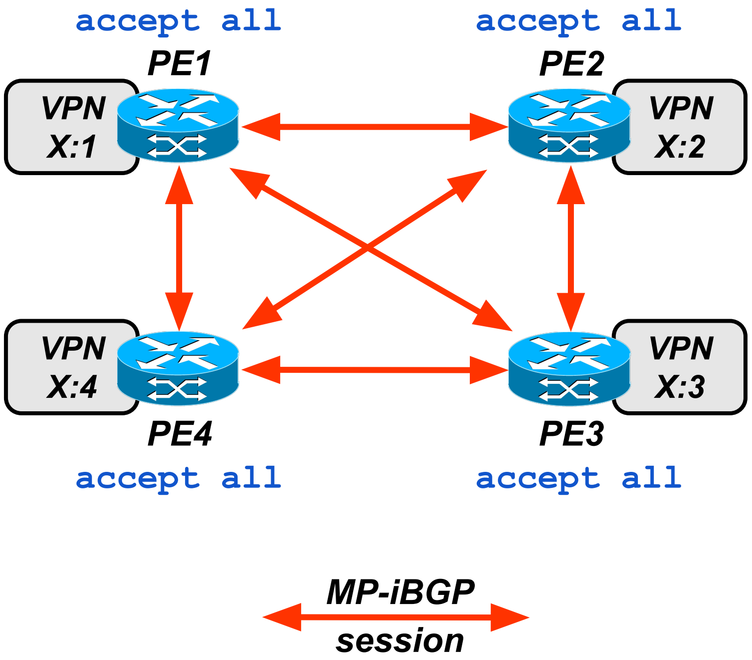
What about the VPN topology?

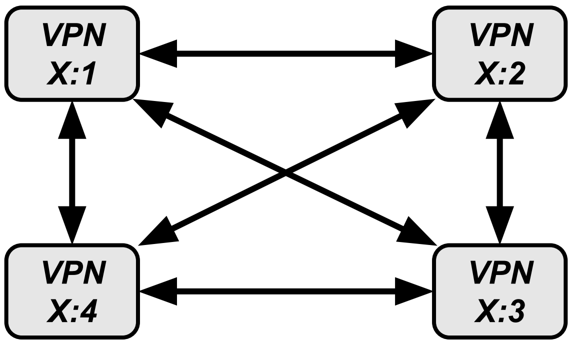
- If MP-iBGP messages are diffused among all PEs, all the VPNs have a full-mesh topology
- PROBLEM: what if I want different topologies for different VPNs?
- BGP principles say that if I have an overlay topology in which MP-iBGP messages are diffused, the (forwarding) topology of VPN-x is the set of the overlay shortest-paths between any couple of nodes
- Since direct connections between two PEs have metric 1, the VPN-x topology matches the overlay topology in which MP-iBGP messages are notified
- Therefore, if the overlay network in which MP-iBGP messages are forwarded is full-mesh, the VPN topology is full-mesh, too

What about the VPN topology?

- To change the logical topology of VPN-x it is necessary to change the MP-iBGP overlay network of VPN-x
- Solution 1: create a different MP-iBGP overlay forwarding topology for each VPN
  - High management effort, cannot aggregate inside the same MP-iBGP message the routing information relative to more VPNs, etc…
- ***Solution 2: keep the MP-iBGP full mesh and filter incoming announcements***
  - Having an overlay full-mesh for MP-iBGP common between PEs
  - Define the specific overlay needed for a given VPN
  - Flood MP-iBGP messages on the common MP-iBGP overlay
  - Receivers elaborate only announces coming from links of the specific overlay

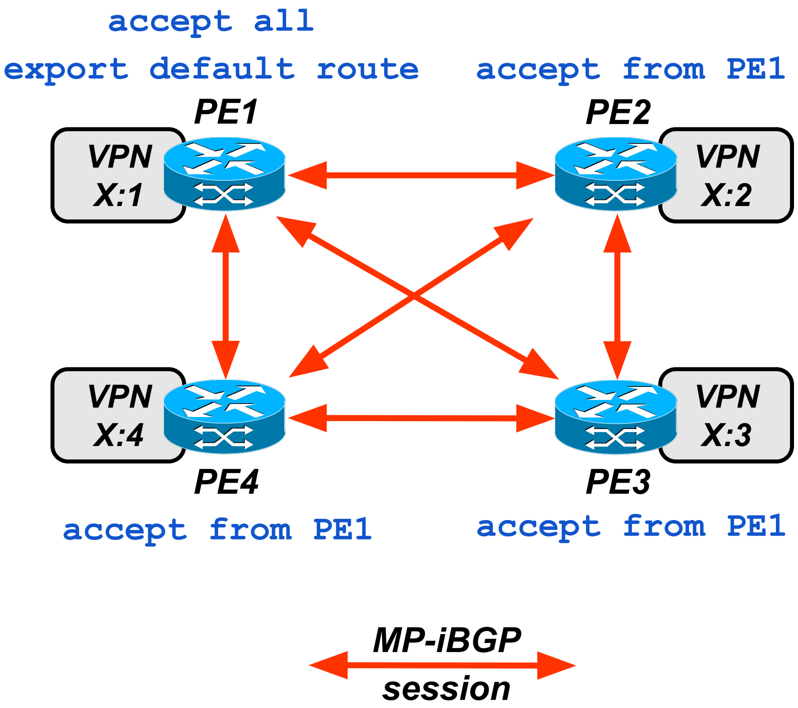
Populating VRFs - VPN Full Mesh

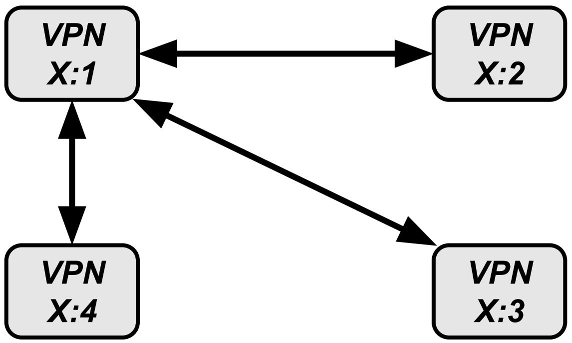




VPN Topology

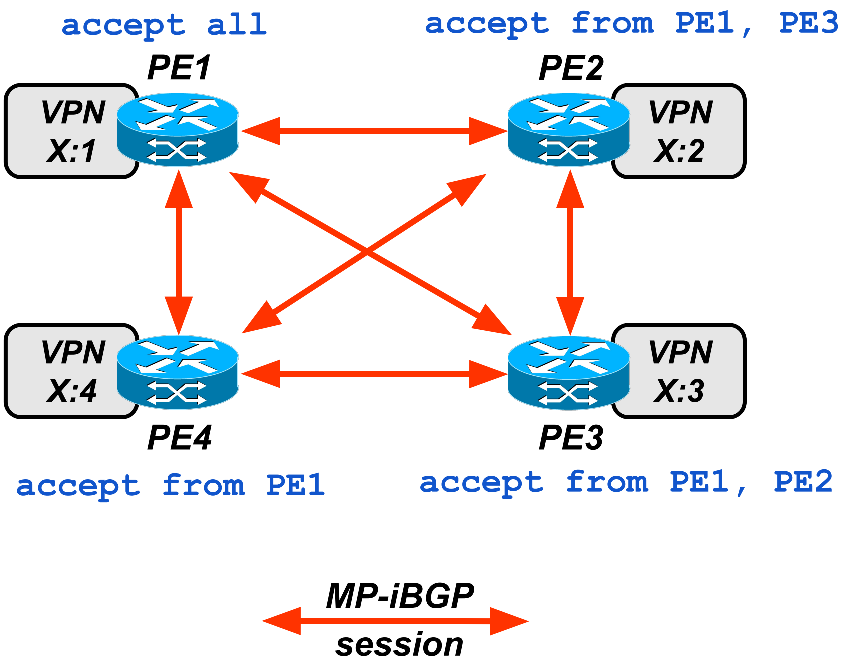
Populating VRFs - VPN Hub (X:1) and Spoke (X:2,3,4)

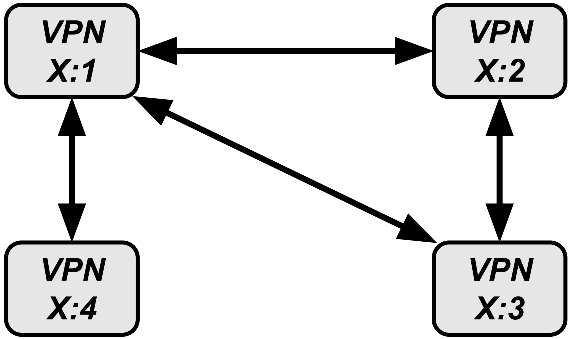




VPN Topology

Populating VRFs - VPN partial mesh



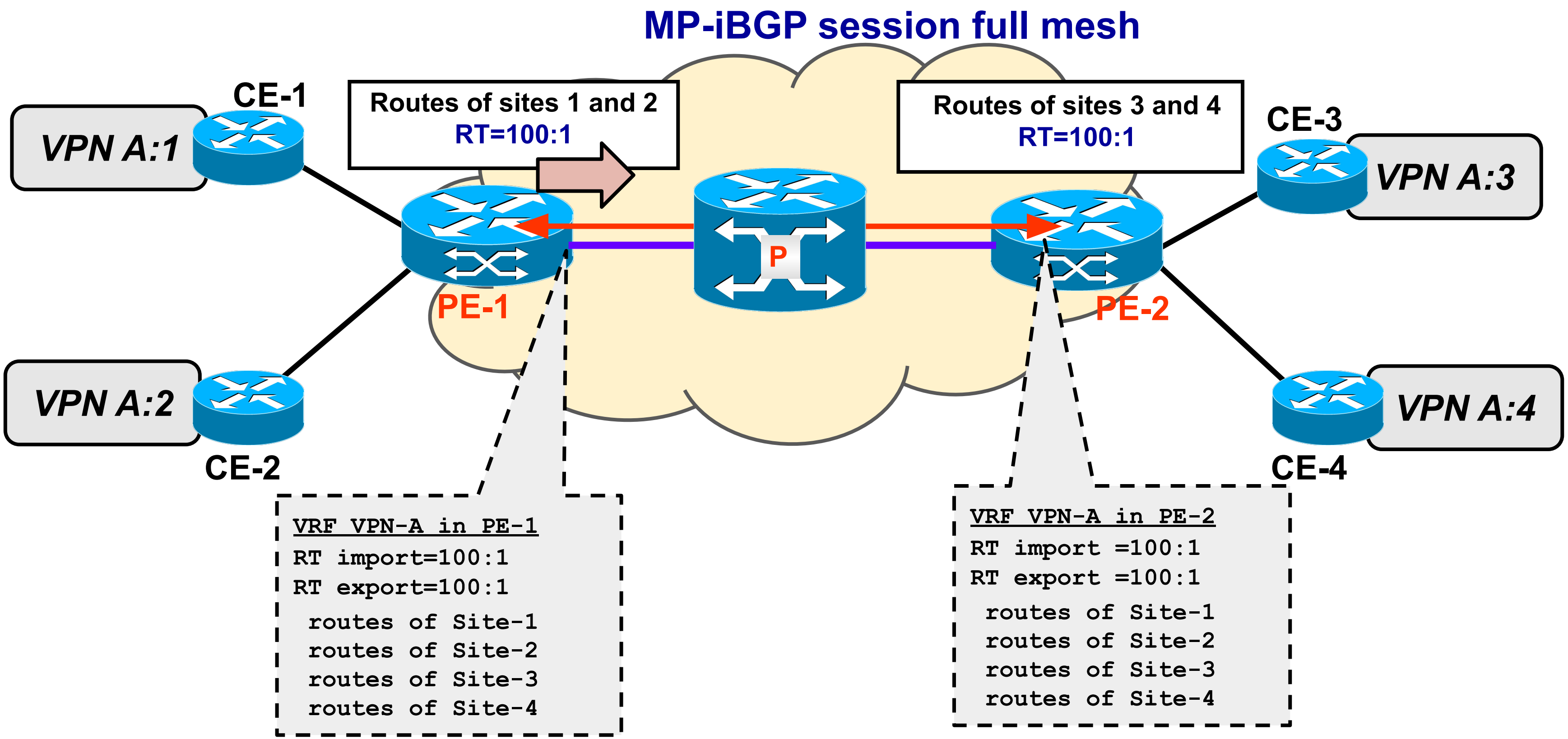


VPN Topology

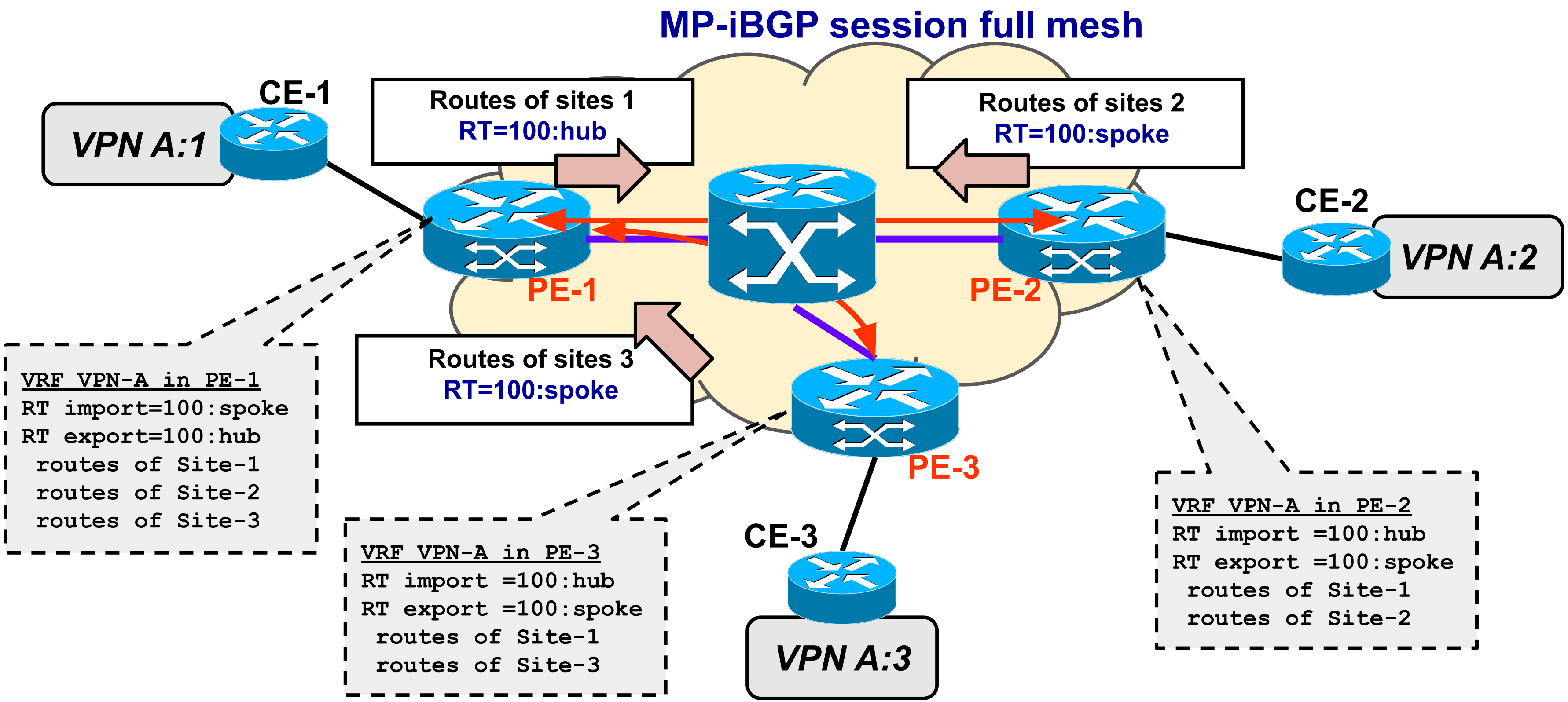
Route Target

- The Route Target concept permits to realize a specific overlay for the VPN-x discussed before. Therefore, permits to define VPN-x topology.
- It’s the VPN/MPLS “way” to tell to a VRF-x to “accept only a subset of MP-iBGP announces”
- How:
  - Each VRF transmitting announces, labels (exports) these announces with a configurable ID (Route target) of 64 bit size
  - Each VRF can receive (import) only announces with a configurable subset of Route Targets

Using the "Route Target": Example 1



Using the "Route Target": Example 2



VPN/MPLS configuration

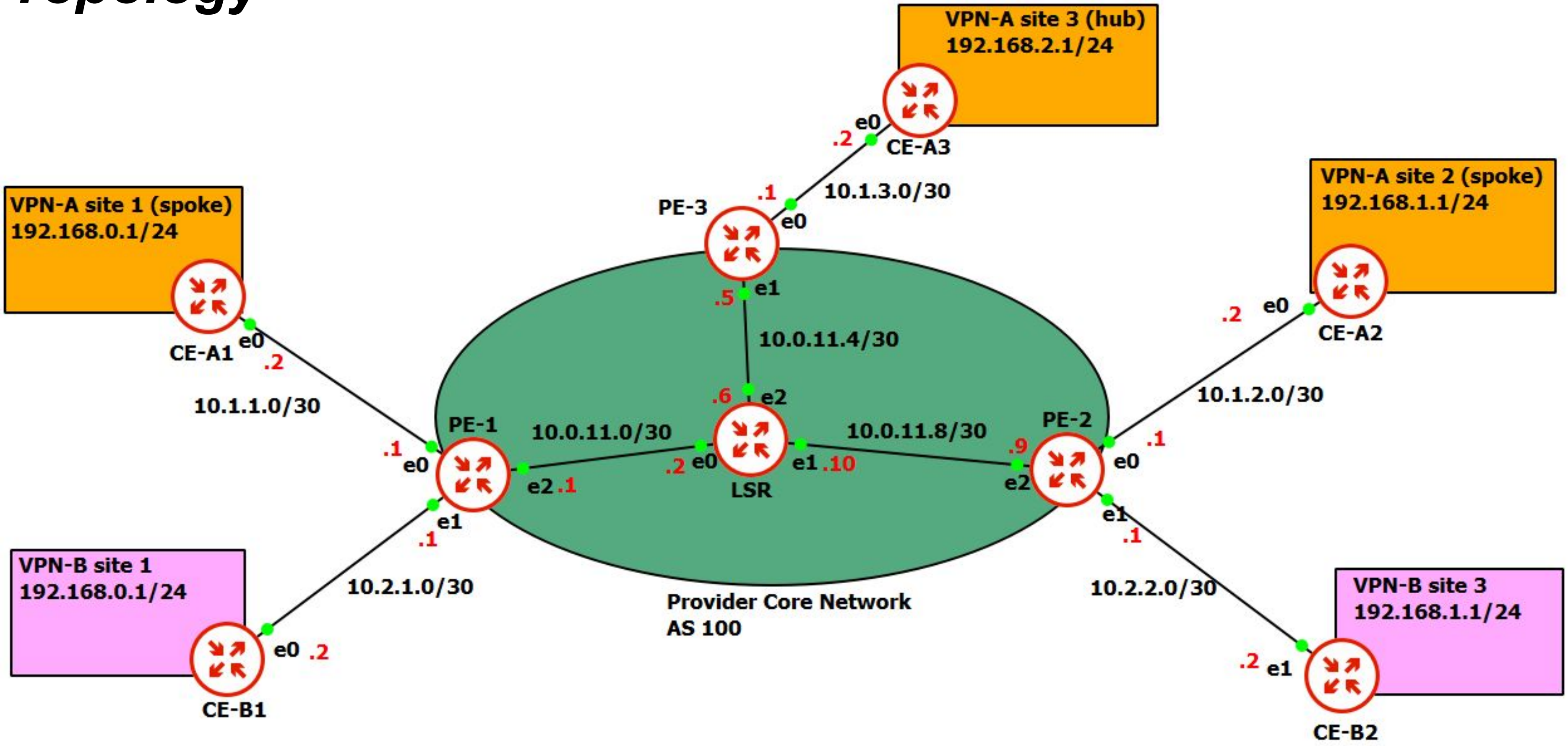
- Initialization
  - Configure LSP MPLS (e.g. with LDP) between all PEs
  - Enable BGP peering for prefixes of type VPNv4 (RD+net_id) between all PEs
- For each new VPN site
  - @ client
    - Notify to ISP the need of another VPN site and the relative topology
    - Install a CE as enterprise gateway
    - Configure the default gateway of the CE with the IP address of the access PE
    - Optional: enable on CE a routing protocol on the CE-PE path (e.g. OSPF)
  - @ provider
    - Initialize a new VRF on access PE
    - Define/Configure the Route Distinguisher
    - Define/Configure Route Import and Route Export and eventually update the import/export RTs on the other PEs, coherently with the requested topology
    - Associate the ingress PE interface with the VRF
    - Enable MP-iBGP on such VRF

Laboratory: BGP/MPLS VPN

NOTE for MPLS support

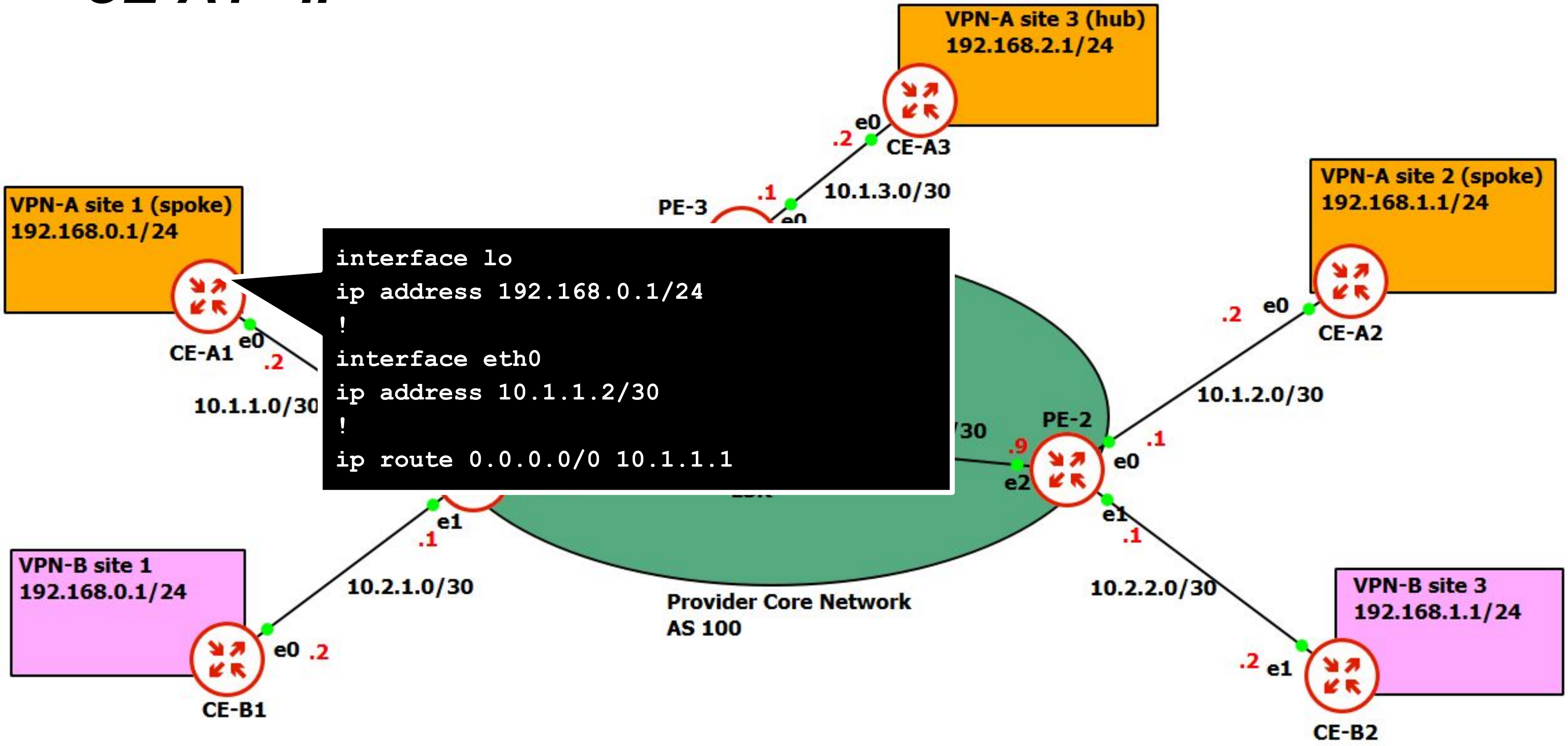
- In Linux, to support MPLS, we need two specific kernel modules implementing it
- These two modules are `mpls-router` and `mpls-iptunnel`
- Since we are using docker containers **in the GNS3 VM**, we have to insert this kernel modules in the GNS3 VM itself, which then will be inherited by the containers running on top of it
- To dynamically load the modules:
  - `modprobe mpls-router mpls-iptunnel`
- To permanently load the modules each time the VM is started:
  - login into the GNS3 VM
  - modify the `/etc/modules` file adding these two lines:
    - `mpls-router`
    - `mpls-iptunnel`
  - save and exit

Topology



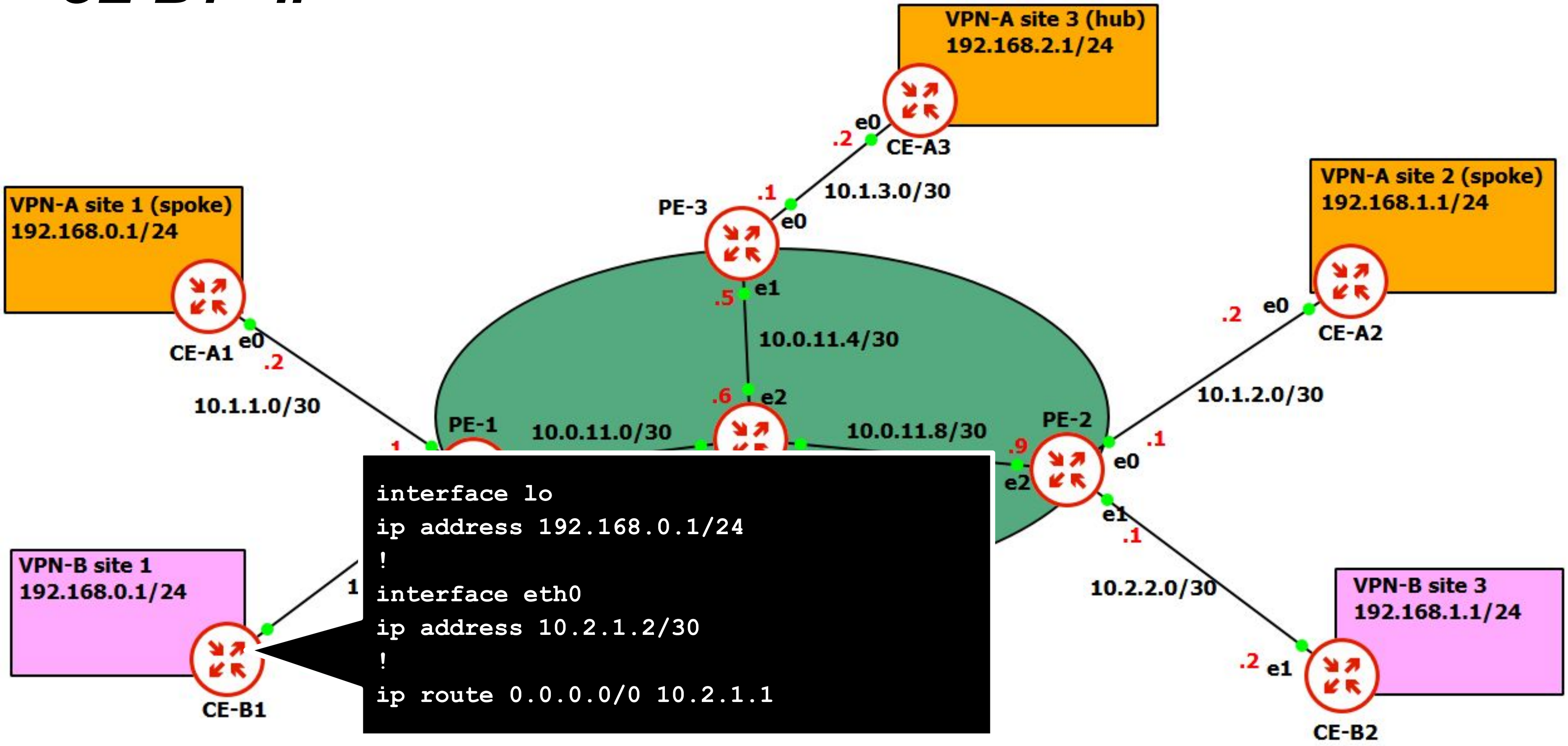
VPN-A site 3 (hub)
192.168.2.1/24
CE-A3
e0
.2
10.1.3.0/30
.1
e0
PE-3
e1
.5
10.0.11.4/30
VPN-A site 1 (spoke)
192.168.0.1/24
CE-A1
e0
.2
10.1.1.0/30
VPN-A site 2 (spoke)
192.168.1.1/24
CE-A2
e0
.2
10.1.2.0/30
.6
e2
PE-1
.1
e0
10.0.11.0/30
e2 .1
.2 e0
LSR
e1 .10
10.0.11.8/30
.9
e2
PE-2
.1
e0
e1
.1
e1
.1
VPN-B site 1
192.168.0.1/24
10.2.1.0/30
CE-B1
e0 .2
Provider Core Network
AS 100
10.2.2.0/30
VPN-B site 3
192.168.1.1/24
.2 e1
CE-B2

CE-A1 - IP



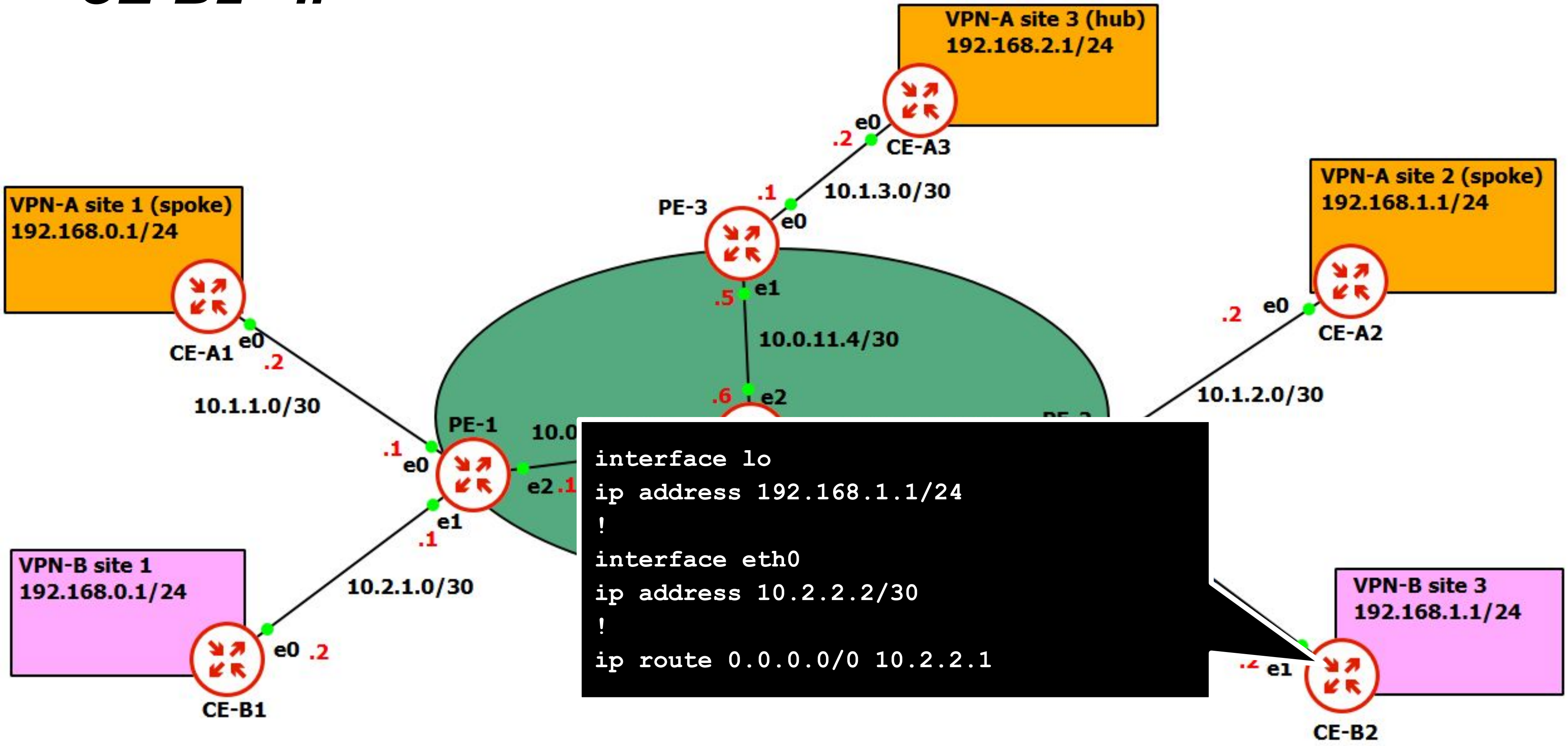

```
interface lo
ip address 192.168.0.1/24
!
interface eth0
ip address 10.1.1.2/30
!
ip route 0.0.0.0/0 10.1.1.1
```

CE-B1 - IP



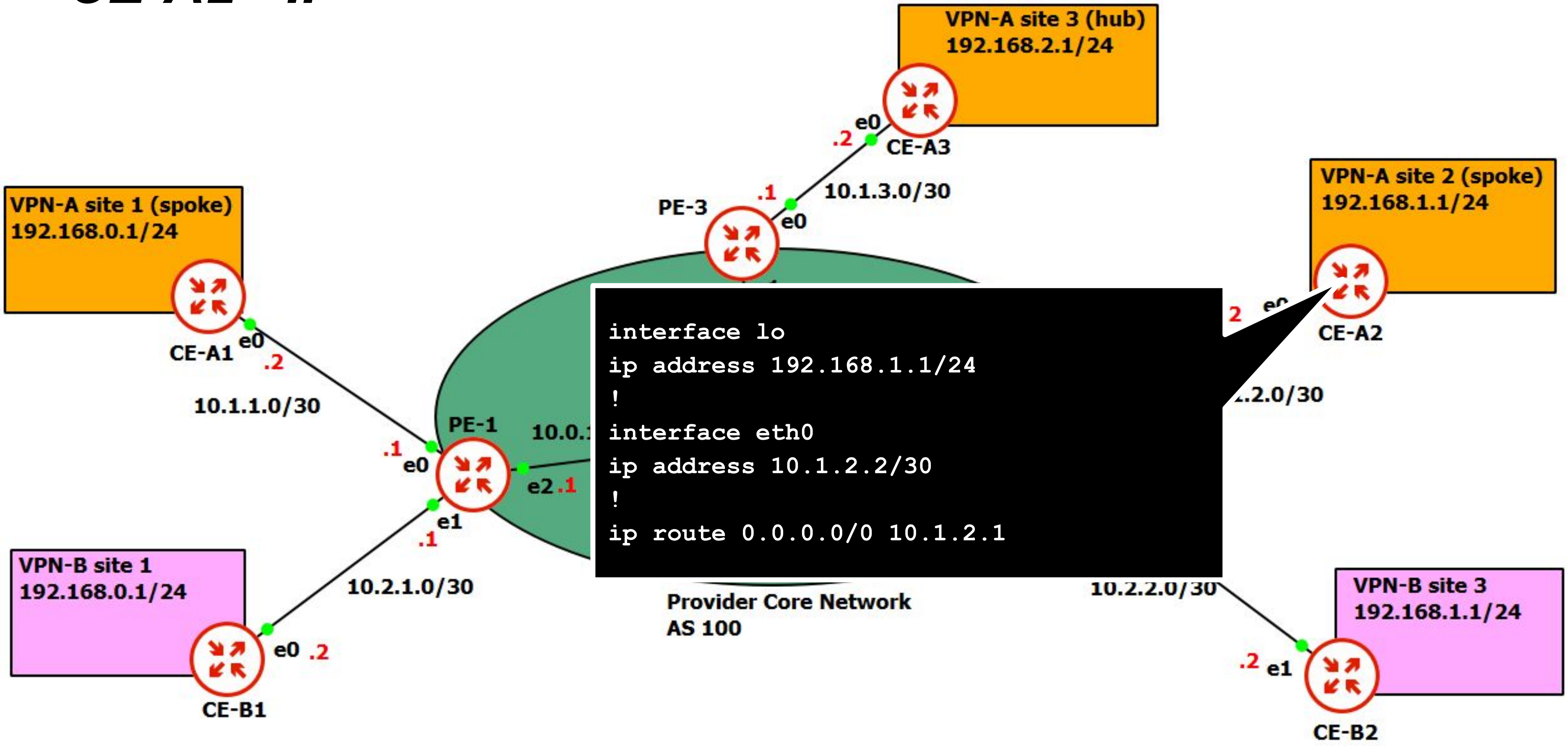

```
interface lo
ip address 192.168.0.1/24
!
interface eth0
ip address 10.2.1.2/30
!
ip route 0.0.0.0/0 10.2.1.1
```

CE-B2 - IP



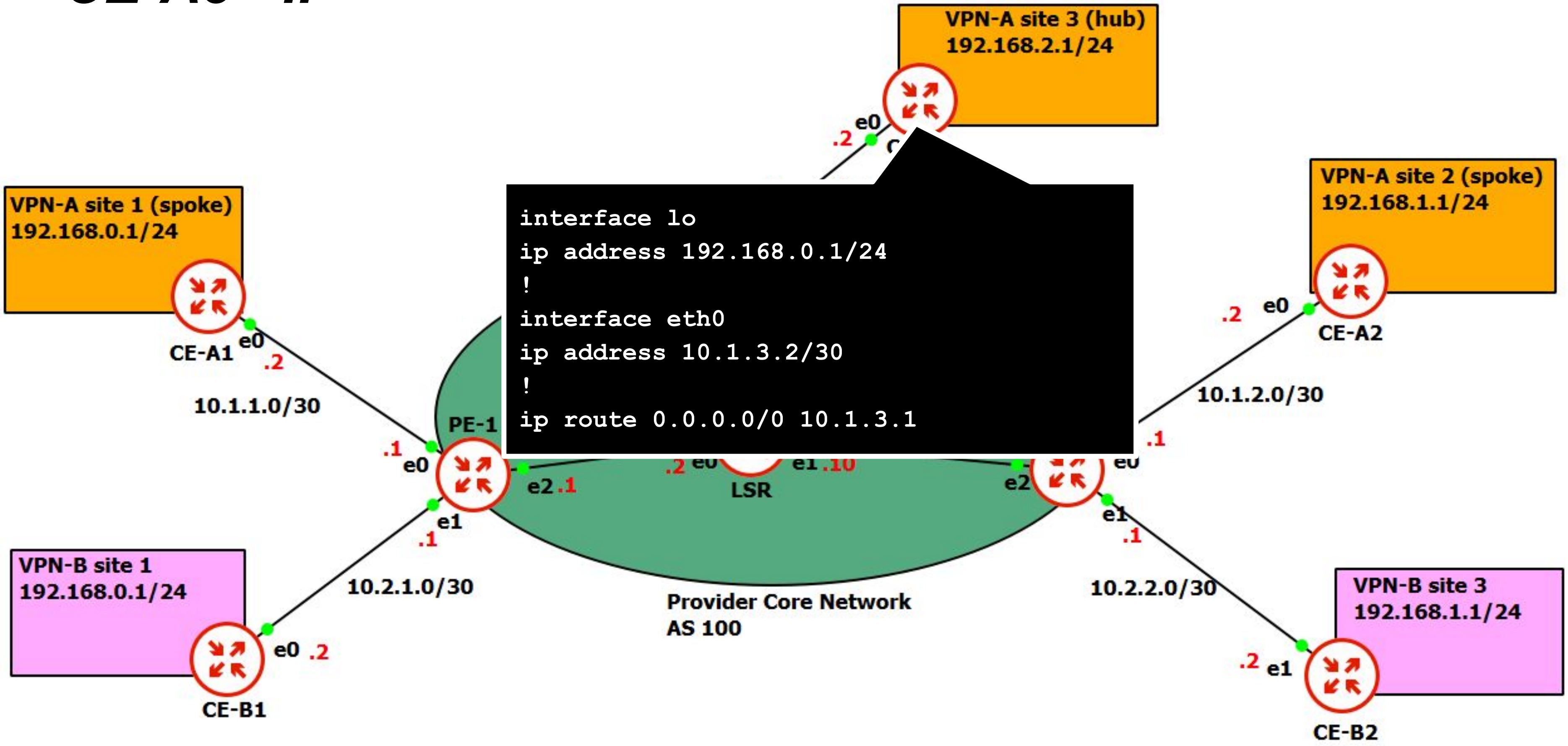

```
interface lo
ip address 192.168.1.1/24
!
interface eth0
ip address 10.2.2.2/30
!
ip route 0.0.0.0/0 10.2.2.1
```

CE-A2 - IP



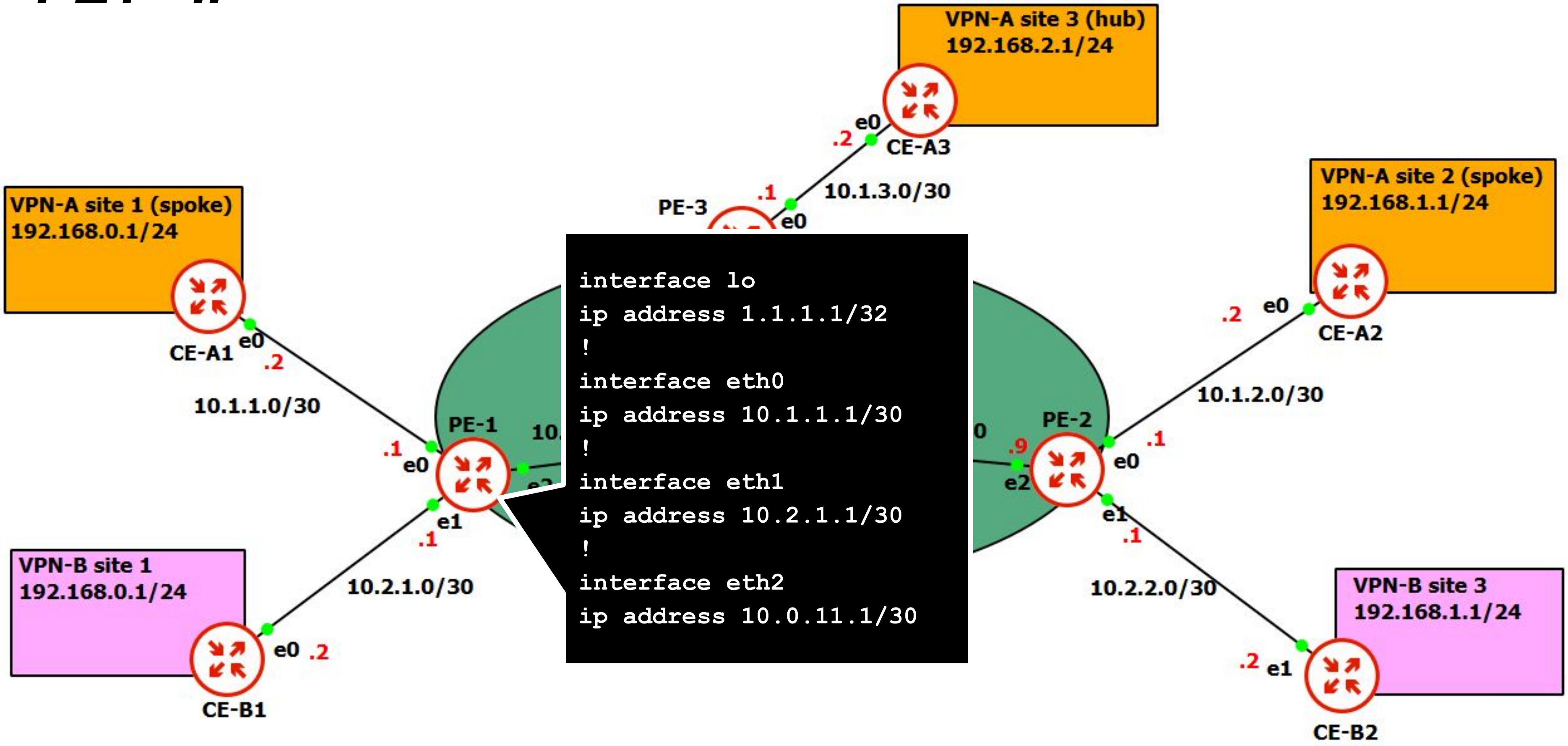

```
interface lo
ip address 192.168.1.1/24
!
interface eth0
ip address 10.1.2.2/30
!
ip route 0.0.0.0/0 10.1.2.1
```

CE-A3 - IP



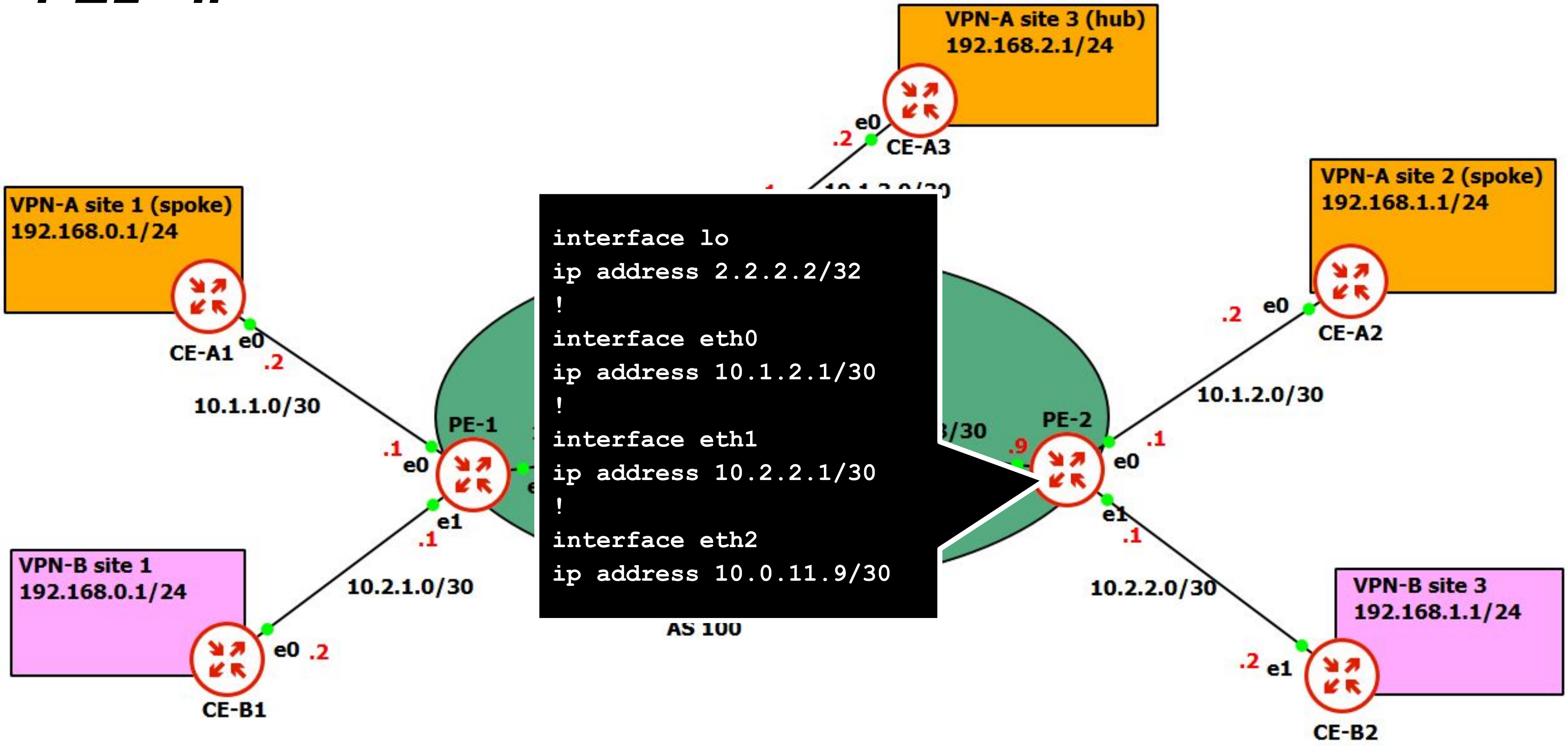

```
interface lo
ip address 192.168.0.1/24
!
interface eth0
ip address 10.1.3.2/30
!
ip route 0.0.0.0/0 10.1.3.1
```

PE1 - IP




```
interface lo
ip address 1.1.1.1/32
!
interface eth0
ip address 10.1.1.1/30
!
interface eth1
ip address 10.2.1.1/30
!
interface eth2
ip address 10.0.11.1/30
```

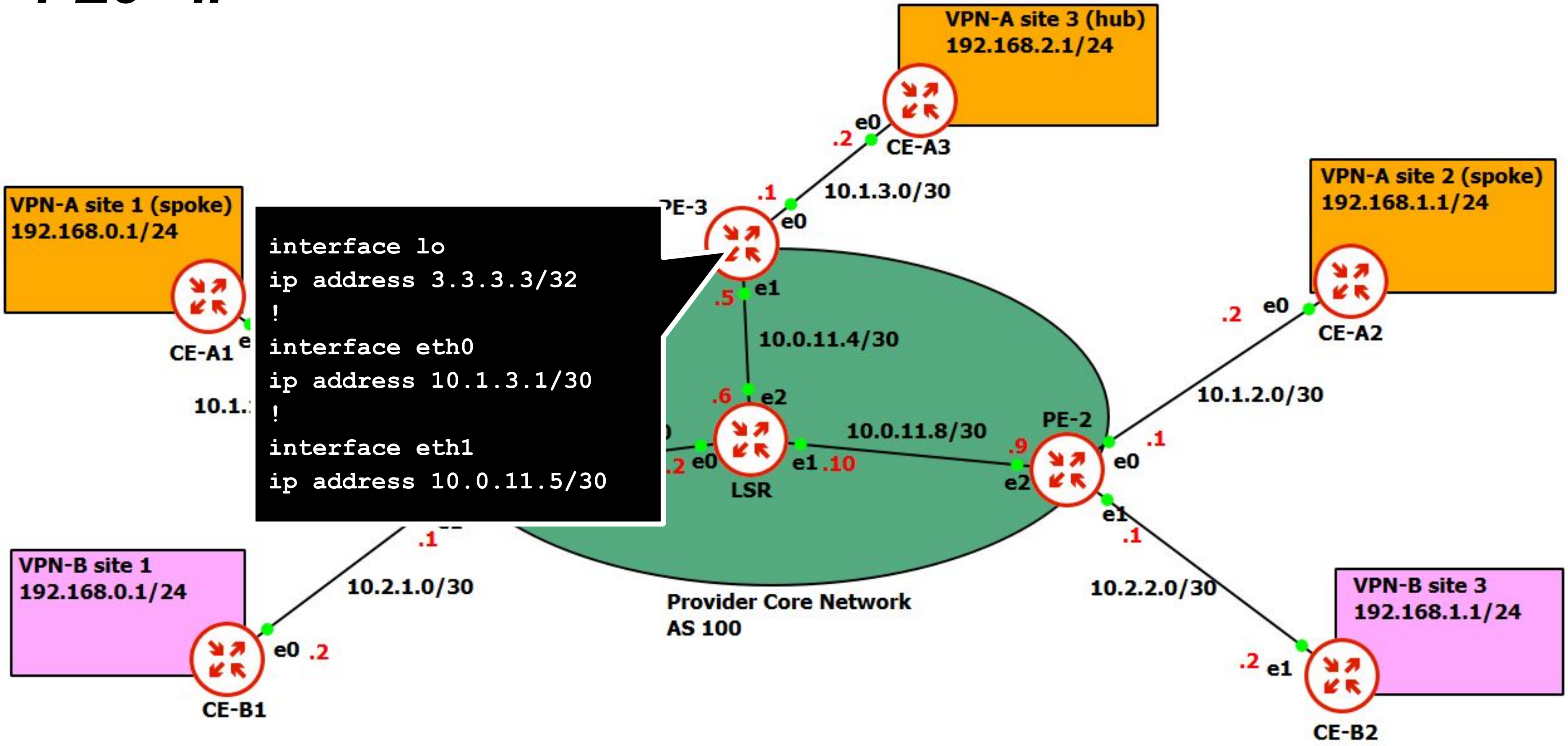
PE2 - IP



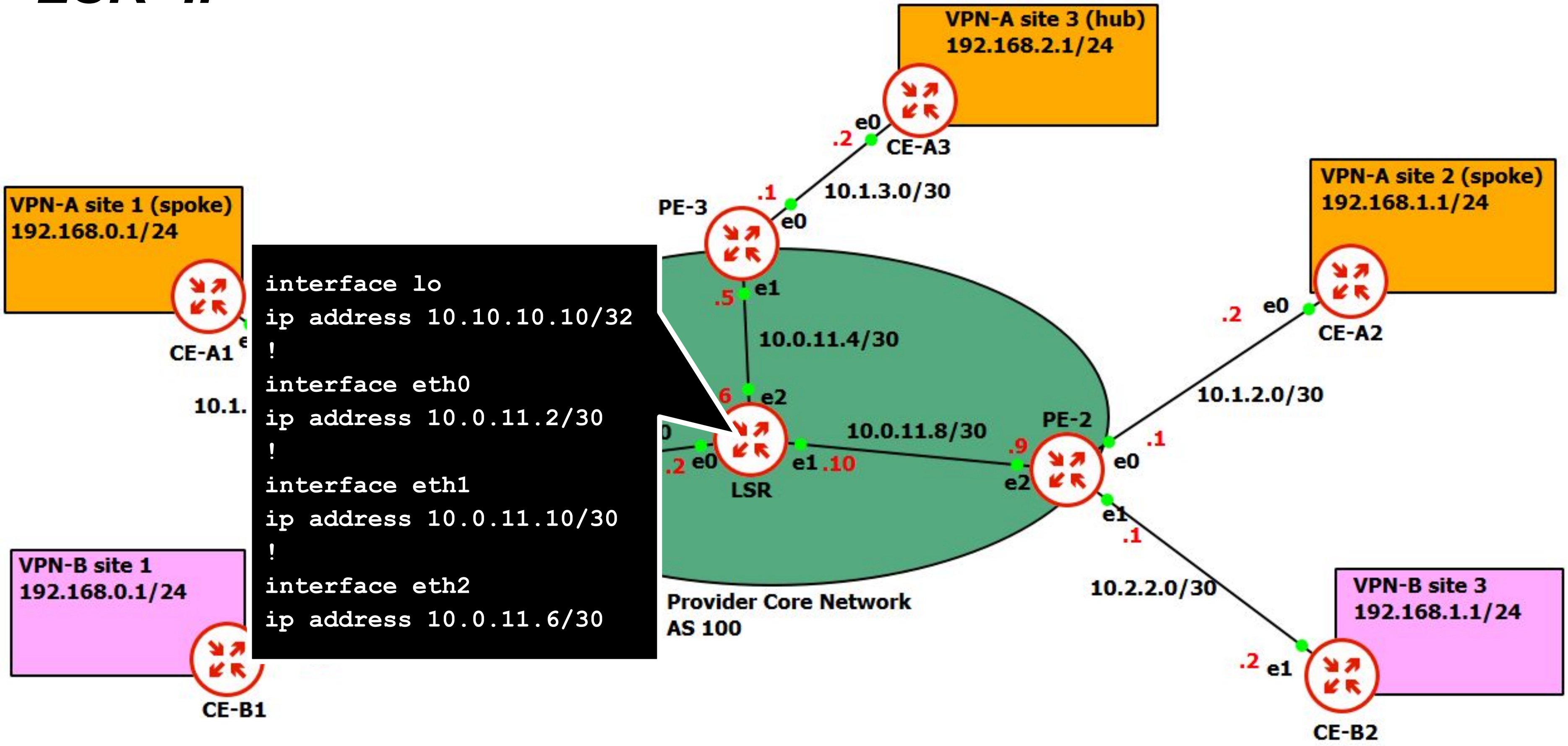

```
interface lo
ip address 2.2.2.2/32
!
interface eth0
ip address 10.1.2.1/30
!
interface eth1
ip address 10.2.2.1/30
!
interface eth2
ip address 10.0.11.9/30
```

PE3 - IP

```
interface lo
ip address 3.3.3.3/32
!
interface eth0
ip address 10.1.3.1/30
!
interface eth1
ip address 10.0.11.5/30
```

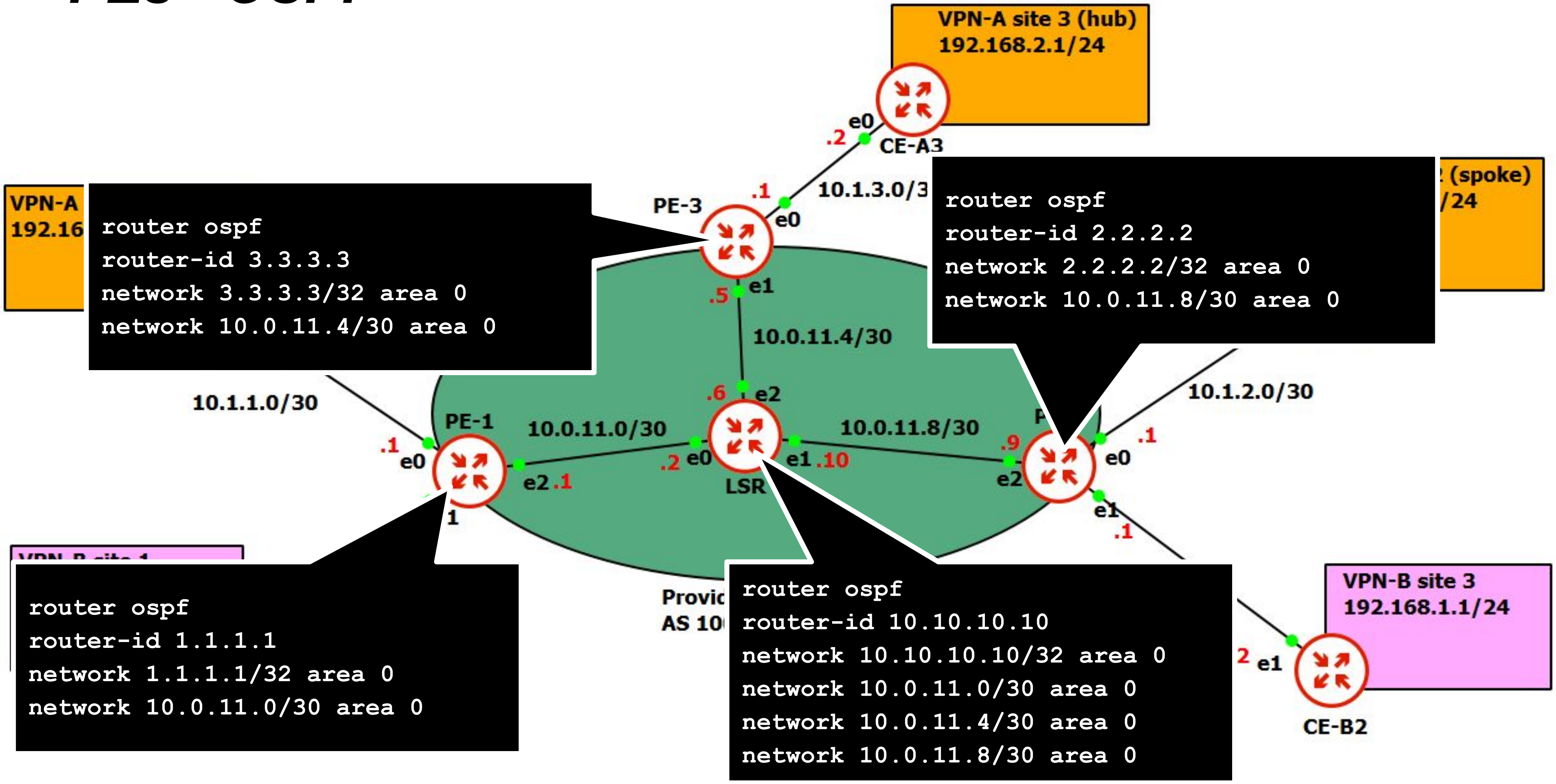


LSR- IP




```
interface lo
ip address 10.10.10.10/32
!
interface eth0
ip address 10.0.11.2/30
!
interface eth1
ip address 10.0.11.10/30
!
interface eth2
ip address 10.0.11.6/30
```

PEs - OSPF




```
router ospf
router-id 3.3.3.3
network 3.3.3.3/32 area 0
network 10.0.11.4/30 area 0
```

```
router ospf
router-id 2.2.2.2
network 2.2.2.2/32 area 0
network 10.0.11.8/30 area 0
```

```
router ospf
router-id 1.1.1.1
network 1.1.1.1/32 area 0
network 10.0.11.0/30 area 0
```

```
router ospf
router-id 10.10.10.10
network 10.10.10.10/32 area 0
network 10.0.11.0/30 area 0
network 10.0.11.4/30 area 0
network 10.0.11.8/30 area 0
```

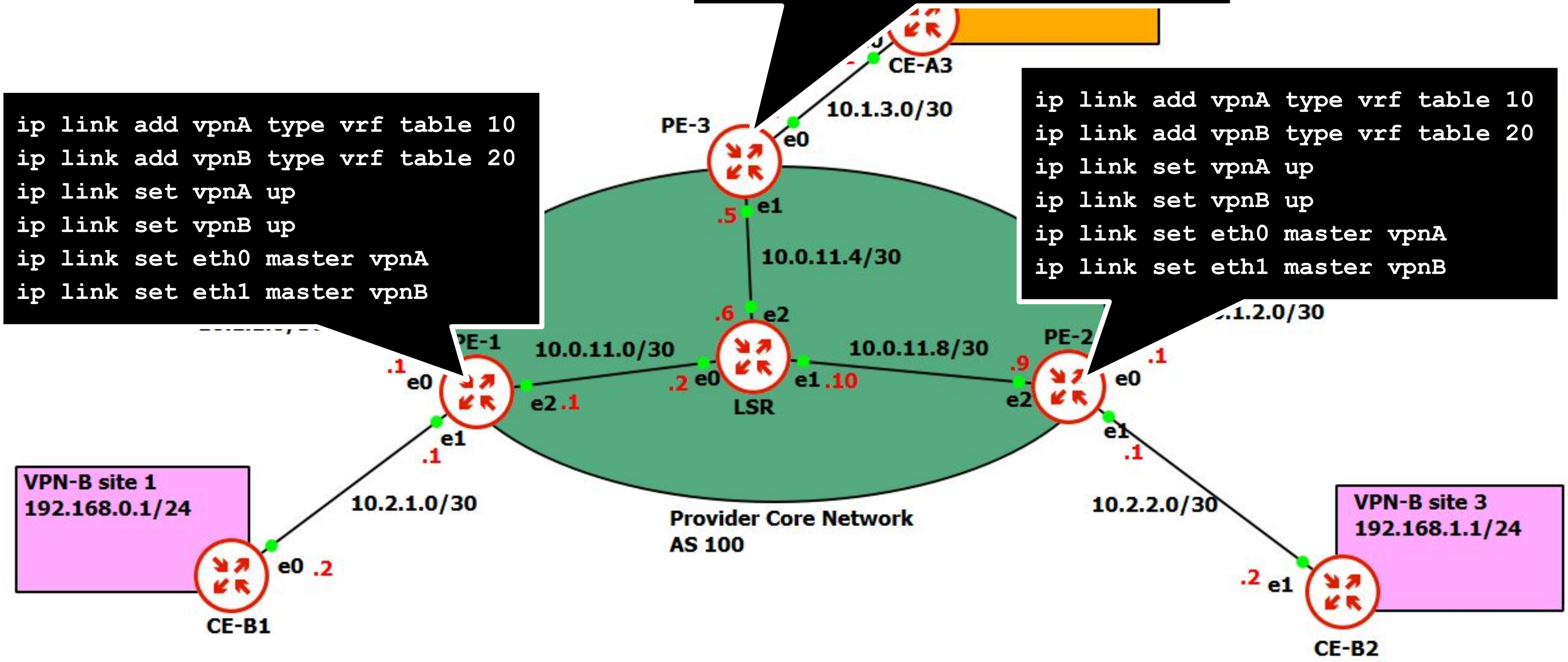
PEs - Define VRF

note: these are bash commands

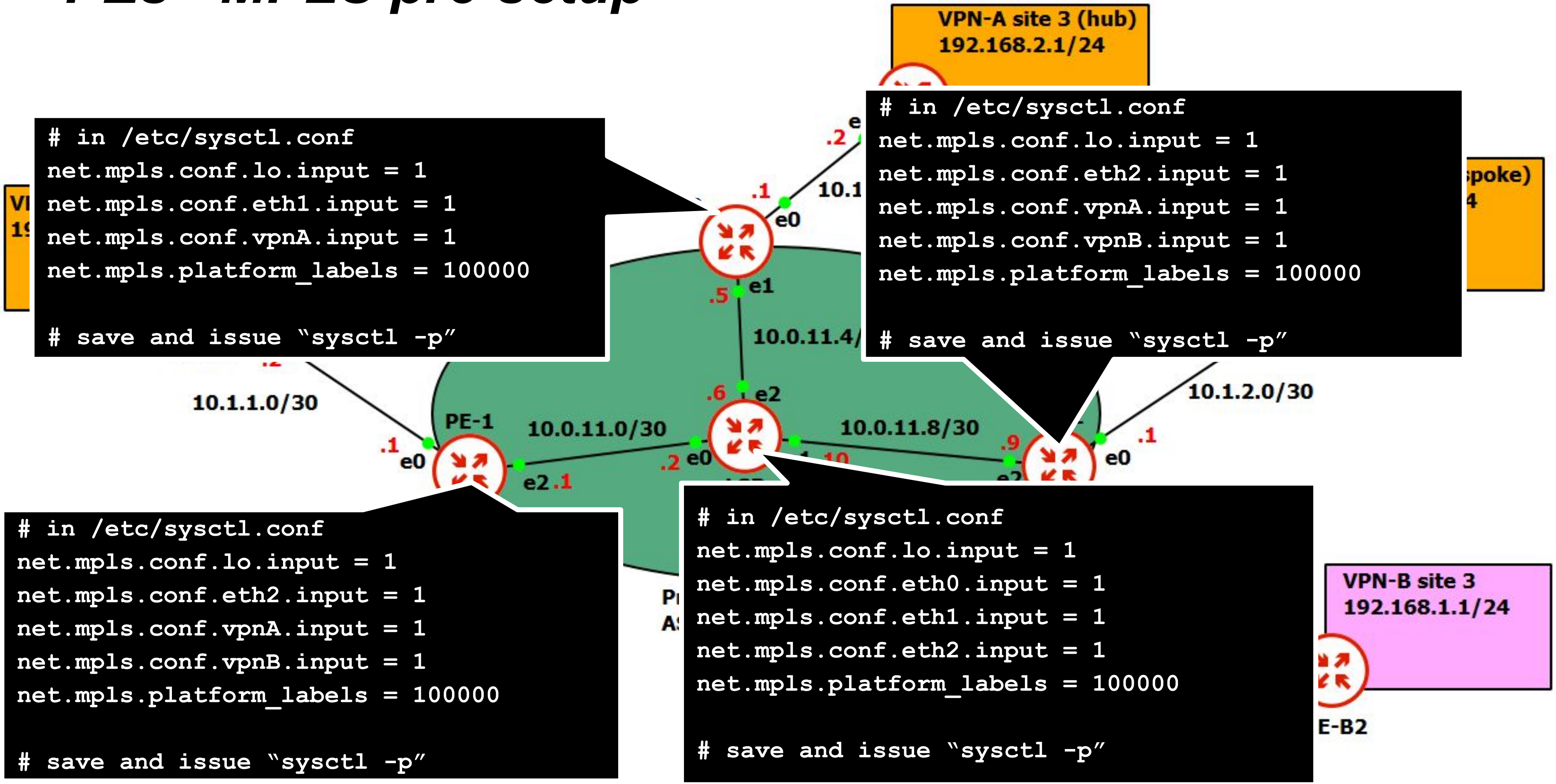
```
ip link add vpnA type vrf table 10
ip link set vpnA up
ip link set eth0 master vpnA
```

```
ip link add vpnA type vrf table 10
ip link add vpnB type vrf table 20
ip link set vpnA up
ip link set vpnB up
ip link set eth0 master vpnA
ip link set eth1 master vpnB
```

```
ip link add vpnA type vrf table 10
ip link add vpnB type vrf table 20
ip link set vpnA up
ip link set vpnB up
ip link set eth0 master vpnA
ip link set eth1 master vpnB
```



PEs - MPLS pre-setup



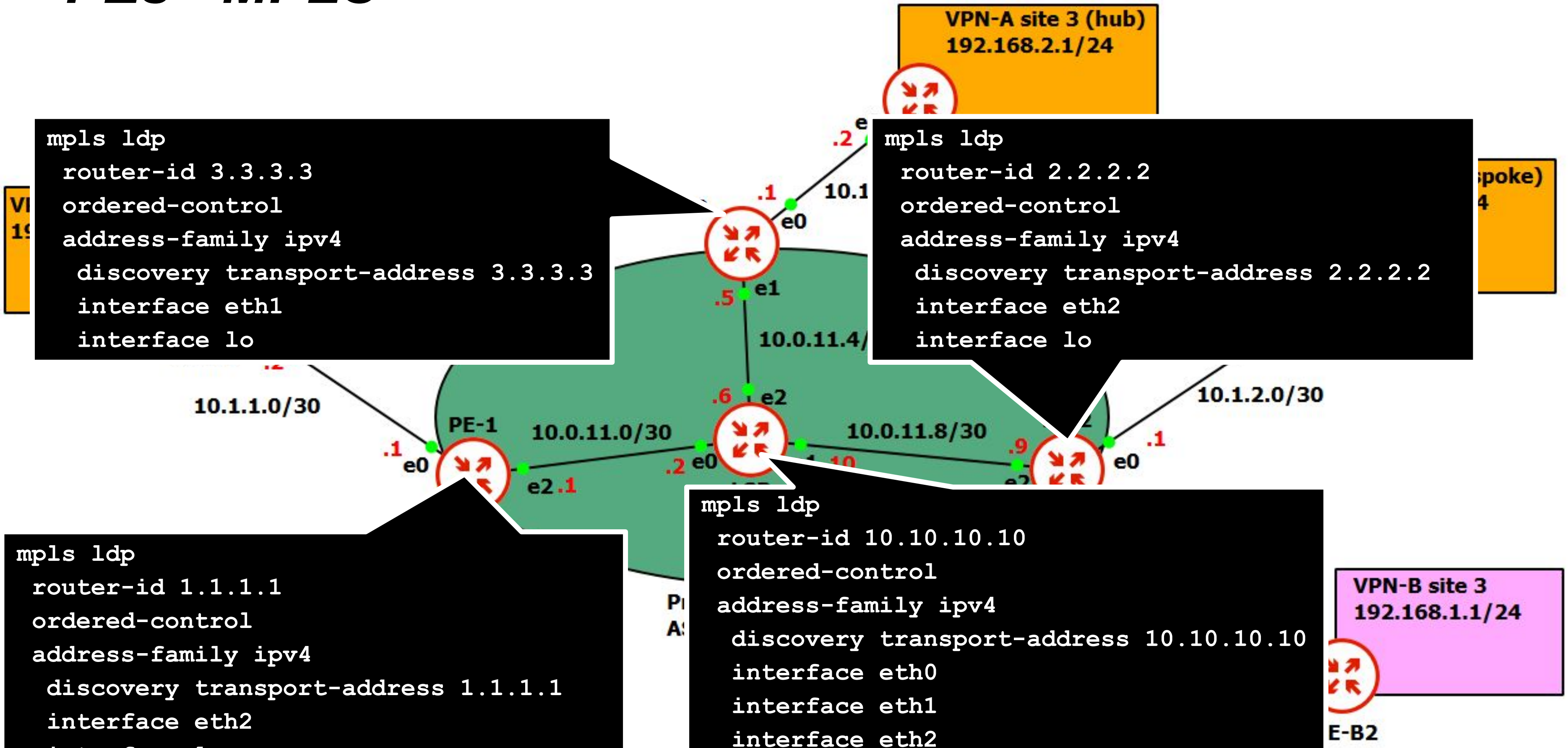

```
# in /etc/sysctl.conf
net.mpls.conf.lo.input = 1
net.mpls.conf.eth1.input = 1
net.mpls.conf.vpnA.input = 1
net.mpls.platform_labels = 100000

# save and issue "sysctl -p"
```

```
# in /etc/sysctl.conf
net.mpls.conf.lo.input = 1
net.mpls.conf.eth2.input = 1
net.mpls.conf.vpnA.input = 1
net.mpls.conf.vpnB.input = 1
net.mpls.platform_labels = 100000

# save and issue "sysctl -p"
```

```
# in /etc/sysctl.conf
net.mpls.conf.lo.input = 1
net.mpls.conf.eth2.input = 1
net.mpls.conf.vpnA.input = 1
net.mpls.conf.vpnB.input = 1
net.mpls.platform_labels = 100000

# save and issue "sysctl -p"
```

```
# in /etc/sysctl.conf
net.mpls.conf.lo.input = 1
net.mpls.conf.eth0.input = 1
net.mpls.conf.eth1.input = 1
net.mpls.conf.eth2.input = 1
net.mpls.platform_labels = 100000

# save and issue "sysctl -p"
```

PEs - MPLS



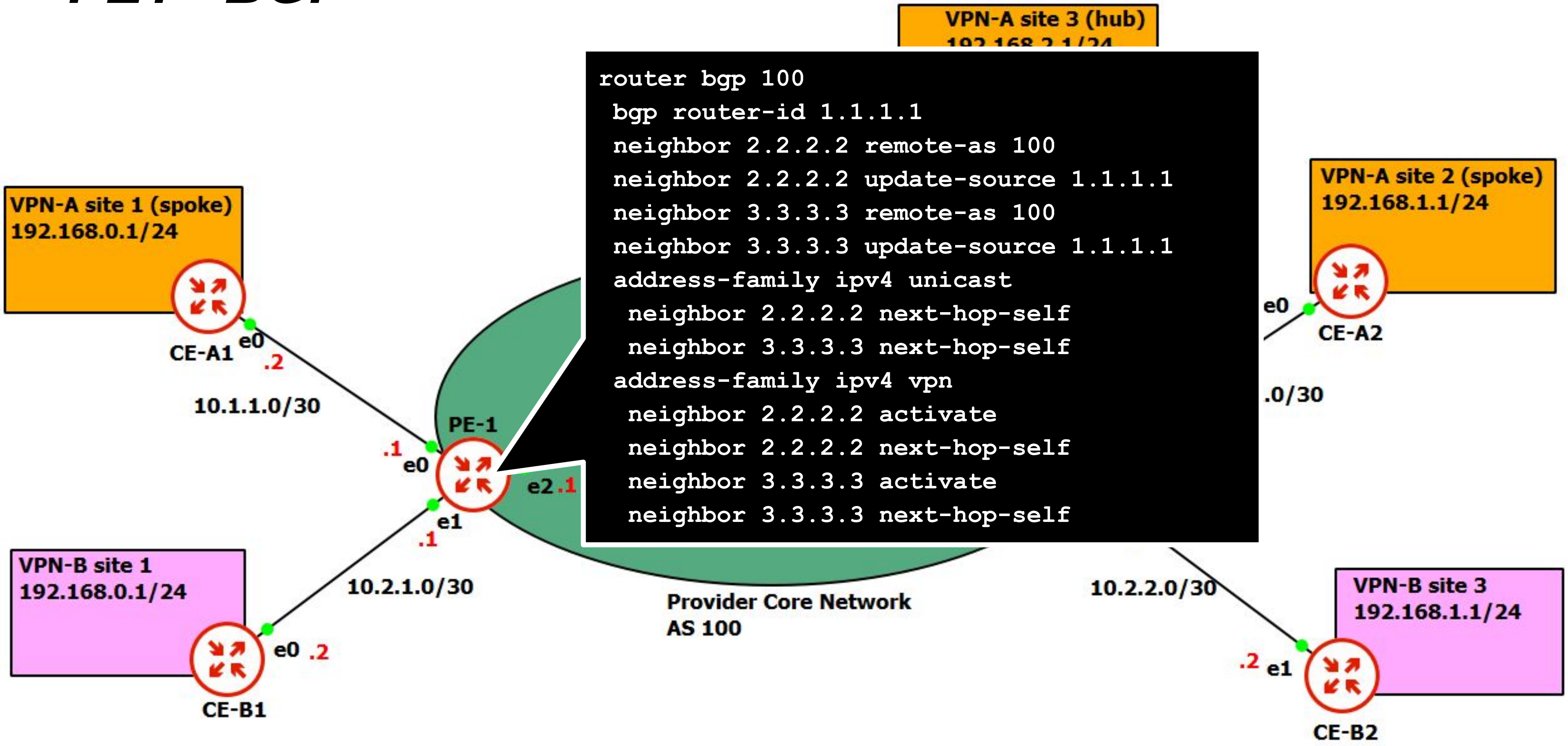

```
mpls ldp
 router-id 3.3.3.3
 ordered-control
 address-family ipv4
  discovery transport-address 3.3.3.3
  interface eth1
  interface lo
```

```
mpls ldp
 router-id 2.2.2.2
 ordered-control
 address-family ipv4
  discovery transport-address 2.2.2.2
  interface eth2
  interface lo
```

```
mpls ldp
 router-id 1.1.1.1
 ordered-control
 address-family ipv4
  discovery transport-address 1.1.1.1
  interface eth2
  interface lo
```

```
mpls ldp
 router-id 10.10.10.10
 ordered-control
 address-family ipv4
  discovery transport-address 10.10.10.10
  interface eth0
  interface eth1
  interface eth2
  interface lo
```

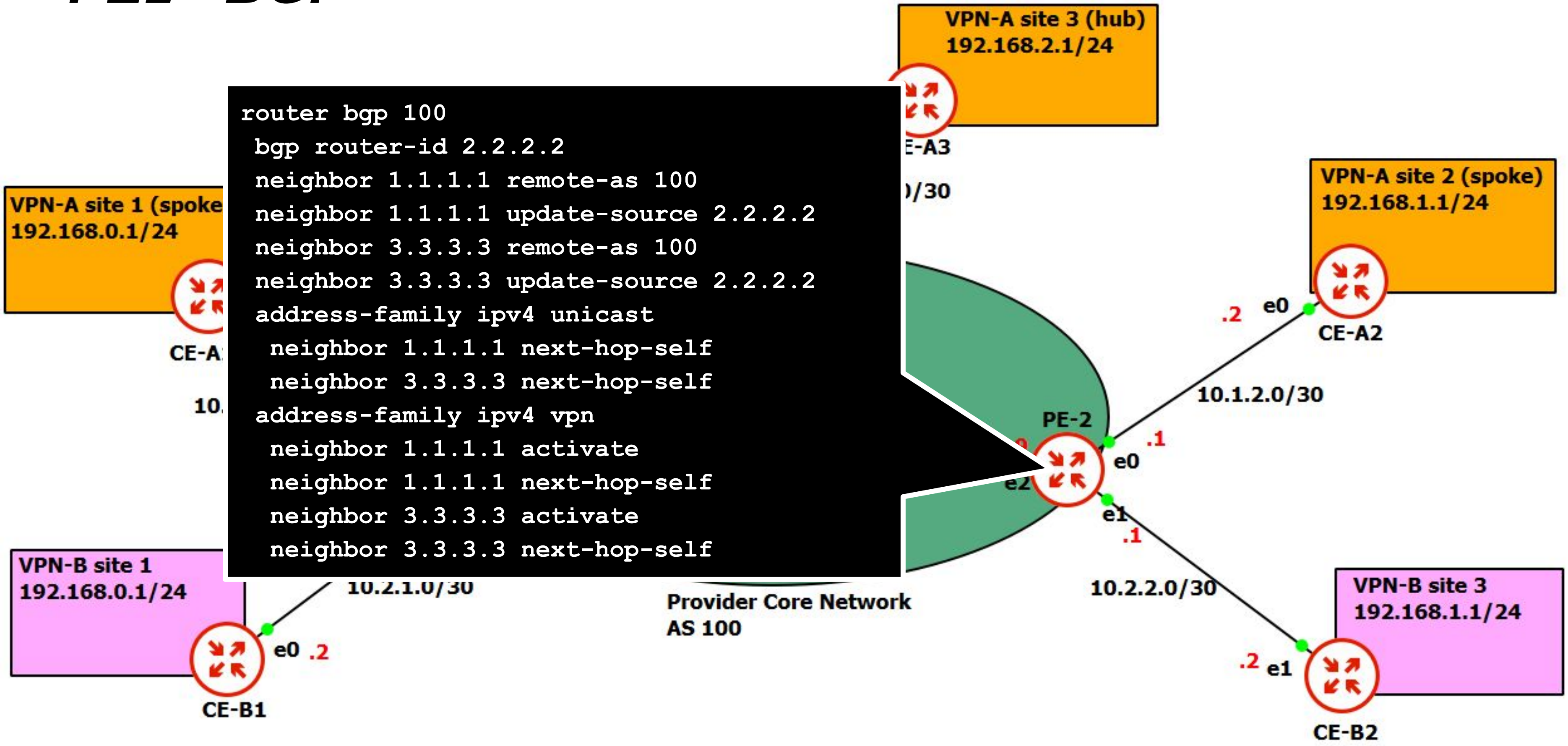
PE1 - BGP



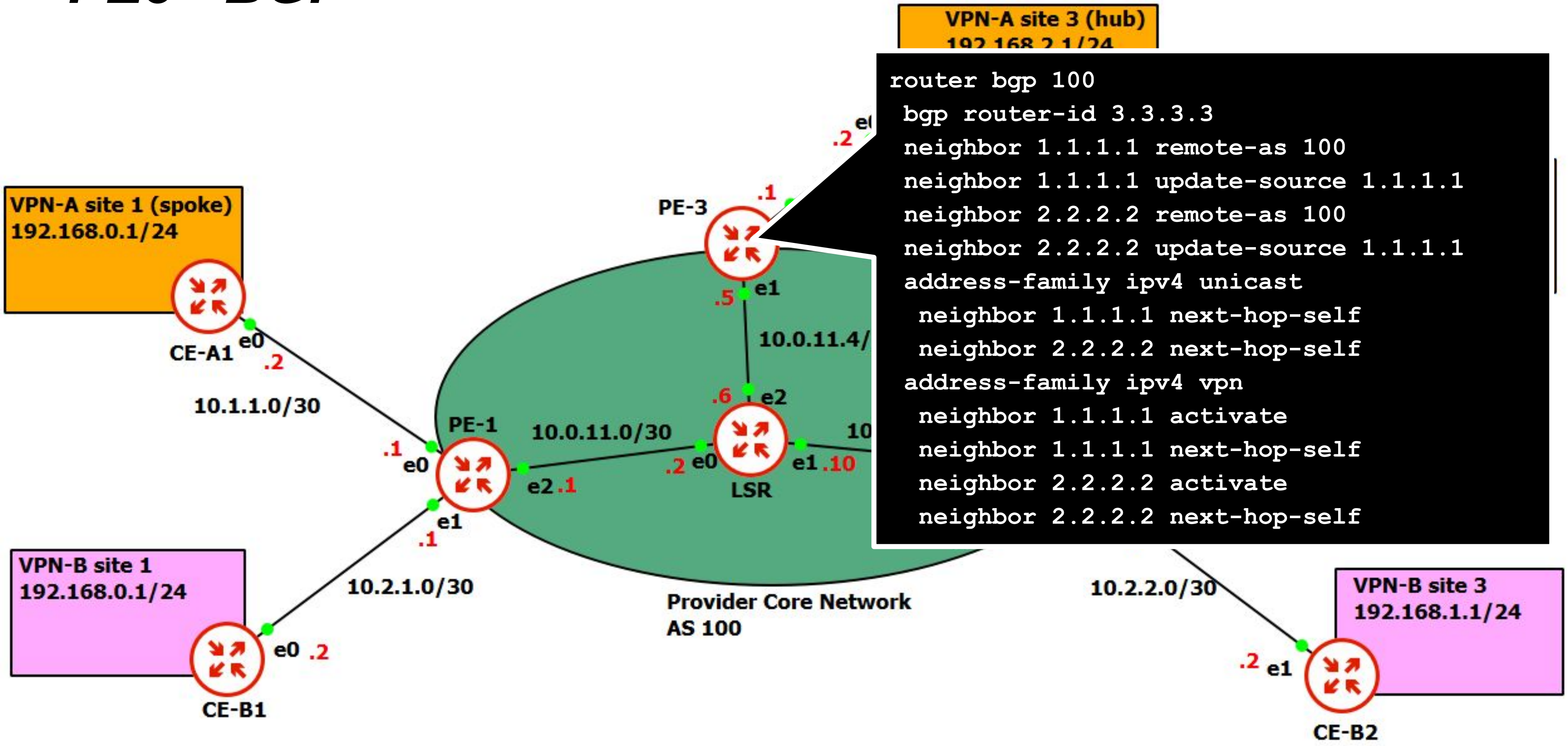

```
router bgp 100
 bgp router-id 1.1.1.1
 neighbor 2.2.2.2 remote-as 100
 neighbor 2.2.2.2 update-source 1.1.1.1
 neighbor 3.3.3.3 remote-as 100
 neighbor 3.3.3.3 update-source 1.1.1.1
 address-family ipv4 unicast
  neighbor 2.2.2.2 next-hop-self
  neighbor 3.3.3.3 next-hop-self
 address-family ipv4 vpn
  neighbor 2.2.2.2 activate
  neighbor 2.2.2.2 next-hop-self
  neighbor 3.3.3.3 activate
  neighbor 3.3.3.3 next-hop-self
```

PE2 - BGP

```
router bgp 100
 bgp router-id 2.2.2.2
 neighbor 1.1.1.1 remote-as 100
 neighbor 1.1.1.1 update-source 2.2.2.2
 neighbor 3.3.3.3 remote-as 100
 neighbor 3.3.3.3 update-source 2.2.2.2
 address-family ipv4 unicast
  neighbor 1.1.1.1 next-hop-self
  neighbor 3.3.3.3 next-hop-self
 address-family ipv4 vpn
  neighbor 1.1.1.1 activate
  neighbor 1.1.1.1 next-hop-self
  neighbor 3.3.3.3 activate
  neighbor 3.3.3.3 next-hop-self
```

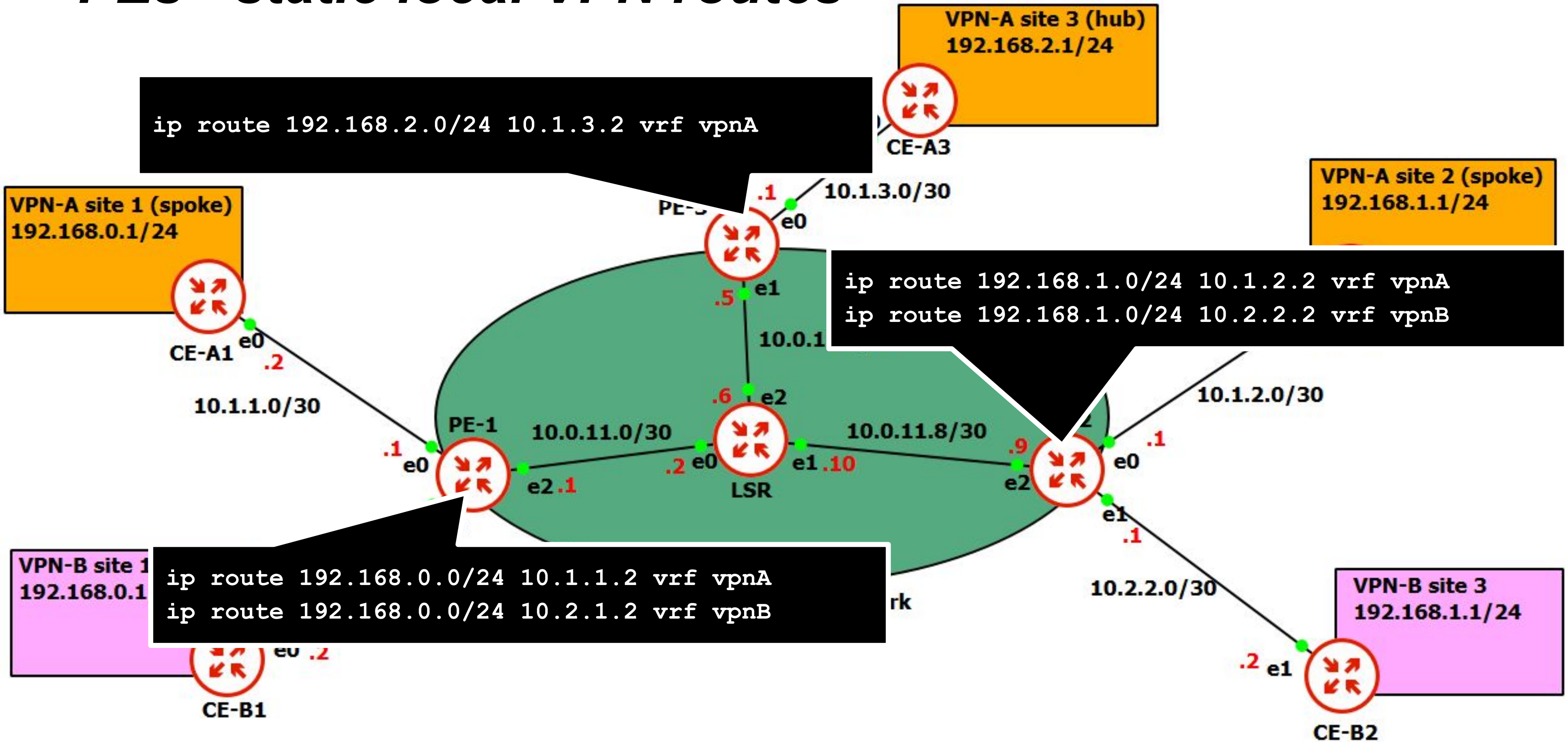


PE3 - BGP




```
router bgp 100
 bgp router-id 3.3.3.3
 neighbor 1.1.1.1 remote-as 100
 neighbor 1.1.1.1 update-source 1.1.1.1
 neighbor 2.2.2.2 remote-as 100
 neighbor 2.2.2.2 update-source 1.1.1.1
 address-family ipv4 unicast
  neighbor 1.1.1.1 next-hop-self
  neighbor 2.2.2.2 next-hop-self
 address-family ipv4 vpn
  neighbor 1.1.1.1 activate
  neighbor 1.1.1.1 next-hop-self
  neighbor 2.2.2.2 activate
  neighbor 2.2.2.2 next-hop-self
```

PEs - static local VPN routes

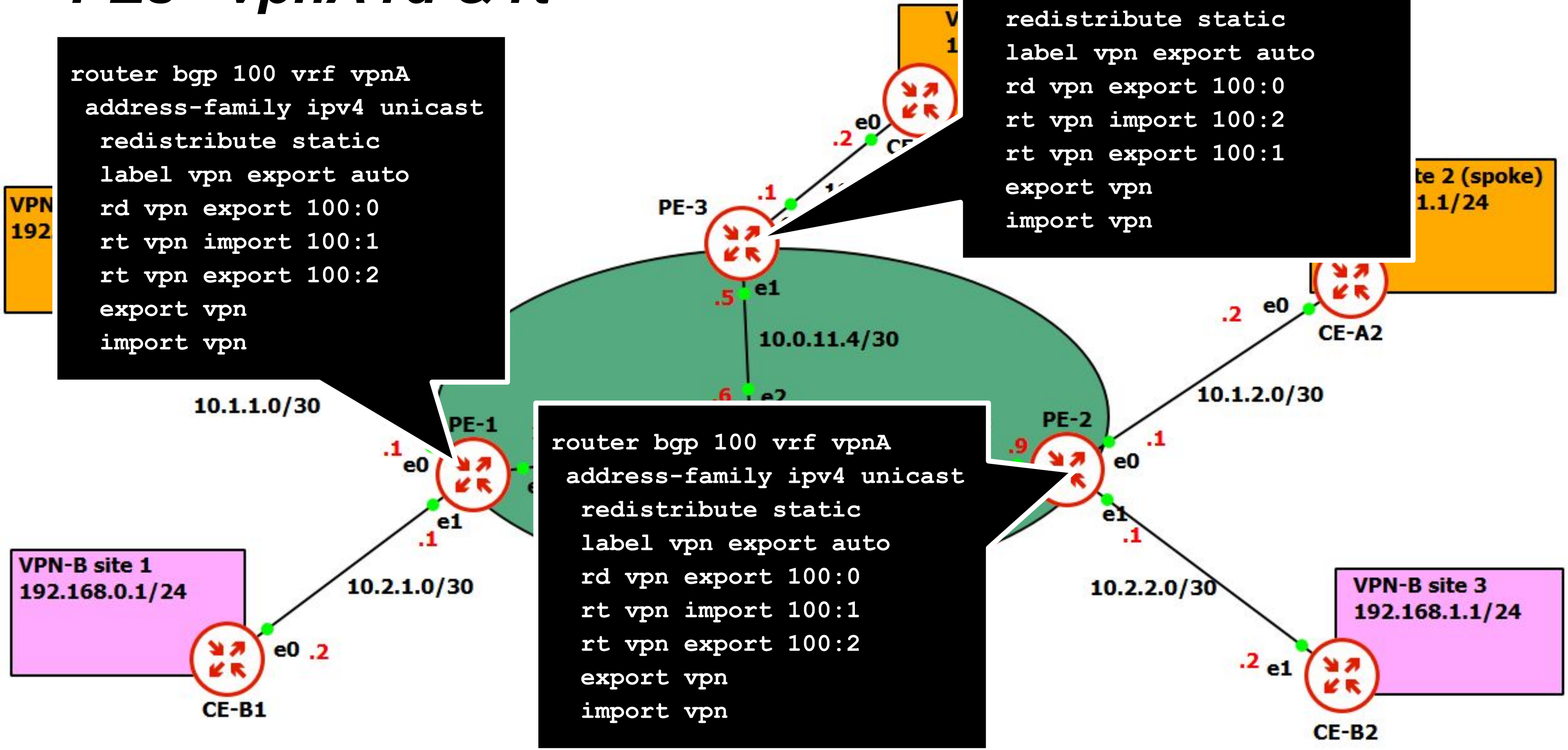


PEs - vpnA rd & rt

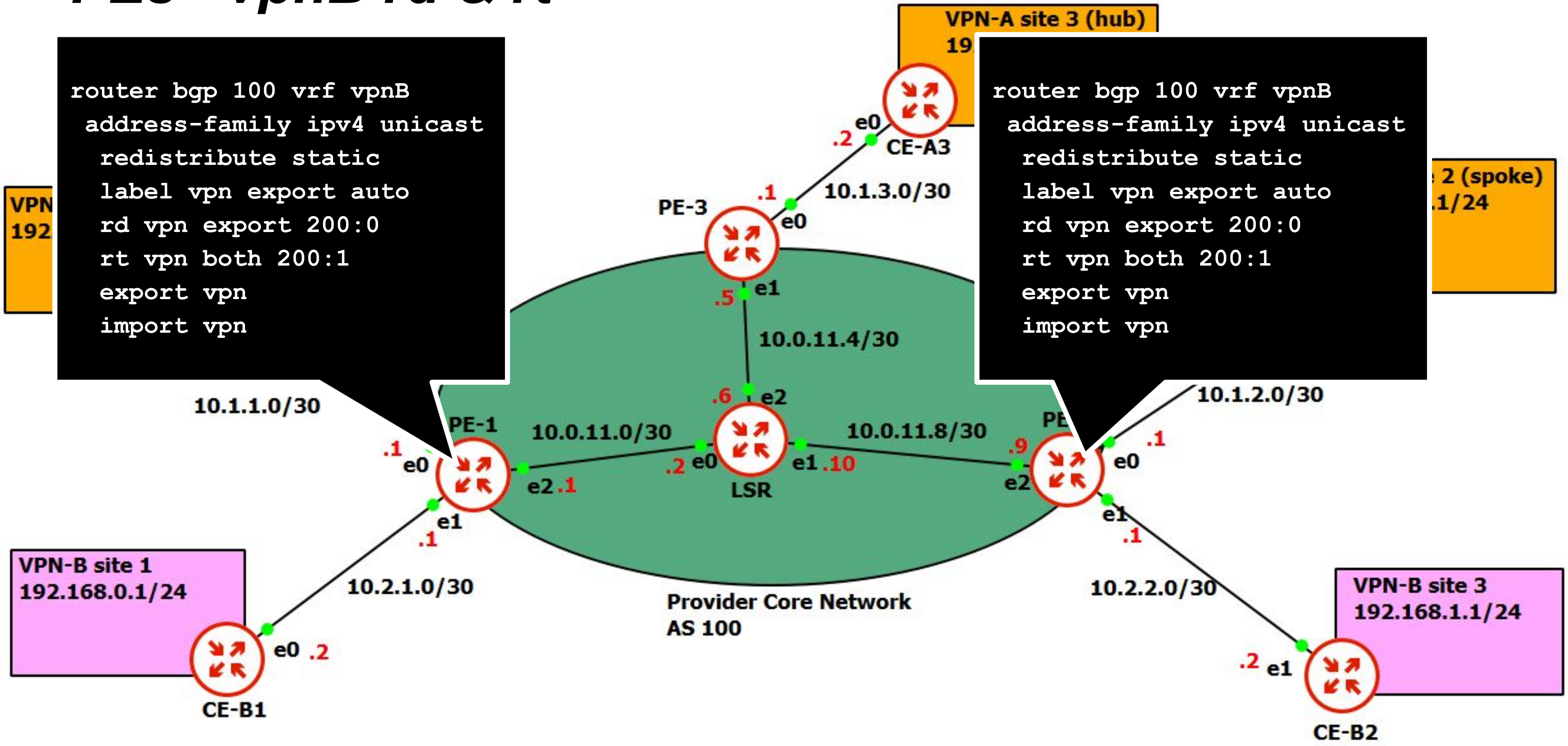
```
router bgp 100 vrf vpnA
 address-family ipv4 unicast
  redistribute static
  label vpn export auto
  rd vpn export 100:0
  rt vpn import 100:2
  rt vpn export 100:1
  export vpn
  import vpn
```

```
router bgp 100 vrf vpnA
 address-family ipv4 unicast
  redistribute static
  label vpn export auto
  rd vpn export 100:0
  rt vpn import 100:1
  rt vpn export 100:2
  export vpn
  import vpn
```

```
router bgp 100 vrf vpnA
 address-family ipv4 unicast
  redistribute static
  label vpn export auto
  rd vpn export 100:0
  rt vpn import 100:1
  rt vpn export 100:2
  export vpn
  import vpn
```



PEs - vpnB rd & rt




```
router bgp 100 vrf vpnB
 address-family ipv4 unicast
  redistribute static
  label vpn export auto
  rd vpn export 200:0
  rt vpn both 200:1
  export vpn
  import vpn
```

```
router bgp 100 vrf vpnB
 address-family ipv4 unicast
  redistribute static
  label vpn export auto
  rd vpn export 200:0
  rt vpn both 200:1
  export vpn
  import vpn
```