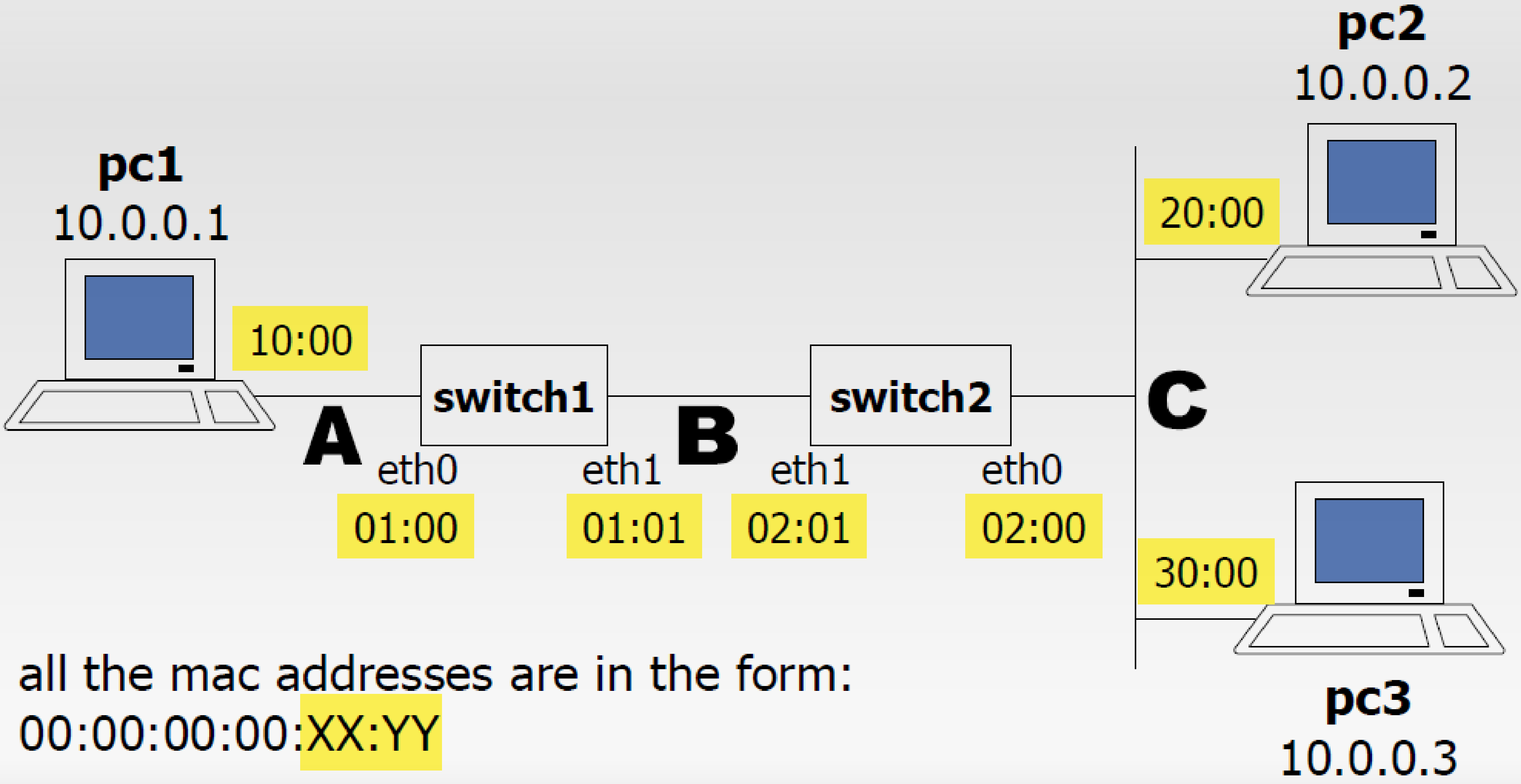
Switch & Bridge

- The **brctl** tool can be used to to set up, maintain, and inspect the ethernet bridge configuration in the linux kernel
- An ethernet bridge is a device commonly used to connect different networks of ethernets together
- Each of the ethernets being connected corresponds to one physical interface in the bridge

Step1: network topology



all the mac addresses are in the form:
00:00:00:00:XX:YY

ABC are collision domains

Step 2: Starting the lab

- To start the lab just do the following
  - cd netkit-lab_two-switches
  - type lstart
- The started lab is made up of
  - 3 virtual machines that implement the pcs
  - 2 virtual machines that implement the switches
  - automatically configured to perform switching
  - all the virtual machines and their network interfaces are automatically configured (see startup files)

Step 3: Configuring network interfaces

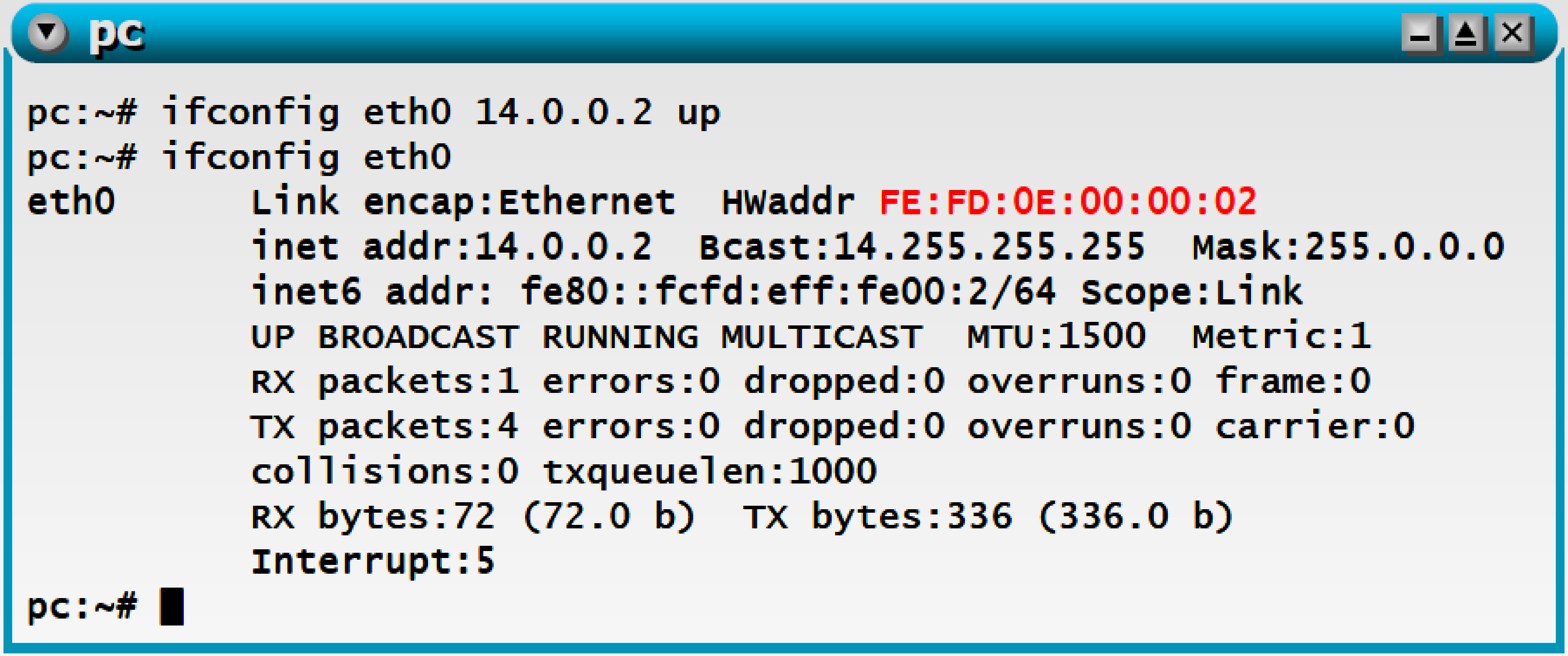
- Real network interfaces have a wired in mac address
  - the first three bytes make up the **Organizationally Unique Identifier (OUI)**, a sequence that matches the vendor of the nic
  - the remaining three bytes are the interface serial number
- MAC address of an interface card manufactured by Asustek inc.:

00:13:D4:**AC:55:4E**

oui **serial**

Step 3: Configuring network interfaces

- Virtual network interfaces are automatically assigned a mac address

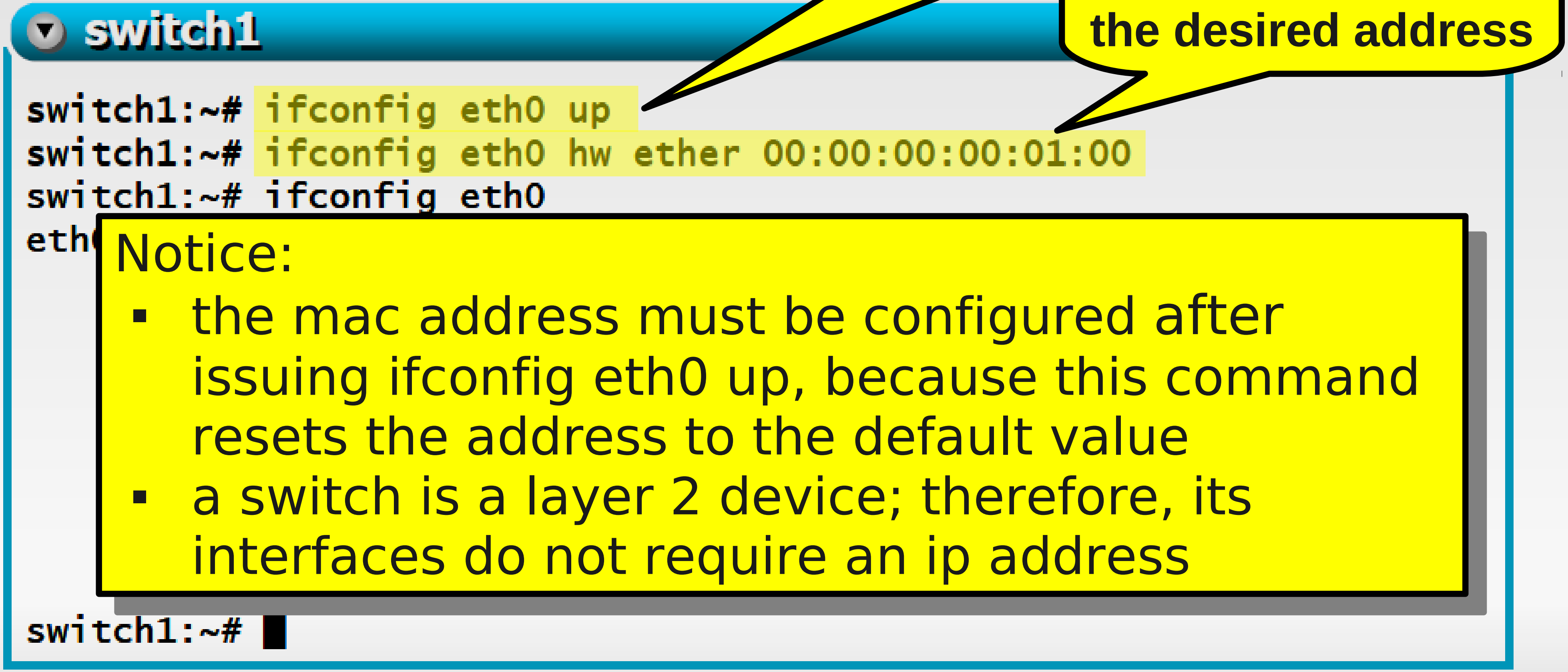


```
pc:~# ifconfig eth0 14.0.0.2 up
pc:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr FE:FD:0E:00:00:02
          inet addr:14.0.0.2  Bcast:14.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::fcfd:eff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:72 (72.0 b)  TX bytes:336 (336.0 b)
          Interrupt:5
pc:~# 
```

- Depending on the version of netkit in use, the mac address might be derived from the ip address

Step 3: Configuring network interfaces

- The mac address of a virtu e can be forcedly configured using t of the ifconfig command



```
switch1:~# ifconfig eth0 up
switch1:~# ifconfig eth0 hw ether 00:00:00:00:01:00
switch1:~# ifconfig eth0
eth

switch1:~# █
```

After this command the interface a default add

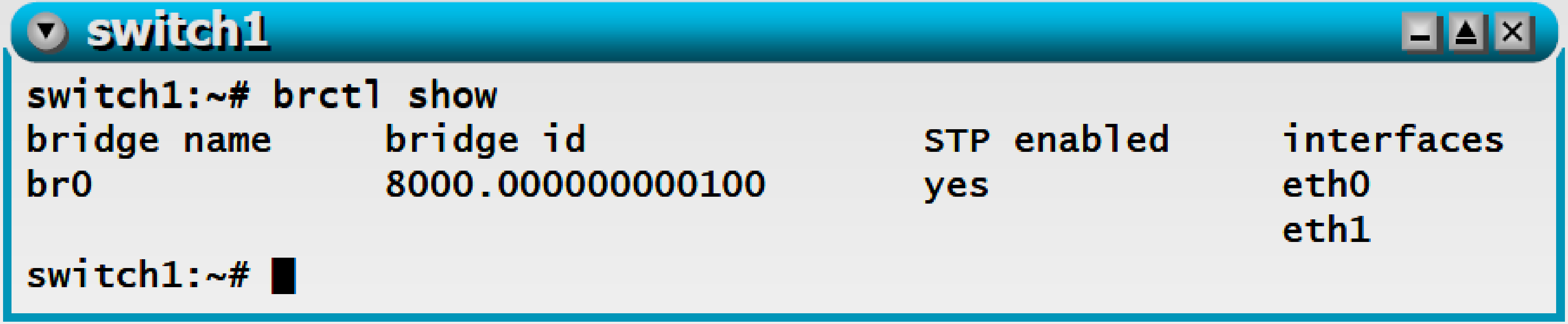
After this command the interface has the desired address

Notice:

- the mac address must be configured after issuing ifconfig eth0 up, because this command resets the address to the default value
- a switch is a layer 2 device; therefore, its interfaces do not require an ip address

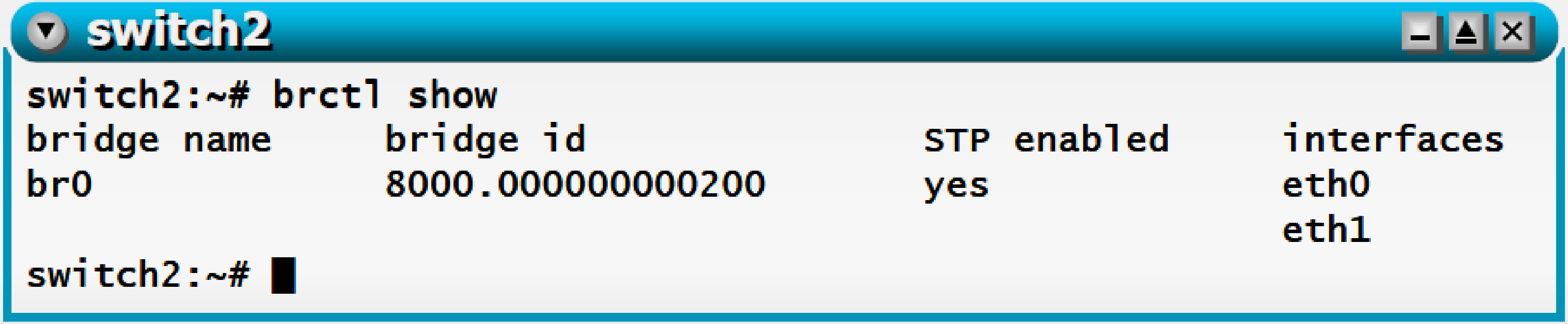
Step 4: Bridging capabilities

- **brctl** allows to check and configure the settings of the bridging capabilities of a virtual machine



```
switch1:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.000000000100       yes             eth0
                                                        eth1

switch1:~# █
```



```
switch2:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.000000000200       yes             eth0
                                                        eth1

switch2:~# █
```

Step 4: Bridging capabilities

- Create a new bridge br0

```
brctl addbr br0
```

- Attach network interfaces to bridge br0

```
brctl addif br0 eth0
brctl addif br0 eth1
```

- Enable the spanning tree protocol on bridge br0

```
brctl stp br0 on
```

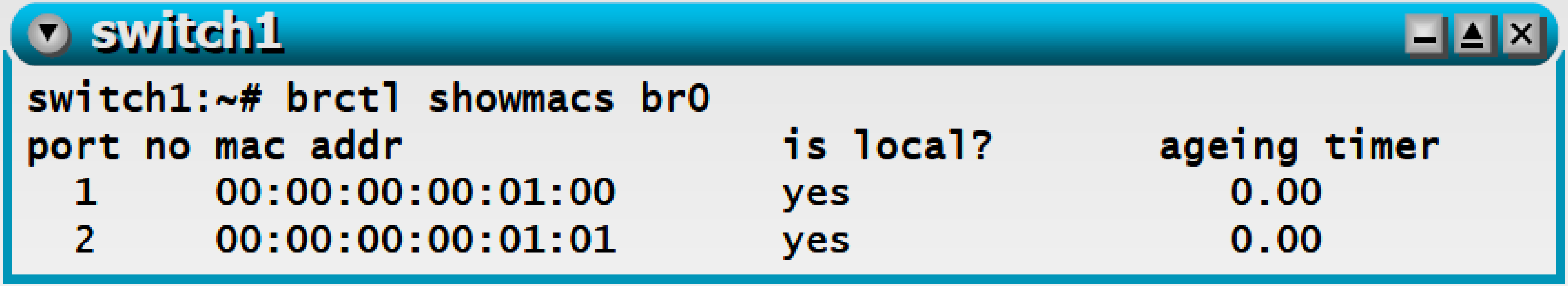
- Enable the bridge

```
ifconfig br0 up
```

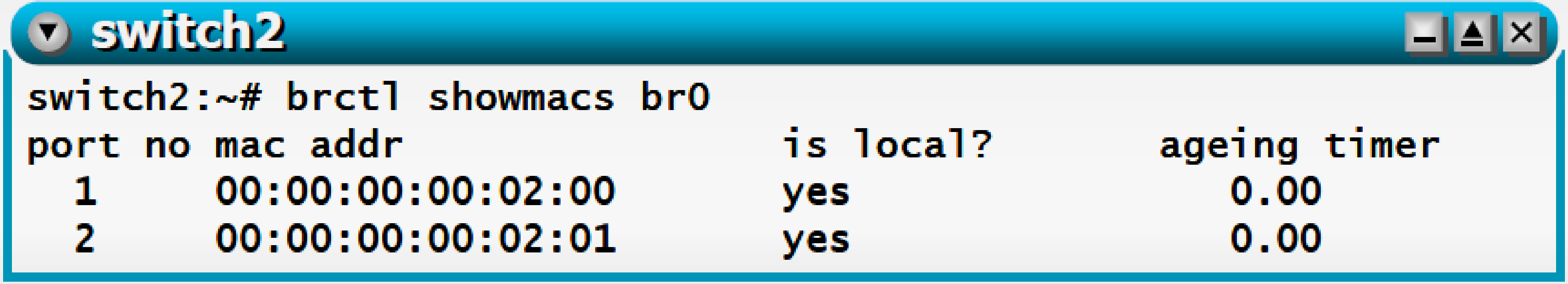
- A virtual machine may enable several bridging processes (on different network interfaces)
- Once configured, a bridge is visible as a network interface that must be brought up in order to work properly

Step 5: Investigating source address tables

- If PCs do not generate any traffic, the source address tables only contain information about local ports



```
switch1:~# brctl showmacs br0
port no mac addr                is local?       ageing timer
  1     00:00:00:00:01:00       yes               0.00
  2     00:00:00:00:01:01       yes               0.00
```

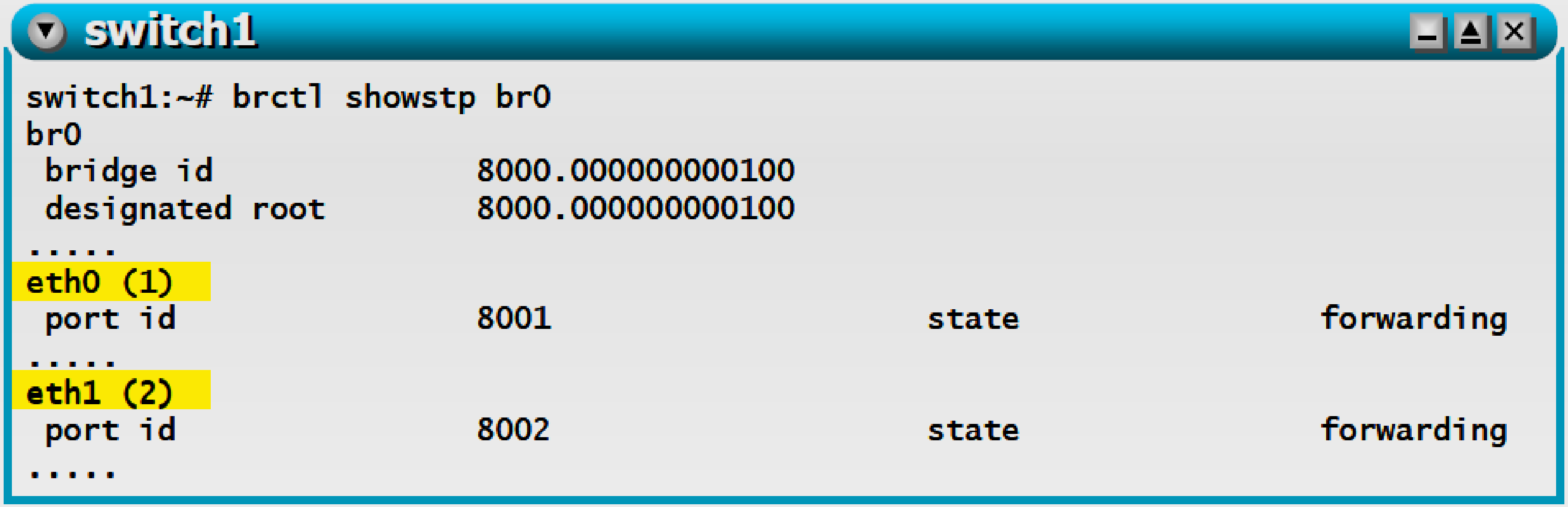


```
switch2:~# brctl showmacs br0
port no mac addr                is local?       ageing timer
  1     00:00:00:00:02:00       yes               0.00
  2     00:00:00:00:02:01       yes               0.00
```

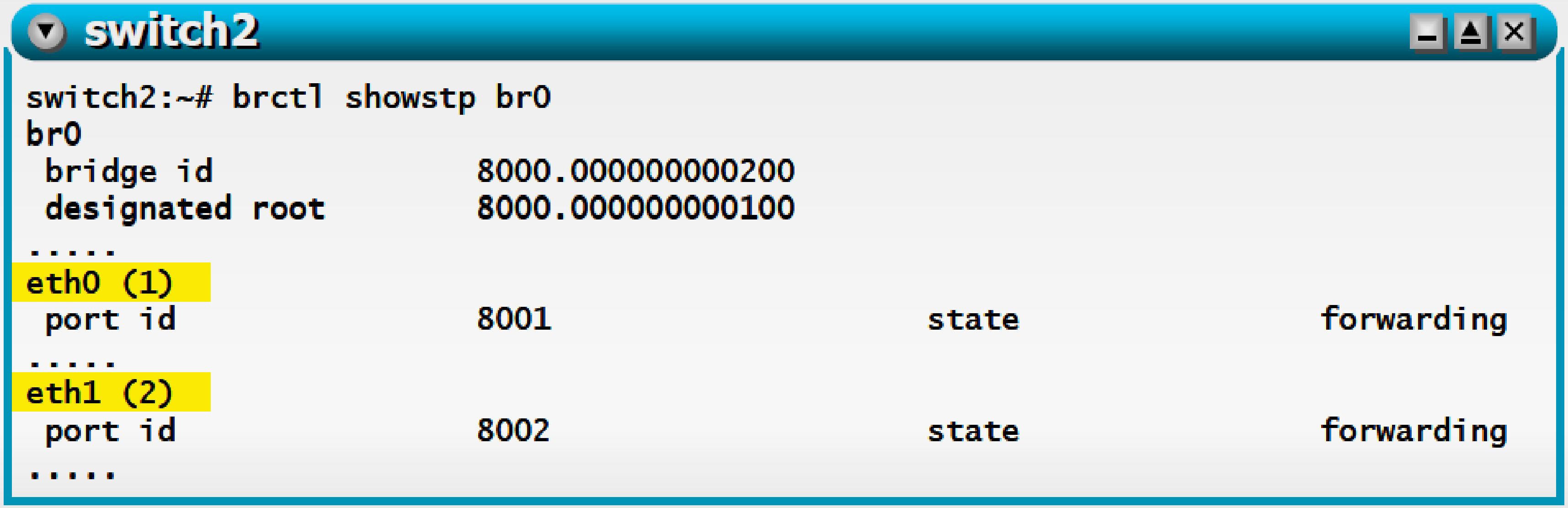
Step 5: Investigating source address tables

- Depending on the configuration, a machine may generate traffic even if not solicited (e.g., broadcast packets)
  - the source address tables of switch1 and switch2 may already contain non-local entries
  - hard to prevent
- Ports (=interfaces) are numbered according to the 802.1d standard
  - the parallelism between kernel interface numbering (ethX) and 802.1d numbering can be obtained by using brctl showstp

Step 5: Investigating source address tables

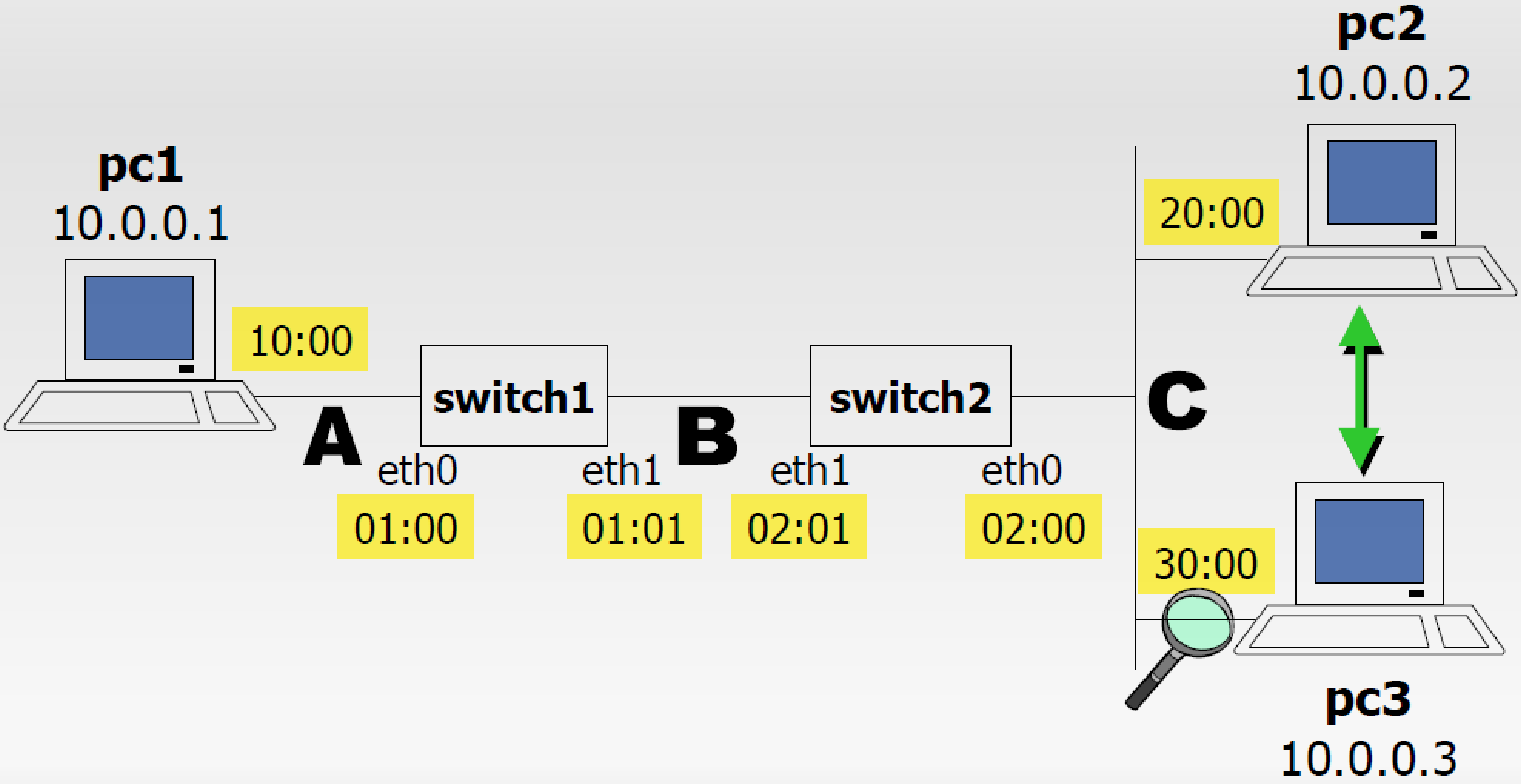


```
switch1:~# brctl showstp br0
br0
 bridge id              8000.000000000100
 designated root        8000.000000000100
.....
eth0 (1)
 port id                8001                    state                forwarding
.....
eth1 (2)
 port id                8002                    state                forwarding
.....
```



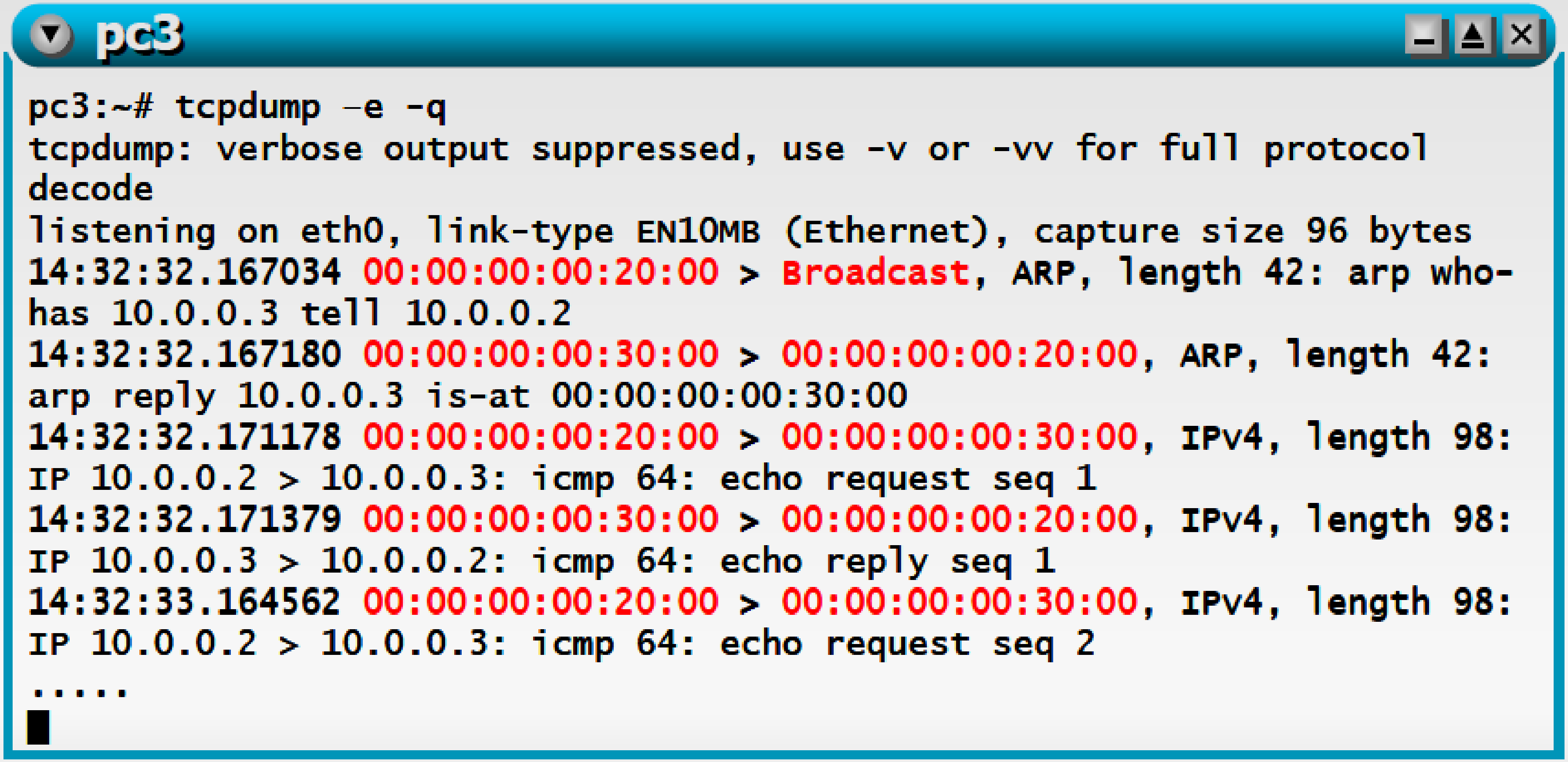
```
switch2:~# brctl showstp br0
br0
 bridge id              8000.000000000200
 designated root        8000.000000000100
.....
eth0 (1)
 port id                8001                    state                forwarding
.....
eth1 (2)
 port id                8002                    state                forwarding
.....
```

Step 6: Evolution of the address tables



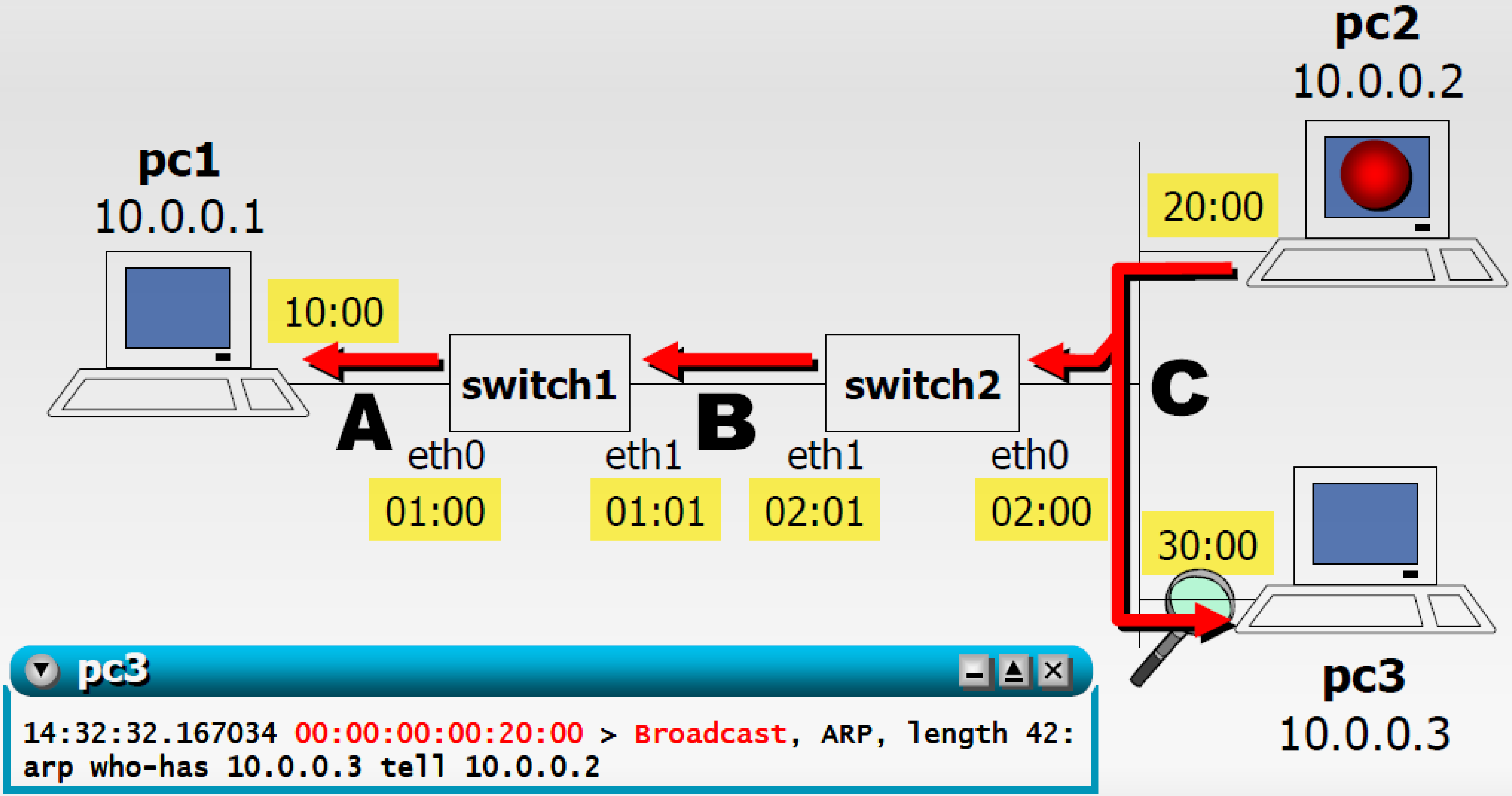
Step 6: Evolution of the address tables

- pc3 sees the traffic exchanged on its collision domain (**C**)



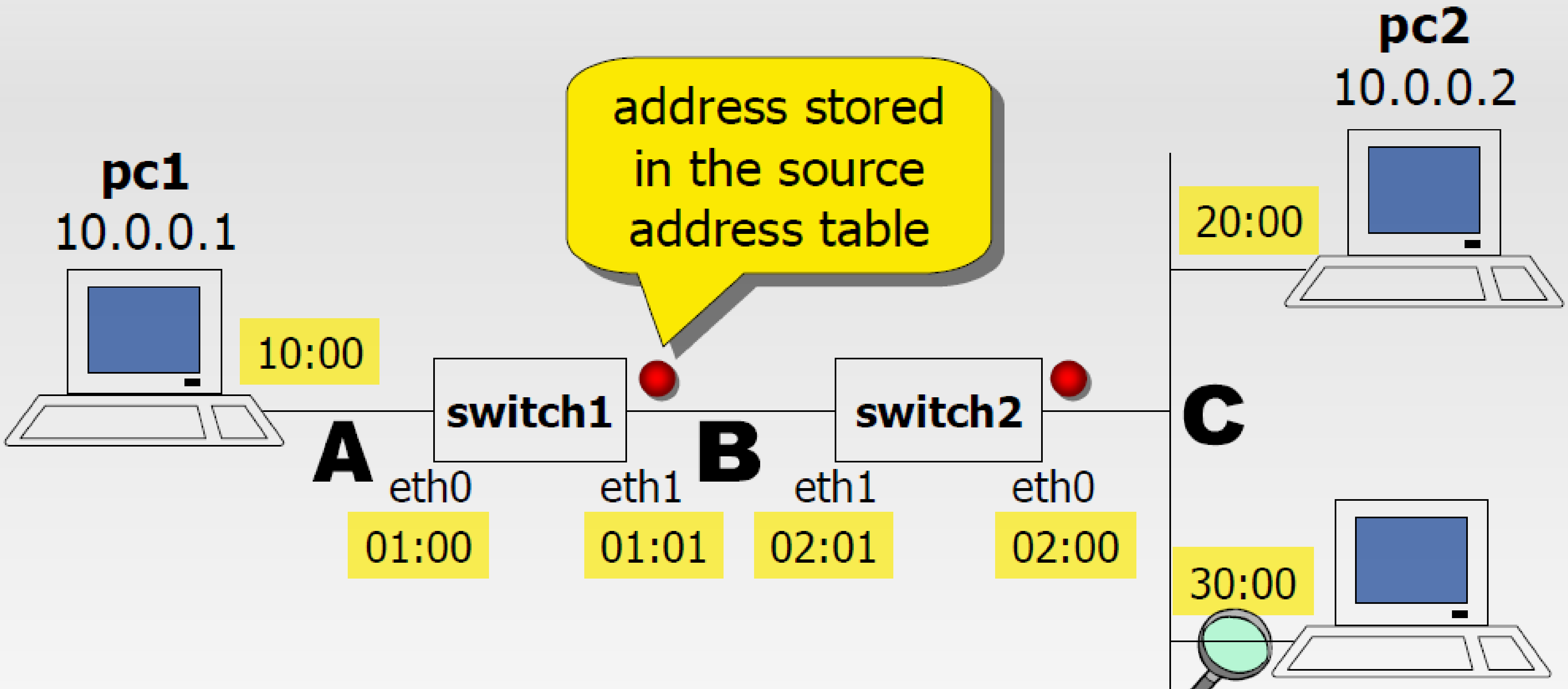
```
pc3:~# tcpdump -e -q
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
14:32:32.167034 00:00:00:00:20:00 > Broadcast, ARP, length 42: arp who-
has 10.0.0.3 tell 10.0.0.2
14:32:32.167180 00:00:00:00:30:00 > 00:00:00:00:20:00, ARP, length 42:
arp reply 10.0.0.3 is-at 00:00:00:00:30:00
14:32:32.171178 00:00:00:00:20:00 > 00:00:00:00:30:00, IPv4, length 98:
IP 10.0.0.2 > 10.0.0.3: icmp 64: echo request seq 1
14:32:32.171379 00:00:00:00:30:00 > 00:00:00:00:20:00, IPv4, length 98:
IP 10.0.0.3 > 10.0.0.2: icmp 64: echo reply seq 1
14:32:33.164562 00:00:00:00:20:00 > 00:00:00:00:30:00, IPv4, length 98:
IP 10.0.0.2 > 10.0.0.3: icmp 64: echo request seq 2
.....
```

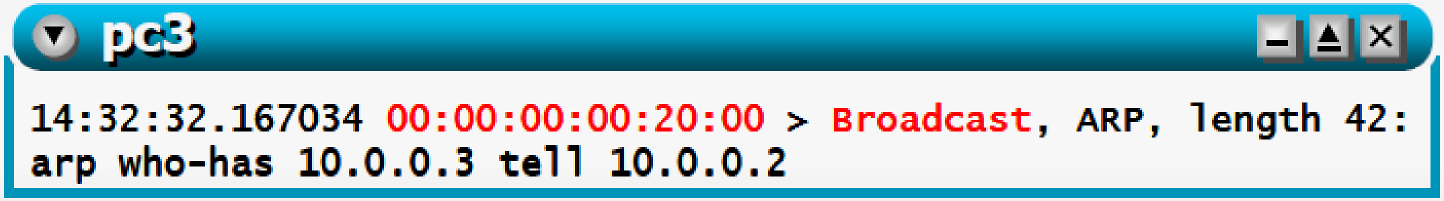
Step 6: Evolution of the address tables



```
14:32:32.167034 00:00:00:00:20:00 > Broadcast, ARP, length 42:
arp who-has 10.0.0.3 tell 10.0.0.2
```

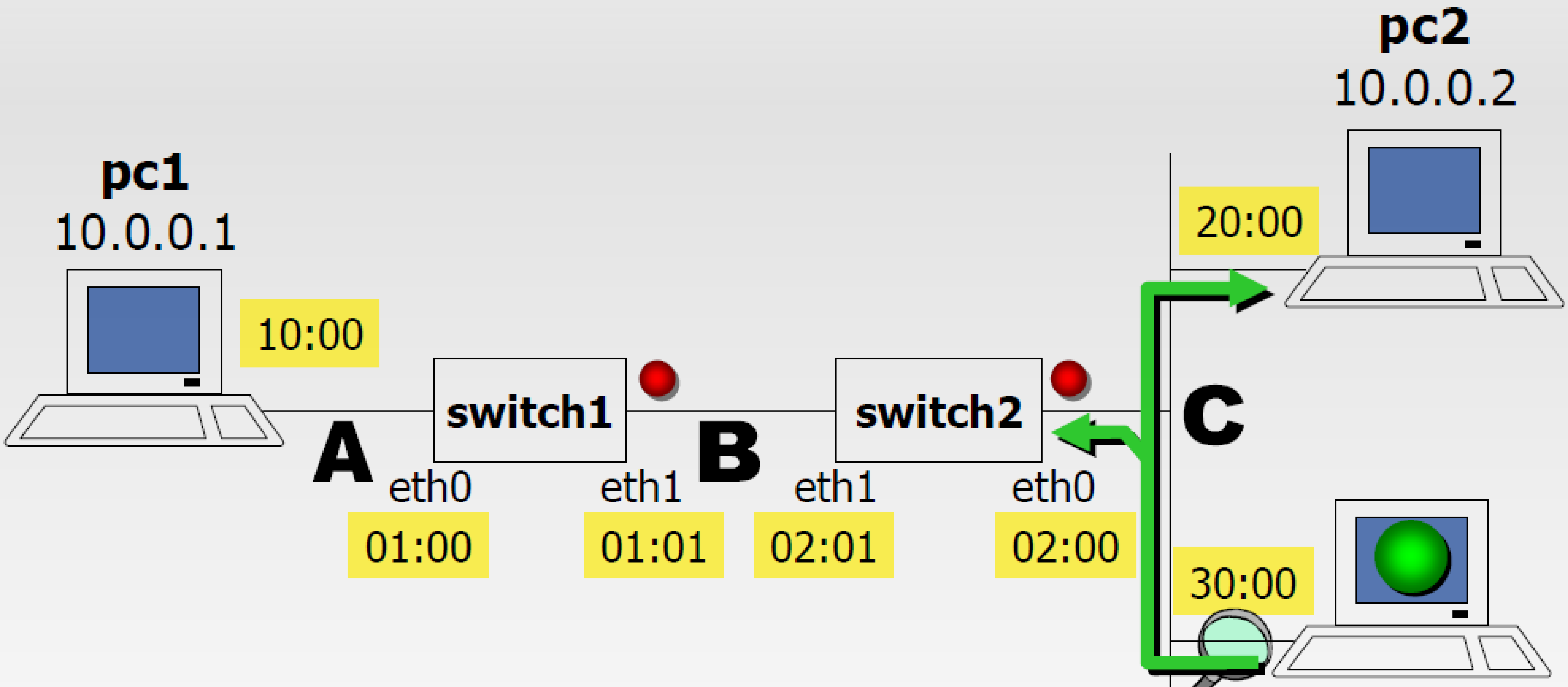
Step 6: Evolution of the address tables

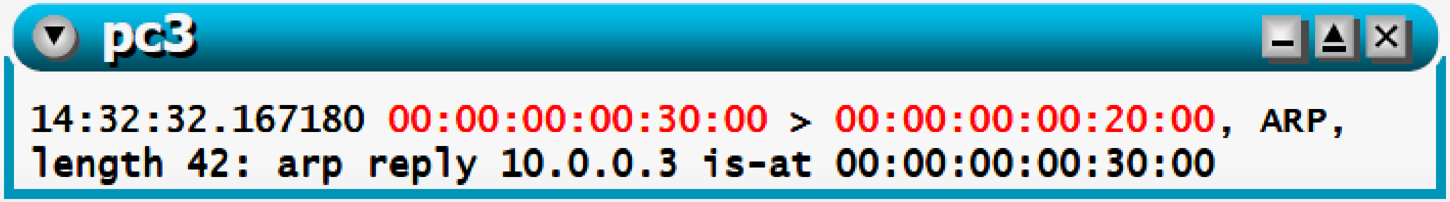




```
14:32:32.167034 00:00:00:00:20:00 > Broadcast, ARP, length 42:
arp who-has 10.0.0.3 tell 10.0.0.2
```

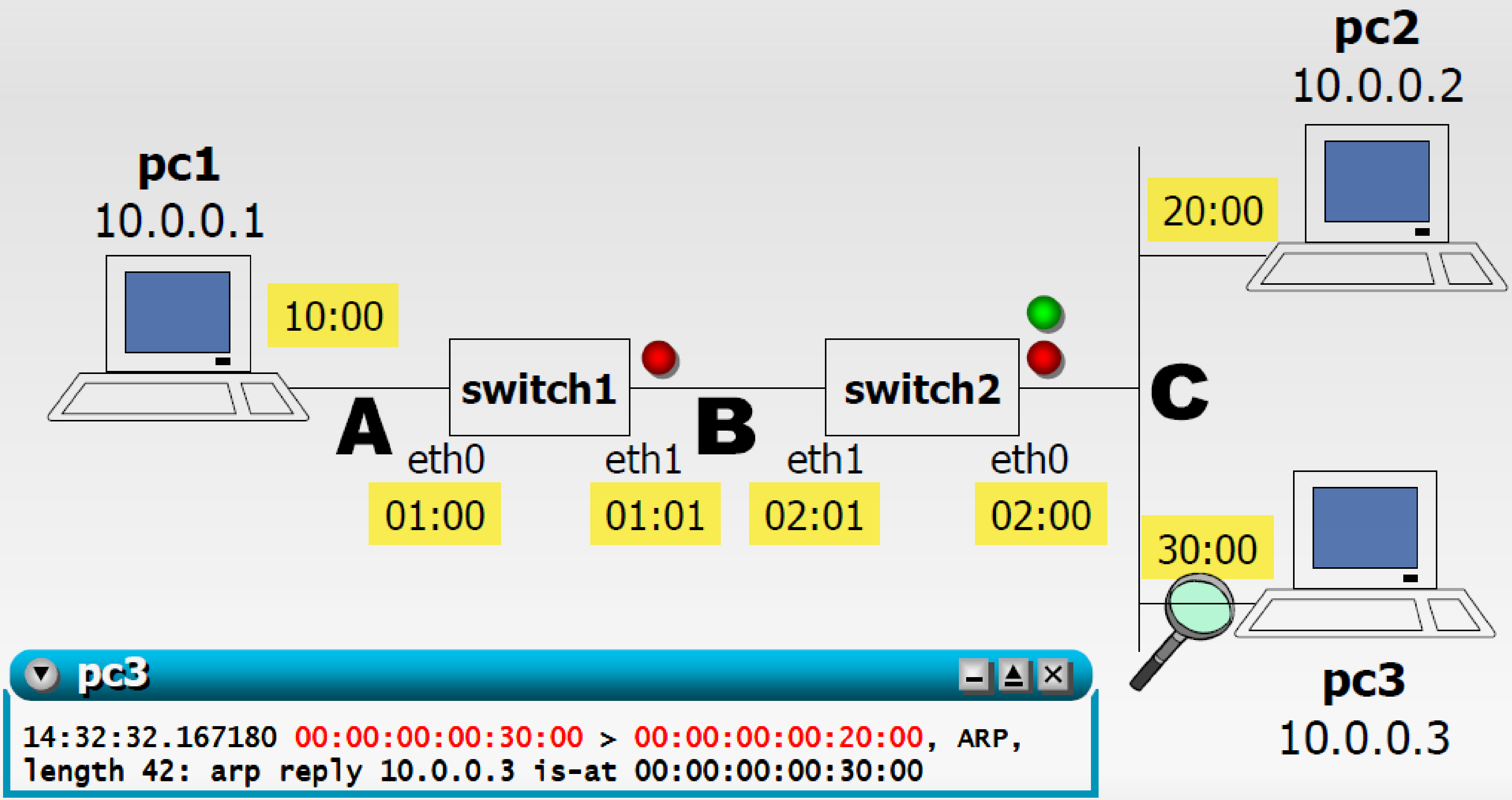
Step 6: Evolution of the address tables





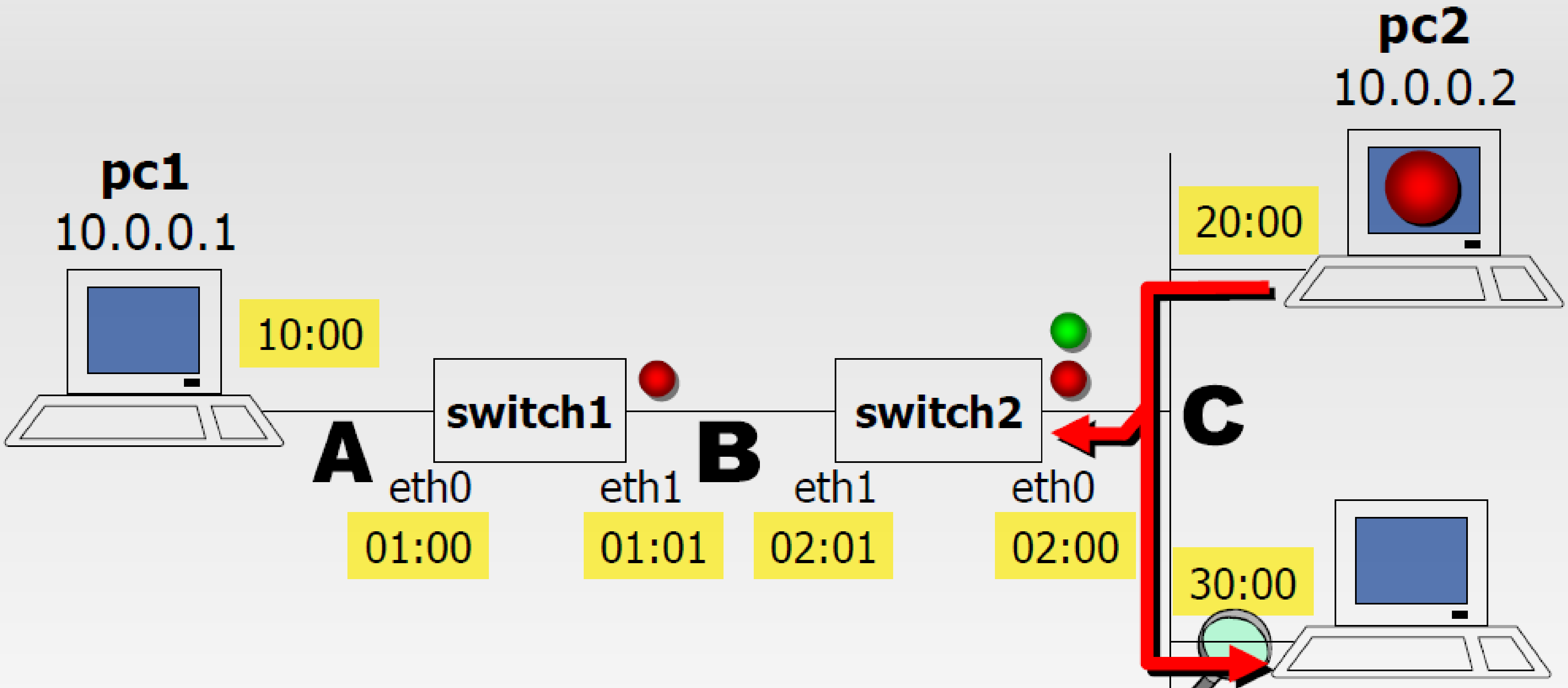
```
14:32:32.167180 00:00:00:00:30:00 > 00:00:00:00:20:00, ARP,
length 42: arp reply 10.0.0.3 is-at 00:00:00:00:30:00
```

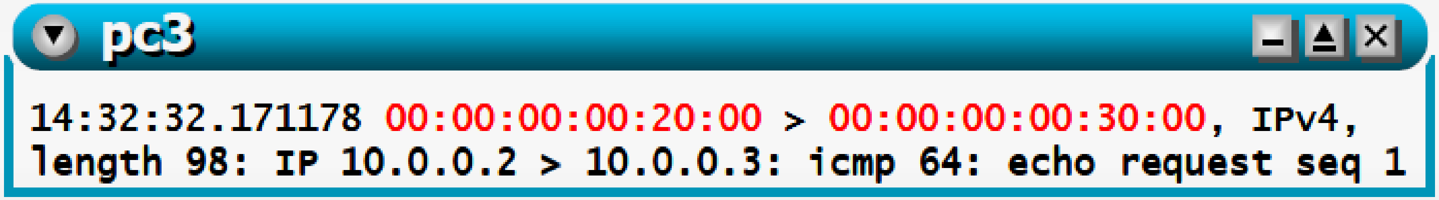
Step 6: Evolution of the address tables



```
14:32:32.167180 00:00:00:00:30:00 > 00:00:00:00:20:00, ARP,
length 42: arp reply 10.0.0.3 is-at 00:00:00:00:30:00
```

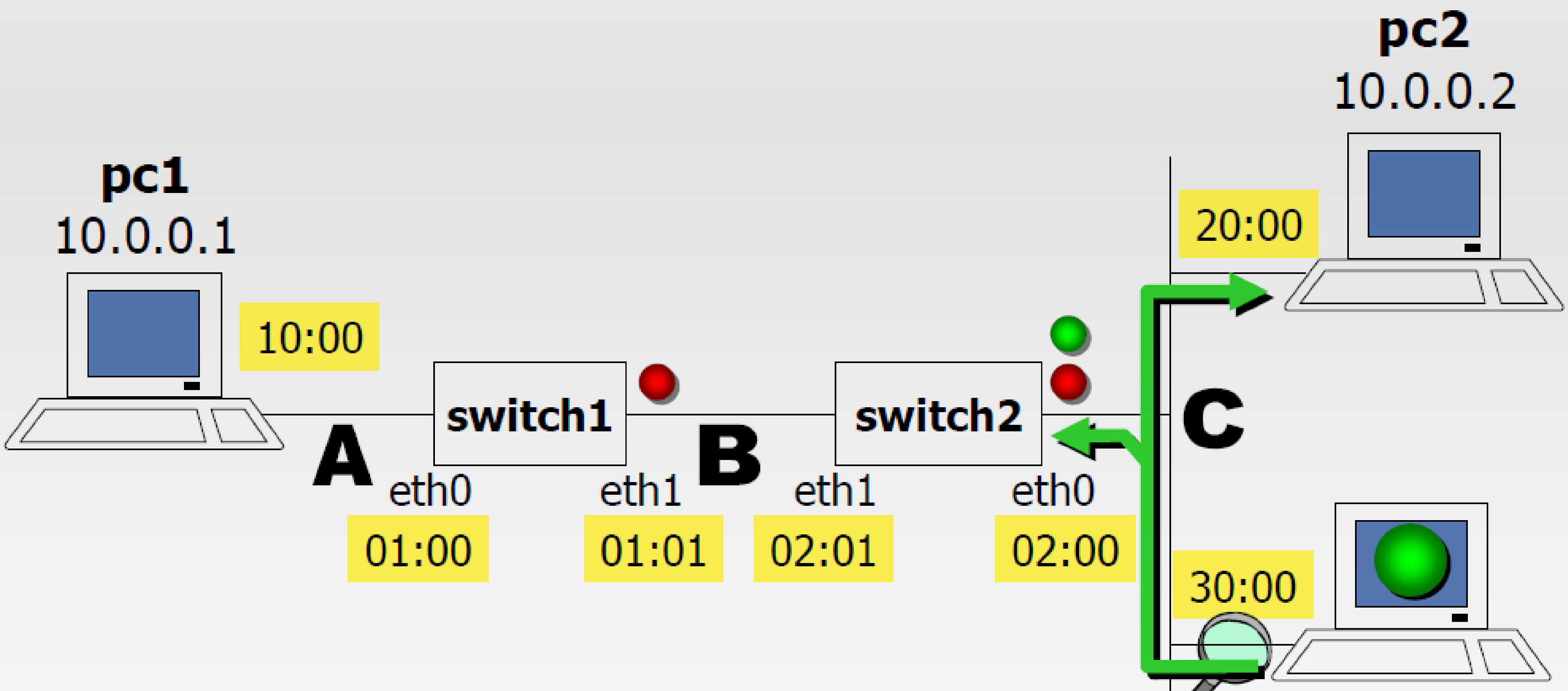
Step 6: Evolution of the address tables

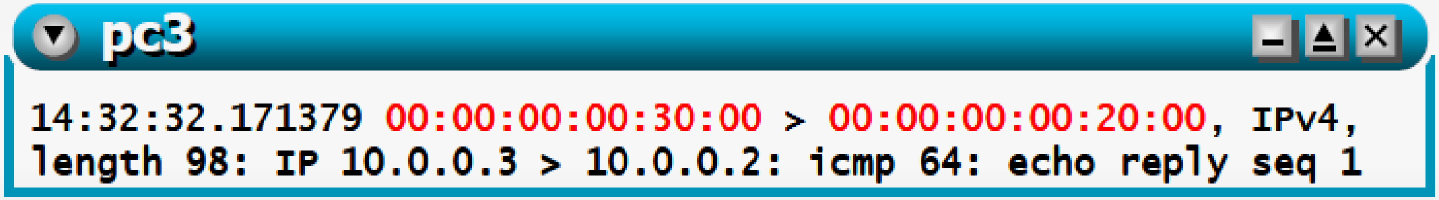




```
14:32:32.171178 00:00:00:00:20:00 > 00:00:00:00:30:00, IPv4,
length 98: IP 10.0.0.2 > 10.0.0.3: icmp 64: echo request seq 1
```

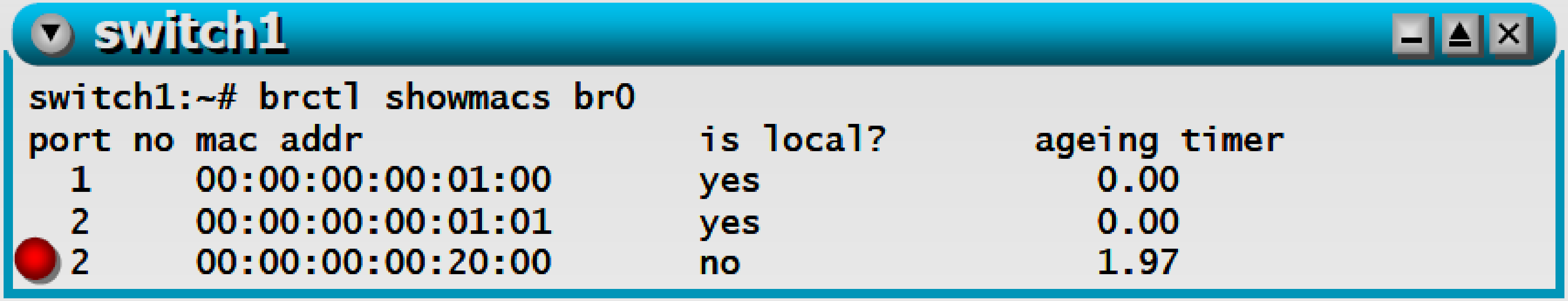
Step 6: Evolution of the address tables





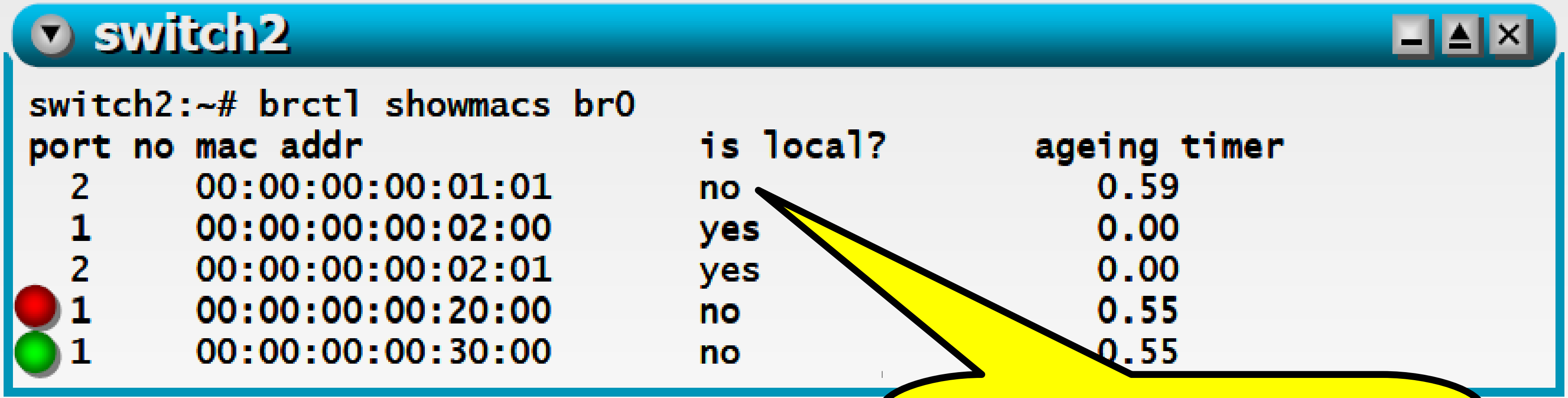
```
14:32:32.171379 00:00:00:00:30:00 > 00:00:00:00:20:00, IPv4,
length 98: IP 10.0.0.3 > 10.0.0.2: icmp 64: echo reply seq 1
```

Step 6: Evolution of the address tables



```
switch1:~# brctl showmacs br0
port no mac addr                is local?       ageing timer
  1     00:00:00:00:01:00       yes                0.00
  2     00:00:00:00:01:01       yes                0.00
  2     00:00:00:00:20:00       no                 1.97
```

switch1/eth0
switch1/eth1
pc2 ●



```
switch2:~# brctl showmacs br0
port no mac addr                is local?       ageing timer
  2     00:00:00:00:01:01       no                 0.59
  1     00:00:00:00:02:00       yes                0.00
  2     00:00:00:00:02:01       yes                0.00
  1     00:00:00:00:20:00       no                 0.55
  1     00:00:00:00:30:00       no                 0.55
```

switch1/eth1
switch2/eth0
switch2/eth1
pc2 ●
pc3 ●

This entry is due to packets exchanged for spanning tree computation

Step 6: Evolution of the address tables

- switch2 knows the positions of pc2 and pc3 since it has seen their traffic
- switch1 does not know the position of pc3 since pc3’s traffic has been filtered out by switch2
- the two switches are not aware of pc1