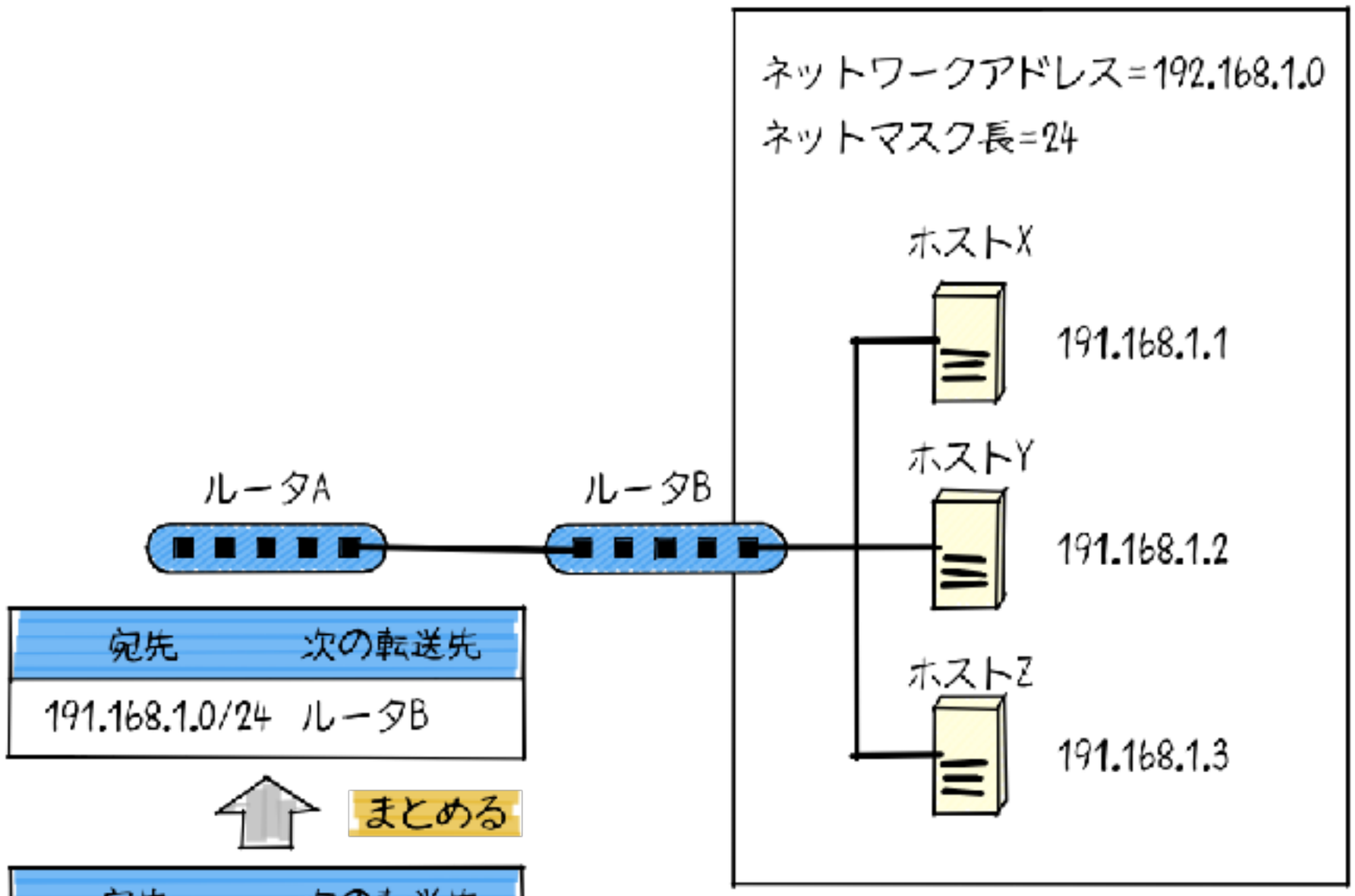
ルータを作ろう 後編

ルーティングテーブルの仕組みとOpenFlow実装

宛先ホストをまとめる

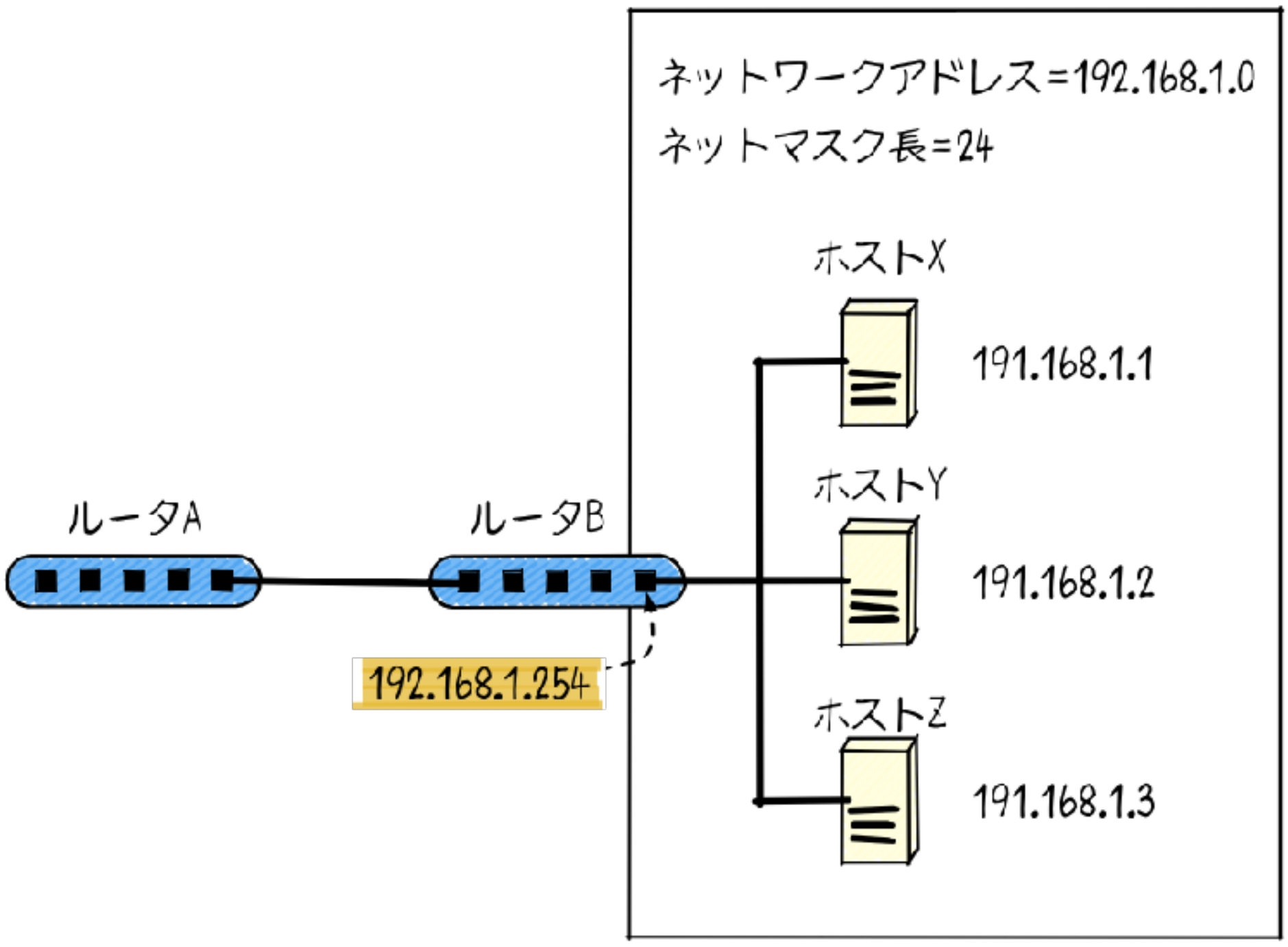


宛先	次の転送先
191.168.1.0/24	ルータB

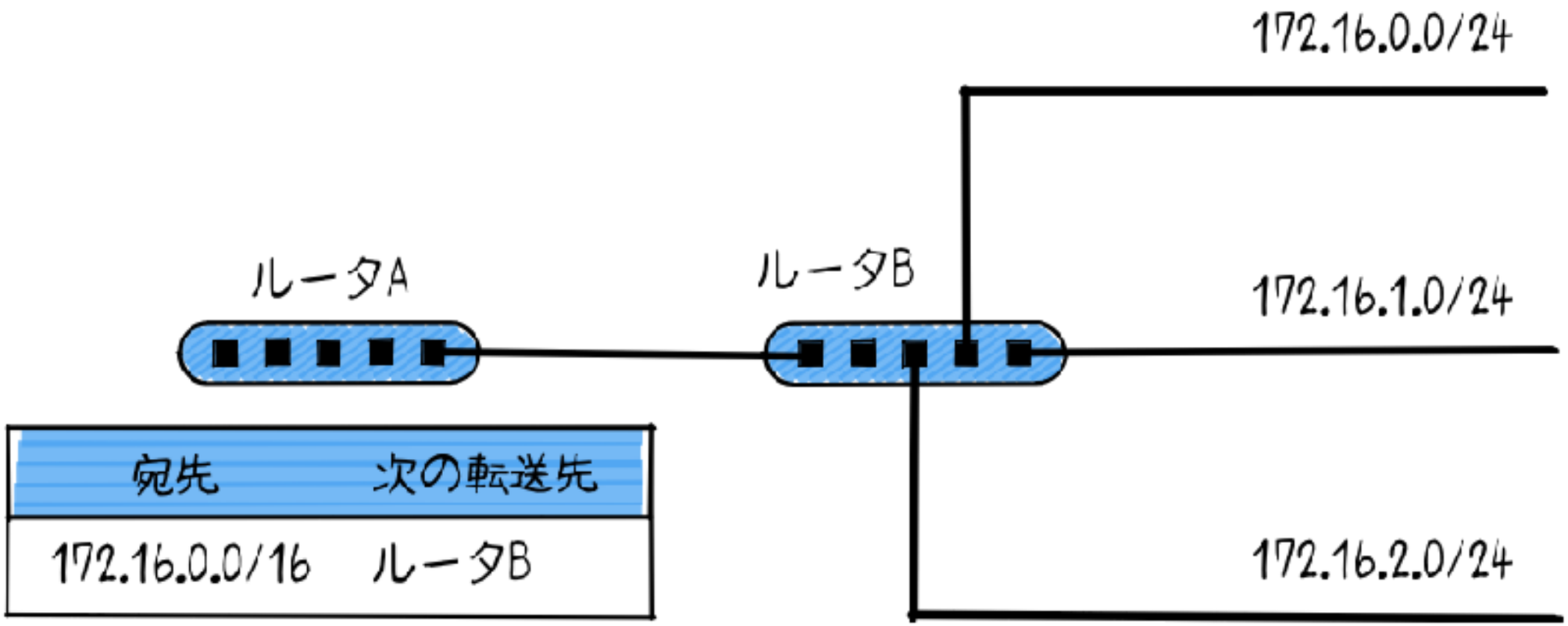
まとめる

宛先	次の転送先
191.168.1.1	ルータB
191.168.1.2	ルータB
191.168.1.3	ルータB

宛先ホストと直接つながっている?



ネットワーク宛エントリをまとめる

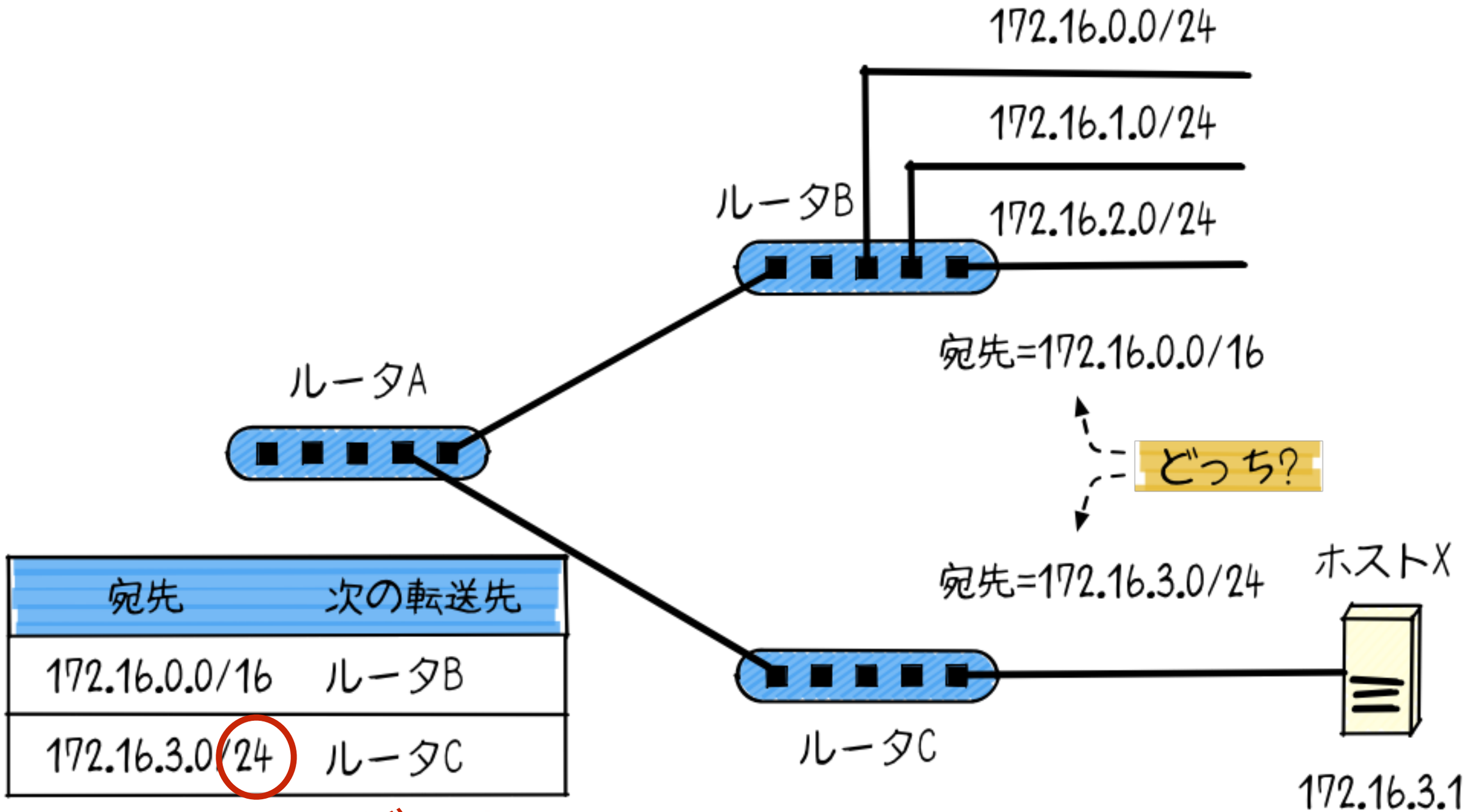


宛先	次の転送先
172.16.0.0/16	ルータB

まとめる

宛先	次の転送先
172.16.0.0/24	ルータB
172.16.1.0/24	ルータB
172.16.2.0/24	ルータB

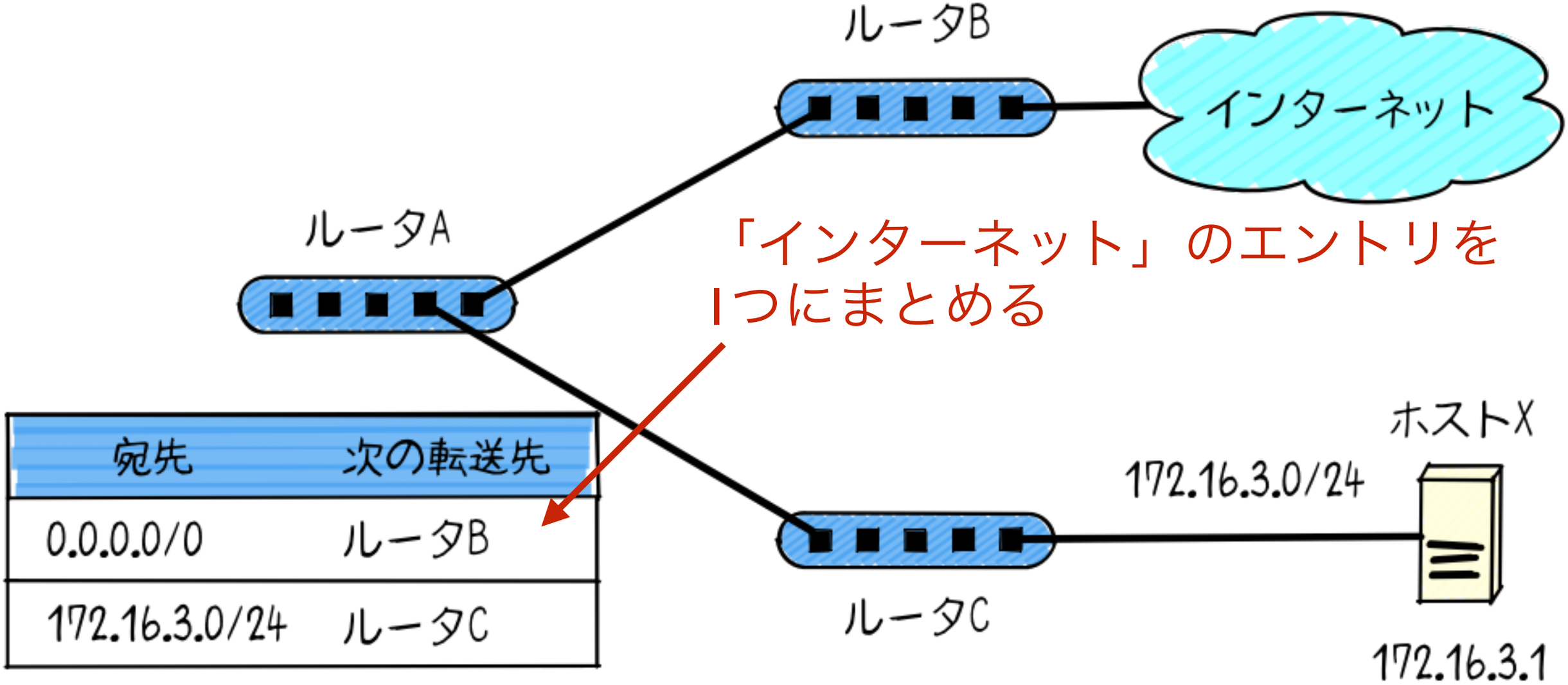
複数のエントリにマッチする場合



宛先	次の転送先
172.16.0.0/16	ルータB
172.16.3.0/24	ルータC

ロンゲストマッチ

デフォルトルート


ルータB
インターネット
ルータA
「インターネット」のエントリを
1つにまとめる
宛先
次の転送先
0.0.0.0/0
ルータB
172.16.3.0/24
ルータC
ルータC
ホストX
172.16.3.0/24
172.16.3.1

PacketInハンドラ
→IPパケットの処理
→書換えと転送

```
def forward(dpid, packet_in)
  next_hop = resolve_next_hop(packet_in.destination_ip_address)

  interface = Interface.find_by_prefix(next_hop)
  return if !interface || (interface.port_number == packet_in.in_port)

  arp_entry = @arp_table.lookup(next_hop)
  if arp_entry
    actions = [SetSourceMacAddress.new(interface.mac_address),
               SetDestinationMacAddress.new(arp_entry.mac_address),
               SendOutPort.new(interface.port_number)]
    send_flow_mod_add(dpid,
                      match: ExactMatch.new(packet_in), actions: actions)
    send_packet_out(dpid, raw_data: packet_in.raw_data, actions: actions)
  else
    send_later(dpid,
               interface: interface,
               destination_ip: next_hop,
               data: packet_in.data)
  end
end
```

- 次の転送先を決める (#resolve_next_hop)
- 出力インタフェースを決める

```
def resolve_next_hop(destination_ip_address)
  interface = Interface.find_by_prefix(destination_ip_address)
  if interface
    destination_ip_address
  else
    @routing_table.lookup(destination_ip_address)
  end
end
```

- 宛先と同じネットワークアドレスを持つインタフェースがあるか?
- なかった場合、ルーティングテーブルを調べる

```
interface = Interface.find_by_prefix(next_hop)
return if !interface || (interface.port_number == packet_in.in_port)
```

- 次の転送先と同じネットワークアドレスを持つインタフェースがあるか?
- なければ転送できないのでパケットを破棄

ルーティングテーブル

```ruby
class RoutingTable
  include Pio

  MAX_NETMASK_LENGTH = 32

  def initialize(route)
    @db = Array.new(MAX_NETMASK_LENGTH + 1) { Hash.new }
    route.each { |each| add(each) }
  end

  def add(options)
    netmask_length = options.fetch(:netmask_length)
    prefix = IPv4Address.new(options.fetch(:destination)).mask(netmask_length)
    @db[netmask_length][prefix.to_i] = IPv4Address.new(options.fetch(:next_hop))
  end

  def lookup(destination_ip_address)
    MAX_NETMASK_LENGTH.downto(0).each do |each|
      prefix = destination_ip_address.mask(each)
      entry = @db[each][prefix.to_i]
      return entry if entry
    end
    nil
  end
end
```

ネットマスク長 (0-32) ごとに
経路を管理

ネットマスク長の長い順
(ロンゲストマッチ) に
経路を探索

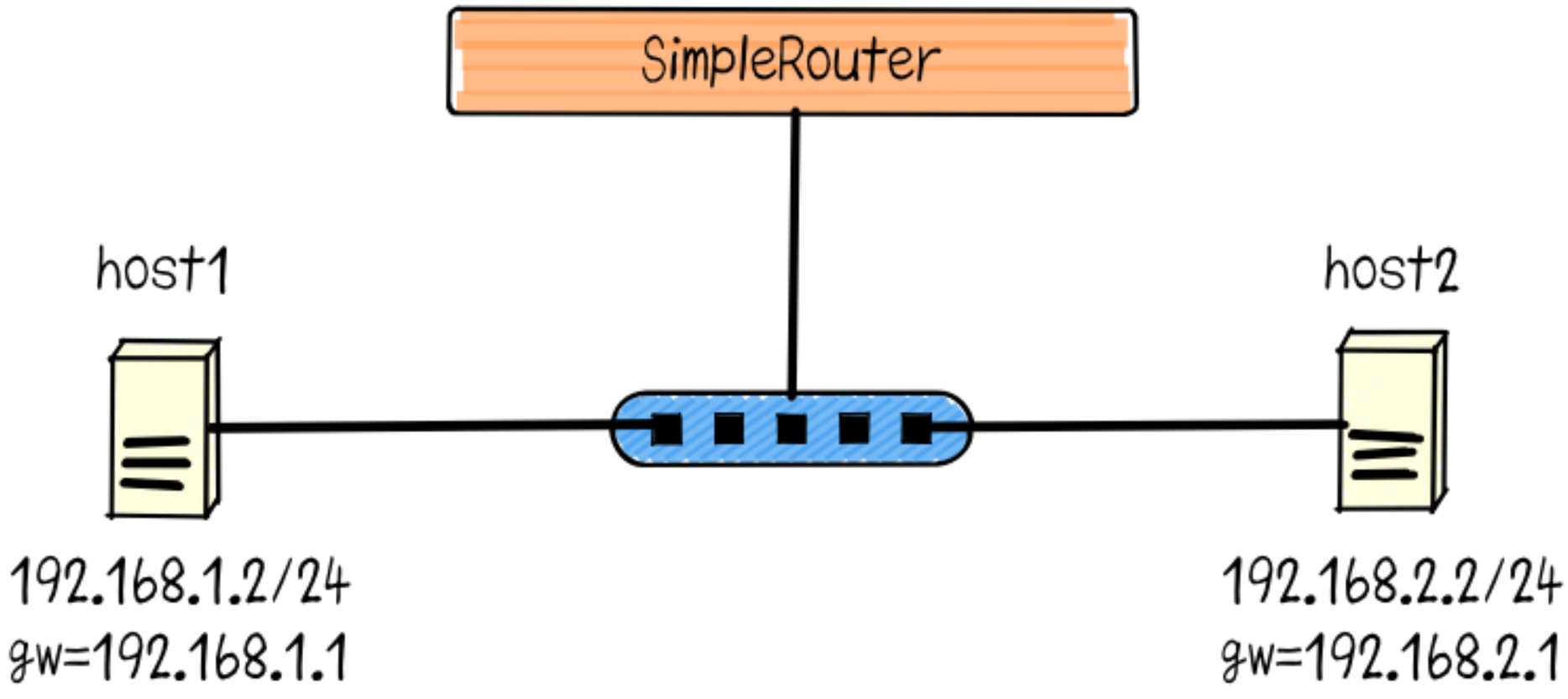
```
module Configuration
  INTERFACES = [
    {
      port: 1,
      mac_address: '01:01:01:01:01:01',
      ip_address: '192.168.1.1',
      netmask_length: 24
    },
    {
      port: 2,
      mac_address: '02:02:02:02:02:02',
      ip_address: '192.168.2.1',
      netmask_length: 24
    }
  ]

  ROUTES = [
    {
      destination: '0.0.0.0',
      netmask_length: 0,
      next_hop: '192.168.1.2'
    }
  ]
end
```

ルータのインタフェース

ルーティングテーブル

実行してみよう


SimpleRouter
host1
host2
192.168.1.2/24
gw=192.168.1.1
192.168.2.2/24
gw=192.168.2.1

設定ファイル

```
vswitch('0x1') { dpid 0x1 }
netns('host1') {
  ip '192.168.1.2'
  netmask '255.255.255.0'
  route net: '0.0.0.0', gateway: '192.168.1.1'
}
netns('host2') {
  ip '192.168.2.2'
  netmask '255.255.255.0'
  route net: '0.0.0.0', gateway: '192.168.2.1'
}
link '0x1', 'host1'
link '0x1', 'host2'
```

```
$ ./bin/trema run ./lib/simple-router.rb
    -c ./trema.conf
```

ルーティングテーブルの確認

```
$ ./bin/trema netns host1
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    0      0        0 host1
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 host1
```

- trema netns [ホスト名]: netns でコマンドを実行
- Ctrl-d で抜ける

ルータに ping

```
$ ./bin/trema netns host1
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=128 time=47.4 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=128 time=15.0 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=128 time=15.0 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=128 time=19.3 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=128 time=14.8 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=128 time=14.4 ms
64 bytes from 192.168.1.1: icmp_seq=7 ttl=128 time=15.1 ms
^C
--- 192.168.1.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time
6008ms
rtt min/avg/max/mdev = 14.425/20.189/47.473/11.245 ms
```

ホスト間の通信

```
$ ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=75.5 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=82.3 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=101 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=64 time=83.3 ms
64 bytes from 192.168.2.2: icmp_seq=5 ttl=64 time=78.2 ms
64 bytes from 192.168.2.2: icmp_seq=6 ttl=64 time=76.4 ms
64 bytes from 192.168.2.2: icmp_seq=7 ttl=64 time=70.9 ms
^C
--- 192.168.2.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time
6008ms
rtt min/avg/max/mdev = 70.995/81.159/101.180/9.050 ms
```

iperf で動作確認

host2

```
$ ./bin/trema netns host2
$ iperf -s --bind 192.168.2.2
------------------------------------------------------------
Server listening on TCP port 5001
Binding to local address 192.168.2.2
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

host1

```
$ ./bin/trema netns host1
$ iperf -c 192.168.2.2 --bind 192.168.1.2
------------------------------------------------------------
Client connecting to 192.168.2.2, TCP port 5001
Binding to local address 192.168.1.2
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.2 port 5001 connected with 192.168.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-16.4 sec   256 KBytes   128 Kbits/sec
```

まとめ

- ルーティングテーブルのしくみ
  - エントリのまとめかた
    - ロンゲストマッチ
    - デフォルトルート
- OpenFlowでの実装方法
- ネットワークネームスペースでの動作確認