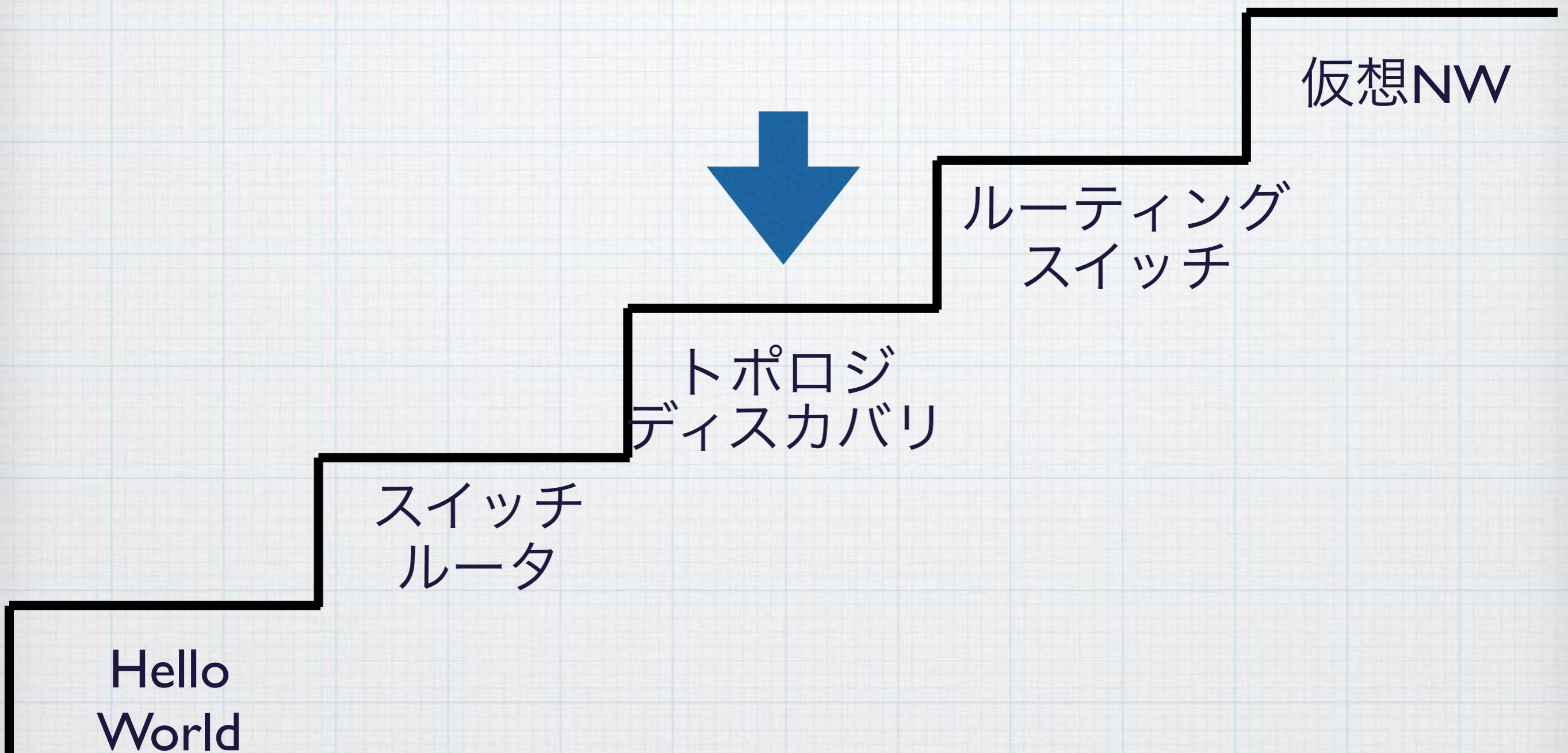
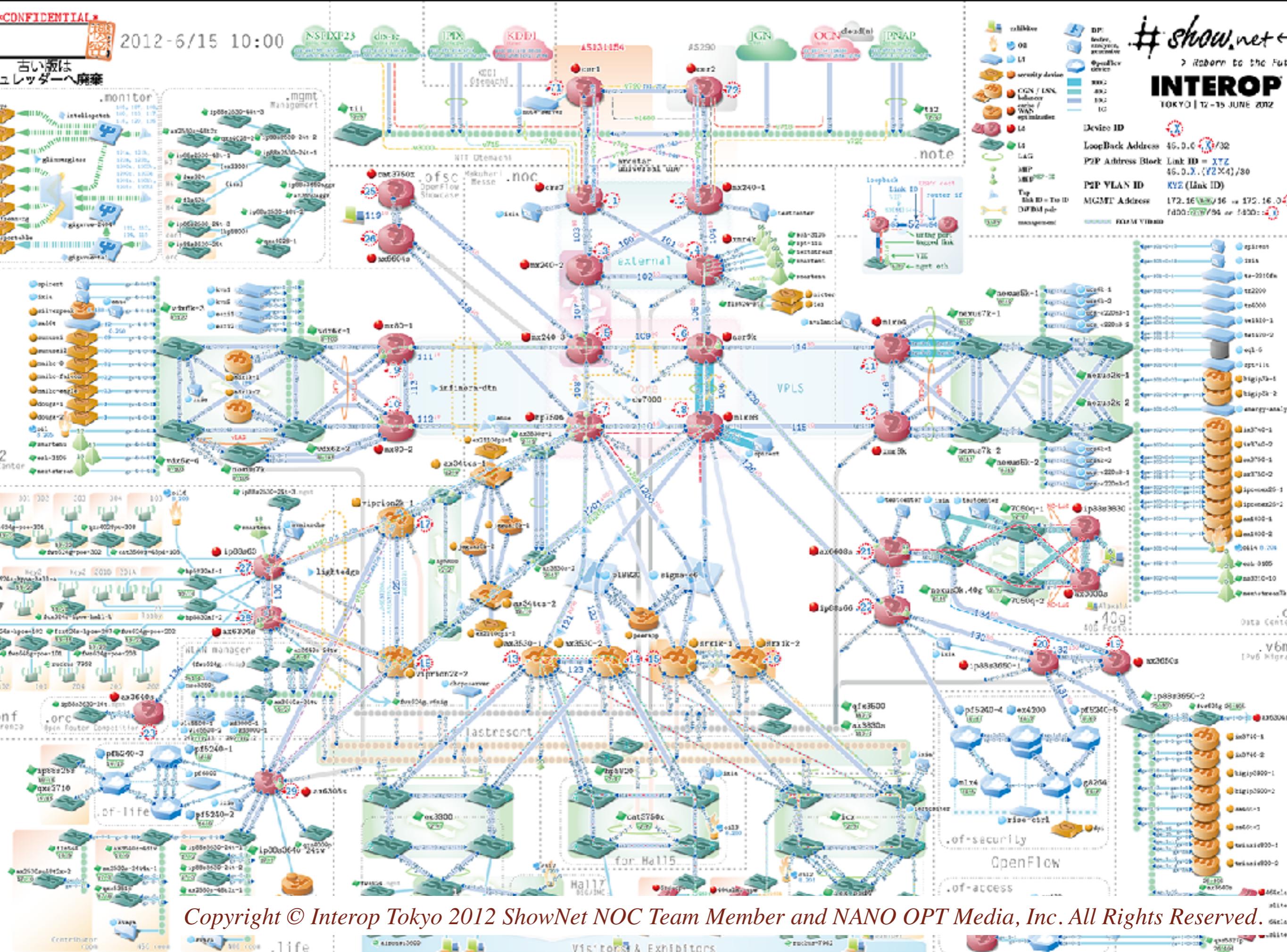
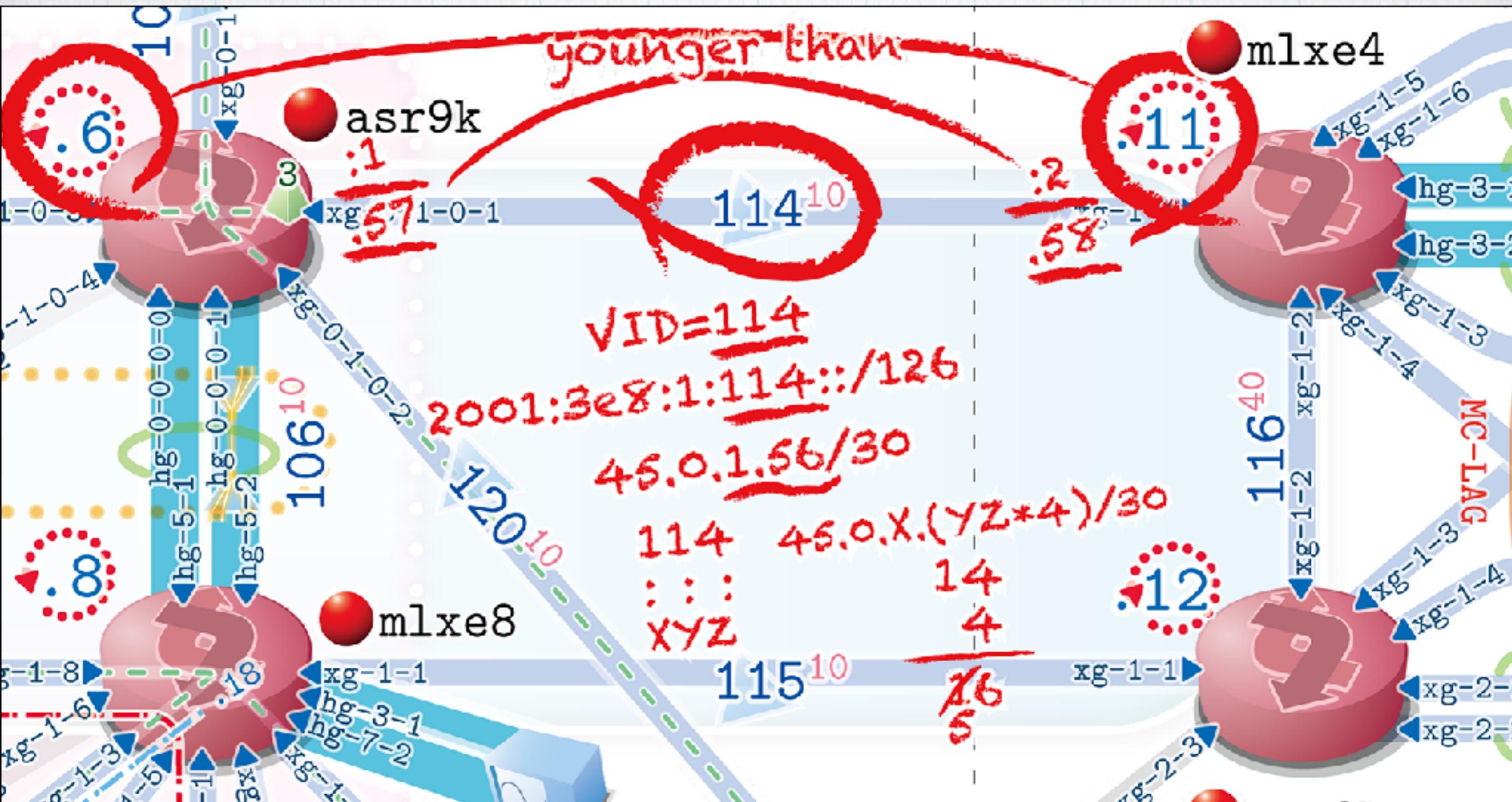


# ネットワークトポロジを 検出しよう

高宮 安仁 @yasuhito



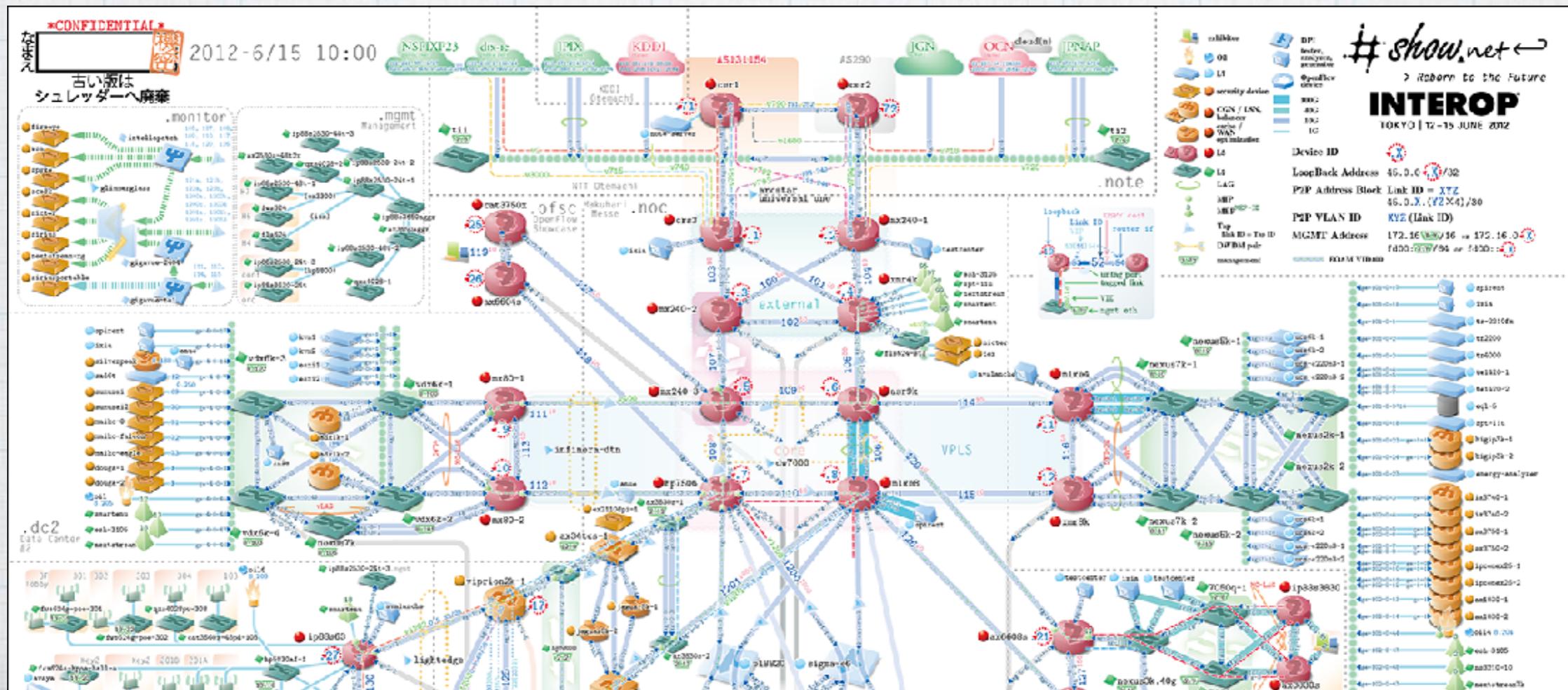




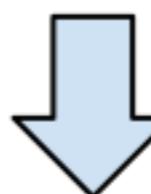
# トポロジ

- 初期構成は静的に与えることができる
- しかし、障害や構成変更で動的に変わる

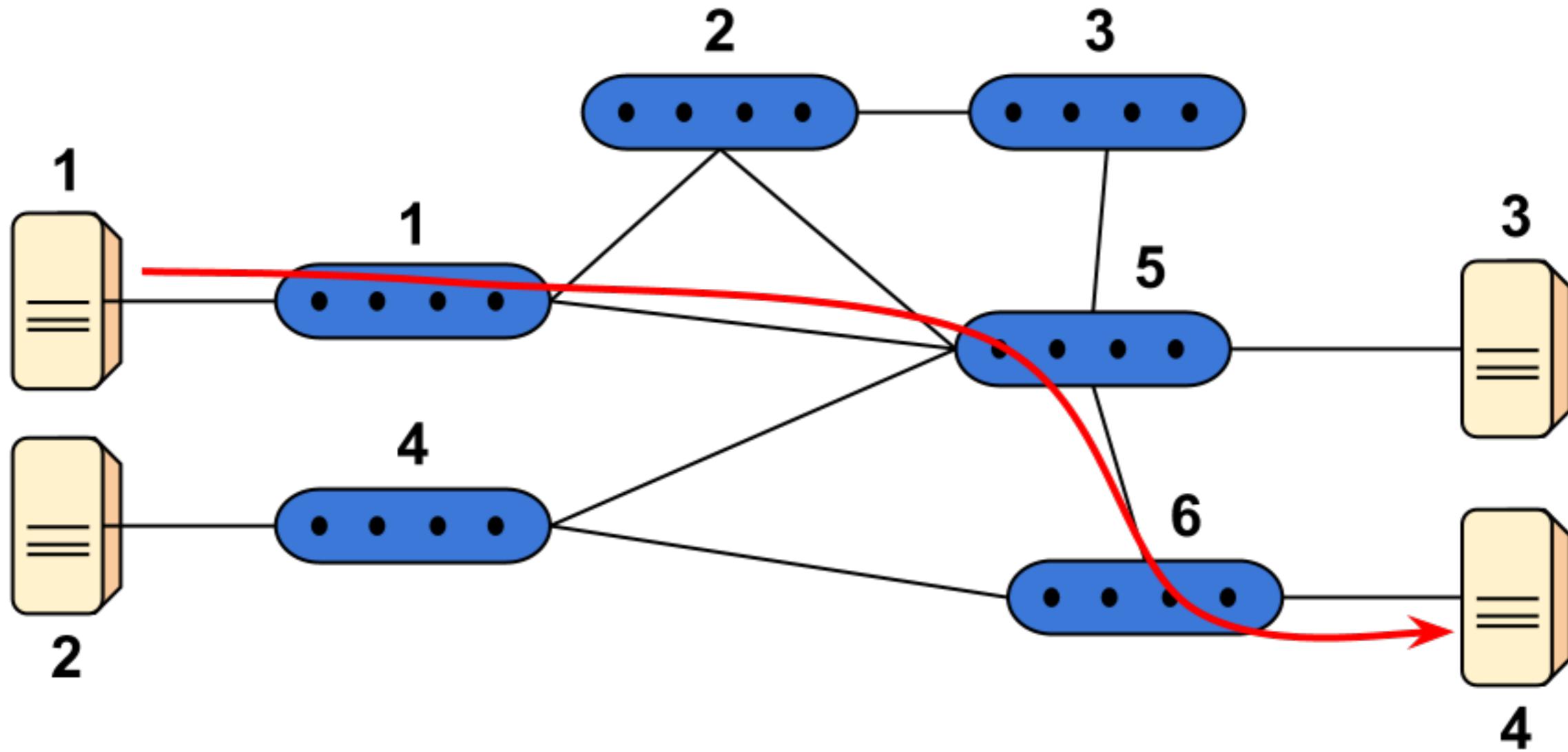
→ トポロジの変化を検知できるといろいろ便利

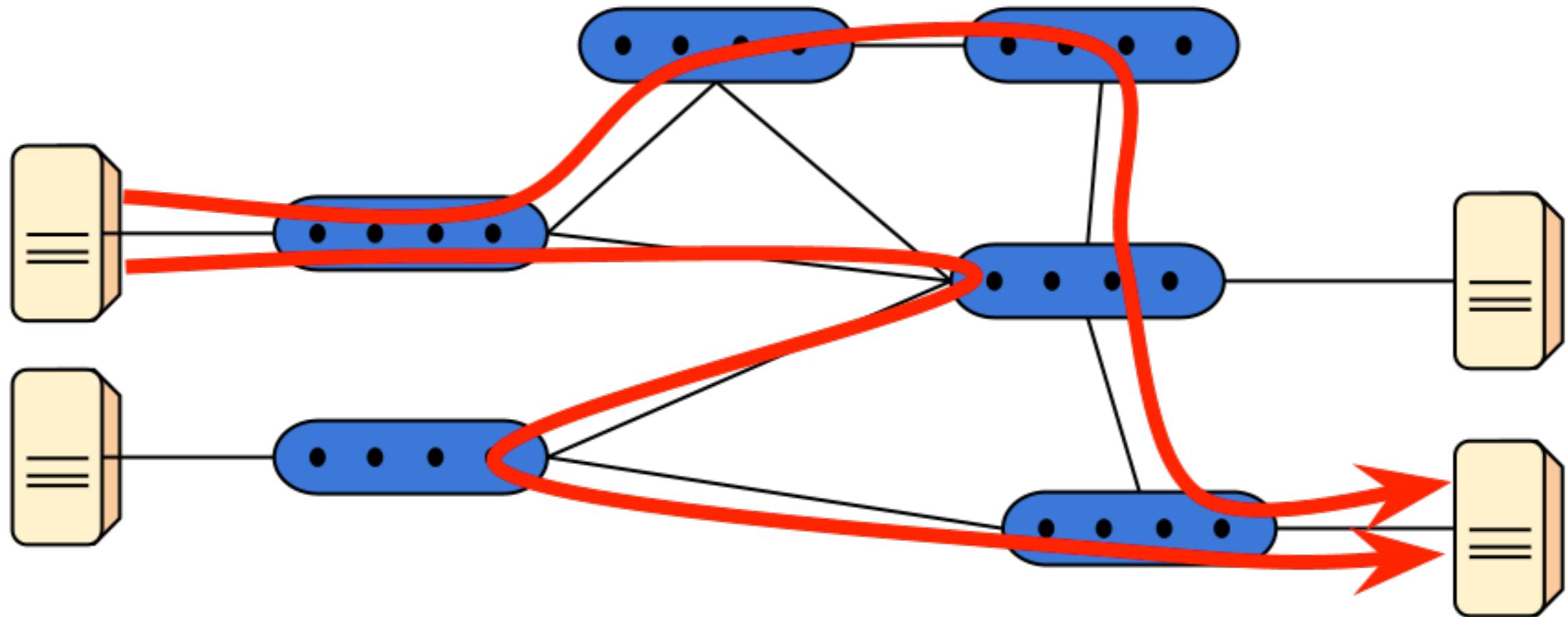


## ルーティングスイッチ



フローの書き込み





- マルチパスで帯域をかせぐ/有効利用
- 最短経路以外でもルーティング可能

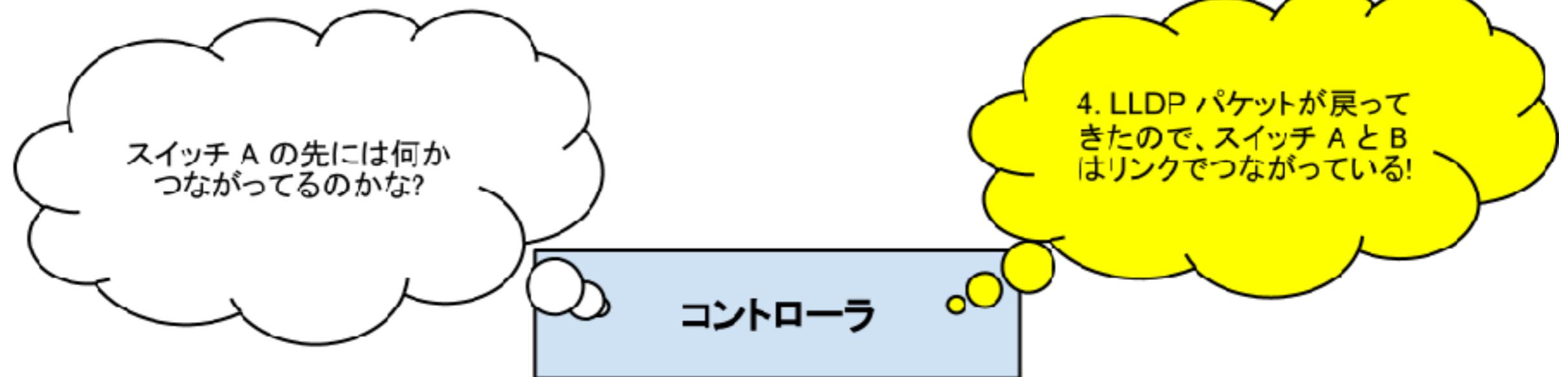
# トポロジの検知

## Link Layer Discovery Protocol (IEEE 802.1AB)

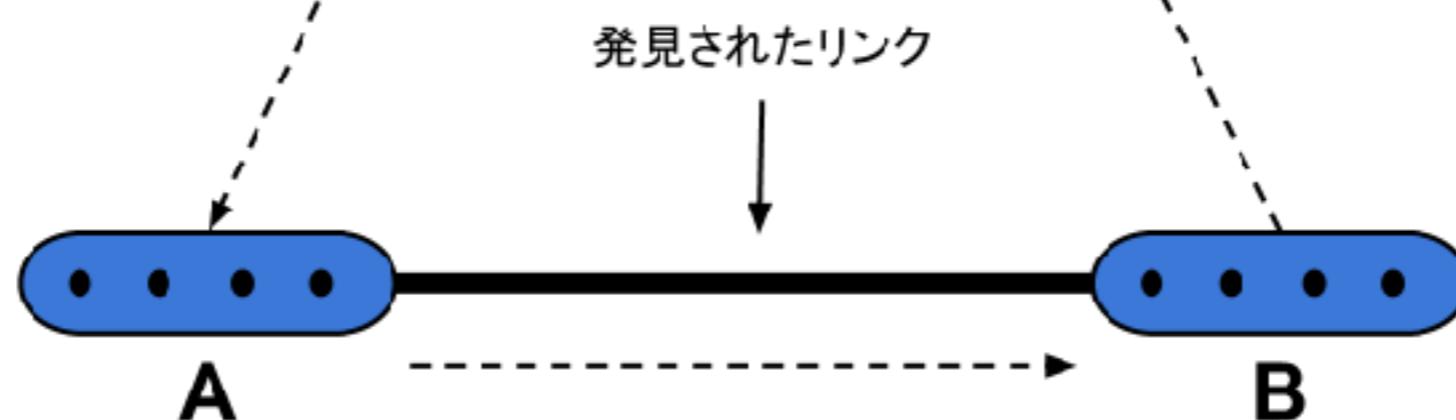
データリンク層の接続を検出するための

一般的なプロトコル

Destination MAC	Source MAC	Ethertype	Chassis ID TLV	Port ID TLV	Time to live TLV	Optional TLVs	End of LLDPDU TLV
01:80:c2:00:00:0e, or 01:80:c2:00:00:03, or 01:80:c2:00:00:00	Station's address	0x88CC	Type=1	Type=2	Type=3	Zero or more complete TLVs	Type=0, Length=0

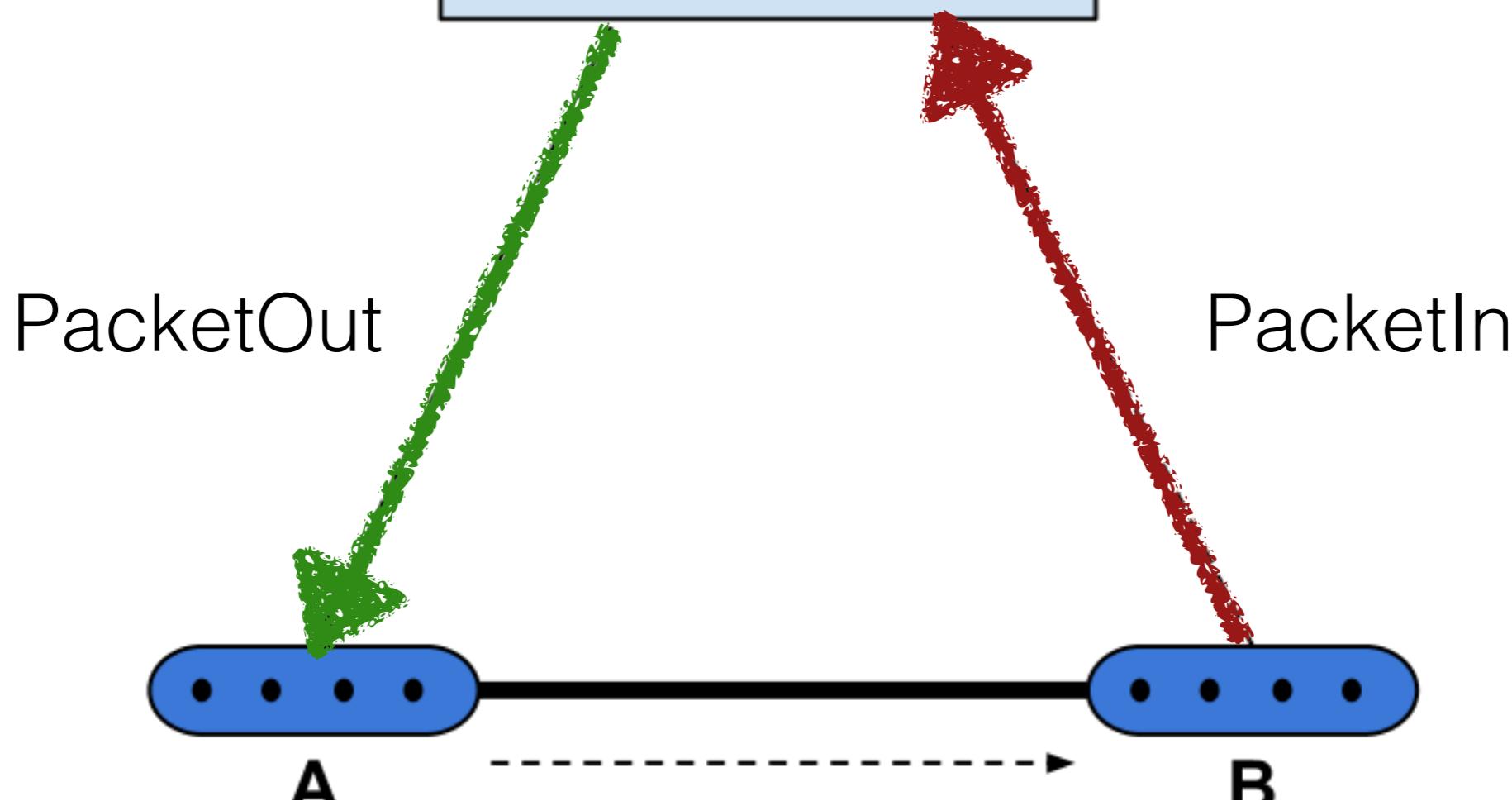


1. たまに LLDP パケットを送信
3. コントローラへと戻る



スイッチ A の先には何か  
つながってるのかな?

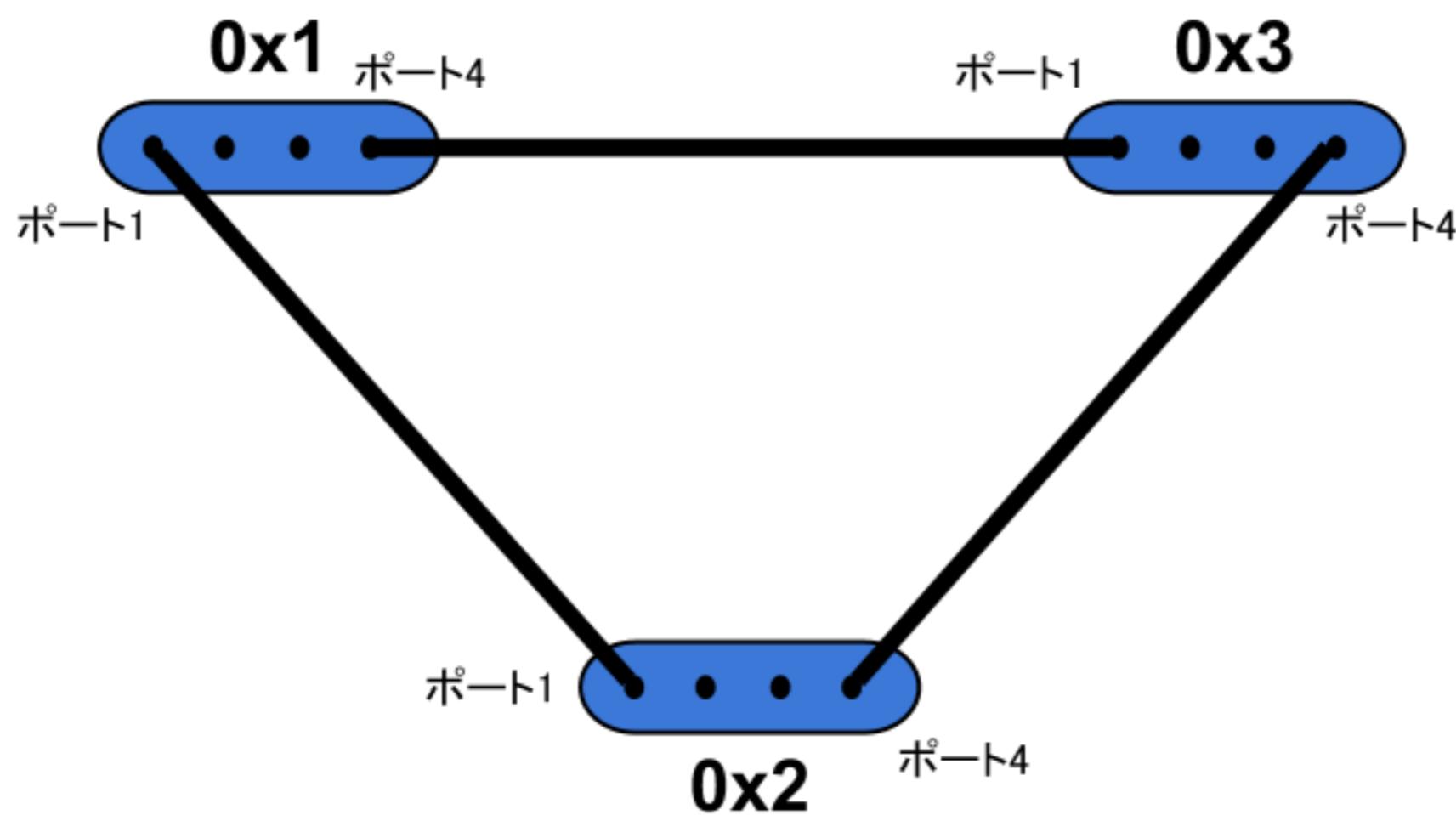
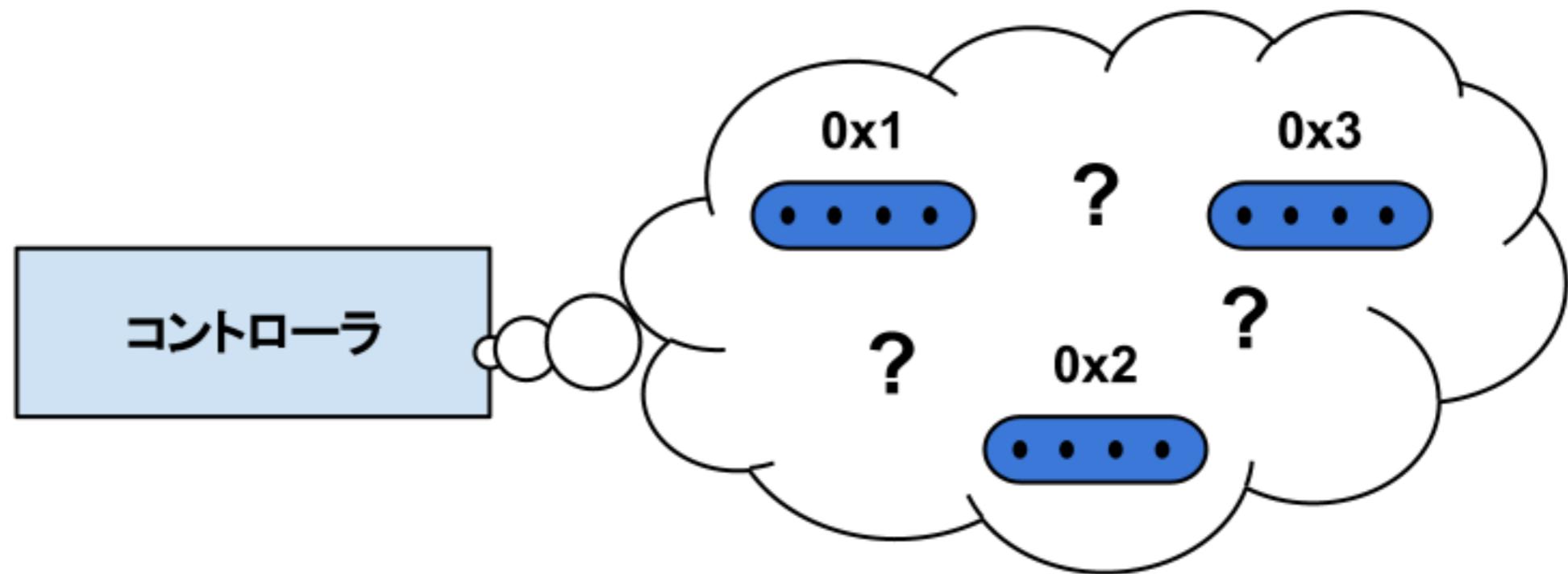
さて、スイッチ A と B  
はリンクでつながっている!

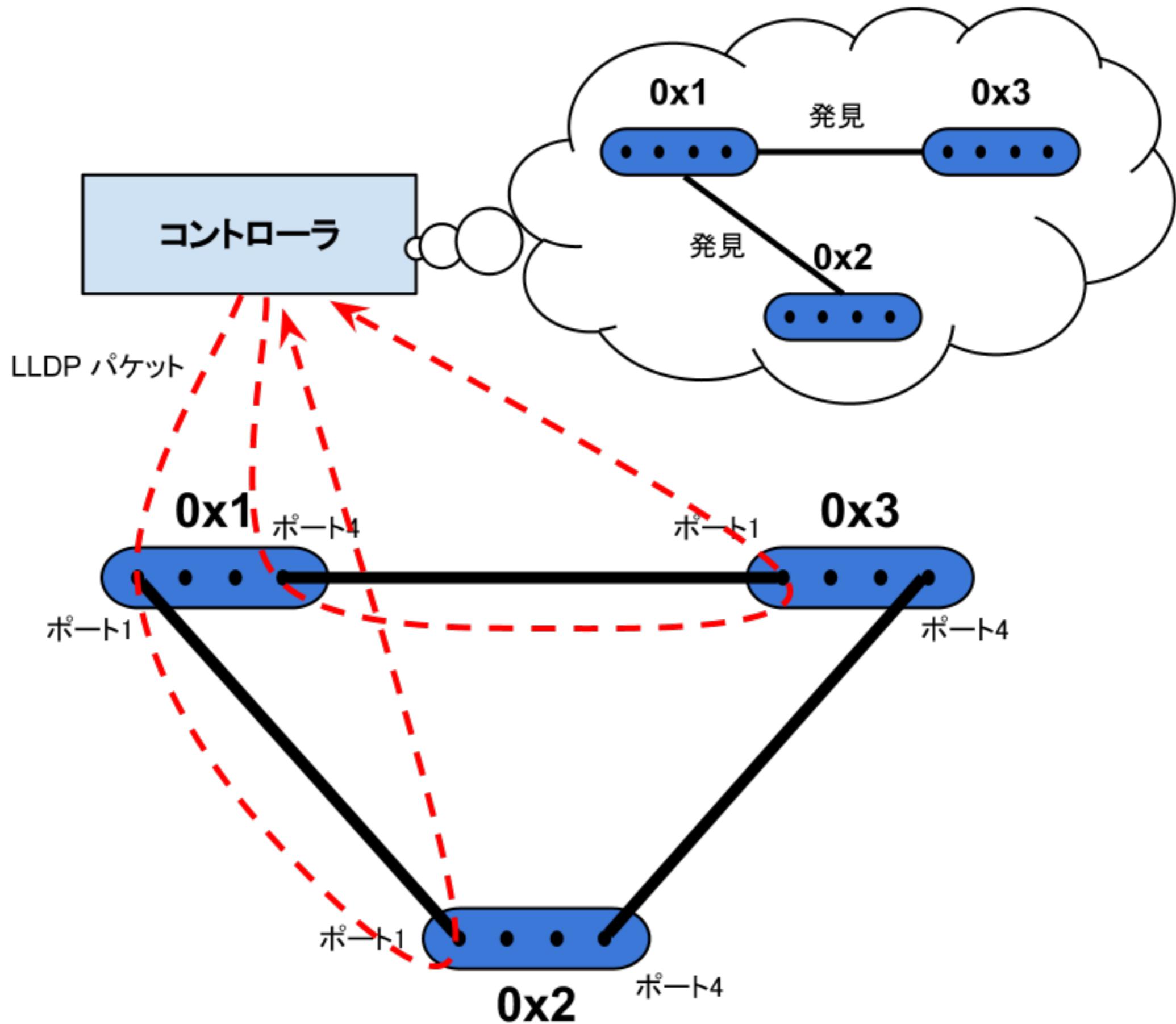


Packet In メッセージ  
受信ポート = 1

LLDP パケット  
スイッチ = 0x1, 送信ポート = 4

スイッチ A のポート 4 ⇄ スイッチ B のポート 1

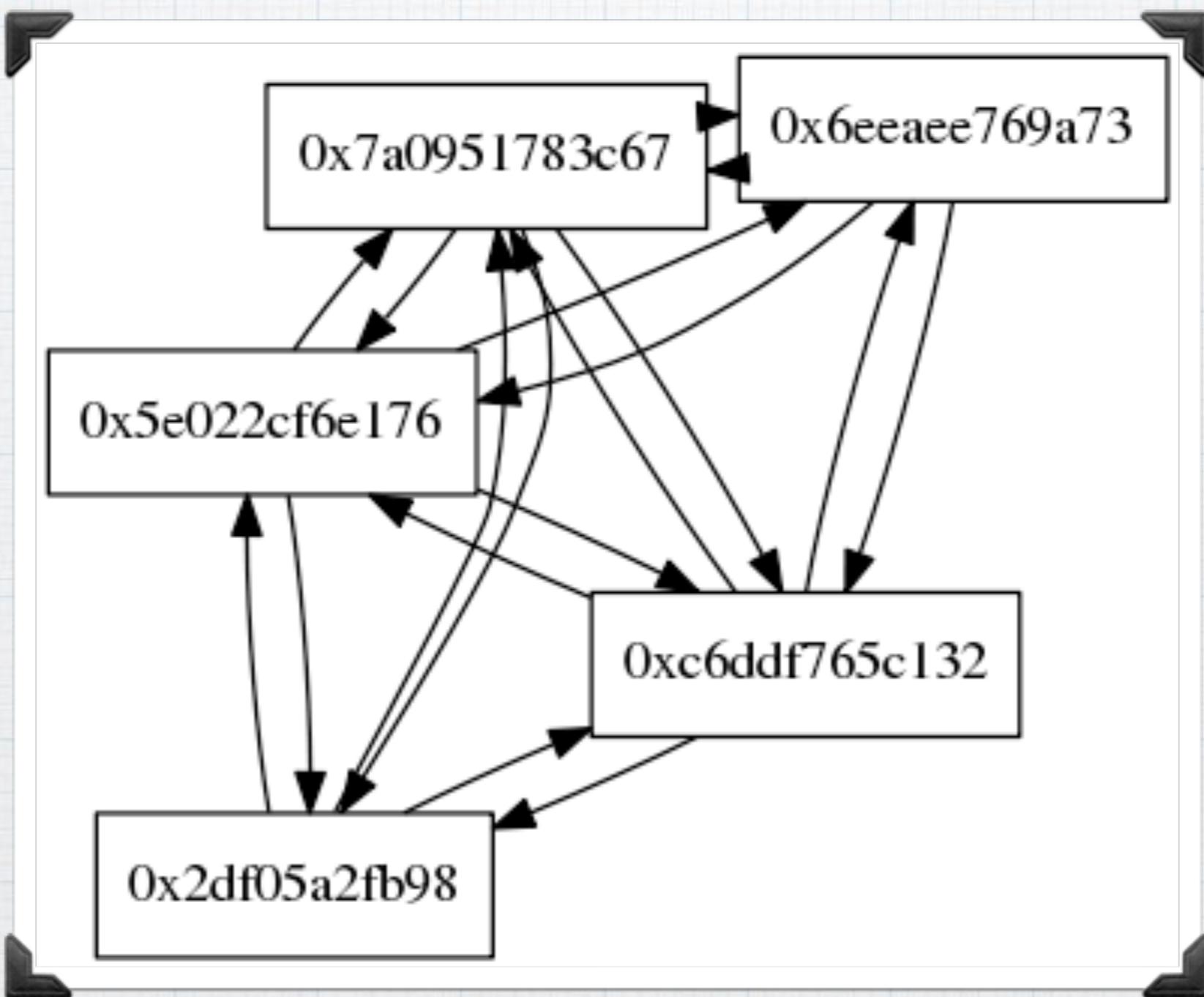




# トポロジ実装例

- topology コントローラ
- トポロジの検知と表示(テキスト/画像)をRubyコード約300行で実装
- 今回の課題は、このコードを元に改造すること

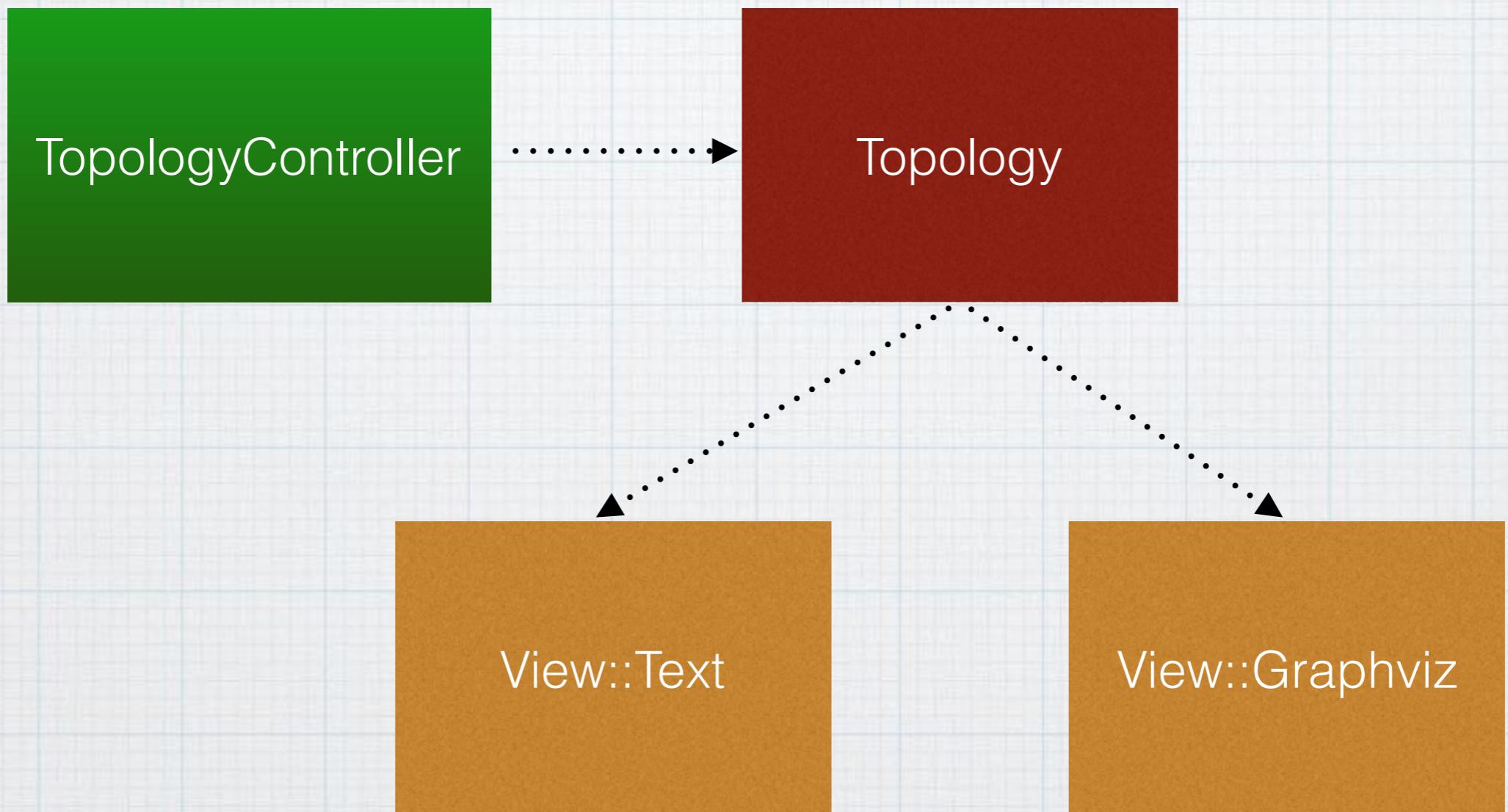
# 実行結果



# 今日やること

1. topology をインストール
2. スイッチ 10 台程度の物理ネットワークを作る
3. 実際にトポロジ画像が表示されることを確認
4. topology のコードを理解

# topology の構造



コントローラ本体

トポロジ情報の  
管理

TopologyController

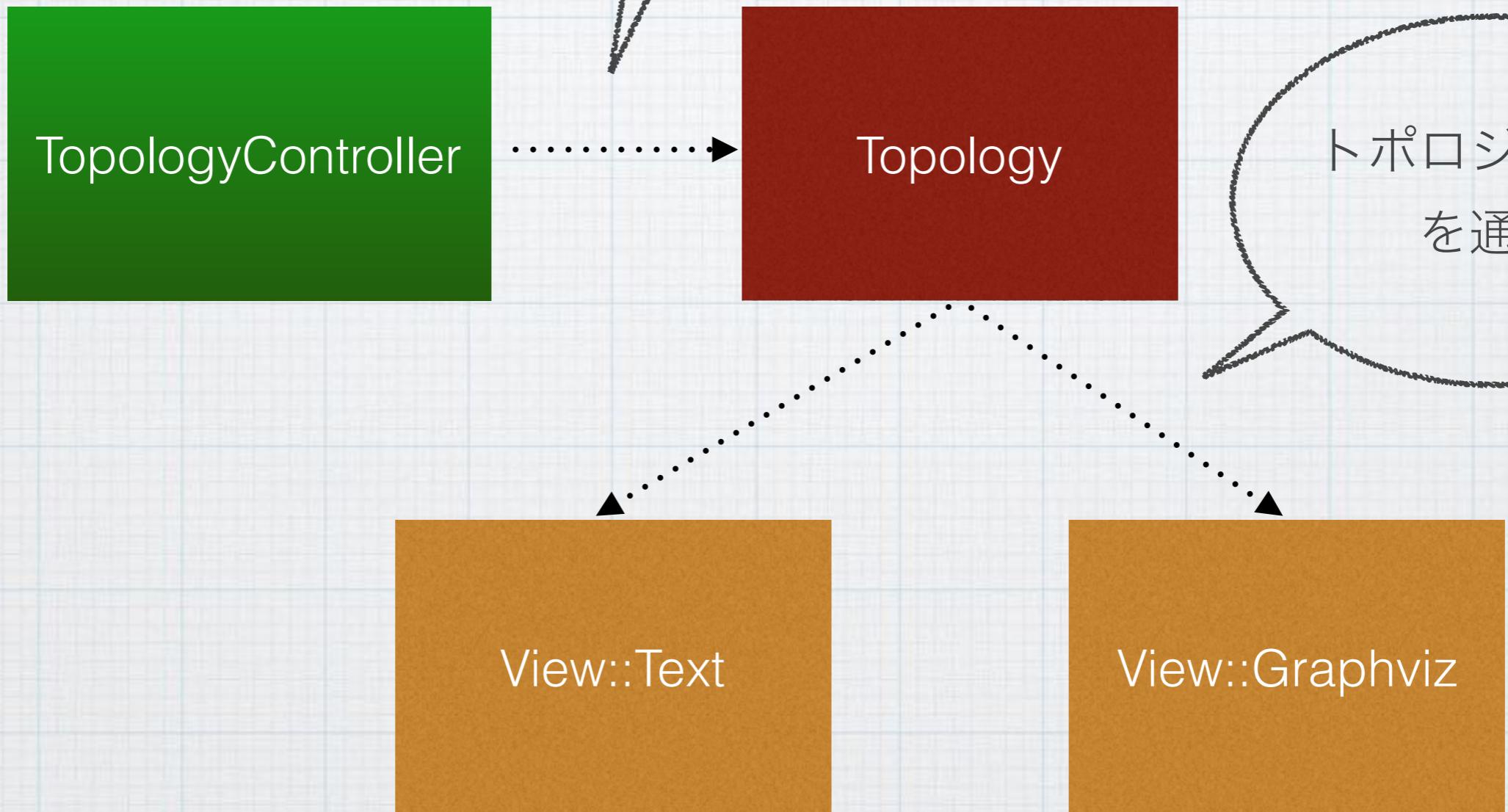
Topology

表示  
(テキスト/画像)

View::Text

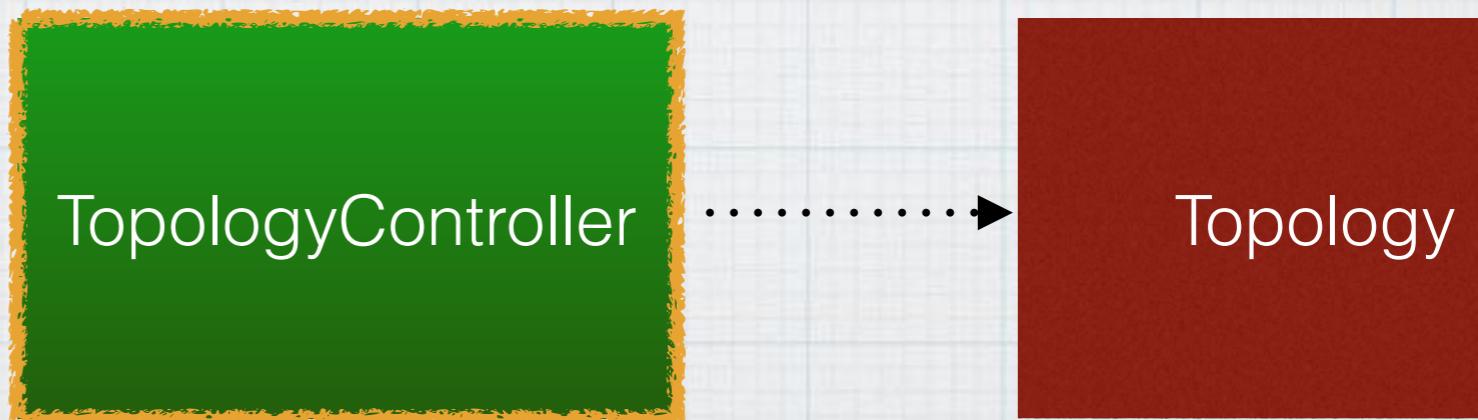
View::Graphviz

トポロジ情報の  
管理



# TopologyController

- スイッチのポート一覧を取得  
FeaturesRequest, FeaturesReply
- ポート情報の更新  
#port\_modify
- LLDP を定期的にばらまく  
#flood\_lldp\_frames
- リンク情報の更新  
#packet\_in



# スイッチのポート一覧

```
def switch_ready(dpid)
  send_message dpid, Features::Request.new
end
```

```
def features_reply(dpid, features_reply)
  @topology.add_switch dpid, features_reply.physical_ports.select(&:up?)
end
```

- トポロジ情報 (@topology) にみつかったスイッチ情報を追加
- physical\_ports.select(&:up?)  
物理ポートのうち“UP”なもの

# ポート状態の変更(UP/DOWN)

```
def port_modify(_dpid, port_status)
    updated_port = port_status.desc
    return if updated_port.local?
    if updated_port.down?
        @topology.delete_port updated_port
    elsif updated_port.up?
        @topology.add_port updated_port
    else
        fail 'Unknown port status.'
    end
end
```

- ポートがDOWNだったらトポロジ情報からポートを消す
- ポートがUPだったらトポロジ情報に追加

# 新しいリンクとホストの検知

```
def packet_in(dpid, packet_in)
    if packet_in.lldp?
        @topology.maybe_add_link Link.new(dpid, packet_in)
    else
        @topology.maybe_add_host(packet_in.source_mac,
                                packet_in.source_ip_address,
                                dpid,
                                packet_in.in_port)
    end
end
```

- LLDPが返ってきたらリンク追加

- それ以外はホストを追加

# リンク情報をLLDPから取り出す

```
class Link
  def initialize(dpid, packet_in)
    lldp = packet_in.data
    @dpid_a = lldp.dpid
    @dpid_b = dpid
    @port_a = lldp.port_number
    @port_b = packet_in.in_port
  end
```

- スイッチ @dpid\_a のポート @port\_a と、  
スイッチ @dpid\_b のポート @port\_b にリンク  
がある

# 定期的にLLDPをばらまく

```
class TopologyController < Trema::Controller
  timer_event :flood_lldp_frames, interval: 1.sec

  def flood_lldp_frames
    @topology.ports.each do |dpid, ports|
      send_lldp dpid, ports
    end
  end

  private

  def send_lldp(dpid, ports)
    ports.each do |each|
      port_number = each.number
      send_packet_out(
        dpid,
        actions: SendOutPort.new(port_number),
        raw_data: ll dp_binary_string(dpid, port_number)
      )
    end
  end
```

1秒ごとにばらまく

すべてのポートに対して…

LLDPを送る

# LLDPのパースと生成

```
# 口ード
require "pio"

# LLDPのパース
Pio::Lldp.read(packet_in.data).dpid #=> 12345

# 生成
lldp = Pio::Lldp.new(:destination_mac => ...,
                      :source_mac => ...)

# 送信
send_packet_out(dpid,
                 :data => lldp.to_binary,
                 ...)
```

# Topology

Observer Pattern で変更を View に通知

- ポートの追加と削除

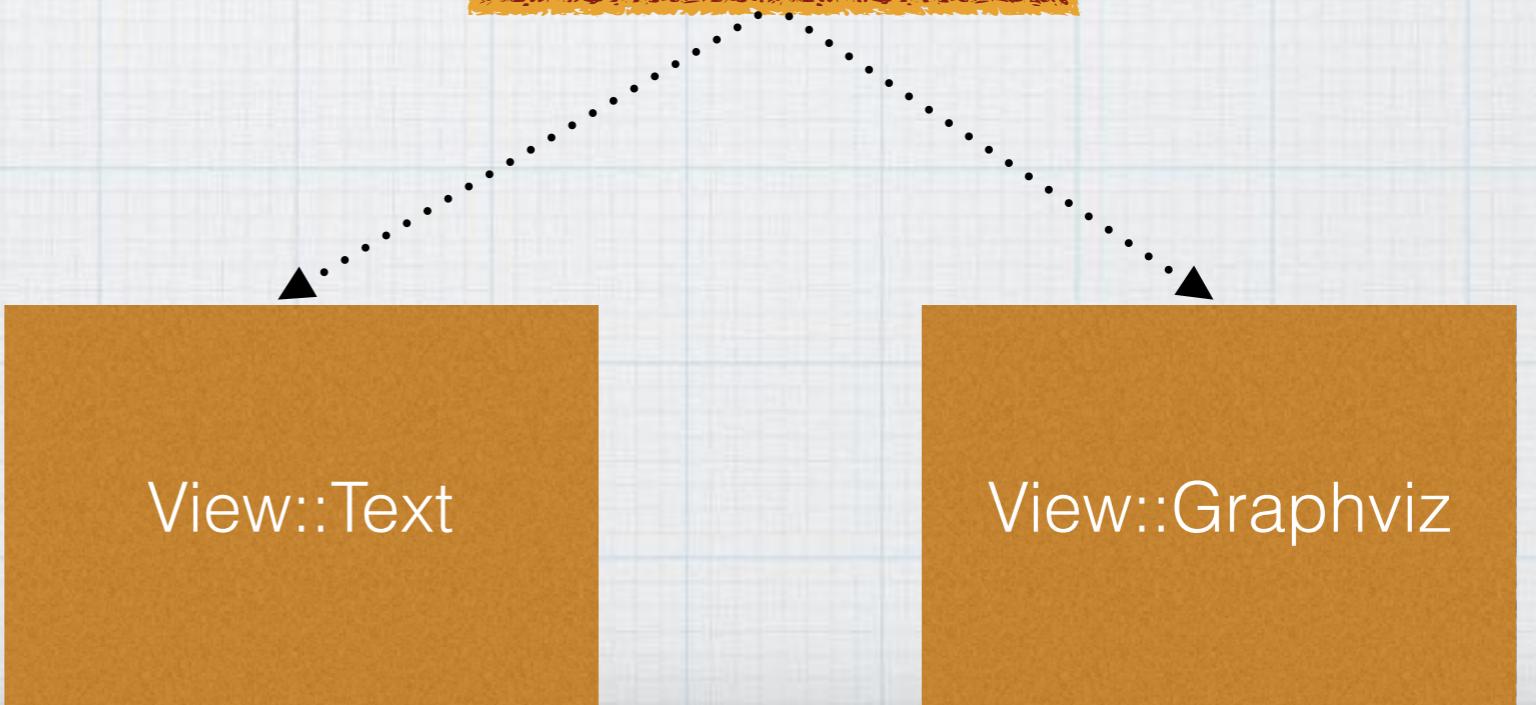
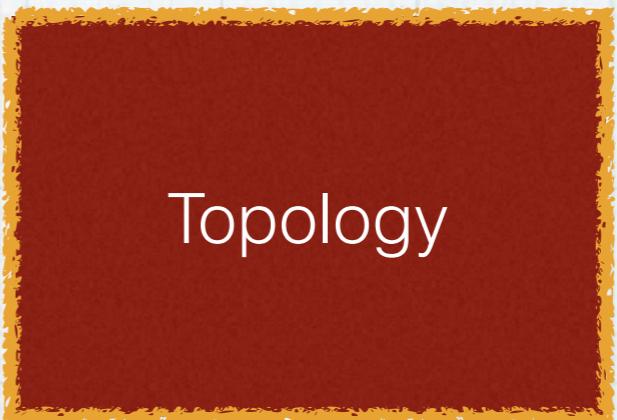
#add\_port, #delete\_port

- ポート状態の変更 (up/down)

#update\_port

- リンクの追加

#add\_link\_by



# オブザーバパターン

```
# TopologyController#start
def start(args)
  @command_line = CommandLine.new(logger)
  @command_line.parse(args)
  @topology = Topology.new
  @topology.add_observer @command_line.view
  logger.info "Topology started (#{@command_line.view})."
end
```

- ・トポロジ情報をビュー  
(@command\_line.view) が監視
- ・トポロジに更新があったら通知してもらう

# 「スイッチ追加」の通知

TopologyController

```
def features_reply(dpid, features_reply)
  @topology.add_switch dpid,
  features_reply.physical_ports.select(&:up?)
end
```

Topology

```
def add_switch(dpid, ports)
  ports.each { |each| add_port(each) }
  maybe_send_handler :add_switch, dpid, self
end
```

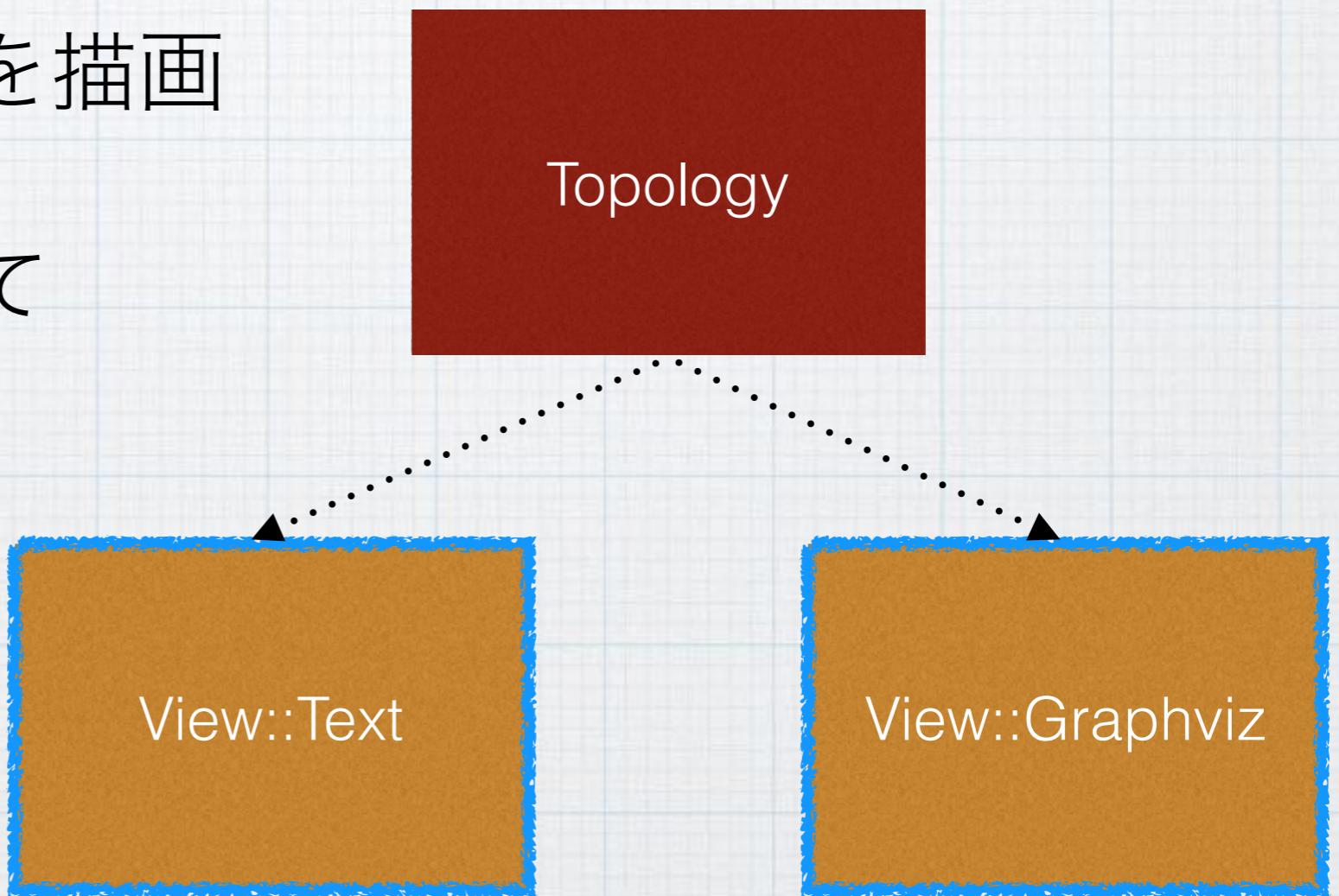
View::Text

```
def add_switch(dpid, topology)
  show_status("Switch #{dpid.to_hex} added",
             topology.switches.map(&:to_hex))
end
```

# View::\*

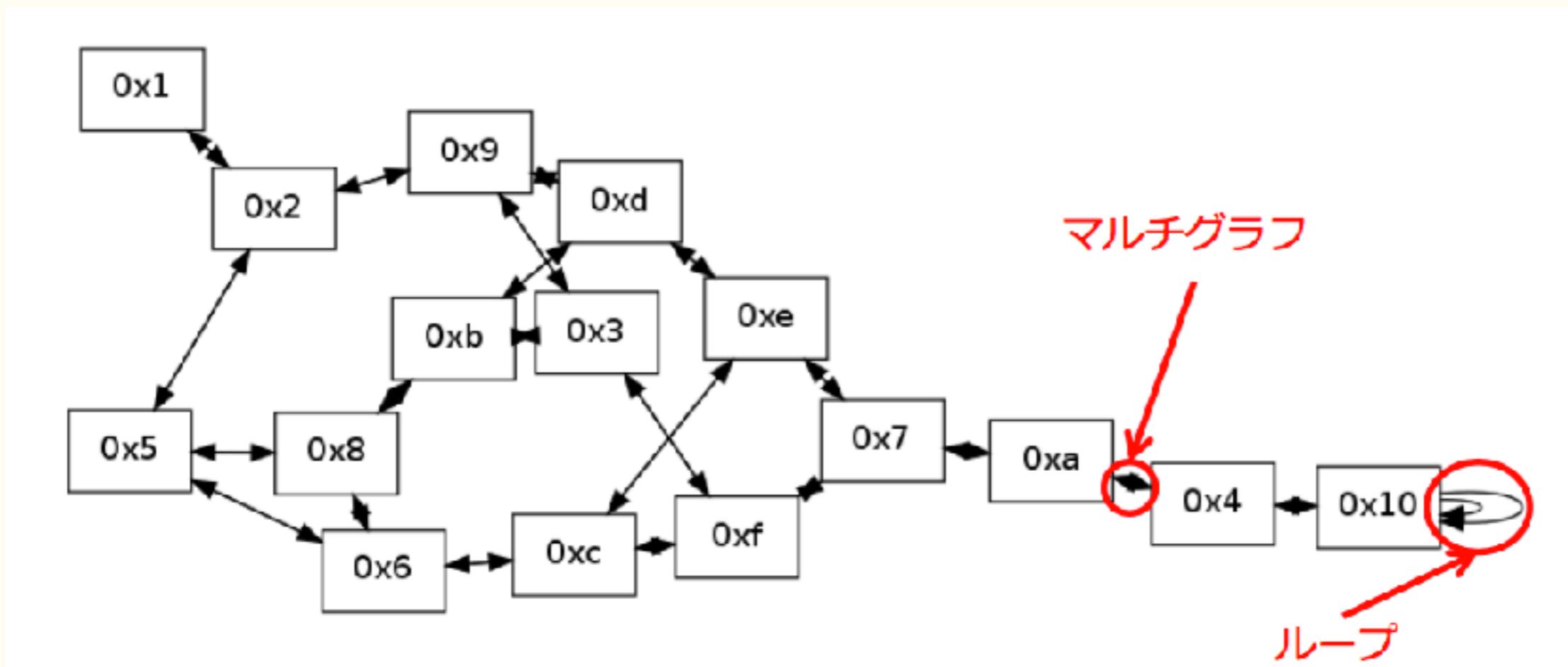
Observer Pattern で Topology の変化を監視

- トポロジの変化を描画
- 表示形式に応じて  
クラスを追加

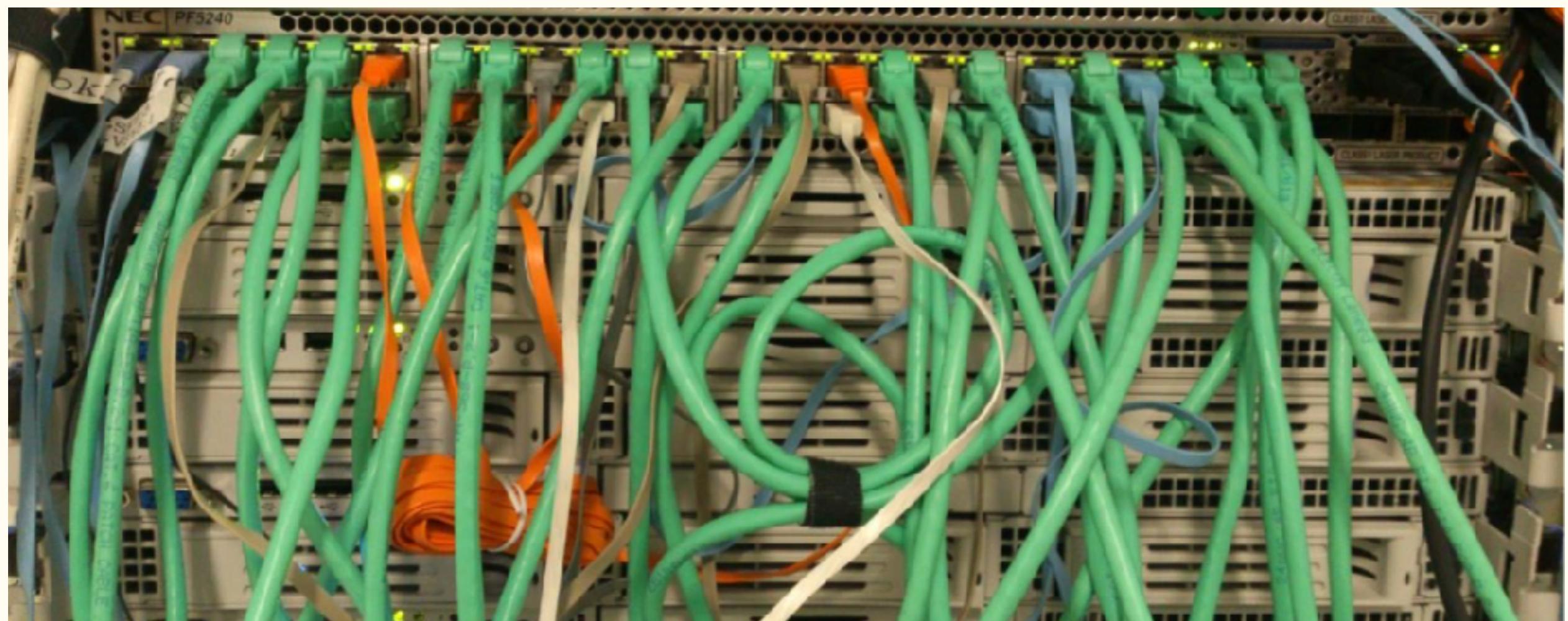


# レポート課題 1

1. 実機スイッチ上に VSI x16 を作成 (各VSIは2ポート以上)
2. 全ポートを適当にケーブリング
3. Topologyを使ってトポロジを表示
4. ケーブルを抜き差ししてトポロジ画像が更新されることを確認

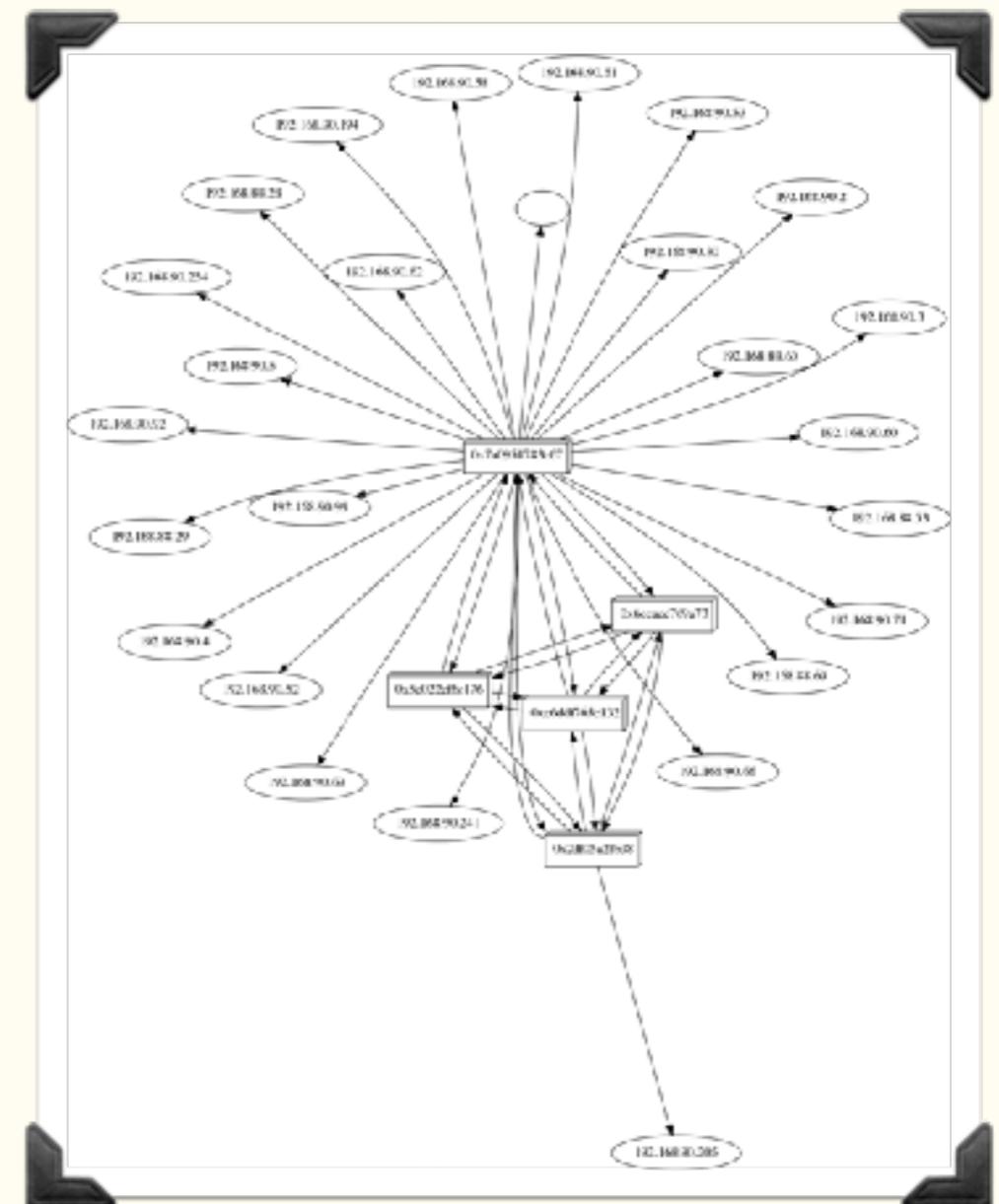


1. VSI x16
2. 各 VSI に 3 ポート割り当て
3. 適当にケーブリング



# レポート課題 2

- topology を改造し、スイッチに加えてホストの接続関係を表示
- ブラウザで表示する機能を追加  
おすすめライブラリは vis.js  
<https://github.com/almende/vis>



Graphviz版タンポポ

# vis.js

- こういう図が簡単に表示できます
- ヒント: View::VisJs クラスを追加

