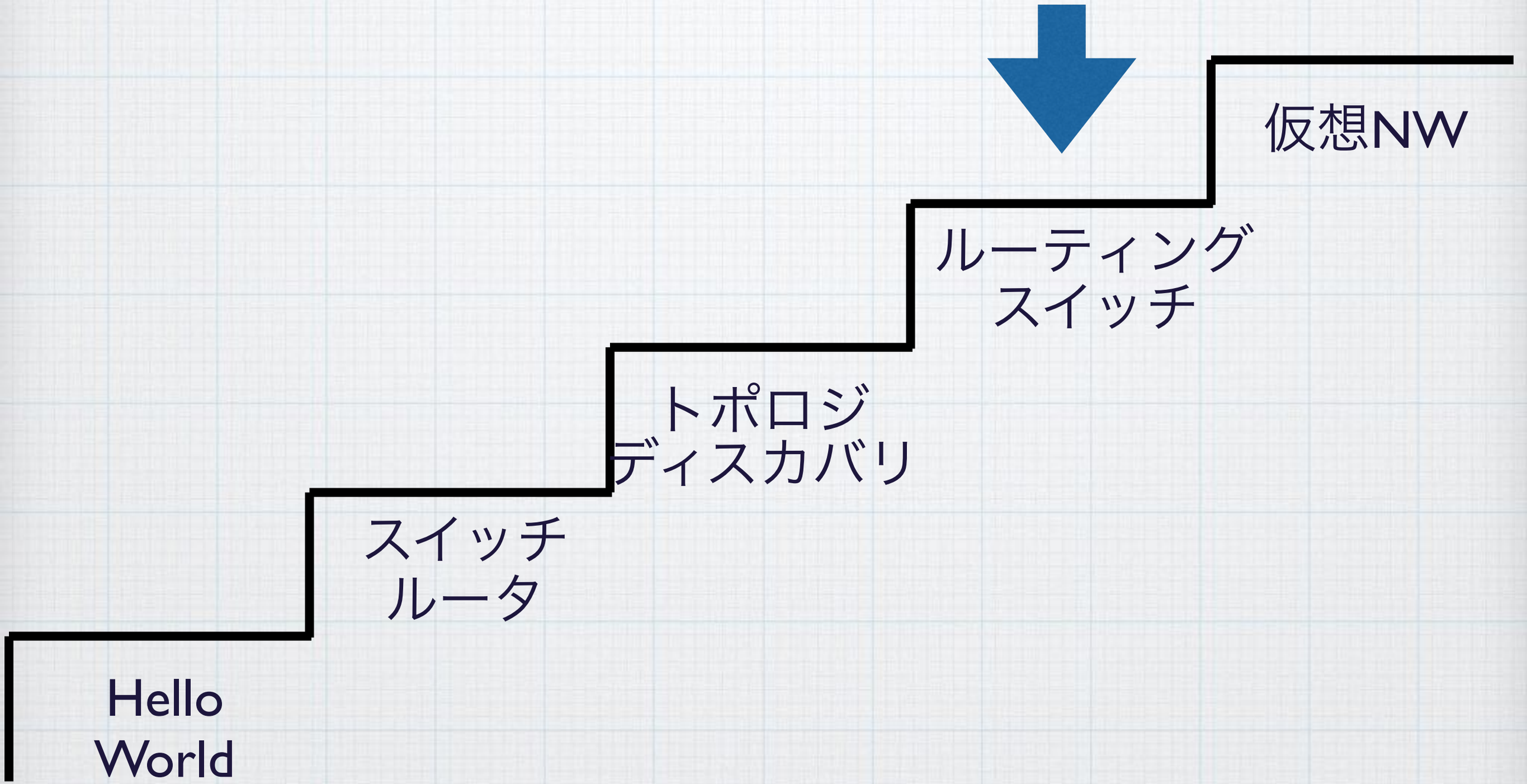
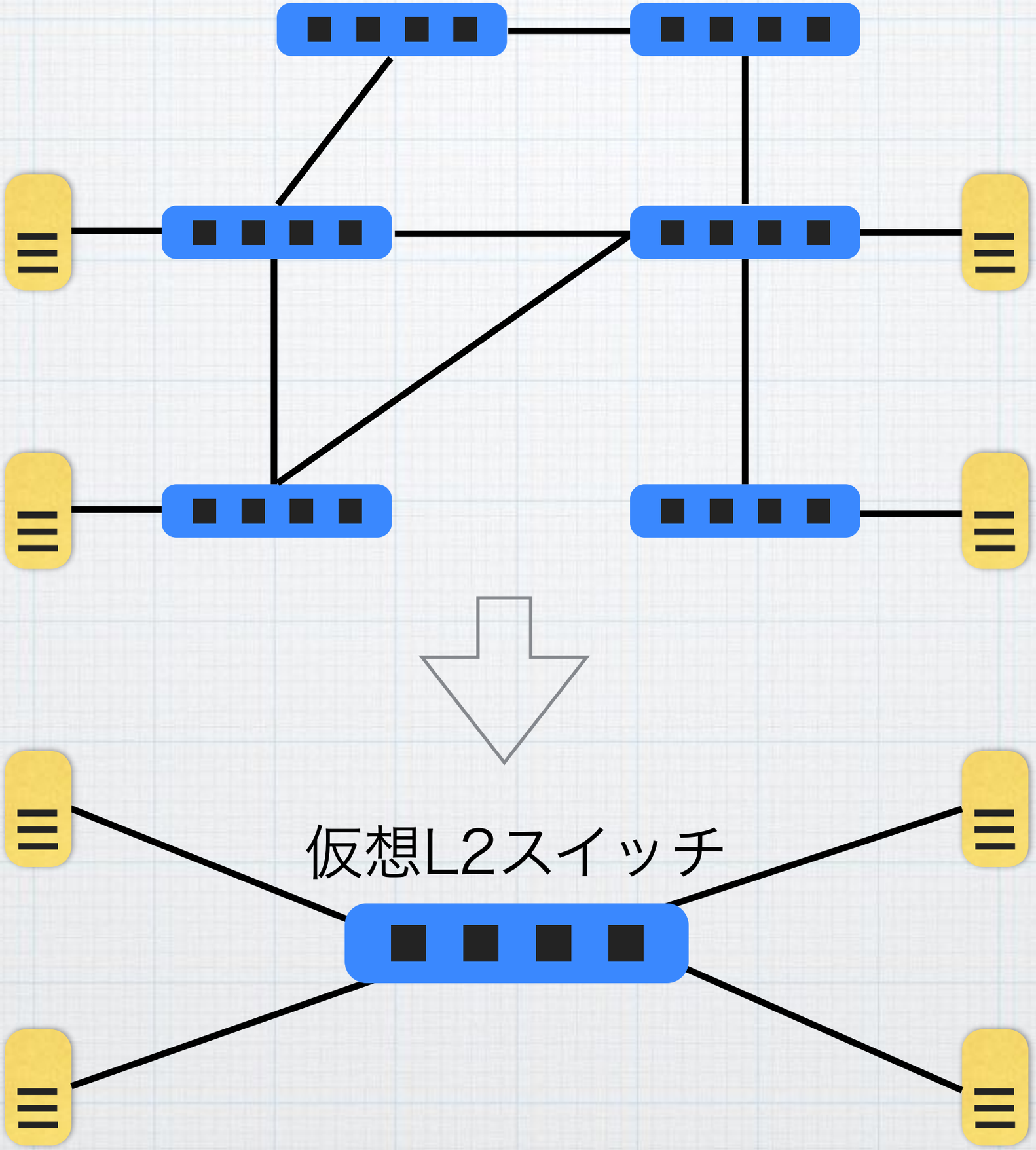
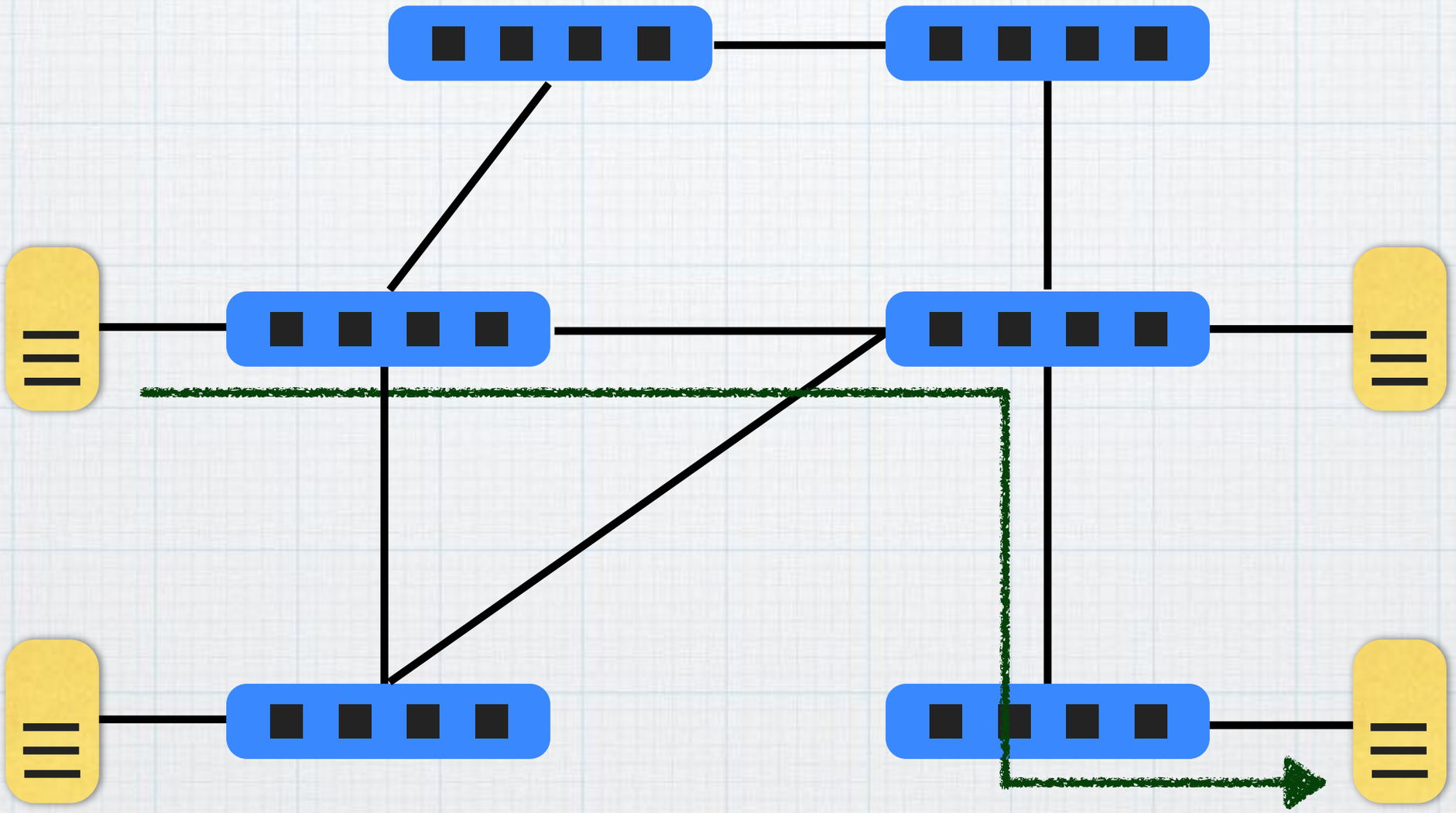
たくさんのスイッチを
制御する

高宮安仁 @yasuhito


仮想NW
ルーティング
スイッチ
トポロジ
ディスカバリ
スイッチ
ルータ
Hello
World


仮想L2スイッチ

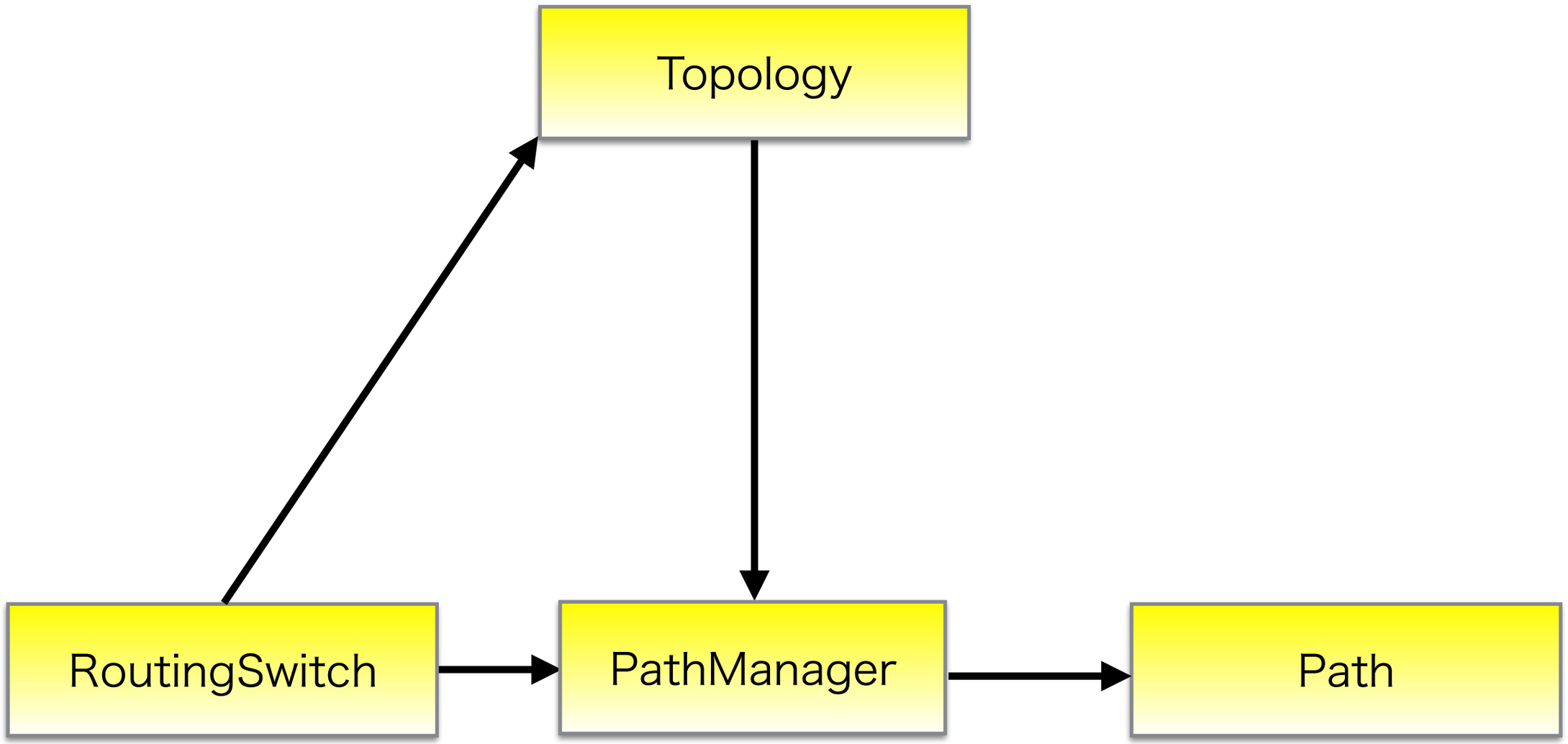
ルーティングの仕組み



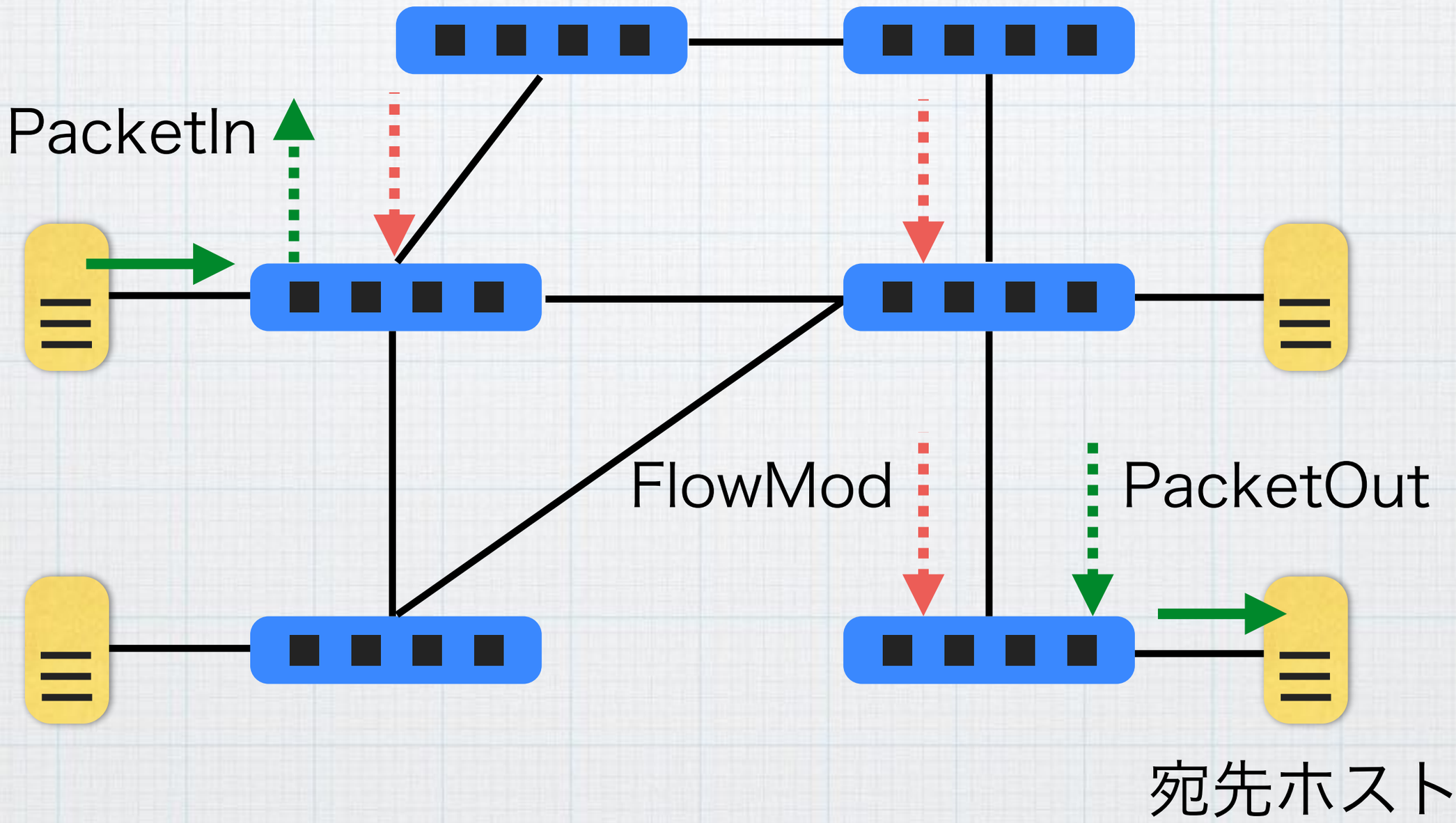
どう書く? コントローラ

- スイッチとの接続
- トポロジ探索
- 最短路
- フローエントリの管理

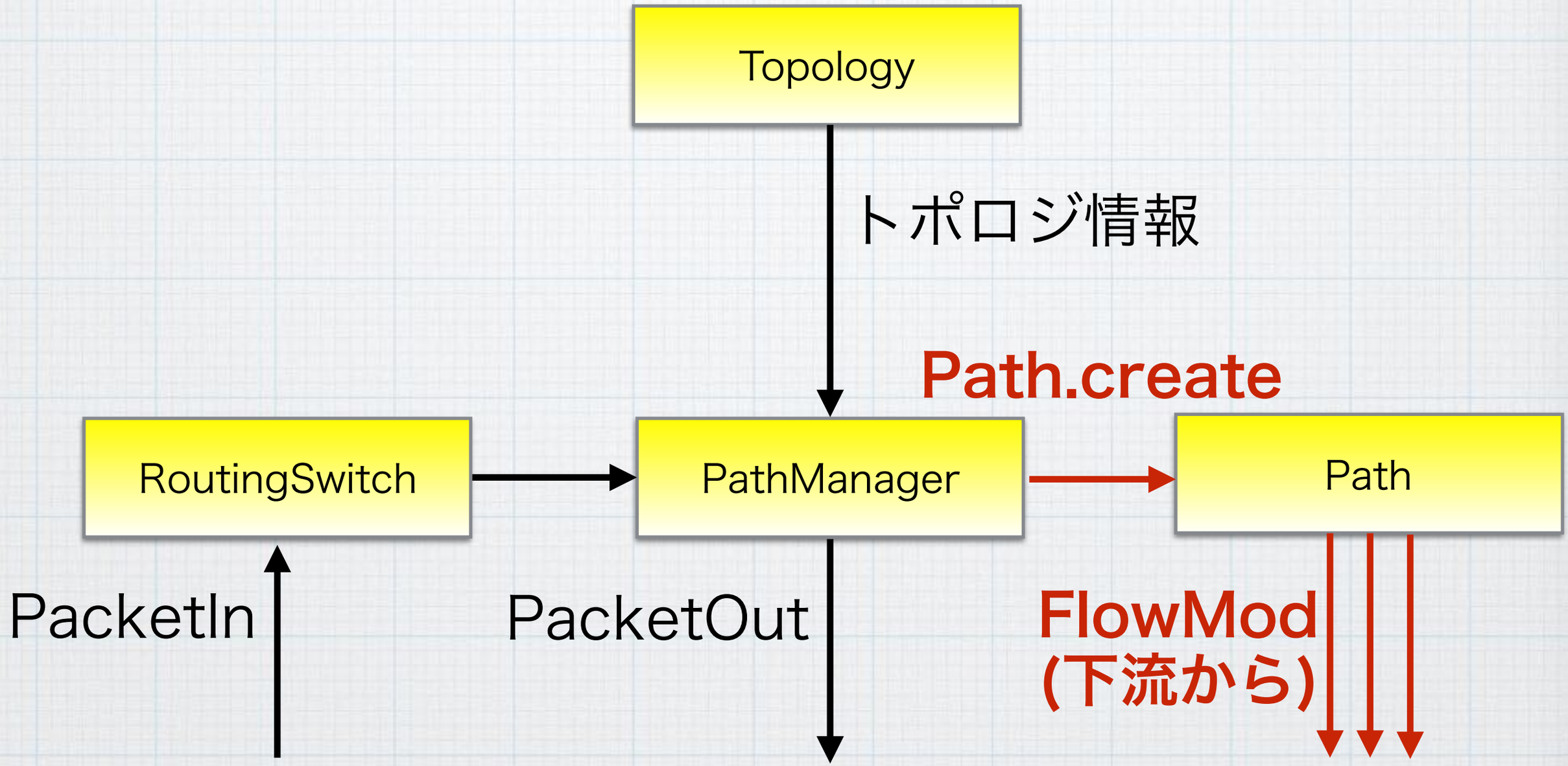
小さなクラスに分ける!



パケットを送信すると…



最短路パスを作る


Topology
トポロジ情報
Path.create
RoutingSwitch
PathManager
Path
PacketIn
PacketOut
FlowMod
(下流から)

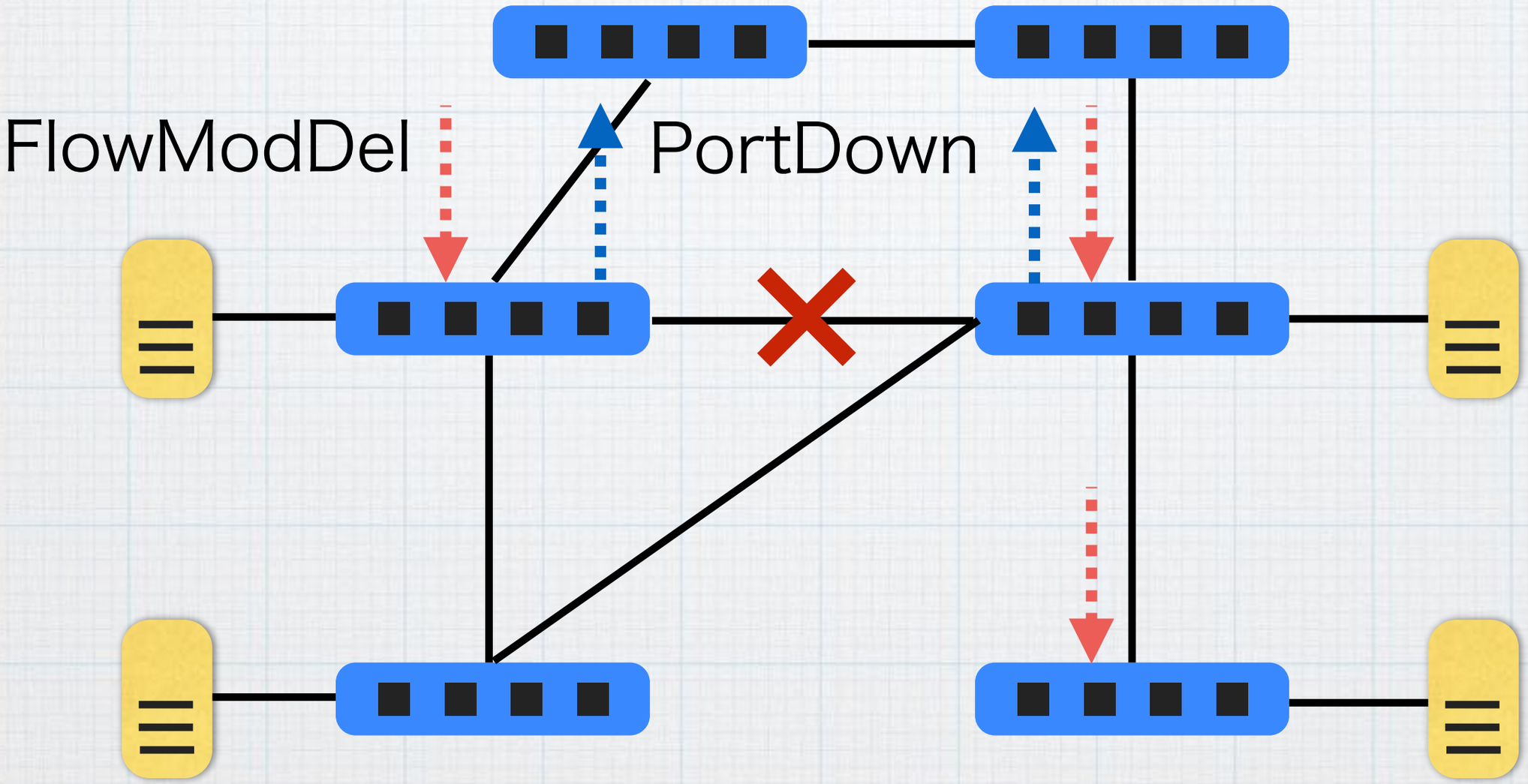
処理のカプセル化

- `Path.create(path, packet_in)`
  パスに沿って下流からFlowModAdd

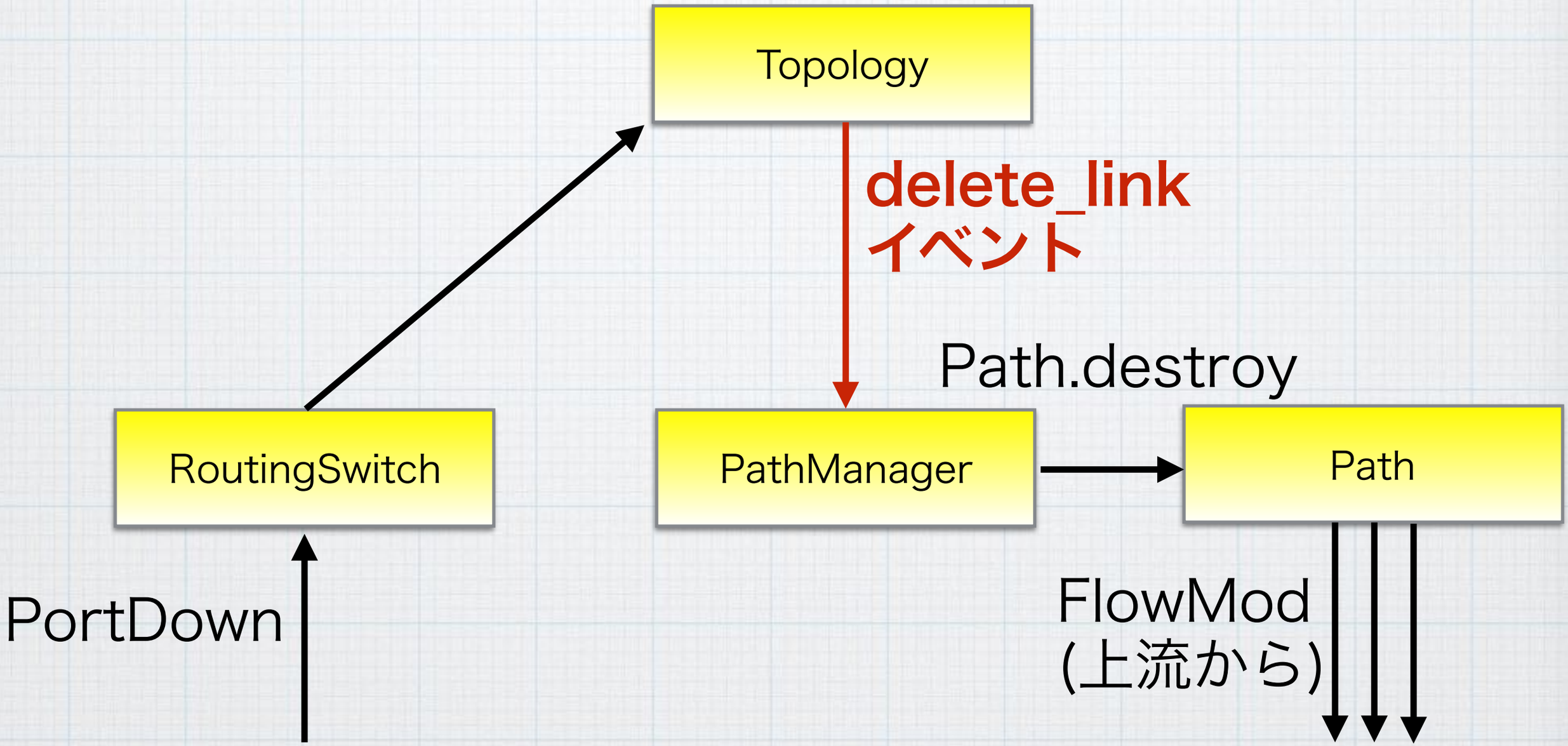
- `Path.destroy(path)`
  パスに沿って上流からFlowModDel

- ```
  Path.select do |each|
    each.link?(port_a, port_b)
  end.each(&:destroy)
  ```
  ポートa⇔ポートbのリンクをすべて消す
  ```

リンクが切れたら…



無効なパスを消す


Topology
delete_link
イベント
Path.destroy
RoutingSwitch
PathManager
Path
PortDown
FlowMod
(上流から)

オブザーバパターン

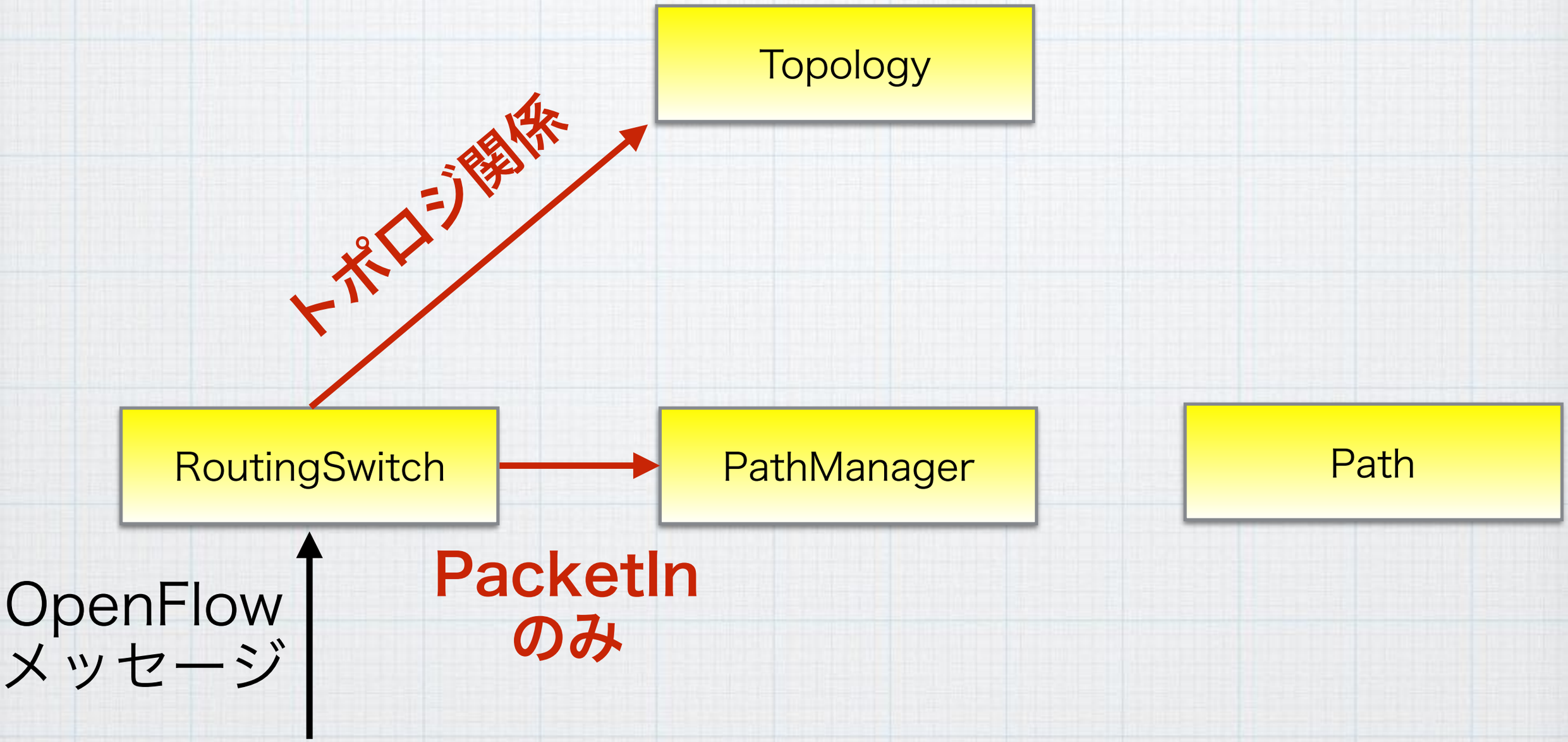
トポロジイベントをPathManagerへ

```
class RoutingSwitch < Trema::Controller
  # ...
  def start_topology
    TopologyController.new { |topo| topo.add_observer @path_manager }.start
  end
end
```

イベントハンドラでパスを消す

```
class PathManager < Trema::Controller
  # ...
  def delete_link(port_a, port_b, _topology)
    @graph.delete_link port_a, port_b
    Path.select { |each| each.link?(port_a, port_b) }.each(&:destroy)
  end
```

メッセージの振り分け


Topology
トポロジ関係
RoutingSwitch
PathManager
Path
OpenFlow
メッセージ
PacketIn
のみ

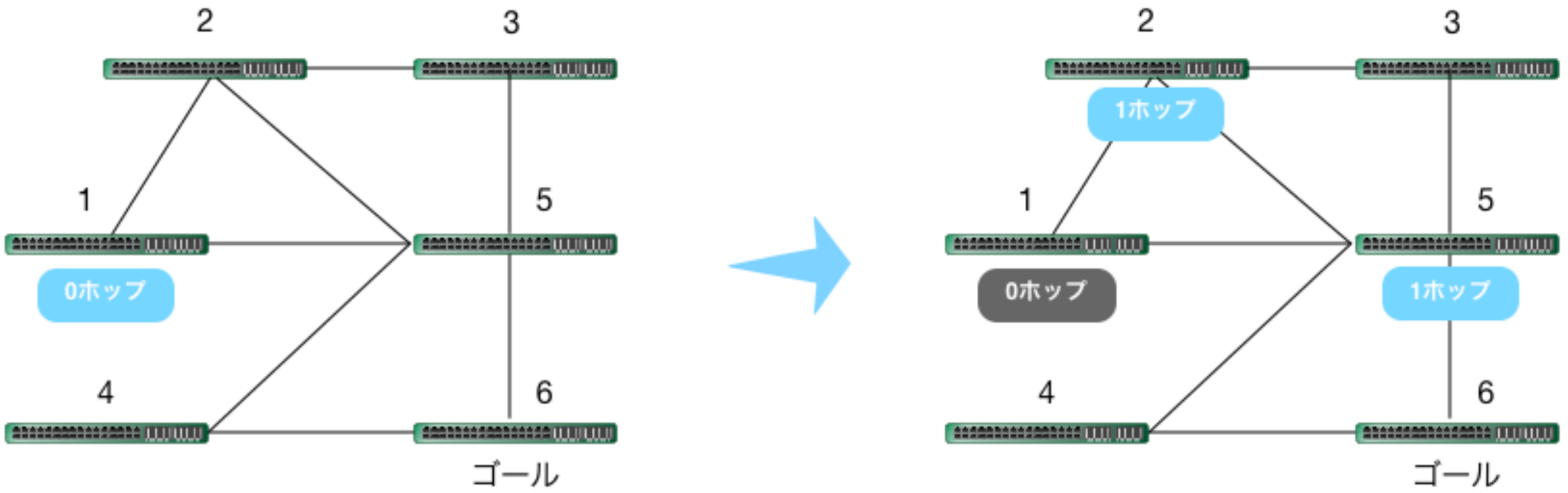
ハンドラの委譲

トポロジ関連メッセージ→Topologyへ

```
class RoutingSwitch < Trema::Controller
  delegate :switch_ready, to: :@topology
  delegate :features_reply, to: :@topology
  delegate :switch_disconnected, to: :@topology
  delegate :port_modify, to: :@topology
```

PacketIn→TopologyとPathManagerへ

```
def packet_in(dpid, packet_in)
  @topology.packet_in(dpid, packet_in)
  @path_manager.packet_in(dpid, packet_in) unless packet_in.lldp?
end
```


2
3
1
5
0ホップ
4
6
ゴール
2
3
1ホップ
1
5
0ホップ
1ホップ
4
6
ゴール
2
3
1ホップ
2ホップ
1
5
0ホップ
1ホップ
4
6
2ホップ
ゴール
2ホップ
2
3
1
5
4
6
ゴール
2ホップ

まとめ

- 小さいクラスを組み合わせよう
  - 部品 (トポロジ) の再利用
  - コードの見通しを良くする
- 必要なのはOOPの基本テク
  - カプセル化、委譲など