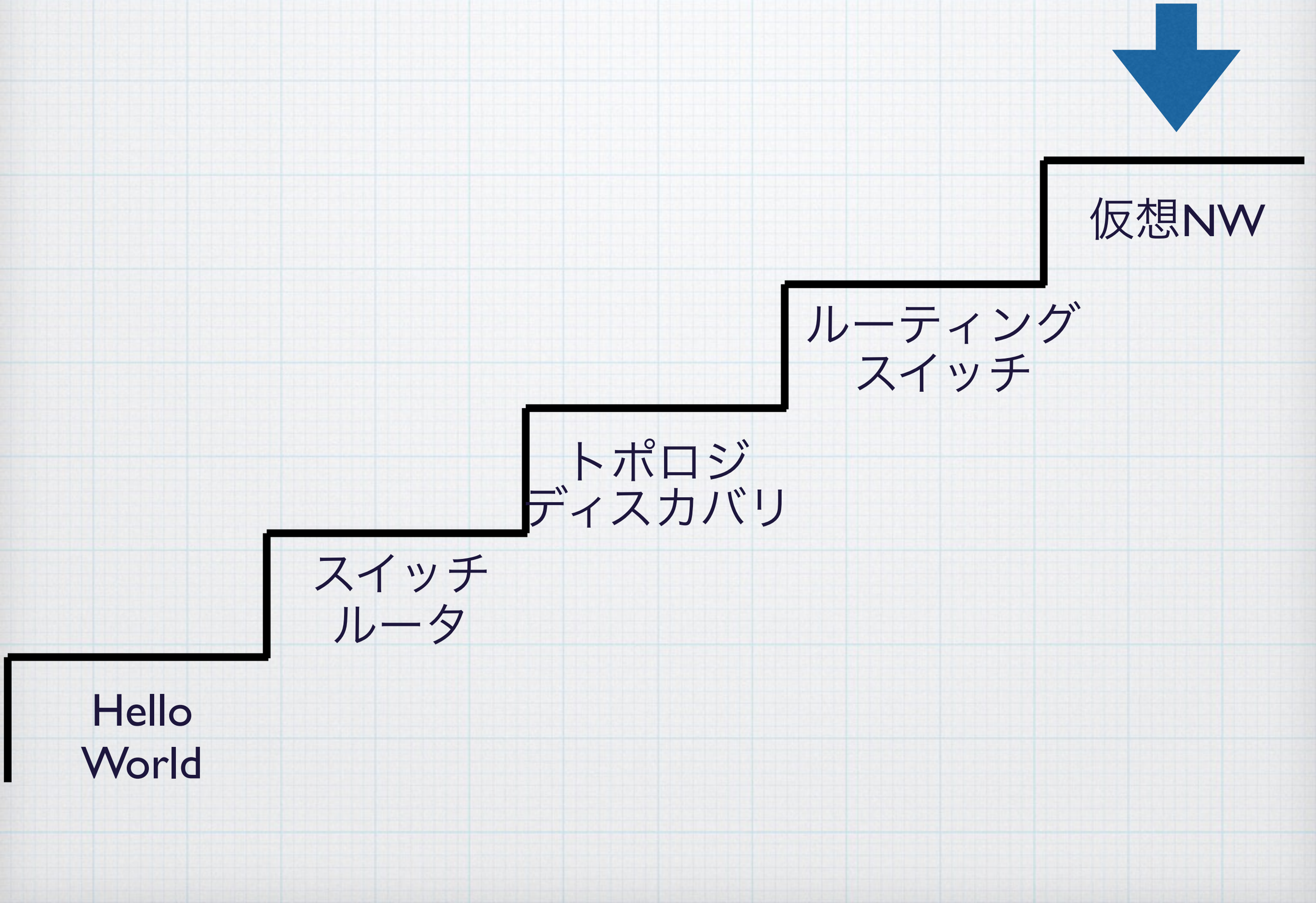
仮想ネットワーク
(スライス機能)

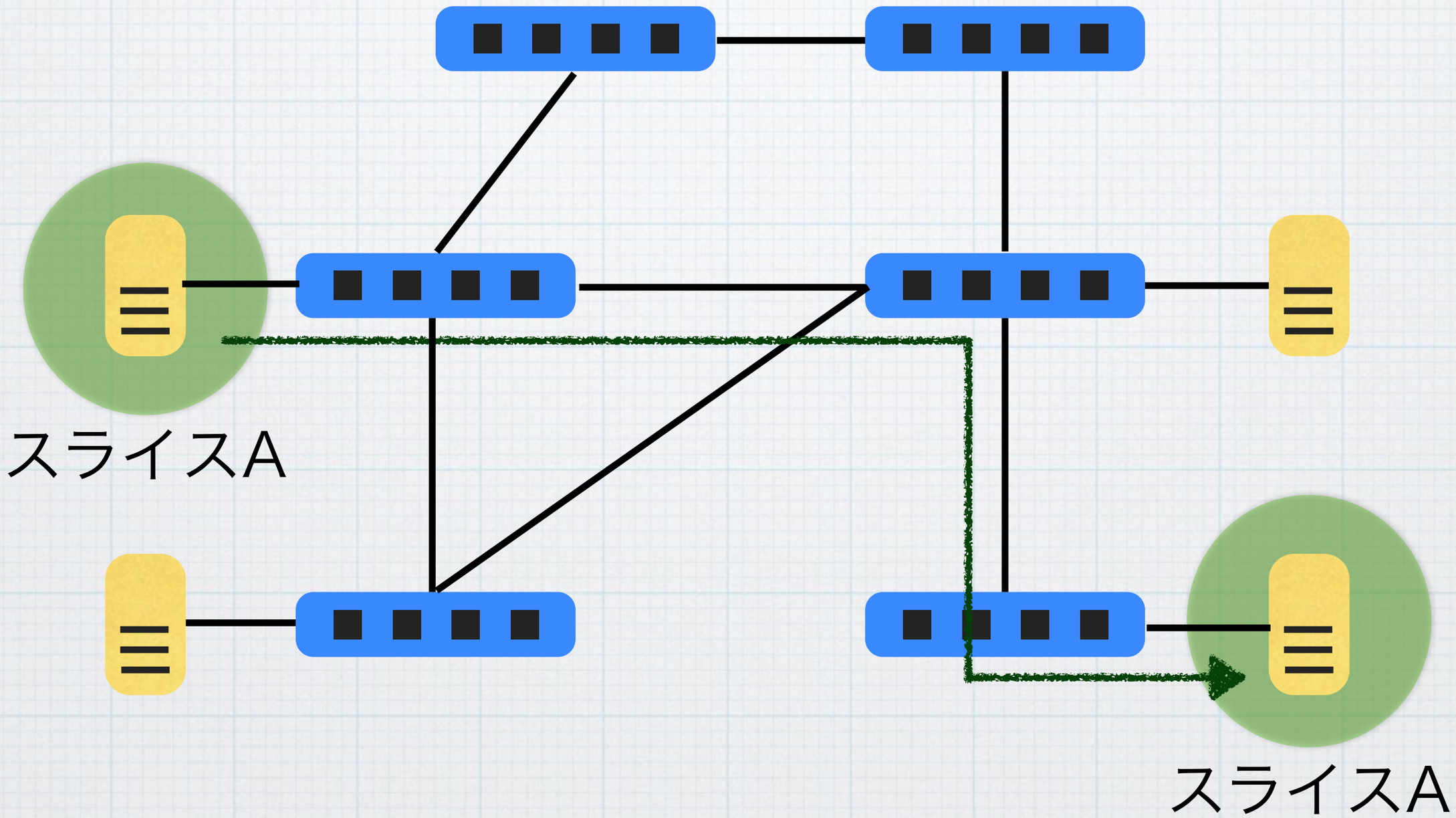
高宮 安仁 @yasuhito


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

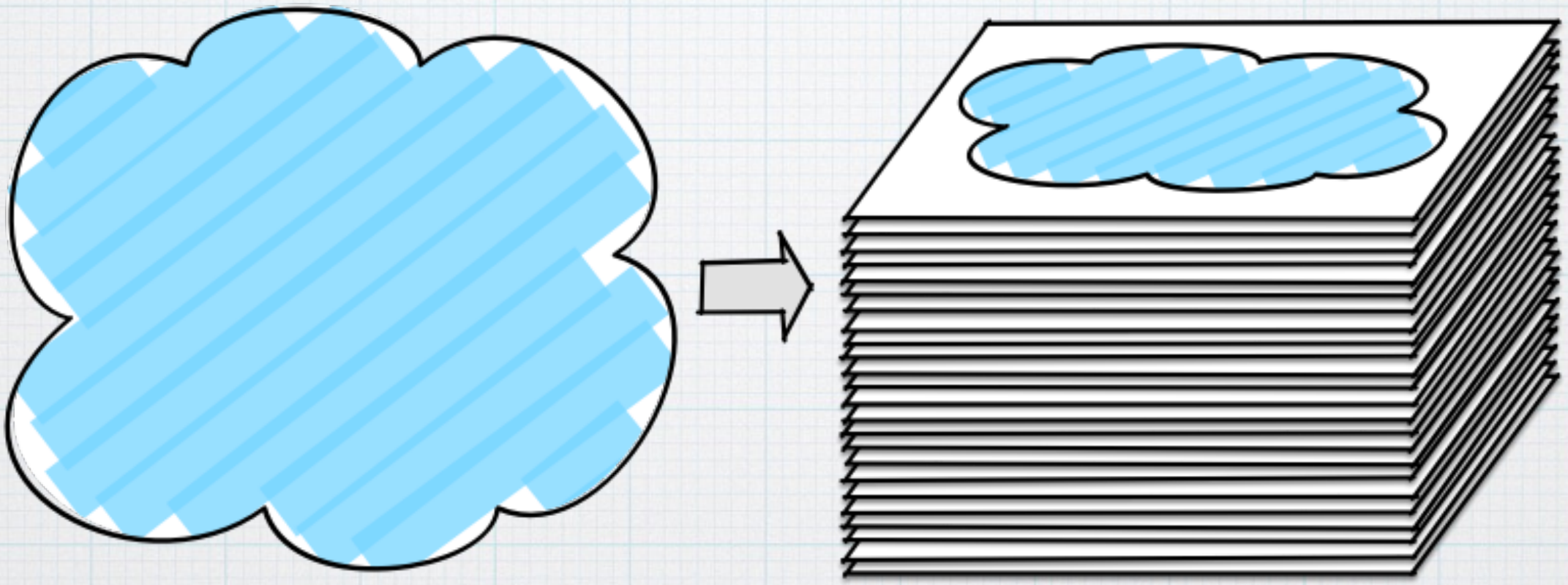
スライサブルスイッチ

機能

ネットワークスライス



「スライス」のイメージ



スライサブルスイッチ

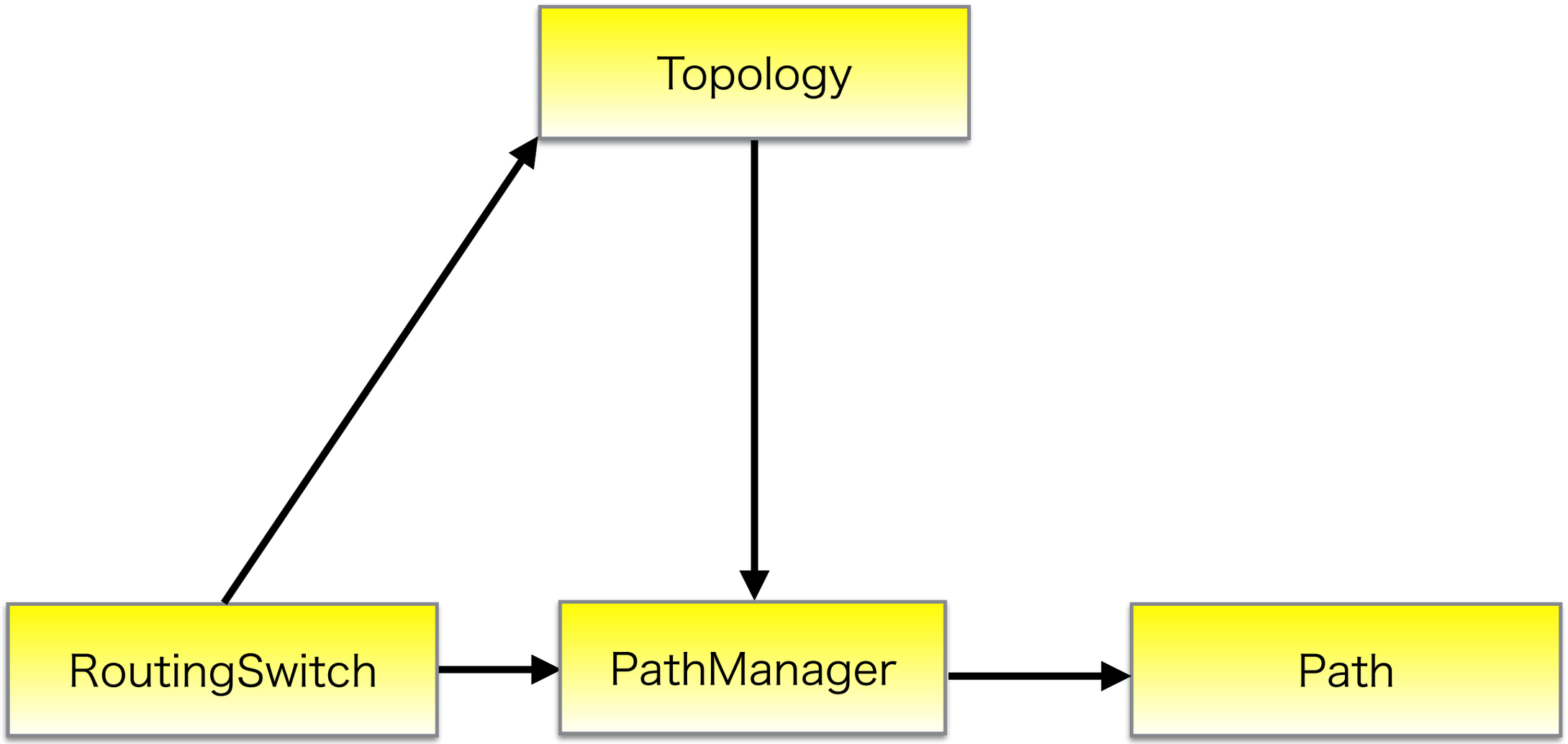
ルーティングスイッチ + スライス機能

- 基本はルーティングスイッチそのまま
- 拡張してスライス機能を追加

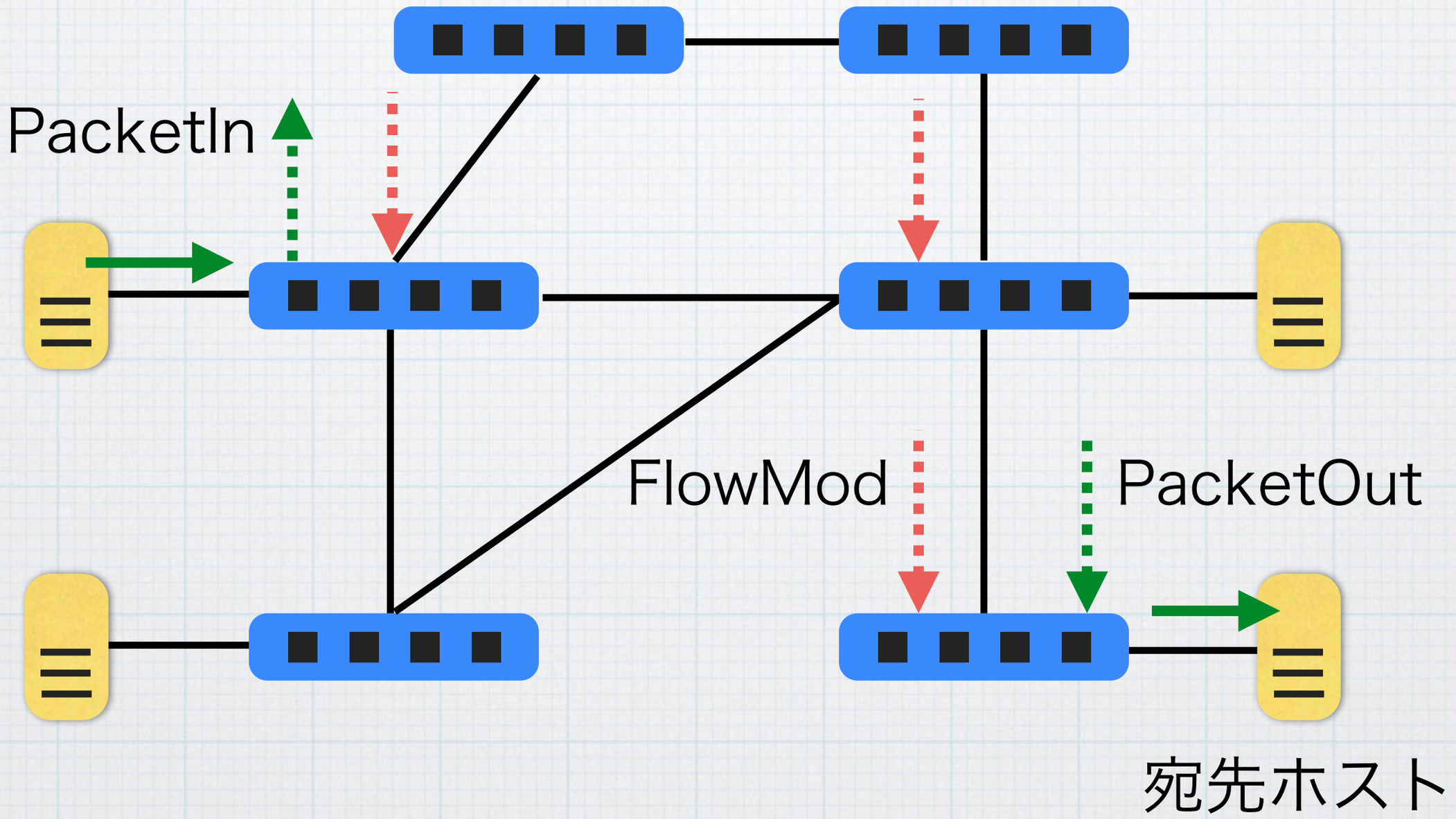
スライサブルスイッチ

ルーティングスイッチのおさらい

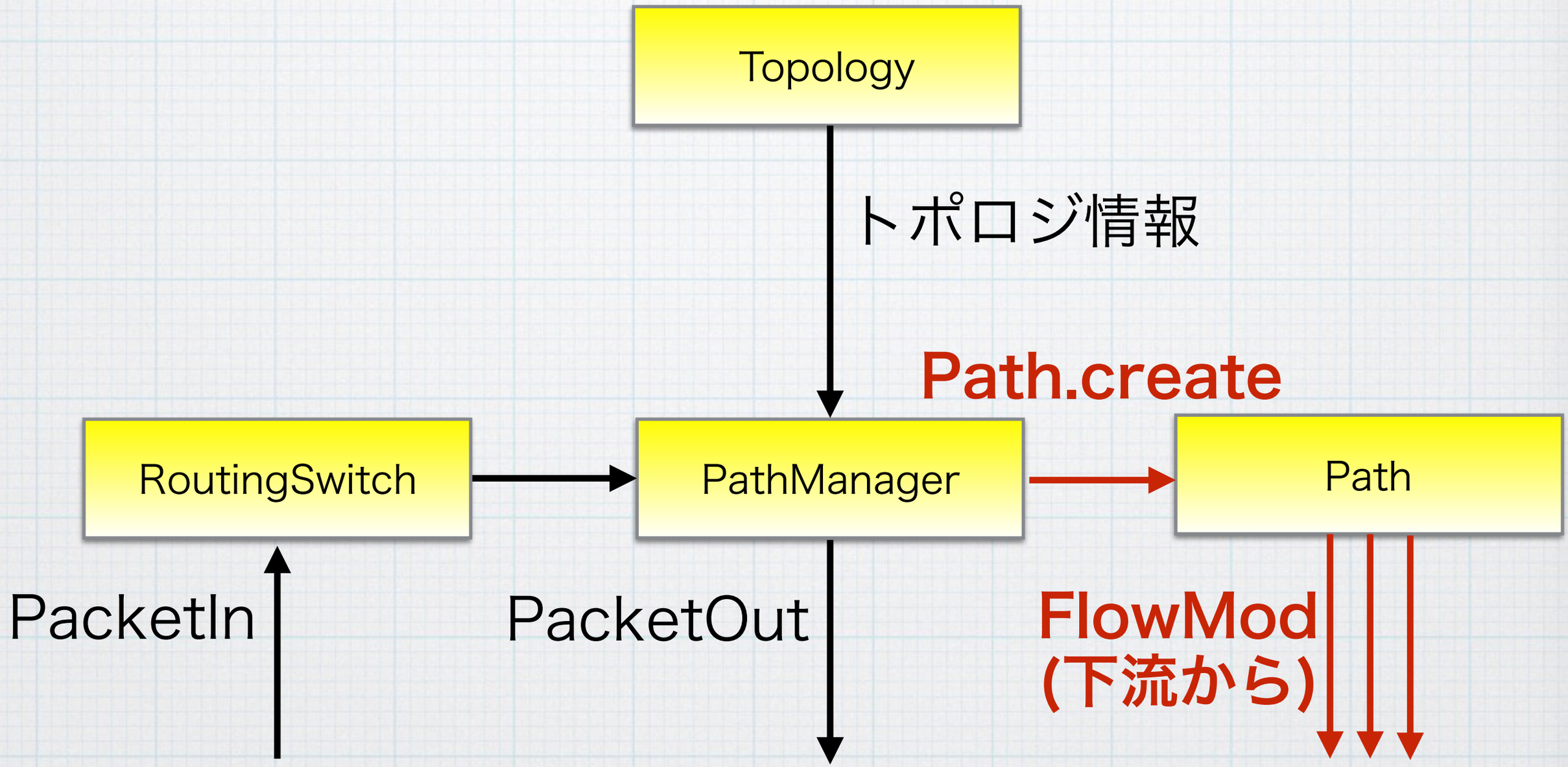
ルーティングスイッチのクラス構造



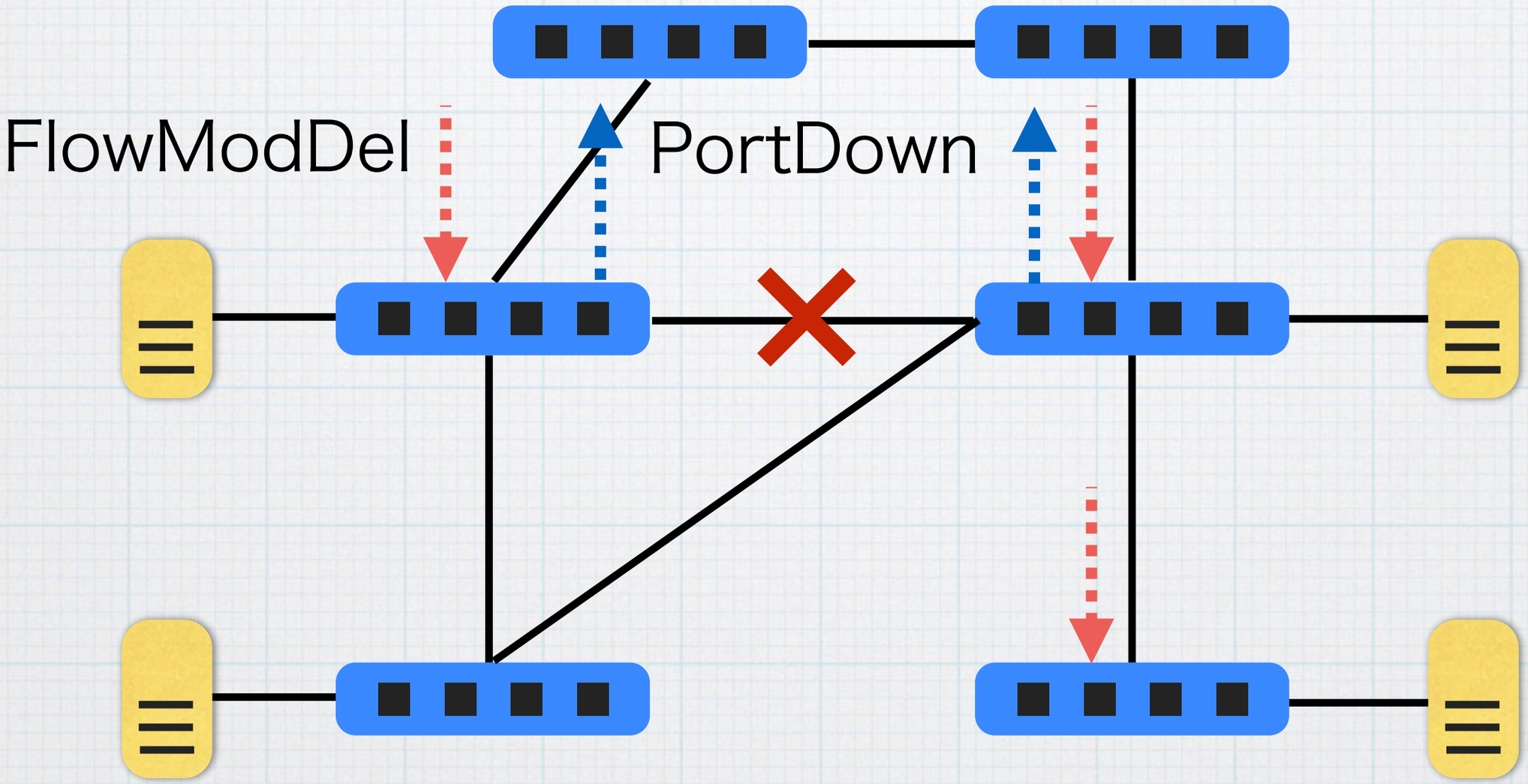
パケットを送信すると…



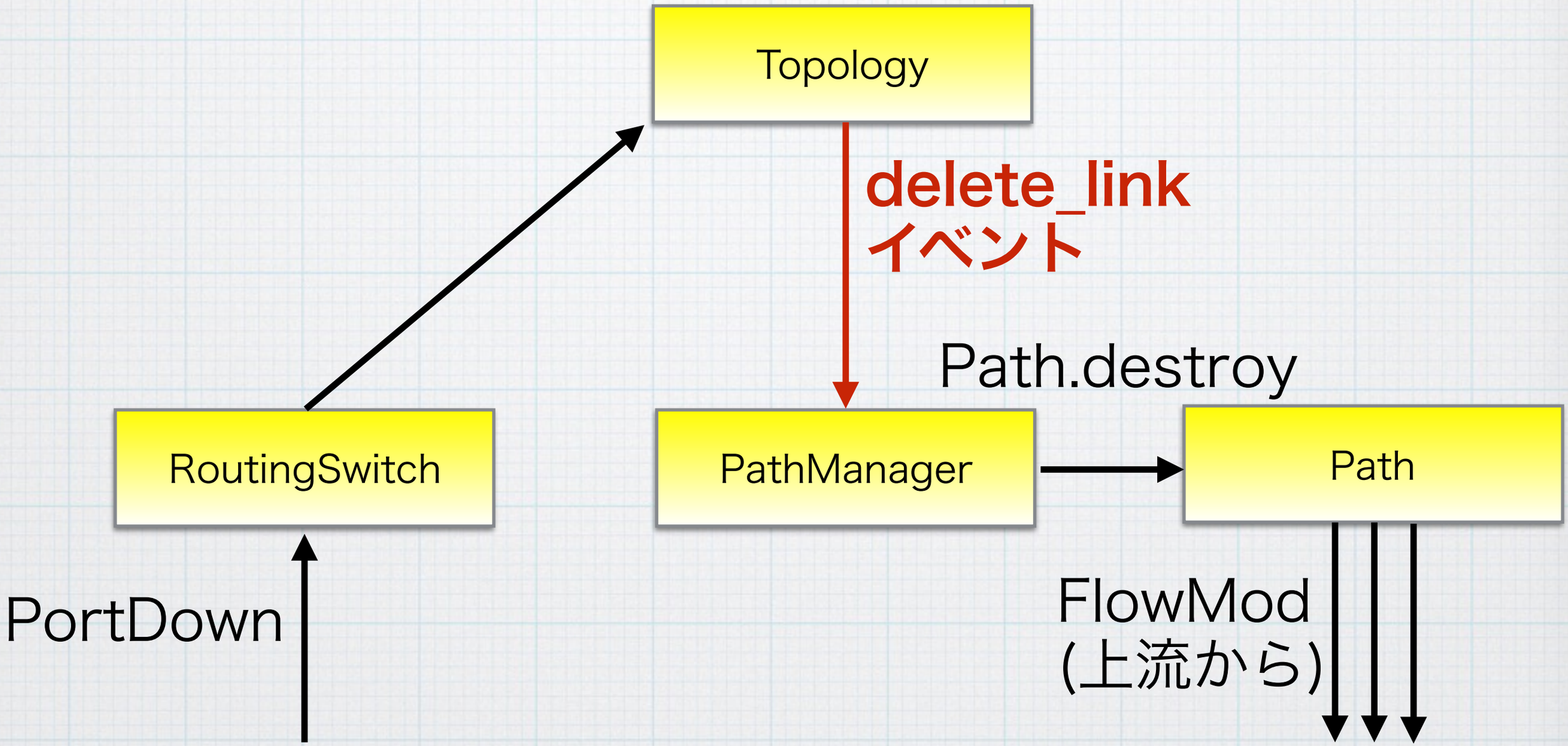
最短路パスを作る


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

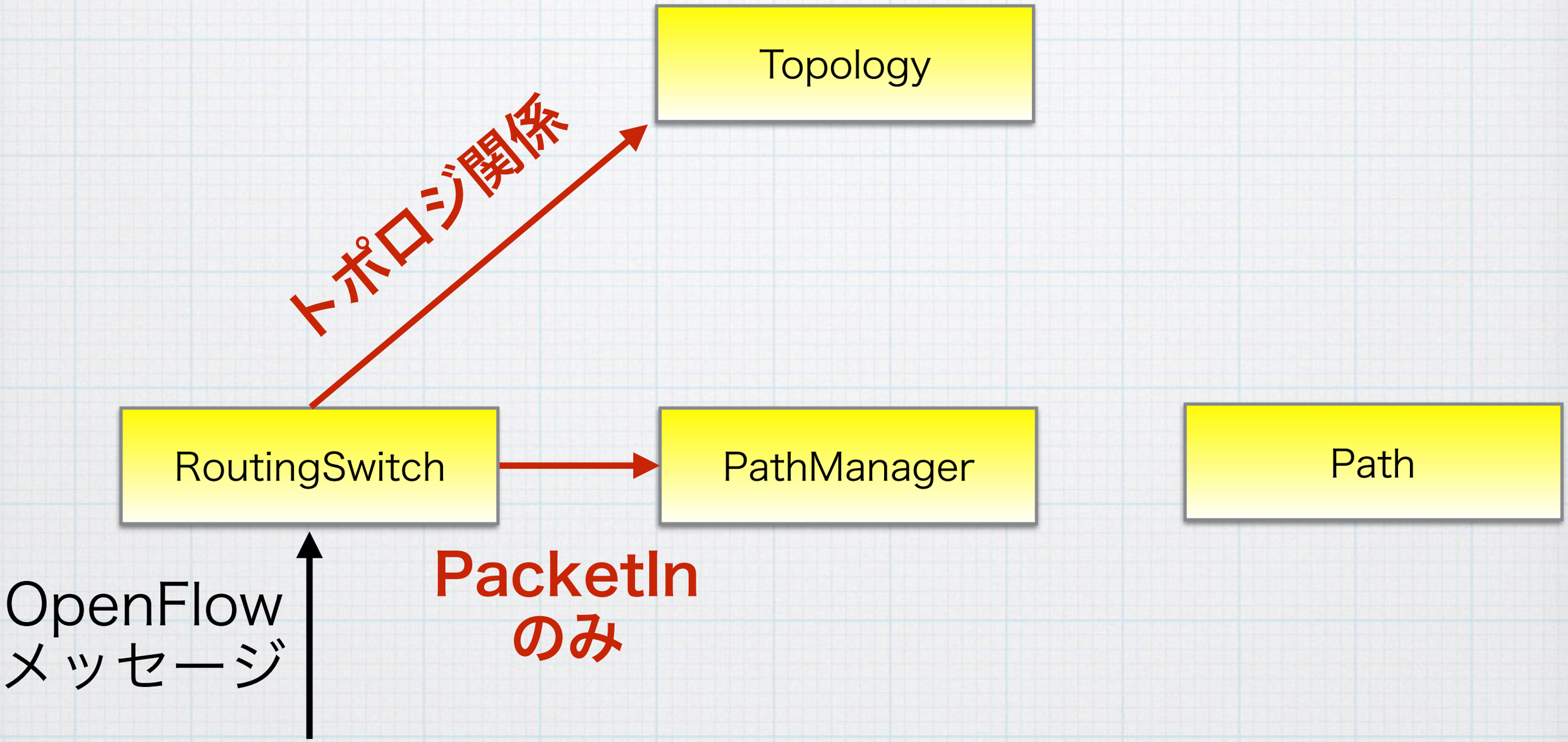
リンクが切れたら…



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)

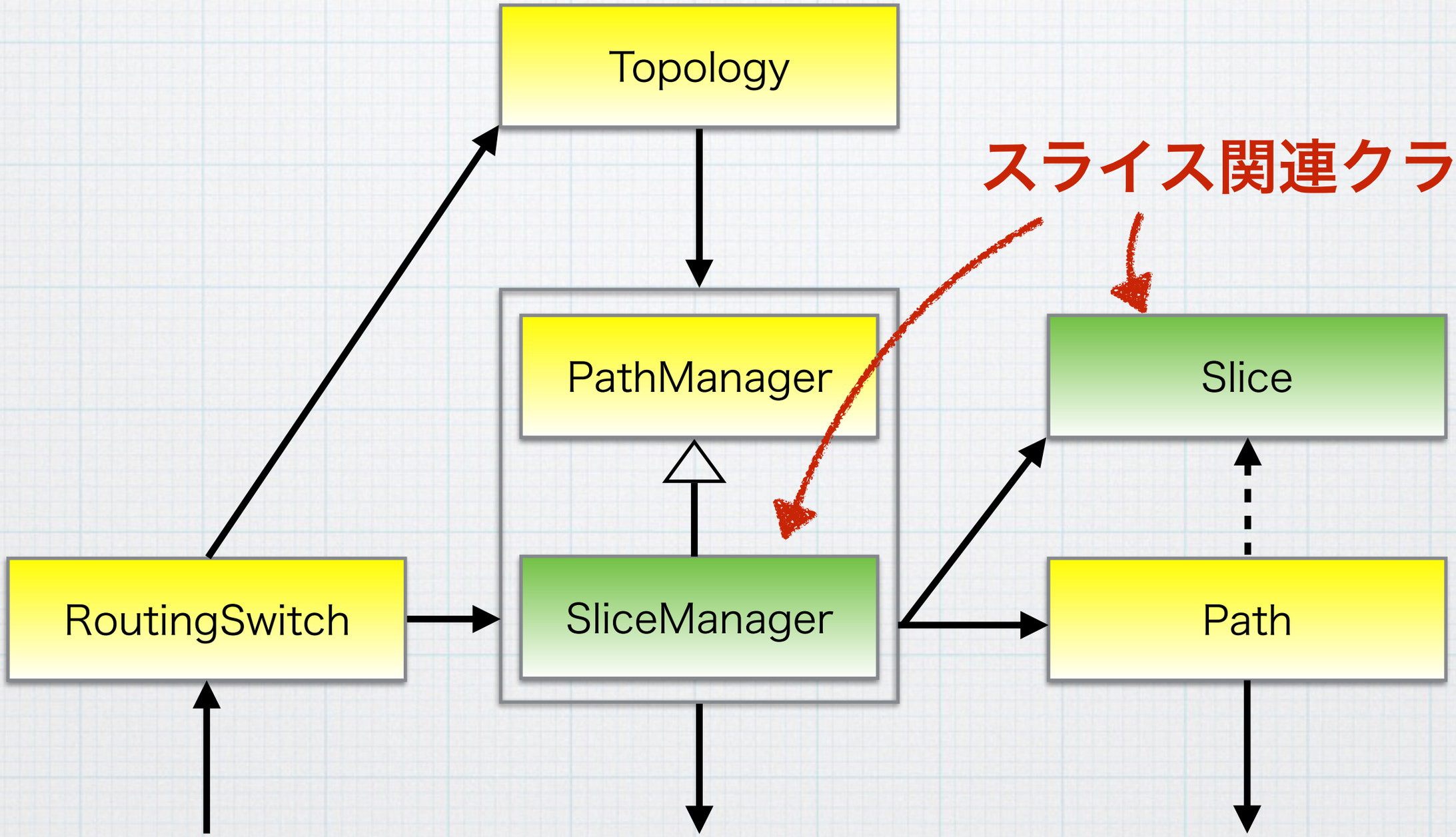
メッセージの振り分け


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

スライサブルスイッチ

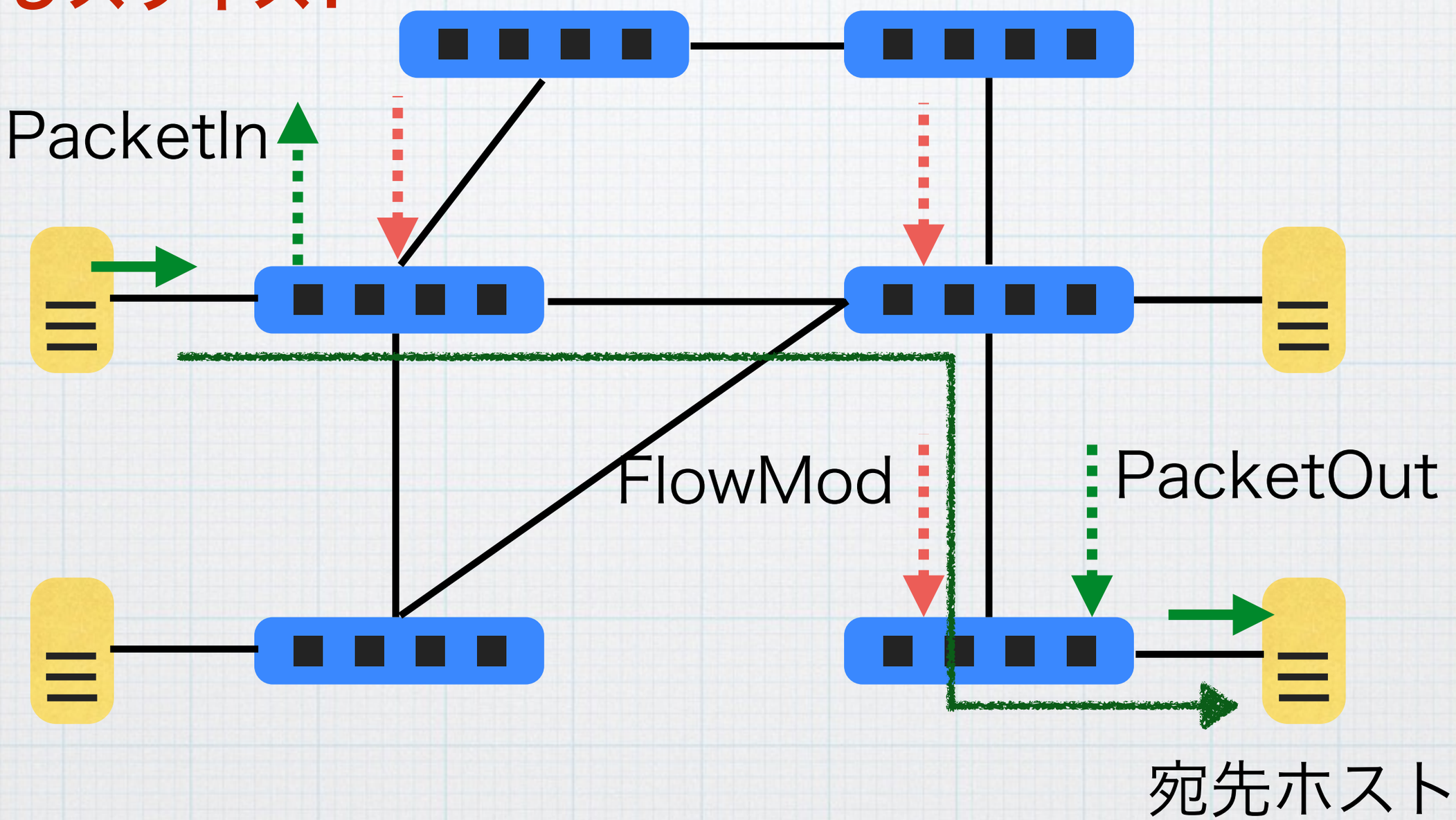
クラス構造

スライス機能の追加

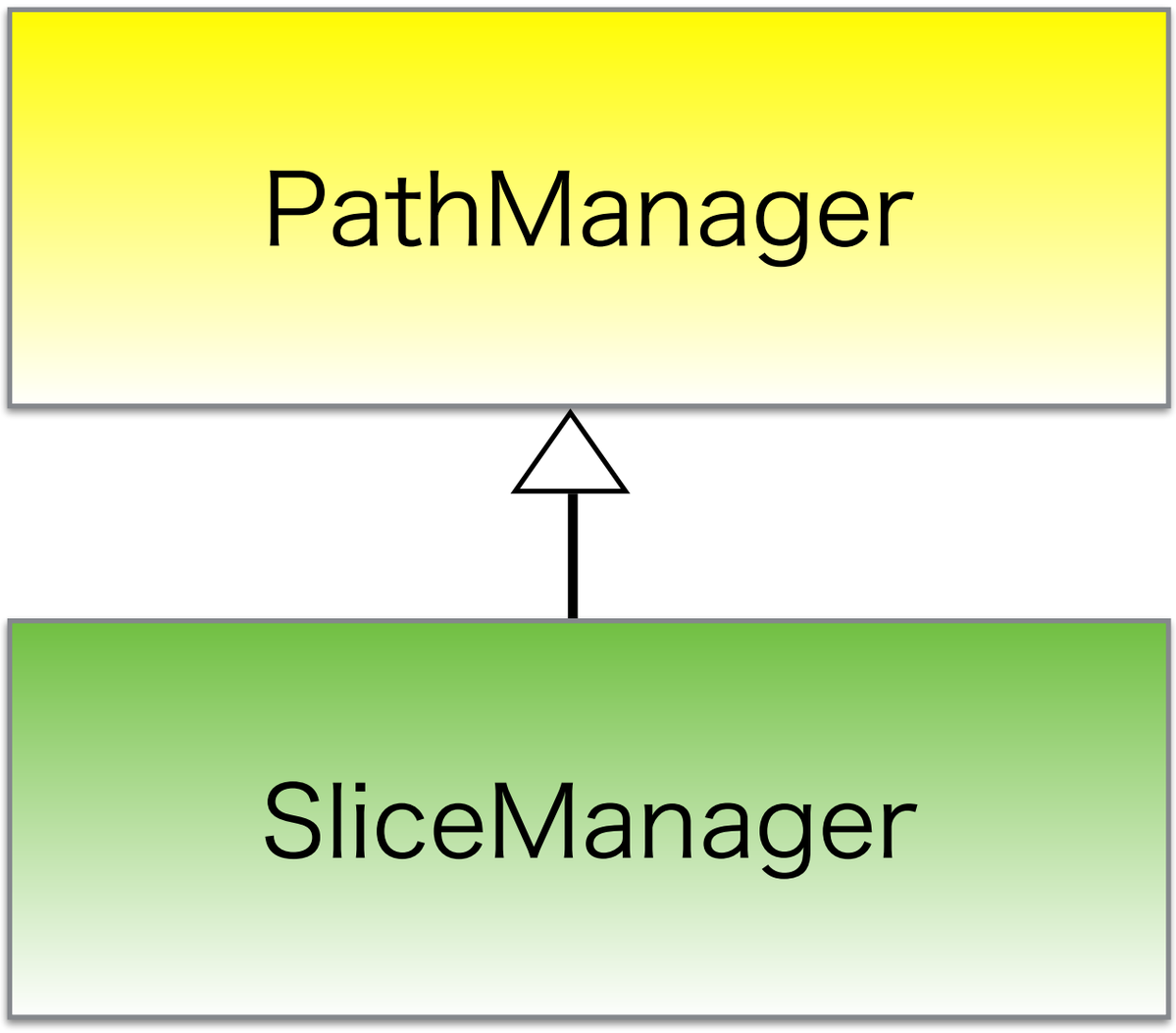

Topology
スライス関連クラス
PathManager
Slice
RoutingSwitch
SliceManager
Path

最短路パス

同じスライス?



継承とオーバーライド



- **Topologyハンドラ**
  トポロジグラフを作る
- **PacketInハンドラ**
  最短路パスを作る

- **Topologyハンドラ(継承)**
- **packet_in(上書き)**
  最短路パスを作る前に
  スライス判定を追加

同じスライスか?

```
def packet_in(_dpid, packet_in)
  slice = Slice.find do |each|
    each.member?(packet_in.slice_source) &&
    each.member?(packet_in.slice_destination(@graph))
  end
  ports = if slice
            path = maybe_create_shortest_path_in_slice(slice.name, packet_in)
            path ? [path.out_port] : []
          else
            external_ports(packet_in)
          end
  packet_out(packet_in.raw_data, ports)
end
```

src&dstが属するスライスを探す

- **スライスがあれば最短路で転送**
- **なければ外部ポートすべてに転送**

Slice クラスの API

```
def self.find(&block)
  all.find(&block)
end
```

条件に合うスライスを探す

```
def self.create(name)
  if find_by(name: name)
    fail SliceAlreadyExistsError, "Slice #{name} already exists"
  end
  new(name).tap { |slice| all << slice }
end
```

スライスを作る

Topology

PathManager

Slice

RoutingSwitch

SliceManager

Path

OpenFlow
メッセージ

PacketOut

FlowMod

カプセル化
・スライス追加・削除
・ホスト追加・削除

スライサブルスイッチ

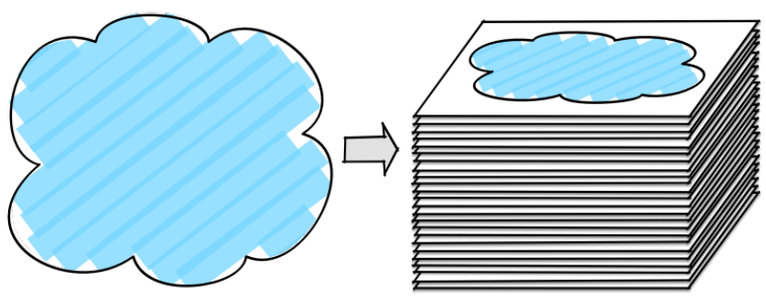
部品として使う

たとえば Mini IaaS

Webインタフェース

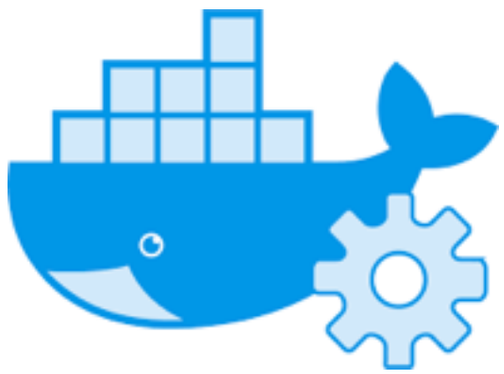
REST API

スライサブルスイッチ



REST API

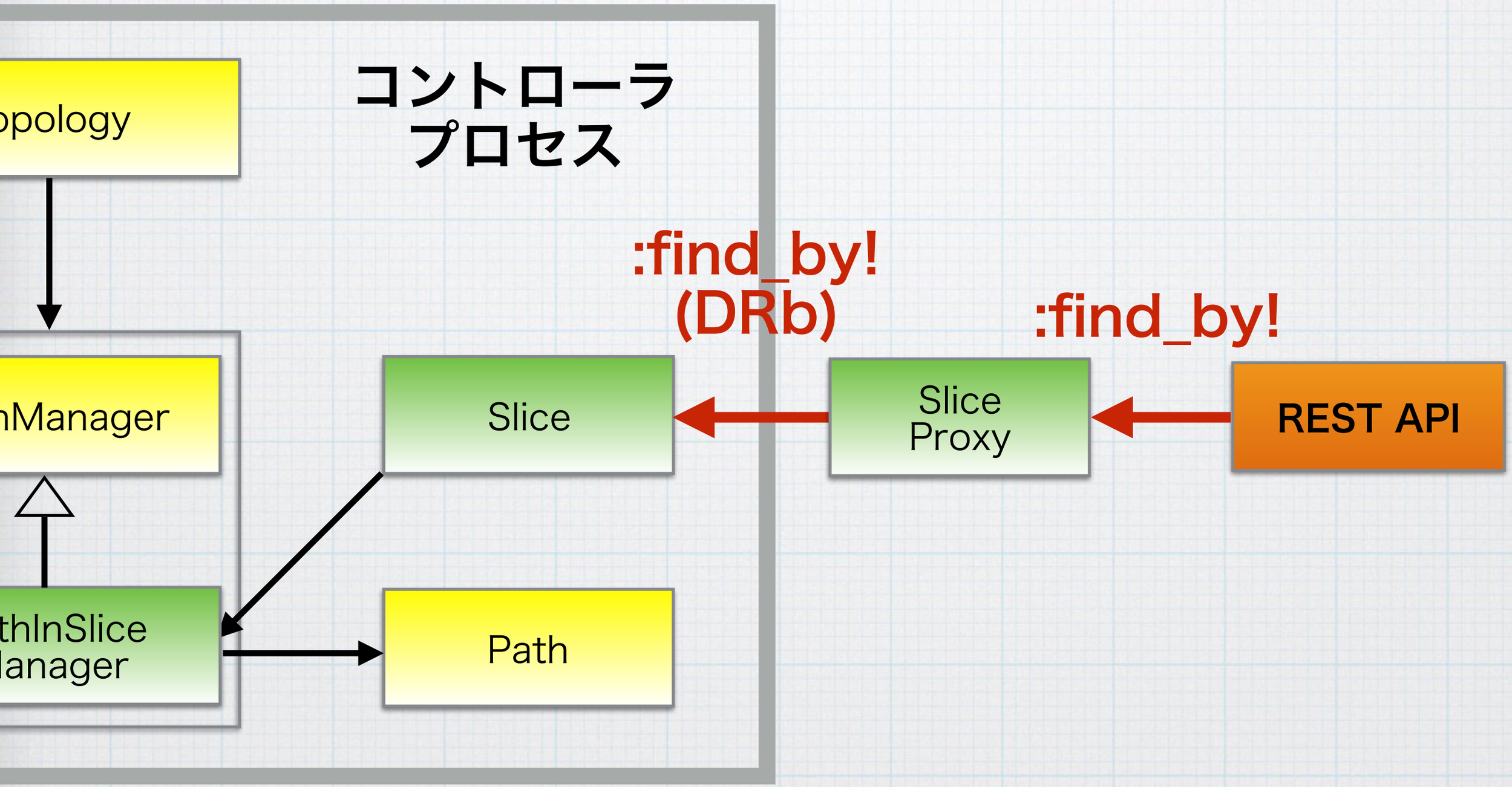
Docker



REST API

Description	Method	URI
Create a slice	POST	`/slices`
Delete a slice	DELETE	`/slices`
List slices	GET	`/slices`
Shows a slice	GET	`/slices/:slice_id`
Add a port to a slice	POST	`/slices/:slice_id/ports`
Delete a port from a slice	DELETE	`/slices/:slice_id/ports`
List ports	GET	`/slices/:slice_id/ports`
Shows a port	GET	`/slices/:slice_id/ports/:port_id`
Adds a host to a slice	POST	`/slices/:slice_id/ports/:port_id/mac_addresses`
Deletes a host from a slice	DELETE	`/slices/:slice_id/ports/:port_id/mac_addresses`
List MAC addresses	GET	`/slices/:slice_id/ports/:port_id/mac_addresses`
Shows a MAC address	GET	`/slices/:slice_id/ports/:port_id/mac_addresses/:mac_address`

REST API


コントローラ
プロセス
:find_by!
(DRb)
:find_by!
REST API
Slice
Slice
Proxy
Path
pology
nManager
thInSlice
lanager

リモートクラス呼び出し

REST API の実装

Sliceクラスのメソッドを呼ぶだけ!

```
get 'slices/:slice_id' do
  rest_api { Slice.find_by!(name: params[:slice_id]) }
end
```

Sliceクラスプロキシ

```
class Slice
  def self.find_by!(query)
    # ...
    remote_klass =
      Trema.trema_process('RoutingSwitch', socket_dir).controller.slice
    remote_klass.find_by!(query)
  end
end
```

スライサブルスイッチ

まとめと課題

まとめ

仮想ネットワークをOpenFlowで実装

- ルーティングスイッチをうまく拡張
  - スライス関連クラスの追加と継承
- REST API: IaaSの部品として使えるように

課題 スライス機能の拡張 (2週間)

スライスの分割・結合

・スライスの分割と結合機能を追加する

スライスの可視化

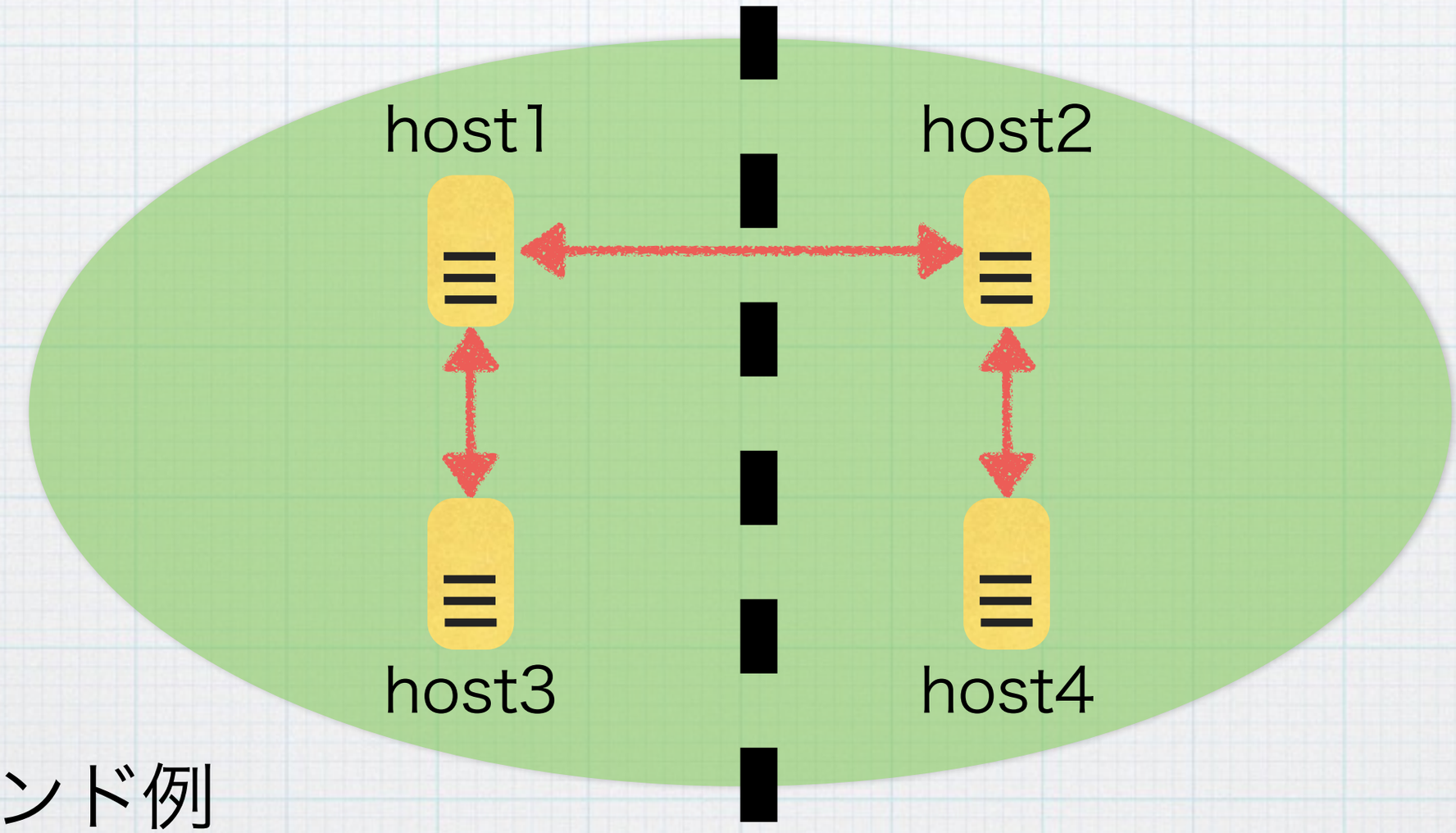
・ブラウザでスライスの状態を表示

REST APIの追加

・分割・統合のできるAPIを追加

以上を実機スイッチで動かしてみよう

スライス分割コマンド



コマンド例

```
$ slice split slice_a
    —into slice_b:host1,host3 slice_c:host2,host4
```