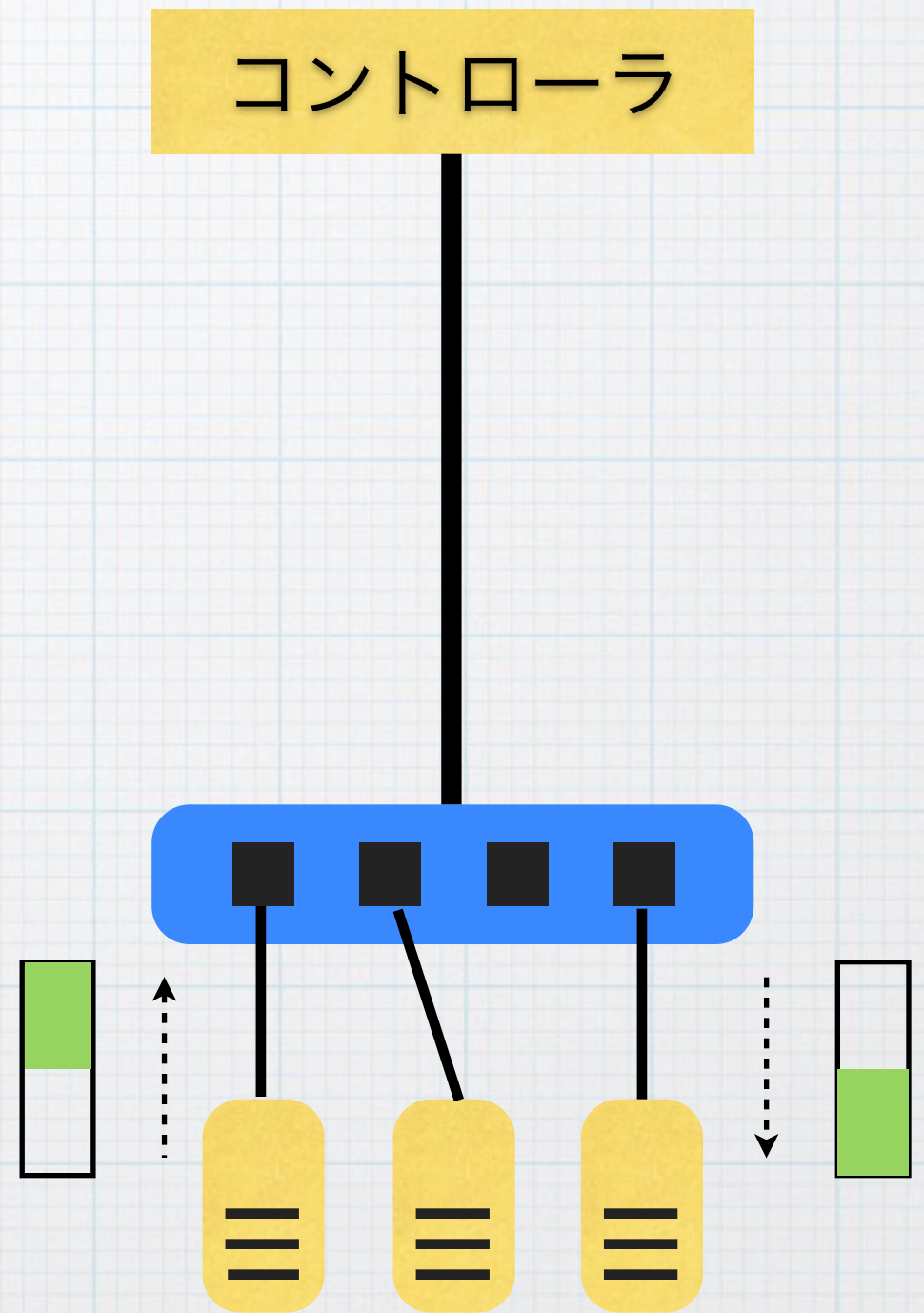


スイッチを作る前の 基礎知識



よくあるセッティングアップ

コントローラ

Flow Table

=

=

host1

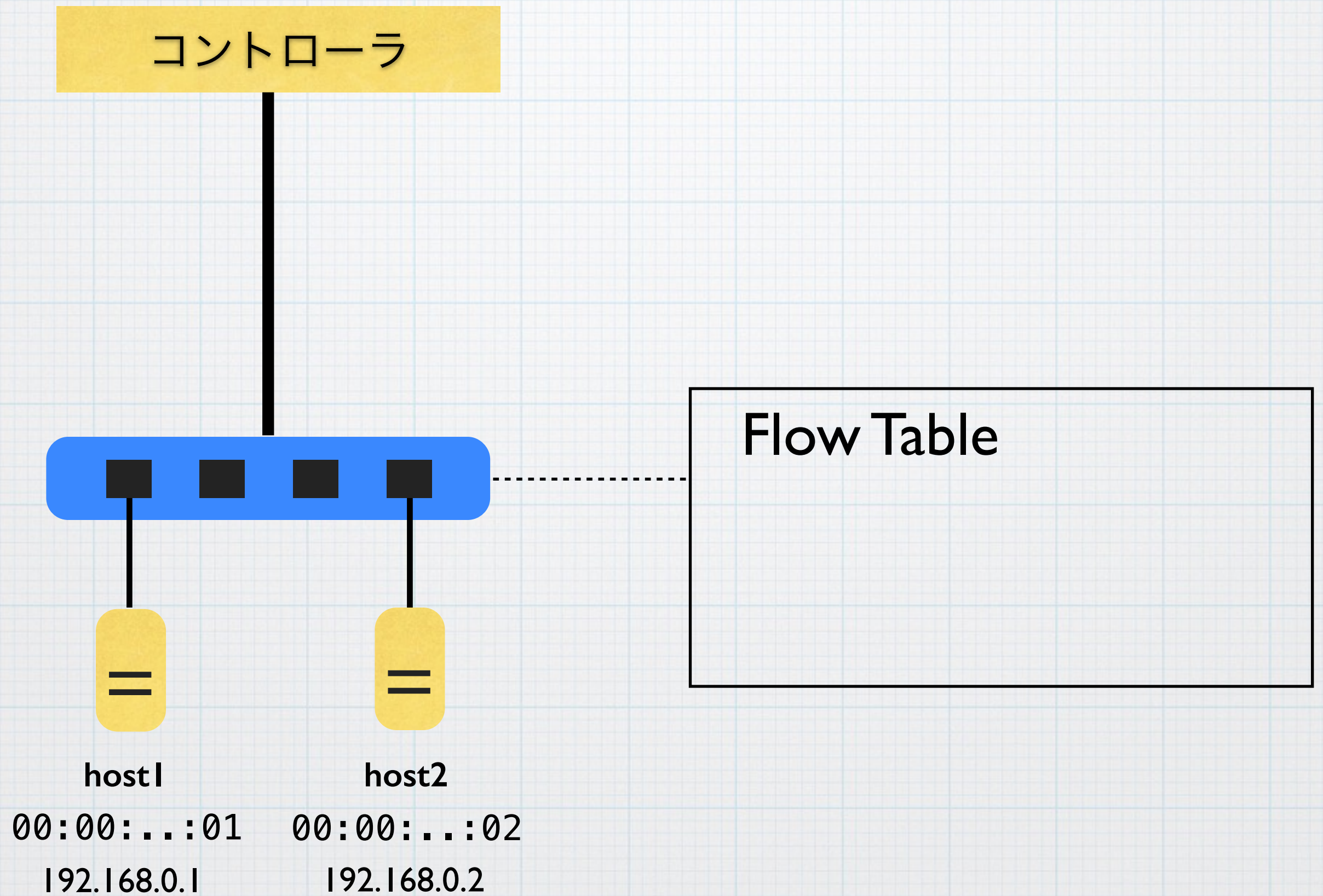
host2

00:00:::01

00:00:::02

192.168.0.1

192.168.0.2



Flow Table

パケットの処理ルールを管理する DB

- スイッチの HW として実装 (速い)
- 「宛先 MAC が 00:11:22:33:44:55 だったらポート 3 番に出す」など
 - マッチングルール: 「〇〇だったら」
 - アクション: 「xx する」

マッチングルール

- ポート 1 番から入ってきたら
- 送信元の MAC アドレスが
02:27:e4:fd:a3:5d だったら
- 宛先の IP アドレスが 192.168.0.30 で
内容が HTTP だったら

```
send_flow_mod_add(  
    datapath_id,  
    match: Match.new(in_port: 1)  
    # ...
```

- ・ポート1番から入ってきたら

```
send_flow_mod_add(  
    datapath_id,  
    match: Match.new(  
        ip_destination_address: '192.168.0.30',  
        transport_destination_port: 80  
    )  
    # ...
```

- 宛先が 192.168.0.30 で
HTTP だったら

指定できる条件

- Ingress port
- Ether src
- Ether dst
- Ether type
- IP src
- IP dst
- IP proto
- IP ToS bits
- TCP/UCP src port
- TCP/UDP dst port
- VLAN id
- VLAN priority


```
send_flow_mod_add(  
    datapath_id,  
    match: Match.new(in_port: 1),  
    actions: SendOutPort.new(4),  
    # ...
```

- ・ ポート1番から入ってきたら
ポート4番に出す

アクション

転送

- `SendOutPort.new`(ポート番号)

書き換え (Match と同じ 12 種類)

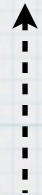
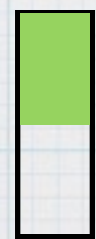
- `SetEtherDestinationAddress.new`(新しい宛先MACアドレス)
- `SetIpDestinationAddress.new`(新しい宛先IPアドレス)
- Etc.

```
send_flow_mod_add(  
    datapath_id,  
    match: Match.new(in_port: 1),  
    actions: [SetEtherDestinationAddress.new(...),  
              SetIpDestinationAddress.new(...),  
              SendOutPort.new(4)]  
  
    # ...
```

- 複数アクションが指定可能
- 書き換え2回→転送

コントローラ

Flow Table



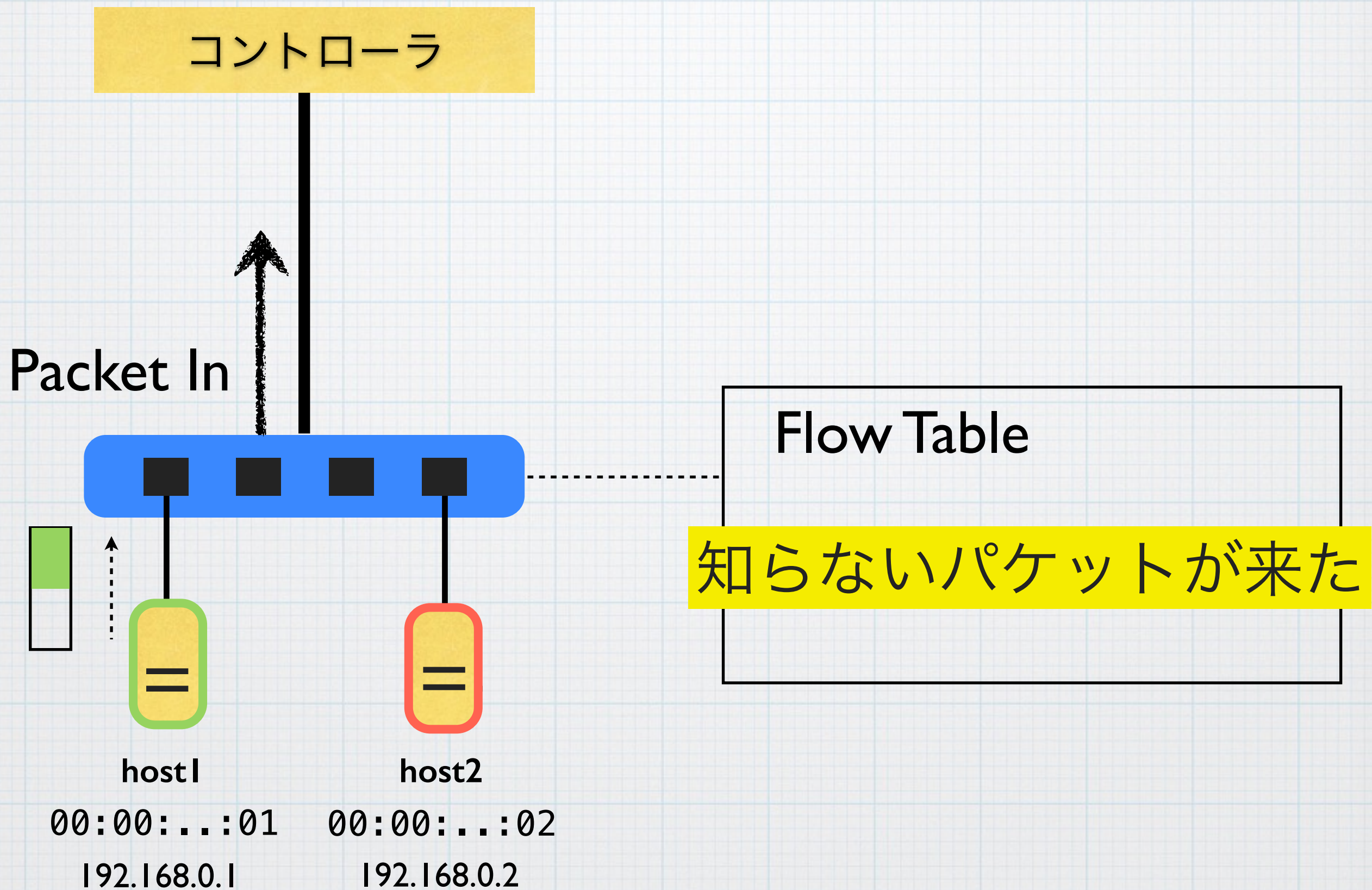
host1

00:00:...:01
192.168.0.1



host2

00:00:...:02
192.168.0.2



Packet In

- フローテーブルで処理できない
パケットを知らせるメッセージ*
(PacketIn オブジェクト)
- “コントローラに指示をあおぐ” イメージ
- Packet In が発生すると遅くなる

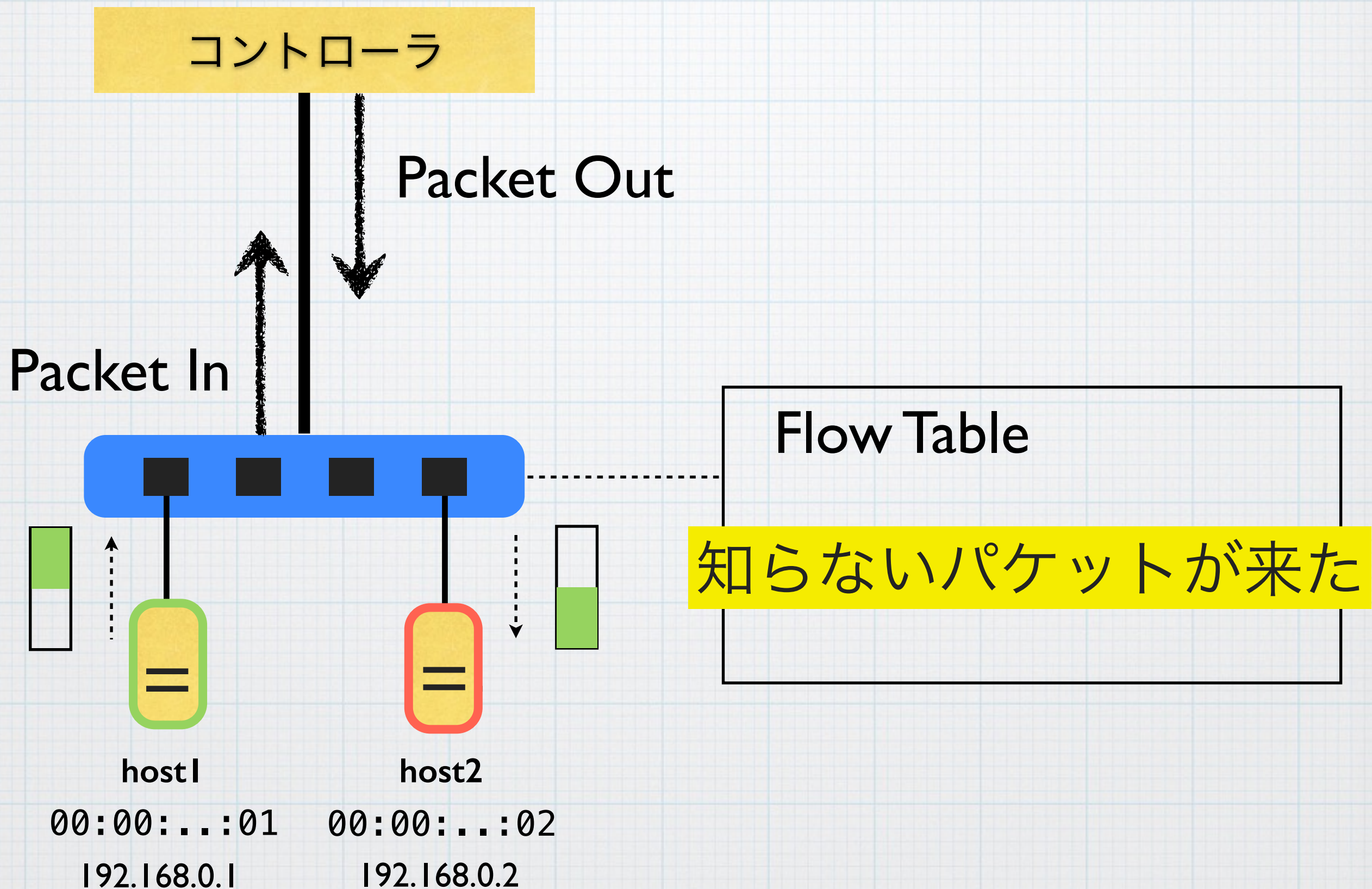
* OpenFlow1.3では動作が異なります

```
def packet_in(dpid, message)
  # messageの中身を見て、
  # ここで何か処理をする
end
```

- ・ ハンドラで PacketIn を拾う
- ・ message = PacketIn オブジェクト

```
% trema send_packets \  
    --source host1 --dest host2
```

- ・ テストパケットの送信
- ・ host1 から host2 へ送る



Packet Out

- (Packet In したパケットを)
指定したポートから出す
(`send_packet_out()`)
- ポートには物理ポートの他、
フラッディングなども指定可

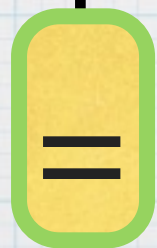
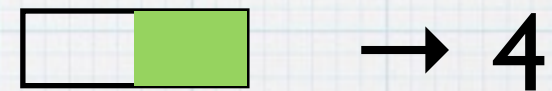

```
send_packet_out(  
    datapath_id,  
    in_port: message.in_port,  
    raw_data: message.raw_data,  
    actions: SendOutPort.new(:flood)  
)
```

- PacketIn と同じパケットを、すべてのポートから出す

コントローラ

Flow Mod

Flow Table



host1

00:00:...:01
192.168.0.1

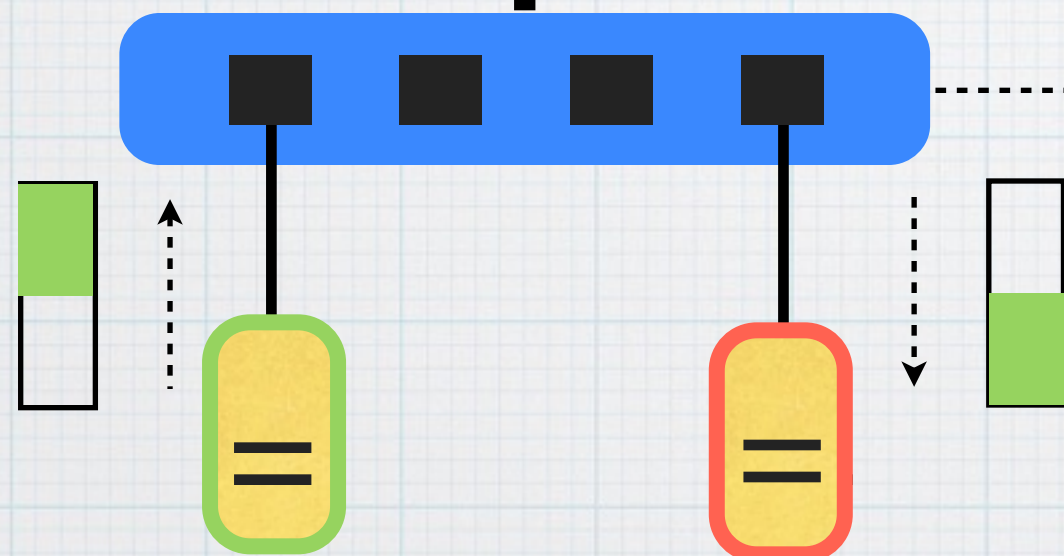


host2

00:00:...:02
192.168.0.2

コントローラ

Packet In
しない



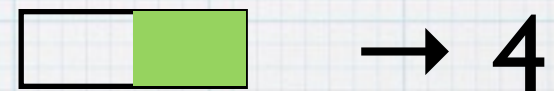
host1

00:00:...:01
192.168.0.1

host2

00:00:...:02
192.168.0.2

Flow Table



Flow Mod

- フローテーブルを変更する
 - フローエントリの追加・変更・削除
- `send_flow_mod_add()` など

```
send_flow_mod_add(  
    datapath_id,  
    match: ExactMatch.new(message),  
    actions: SendOutPort.new(4)  
)
```

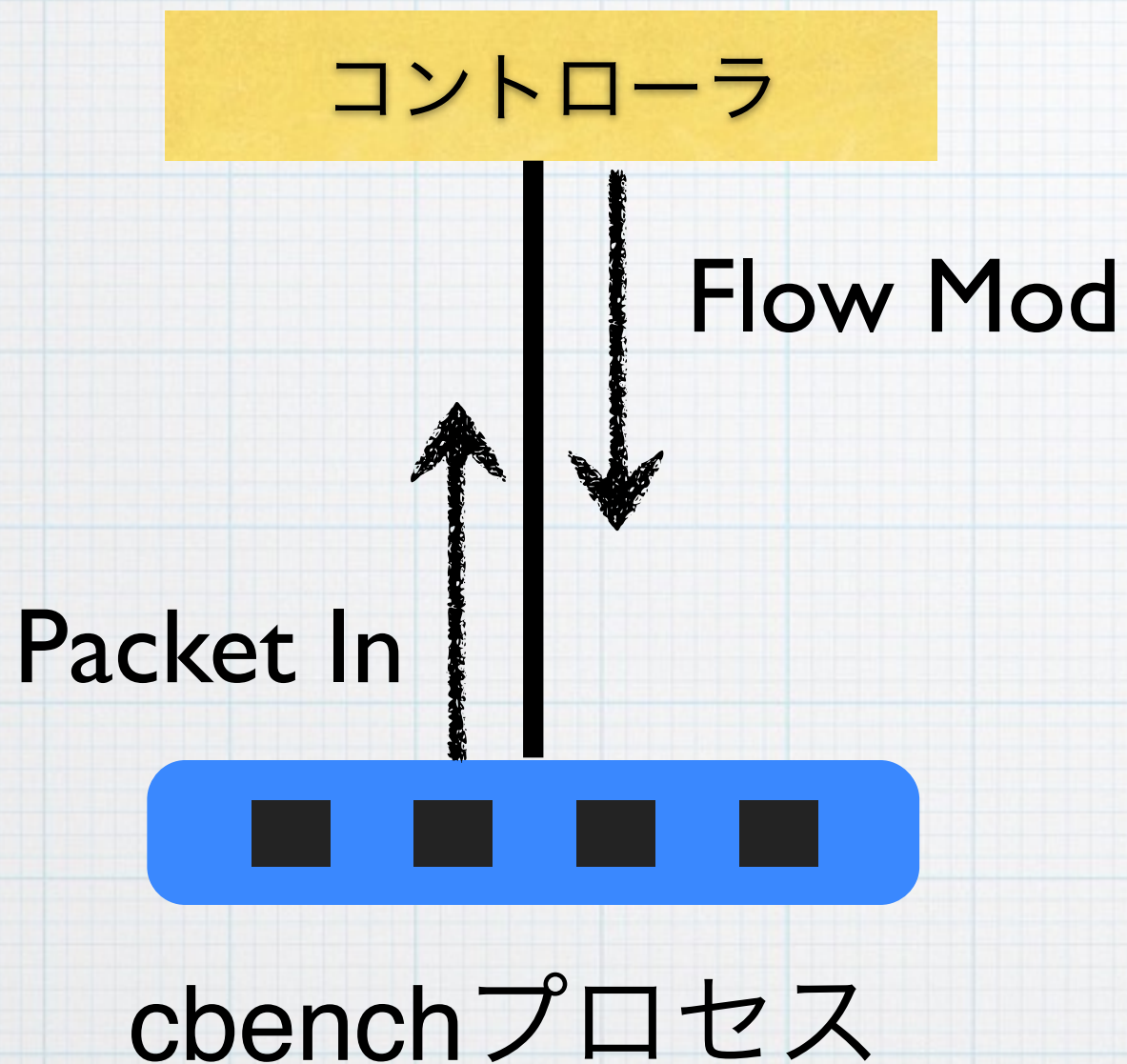
- packet_in と同じパケットは、
今後ポート4番に出す


```
% trema dump_flows 0xabc
```

- ・ フローのダンプ
- ・ 0xabcのフローテーブルを表示

課題

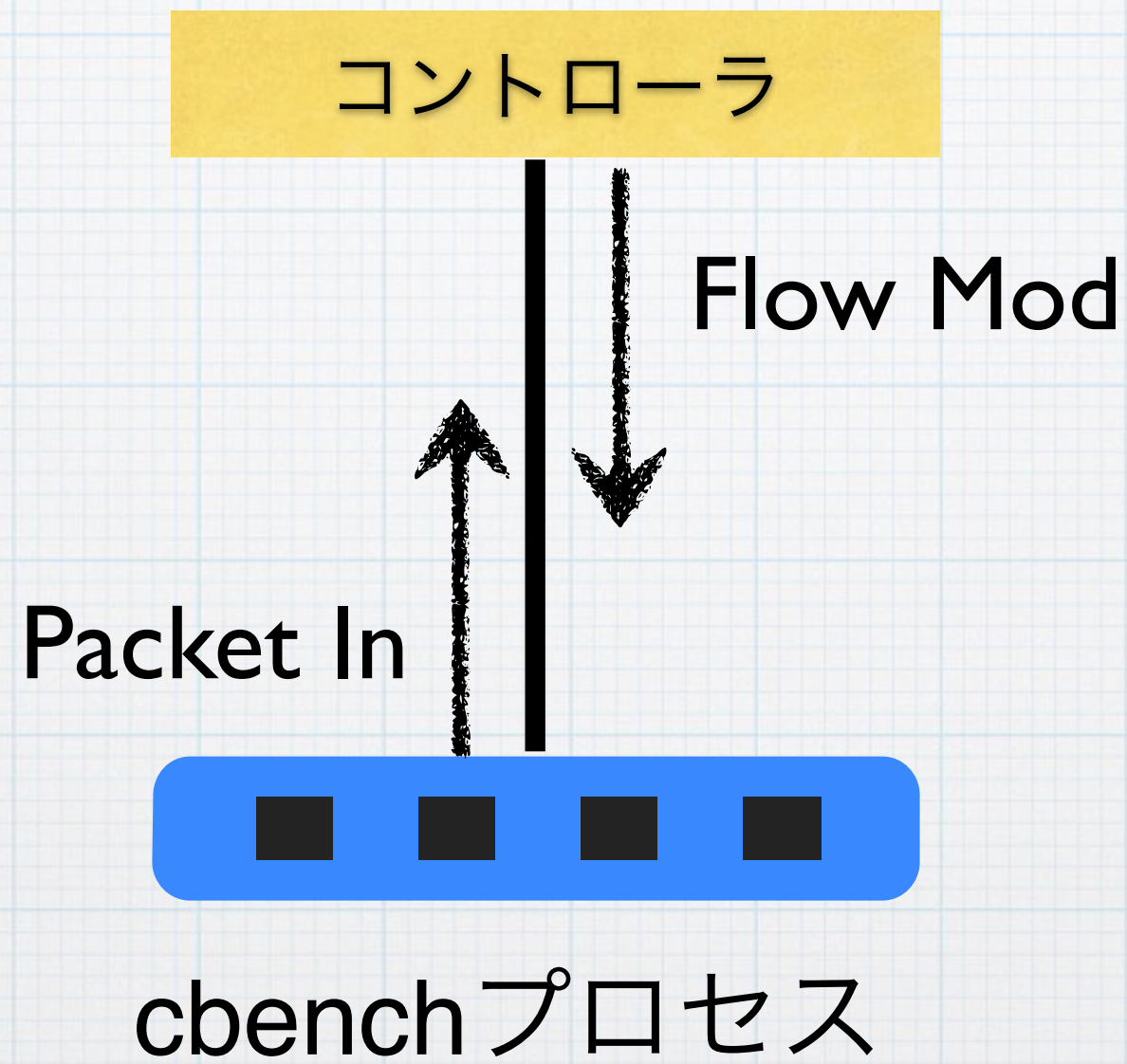
cbenchを
動かしてみよう



cbenchとは

OpenFlow用のマイクロベンチマーク

- cbenchプロセスがスイッチのふりをしてTremaに接続し、Packet Inを送りまくる
- Tremaは決められたFlow Modを返す
- 一定時間内にたくさん返したほうが高スコア



```
send_flow_mod_add(  
    datapath_id,  
    match: ExactMatch.new(message),  
    buffer_id: message.buffer_id,  
    actions:  
    SendOutPort.new(message.in_port + 1)  
)
```

- cbenchに送るFlowMod

課題 cbenchを実行

- <https://github.com/handai-trema/syllabus> で、
 - 課題 → cbench をクリック
 - できたりポジトリを `git clone`
 - `bundle install -binstubs`
 - README の通りに実行

レポート課題

- cbenchを高速化してみよう
- プロファイラでボトルネックを探す
 - “Ruby プロファイラ” で検索
- (発展課題)ボトルネックを改善
 - cbench.rbやTremaを改造

まとめ

- OpenFlowの基本

Packet In/Packet Out/Flow Mod

- テストパケットの送りかた
- フローテーブルのダンプ