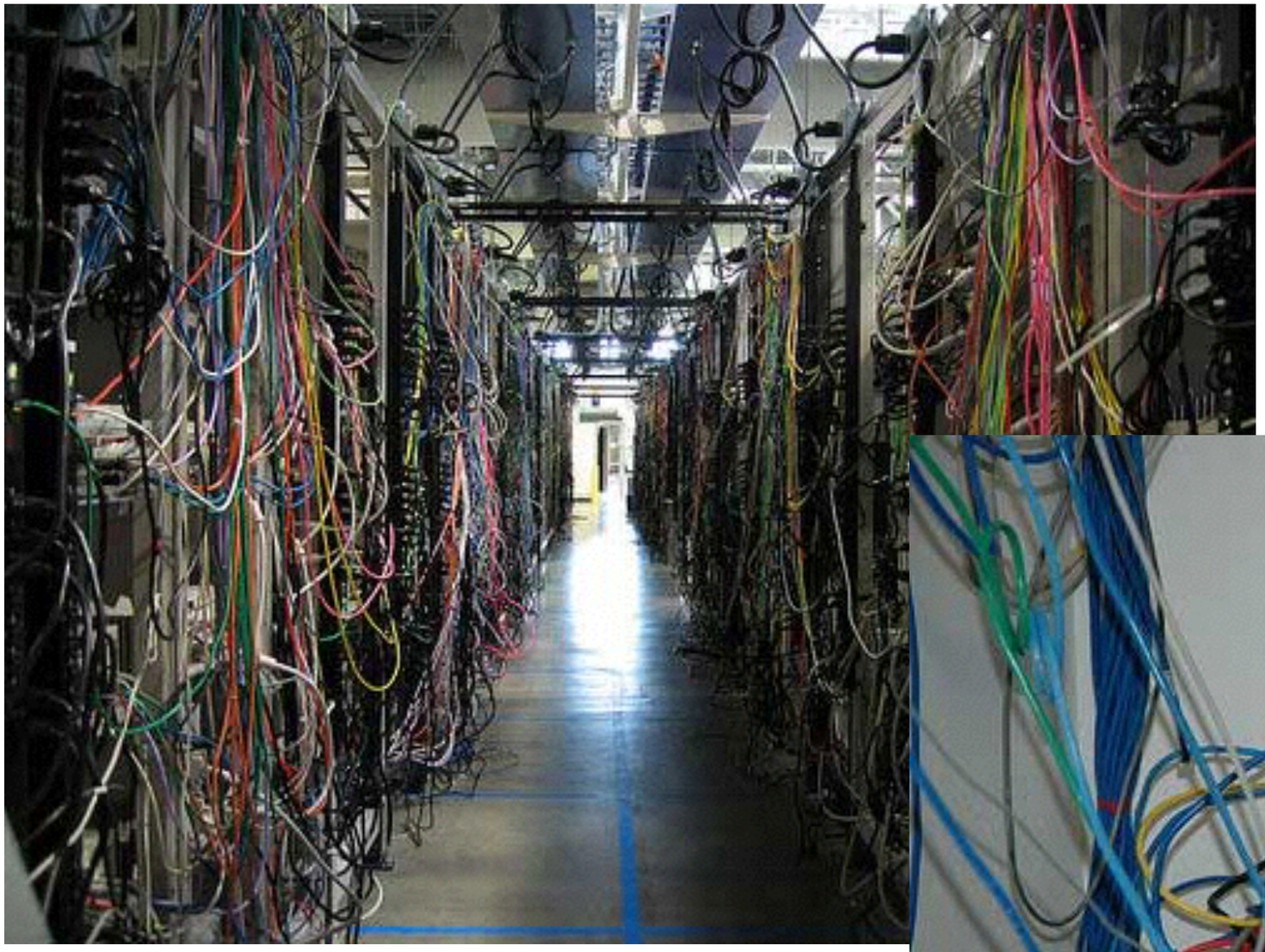
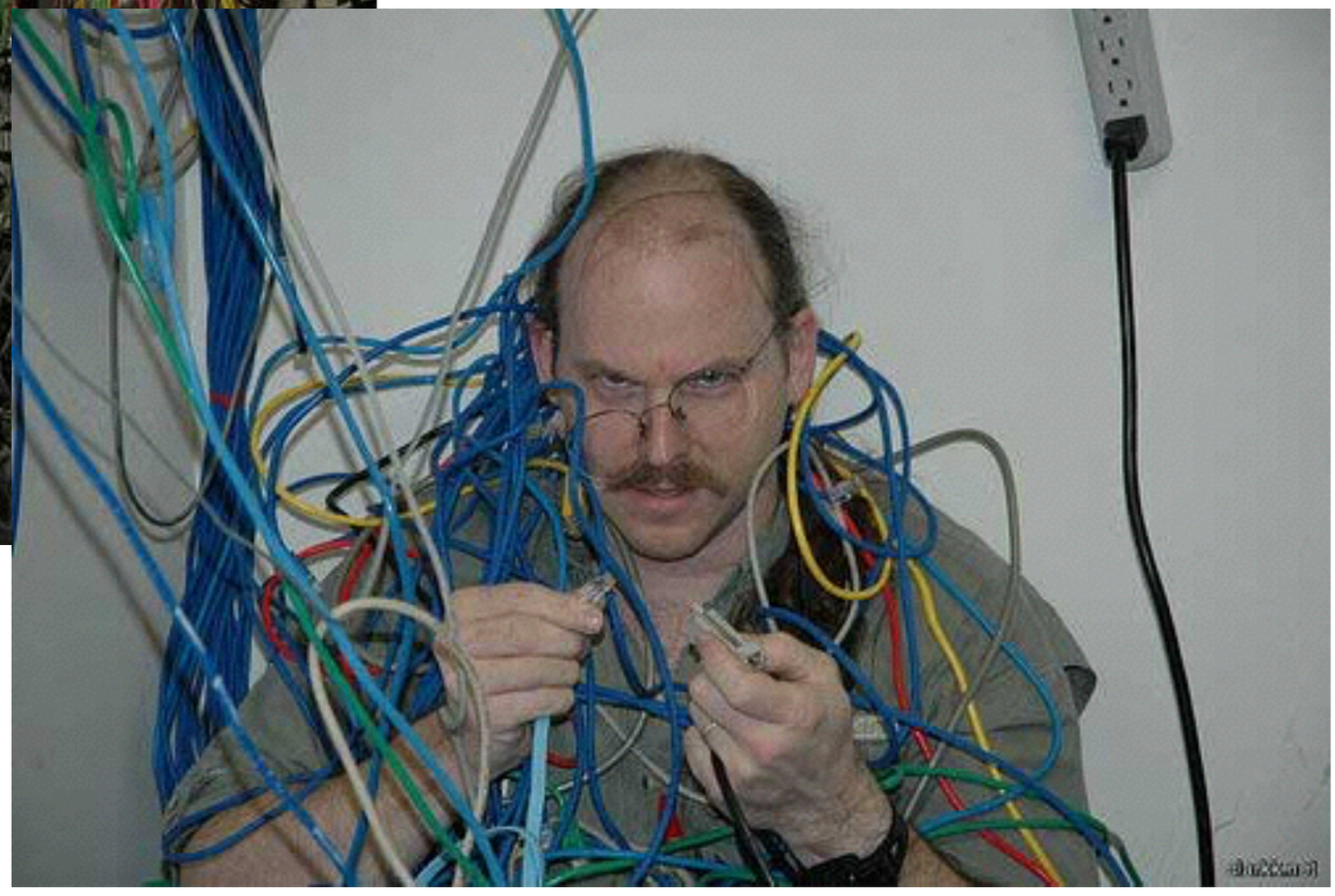
パッチパネルを作ろう

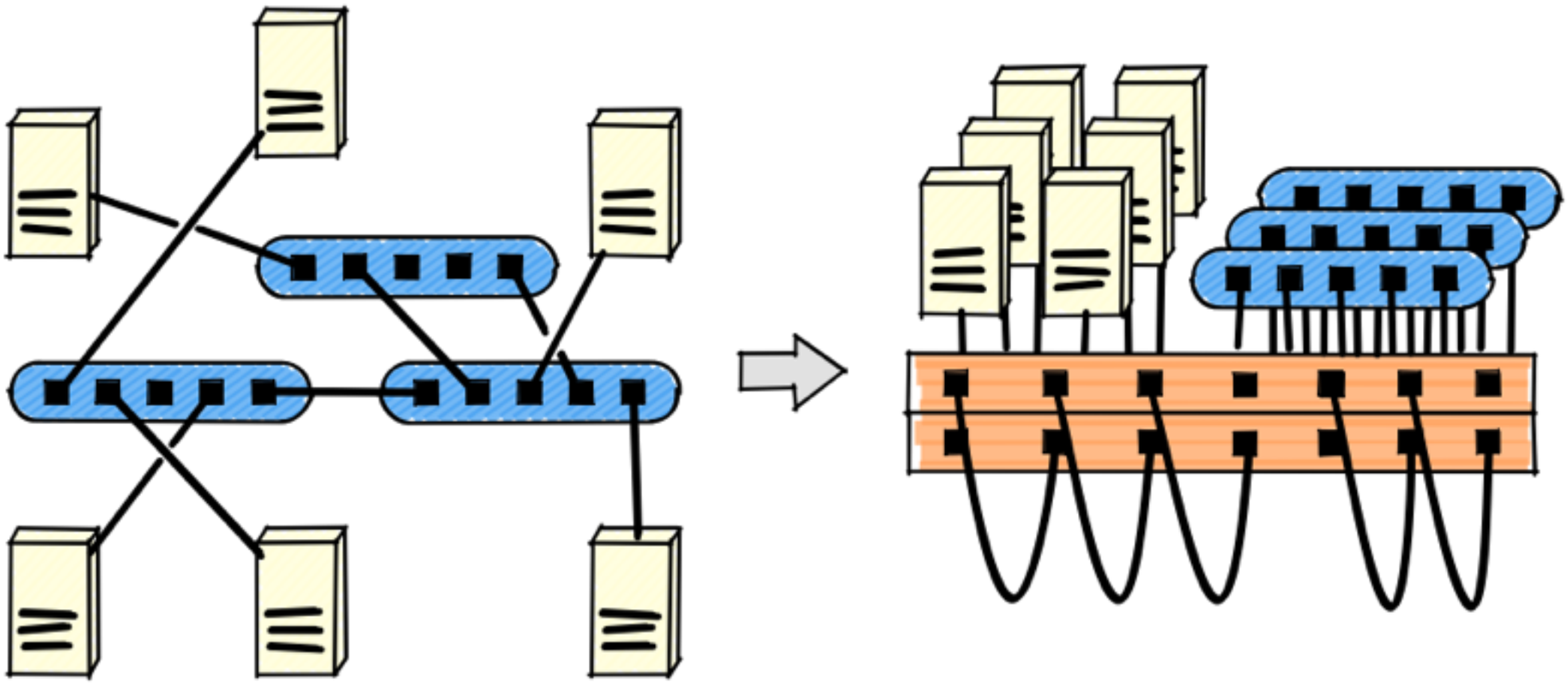
パッチパネルとは





http://www.vibrant.com/cable-messes.html

パッチパネル



・物理配線をパッチパネルで中継

・パッチ操作だけで配線が自由に変更できる

インテリジェント パッチパネル

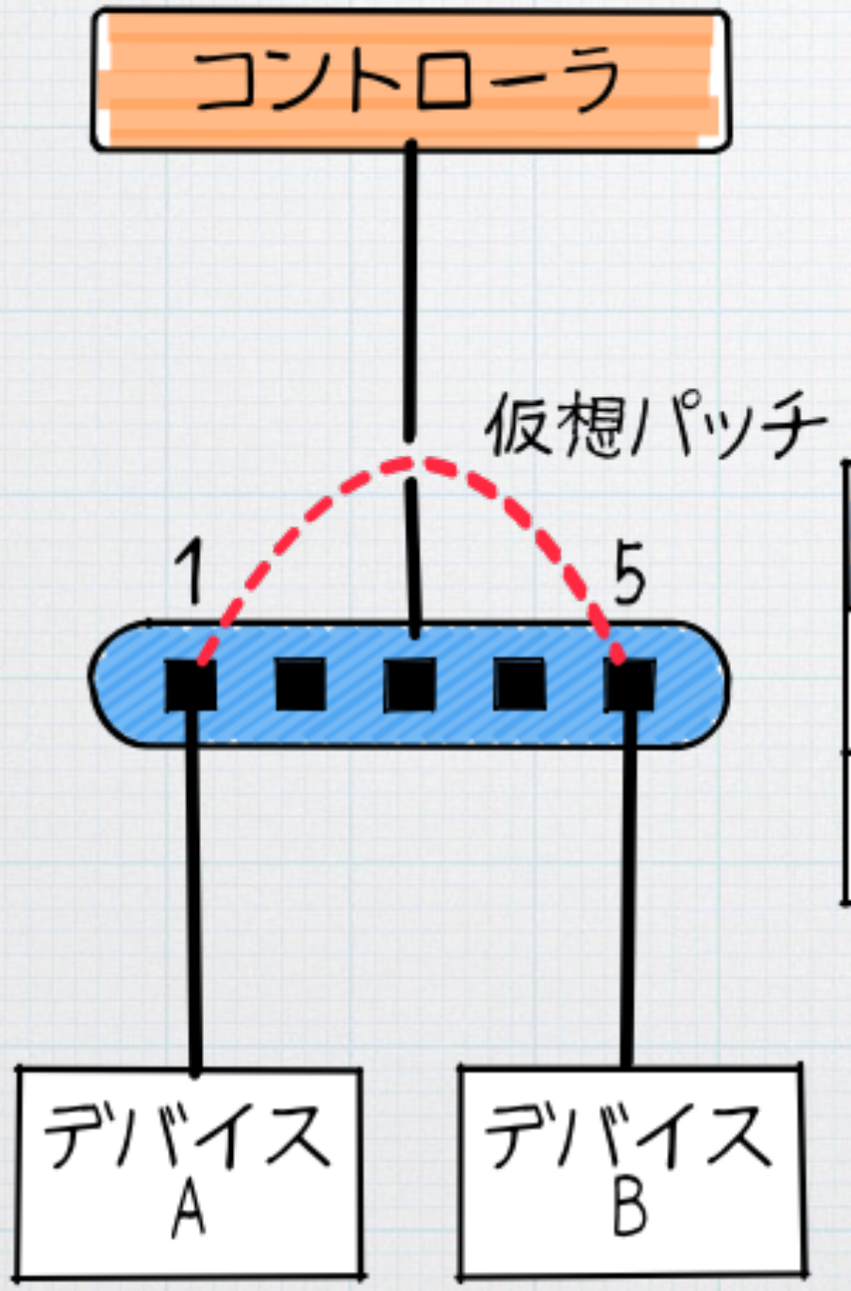
http://www.leviton.com/OA_HTML/SectionDisplay.jsp?section=62268&minisite=10251

・パッチ操作をリモート化

・サーバ室まで行く手間が省ける

・買うと大体100万円以上。けっこう高い

OpenFlow版パッチパネル



マッチフィールド	アクション
ポート1番へ入力	ポート5番へ出力
ポート5番へ入力	ポート1番へ出力

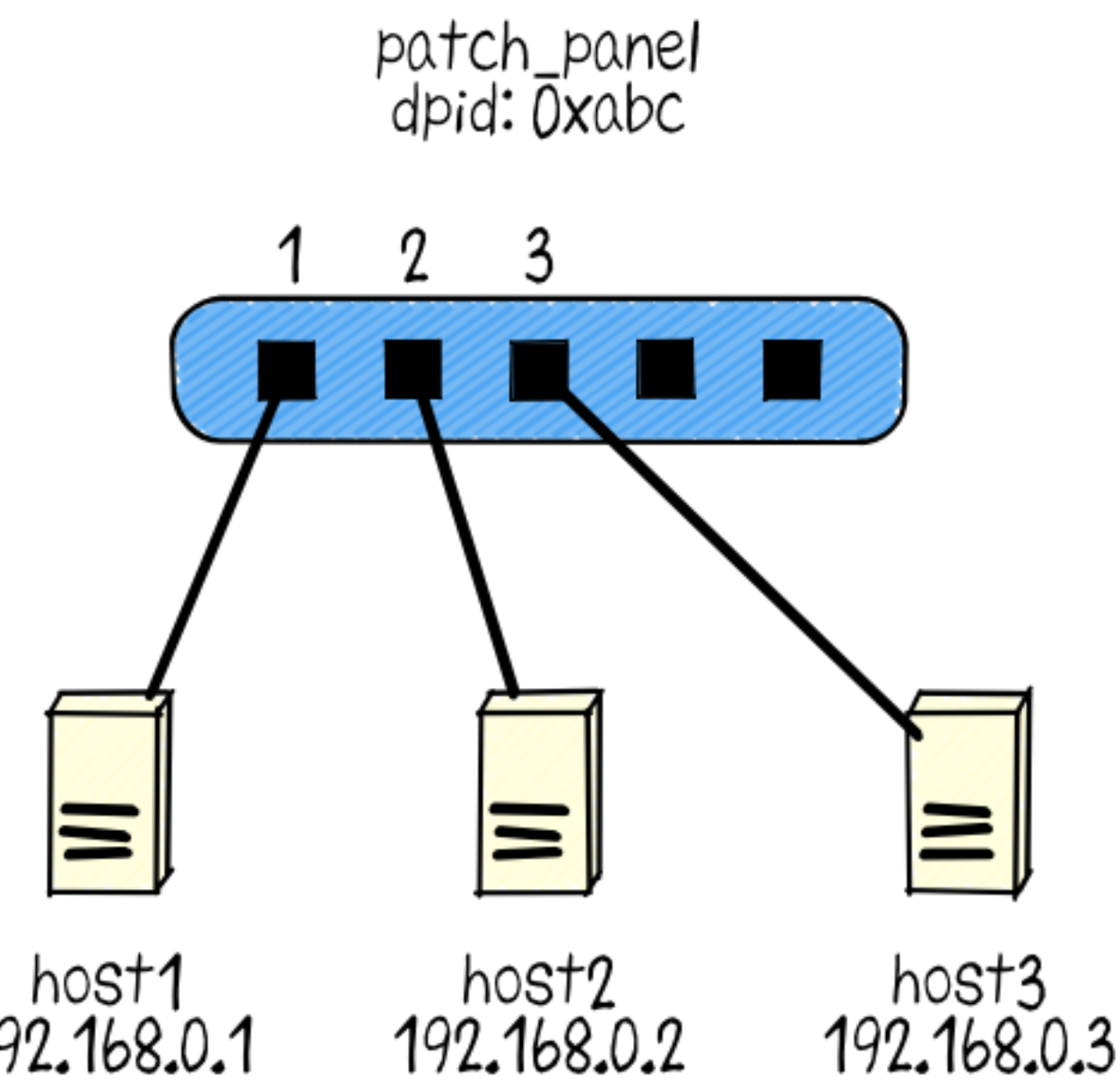
動かしてみよう

patch_panel リポジトリ

使う準備: 課題用リポジトリで bundle install

- 主なファイル
  - patch_panel.conf: 仮想NW設定ファイル
  - lib/patch_panel.rb: コントローラ
  - bin/patch_panel: 操作コマンド

設定NWファイル




```
vswitch('patch_panel') { datapath_id 0xabc }

vhost ('host1') { ip '192.168.0.1' }
vhost ('host2') { ip '192.168.0.2' }
vhost ('host3') { ip '192.168.0.3' }

link 'patch_panel', 'host1'
link 'patch_panel', 'host2'
link 'patch_panel', 'host3'
```

起動

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

パケット送受信

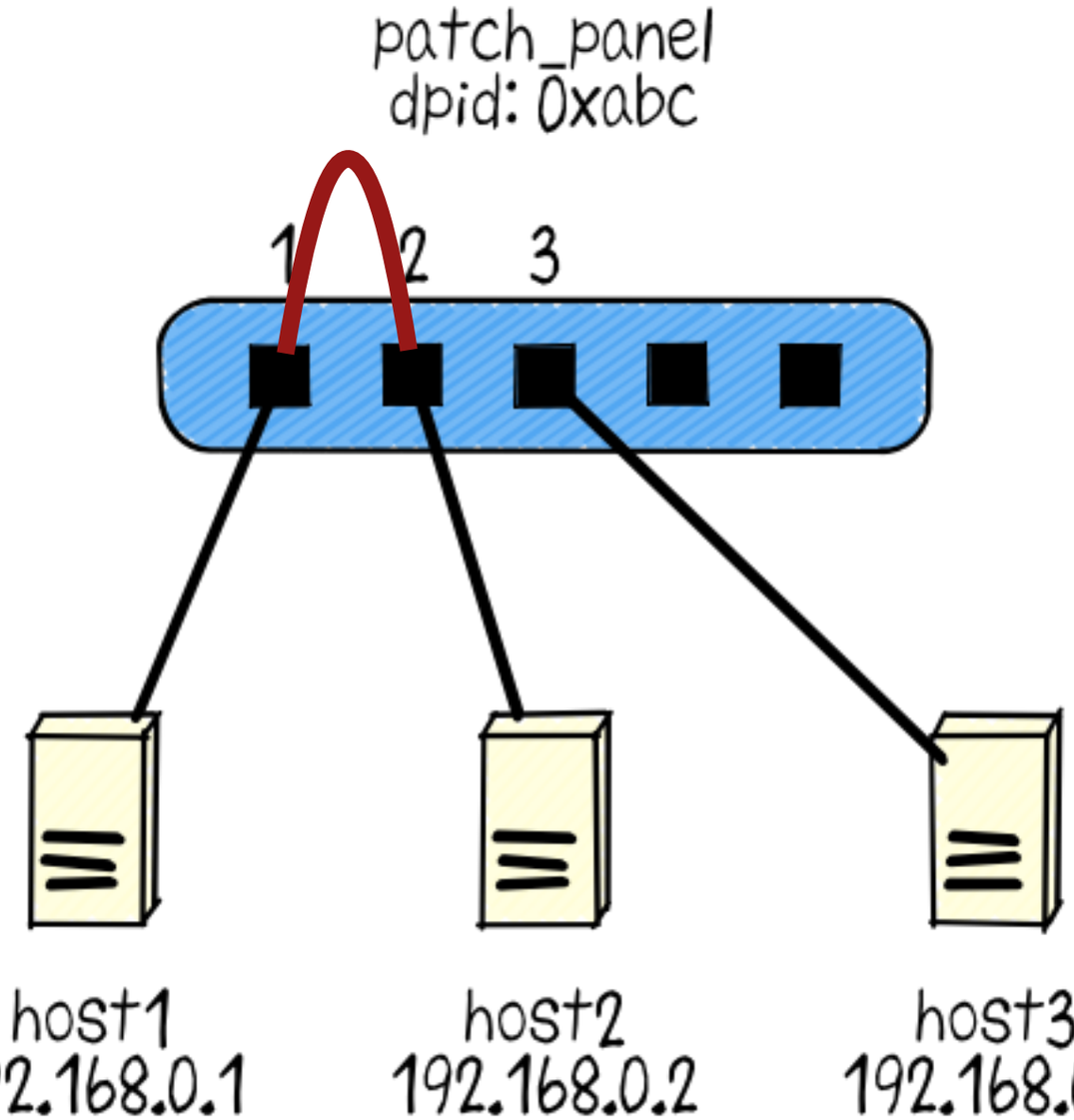
```
$ ./bin/trema send_packets
    –source host1 –dest host2
```

```
$ ./bin/trema send_packets
    –source host2 –dest host1
```

送受信の確認

```
$ ./bin/trema show_stats host1
```

```
$ ./bin/trema show_stats host1
```



次のコマンドでパッチを追加:

```
./bin/patch_panel create 0xabc 1 2
```

ソースコード

初期化

パッチ
追加・削除

フロー
追加・削除

```
class PatchPanel < Trema::Controller
  def start(_args)
    @patch = Hash.new([].freeze)
    logger.info "#{name} started."
  end

  def switch_ready(dpid)
    @patch[dpid].each do |port_a, port_b|
      delete_flow_entries dpid, port_a, port_b
      add_flow_entries dpid, port_a, port_b
    end
  end

  def create_patch(dpid, port_a, port_b)
    add_flow_entries dpid, port_a, port_b
    @patch[dpid] += [port_a, port_b].sort
  end

  def delete_patch(dpid, port_a, port_b)
    …
  end

  private

  def add_flow_entries(dpid, port_a, port_b)
    send_flow_mod_add(dpid,
                      match: Match.new(in_port: port_a),
                      actions: SendOutPort.new(port_b))
    send_flow_mod_add(dpid,
                      match: Match.new(in_port: port_b),
                      actions: SendOutPort.new(port_a))
  end

  def delete_flow_entries(dpid, port_a, port_b) …
```

パッチ情報の初期化

```
class PatchPanel < Trema::Controller
  def start(_args)
    @patch = Hash.new([].freeze)
    logger.info "#{name} started."
  end
```

パッチ初期値: @patch[0xabc] = []

1と2をパッチ: @patch[0xabc] = [[1, 2]]

3と4をパッチ: @patch[0xabc] = [[1, 2], [3,4]]

フローテーブルの初期化

```
def switch_ready(dpid)
  @patch[dpid].each do |port_a, port_b|
    delete_flow_entries dpid, port_a, port_b
    add_flow_entries dpid, port_a, port_b
  end
end
```

@patch[0xabc] = [[1, 2], [3,4]]だったら、

port_a, port_b = [1, 2], [3, 4] で
do…end を 2 回実行（イテレータ）

パッチの作成

```
def create_patch(dpid, port_a, port_b)
  add_flow_entries dpid, port_a, port_b
  @patch[dpid] += [port_a, port_b].sort
end
```

1. add_flow_entries でフローエントリを追加（くわしくは次のスライド）
2. パッチ情報（@patch）を更新

フローエントリの追加

```
def add_flow_entries(dpid, port_a, port_b)
  send_flow_mod_add(dpid,
                    match: Match.new(in_port: port_a),
                    actions: SendOutPort.new(port_b))
  send_flow_mod_add(dpid,
                    match: Match.new(in_port: port_b),
                    actions: SendOutPort.new(port_a))
end
```

port_a→port_b と port_b→port_a
の 2 方向分のフローエントリ 2 個を追加

フローエントリの削除

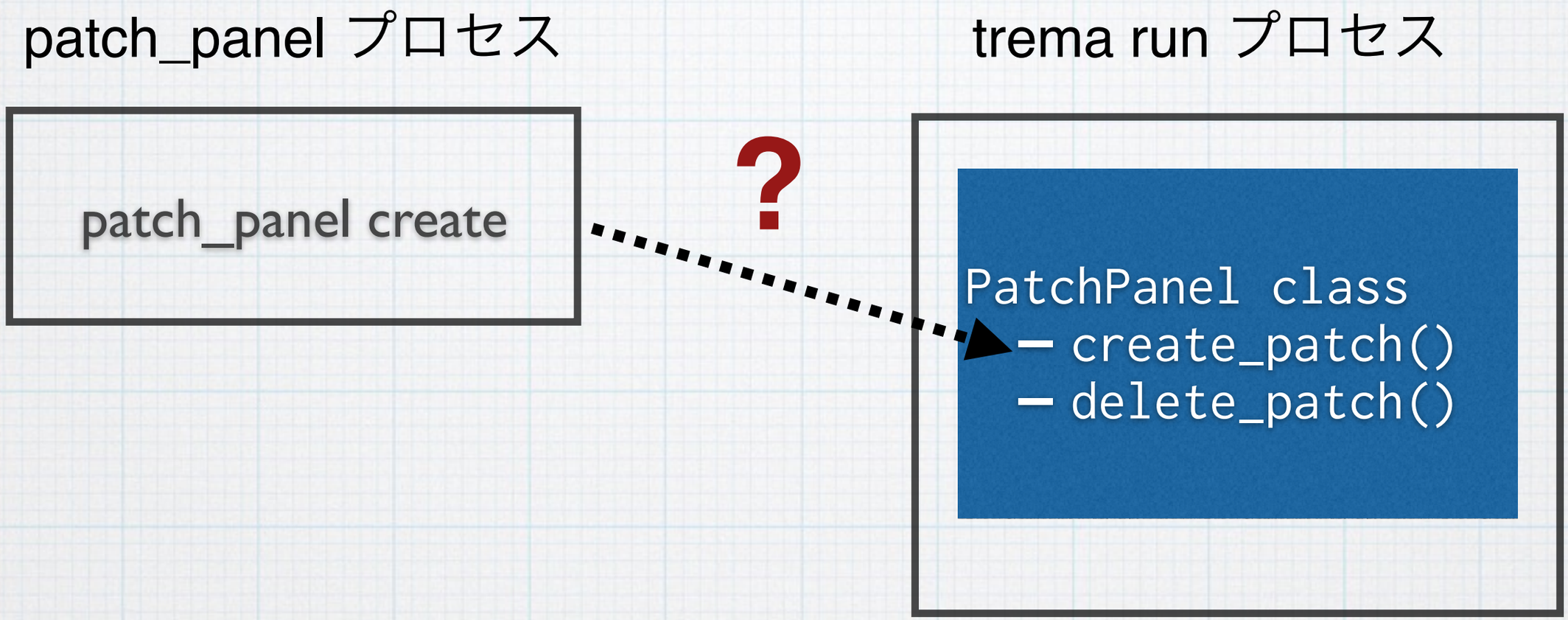
```
def delete_flow_entries(dpid, port_a, port_b)
  send_flow_mod_delete(dpid,
                       match: Match.new(in_port: port_a))
  send_flow_mod_delete(dpid,
                       match: Match.new(in_port: port_b))
end
```

send_flow_mod_delete は send_flow_mod_add の逆にフローエントリを消す

これらを

呼ぶには?

```ruby
class PatchPanel < Trema::Controller
  def start(_args)
    @patch = Hash.new([].freeze)
    logger.info "#{name} started."
  end

  def switch_ready(dpid)
    @patch[dpid].each do |port_a, port_b|
      delete_flow_entries dpid, port_a, port_b
      add_flow_entries dpid, port_a, port_b
    end
  end

  def create_patch(dpid, port_a, port_b)
    add_flow_entries dpid, port_a, port_b
    @patch[dpid] += [port_a, port_b].sort
  end

  def delete_patch(dpid, port_a, port_b)
    …
  end

  private

  def add_flow_entries(dpid, port_a, port_b)
    send_flow_mod_add(dpid,
                      match: Match.new(in_port: port_a),
                      actions: SendOutPort.new(port_b))
    send_flow_mod_add(dpid,
                      match: Match.new(in_port: port_b),
                      actions: SendOutPort.new(port_a))
  end

  def delete_flow_entries(dpid, port_a, port_b) …
```


patch_panel プロセス
trema run プロセス
patch_panel create
?
PatchPanel class
– create_patch()
– delete_patch()

patch_panel プロセス

patch_panel create

ソケット経由で
リモートメソッド
呼出し

trema run プロセス

PatchPanel class
- create_patch()
- delete_patch()

作成

UNIX ドメイン
ソケット

./bin/patch_panel create

```
desc 'Creates a new patch'
arg_name 'dpid port#1 port#2'
command :create do |c|
  c.desc 'Location to find socket files'
  c.flag [:S, :socket_dir], default_value: Trema::DEFAULT_SOCKET_DIR

  c.action do |_global_options, options, args|
    dpid = args[0].hex
    port1 = args[1].to_i
    port2 = args[2].to_i
    Trema.trema_process('PatchPanel', options[:socket_dir]).controller.
      create_patch(dpid, port1, port2)
  end
end
```

- Trema.trema_process で trema run プロセスにアクセス
- #controller でコントローラオブジェクトにアクセス

まとめ

- OpenFlowでのパッチパネルの実装方法
- フローエントリの消しかた
- コントローラの操作コマンドの書き方

  → これを活かして、実用的な
  コントローラを作ろう