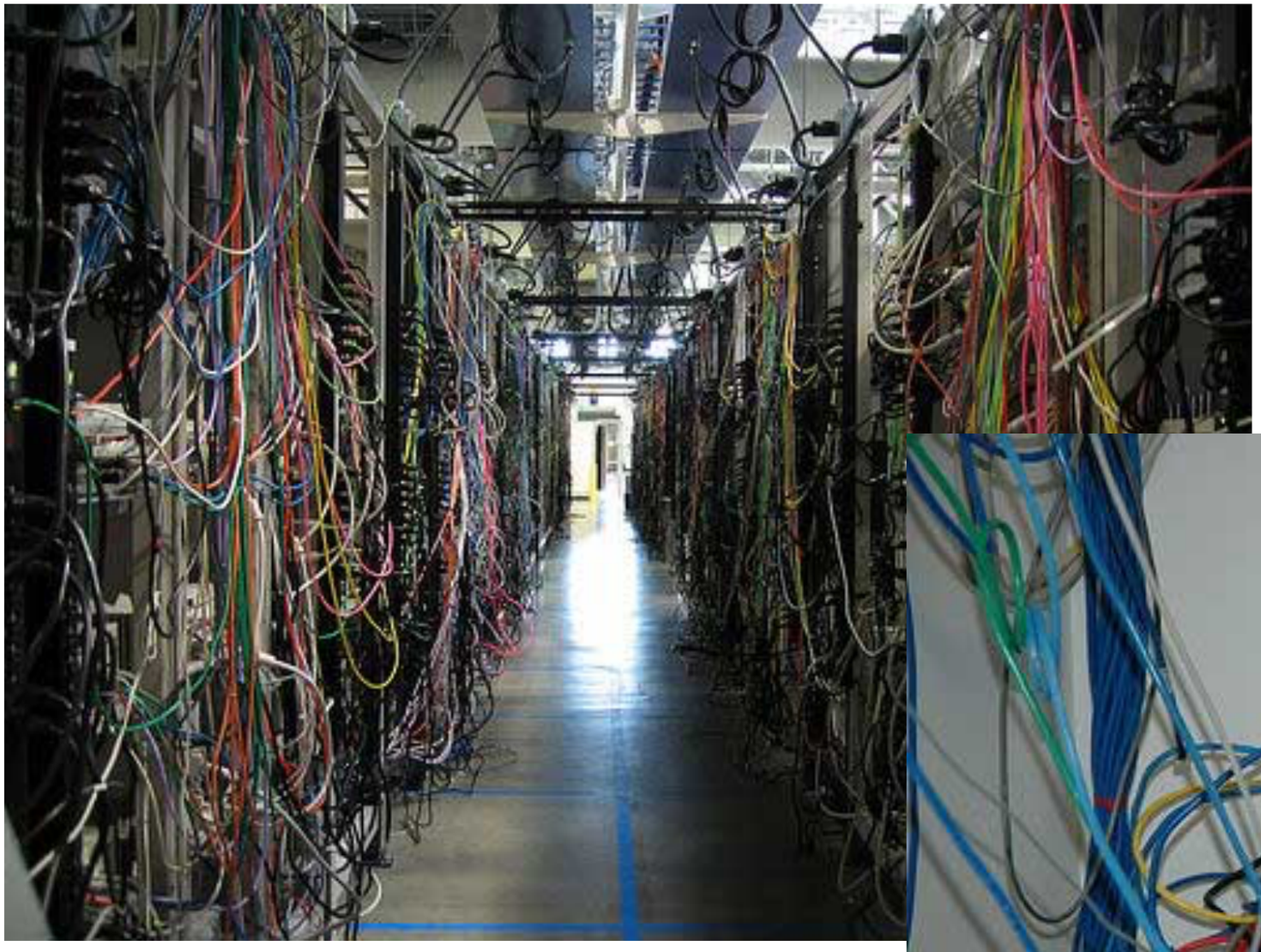
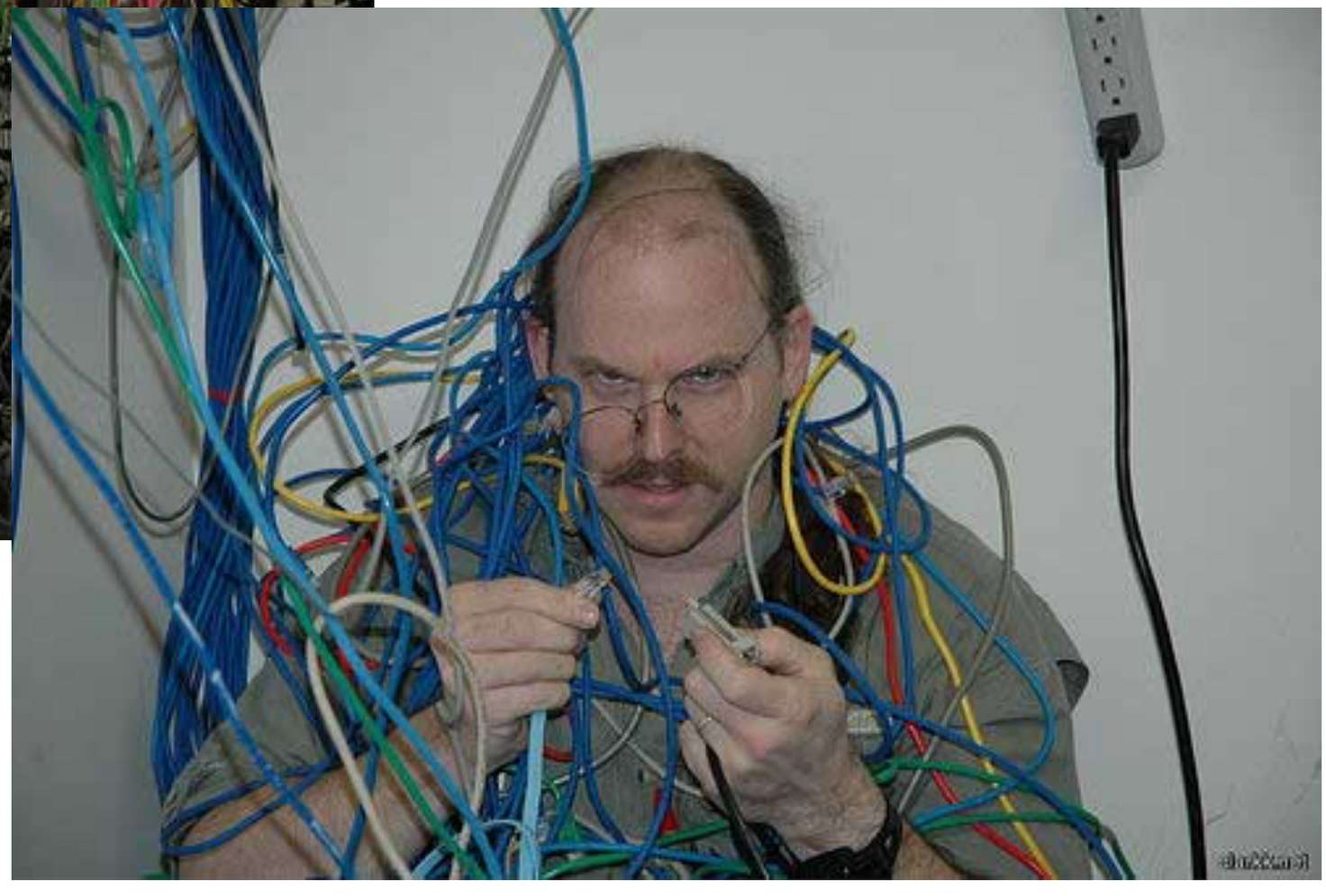
Build a Patch Panel with OpenFlow

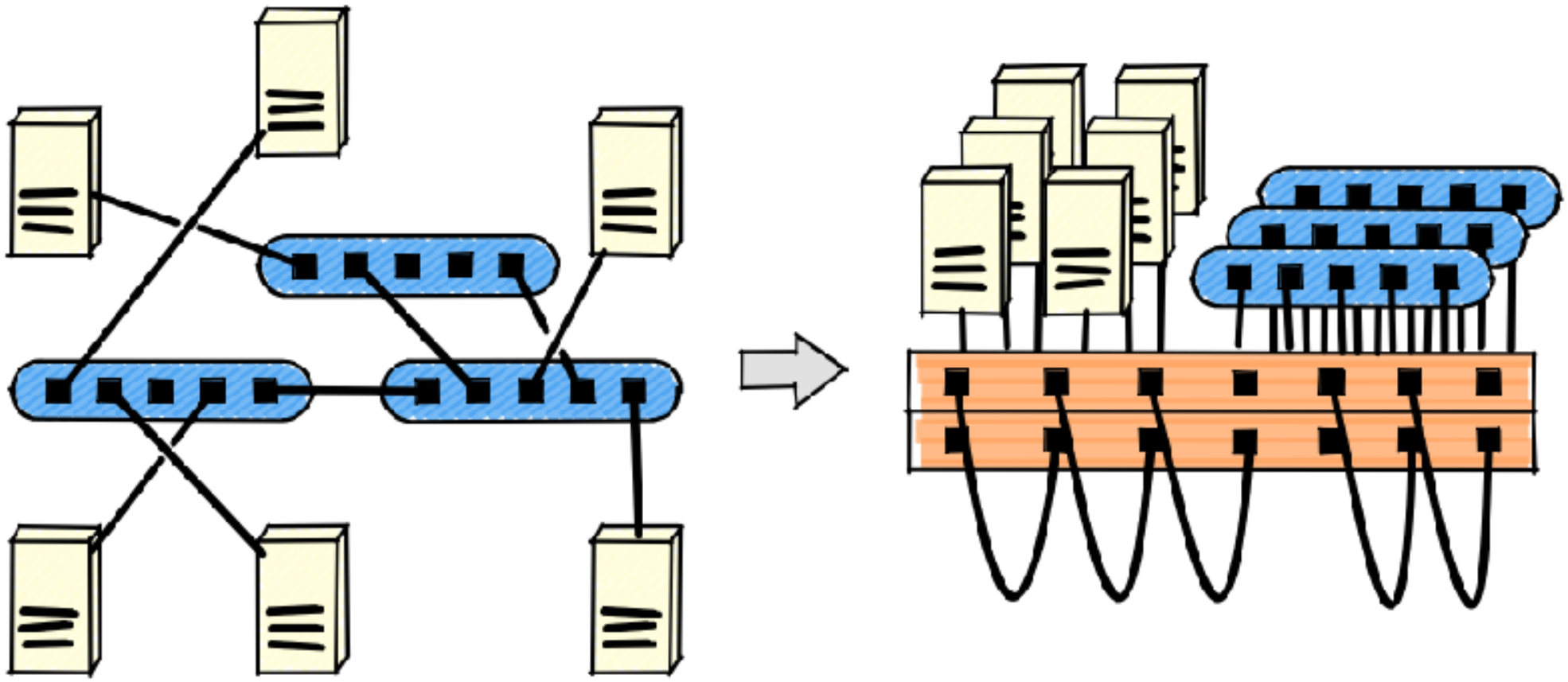
What is a patch panel?





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

Patch Panel



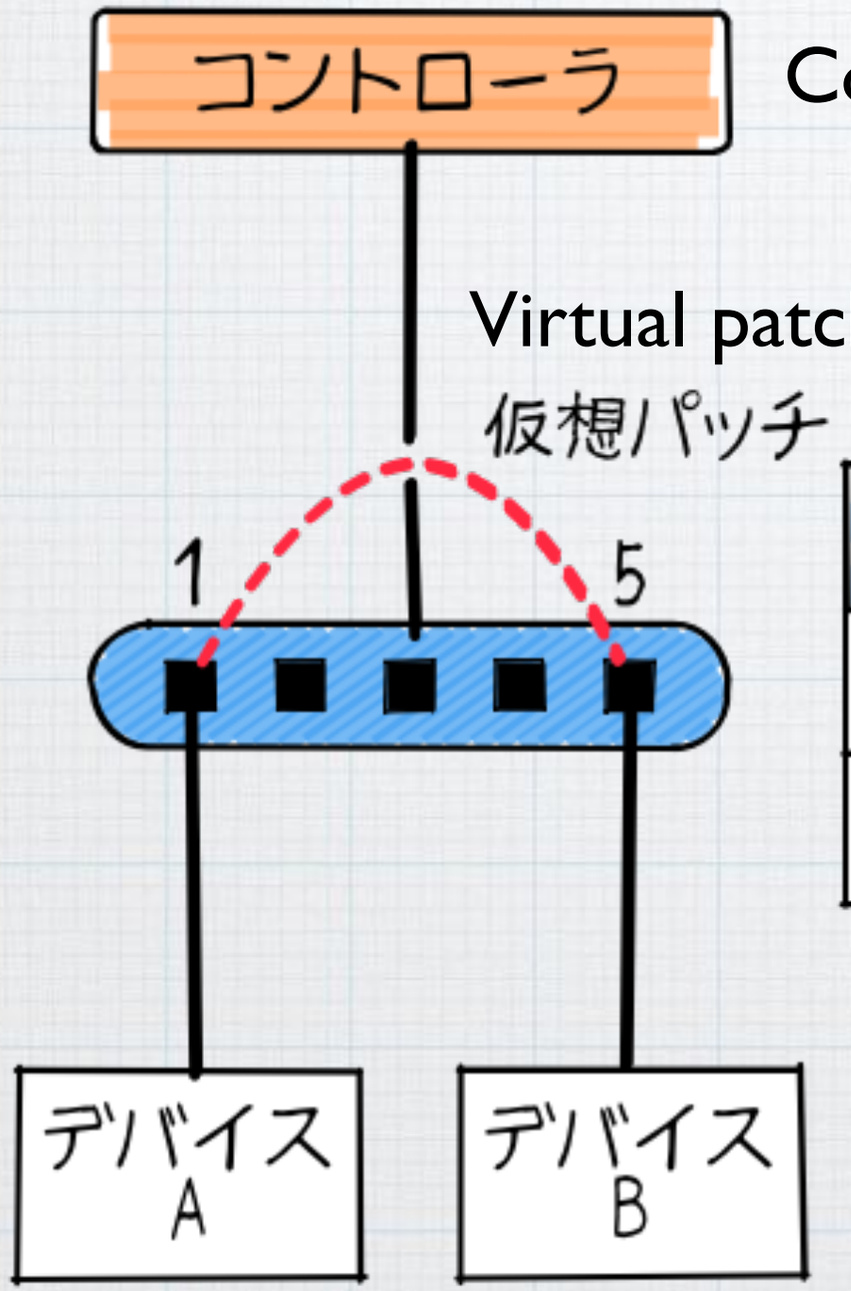
- A device to interconnect circuits
- Circuits can be easily modified by changing patches

Intelligent Patch Panel

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

- Remote control of patch configurations
  - Operators do not have to go to a server room
- Expensive (More than 1 million JPY)

Patch Panel with OpenFlow



Match field	Action
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

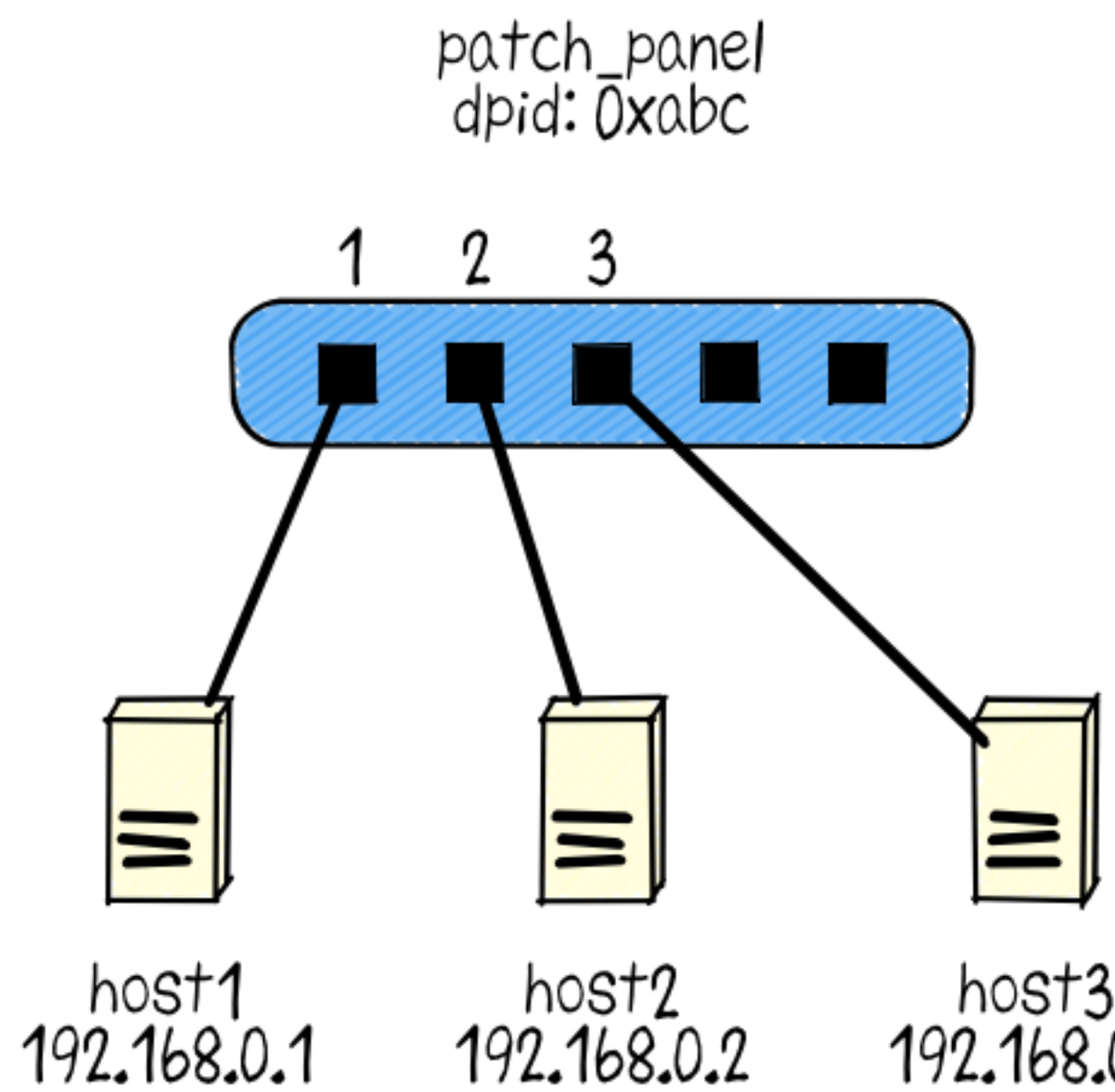
Let’ s run a patch panel

patch_panel repository

Prerequisite: run “bundle install” on the repository

- Files
  - patch_panel.conf: Virtual network configuration
  - lib/patch_panel.rb: Controller
  - bin/patch_panel: Patch configuration commands

Virtual Network Configuration




```
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'
```

Run

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

Receive Packets

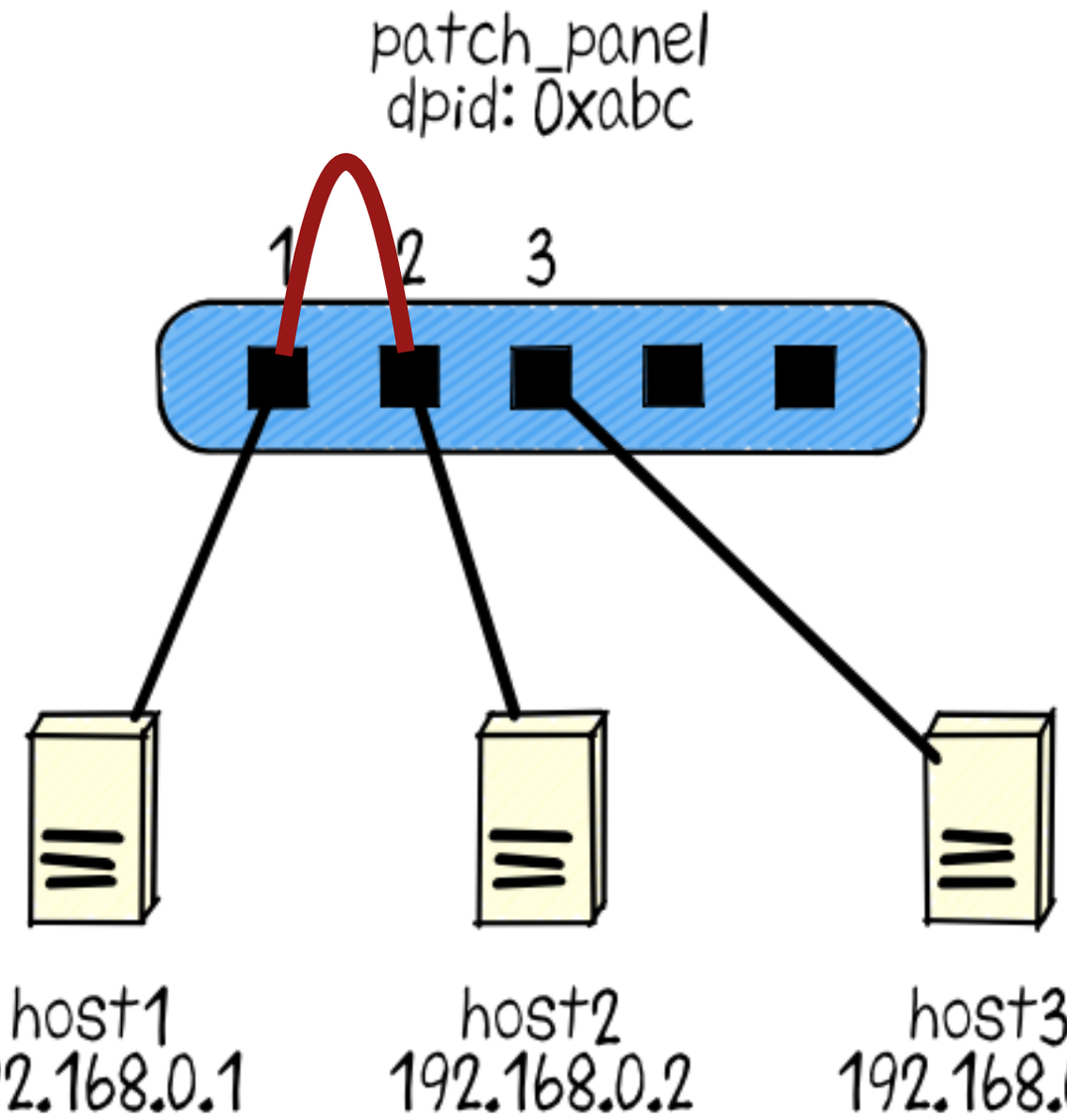
```
$ ./bin/trema send_packets
    –source host1 –dest host2
```

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

Confirm RX/TX

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

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



Add a patch:

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

Source code

Initialization

Add/delete a patch

Add/delete a flow

```
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) …
```

Initialize Patch Configurations

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

Initialize patch configurations: @patch[0xabc] = []

Add a patch between 1 and 2: @patch[0xabc] = [[1, 2]]

Add a patch between 3 and 4: @patch[0xabc] = [[1, 2], [3,4]]

Initialize Flow Table

```
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
```

Assume the case of @patch[0xabc] = [[1, 2], [3,4]]

In this case, operations between do…end are performed twice with being port_a, port_b = [1, 2], [3, 4].

Create a Patch

```
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 a flow entry, using add_flow_entries
2. Update patch information stored in @patch

Add Flow Entries

```
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
```

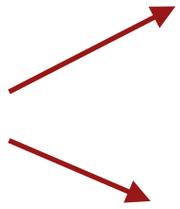
Add two flow entries, flows from port_a to port_b and port_b to port_a, i.e., create a full-duplex flow.

Delete Flow Entries

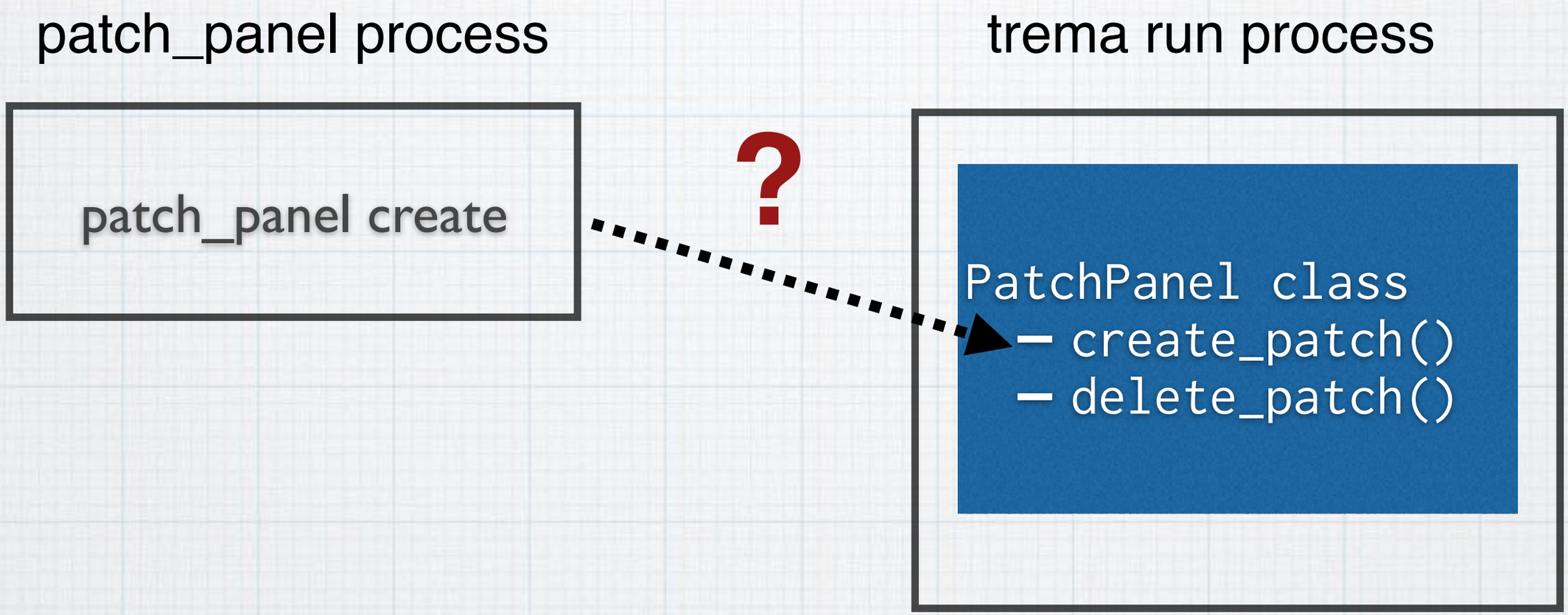
```
def delete_flow_entries(dpid, port_a, port_b)
  send_flow_mod_delete(dpid,
                       match: Match.new(in_port: port_a))
  send_flow_mod_delete(dpid,
                       match: Match.new(in_port: port_b))
end
```

In contrast to add_flow_entries, delete_flow_entries deletes the two flow entries.

How to call these methods?



```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 process
trema run process
patch_panel create
?
PatchPanel class
– create_patch()
– delete_patch()

patch_panel process

trema run process

patch_panel create

```
PatchPanel class
  – create_patch()
  – delete_patch()
```

Call remote methods
via UNIX sockets

Create

UNIX Sockets

./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
```

- Access the trema run process with Trema.trema_process
- Access the controller object with #controller

Conclusion

- How to implement a patch panel with OpenFlow
- How to add/delete flow entries
- How to make commands to control a controller
  → Build a practical controller, using these techniques