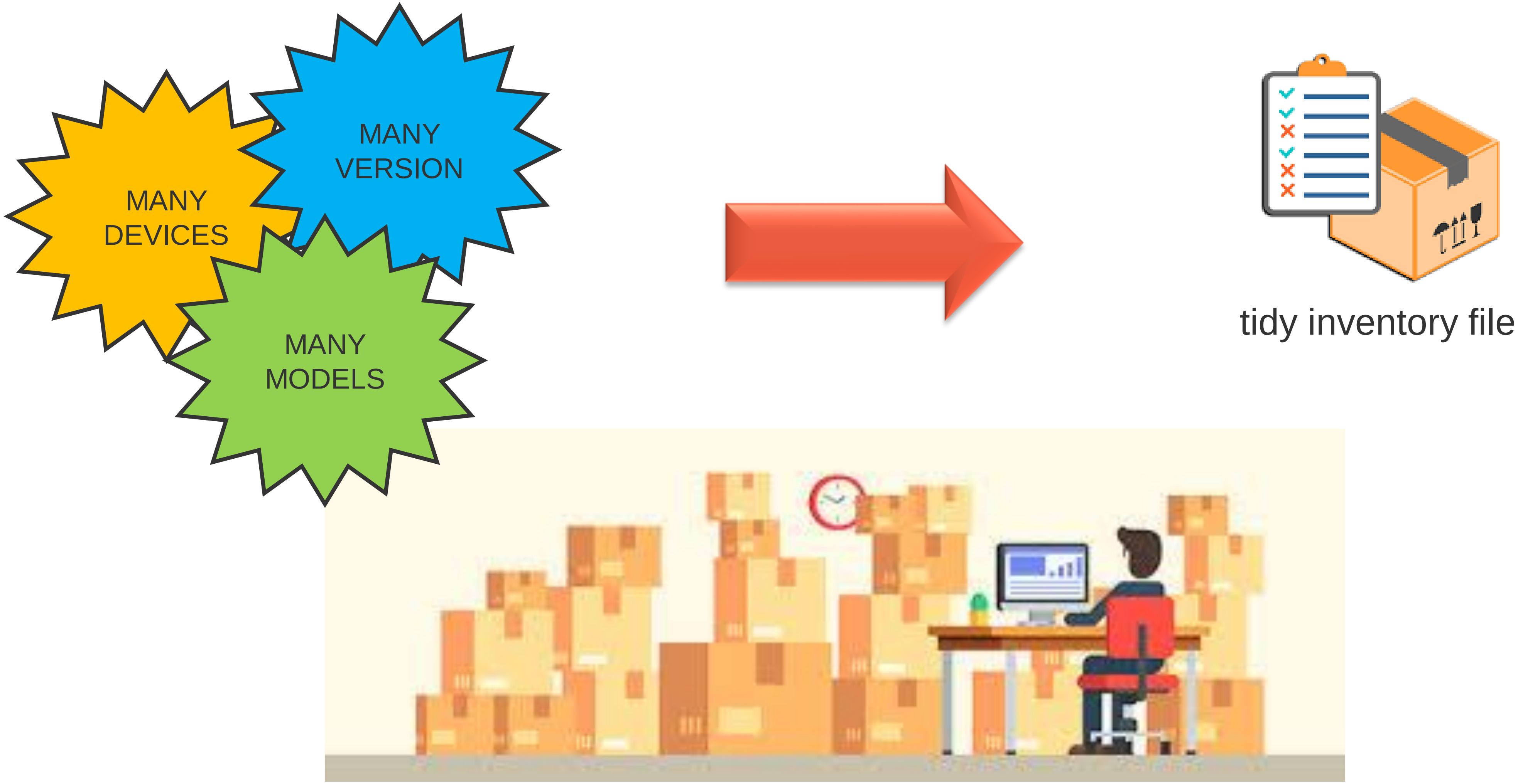
USING PYEZ TO AUTOMATE YOUR NETWORK

Umberto Manferdini

WHAT’S IN MY INSTALL BASE?



MANY DEVICES

MANY VERSION

MANY MODELS

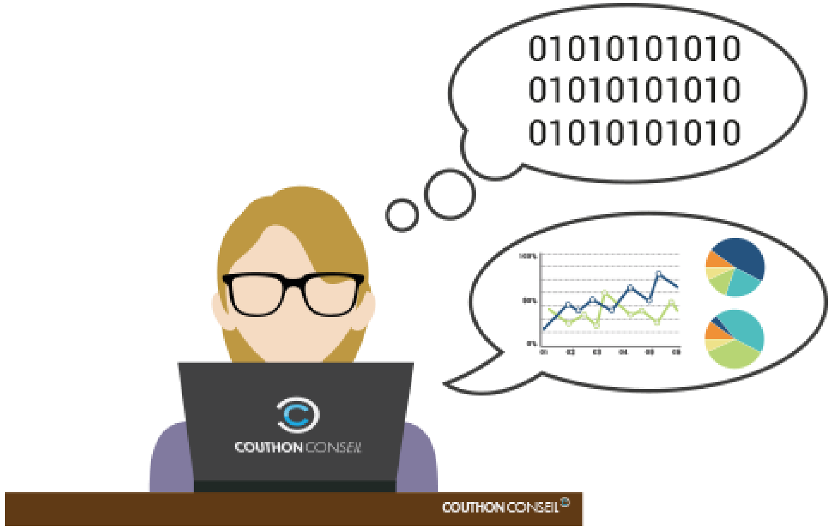
tidy inventory file

PYEZ GATHER FACTS FROM A DEVICE

PYEZ COLLECTS FACTS

```
>>> dev.facts
{'2RE': False,
 'HOME': '/root',
 'RE0': {'last_reboot_reason': 'Router rebooted after a normal
shutdown.',
         'mastership_state': 'master',
         'model': 'RE-VMX',
         'status': 'OK',
         'up_time': '2 days, 1 hour, 25 minutes, 13 seconds'},
 'RE1': None,
 'RE_hw_mi': False,
'domain': 'englab.juniper.net',
 'fqdn': 'r3_re.englab.juniper.net',
'hostname_info': {'re0': 'r3_re'},
 …
}
```

WHAT CAN I SHOW?



Which data?

- Individual devices
- Aggregated information
  - Versions
  - models

OUTPUT EXAMPLE

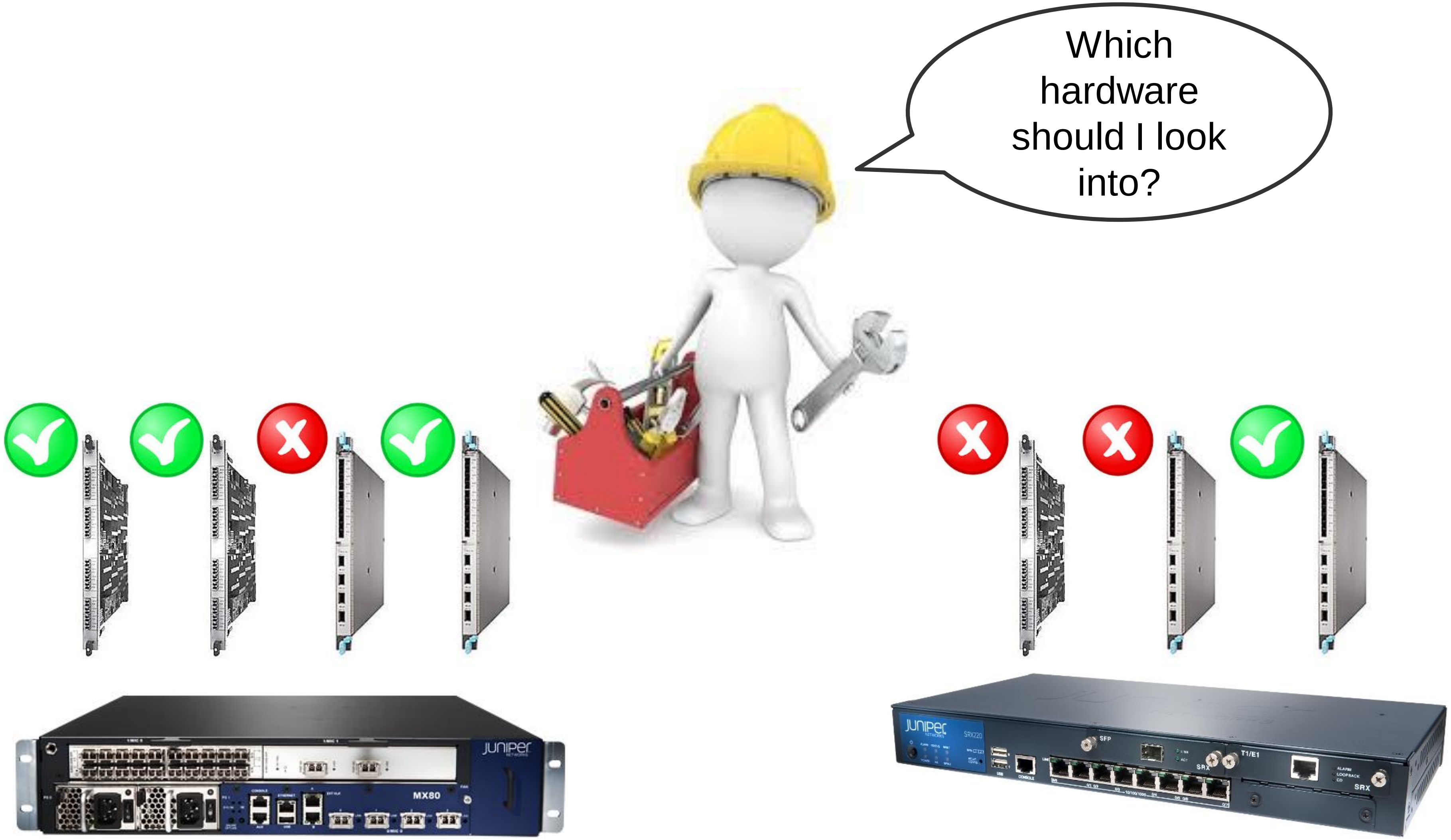
```
root@ubuntu:~/pyez# python inventory.py -d g
Starting collecting infos...

Installed base versions
16.1R3.10: 8
Installed base models
MX960: 8
Installed base double RE architecture
False: 8
Individual routers information
r4:
        hostname: r4_re
        mgmt ip: 10.92.35.194
        model: MX960
        version: 16.1R3.10
        serial number: VMX1e0a
        double RE: False
------------------
…other devices
```

Aggregated information to know how my install base looks like

If needed individual information for each router

IS MY HARDWARE HEALTHY?



Which hardware should I look into?

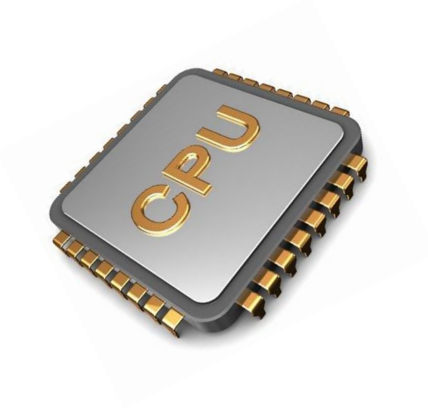
HUNTING ISSUES!

We can monitor:

- State
- Model
- CPU
- Memory

We may set thresholds to highlight cards with:

- High CPU
- High memory



OUTPUT EXAMPLE

```
root@ubuntu:~/pyez# python fpcs.py -f ahosts -c 20 -m 20
Using the following thresholds:
        cpu: 20%
        memory: 20%
Starting collecting infos...
Devices FPCs inventory:
r1      ->      FPC 0:Virtual FPC
r2      ->      FPC 0:Virtual FPC
r3      ->      FPC 0:Virtual FPC

*******************
Devices relevant information
r1
        Device has
        online FPCs: 0
        empty FPCs: 11
        offline FPCs: 1
        Device has 1 fpc in offline state. They are in slots ['0'].
----------
r2
        Device has
        online FPCs: 1
        empty FPCs: 11
        offline FPCs: 0
        FPCS with high cpu values: 1
                slot 0 : 21%
        FPCs with high memory values: 1
                slot 0 : 30%
----------
```

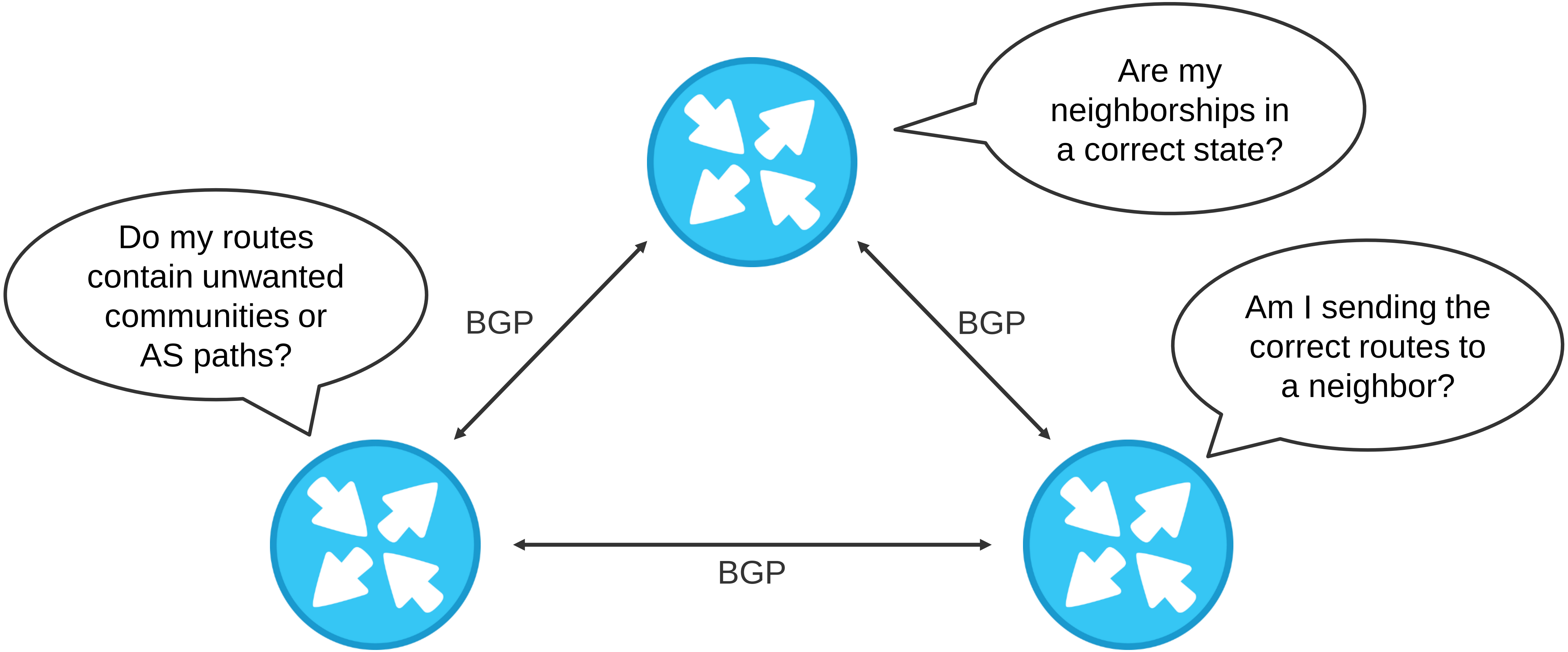
Configured memory and cpu thresholds

For each device, we can print the list of installed FPC and their model

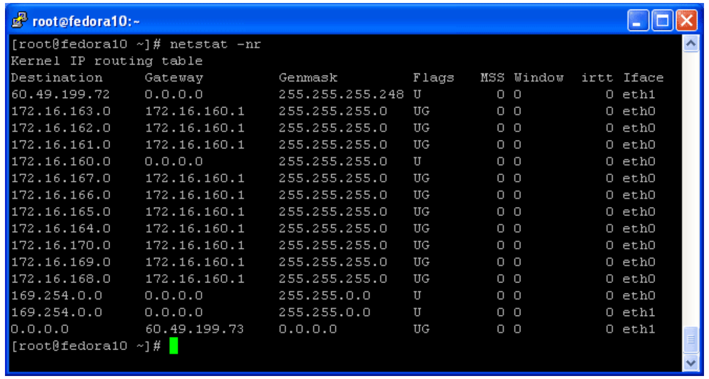
We scan devices, looking for:

- FPC state
  - Online
  - Empty
  - Offline
- High values
  - cpu
  - memory

BGP AS A FRIEND


Are my neighborships in a correct state?
Do my routes contain unwanted communities or AS paths?
BGP
BGP
Am I sending the correct routes to a neighbor?
BGP

TRANSFORM EVERY JUNOS INFORMATION INTO DATA!



```
[root@fedora10 ~]# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
60.49.199.72    0.0.0.0         255.255.255.248 U         0 0          0 eth1
172.16.163.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.162.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.161.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.160.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
172.16.167.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.166.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.165.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.164.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.170.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.169.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.168.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth1
0.0.0.0         60.49.199.73    0.0.0.0         UG        0 0          0 eth1
[root@fedora10 ~]#
```

use PyEz to get BGP routes and state via Junos RPC

Python dictionary-like data structure

Import and process many information:

- Route
- AS path
- Community
- state
- Etc…

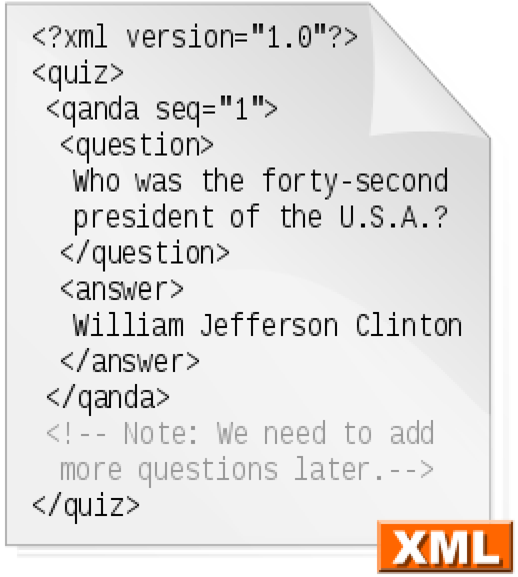
THE SHOPPING LIST

```
root@ubuntu:~/pyez# cat yhosts.yaml
r3:
  ip: "10.92.35.196"
  badas:
    - "700"
    - "400 500"
  badcomm:
    - "400:1"
  badroutes:
    - "20.1.0.0/16"
    - "20.2.0.0/15"
  checkann:
    "192.168.23.0":
      - "20.3.0.0/16"
      - "20.4.0.0/16"
    "192.168.34.0":
      - "10.20.0.0/16"
```

For each we router we tell what we want to check:

- Routes with undesired AS paths
- Routes tagged with unwanted communities
- Table contains routes that should not be there
- If we advertise certain routes to a certain neighbor

FROM JUNOS API TO PYTHON VIEWS



```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

Through PyEz we create customized data structure based on XML replies of Junos RPCs

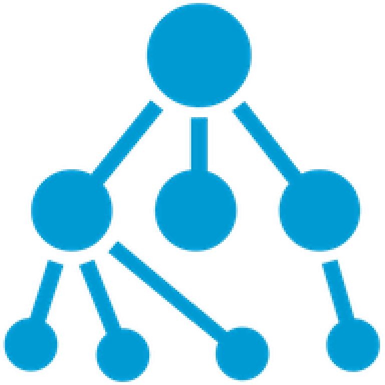
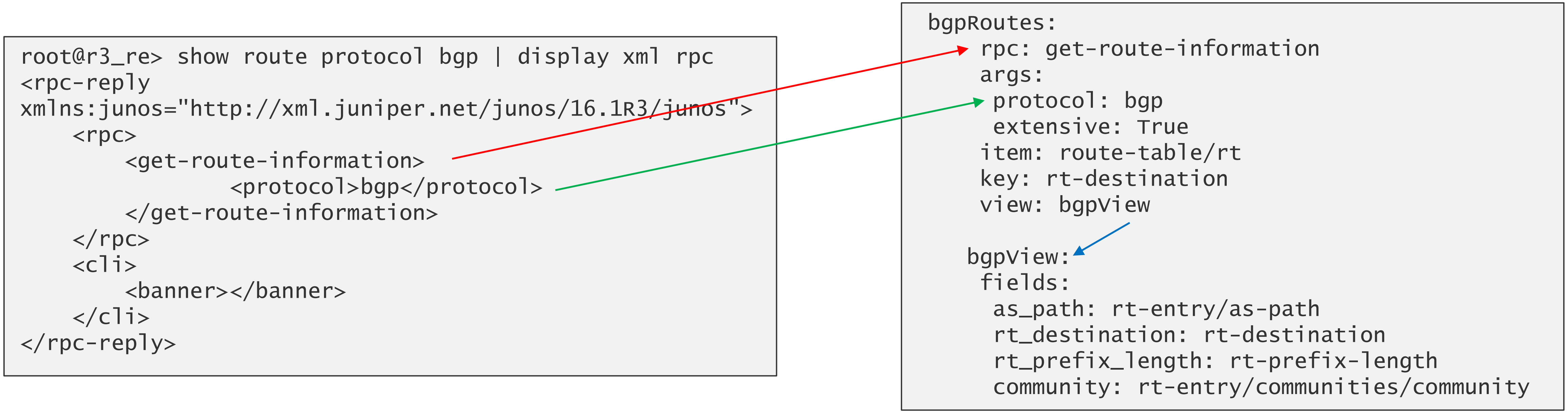


Table is a python data structure populated with information taken from device

CREATING YOUR OWN CUSTOM RPC

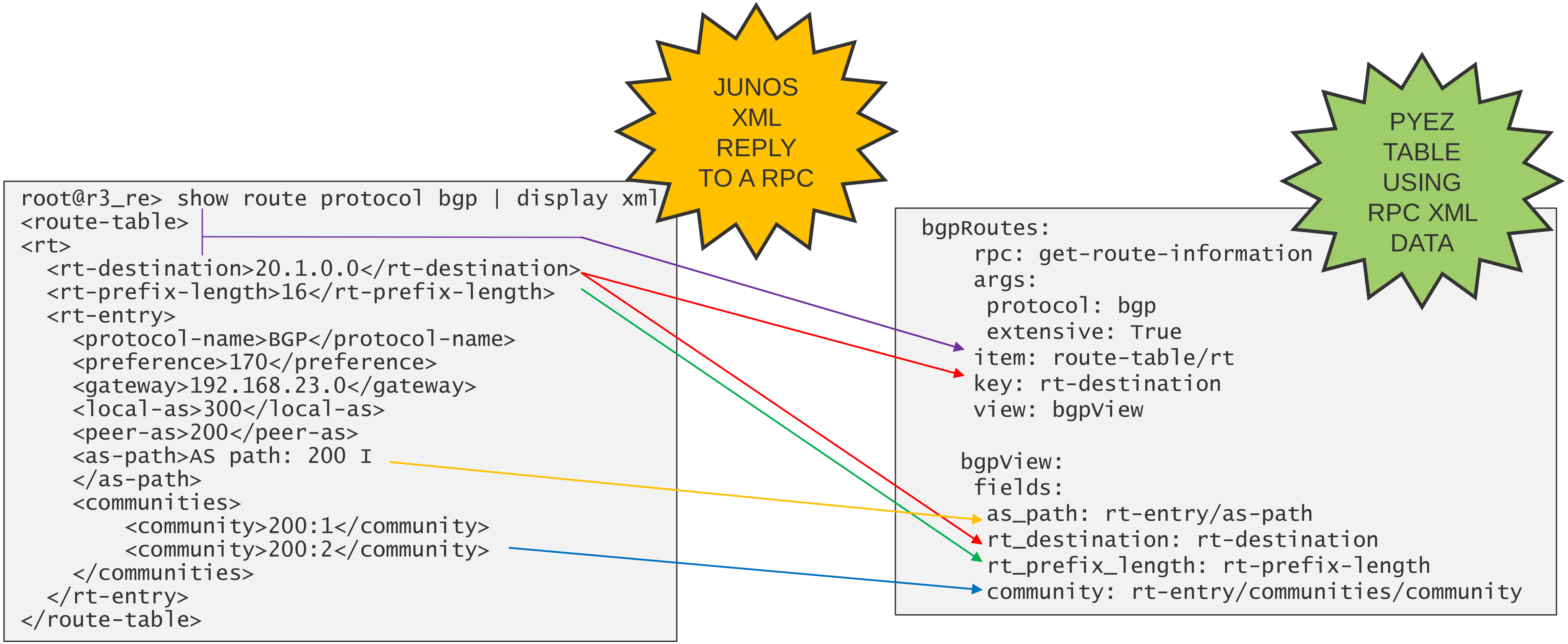


```
root@r3_re> show route protocol bgp | display xml rpc
<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/16.1R3/junos">
    <rpc>
        <get-route-information>
                <protocol>bgp</protocol>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

```
bgpRoutes:
    rpc: get-route-information
    args:
     protocol: bgp
     extensive: True
    item: route-table/rt
    key: rt-destination
    view: bgpView

   bgpView:
    fields:
     as_path: rt-entry/as-path
     rt_destination: rt-destination
     rt_prefix_length: rt-prefix-length
     community: rt-entry/communities/community
```

MAPPING JUNOS DATA INTO PYTHON



JUNOS XML REPLY TO A RPC

```
root@r3_re> show route protocol bgp | display xml
<route-table>
<rt>
  <rt-destination>20.1.0.0</rt-destination>
  <rt-prefix-length>16</rt-prefix-length>
  <rt-entry>
    <protocol-name>BGP</protocol-name>
    <preference>170</preference>
    <gateway>192.168.23.0</gateway>
    <local-as>300</local-as>
    <peer-as>200</peer-as>
    <as-path>AS path: 200 I
    </as-path>
    <communities>
        <community>200:1</community>
        <community>200:2</community>
    </communities>
  </rt-entry>
</route-table>
```

PYEZ TABLE USING RPC XML DATA

```
bgpRoutes:
    rpc: get-route-information
    args:
     protocol: bgp
     extensive: True
    item: route-table/rt
    key: rt-destination
    view: bgpView

   bgpView:
    fields:
     as_path: rt-entry/as-path
     rt_destination: rt-destination
     rt_prefix_length: rt-prefix-length
     community: rt-entry/communities/community
```

GETTING THE DATA YOU WANT!

```
bt = bgpRoutes(dev).get()
```

```
bgpRoutes:
    rpc: get-route-information
    args:
     protocol: bgp
     extensive: True
    item: route-table/rt
    key: rt-destination
    view: bgpView
bgpView:
    fields:
     as_path: rt-entry/as-path
     rt_destination: rt-destination
     rt_prefix_length: rt-prefix-length
     community: rt-entry/communities/community
```

THIS IS OUR “NEW” RPC

The idea is simple:

- Study the structure of a Junos RPC XML reply
- Detect interesting field
- Create a custom RPC and the view which includes the fields you are interested in
- Run the RPC against the device and get the desired information

OUTPUT EXAMPLES

```
root@ubuntu:~/pyez# python bgp.py
Starting collecting infos...

Working on r3 at 10.92.35.196...

All BGP sessions are in a Established state.

Route 20.1.0.0/16 contains:
        is a bad route.
Route 20.3.0.0/16 contains:
        bad AS PATH(s): ['400 500']
Route 20.4.0.0/16 contains:
        bad AS PATH(s): ['400 500']

Checking advertised routes to neighbor 192.168.23.0 ['20.3.0.0/16', '20.4.0.0/16']
        All OK!
Checking advertised routes to neighbor 192.168.34.0 ['10.20.0.0/16']
        10.20.0.0/16 not found
```

→ Checking BGP sessions state

→ Detecting unwanted situations

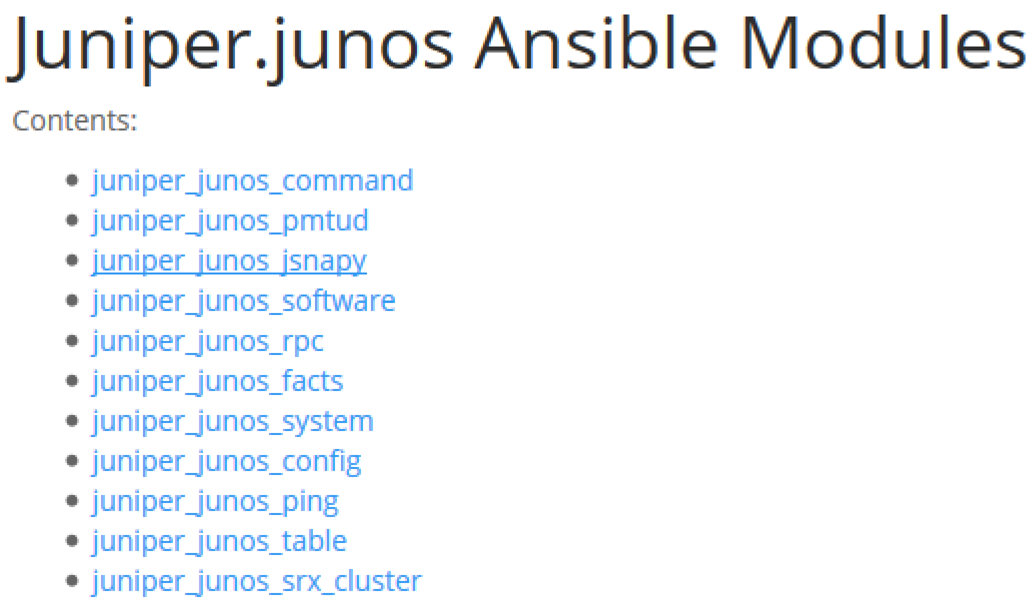
→ Check announces

BEYOND PYTHON…ANSIBLE

- Ansible is an open source automation tool
- Has a standard pool of modules to perform different tasks
- Juniper has its own modules to work with devices
- Still makes use of NETCONF over SSH
- Easy to use
- Non programmers friendly
- Relies on PyEz

JUNIPER IS ANSIBLE READY!

- Run Junos commands
- Gather device facts
- Upgrade devices
- Create PyEz tables
- Run Jsnapy tests
- Configure devices



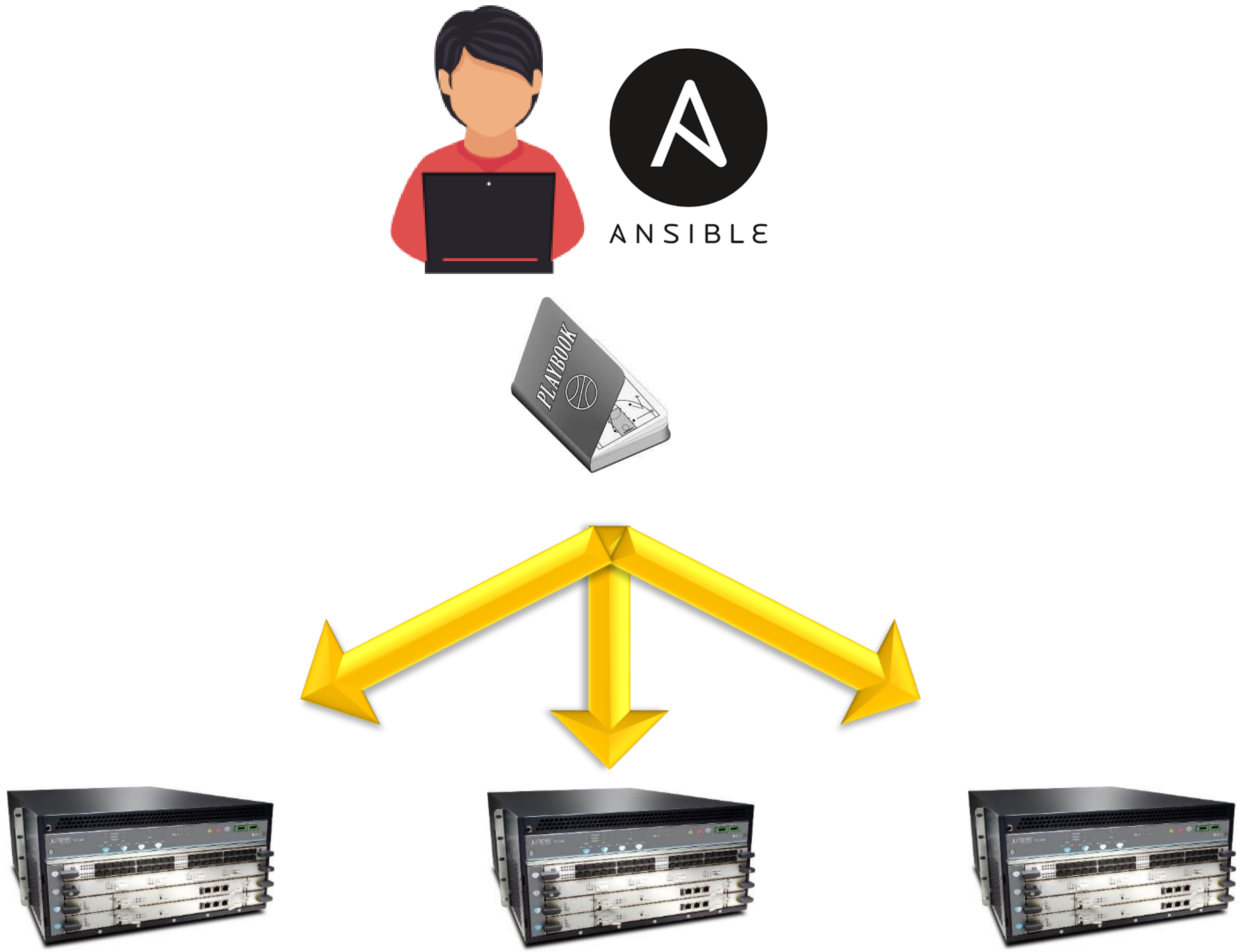
Juniper.junos Ansible Modules

Contents:

- juniper_junos_command
- juniper_junos_pmtud
- juniper_junos_jsnapy
- juniper_junos_software
- juniper_junos_rpc
- juniper_junos_facts
- juniper_junos_system
- juniper_junos_config
- juniper_junos_ping
- juniper_junos_table
- juniper_junos_srx_cluster

ANSIBLE WORKFLOWS

- User remotely runs an Ansible playbook
- A playbook contains all the tasks that must be performed on the devices
- Tasks can be run on a limited set of devices
- Ansible automatically takes care of parallelization and error handling



HOW DOES A PLAYBOOK LOOK LIKE?

```
---
- name: Retrieve data from a Junos device using a PyEZ table/view.
  hosts: all_devices
  connection: local
  gather_facts: no
  roles:
    - Juniper.junos

  tasks:
    - name: Retrieve Chassis Information Using PyEZ-included Table
      juniper_junos_table:
        host: "{{ junos_host }}"
        user: "{{ ADMUSER }}"
        passwd: "{{ ADMPASS }}"
        file: "chassis_inv.yaml"
        path: "./tables/"
      register: response
    - name: Print response
      debug:
        var: response['resource'][0]
    - name: ensure file exists
      copy:
        dest: "{{playbook_dir}}/output/{{inventory_hostname}}.txt"
        force: no
        mode: 0555
        content: |
          {{ response['resource'] | to_nice_json }}
```

```
ChassisInv:
 rpc: get-chassis-inventory
 item: chassis/chassis-module
 key: name
 view: chassisModulesView

ChassisInView:
 fields:
  name: name
  description: description
  sn: serial-number
```

JSON

everywhere