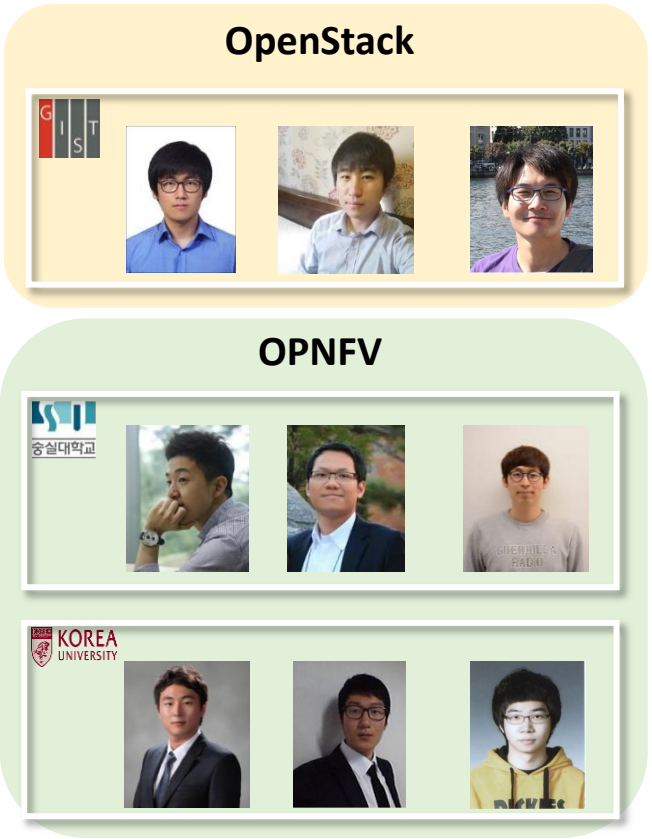
K-ONE: OpenStack/OPNFV에 기반한 오픈소스 네트워킹 요소 소프트웨어 묶음



K-ONE

K-ONE OpenStack & OPNFV Team

GIST NetCS Lab.
숭실대학교 DCN Lab.
고려대학교 MNC Lab.

openstack CLOUD SOFTWARE OPNFV

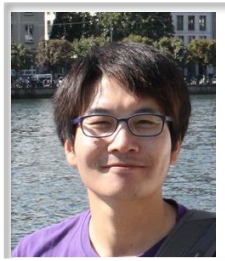
K-Cluster Early Prototype

Jun-Sik Shin (jsshin@smartx.kr)

1st semester of Ph.D. Course
Networked Computing Systems Lab.
Gwangju Institute of Science and Technology

Jungsu Han (jshan@smartx.kr)

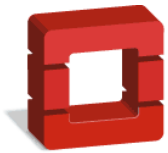
1st semester of Ph.D. Course
Networked Computing Systems Lab.
Gwangju Institute of Science and Technology



Namgon Lucas Kim (namgon@smartx.kr)

8th semester of Ph.D. Course
Networked Computing Systems Lab.
Gwangju Institute of Science and Technology

GIST Team



openstack™
CLOUD SOFTWARE

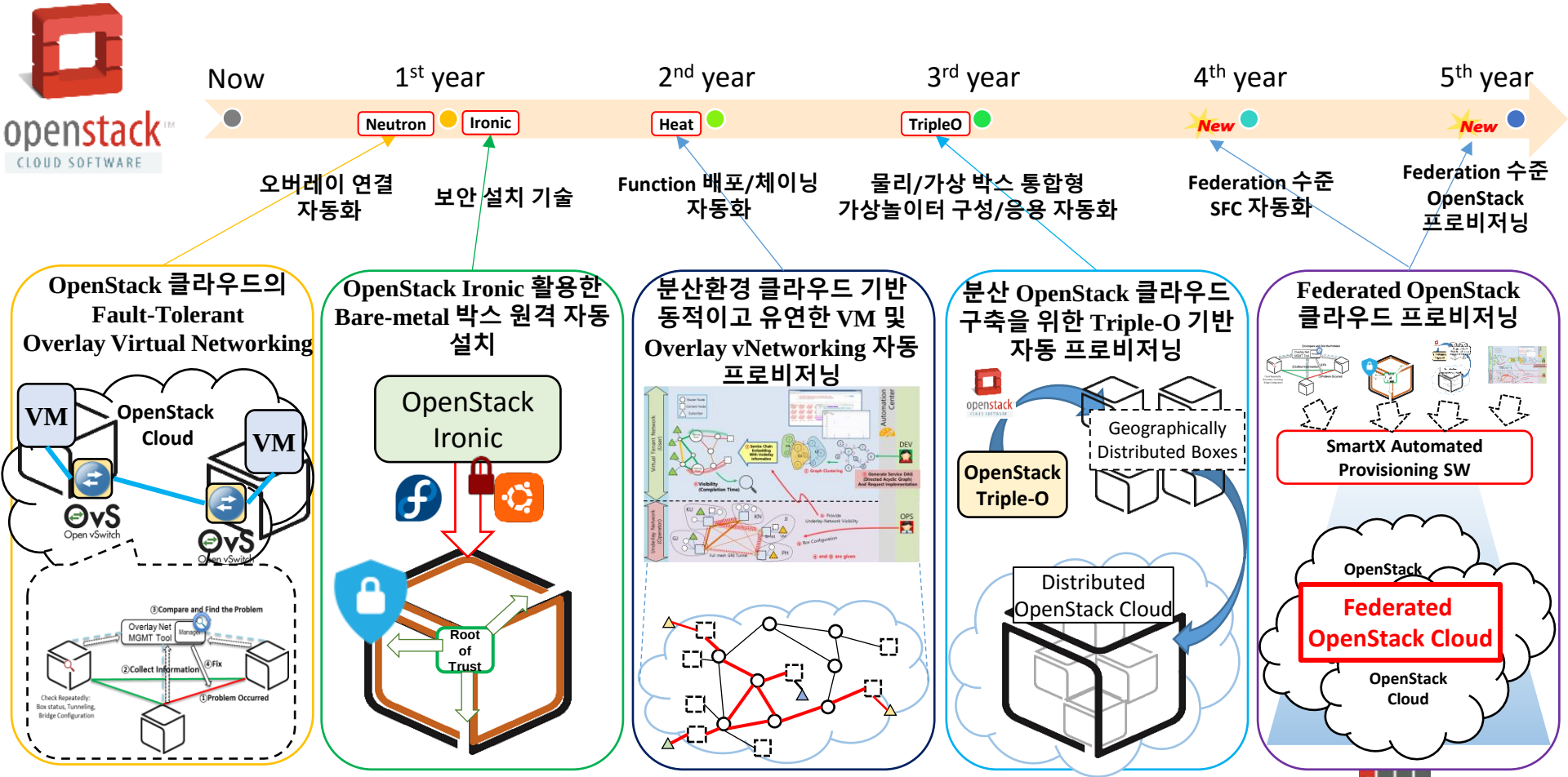
Assisted by
Jeongju Bae, Seungryong Kim, Jungi Lee
of NetCS Lab.

Contents

- K-ONE OpenStack Task Roadmap of GIST Team
- K-Cluster Concepts
- K-Cluster Early Prototype: Design & Configuration
- K-Cluster Early Prototype Demo Scenario

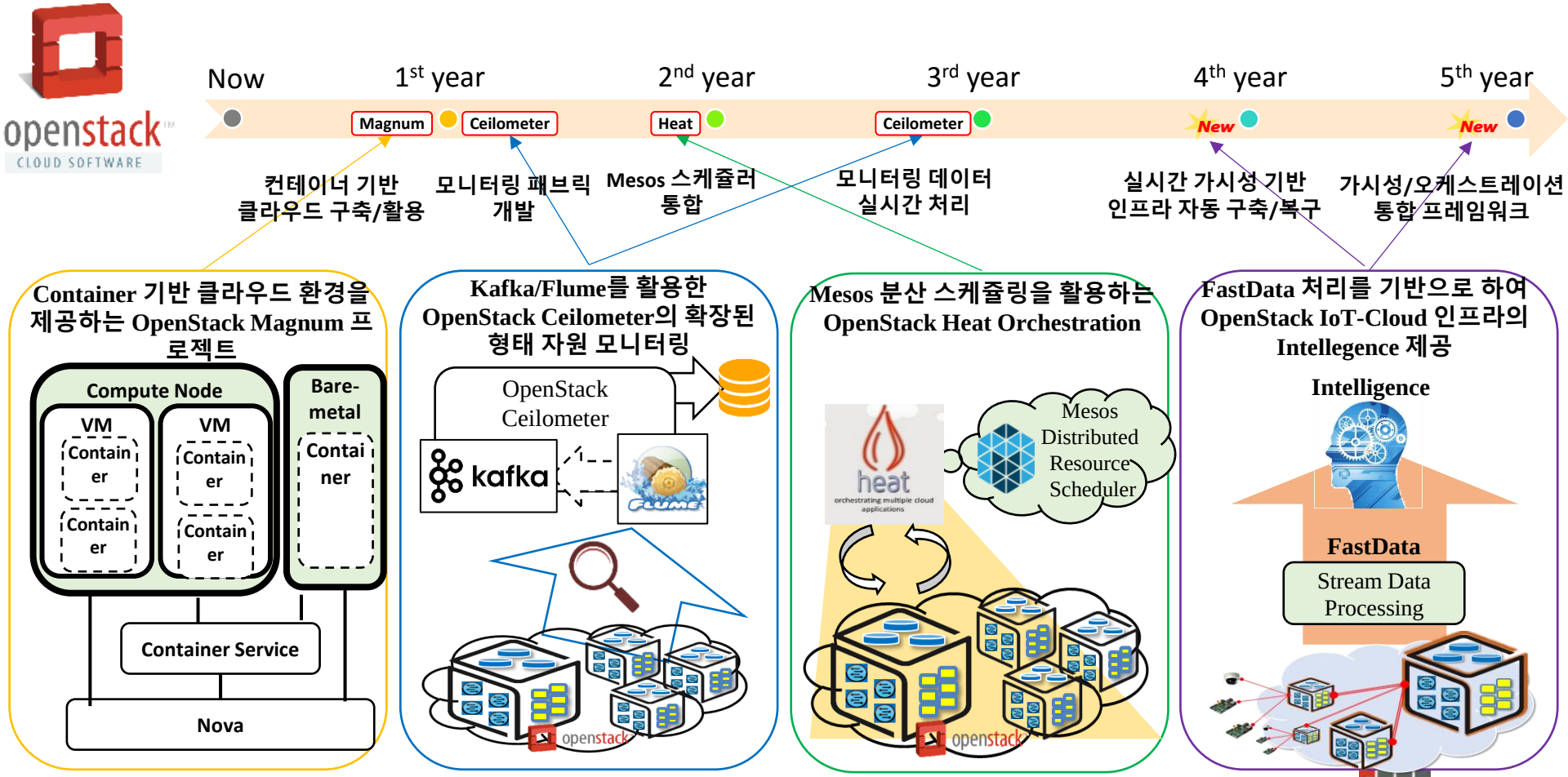
K-ONE OpenStack Task #1 Provisioning Roadmap

- 목표: OpenStack 기반 분산 클라우드, 나아가 Federated 클라우드를 대상으로 Box/Functions/Inter-connect 관점을 아우르는 자동화된 프로비저닝 소프트웨어를 개발

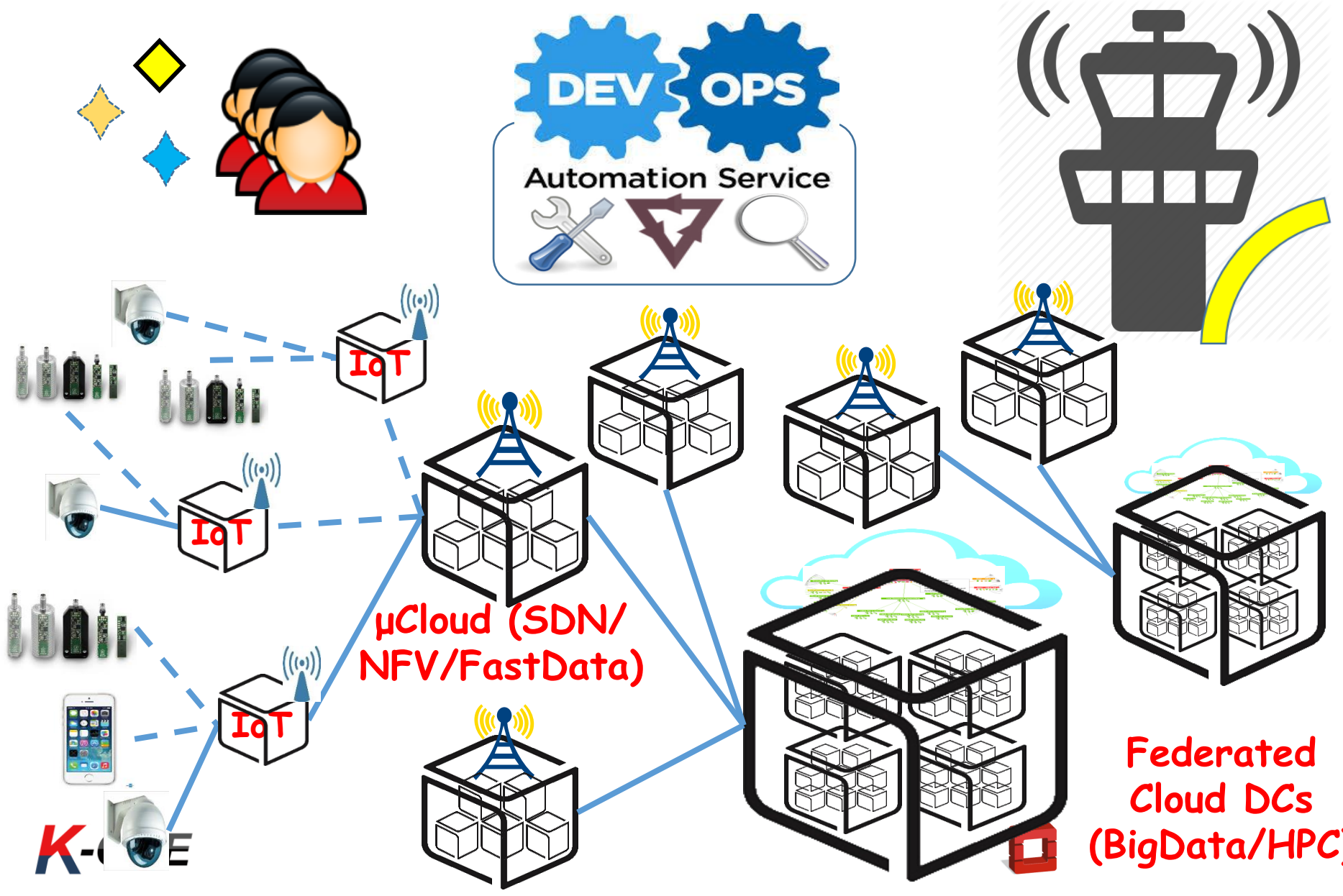


K-ONE OpenStack Task #2 Orchestration/Visibility Roadmap

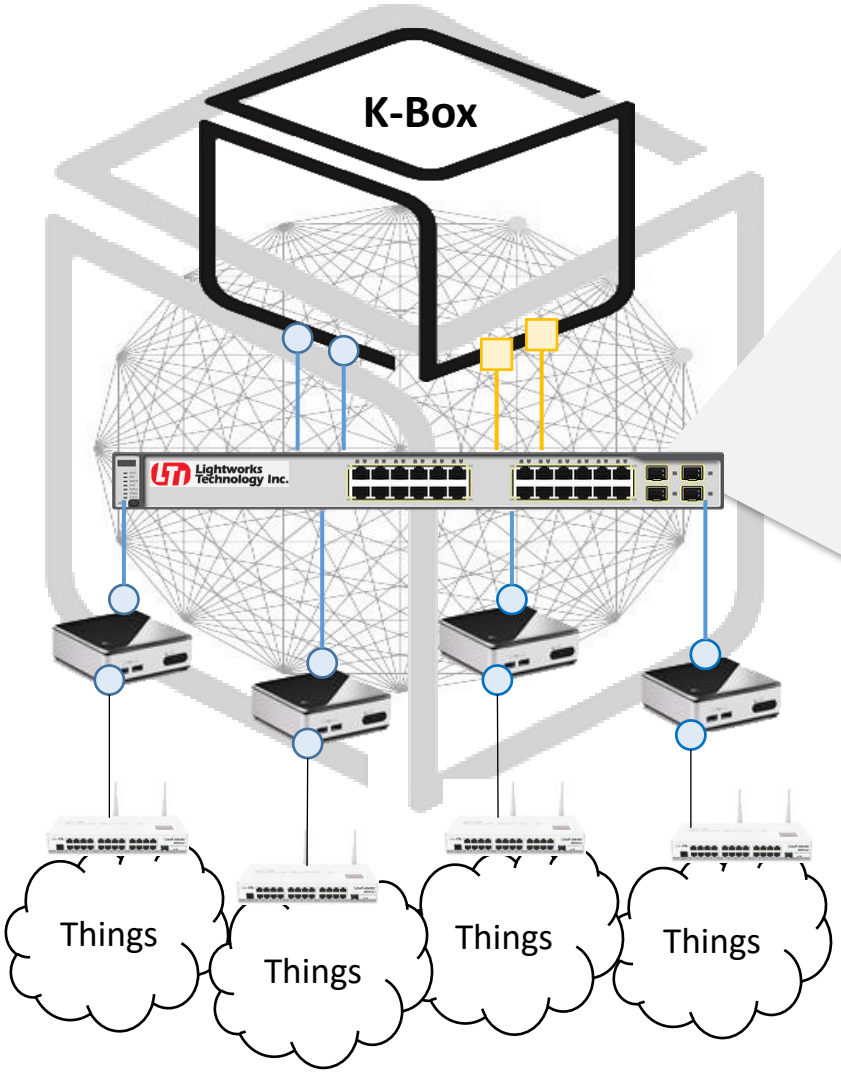
- 목표: Intelligent 한 IoT-Cloud 인프라 활용을 위하여 오픈스택 클라우드의 다계층 Visibility와 Orchestration을 제공하는 SmartX Orchestration/Visibility 도구 개발

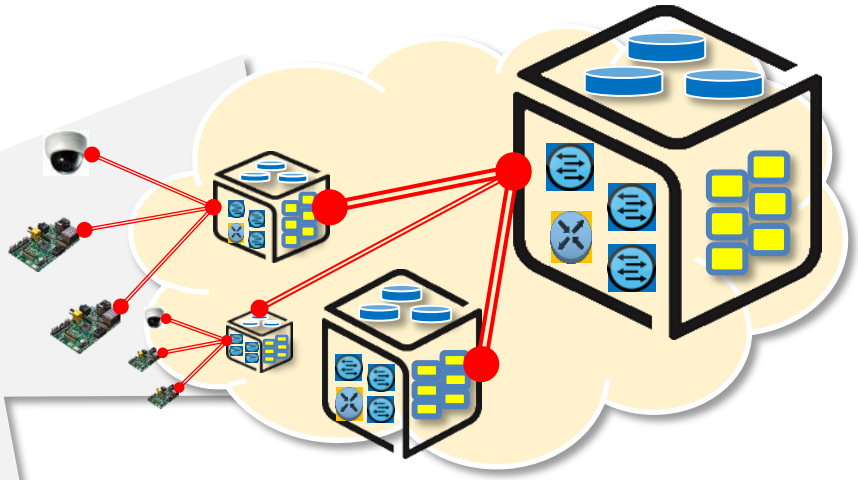


Convergent Software-defined Infrastructure



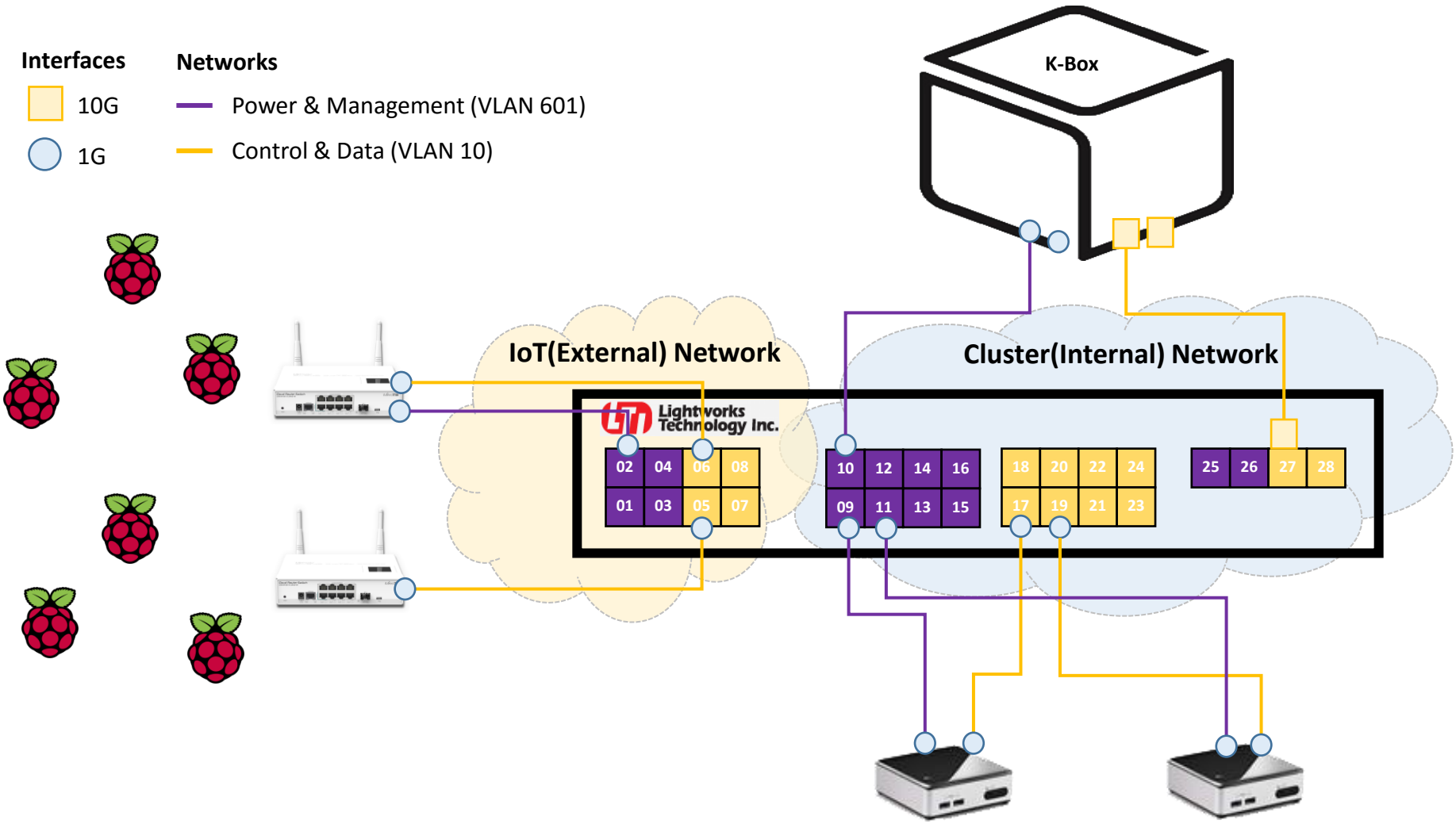
What & Why K-Cluster?



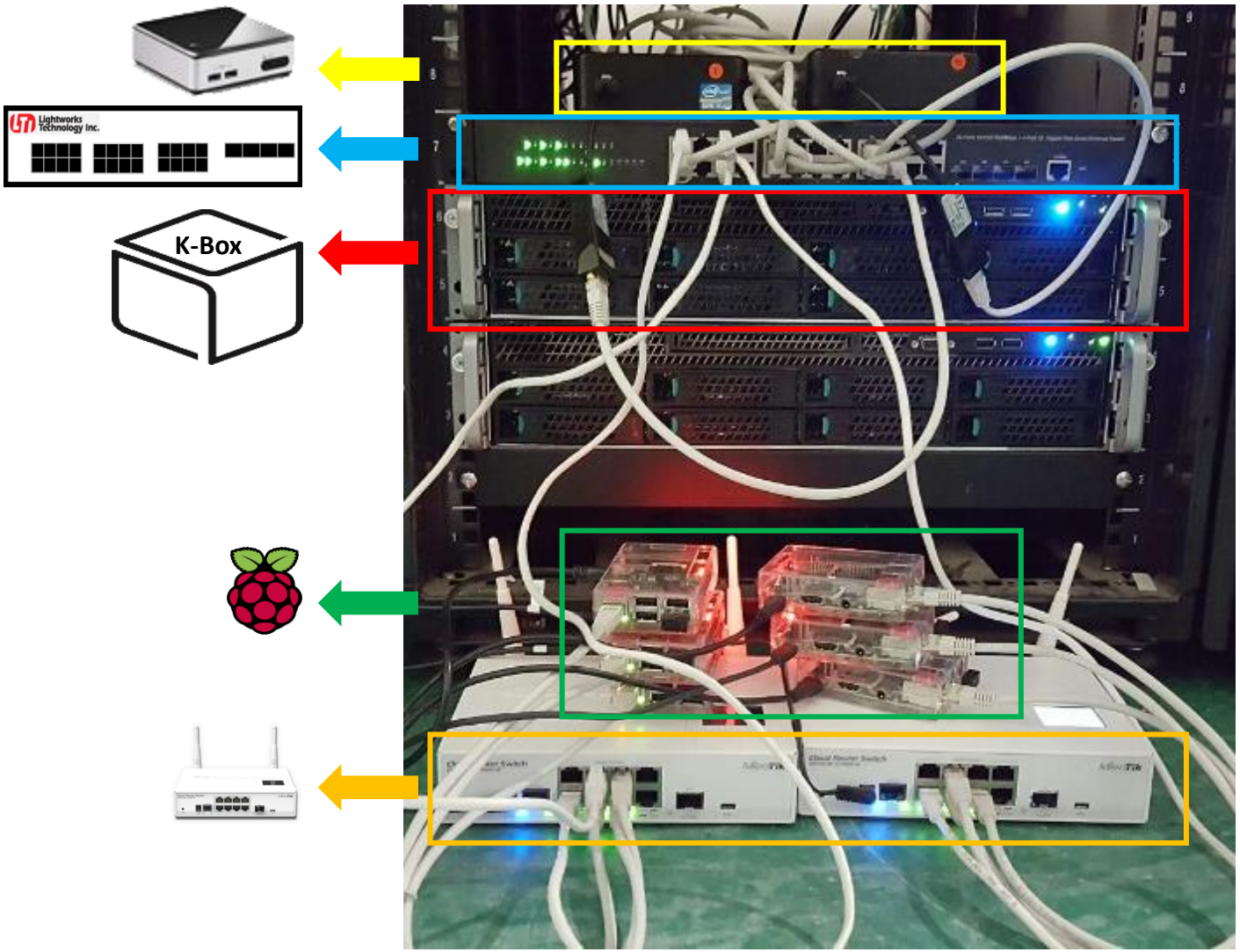


- Edge/Front-end Computing Concept
- Reference Model of µCloud Cluster
  - Cheap & Easy
- Playground for…
  - IoT-µCloud
  - SDN
  - NFV
- Automated Operation with SmartX DevOps Tower
  - SmartX Provisioning/Visibility/Orchestration/ Intelligence Center

K-Cluster Early Prototype: Hardware Configuration



K-Cluster Early Prototype: Actual Picture



Cheap

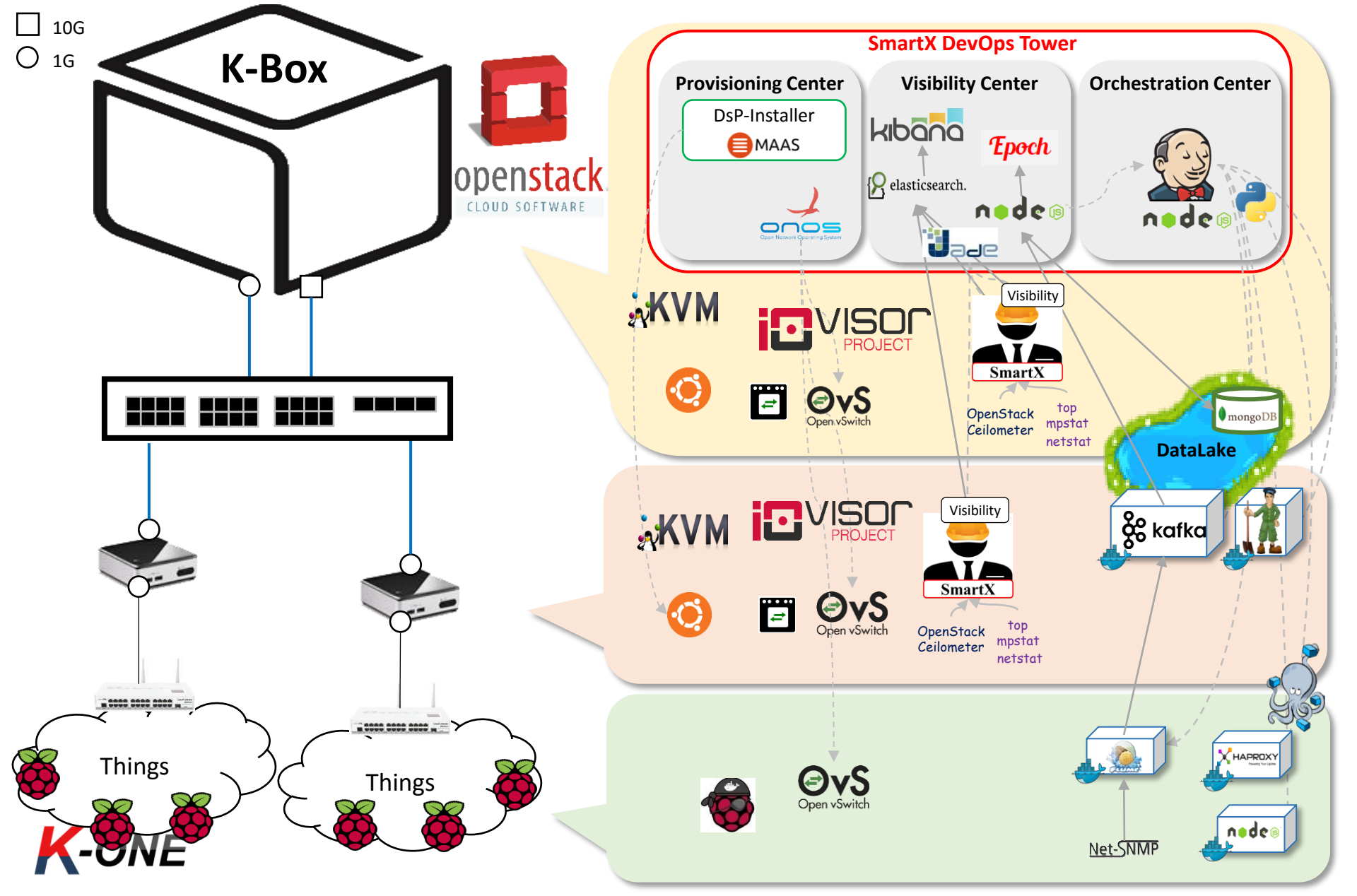
&

Small

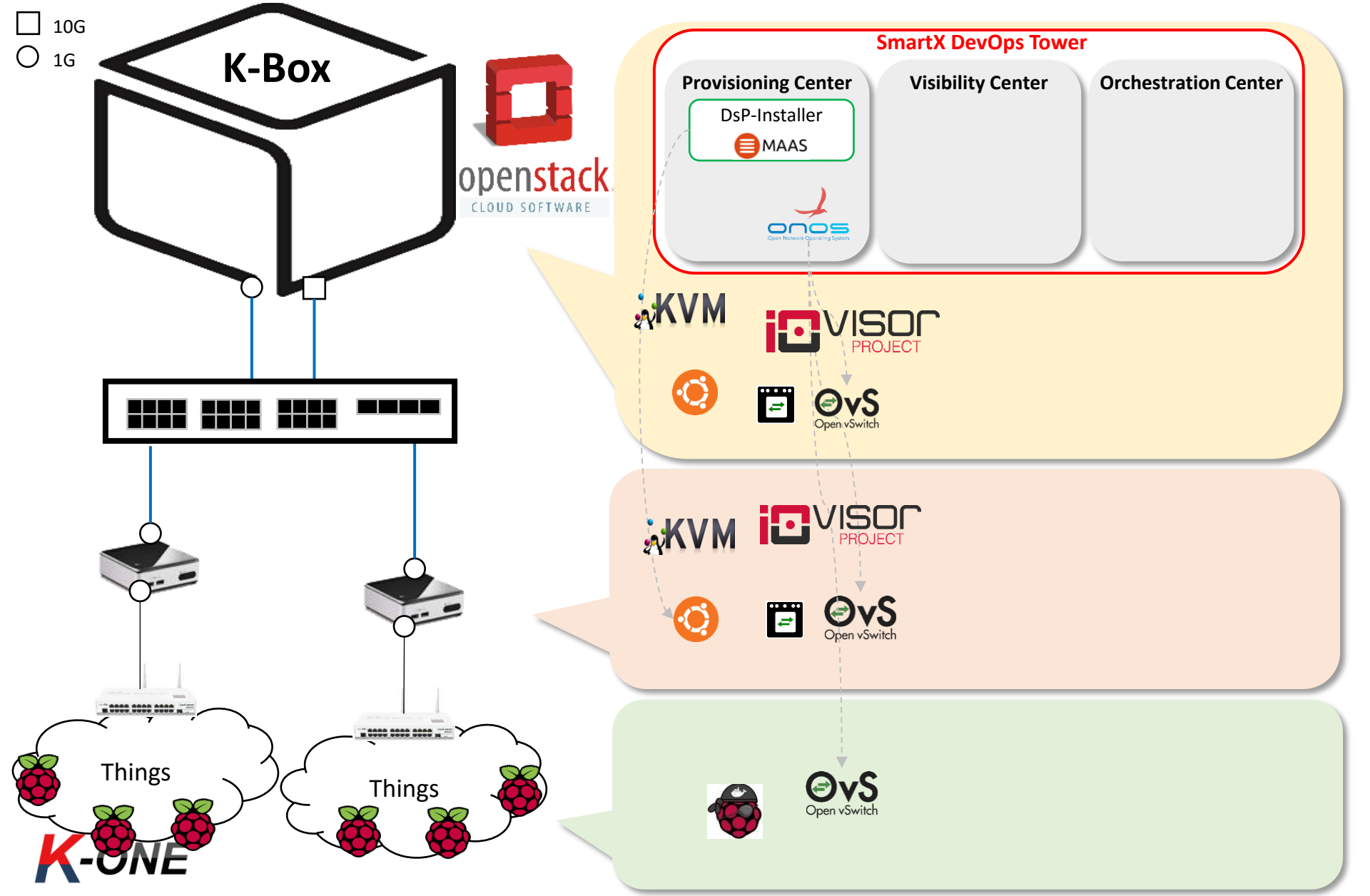
&

Easy

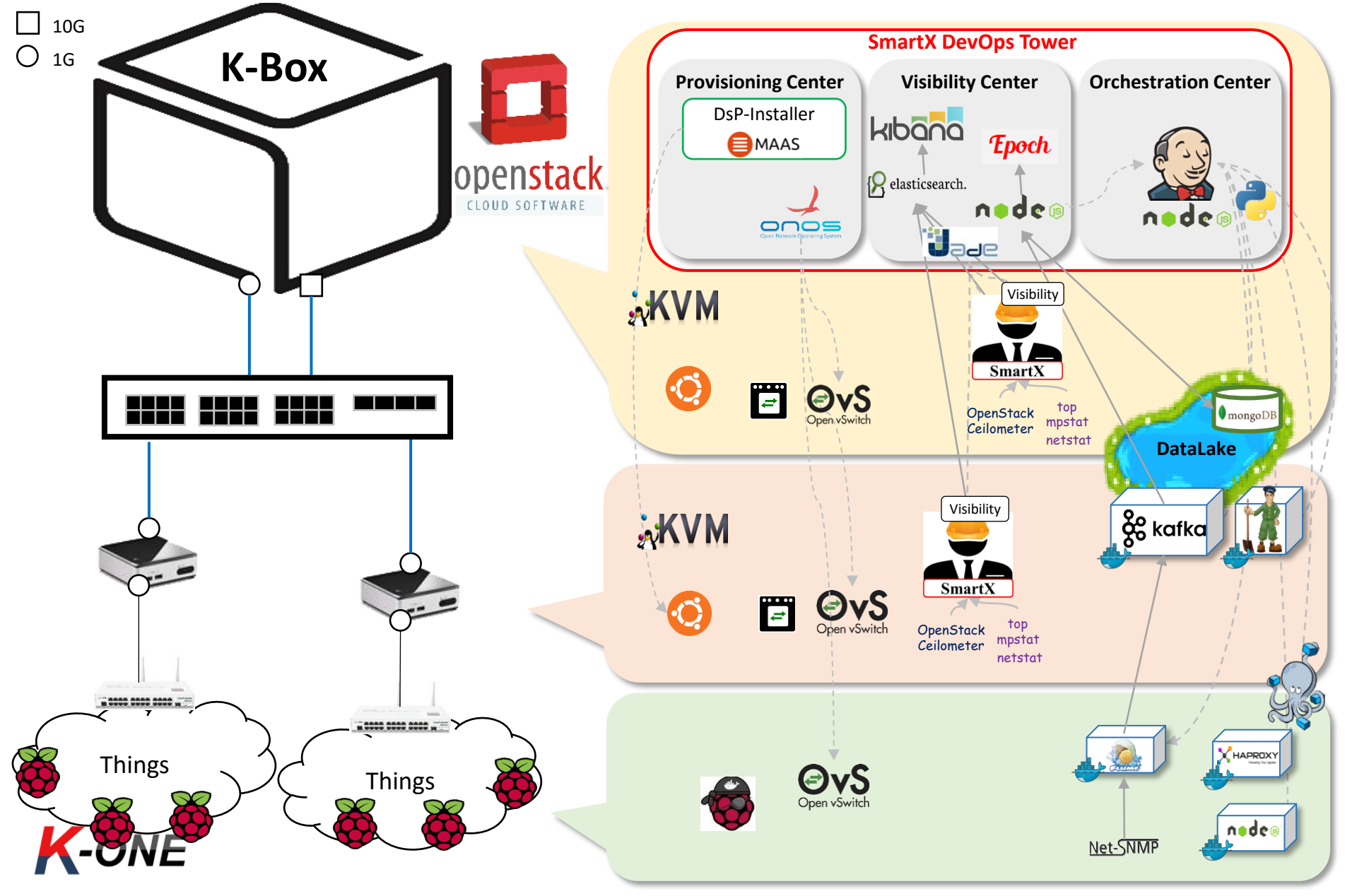
K-Cluster Early Prototype: All Software



K-Cluster Early Prototype: Software for SDN/NFV



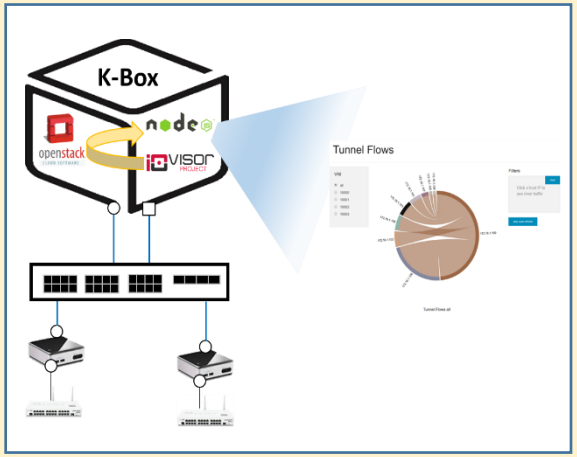
K-Cluster Early Prototype: Software for IoT-Cloud



K-Cluster Early Prototype Demo Scenario Summary

Scenario #1

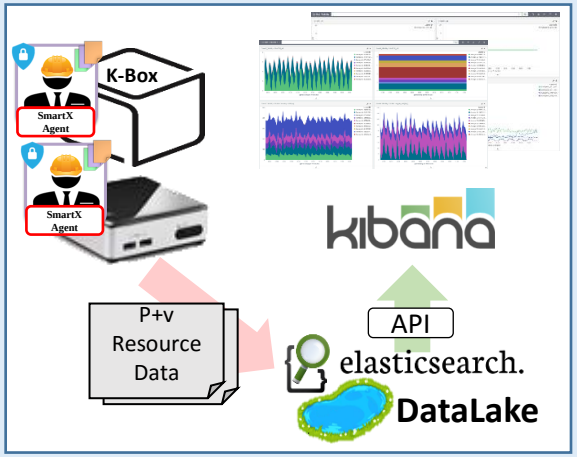
IO Visor-based Tunnel Monitoring



- **Monitor VXLAN Tunnels inter-connecting OpenStack Boxes by using IO Visor**
- IO Visor VXLAN Monitoring
  1. Filter VXLAN packets from all Packets
  2. Save the packet data into JSON format
  3. Visualize VXLAN tunnels on node.JS

Scenario #2

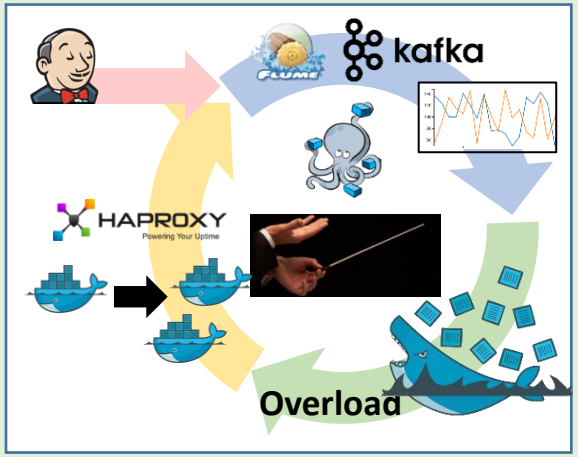
Ops Visibility for K-Box & NUCs via SmartX Agent



- **(p+v) Resource Visibility over OpenStack Cloud**
  - pResource (NUC Boxes)
  - vResource (VMs inside NUC)
- Collecting Data
  - OpenStack Ceilometer (virtual)
  - top, mpstat (physical)
  - netstat (physical)
- Storing Data
  - Elasticsearch
- Data Visualization
  - Kibana

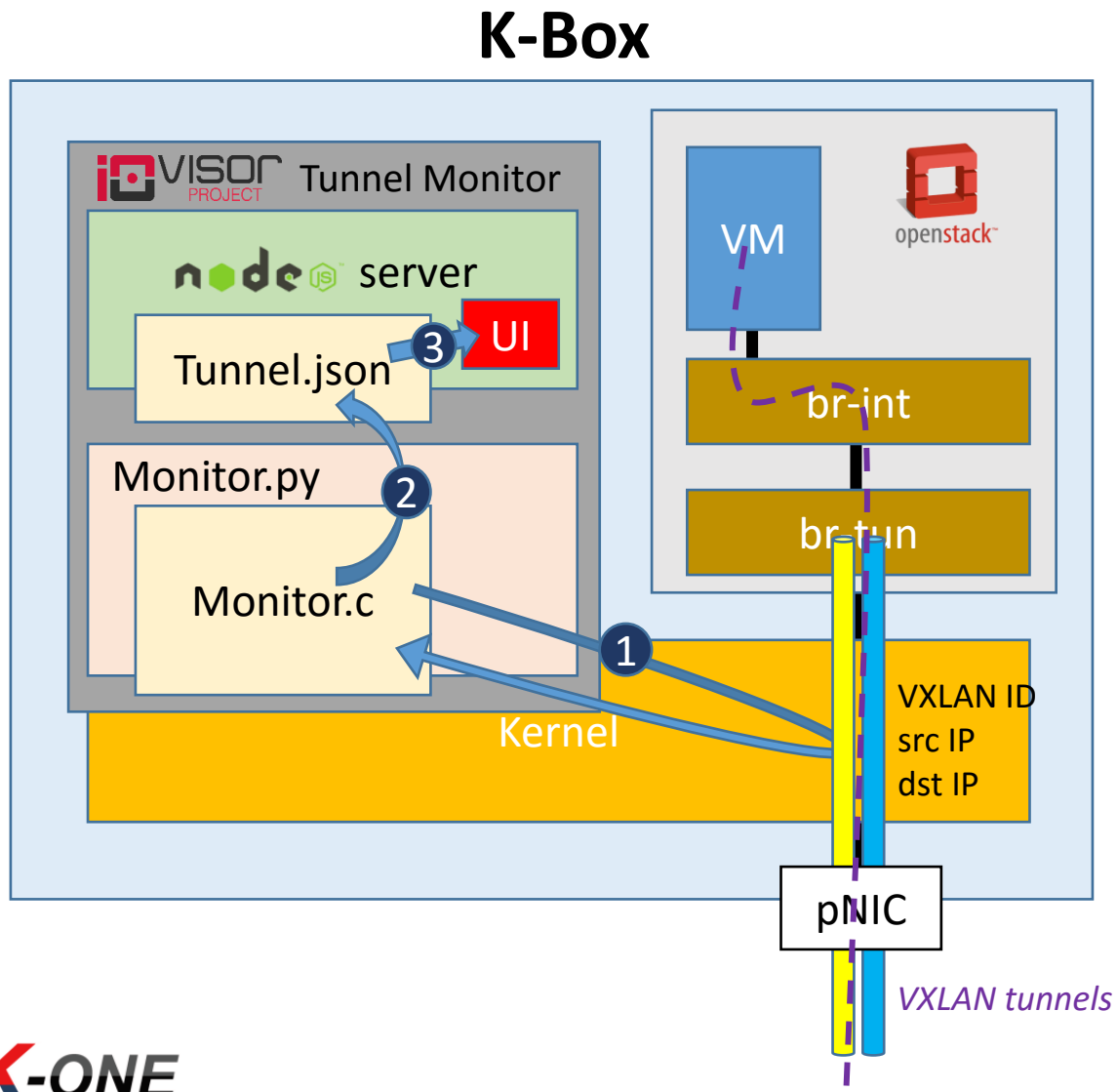
Scenario #3

Ops Visibility for RPIs via Kafka & Automated Deployment

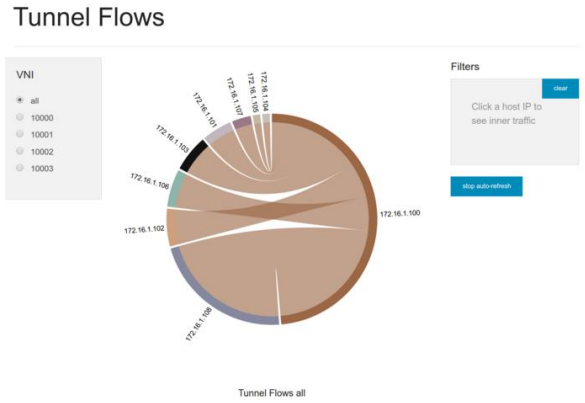


- **Automated service management with Operation Data**
- Service deployment
  - Deploying dockerized services
- Operation data monitoring
  - Collecting resource status of RPi2
  - Charting this data in nodejs web server
  - Alerting overload for service management
- Service management
  - Scaling for load balancing

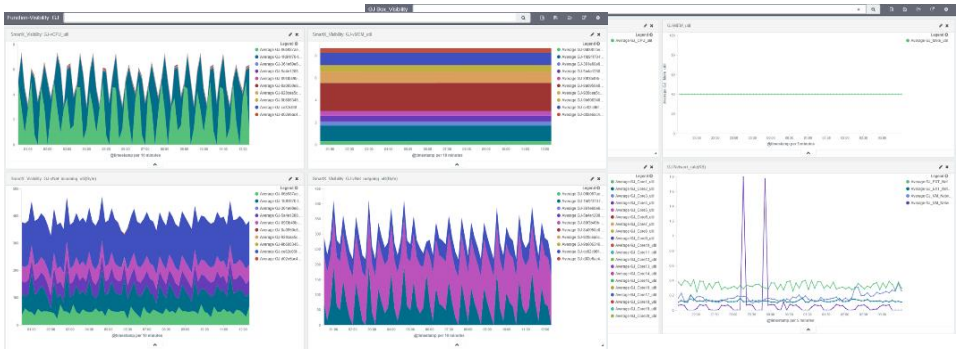
K-Cluster Early Prototype Demo Scenario #1: IO Visor-based VXLAN Tunnel Monitoring



1. Trace vxlan tunnels of OpenStack
2. Save the data into json file
3. Show in web UI



K-Cluster Early Prototype Demo Scenario #2: Ops Visibility for K-Box & NUCs via SmartX Agent



K-Box

elasticsearch.

SmartX Visibility Center

kibana

Visualization

Agent Deployment

Agent Management

Store Resource Data

- CPU
- Memory
- Network

Visibility

SmartX

Sending Resource Data

REST API

SmartX Agent

Initialization

Starting Threads Separately

Installation & Configuration

mpstat
netstat

pResource Data Parser

pResource Collector

REST API

OpenStack Ceilometer

vResource Data Parser

vResource Collector

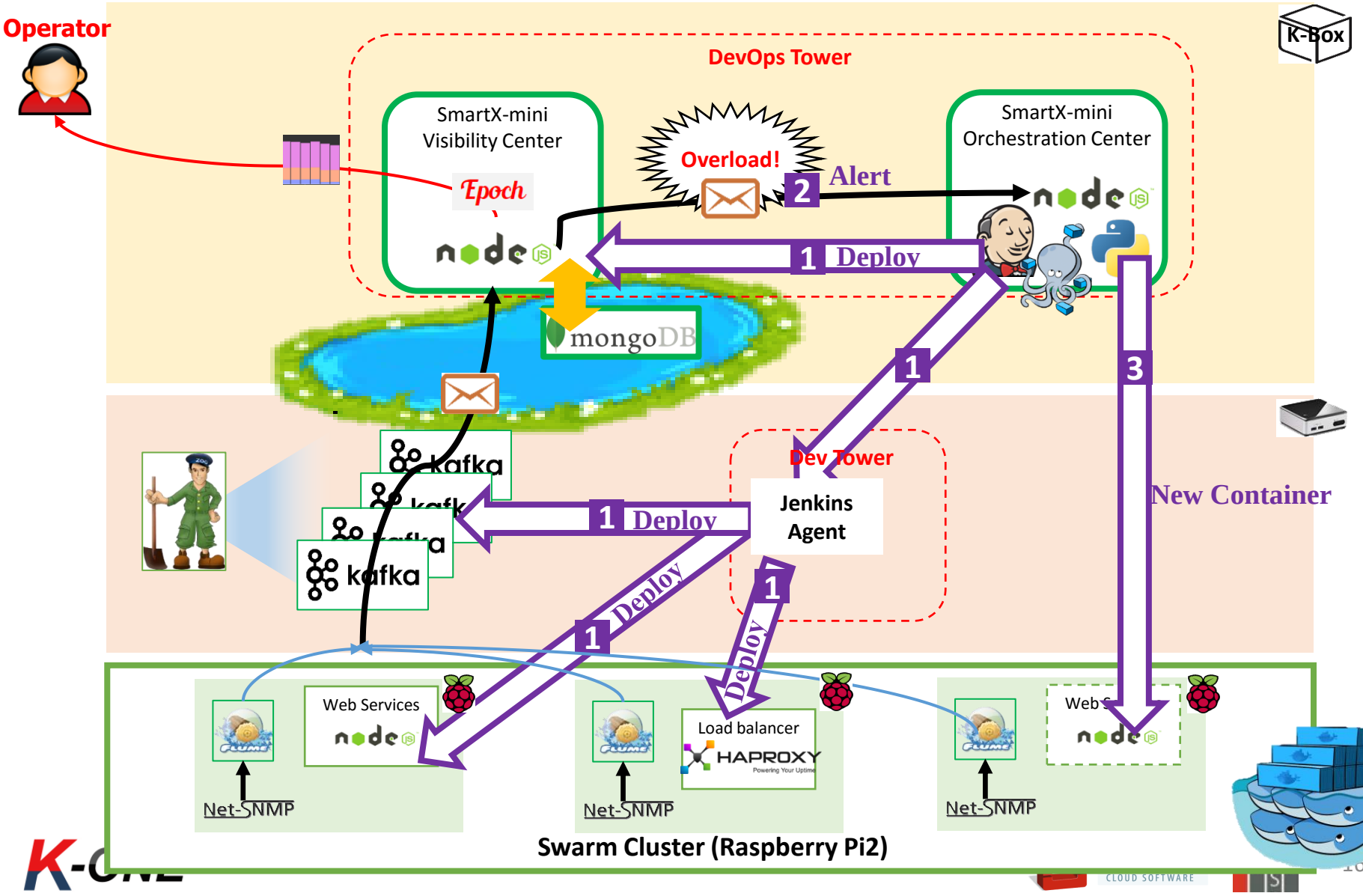
API

top

mpstat netstat

K-ONE

K-Cluster Early Prototype Demo Scenario #3: Ops Visibility for Pis via Kafka & Automated Deployment




?
Q
&
&
A
!

OpenDayLight/OPNFV 기반 Tacker에서의 계층적 서비스기능 체인화 기술

Hyunsik Yang (yangun@dcn.ssu.ac.kr)

3rd semester of Ph.D. Course
Distributed Computing Networking Lab
Soongsil University (SSU)

Do Truong Xuan (xuan@dcn.ssu.ac.kr)

6th semester of Ph.D. Course
Distributed Computing Networking Lab
Soongsil University (SSU)

Minsik Kim (kms313@ssu.ac.kr)

3rd semester of Master Course
Distributed Computing Networking Lab
Soongsil University (SSU)

숭실대학교 Team

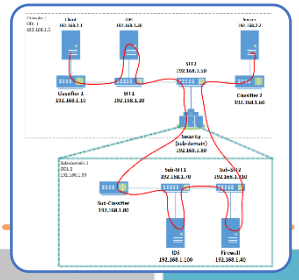
Assisted by
Vu anh Vu, Doan Van Tung
of DCN Lab.

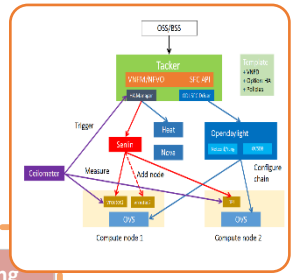
Content

- Overview – Architecture
- H-SFC based High Availability
- Automated Clustering service for NFVO
- Alarm based Monitoring driver

Overview - Architecture

① H-SFC based High Availability
② Automated Clustering service for VNF
③ Alarm based Monitoring driver





NFV Orchestrator

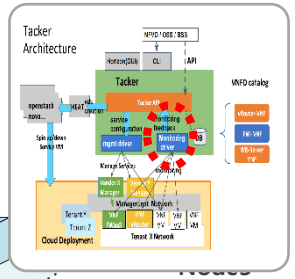
Service Function Chaining | VNF Placement | VIM Resource Check | service Template | Hierarchical SFC | Clustering Service

OPNFV VNF Manager

VNF Catalog | VNF Life-cycle management | VNF Configuration | VNF Health Monitoring | State monitoring Framework

openstack

OPEN DAYLIGHT

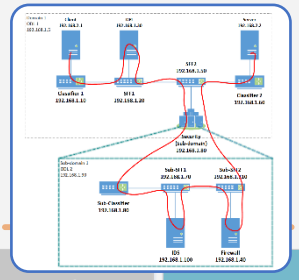


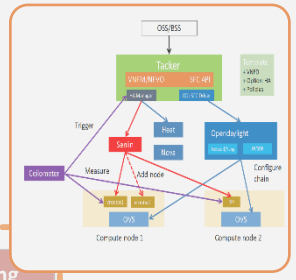
openstack

SFC1
SFC2
Node1
Node2
Node3
Site A
Multi Site
Node1
Node2
Site B
hSFC1
hSFC2

H-SFC based High Availability

① H-SFC based High Availability
② Automated Clustering service for NFVO
③ Alarm based Monitoring driver





NFV Orchestrator

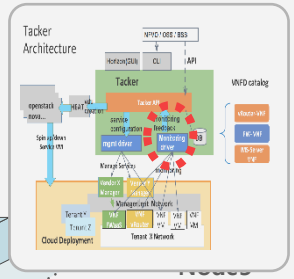
Service Function Chaining | VNF Placement | VIM Resource Check | service Template | Hierarchical SFC | Clustering Service

OPNFV VNF Manager

VNF Catalog | VNF Life-cycle management | VNF Configuration | VNF Health Monitoring | State monitoring Framework

openstack

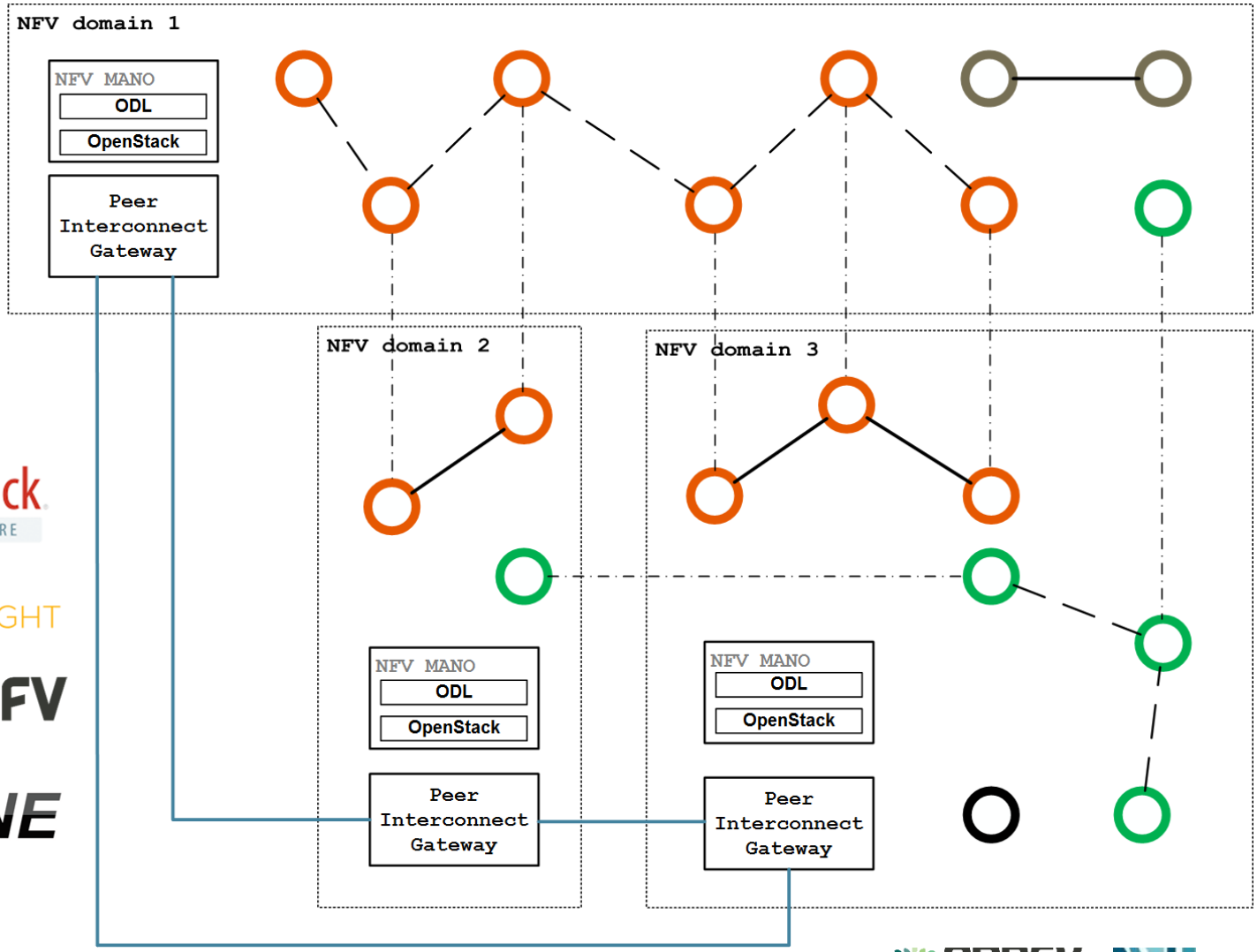
OPEN DAYLIGHT



openstack

SFC1
SFC2
Node1
Node2
Node3
Site A
Multi Site
Node1
Node2
hSFC1
hSFC2
Site B

H-SFC based High Availability-Multi domain SFC Architecture



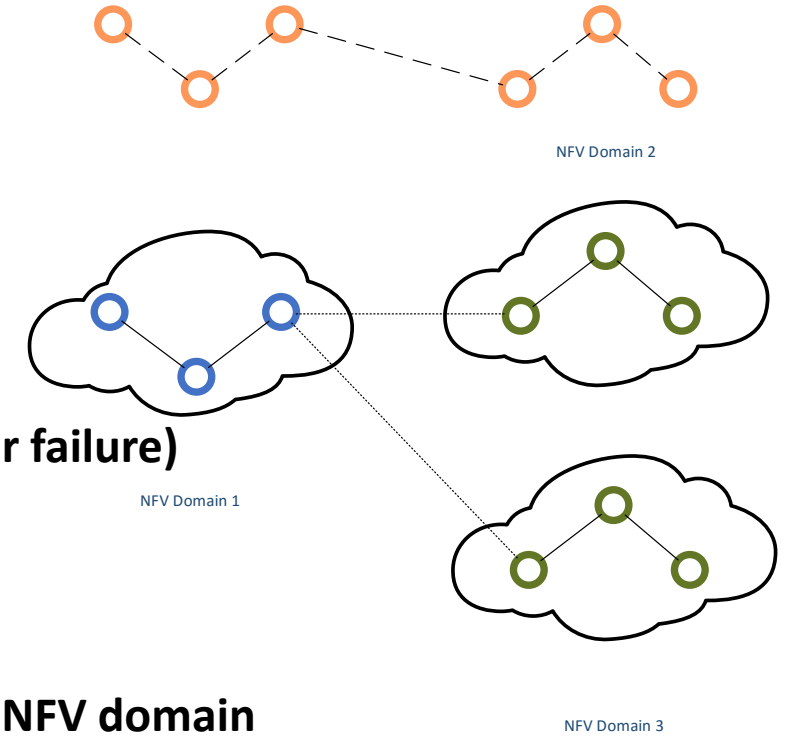
openstack CLOUD SOFTWARE

OPEN DAYLIGHT

OPNFV

K-ONE

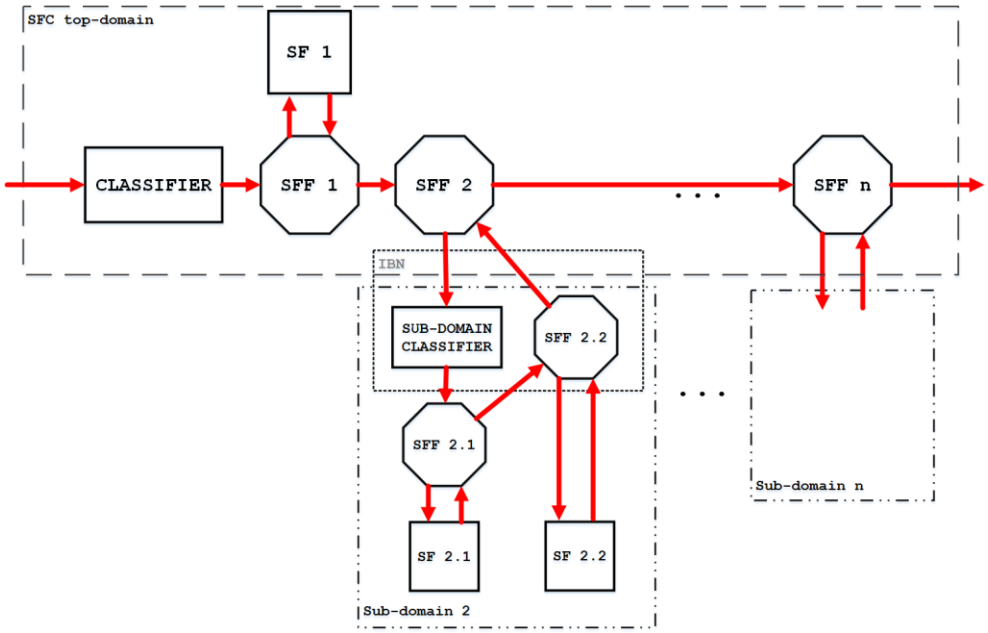
H-SFC based High Availability-Scenario



- **Decentralized Service Chain**
  - **Avoid single point of failure**
  - **Quick recover at failure (even datacenter failure)**
  - **Easy to migrate service chain**
- **Service function scaling:**
  - **Scale up/down is easily handle within a NFV domain**
  - **Scale in/out quickly by reusing service chain in other NFV domain**
- **Load balancing between NFV domain**
  - **“Service chain load balancing” instead of SF load balancing**
  - **Could apply multiple layers of load balancing**

Hierarchical SFC

- Allowing an SFC to be decomposed from a large-scale network into multiple sites
- Each domain is managed by an independent SFC manager
- Top-level service function paths carry packets from classifiers through a series of SFFs and sub-domains, with the operations within sub-domains being opaque to the higher levels



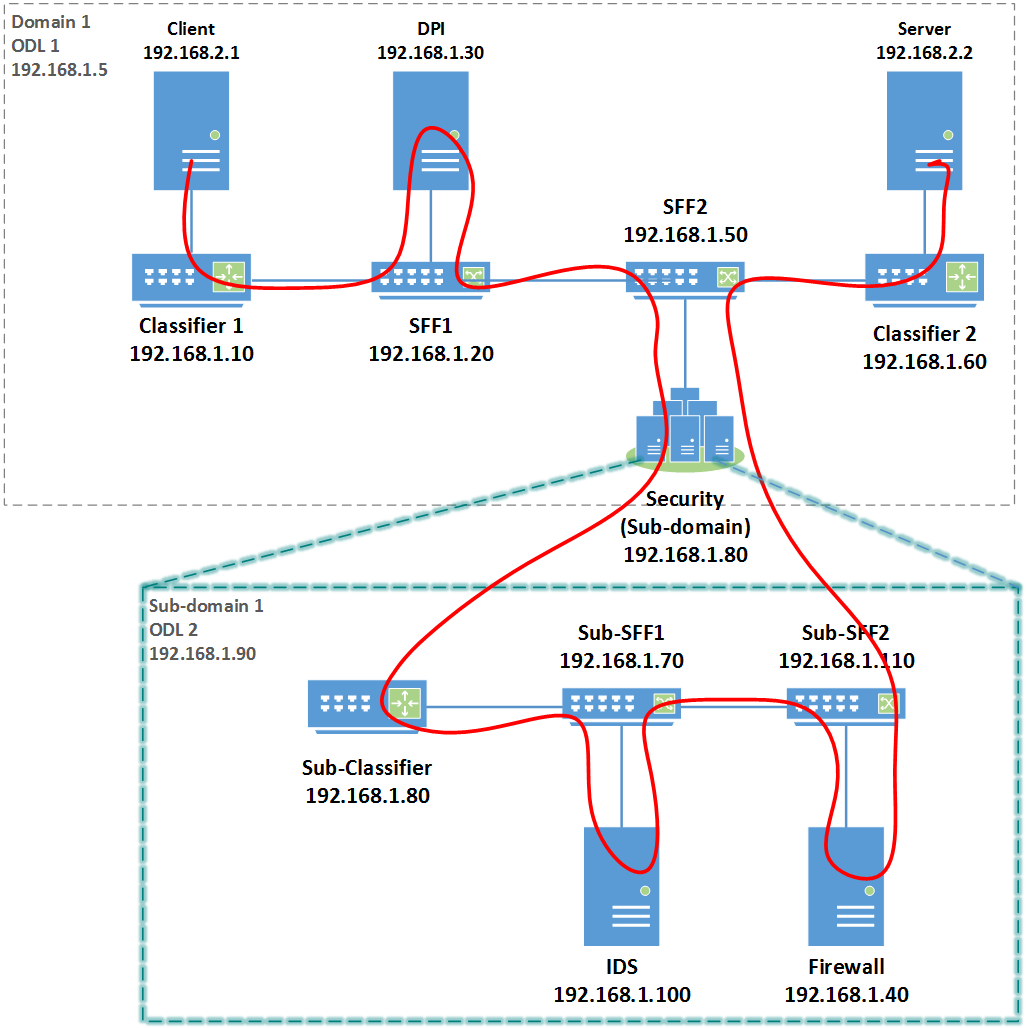
Hierarchical SFC benefits

- Enable implementing SFC across a large, geographically dispersed network comprised of millions of hosts and thousands of network forwarding elements, involving multiple operational teams (with varying functional responsibilities)

- **Simplify SFC managing and orchestrating** by decomposing complex SFC system in a nested manner

- **Simplify the mechanisms of scaling in and scaling out service functions**
  - All of the complexities of load-balancing among multiple SFs can be handled within a sub-domain, under control of the classifier
  - Allowing the higher-level domain to be oblivious to the existence of multiple SF instances

H-SFC based High Availability-Testbed Environment

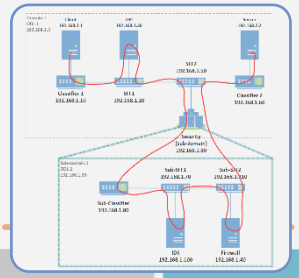


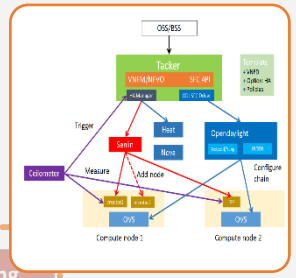
H-SFC based High Availability-Testbed Environment

- Top-domain Chain: Incoming → DPI → Firewall → Server
- Top-domain path: Client → Classifier1 → SFF1 → DPI → SFF2 → Firewall (subdomain) → Classifier2 → Server
- Subdomain path: sub-Classifier → sub-SFF1 → Firewall
- Scenario:
  - Sending requests from client (192.168.2.1) to server (192.168.2.2)
  - Capture and analyze packet at sub-domain ingress, egress, service function, …

Automated Clustering service for NFVO

① H-SFC based High Availability
② Automated Clustering service for NFVO
③ Alarm based Monitoring driver





NFV Orchestrator

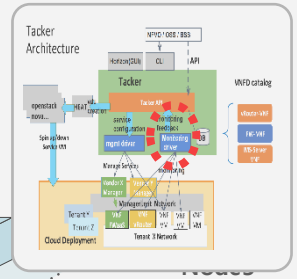
Service Function Chaining | VNF Placement | VIM Resource Check | service Template | Hierarchical SFC | Clustering Service

OPNFV VNF Manager

VNF Catalog | VNF Life-cycle management | VNF Configuration | VNF Health Monitoring | State monitoring Framework

openstack

OPEN DAYLIGHT



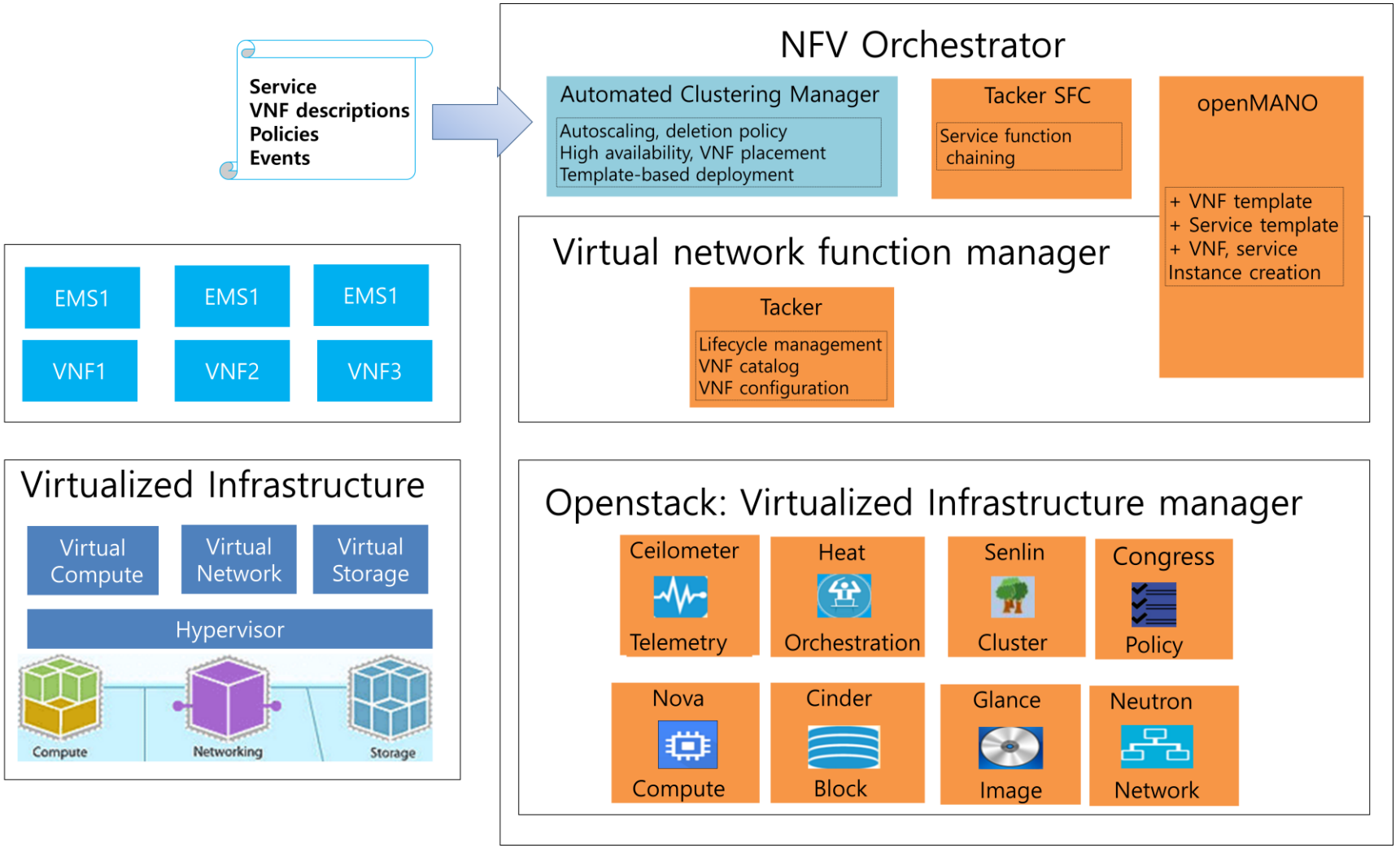
openstack

Node3
Node2
Node1
SFC1
SFC2
Site A

Multi Site

Node2
Node1
hSFC1
hSFC2
Site B

Automated Clustering service for NFVO-Introduction

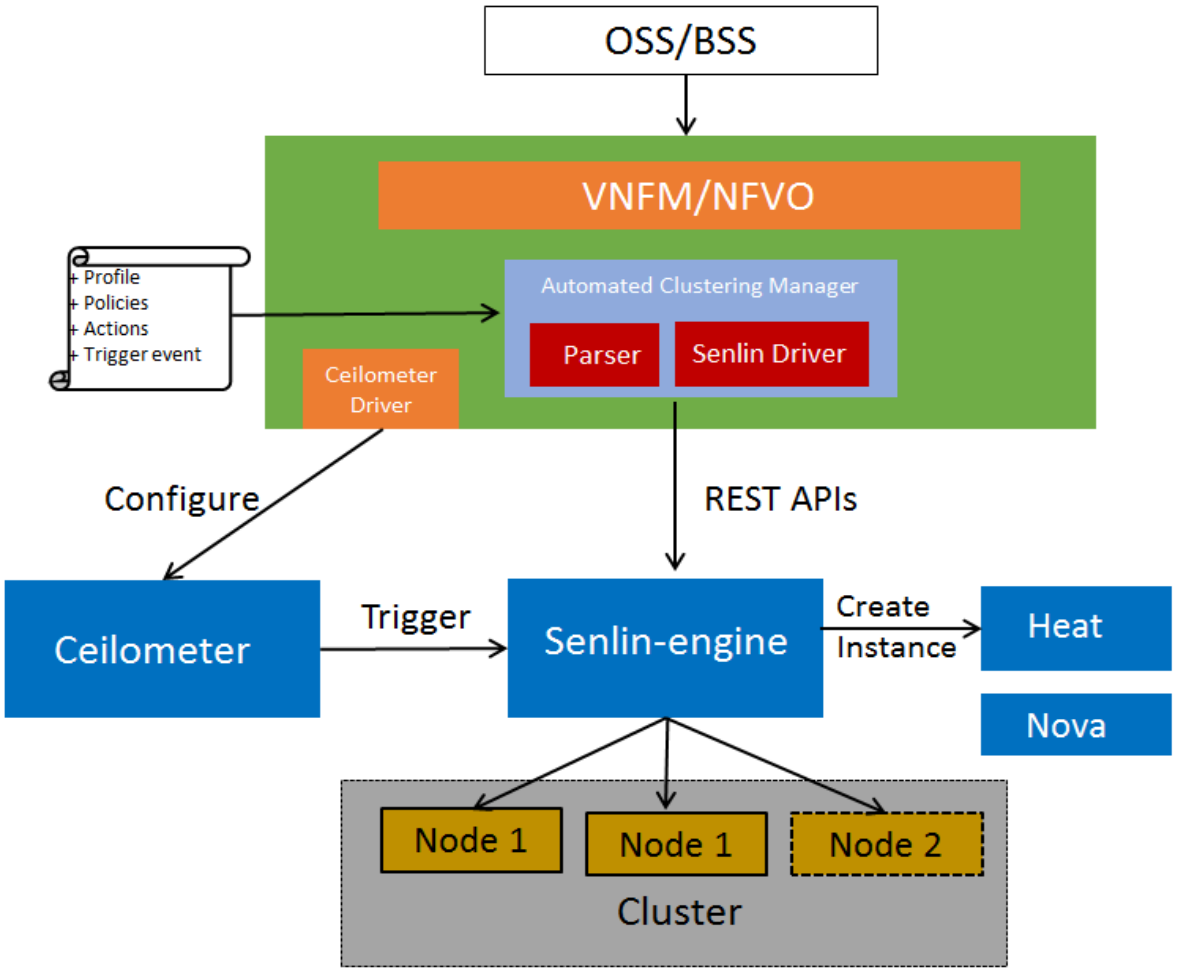


Automated Clustering service for NFVO-architecture

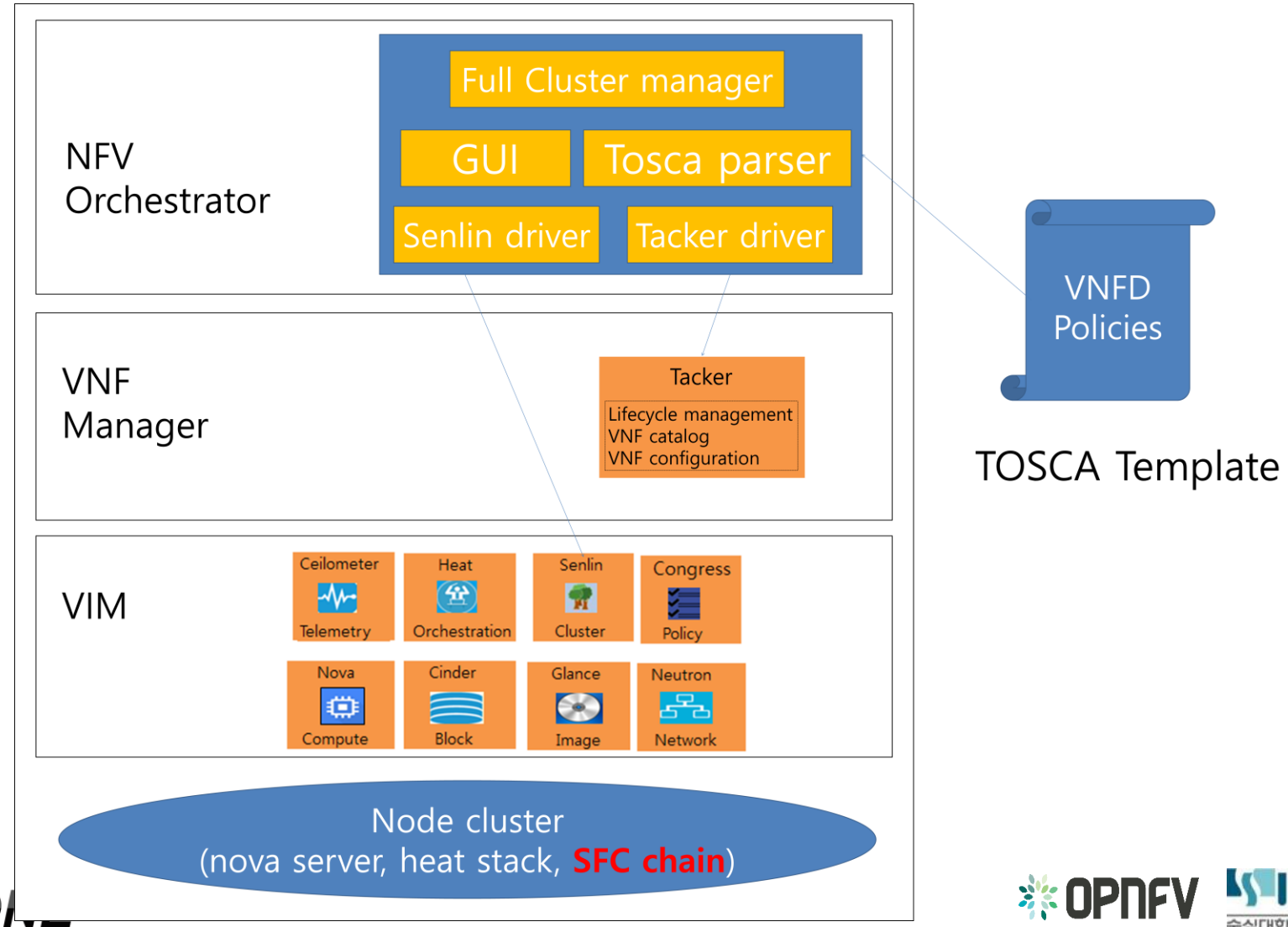
Operator

```
# template file for deploying a cluster of no

profile:
    name: test-profile
    spec: nova_server.yaml
cluster:
    name: test-cluster
    capacity: 1
    min: 1
    max: 3
policies:
    - policy1:
        name: test-policy8
        spec: scaling_policy.yaml
        enable: TRUE
    - policy2:
        name: test-policy9
        spec: deletion_policy.yaml
        enable: FALSE
triggers:
    - trigger1:
        action:
            name: test-action1
            value: CLUSTER_SCALE_OUT
        receiver:
            name: test-receiver1
    - trigger2:
        action:
            name: test-action2
            value:
        receiver:
            name: test-receiver2
```

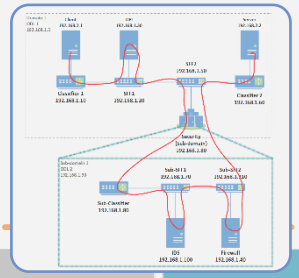


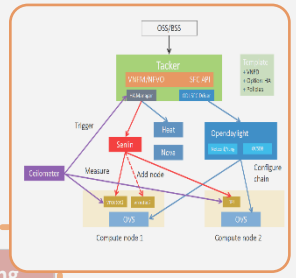
Automated Clustering service for NFVO



Alarm based Monitoring driver

① H-SFC based High Availability
② Automated Clustering service for NFVO
③ Alarm based Monitoring driver





NFV Orchestrator

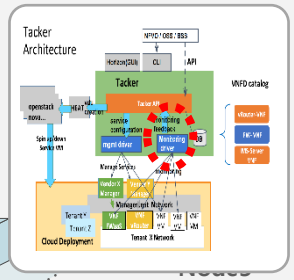
Service Function Chaining | VNF Placement | VIM Resource Check | service Template | Hierarchical SFC | Clustering Service

OPNFV VNF Manager

VNF Catalog | VNF Life-cycle management | VNF Configuration | VNF Health Monitoring | State monitoring Framework

openstack

OPEN DAYLIGHT



openstack

Node3
Node2
Node1
SFC1
SFC2
Site A

Multi Site

Node2
Node1
hSFC1
hSFC2
Site B

Alarm based Monitoring driver-Introduction

UC2: Stateful VNF with Redundancy

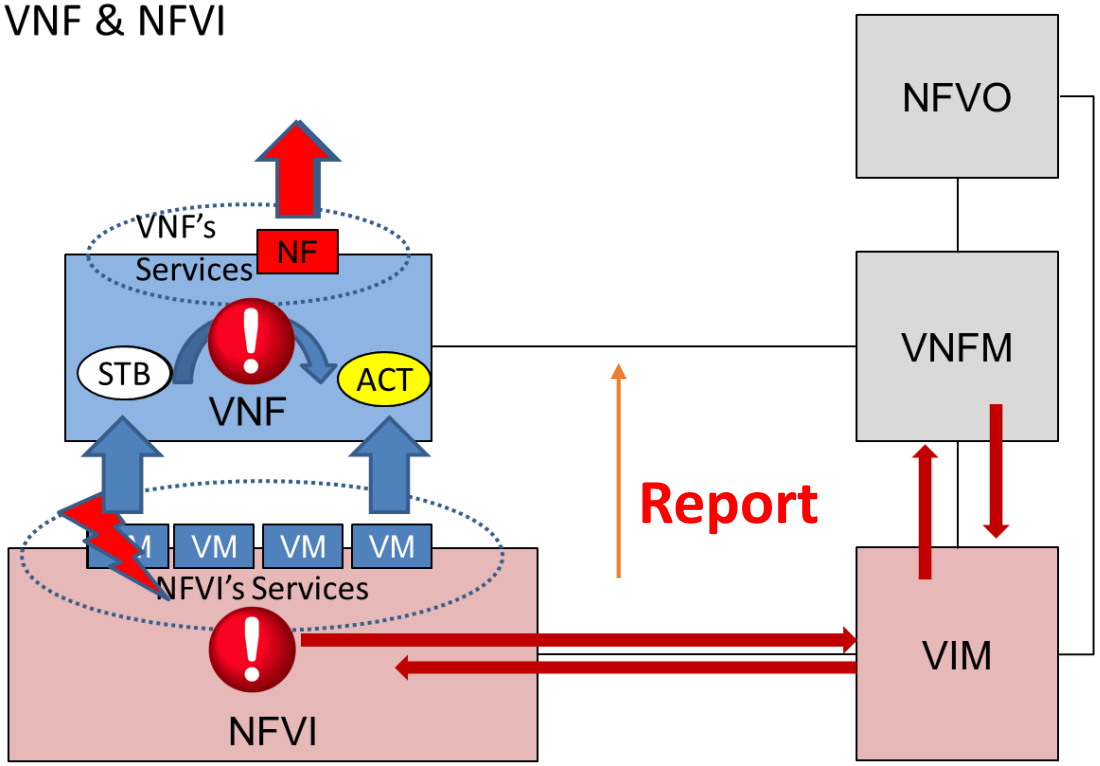
Failure detection: VNF & NFVI

Recovery time

1. VM fails
2. VM Service fails
3. VNFC fails
4. NF fails*
5a. VNF detects the failure
6a. VNF fails over
7a. NF recovers

5b. NFVI detects the failure
6b. NFVI reports to VIM
7b. VIM reports to VNFM
8. VNFM ok to VIM
9. VIM repairs VM
10. VM Service recovers

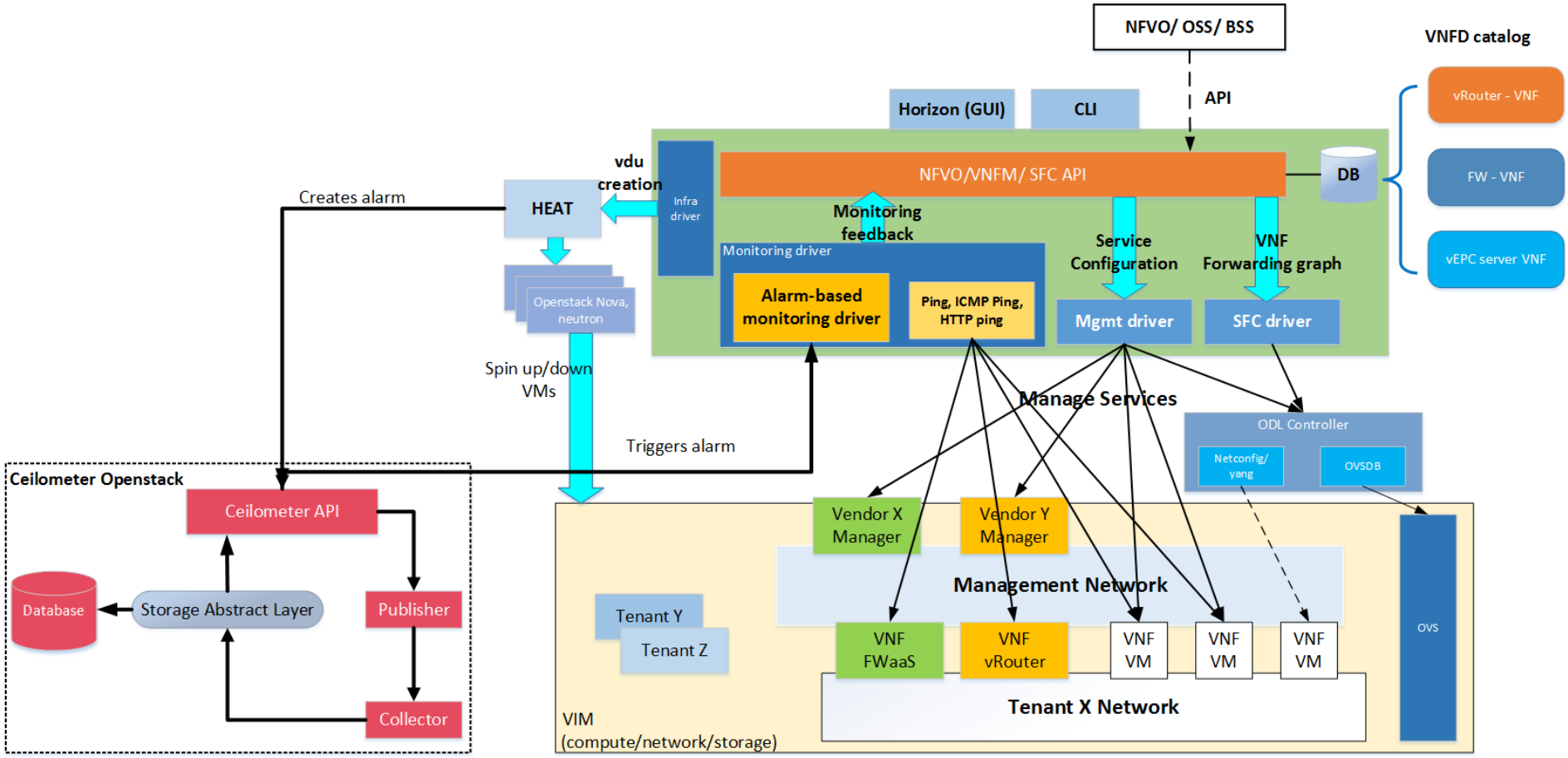
11. VNF repairs VNFC



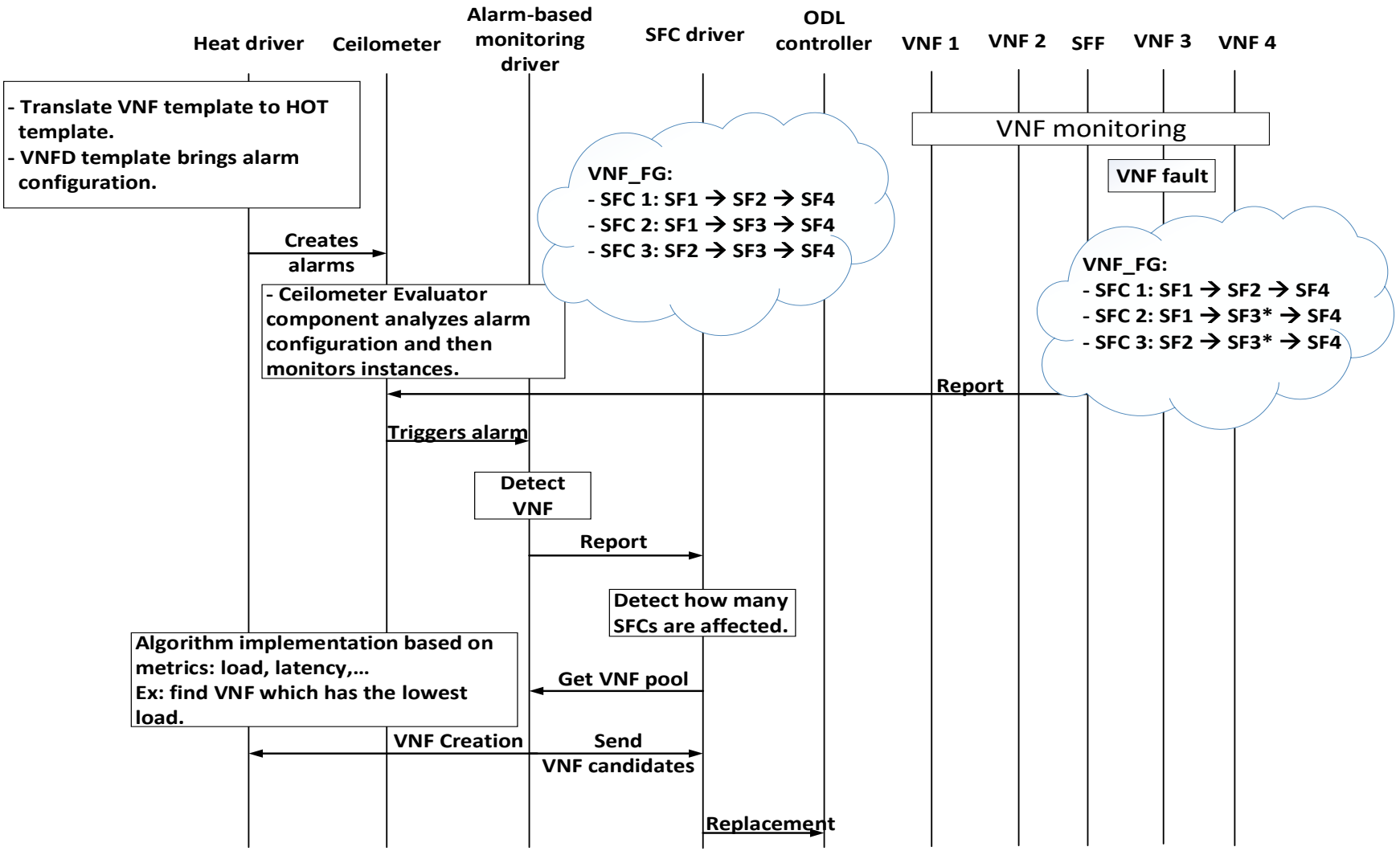
*Steps 1-4 are simultaneous they are separated for clarity

Source: HA Project

Alarm based Monitoring driver-architecture



Alarm based Monitoring driver-Pro



Alarm based Monitoring driver-VNFD

```
template_name: sample-vnfd-cms
description: demo-example

service_properties:
  Id: sample-vnfd
  vendor: tacker
  version: 1

vdus:
  vdu1:
    id: vdu1
    vm_image: cirros-0.3.4-x86_64-uec
    instance_type: m1.tiny

    network_interfaces:
      management:
        network: net_mgmt
        management: true
      pkt_in:
        network: net0
      pkt_out:
        network: net1

    placement_policy:
      availability_zone: nova

    auto-scaling: noop
    monitoring_policy:
      cpu_util:
        monitoring_params:
          statistic: avg
          period: 600
          evaluation_periods: 1
          threshold: 15
        actions:
          failure_policy: scale-down

    config:
      param0: key0
      param1: key1
```

High Load use case

```
template_name: sample-vnfd-cms
description: demo-example

service_properties:
  Id: sample-vnfd
  vendor: tacker
  version: 1

vdus:
  vdu1:
    id: vdu1
    vm_image: cirros-0.3.4-x86_64-uec
    instance_type: m1.tiny

    network_interfaces:
      management:
        network: net_mgmt
        management: true
      pkt_in:
        network: net0
      pkt_out:
        network: net1

    placement_policy:
      availability_zone: nova

    auto-scaling: noop
    monitoring_policy:
      cpu_util:
        monitoring_params:
          statistic: avg
          period: 60
          evaluation_periods: 1
          threshold: 50
        actions:
          failure_policy: scale-up

    config:
      param0: key0
      param1: key1
```

Light Load use case

Q & A

Load/Path-Aware Service Function Scheduler in OpenDaylight/OPNFV SFC

Dongeun Suh (fever1989@korea.ac.kr)

4th semester of Ph.D. Course
Mobile Network & Communication Lab.
Korea University (KU)

Jaewook Lee (iioiioiio123@korea.ac.kr)

4th semester of Ph.D. Course
Mobile Network & Communication Lab.
Korea University (KU)

Hosung Baek (go1emd@korea.ac.kr)

3rd semester of Master Course
Mobile Network & Communication Lab.
Korea University (KU)

고려대학교 Team

NFV & Legacy Network Functions

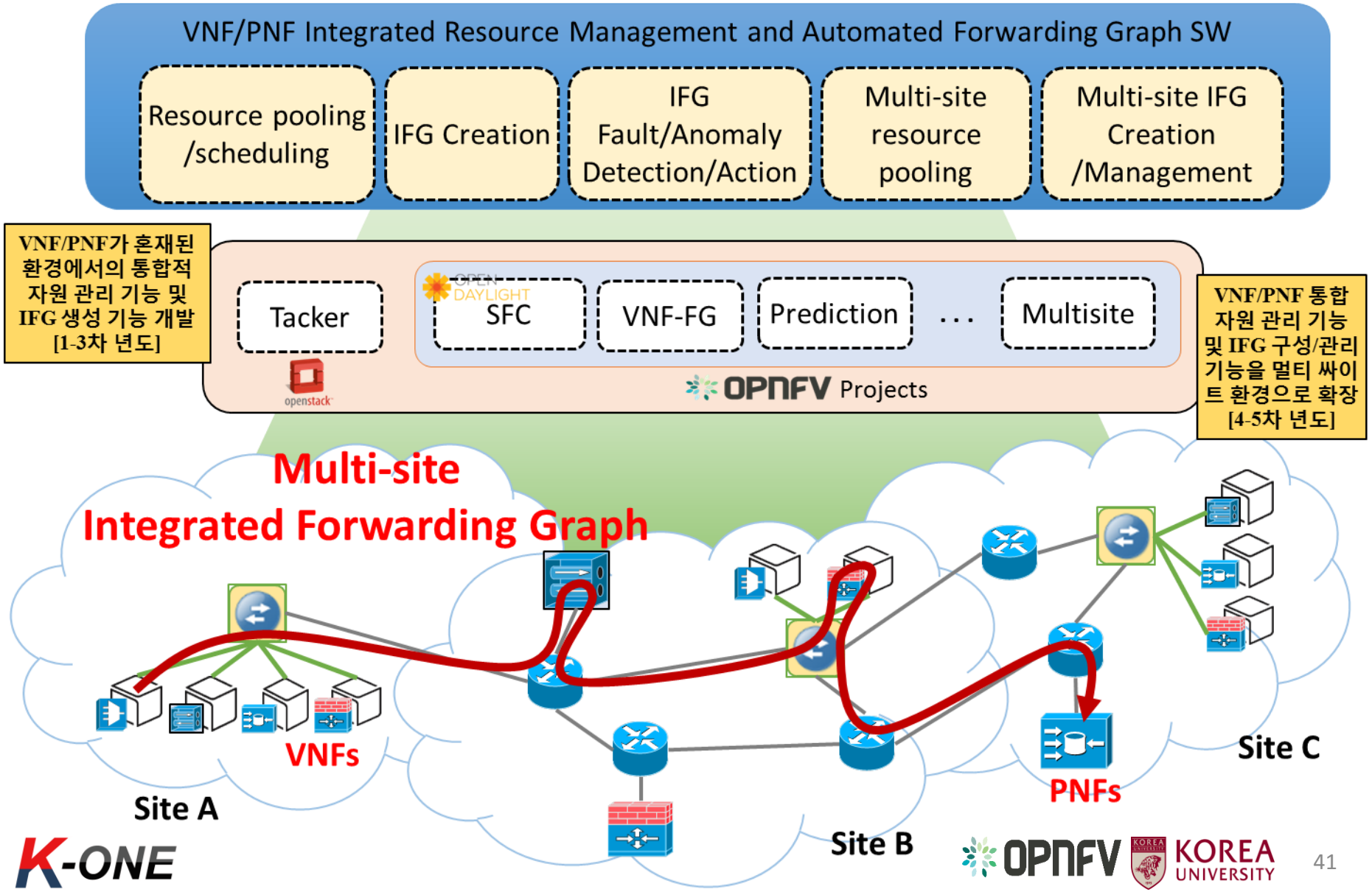
- Virtualized Network Function (VNF) and legacy Physical Network Function (PNF) must co-exist
- VNFs can be dynamically provisioned, started, paused, stopped, in real-time whereas the management procedures over PNFs are much more limited and static
- Integrated resource management / Integrated construction of service chains must be developed
  - Different features of VNF/PNF must be considered

[K-ONE OPNFV #2] Task Overview (1/2)

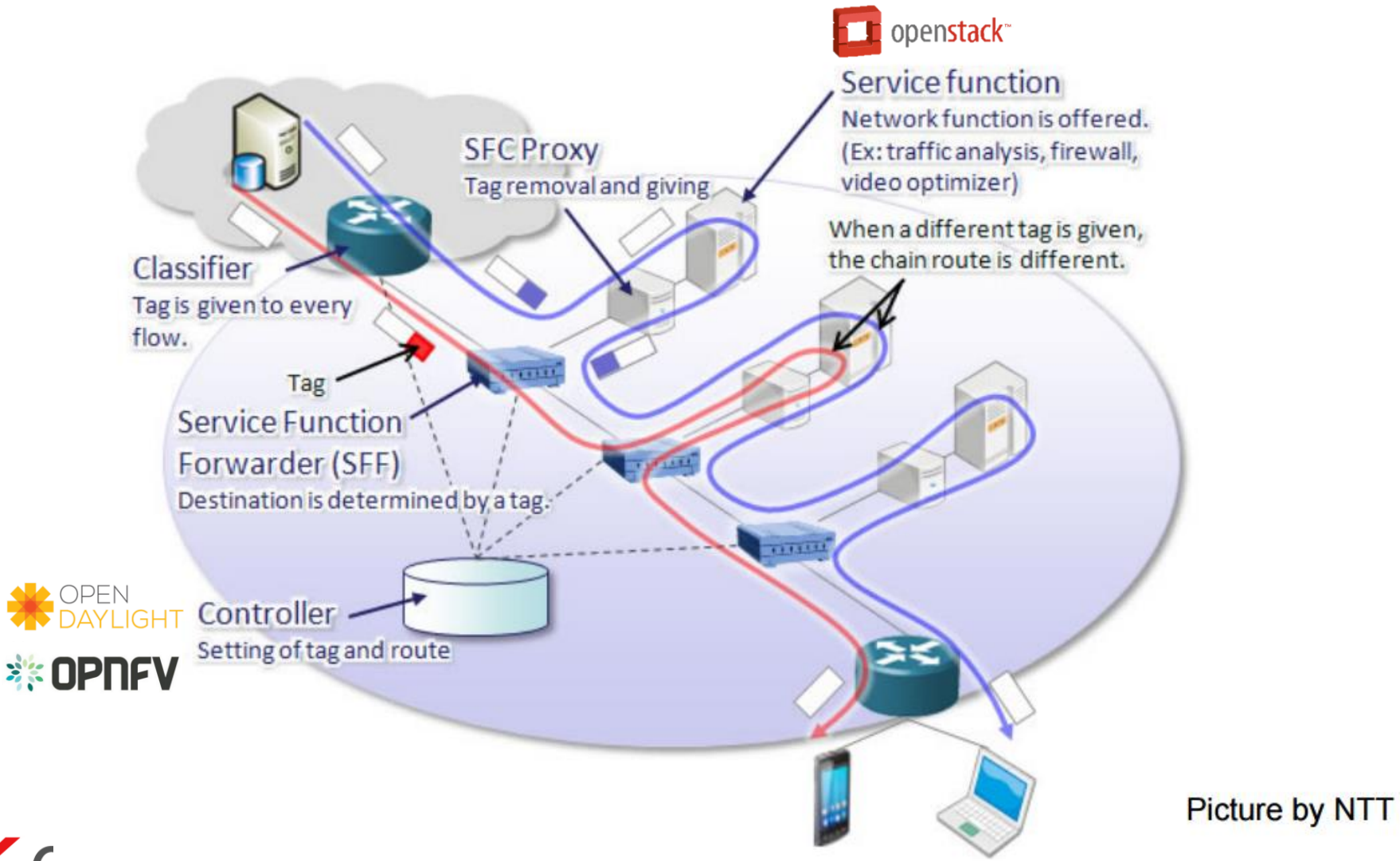
- 목표: Virtualized Network Function (VNF)와 기존의 Physical Network Function (PNF)의 자원을 통합적으로 관리/운용하고 Forwarding Graph를 동적으로 구성할 수 있는 소프트웨어를 개발함

- 기능
  1) VNF/PNF Compute/Storage/Network 자원 풀링 및 스케줄링 기능 개발
  2) VNF/PNF 통합 IFG 구성 기능 개발
  3) VNF/PNF IFG의 결함/이상 현상 감지 및 동적 대처 기능 개발
  4) Multi-site federation 클라우드 환경을 지원하는 통합 자원 관리 기능 개발
  5) Multi-site federation 클라우드 환경을 지원하는 통합 IFG 구성 및 관리 기능 개발

[K-ONE OPNFV #2] Task Overview (2/2)



Service Function Chaining

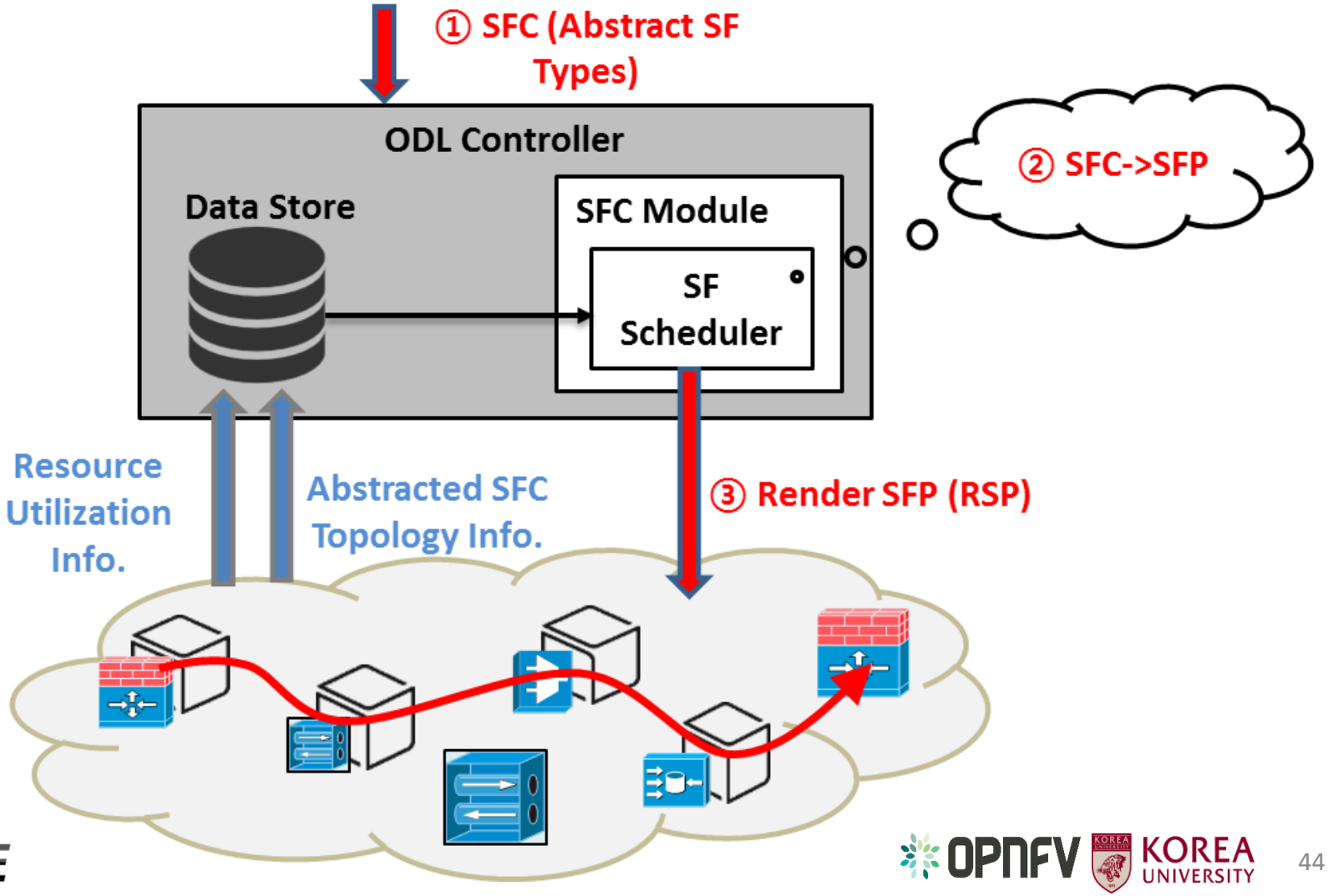


Picture by NTT

Service Function Scheduler

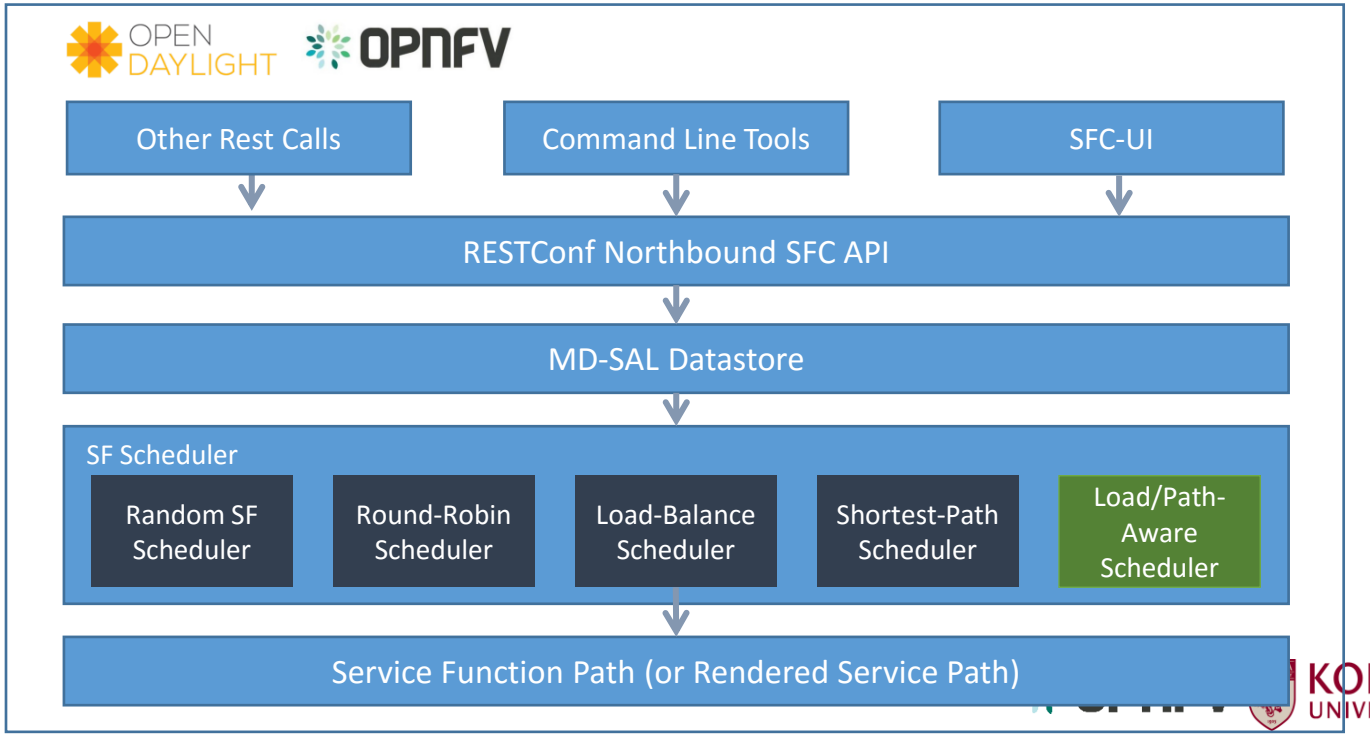
- SFC components
  - The Service Function Chain (SFC) defines an ordered set of Service Function (SF) types
    - e.g., [FW -> NAT -> DPI]
  - The Service Function Path (SFP) defines the specific SF and SFF instances defined in a SFC
    - e.g., [FW_1 -> NAT_2 -> DPI_1]
- SF Scheduler (i.e., SF Selection Algorithm)
  - Select SF instances to be included in the SFP
    - mapping from SFC to SFP
  - Responsible for distributing loads of SF instances while minimizing the length of SFP

SF Scheduler in OPNFV/ODL SFC (1/2)



SF Scheduler in OPNFV/ODL SFC (2/2)

- The SFC project in OPNFV/ODL provides several SF schedulers
  - Too naïve to be used
- Proposed/Implemented Load/Path-Aware SF Scheduler
  - Fairly distributes loads of SF instances while limiting the length of SFP



Existing SF Scheduler

- Random Scheduler
  - 주어진 SF type에 대한 Candidate 인스턴스들 중 임의의 인스턴스를 선택
- Round Robin Scheduler
  - 임의의 SF type에 대한 매 호출마다 다른 SF instance를 선택
- Load Balancing Scheduler
  - SF type에 대한 Candidate 인스턴스들 중 가장 낮은 CPU Utilization을 갖는 SF 인스턴스를 선택
- Shortest Path Scheduler
  - 주어진 Chain의 첫번째 SF type에 대해서는 임의로 선택
  - 그 이후에는 이전에 선택된 SF 인스턴스로부터 모든 Candidate 인스턴스들간의 최단 경로 길이를 계산하여 그 중 가장 짧은 길이를 갖는 인스턴스를 선택

Load/Path-Aware SF Scheduler

- SF 인스턴스들의 부하를 측정 결과 (CPU 사용량)를 기반으로 부하 분산과 Path 길이를 동시에 고려하는SF Scheduler를 고안
- Path의 길이 제한을 상황에 따라 적응적으로 설정 가능
  - *L_TH:* 이전에 선택된 SF 인스턴스로부터 다음에 선택될 SF 인스턴스까지의 홉 수에 대한 Constraint
- 동작 과정
  1. 첫번째 SF Type에 대해서는 CPU Utilization이 가장 낮은 인스턴스를 선택
  2. Chain 내 모든 SF Type들의 인스턴스들이 결정될 때 까지 A-B를 반복
     A. 주어진 SF Type에 대한 Candidate 인스턴스들 중, 이전에 선택된 SF 인스턴스와의 길이가 L_TH보다 크면 Candidate 집합에서 제외시킴
     B. 정제된 Candidate 집합에서 최소 CPU Utilization을 갖는 인스턴스를 선택

Test Environment (1/2)

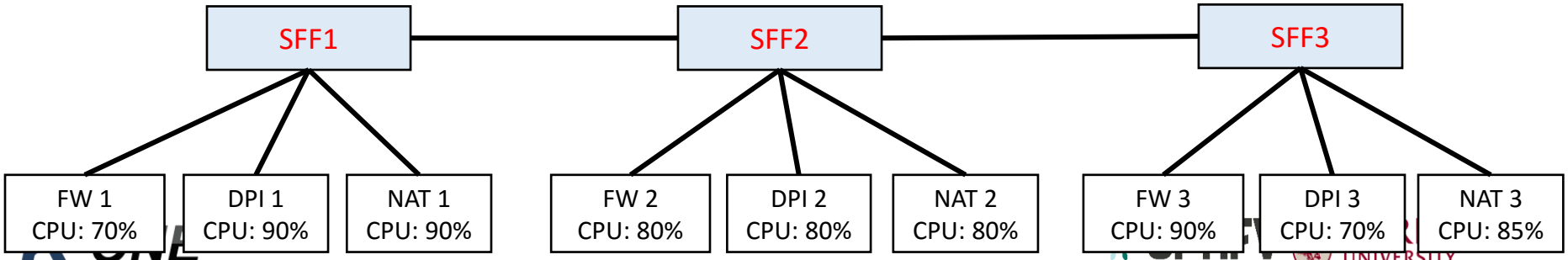
- Load/Path-Aware SF Scheduler에 대하여 JUnit 테스트 기반의 기능 검증을 진행
  - JUnit은 빌드타임 테스트를 위한 프레임워크로 모든 ODL OSGi Bundle들은 JUnit 테스트 코드를 작성하여 기능을 검증
- 테스트 시나리오
  - JUnit 테스트 코드 상에 SF,SFF 들로 구성된 토폴로지를 구성
  - 제안 SF Scheduler의 L_TH 값을 바꾸어가며 통해 주어진 SFC에 대해 어떻게 SFP를 생성하지를 확인

Test Environment (2/2)

- Chain
  - [FW -> DPI -> NAT]
- CPU Utilization

Type	FW	DPI	NAT
SF Instance / CPU Utilization	FW 1 / 70 %	DPI 1 / 90 %	NAT 1 / 90 %
	FW 2 / 80 %	DPI 2 / 80 %	NAT 2 / 80 %
	FW 3 / 90 %	DPI 3 / 70 %	NAT 3 / 85 %

- Topology



Test Topology Generation Log

```
[Topology] Add ServiceFunction: SfName [_value=simple_fw_1]
[Topology] Add ServiceFunction: SfName [_value=simple_fw_2]
[Topology] Add ServiceFunction: SfName [_value=simple_fw_3]
[Topology] Add ServiceFunction: SfName [_value=simple_dpi_1]
[Topology] Add ServiceFunction: SfName [_value=simple_dpi_2]
[Topology] Add ServiceFunction: SfName [_value=simple_dpi_3]
[Topology] Add ServiceFunction: SfName [_value=simple_nat_1]
[Topology] Add ServiceFunction: SfName [_value=simple_nat_2]
[Topology] Add ServiceFunction: SfName [_value=simple_nat_3]
[Topology] Add ServiceFunctionForwarder: SffName [_value=SFF1]
[Topology] Add ServiceFunctionForwarder: SffName [_value=SFF3]
[Topology] Add ServiceFunctionForwarder: SffName [_value=SFF2]
[Topology] Add SFF-to-SFF edge: SffName [_value=SFF1] => SffName [_value=SFF2]
[Topology] Add SFF-to-SFF edge: SffName [_value=SFF3] => SffName [_value=SFF2]
[Topology] Add SFF-to-SFF edge: SffName [_value=SFF2] => SffName [_value=SFF1]
[Topology] Add SFF-to-SFF edge: SffName [_value=SFF2] => SffName [_value=SFF3]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_nat_1] => SffName [_value=SFF1]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_fw_1] => SffName [_value=SFF1]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_dpi_1] => SffName [_value=SFF1]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_nat_3] => SffName [_value=SFF3]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_fw_3] => SffName [_value=SFF3]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_dpi_3] => SffName [_value=SFF3]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_fw_2] => SffName [_value=SFF2]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_dpi_2] => SffName [_value=SFF2]
[Topology] Add SF-to-SFF edge: SfName [_value=simple_nat_2] => SffName [_value=SFF2]
[Topology] Setting ServiceFunction SfName [_value=simple_fw_1] ServiceFunctionType simple_fw_ CPU : 70
[Topology] Setting ServiceFunction SfName [_value=simple_fw_2] ServiceFunctionType simple_fw_ CPU : 80
[Topology] Setting ServiceFunction SfName [_value=simple_fw_3] ServiceFunctionType simple_fw_ CPU : 90
[Topology] Setting ServiceFunction SfName [_value=simple_dpi_1] ServiceFunctionType simple_dpi_ CPU : 90
[Topology] Setting ServiceFunction SfName [_value=simple_dpi_2] ServiceFunctionType simple_dpi_ CPU : 80
[Topology] Setting ServiceFunction SfName [_value=simple_dpi_3] ServiceFunctionType simple_dpi_ CPU : 70
[Topology] Setting ServiceFunction SfName [_value=simple_nat_1] ServiceFunctionType simple_nat_ CPU : 90
[Topology] Setting ServiceFunction SfName [_value=simple_nat_2] ServiceFunctionType simple_nat_ CPU : 80
[Topology] Setting ServiceFunction SfName [_value=simple_nat_3] ServiceFunctionType simple_nat_ CPU : 85
```

Create SFs and SFFs

Connect between SFFs
SFF1 ---- SFF2 ---- SFF3

Connect between SFF and SFs
SFF1 ---- nat_1, fw_1, dpi_1
SFF2 ---- nat_2, fw_2, dpi_2
SFF3 ---- nat_3, fw_3, dpi_3

Set CPU utilization to each SFs

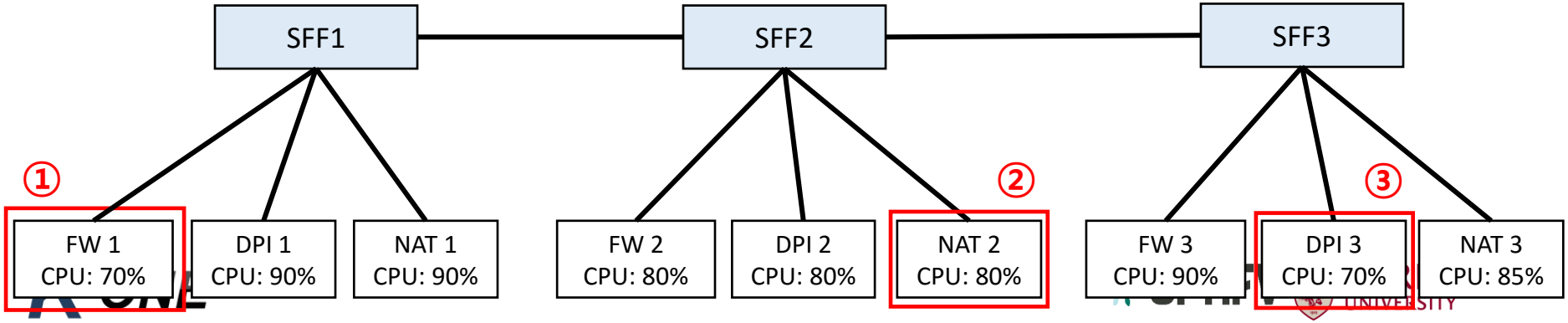
Test Result: L_TH=4

- INPUT: SFC = [FW -> DPI -> NAT]
- RESULT: SFP = [FW1 -> DPI 3 -> NAT 2]

Selected Instance

Candidate Set

Type	FW	DPI	NAT
SF Instance / CPU Utilization	FW 1 / 70 %	DPI 1 / 90 %	NAT 1 / 90 %
	FW 2 / 80 %	DPI 2 / 80 %	NAT 2 / 80 %
	FW 3 / 90 %	DPI 3 / 70 %	NAT 3 / 85 %



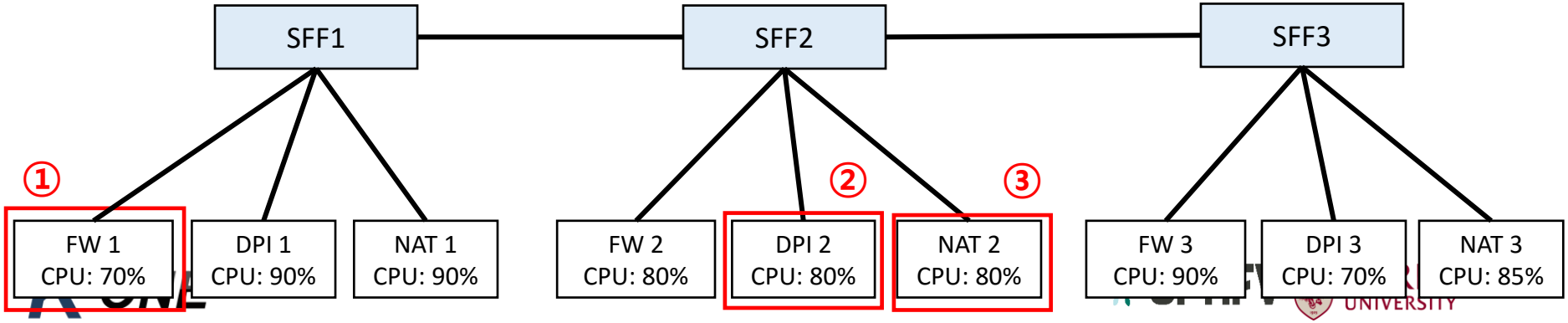
Test Result: L_TH=3

- INPUT: SFC = [FW -> DPI -> NAT]
- RESULT: SFP = [FW1 -> DPI 2 -> NAT 2]

Selected Instance

Candidate Set

Type	FW	DPI	NAT
SF Instance / CPU Utilization	FW 1 / 70 %	DPI 1 / 90 %	NAT 1 / 90 %
	FW 2 / 80 %	DPI 2 / 80 %	NAT 2 / 80 %
	FW 3 / 90 %	DPI 3 / 70 %	NAT 3 / 85 %



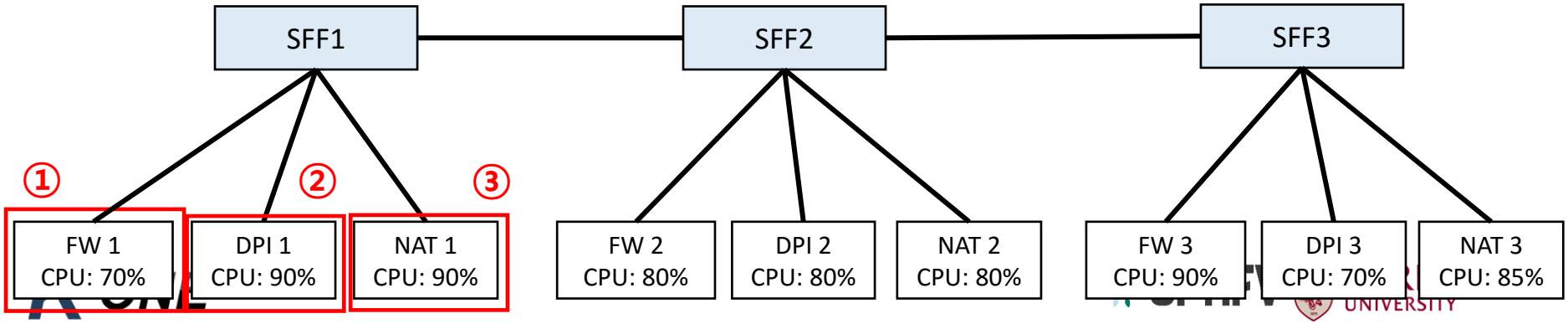
Test Result: L_TH=2

Selected Instance

Candidate Set

- INPUT: SFC = [FW -> DPI -> NAT]
- RESULT: SFP = [FW1 -> DPI 2 -> NAT 3]

Type	FW	DPI	NAT
SF Instance / CPU Utilization	FW 1 / 70 %	DPI 1 / 90 %	NAT 1 / 90 %
	FW 2 / 80 %	DPI 2 / 80 %	NAT 2 / 80 %
	FW 3 / 90 %	DPI 3 / 70 %	NAT 3 / 85 %



Future Work

- Differentiated scheduling for VNF/PNF will be added in the SF scheduler

- Integrated resource monitoring and management for IFG on heterogeneous resources (virtual and physical)

Thank you for your attention!