ODENOS Hands-On

Rev 1.1

Cloud System Research Laboratories
NEC Corporation

This research is a part of the project for “Research and Development of Network Virtualization Technology” supported by the Ministry of Internal Affairs and Communications.

Today’s Agenda

1. ODENOSの概要（30分）
2. ハンズオンセッション（60分）

（注）セットアップに時間がかかるので、概要説明中にセットアップ作業を並行して進めるのをお勧めします

ODENOS Setup

- System Requirement
  - CPU: Intel x64 (with compatible)
  - Memory: >= 2GB
  - Ubuntu Desktop 14.04 LTS
- 構築方法
  - Dockerを使う場合（おすすめ）
    1. sudo apt-get install docker.io
    2. sudo docker pull odenos/odenos-handson
  - ベアメタルにインストールする場合（参考）
    - https://github.com/o3project/odenos/blob/develop/doc/QUICKSTART.md

- セットアップが上手くいかなかった場合は、ハンズオンセッションにて対応します

Outline

ODENOS

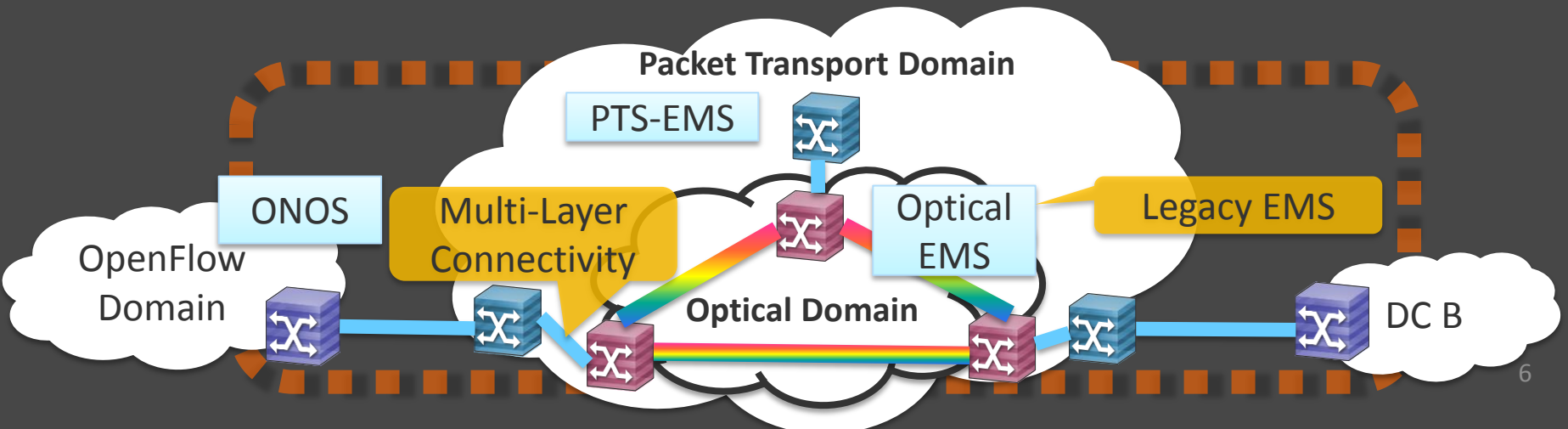
(Object-DEfined Network Orchestrator System)

- SDK to build a heterogeneous network orchestrator
  - Open source software, distributed on GitHub now!
    - https://github.com/o3project/odenos
  - Developed by O3 Project
- ODENOS' aim:
  - Building a reusable network orchestrator easily
  - Flexible end-to-end control of WAN
- **ODENOS' Key:**
  - **Network Abstraction**
  - **Network Conversion**

Wide Area Network

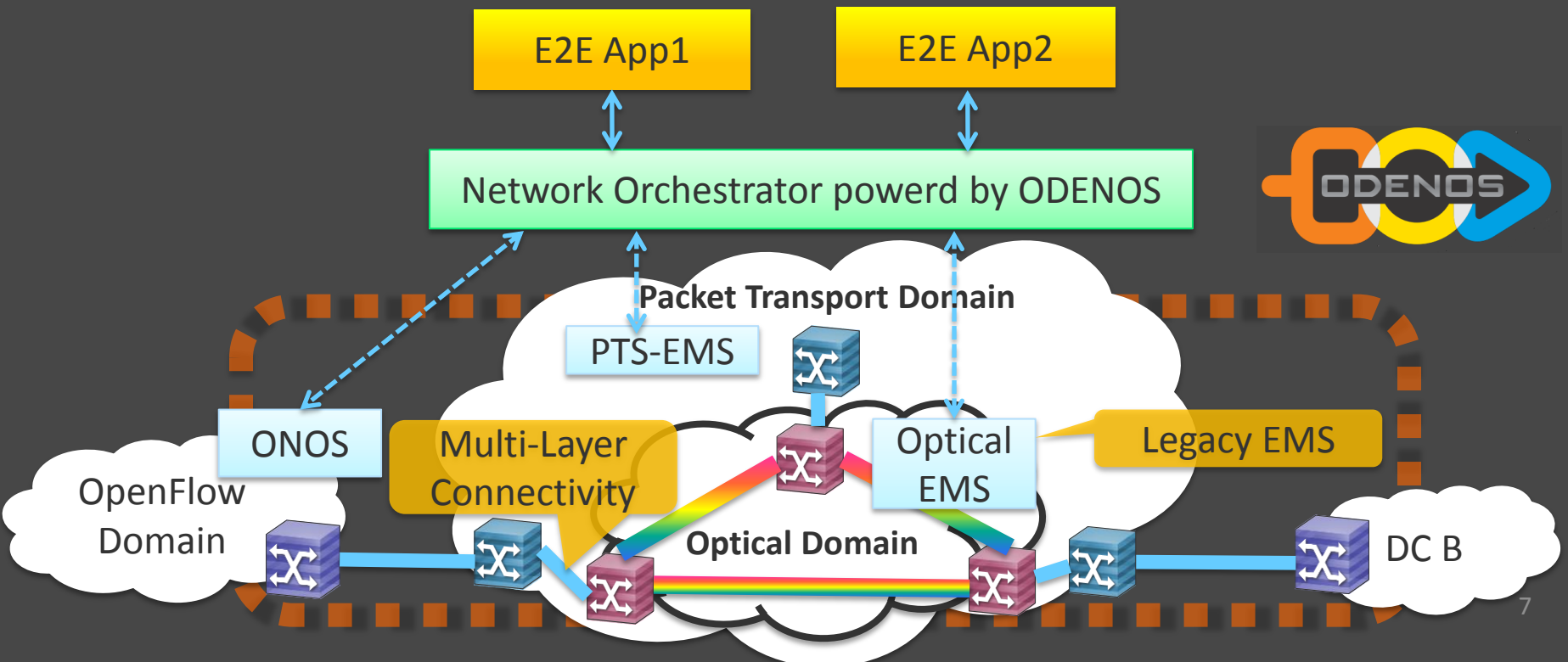
- Network of networks (Heterogeneously)
  - Multi-layer, multi-vendor, multi-domain network
  - Many legacy network equipment
- Accommodating many network services
  - Correlation of network services is very complicated

Developer need a solution to manage network!



Network Orchestrator

- Integrating networks which controlled by different kind of controller or protocols
- Providing unified north-bound API for develop end-to-end network application

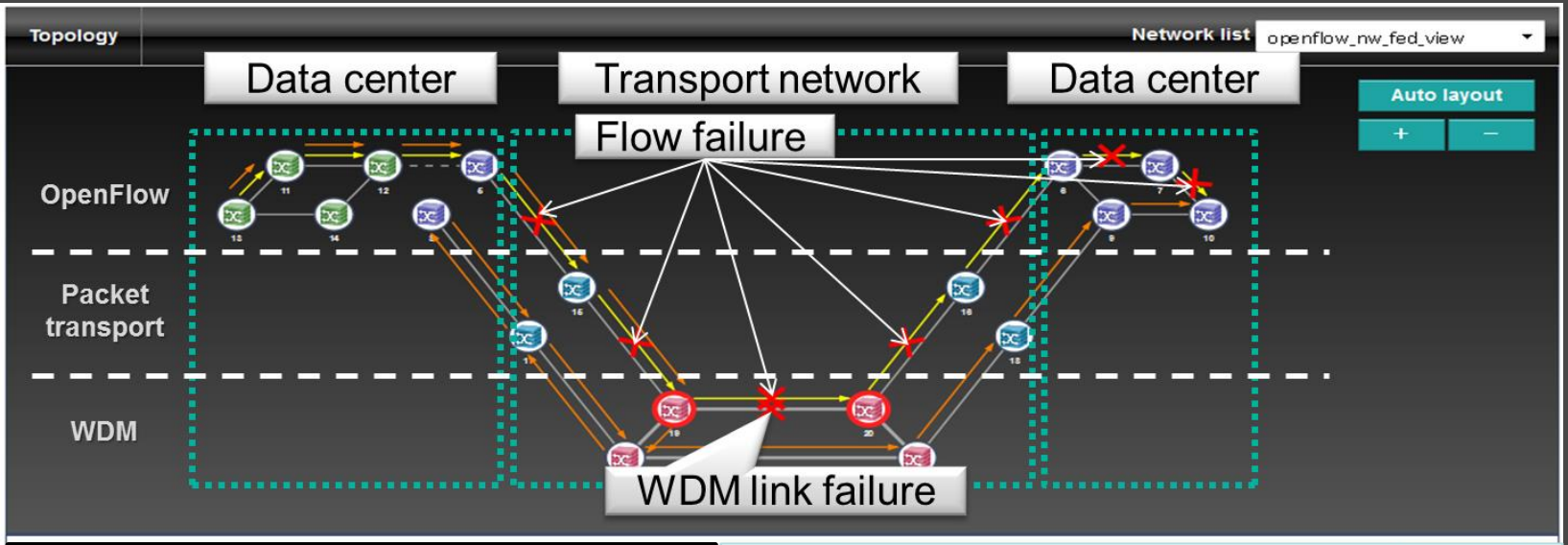


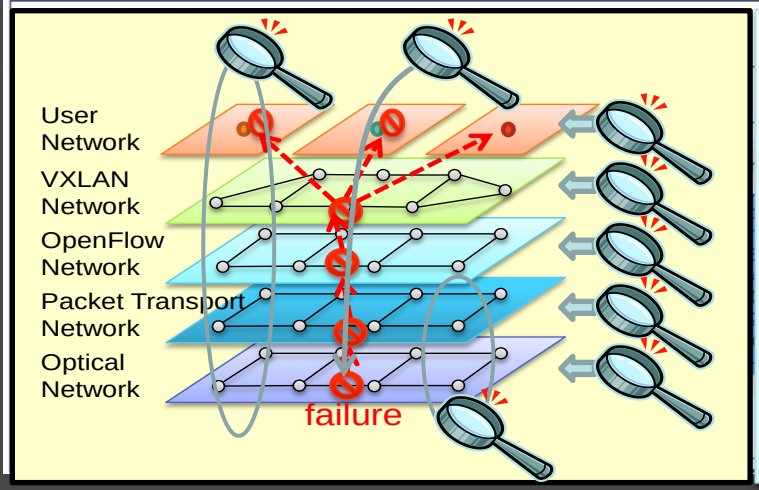
ODENOS

(Object-DEfined Network Orchestrator System)

- SDK to build a heterogeneous network orchestrator
  - Open source software, distributed on GitHub now!
    - https://github.com/o3project/odenos
  - Developed by O3 Project
- ODENOS' aim:
  - Building a reusable network orchestrator easily
  - Flexible end-to-end control of WAN
- **ODENOS' Key:**
  - **Network Abstraction**
  - **Network Conversion**

Use case: Failure analysis of heterogeneous network

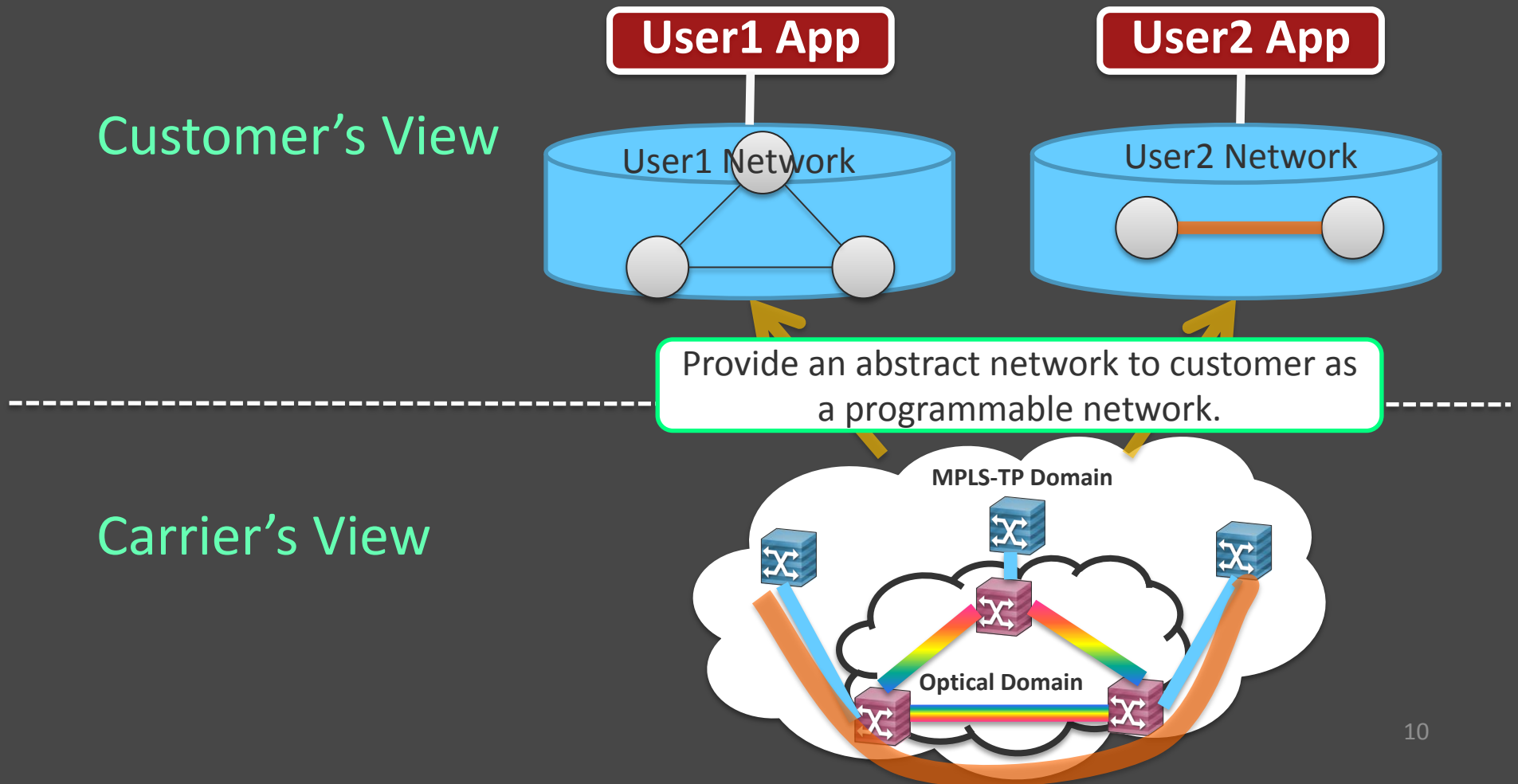




- Multi-layer topology visualization from logical network instances
- Inter-layer correlation mapping through operators
- Trouble shooting, failure analysis, etc.

Use Case: Programmable Network Provisioning

- Providing programmable network to multiple customer

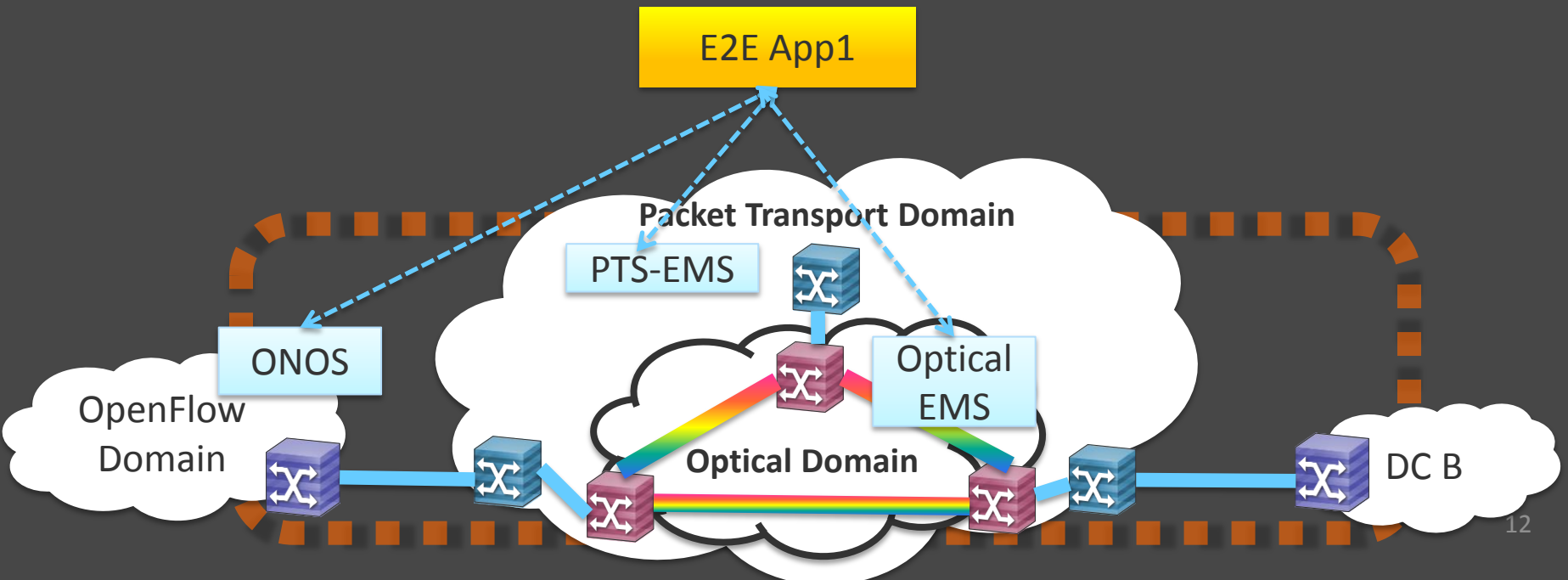


Today's Outline

- What's ODENOS 01
- ODENOS Design 02
  - Network Abstraction
  - Network Conversion
- ODENOS Implementation 03
- Hands-on 04

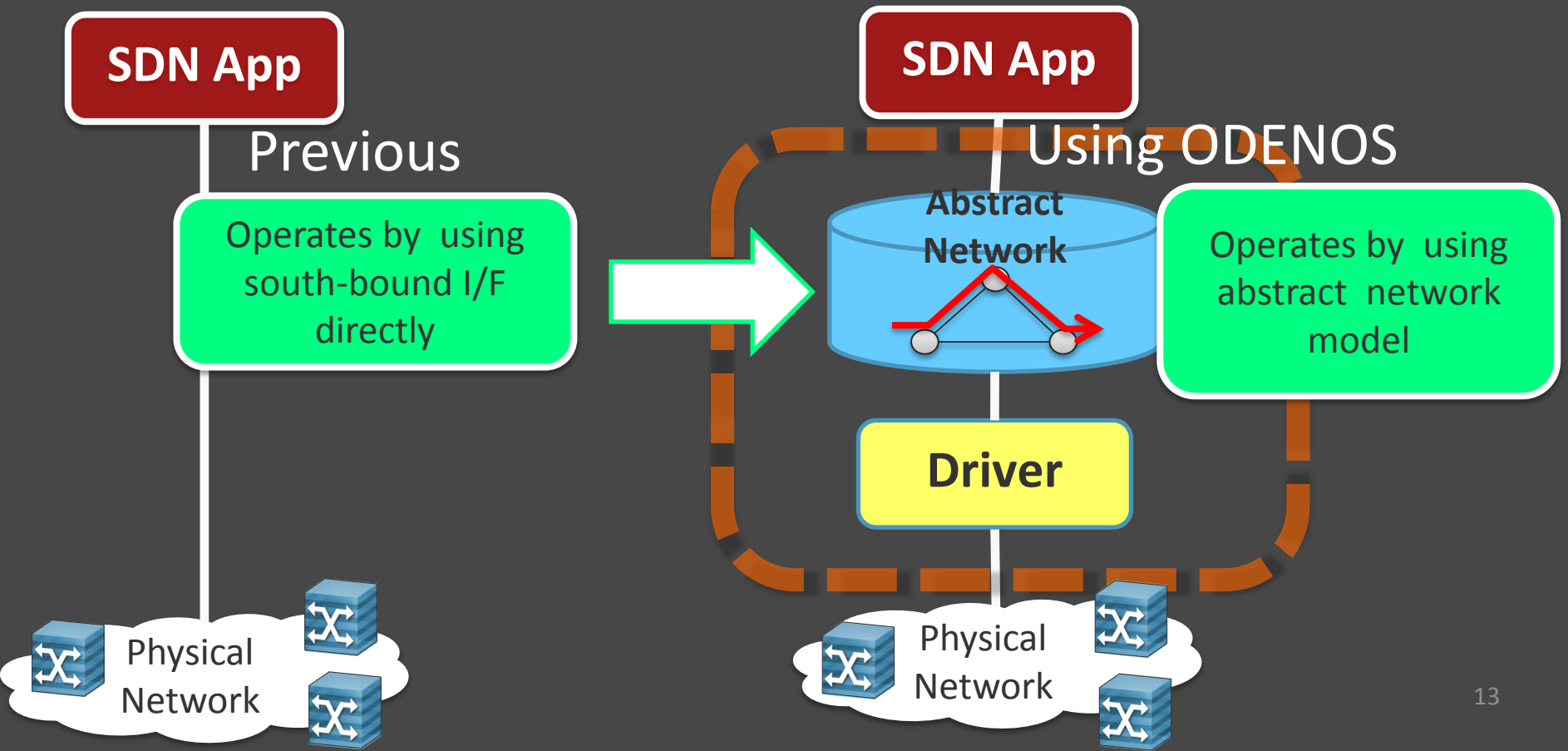
How to control heterogeneous network?

- There are many kinds of interface to control network
  - OpenFlow, NetConf, Vendor specific API to access EMS
- Developer has to implement application logic to support these different APIs, It is difficult work



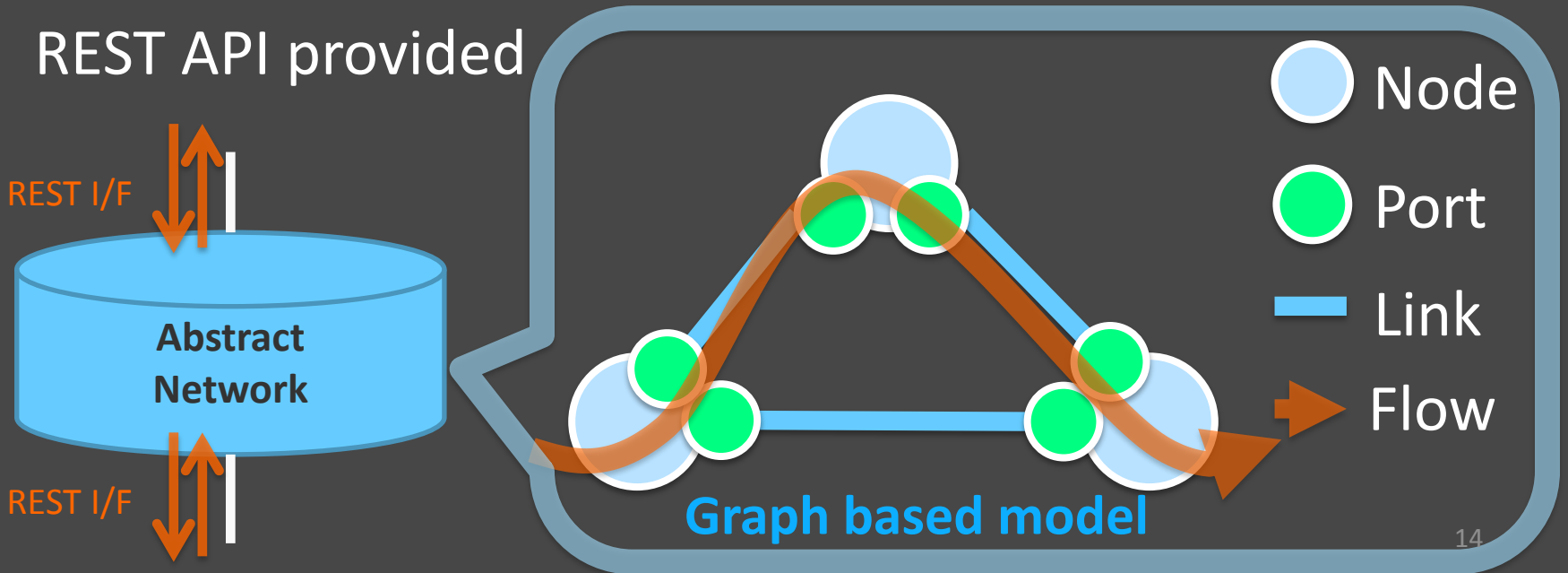
Key Idea:Network Abstraction

- Application controls only abstract network regardless any kinds of physical network

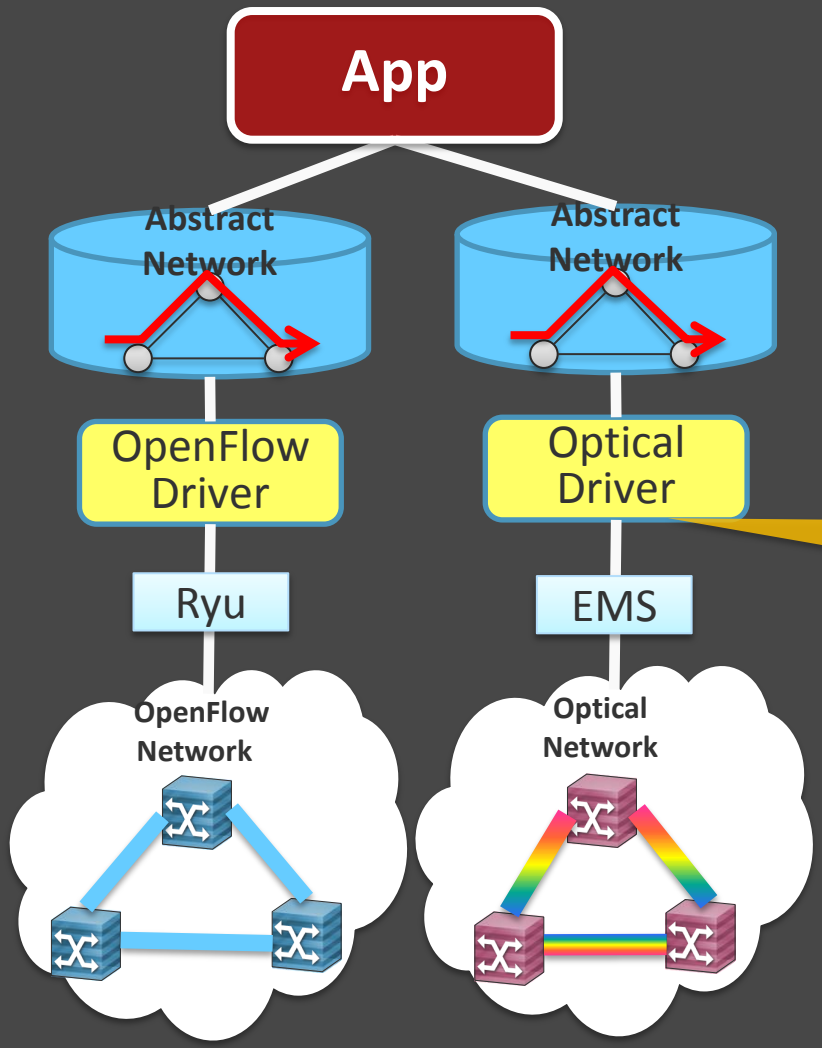


Abstract Network Model

- Graph based model
  - Topology: node, port and link
  - Flow: sequence of transit node and ingress/egress ports
  - Packet: communication between controller and network equipment (like SNMP Trap, OpenFlow Packet-In/Out)
- REST API provided



Driver Component



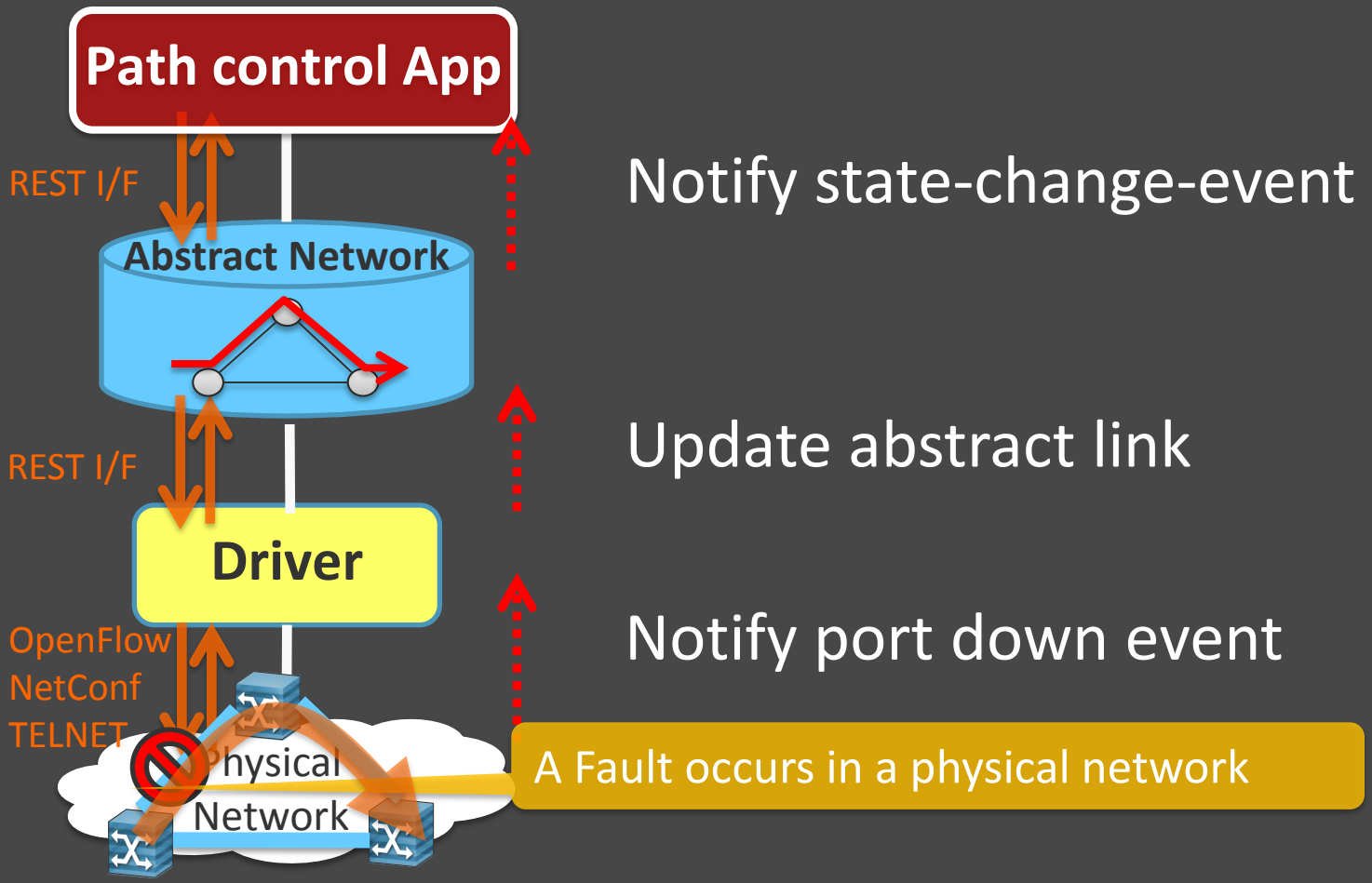
- manages physical equipment using network-specific south-bound API
- manages a mapping information between abstract network and physical network

Driver is implemented as thin wrapper of some controller

Developer can write application logic independently of south-bound API

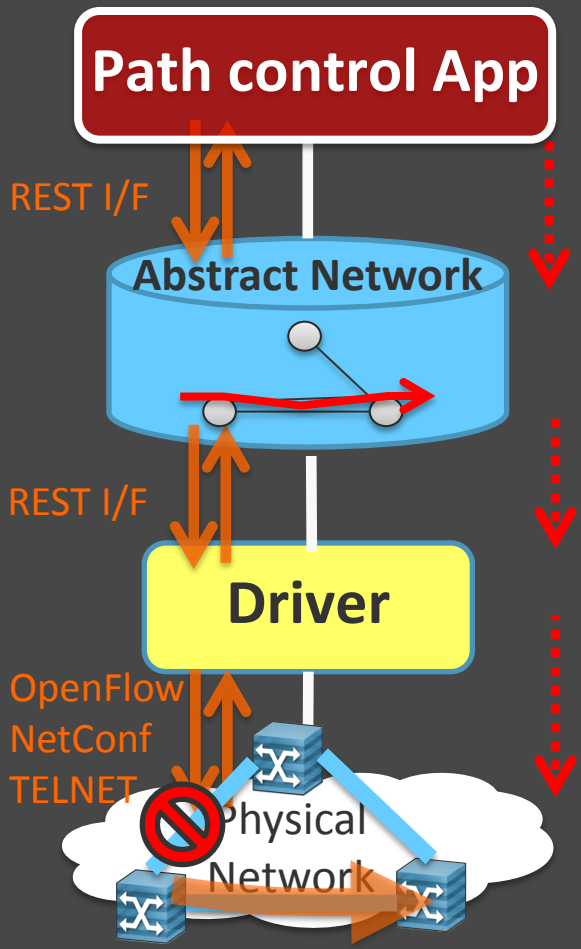
How Application Work with ODENOS

Changes of state are notified as events along by the connection



How Application Work with ODENOS

Changes of state are notified as events along by the connection



Update abstract path

Notify path-change-event

Send flow_mod message

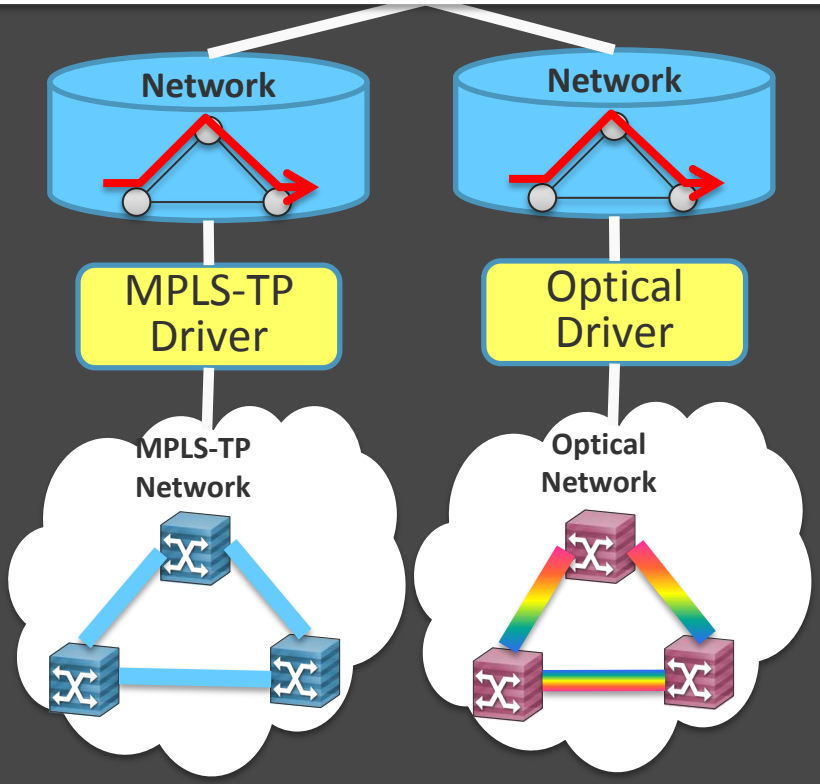
Today's Outline

- What's ODENOS 01
- ODENOS Design 02
  - Network Abstraction
  - Network Conversion
- ODENOS Implementation 03
- Hands-on 04

Application Complexity

UNCONTROLLABLE!

App with:
slicing, topology conversion, configure multi-layer connectivity, and more functionalities

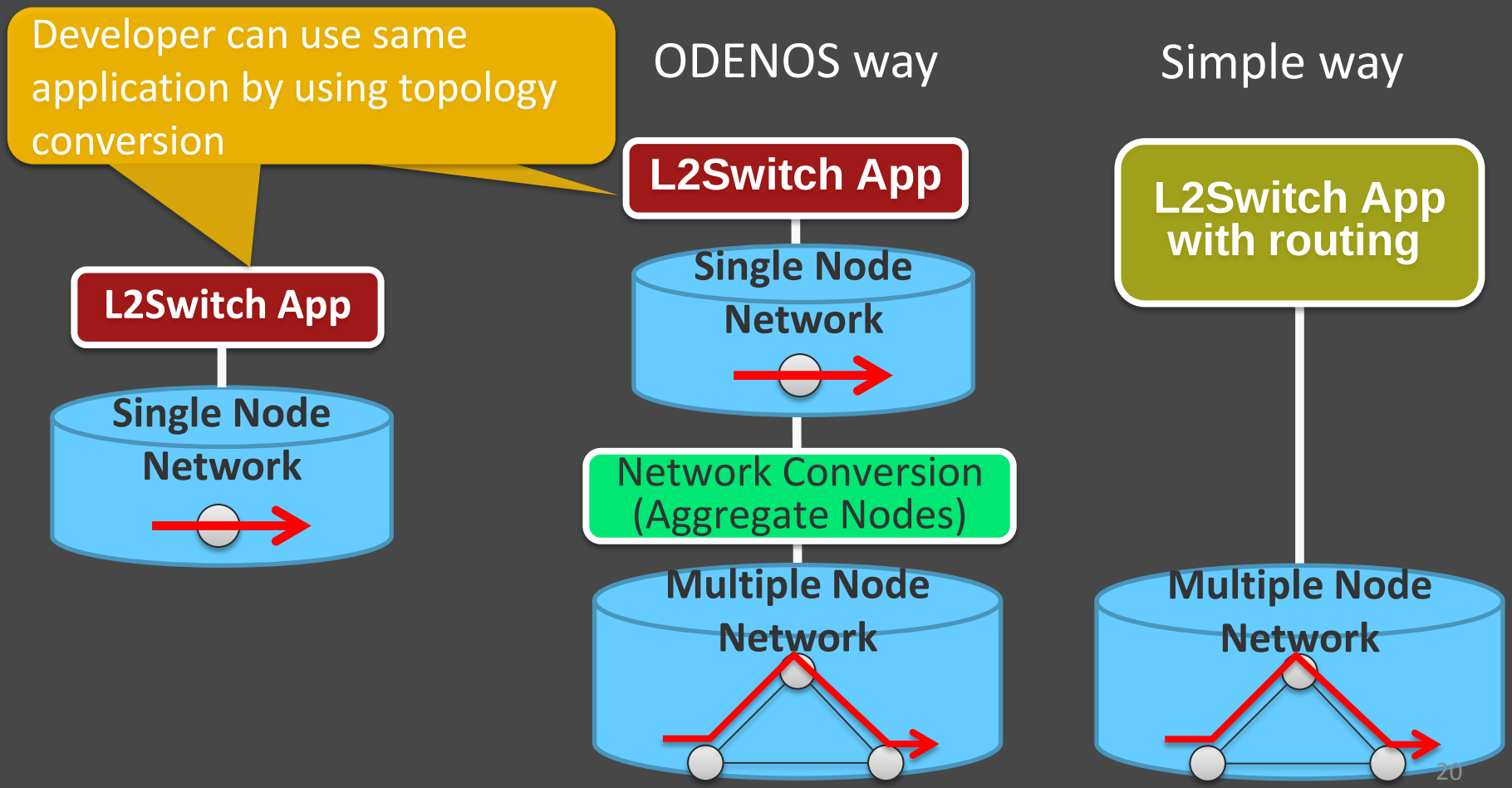


- One big application
  - High complexity
  - Non-reusable
- Many applications have common functionalities
  - Slicing the network
  - Managing the multi-layer connectivity

Be simple!

Key Idea: Network Conversion

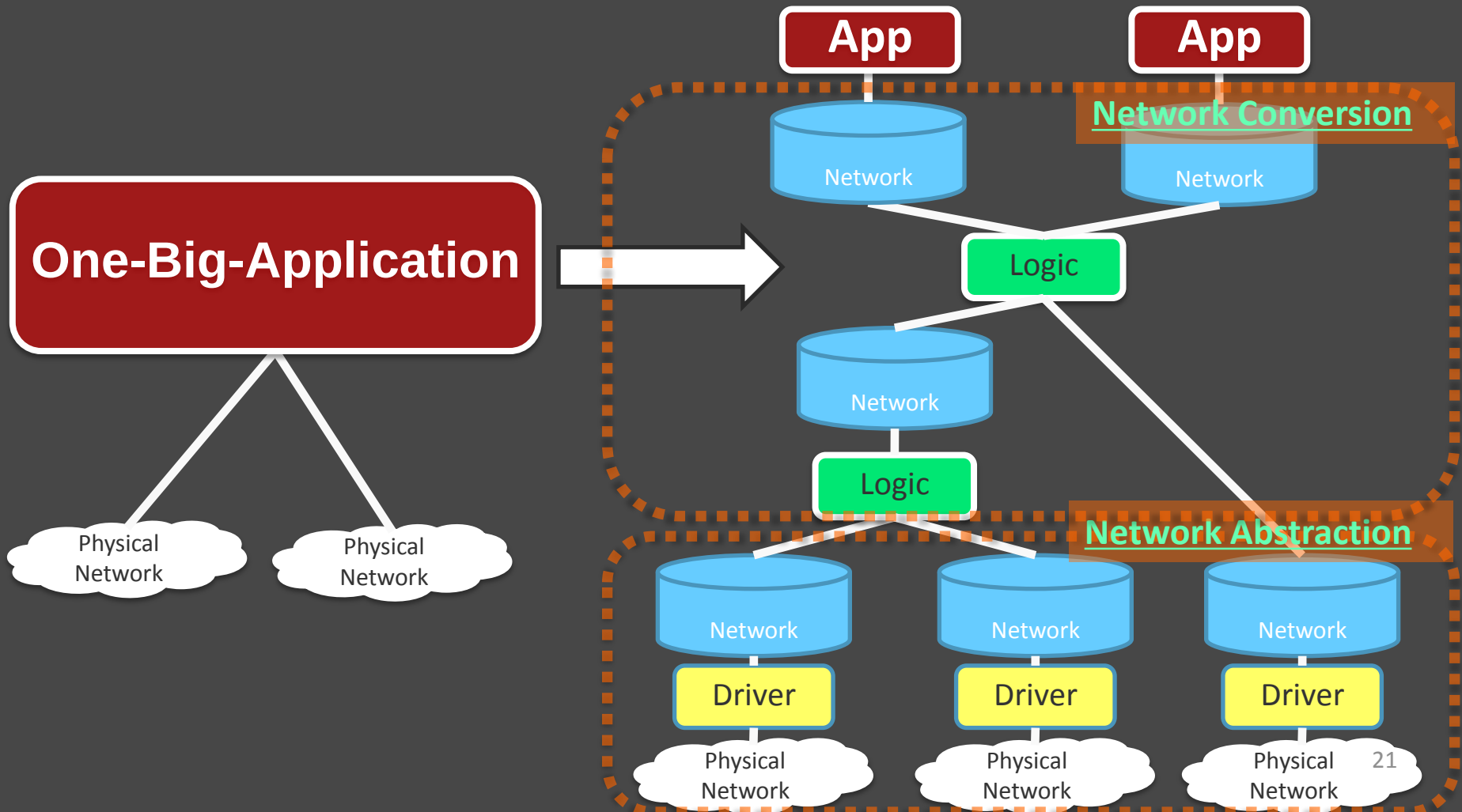
- Converting a complex network to more simple one instead of building a complex application



Overview of Controller

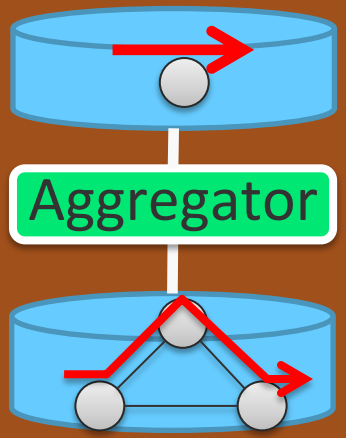
Previous

Using ODENOS

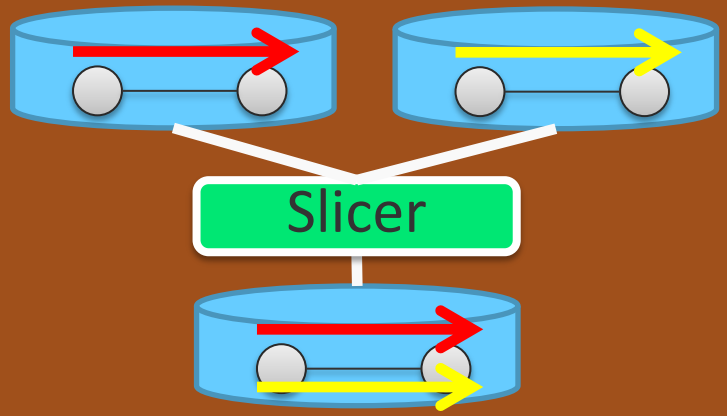


Four Types of Typical Logics

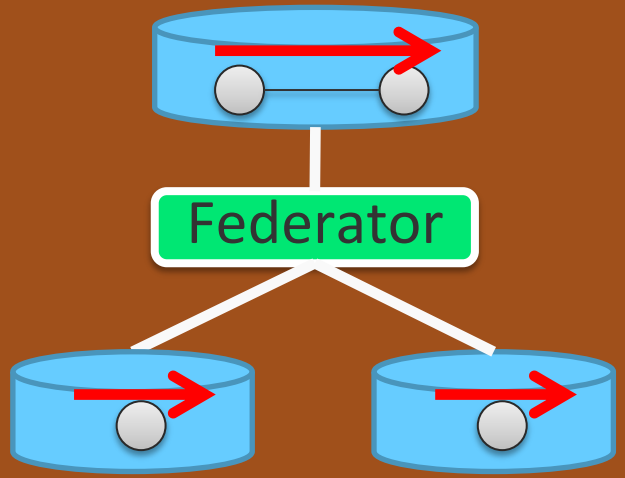
Entire network into a logical node



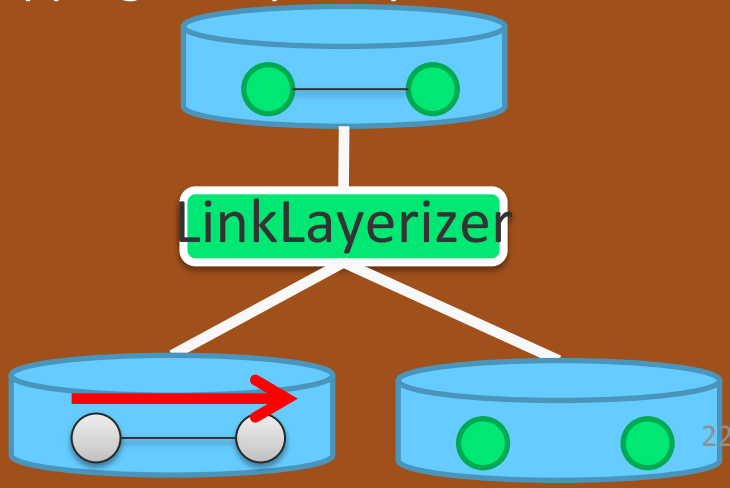
Network into multiple virtual networks



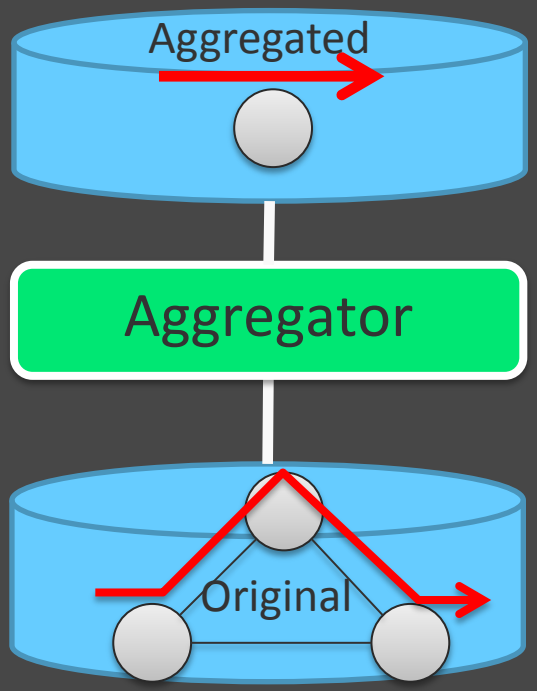
Combining networks into a network



Mapping multiple layers into a network

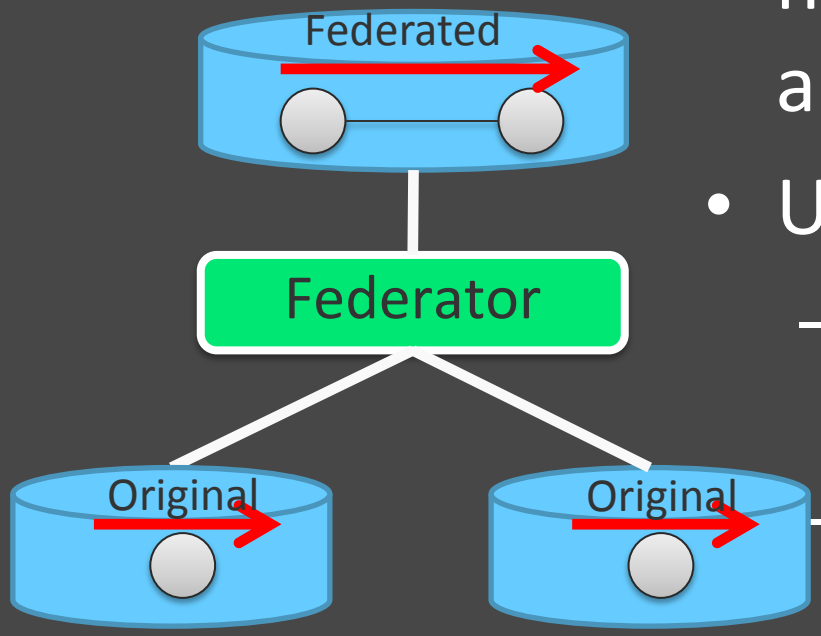


Aggregator



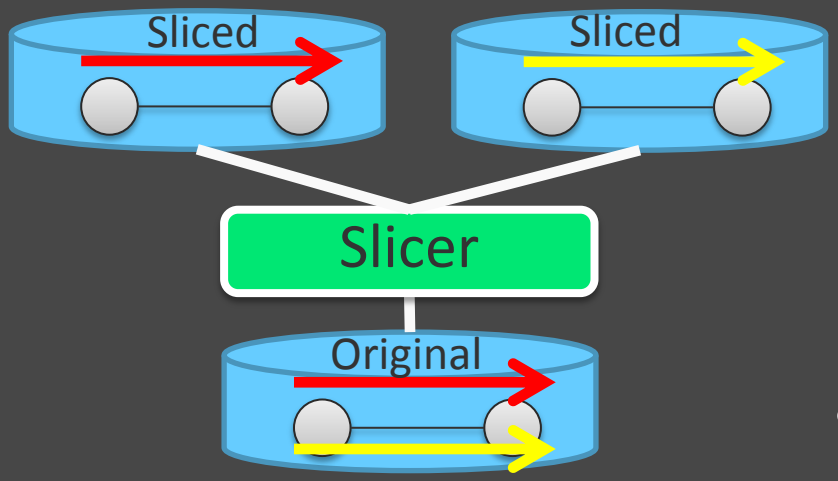
- Aggregates an entire network nodes into a single logical node
- Use cases:
  - One single switch abstraction
  - Implementing an application without considering of multi-node network
  - Hiding a detail of underlay network topology

Federator



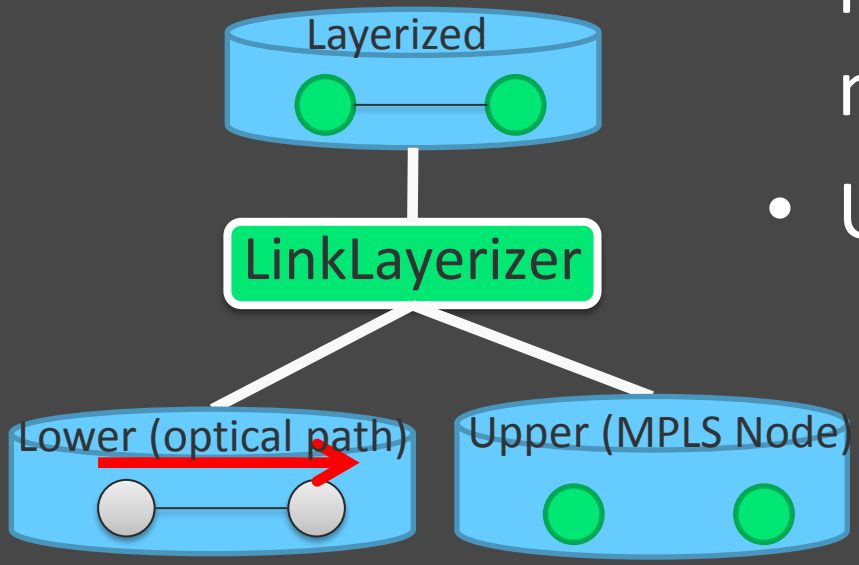
- Combines multiple abstract networks into a single abstract network
- Use case:
  - Combines different domain networks
  - Combines multiple openflow networks which controlled by different controller

Slicer



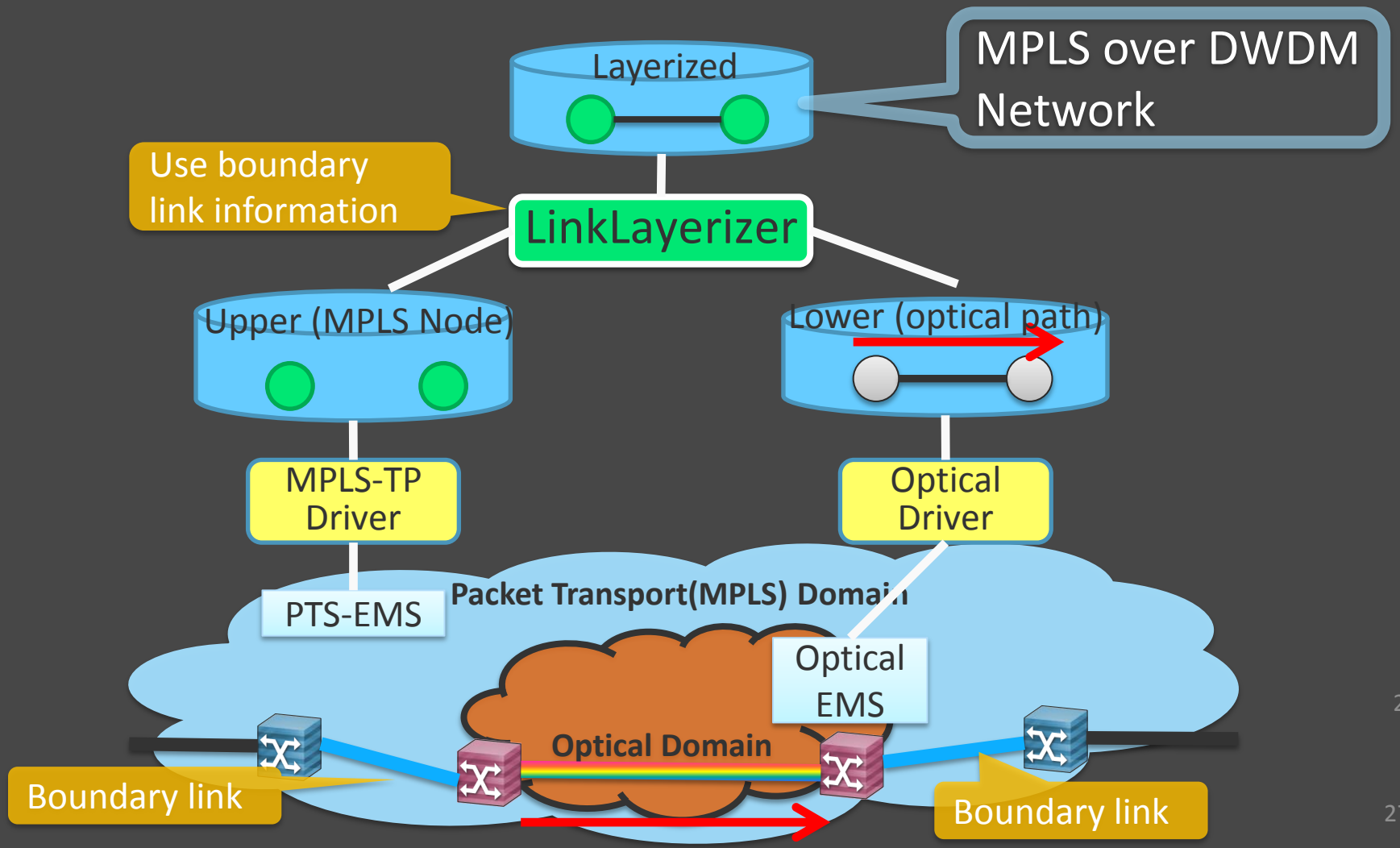
- Slices a network into multiple abstract network with same topology but isolated name space of flow
- Use case:
  - Provisioning virtual networks on shared infrastracture

LinkLayerizer



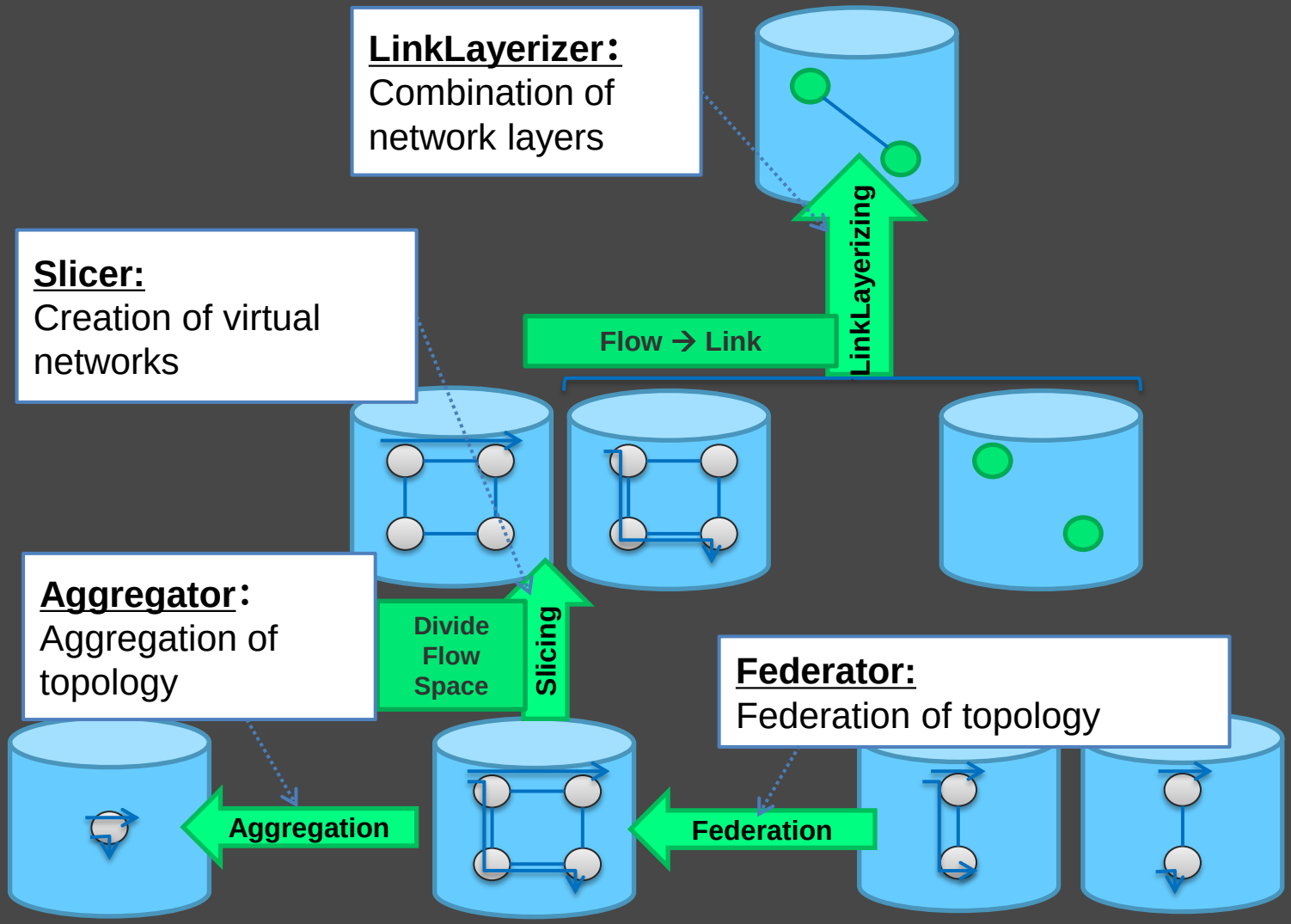
- Integrates multiple networks layers into a single abstracted network
- Use case:
  - Integrates WDM and PTN networks
  - Integrates PTN and OpenFlow networks

Multi-layer network integration by LinkLayerizer

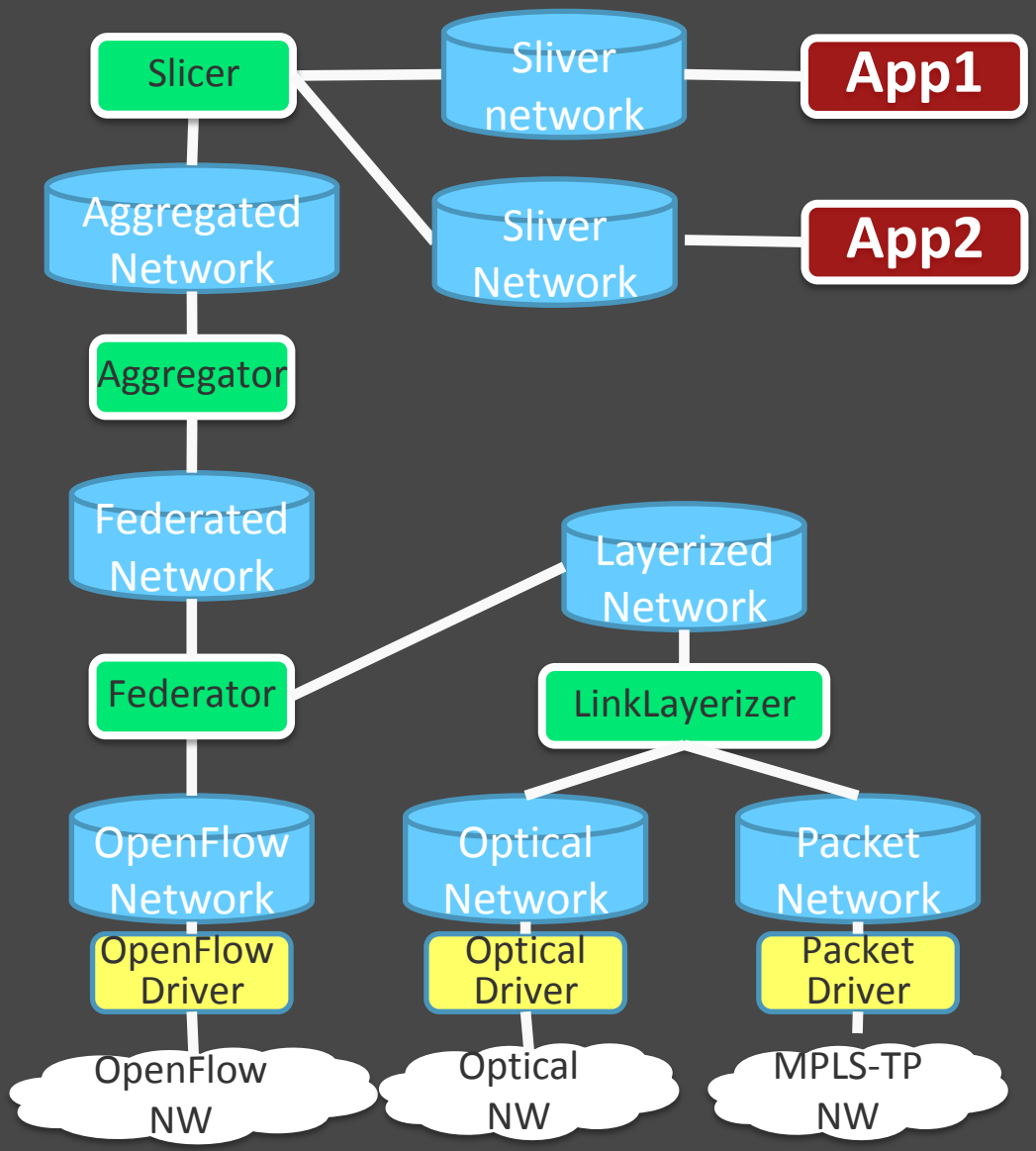


27

Relation of Conversion

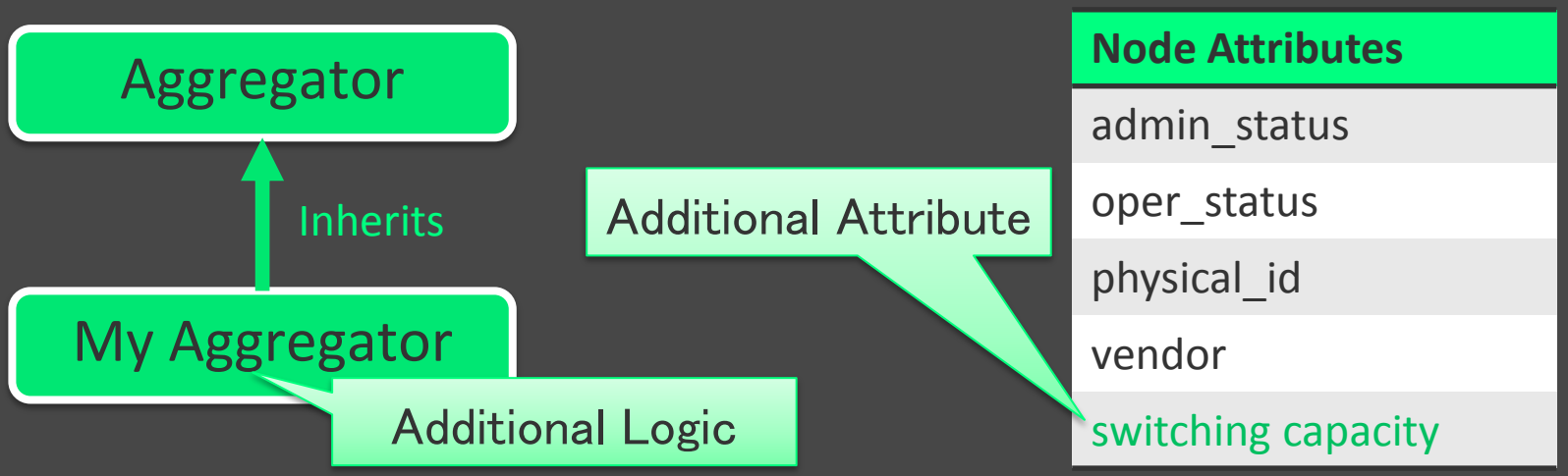


Controller Example



Model Extensibility

- Developer can extend a logic and network model
  - You can create new logic with additional functionality using the inherit
  - Network model can accept additional attributes
    - Default definition is very simple



Today's Outline

What's ODENOS 01

ODENOS Design 02

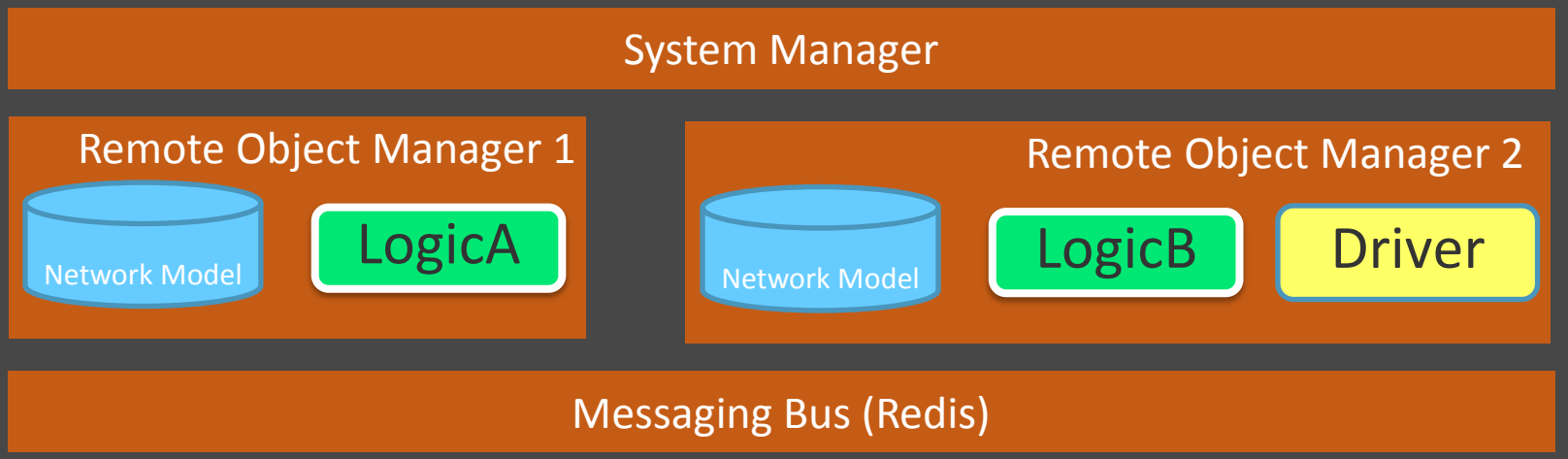
- Network Abstraction
- Network Conversion

ODENOS Implementation 03

Hands-on 04

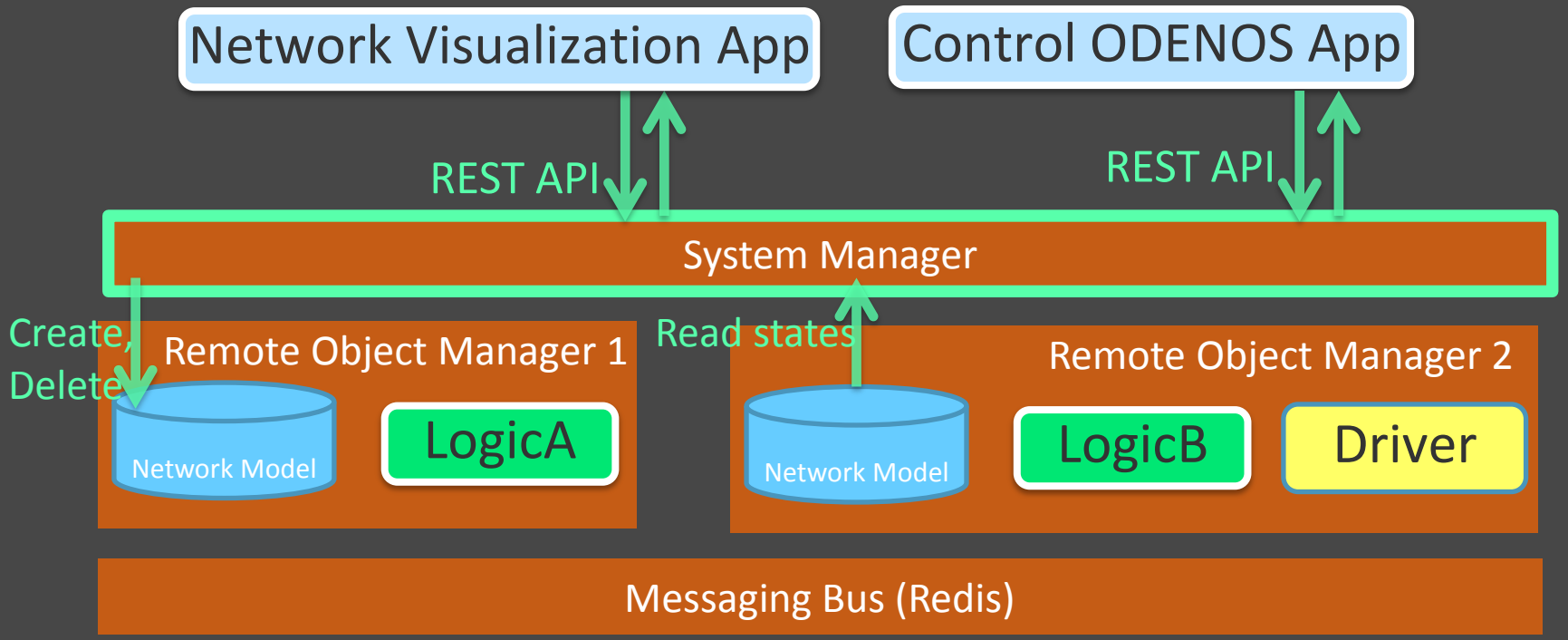
ODENOS System Architecture

- System Manager
  - Create and delete remote objects in remote object manager
- Remote Object Manager
  - Host remote objects (Network instance, Logic and Driver)
- Messaging Bus (Pub/Sub based)
  - Transport messages among remote objects



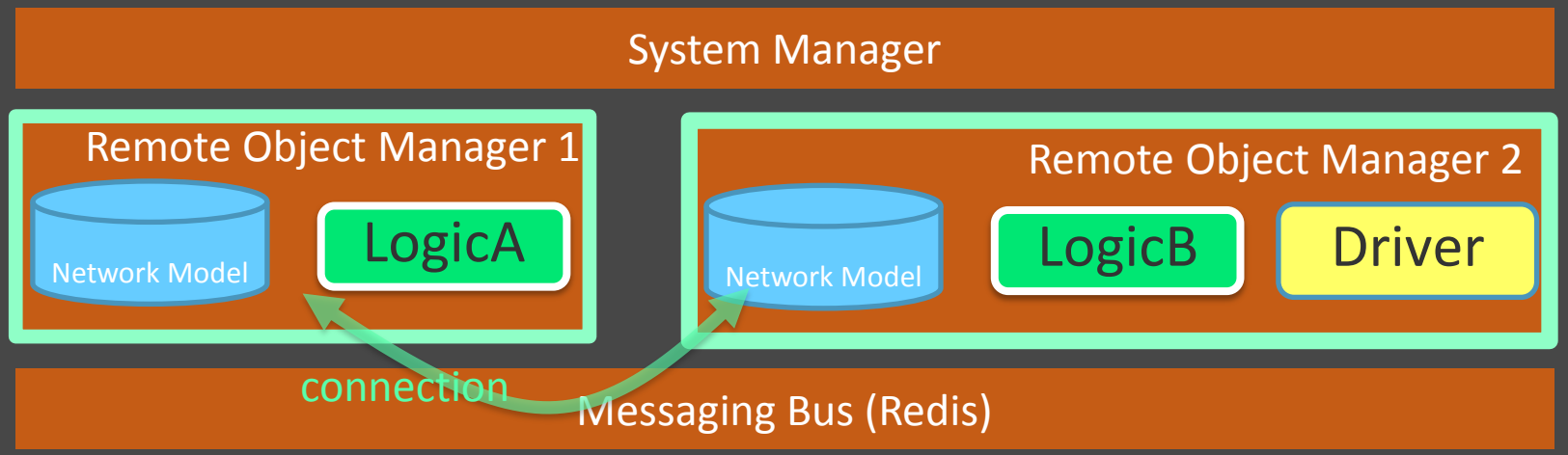
System Manager

- Maintains ODENOS processes (only one at a time in the system)
- Provides an interface to remote objects as CRUD, and forward request to Remote Object Manager
  - System Manager doesn't have object instances (work as a proxy)

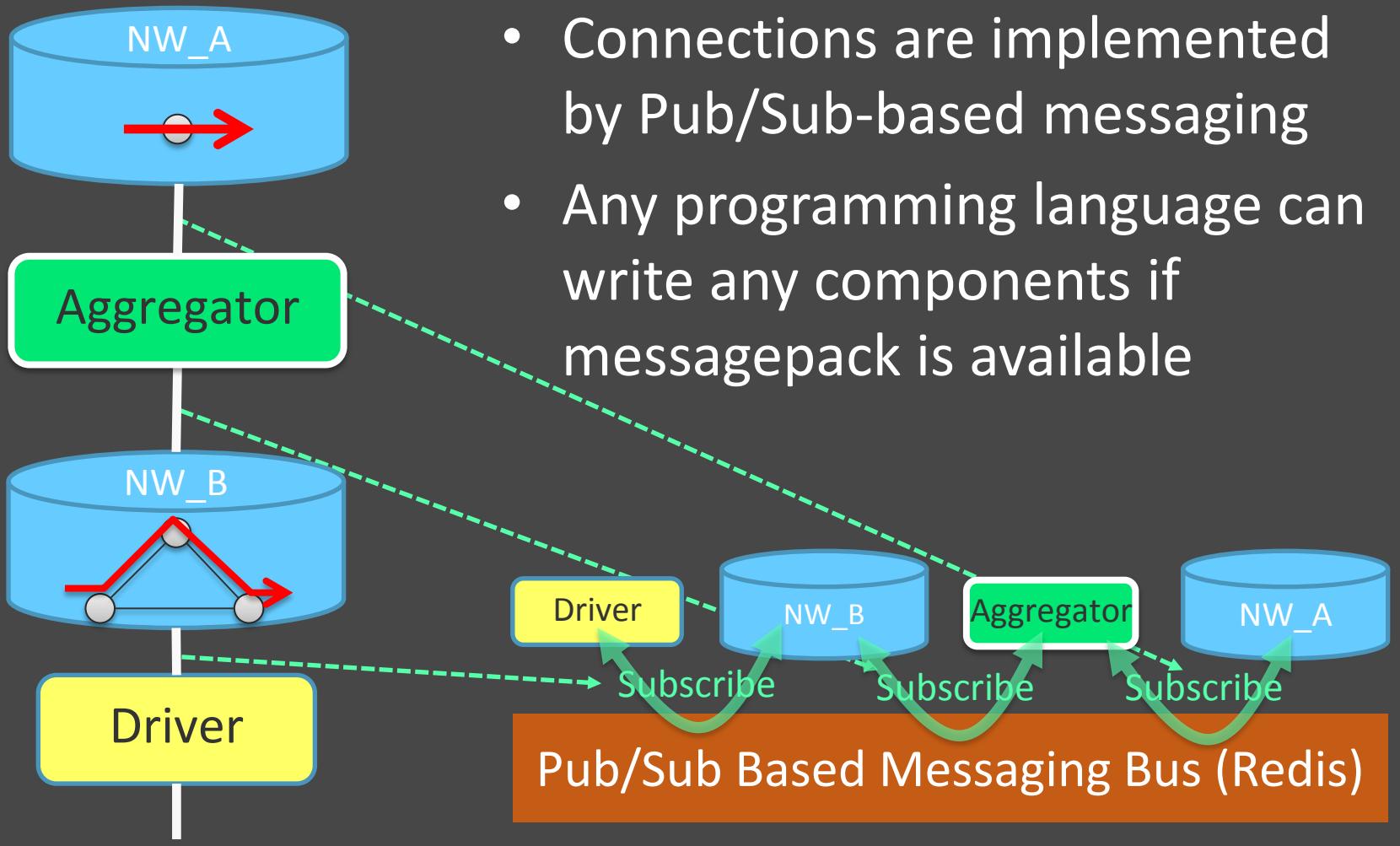


Remote Object Manager

- Manages remote objects (NW, Logic and Driver) and connections among them
- Runs as one or more process
  - To support geographically distributed controllers
  - To support multiple programing languages



Implementation of Connection among Components



- Connections are implemented by Pub/Sub-based messaging
- Any programming language can write any components if messagepack is available

Implementation of Aggregator

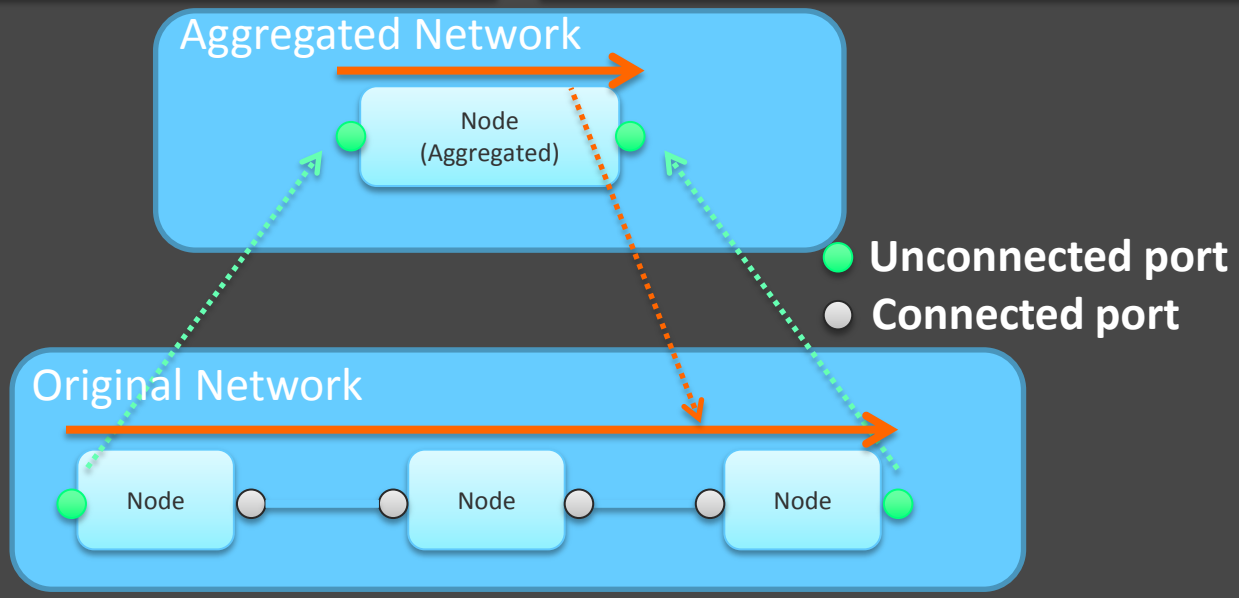
INPUT: Aggregated/Original NW

Converting Topology

1. Aggregate original network nodes into a single aggregated node
2. Copy original network ports that not connected with internal link to aggregated network

Converting Flow

1. Calculate path in an original network and configure it



Implementation of Federator

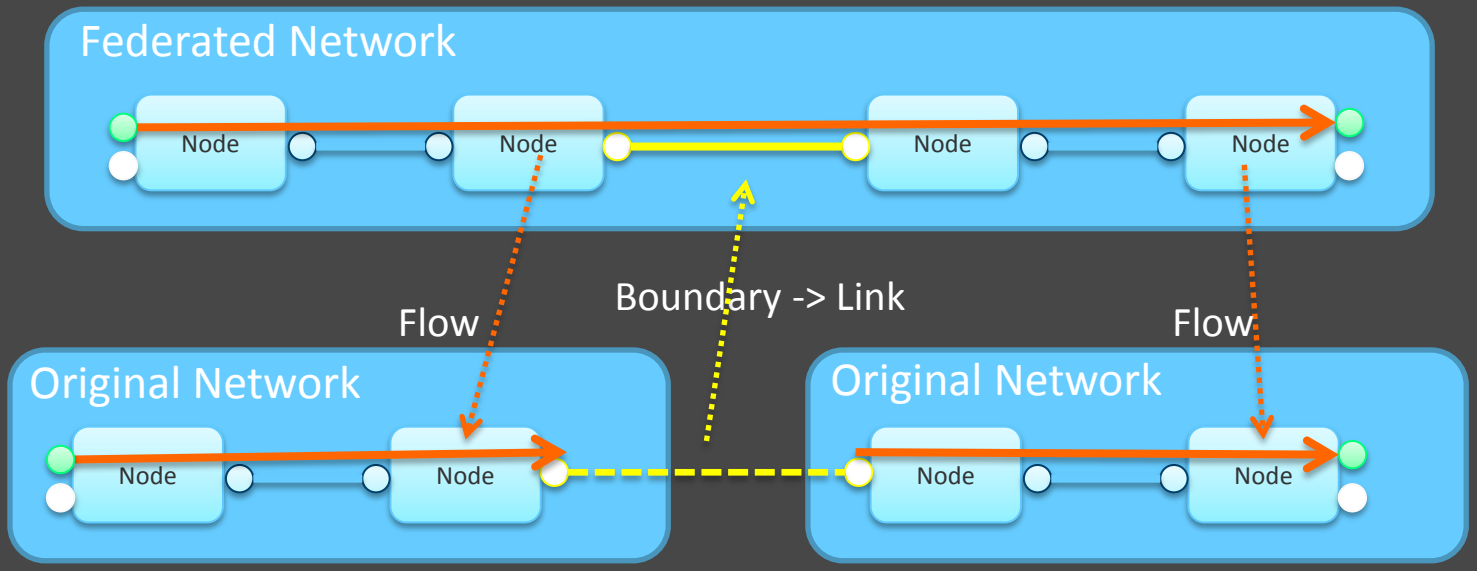
INPUT: Federated Network, Original Network, Boundary Setting (pair of port)

Converting Topology

1. Copy nodes and links in original network to federated network
2. Create a boundary link in federated network

Converting Flow

1. Divide flow into each original network and configure them



LinkLayerizer

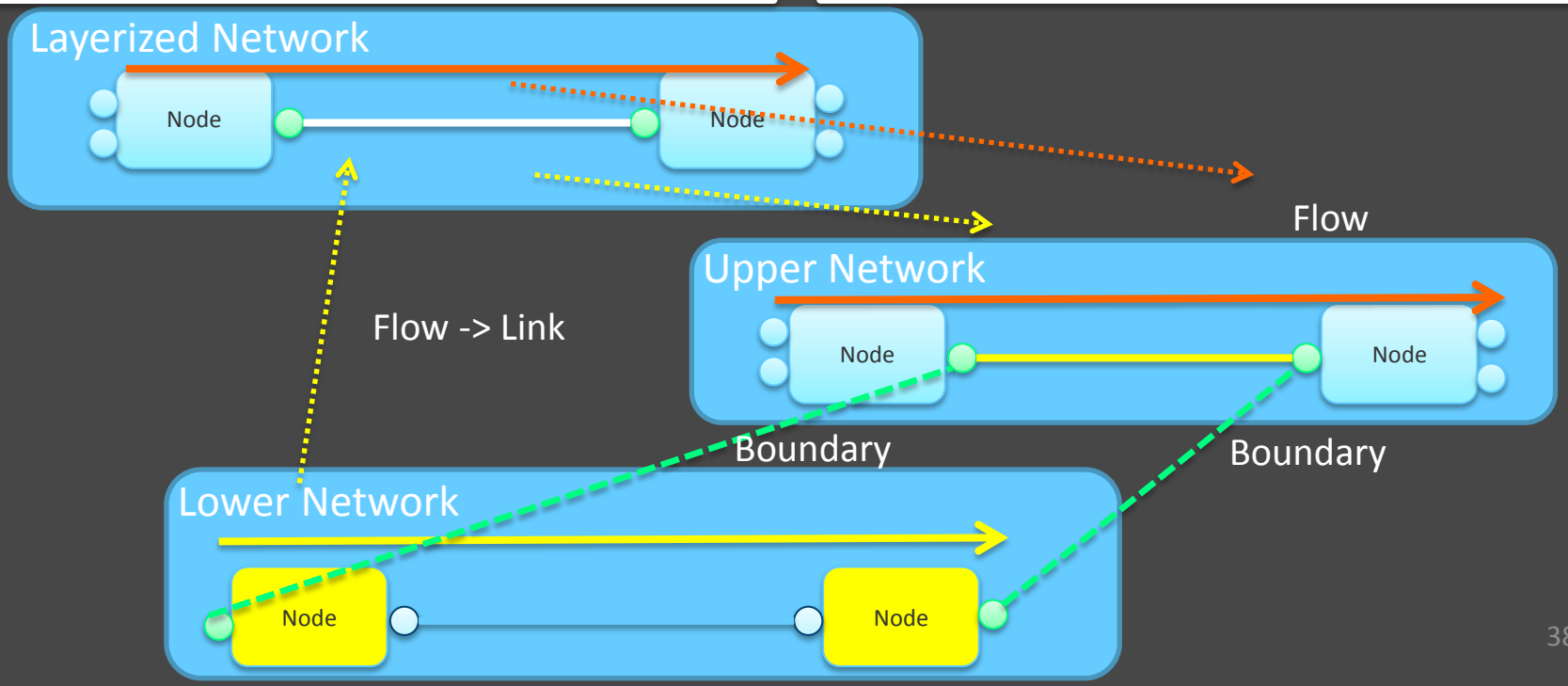
INPUT : Upper/Lower Network, Layerized Network, Boundary Setting(flow)

Converting Topology

1. Copy upper network nodes to layerized network
2. Convert the lower network flows to upper network links

Converting Flow

1. Configure flow to upper network



Slicer

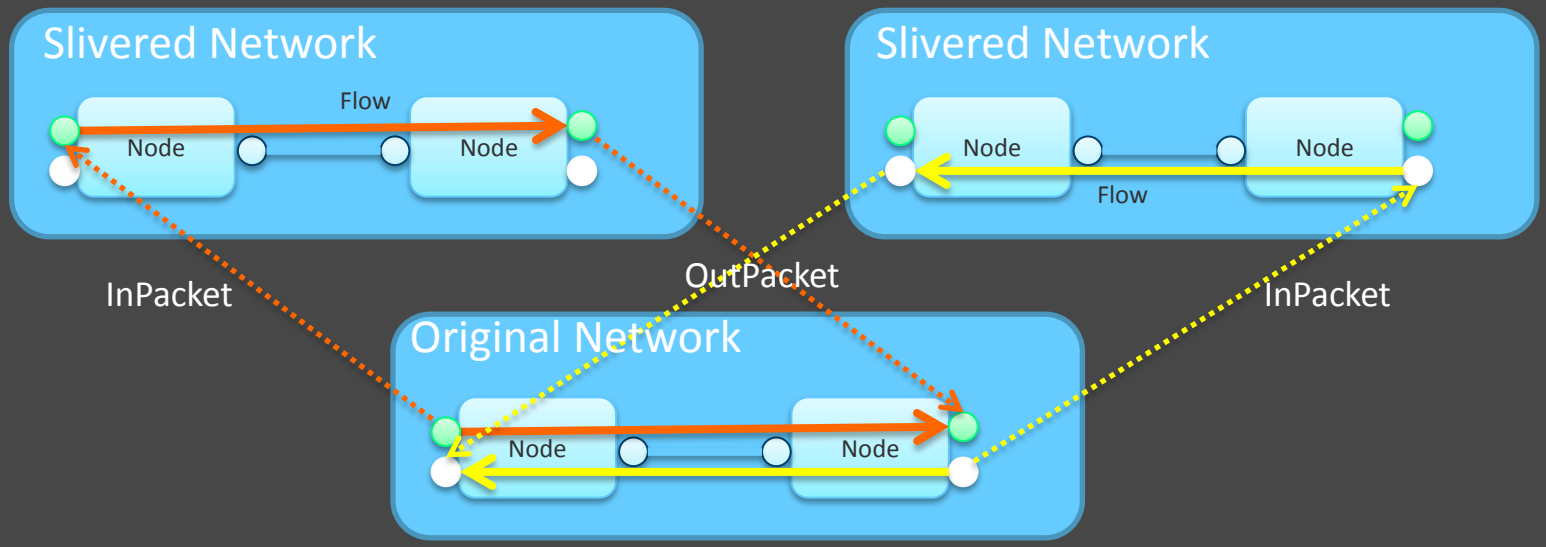
INPUT: Slivered Networks, Original Network, Slicing Policy(VLAN,MPLSTag,etc…)

Converting Topology

1. Copy nodes and links in original network to slivered networks

Converting Flow

1. Configure flow to the original NW with slicing tags



Today's Outline

- What's ODENOS 01
- ODENOS Design 02
  - Network Abstraction
  - Network Conversion
- ODENOS Implementation 03
- Hands-on 04

ODENOS Setup

- System Requirement
  - CPU: Intel x64 (with compatible)
  - Memory: >= 2GB
  - Ubuntu 14.04
- 構築方法
  - Dockerを使う場合（おすすめ）
    1. sudo apt-get install docker.io
    2. sudo docker pull odenos/odenos-handson
  - ベアメタルにインストールする場合（参考）
    - https://github.com/o3project/odenos/blob/develop/doc/QUICKSTART.md

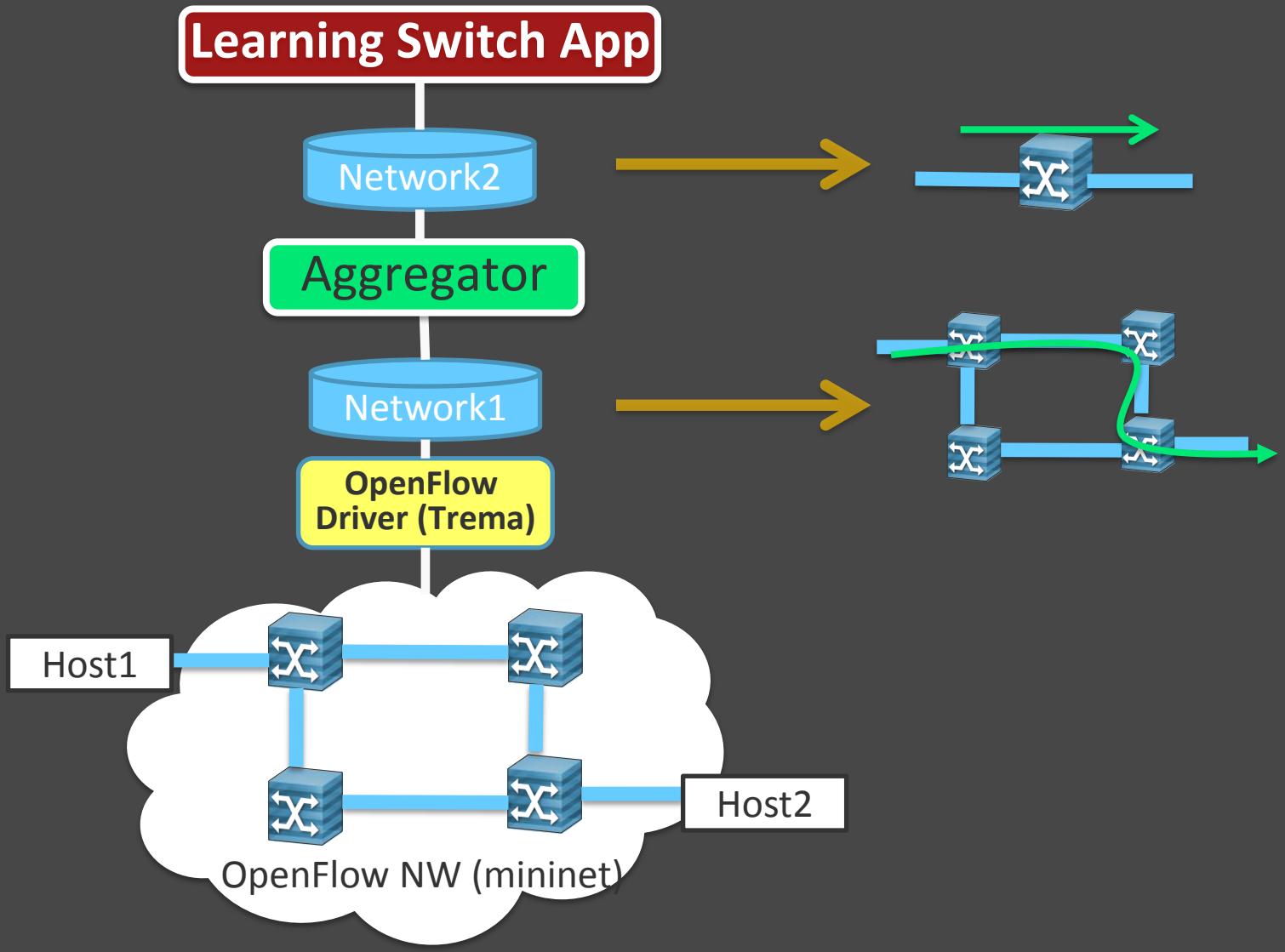
- セットアップが上手くいかなかった場合は、ハンズオンセッションにて対応します

Hands-On

- 目標
  - ODENOSを使ってOpenFlowスイッチ環境(mininet)を制御し、GUIで動作を確認しよう(apps/mininet_example)
    - Single network, multi nodeの構成をスクリプトで構築
  - Advanced
    - REST I/Fを使って構築
    - Flowを手動で投入/投入するアプリケーションを作成してみる
    - その他のapps/mininet_exampleを動かしてみる
    - Logicの組み合わせを変更するMininet(openvswitch)の代わりに他スイッチを制御する

なにか困ったら質問して下さい！

Single network, multi node構成



Single network, multi node構成

- DockerでOdenOSを試すためにはコンテナのホストマシンにOpenvswitchがインストールされている必要あるので注意
  - sudo apt-get install openvswitch-switch
  - sudo modprobe openvswitch

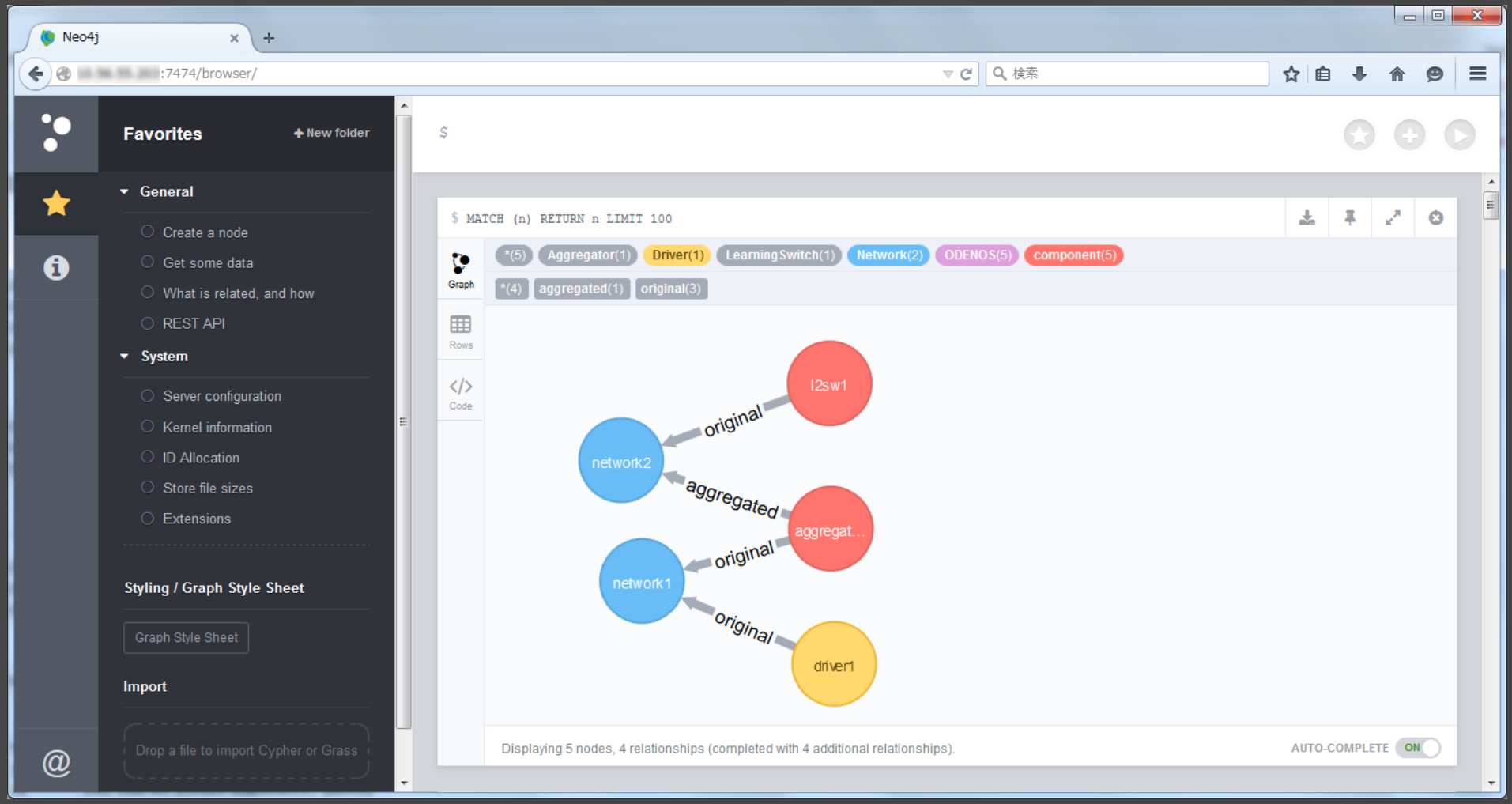
- Dockerで構築した場合の起動手順

1. sudo docker run -it --privileged=true -p 7474:7474 -p 10080:10080 odenos/odenos-handson
2. cd odenos/apps/mininet_examples/single_network_control
3. ./start_odenos.sh restart
4. ./start_mininet.py

動作の確認方法

- Mininetを使ってpingが通ることを確認
  - mininet> h1 ping h2
- REST APIを使ってcomponentが登録されていることを確認
  - curl http://localhost:10080/systemmanager/components
  - curl http://localhost:10080/systemmanager/connections
  - Other example:
    - https://github.com/o3project/odenos/tree/develop/apps/mininet_examples/single_network_control

コンポーネント接続の可視化 (Neo4j_Adapter)



Neo4j_Adapterを使った可視化

- ODENOSには、標準でデータをNeo4jに書き出すAdapterが添付されています

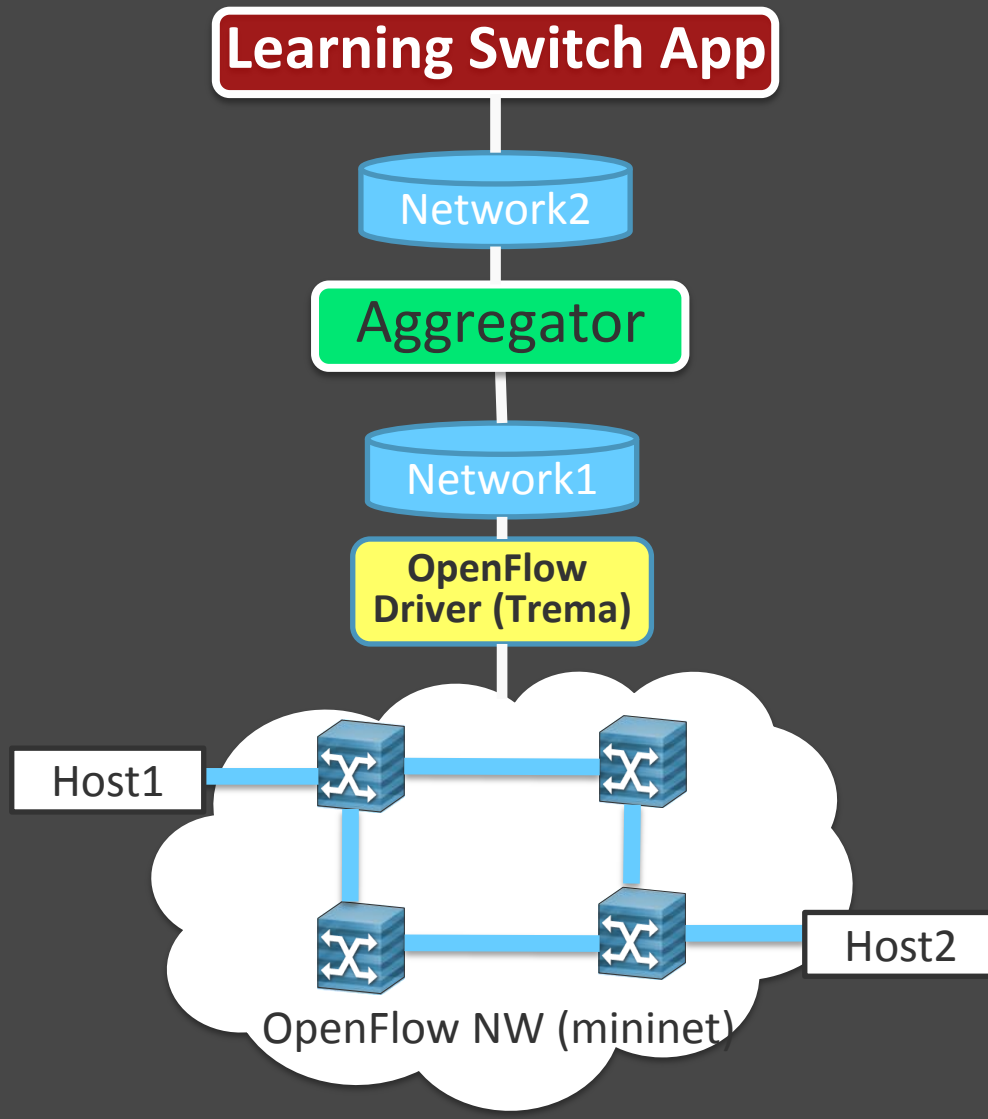
1. Neo4jにODENOSのデータを反映
   - cd ~/odenos ; PYTHONPATH=./lib/python/ ./apps/neo4j/neo4jsync.py
   - cd ~/odenos ; PYTHONPATH=./lib/python/ ./apps/neo4j/neo4jsync.py topology
     - 各NW Componentのトポロジも可視化する場合はこちら
2. Neo4jの標準ブラウザで確認
   1. Graph StyleSheetをブラウザから登録
      1. https://raw.githubusercontent.com/o3project/odenos/develop/apps/neo4j/graphstyle.grass
   2. http://localhost:7474 にアクセス

REST I/Fを用いた構築方法(L2SW)

start_odenos.sh

1. ODENOS Coreの起動
   - MessageBus
   - System Manager
   - Remote Object Manager
2. Componentの作成
   - Network Component
   - Learning Switch App
   - Driver
   - Aggregator
3. Component間の接続

REST I/Fで操作可能



REST I/Fを用いた構築方法(L2SW)

必要な手順

1. ODENOS Coreの起動
    1. cd ~/odenos ;
    2. ./odenos start –c ~/odenos/apps/mininet_examples/single_network_control/odenos.conf
2. Componentの作成(後ページで説明)
    - /systemmanager/components
3. Component間の接続(後ページで説明)
    - /systemmanager/connections
4. Neo4jへのデータ反映(可視化)
    - cd ~/odenos ;
    - PYTHONPATH=./lib/python/ ./apps/neo4j/neo4jsync.py topology
5. Mininetの起動
    - cd ~/odenos/apps/mininet_examples/single_network_control/ ;
    - ./start_mininet.py

REST I/Fを用いた構築方法(L2SW)
- Componentの作成 -

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/components/lsw -X PUT -d
'{"type": "LearningSwitch", "id": "lsw"}‘
```

Learning Switch App

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/components/network0 -X
PUT -d '{"type": "Network", "id": "network1"}'
```

Network2

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/components/agg -X PUT -d
'{"type": "Aggregator", "id": "agg"}'
```

Aggregator

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/components/network1 -X
PUT -d '{"type": "Network", "id": "network0"}'
```

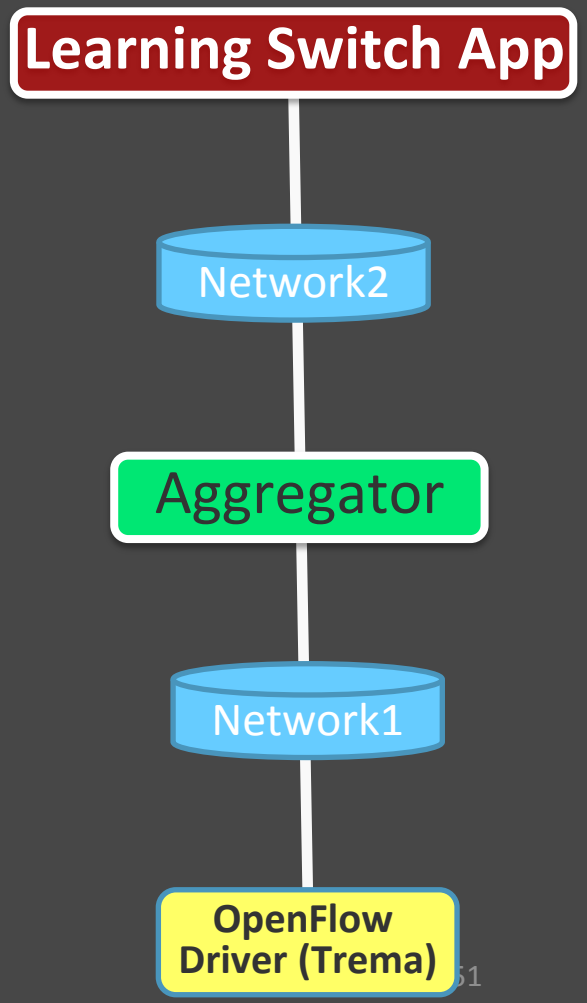
Network1

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/components/ofd -X PUT -d
'{"type": "OpenFlowDriver", "id": "ofd"}'
```

OpenFlow Driver (Trema)

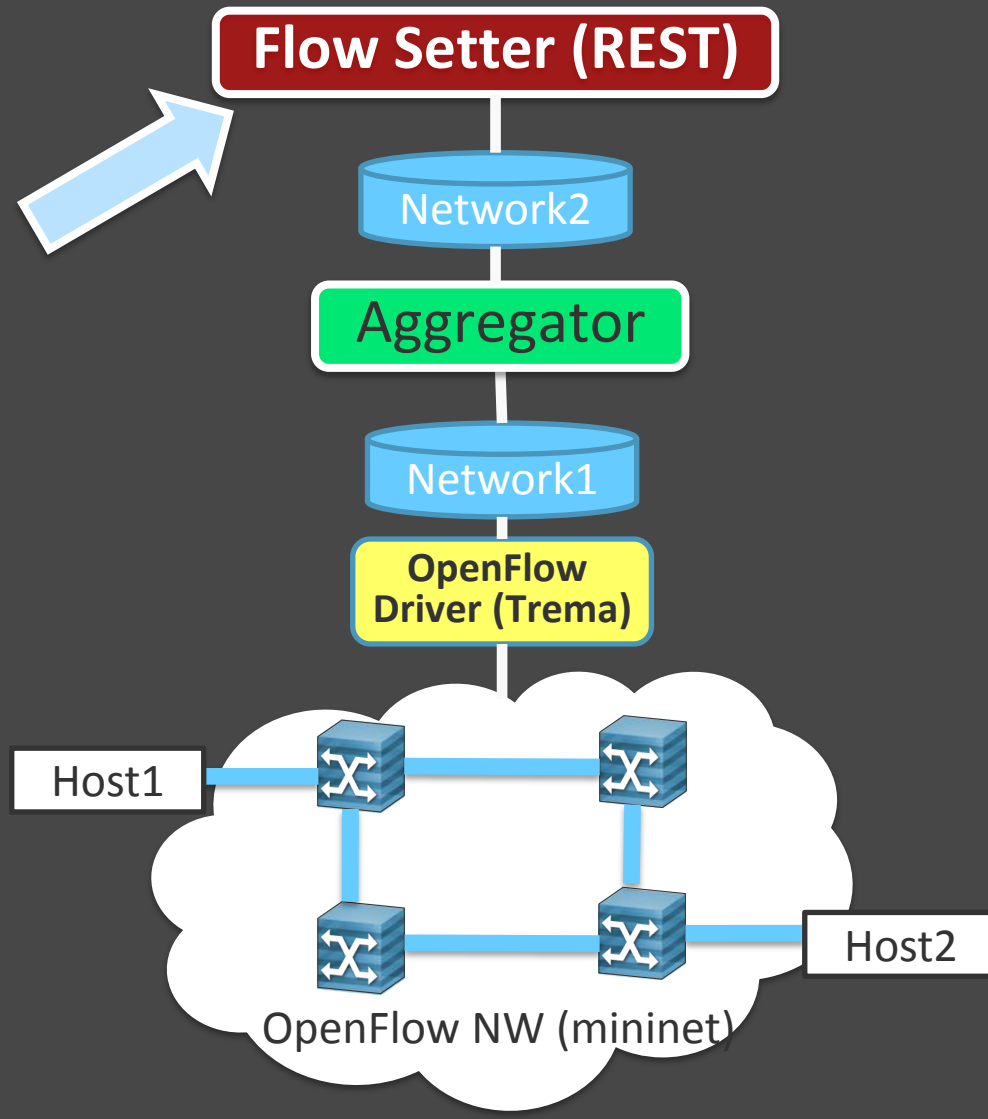
REST I/Fを用いた構築方法(L2SW)
- Component間接続の作成 -

```
curl -w "$FORMAT"
http://localhost:10080/systemmanager/connections -X POST -d
'{"id": "conn0", "type": "LogicAndNetwork",
"connection_type":"original", "logic_id": "lsw",
"network_id":"network1"}'
curl -w "$FORMAT"
http://localhost:10080/systemmanager/connections -X POST -d
'{"id": "conn1", "type": "LogicAndNetwork",
"connection_type":"aggregated", "logic_id": "agg",
"network_id":"network1"}'
curl -w "$FORMAT"
http://localhost:10080/systemmanager/connections -X POST -d
'{"id": "conn2", "type": "LogicAndNetwork",
"connection_type":"original", "logic_id": "agg",
"network_id":"network0"'}
curl -w "$FORMAT"
http://localhost:10080/systemmanager/connections -X POST -d
'{"id": "conn3", "type": "LogicAndNetwork",
"connection_type":"original", "logic_id": "ofd",
"network_id":"network0"}'
```



REST I/Fを用いた構築方法(Flowset)

L2Switchが設定していたFlowをRESTを使って設定してみよう！



REST I/Fを用いた構築方法(Setter)
- Componentの作成 -

#Network2に設定するFlow情報(設定方法は以下などを参考にして下さい)

https://github.com/o3project/odenos/blob/develop/doc/api/index.md

curl -w "$FORMAT" http://localhost:10080/network1/flows/flow01 -X PUT -d
'{"flow_id":"flow01","owner":"","enabled":true,"attributes":{"latency":"0",
"req_latency":"0",
"bandwidth":"0"},"type":"OFPFlow","idle_timeout":90,"hard_timeout":90,"matches":[{"type":"OFPFlowMatch","in_node":"agg","in_port":"node0x3_port3@0x3"}],"path":[],"edge_actions":{"agg":[{"type":"FlowActionOutput","output":"node0x1_port3@0x1"}]}}‘

curl -w "$FORMAT" http://localhost:10080/network1/flows/flow02 -X PUT -d
'{"flow_id":"flow02","owner":"","enabled":true,"attributes":{"latency":"0",
"req_latency":"0",
"bandwidth":"0"},"type":"OFPFlow","idle_timeout":90,"hard_timeout":90,"matches":[{"type":"OFPFlowMatch","in_node":"agg","in_port":"node0x1_port3@0x1"}],"path":[],"edge_actions":{"agg":[{"type":"FlowActionOutput","output":"node0x3_port3@0x3"}]}}'

Hands-On

- 目標
  - ODENOSを使ってOpenFlowスイッチ環境(mininet)を制御し、GUIで動作を確認しよう(apps/mininet_example)
    - Single network, multi nodeの構成をスクリプトで構築
  - Advanced
    - REST I/Fを使って構築
    - Flowを手動で投入/投入するアプリケーションを作成してみる
    - その他のapps/mininet_exampleを動かしてみる
    - Logicの組み合わせを変更するMininet(openvswitch)の代わりに他スイッチを制御する

なにか困ったら質問して下さい！