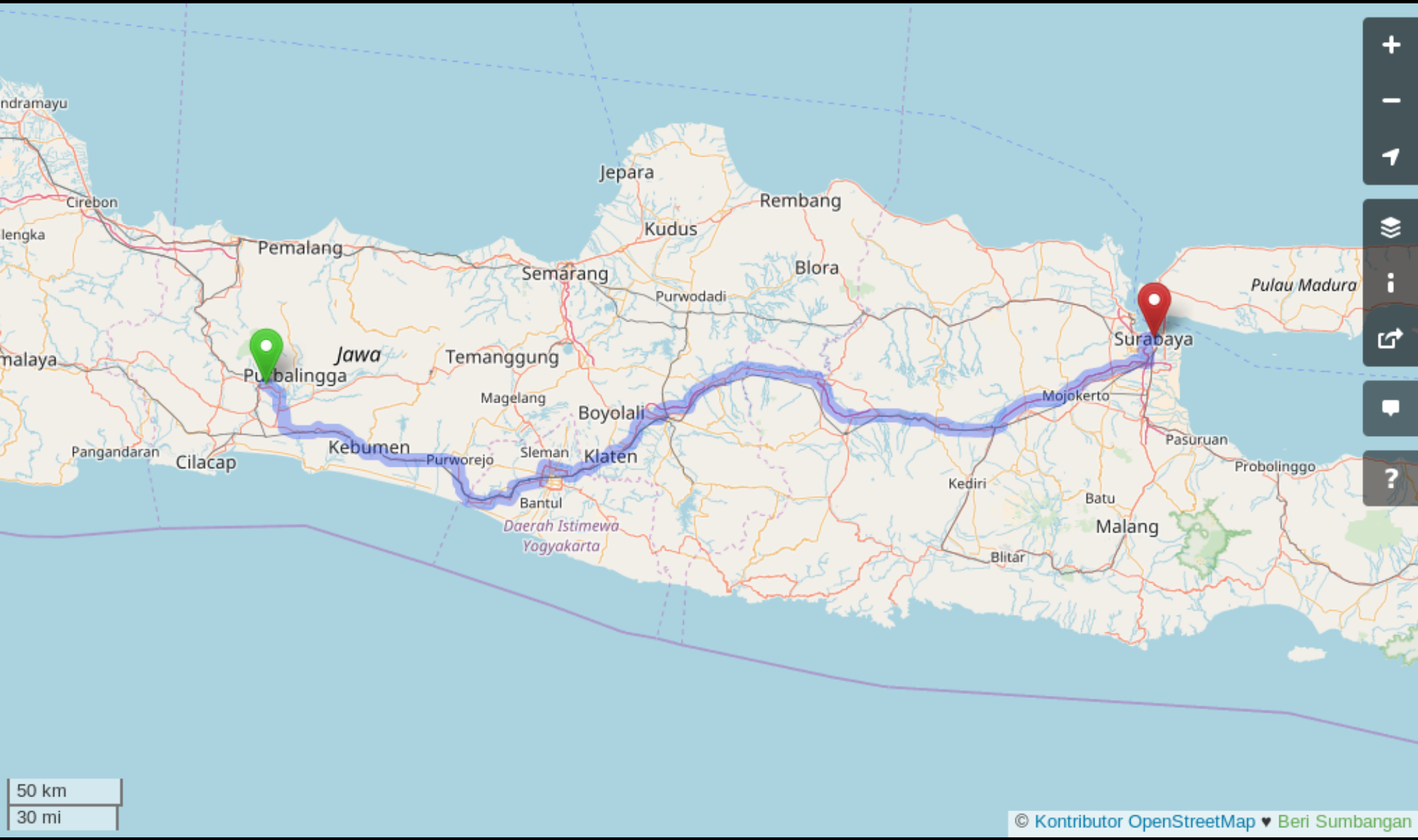
# Halo!

# Iwan stwn

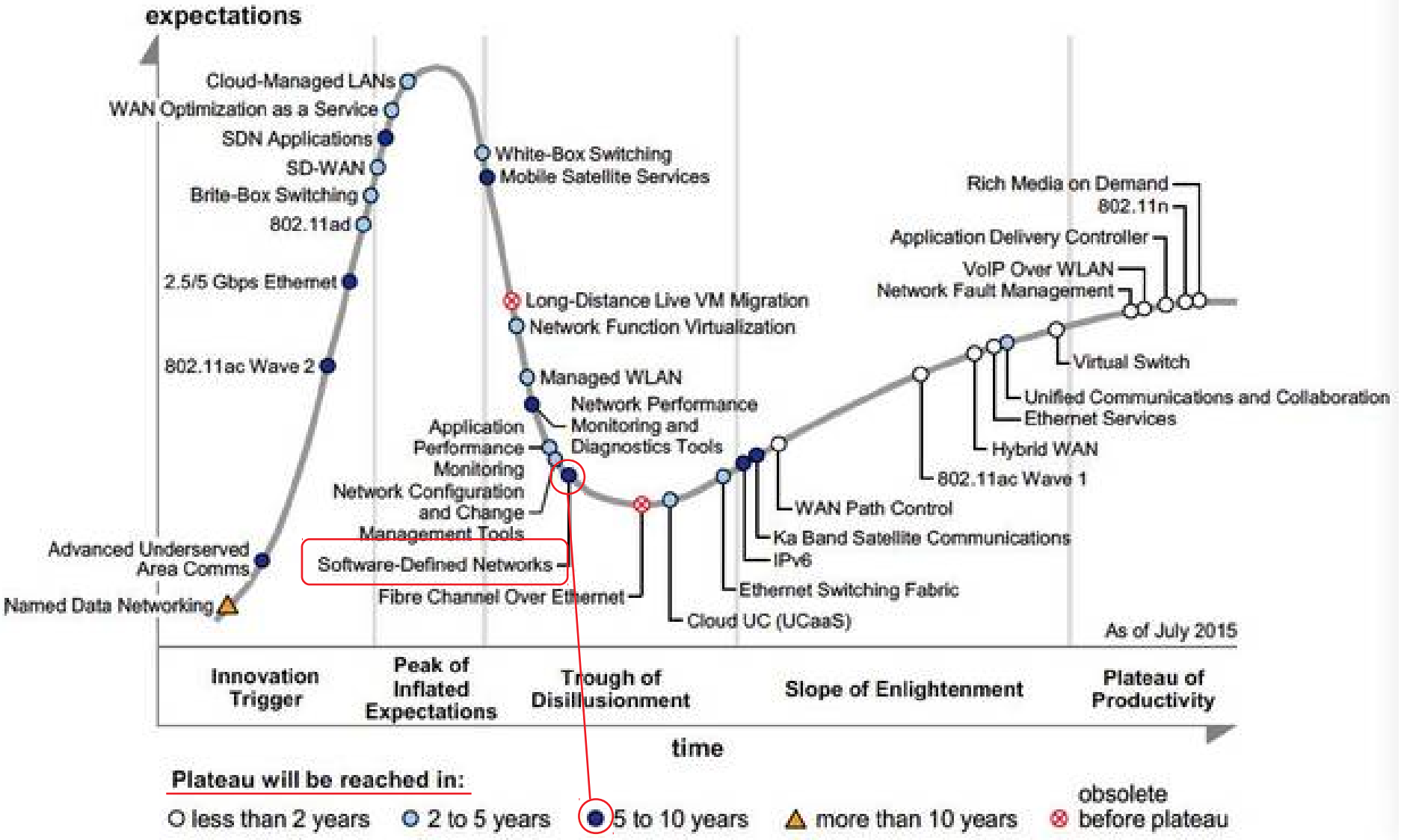

Jepara
Rembang
Kudus
Pemalang
Semarang
Blora
Purwodadi
Pulau Madura
Surabaya
Jawa
Temanggung
Magelang
Boyolali
Mojokerto
Kebumen
Purworejo
Sleman
Klaten
Pasuruan
Probolinggo
Cilacap
Kediri
Batu
Malang
Bantul
Daerah Istimewa Yogyakarta
Blitar
50 km
30 mi
© Kontributor OpenStreetMap ♥ Beri Sumbangan

# Who am I?

- A teaching staff at Department of Electrical Engineering, Universitas Jenderal Soedirman
- Interested in systems and networks
  - Dabbles in (Cluster) **Computing**
  - (Re)learning **Networking** and related advances
- More of a sysadmin/netadmin
  - Starting to learn Python

# Software-Defined Networking

Iwan Setiawan <stwn at unsoed.ac.id>

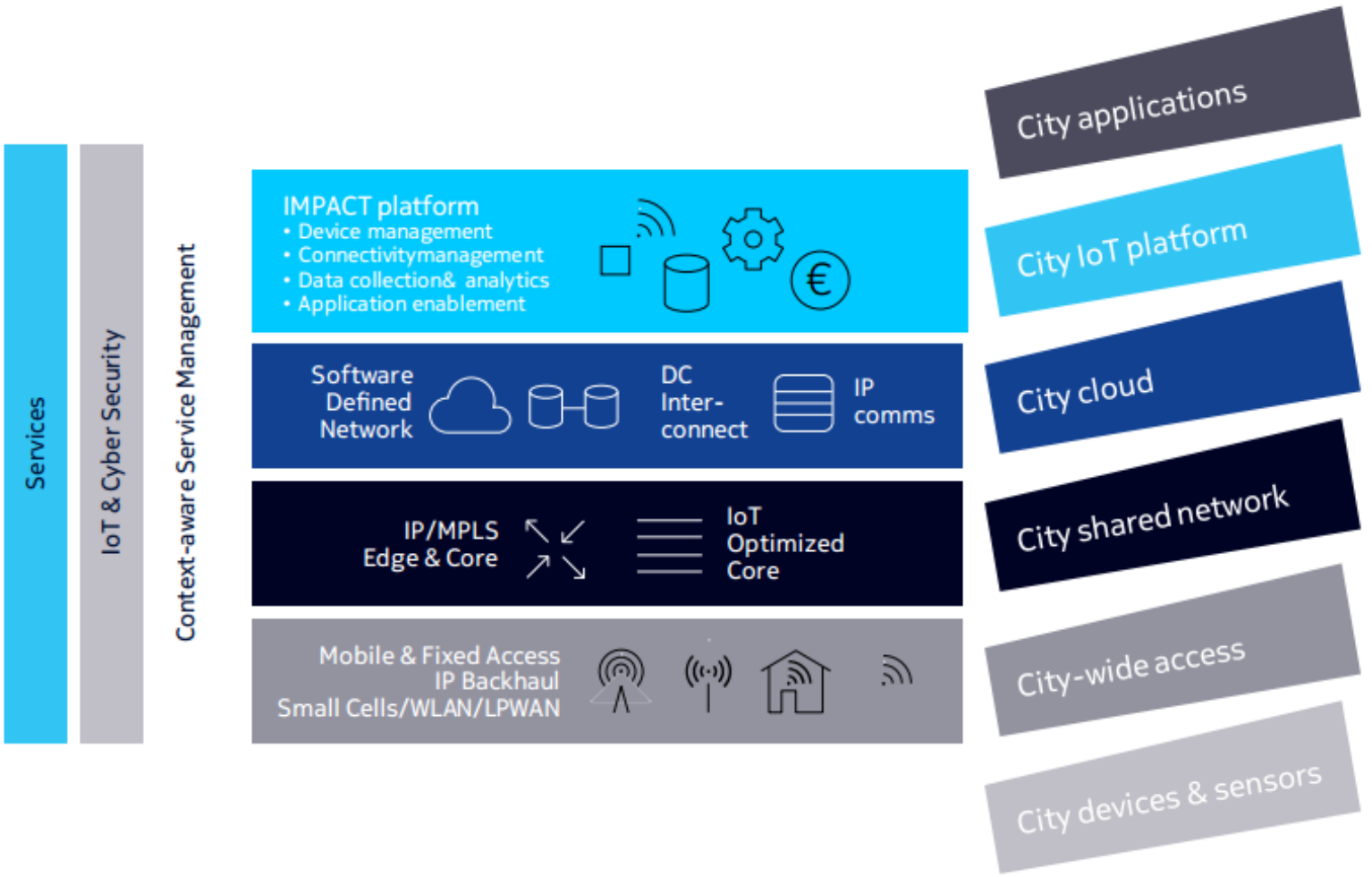



Source: Gartner (July 2015)

# Trends on SDN

- Gartner’s Hype Cycle for Net. and Commun. 2015: SDN is one of the technologies that would reach plateau in 5-10 years
- AT&T has a plan to migrate its networks to SDN by 75% in 2020
- Organizations have been using SDN in their networks for research and operation

# Nokia Smart City Building Blocks



# Software-Defined Networking

Iwan Setiawan <stwn at unsoed.ac.id>

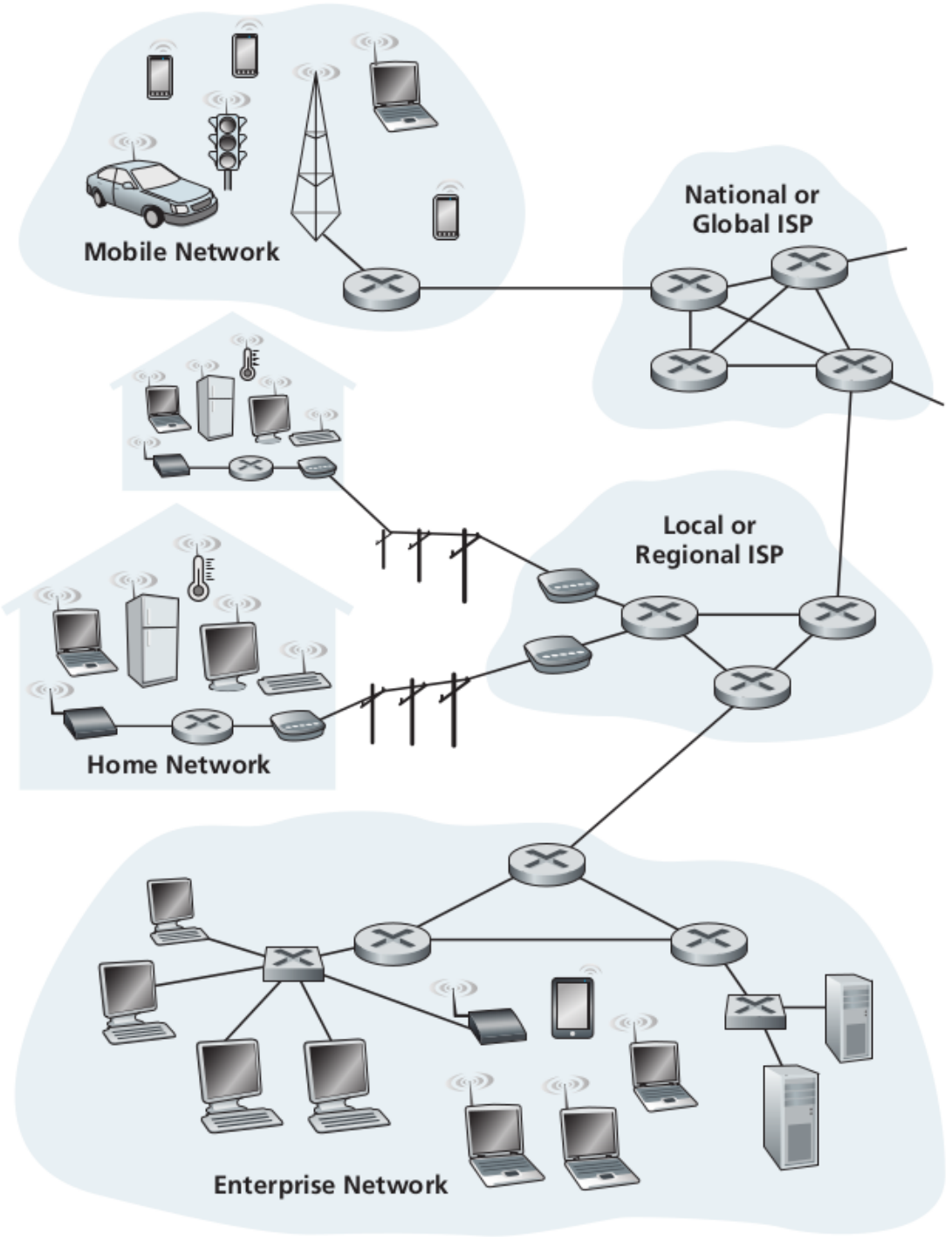
# Python ~~Software~~-Defined Networking

Iwan Setiawan <stwn at unsoed.ac.id>

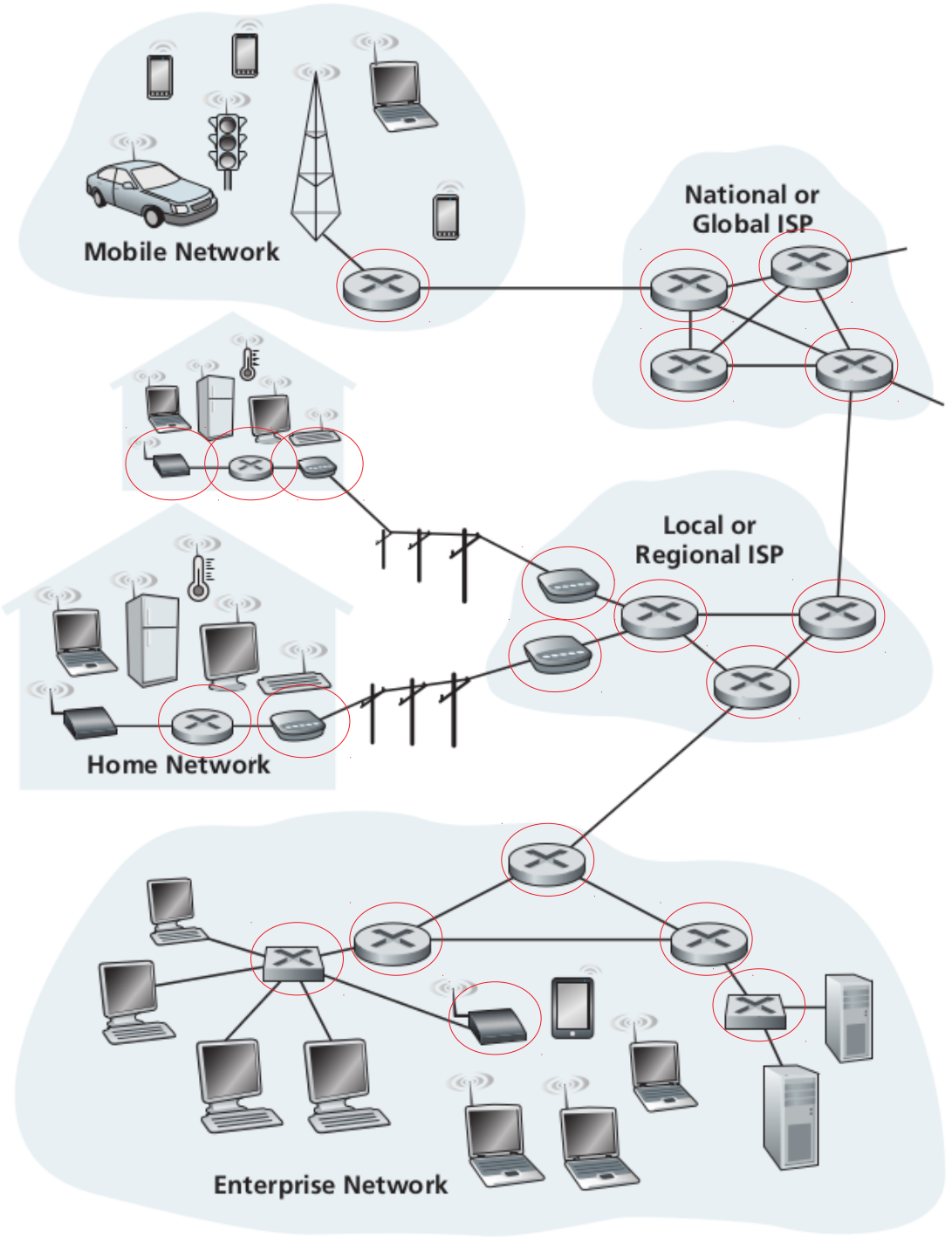
PYCON 2017 - Surabaya, Indonesia

UNIVERSITAS JENDERAL SOEDIRMAN 1963

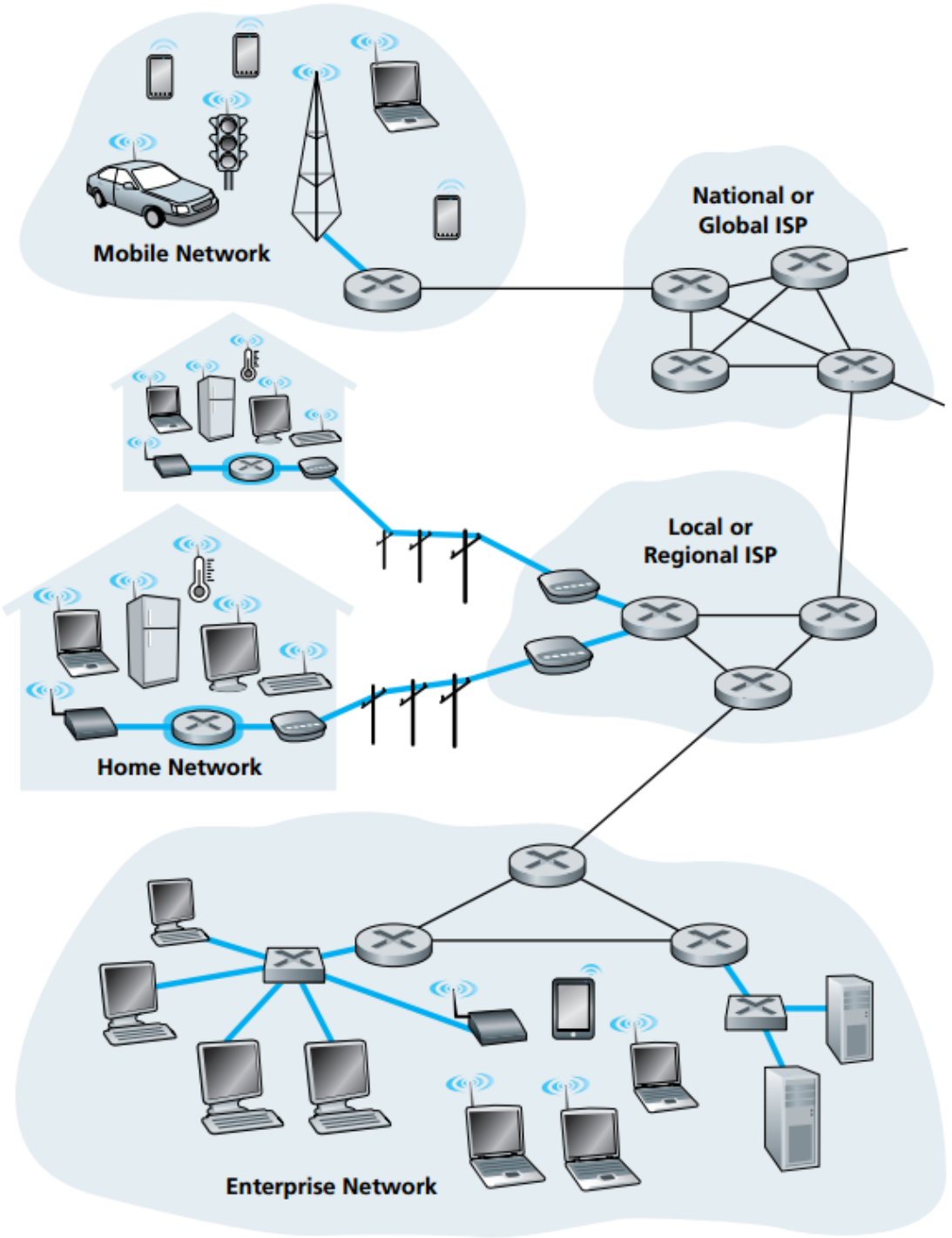
# Networking



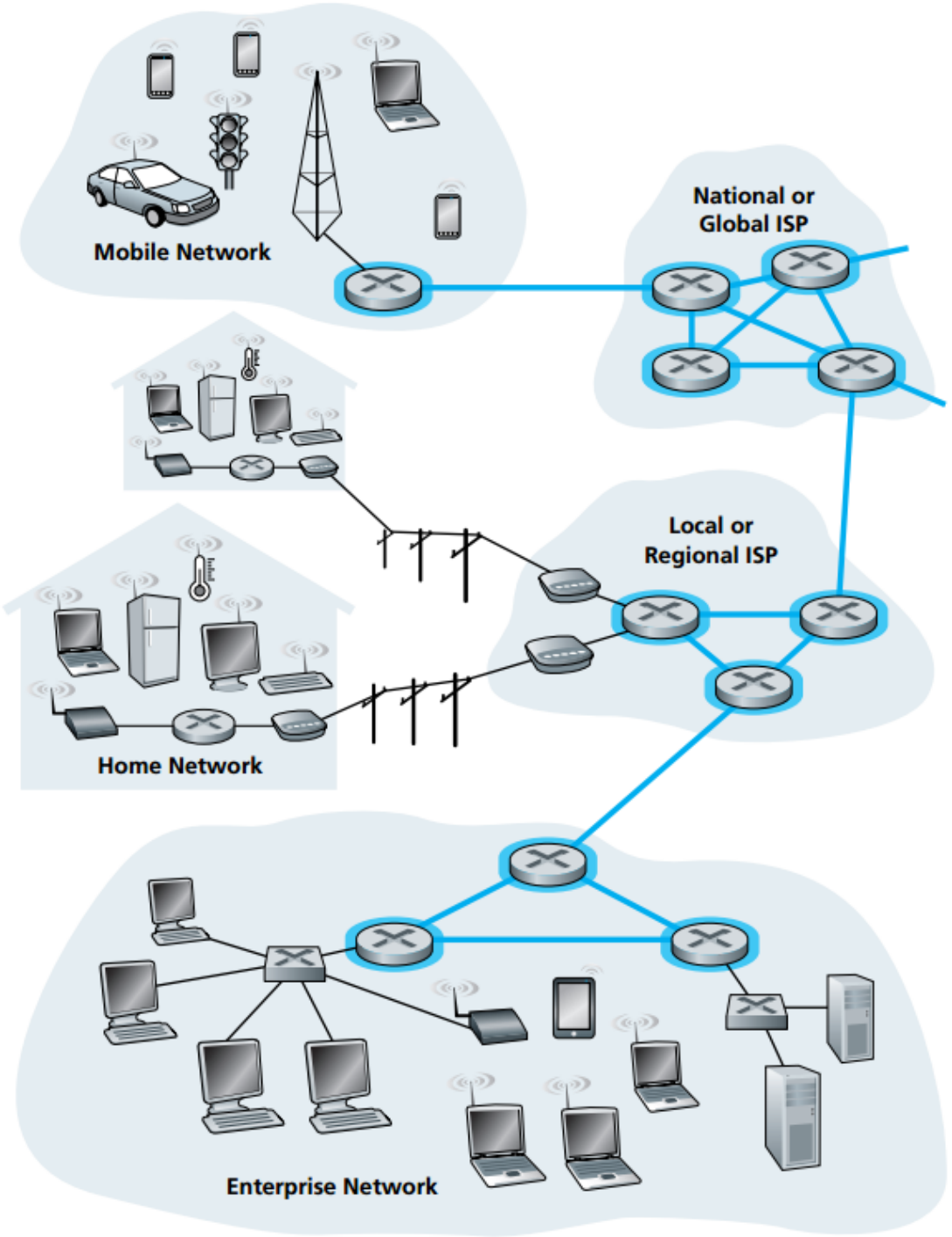

Kurose and Ross (2017)



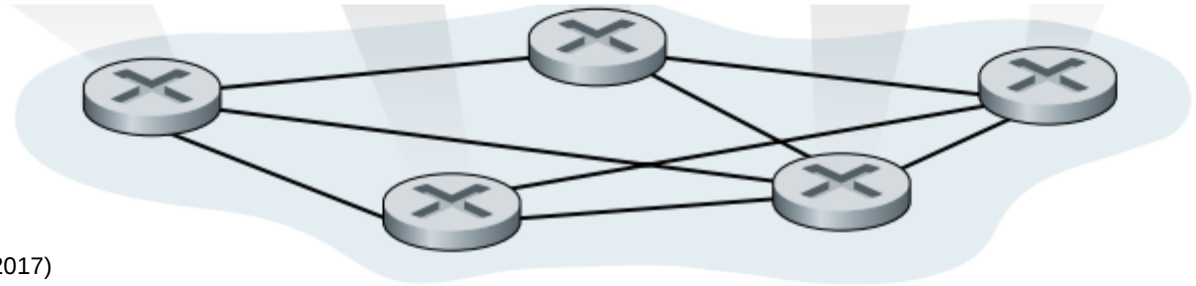

Kurose and Ross (2017)



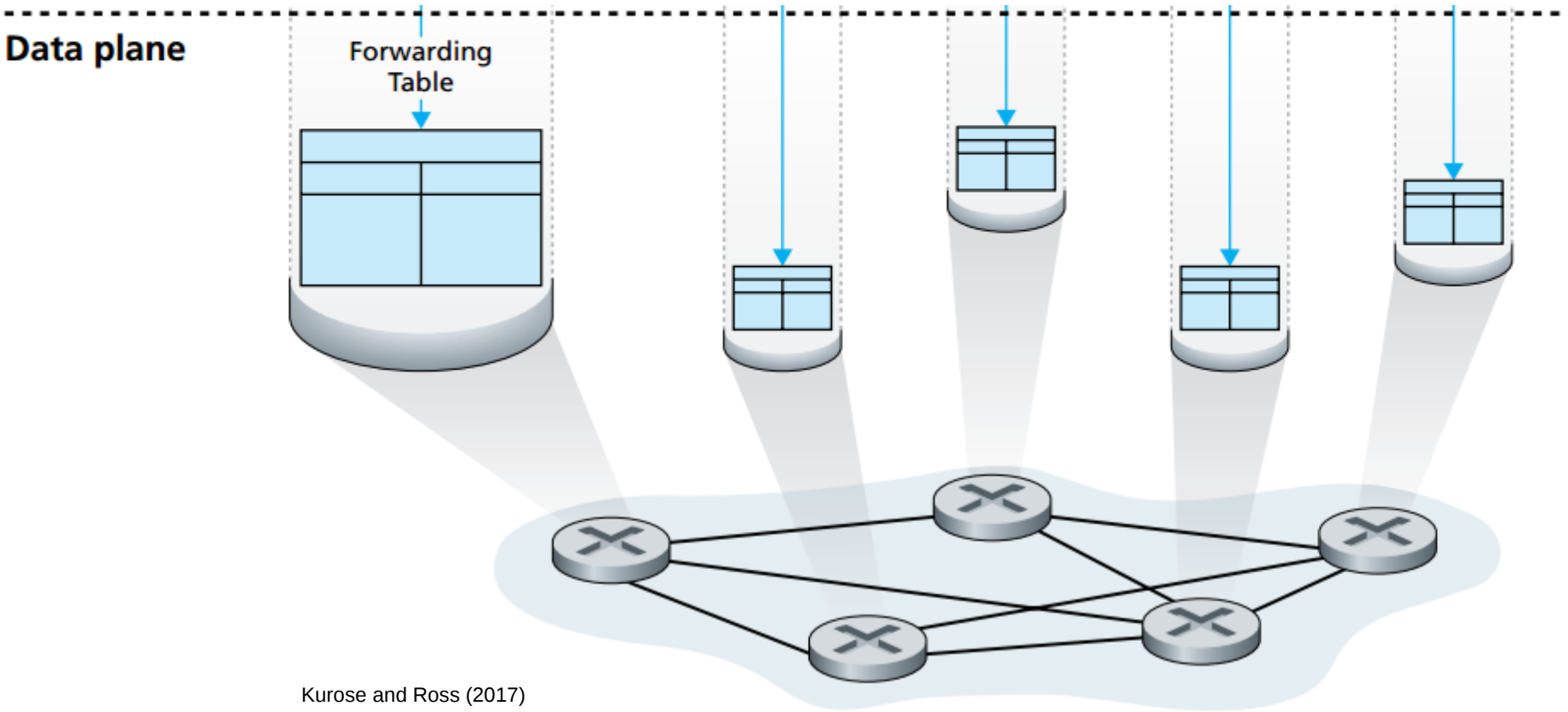

Kurose and Ross (2017)



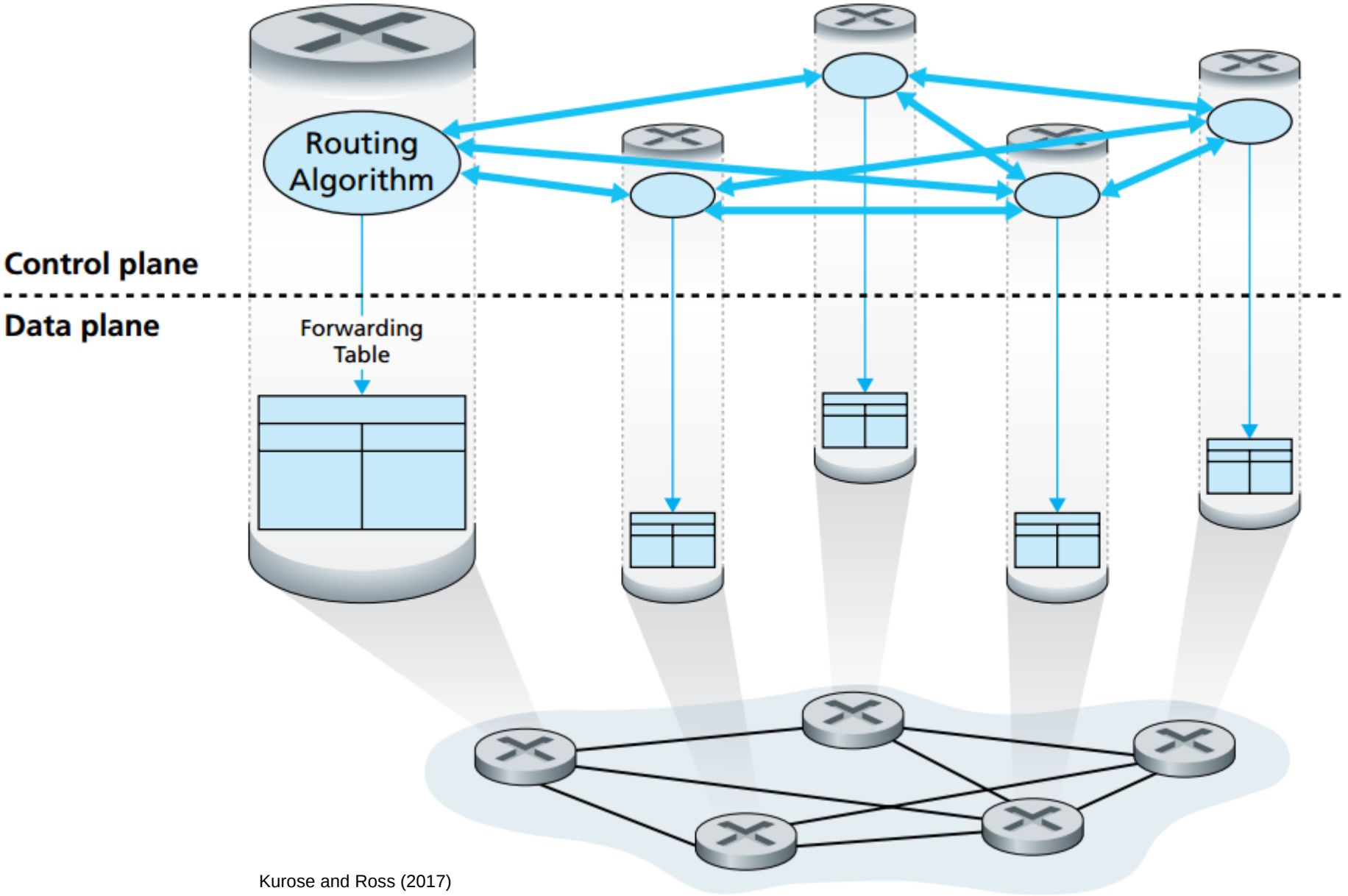

Kurose and Ross (2017)



Kurose and Ross (2017)



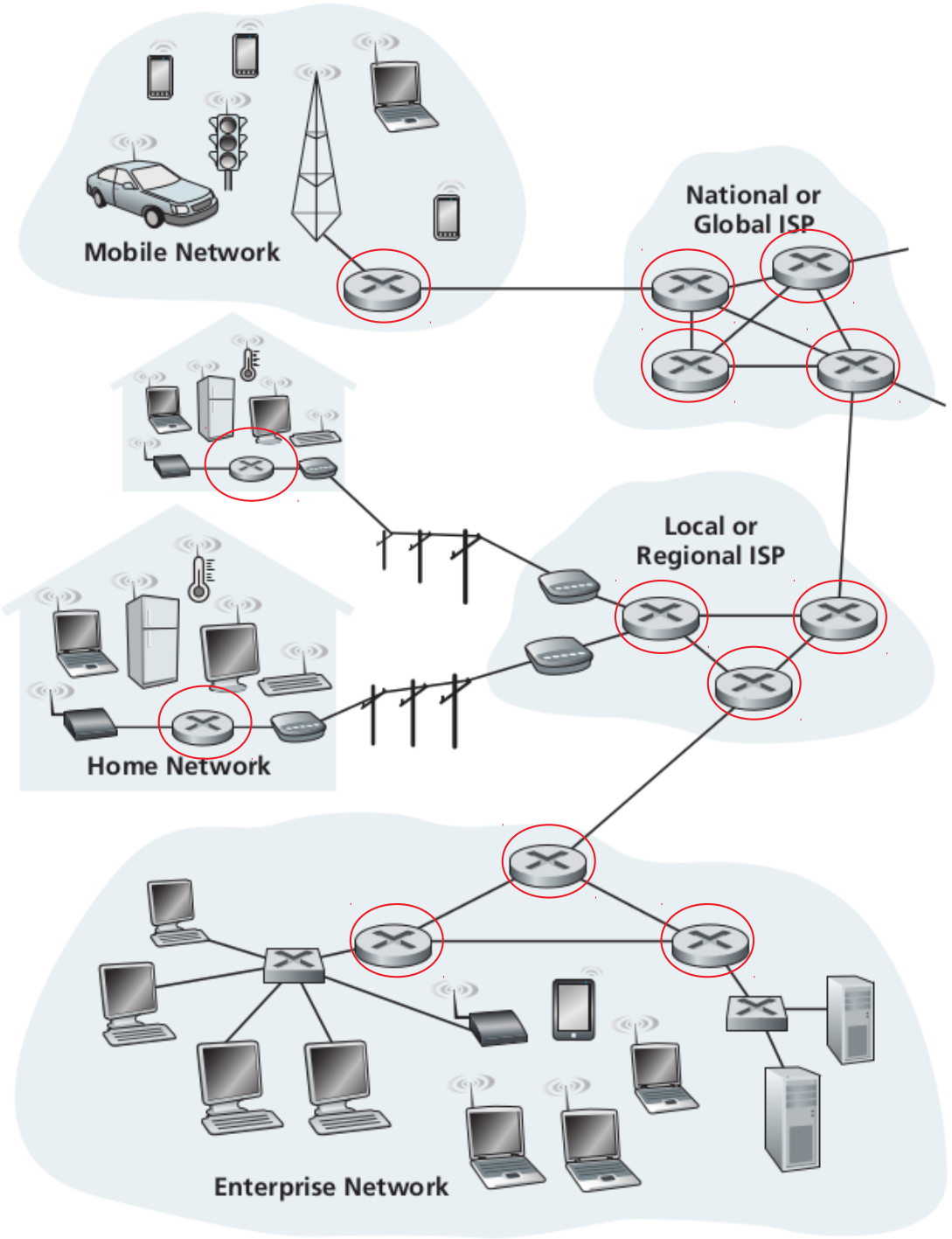

Kurose and Ross (2017)




Kurose and Ross (2017)

# Two Planes in Networking

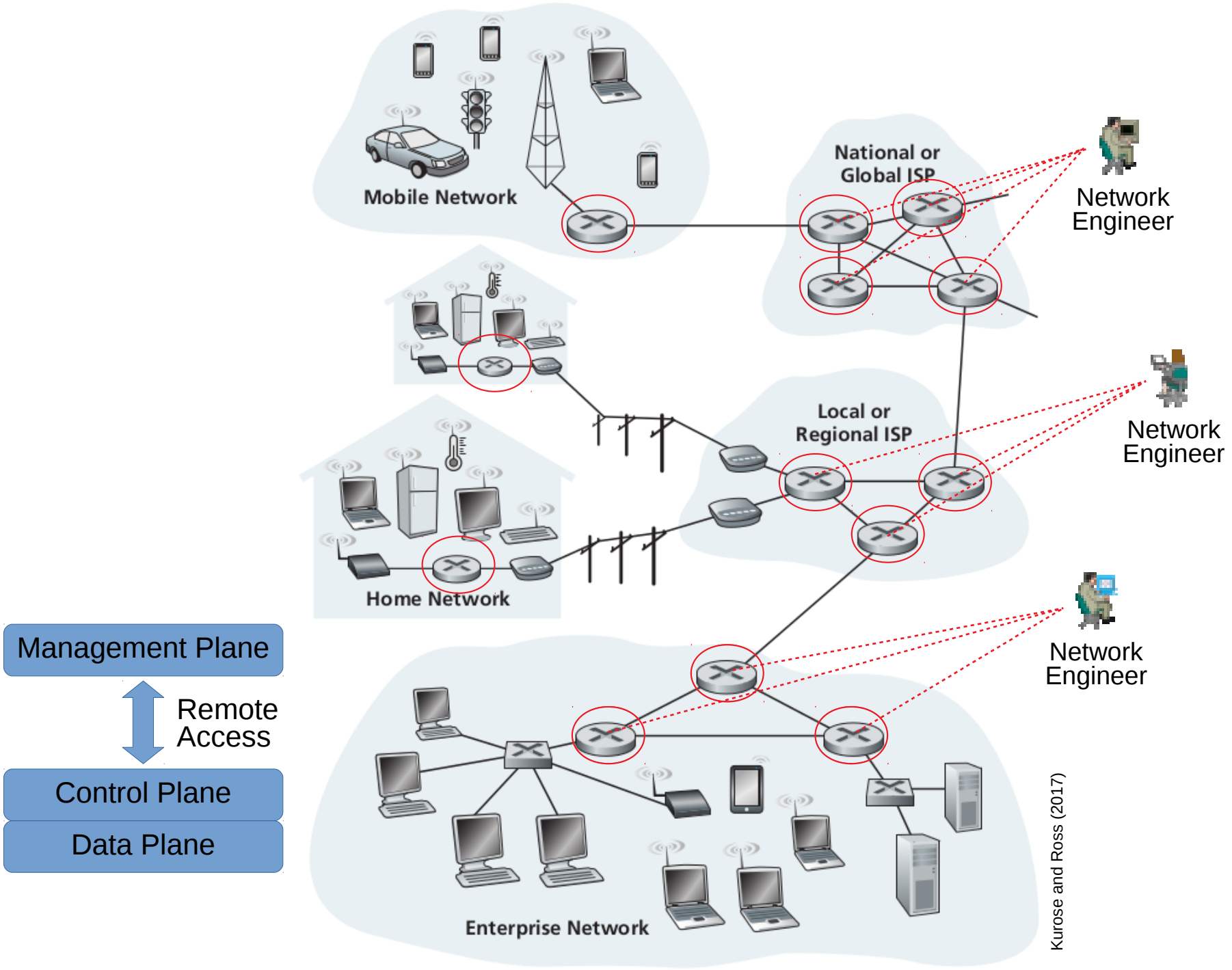
- **Data plane**: forwarding elements/devices
  - Network switch: inbound-outbound forwarding
  - Hardware-based frame switching (ASIC)
- **Control plane**: logic control, algorithm
  - Routing: data path that should be taken by packets
  - Software-based packet switching by processing packet, e.g. L3 packet
- The two planes are coupled in network devices, such as routers or L3 devices

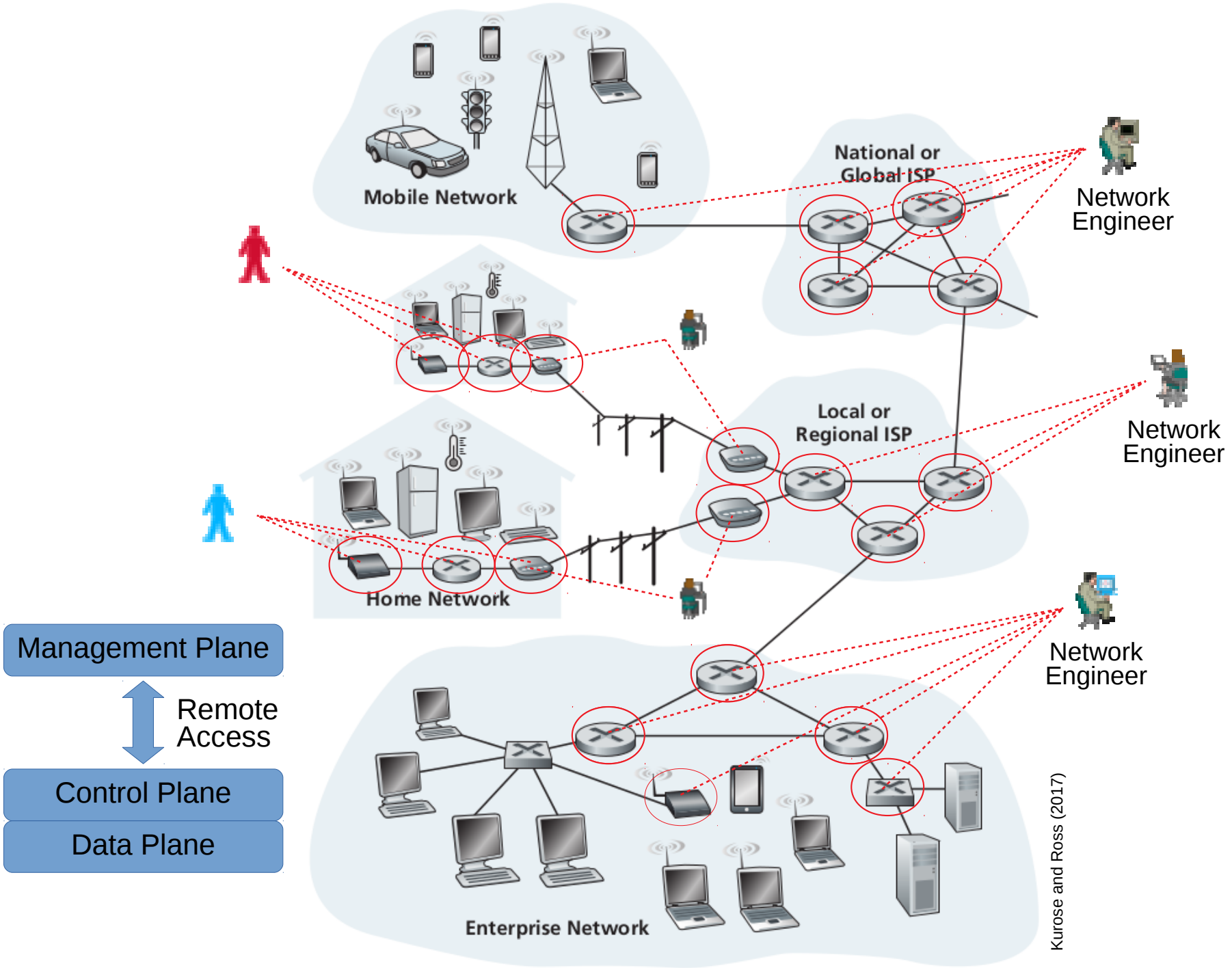



Control Plane

Data Plane

Kurose and Ross (2017)

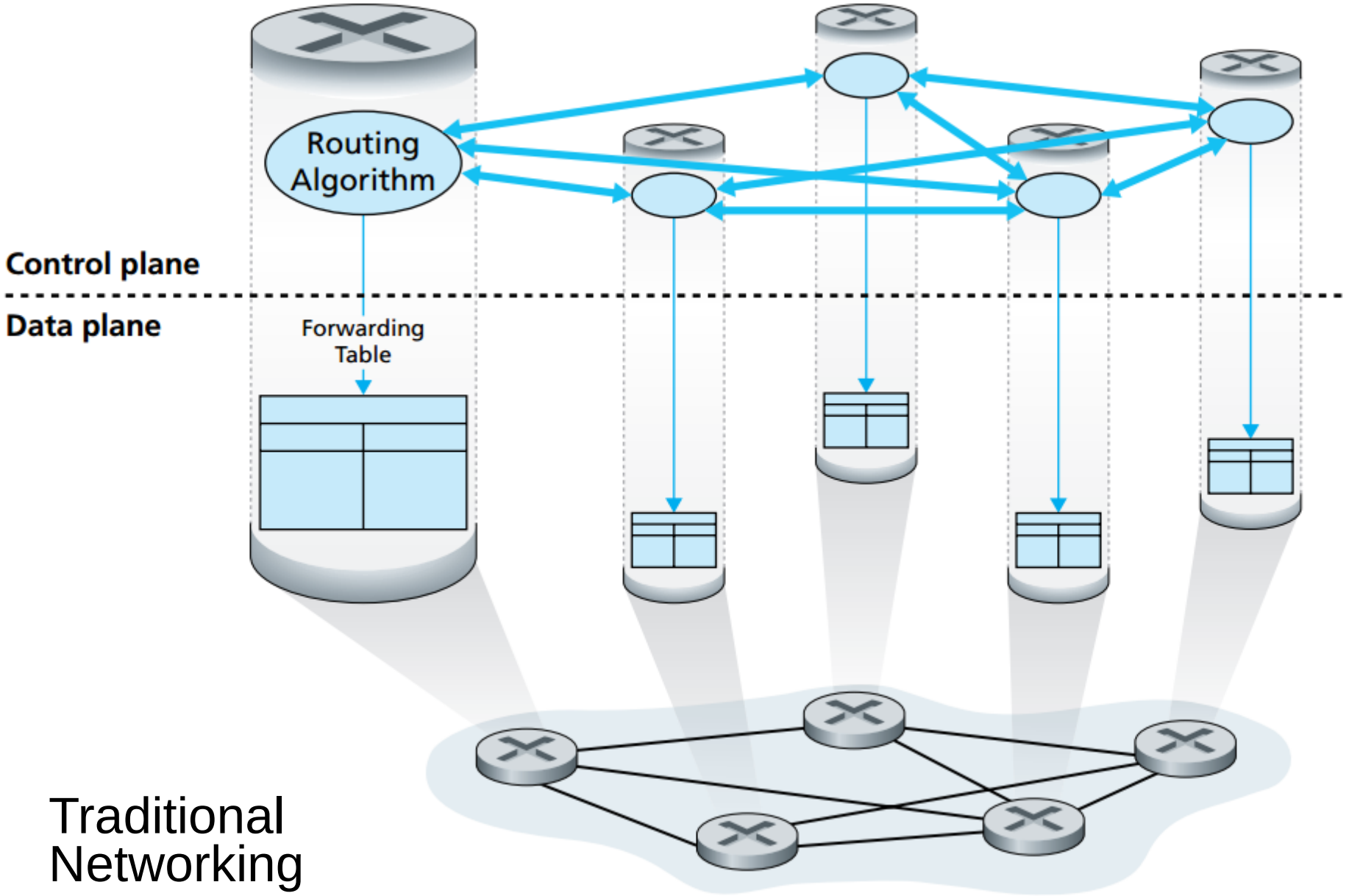

Mobile Network
National or
Global ISP
Network
Engineer
Local or
Regional ISP
Network
Engineer
Home Network
Network
Engineer
Enterprise Network
Management Plane
Remote
Access
Control Plane
Data Plane
Kurose and Ross (2017)


Mobile Network
National or Global ISP
Network Engineer
Local or Regional ISP
Network Engineer
Home Network
Network Engineer
Enterprise Network
Management Plane
Remote Access
Control Plane
Data Plane
Kurose and Ross (2017)

# Issues

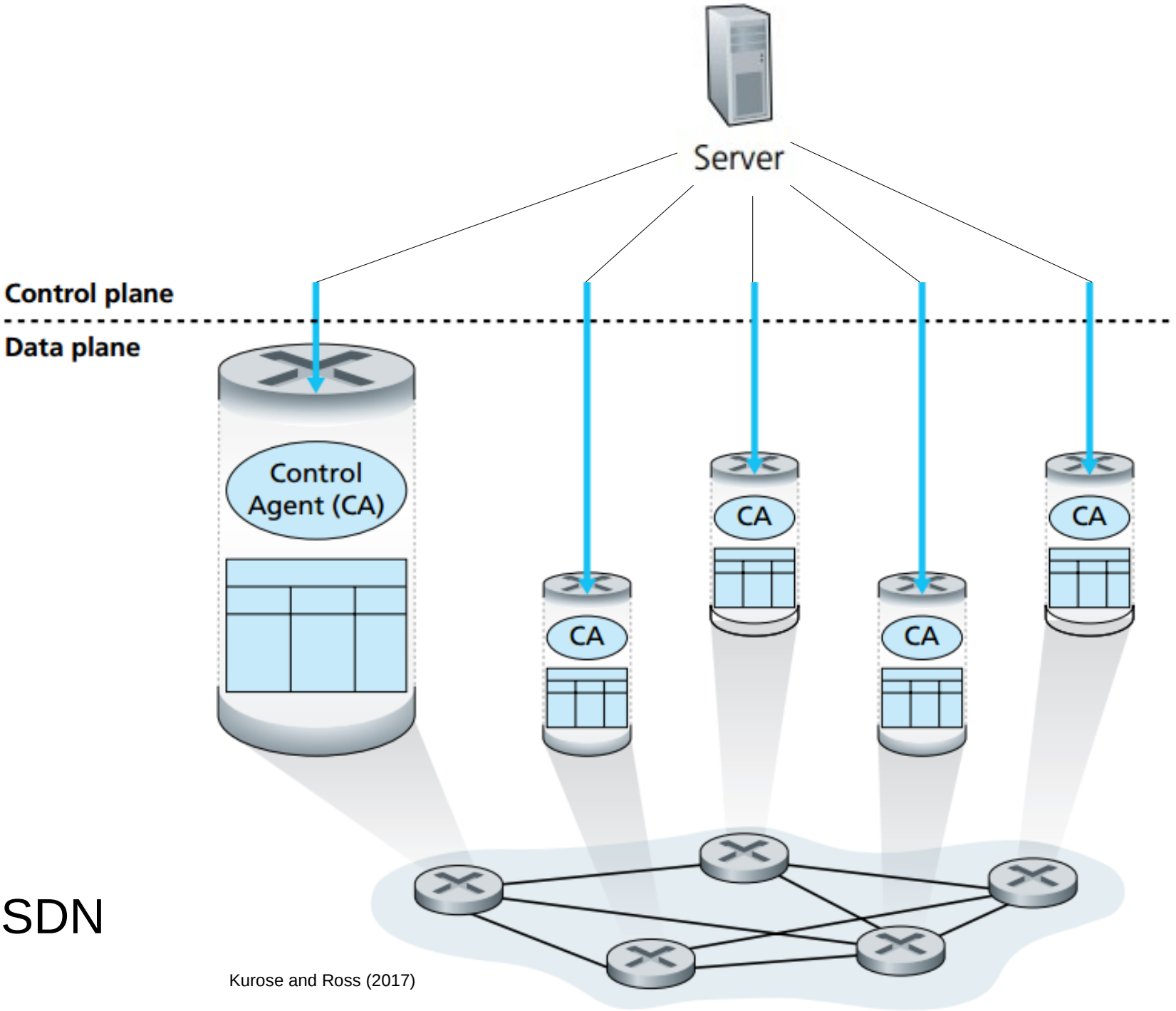
- Network management
  - Abstractions
  - Open interfaces
- Flexibility
  - Reconfiguration based on events: failures, faults, demands, attacks, ...
- Programmability
  - Innovations

# Software-Defined Networking (SDN)



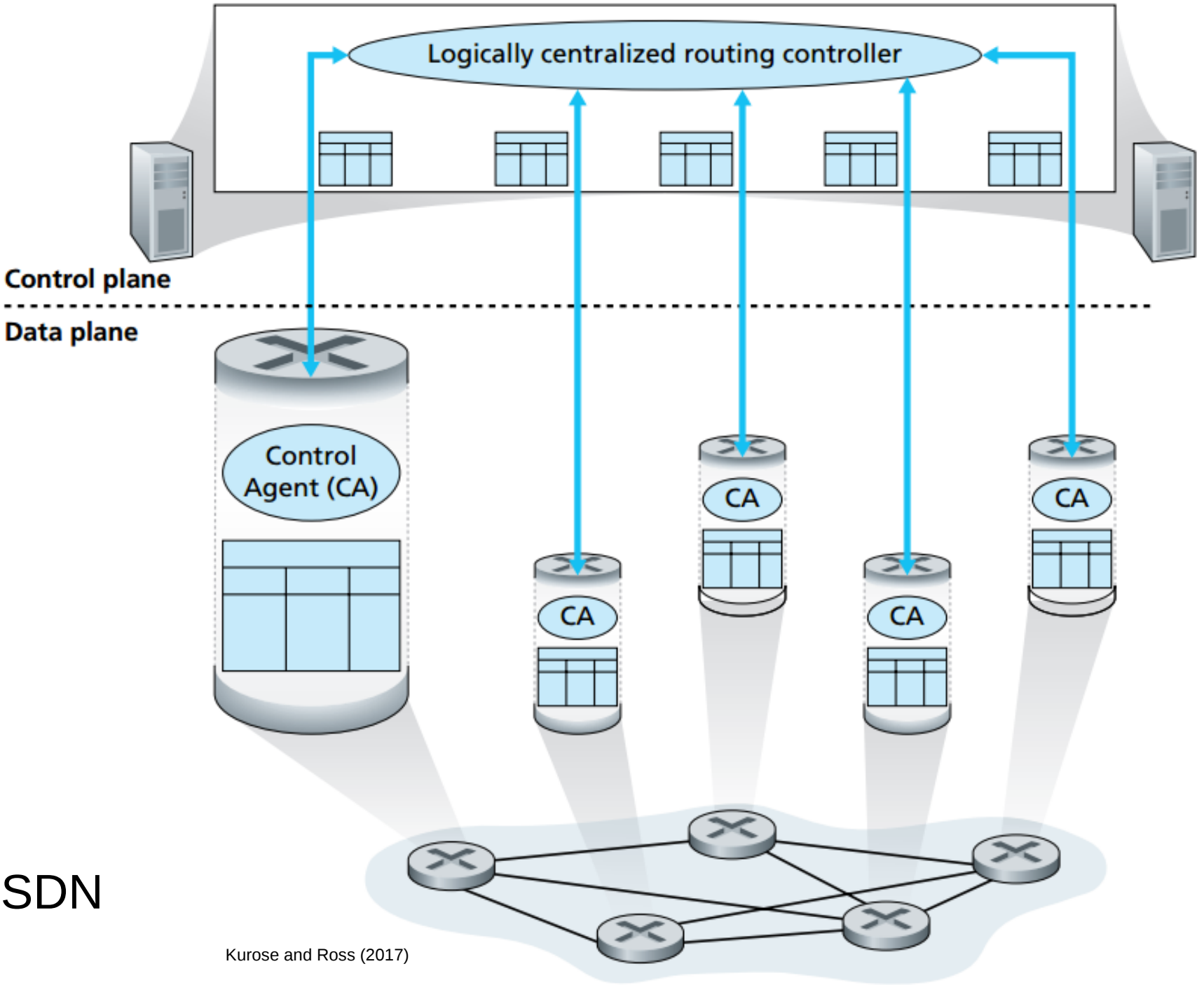

# Traditional Networking

Kurose and Ross (2017)




# SDN

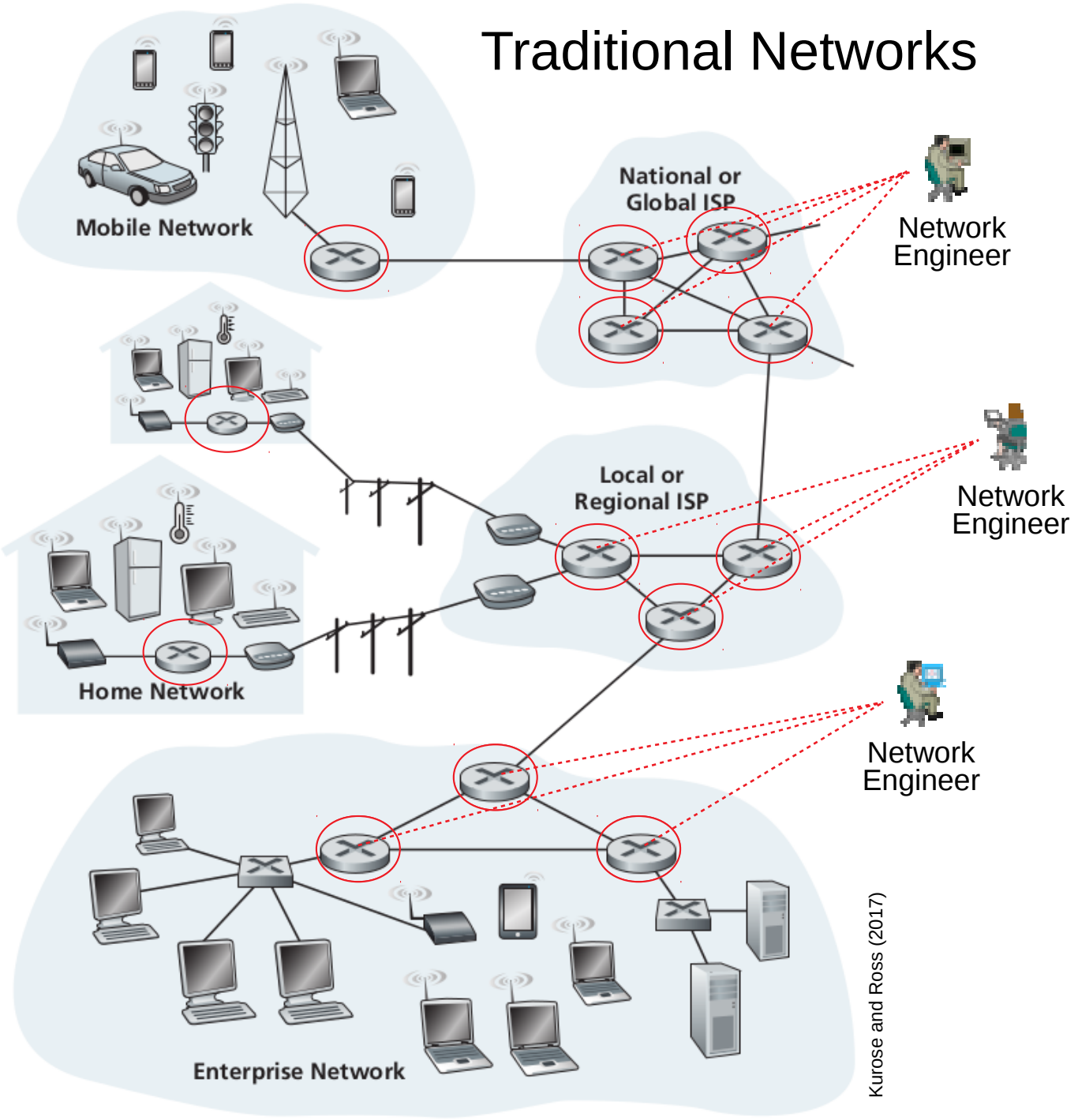
Kurose and Ross (2017)




# SDN

Kurose and Ross (2017)

# Traditional Networks



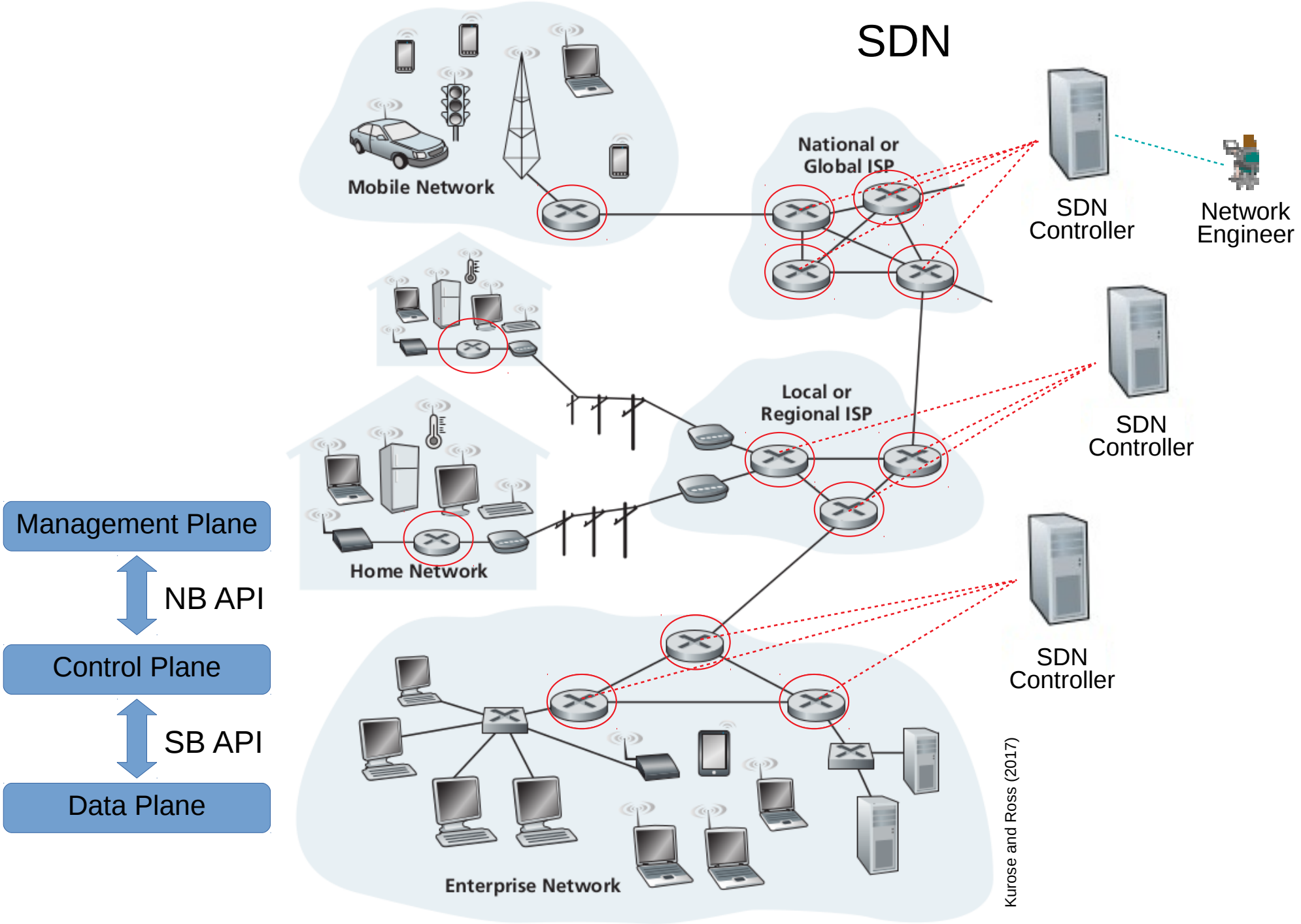

Management Plane

Remote Access

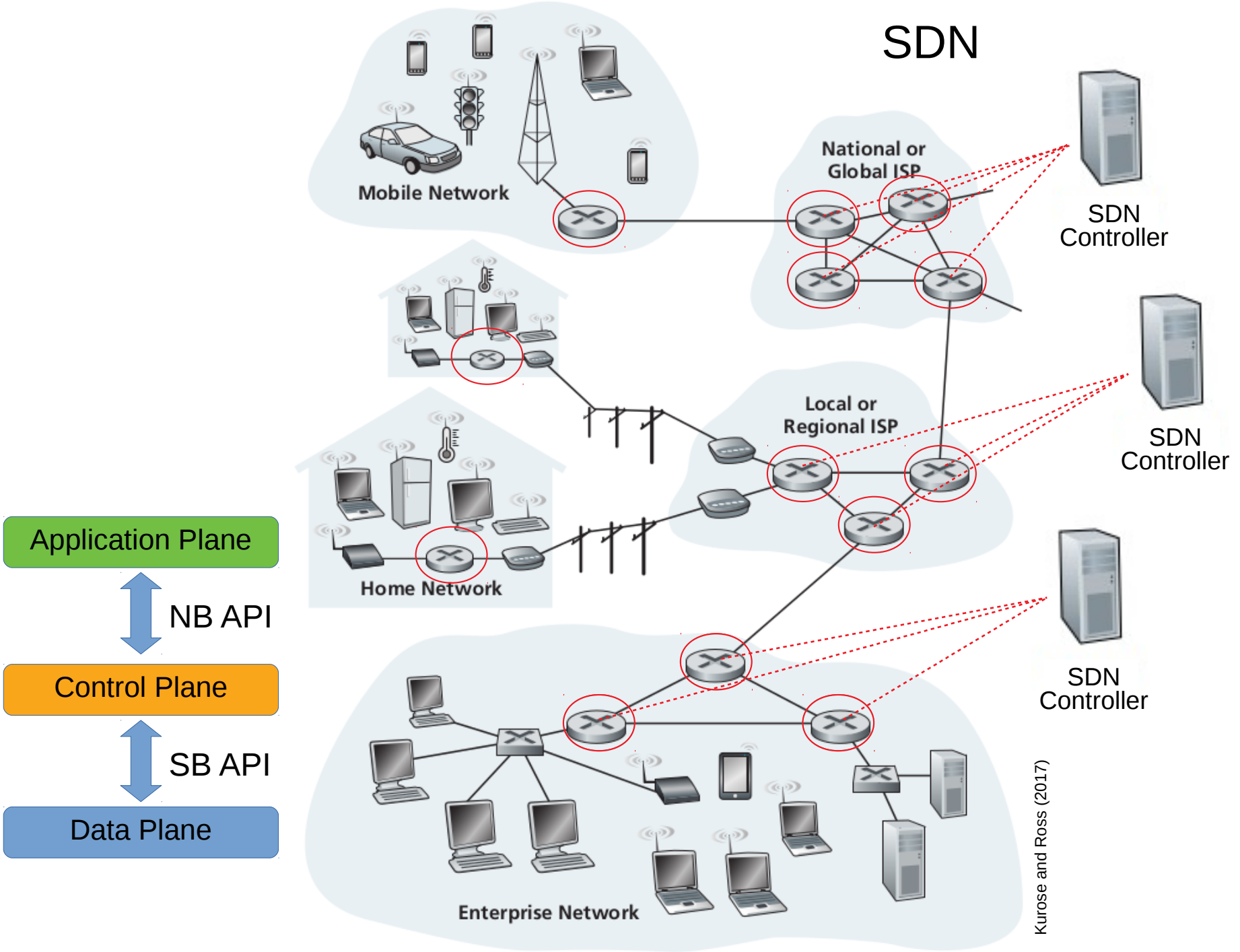
Control Plane

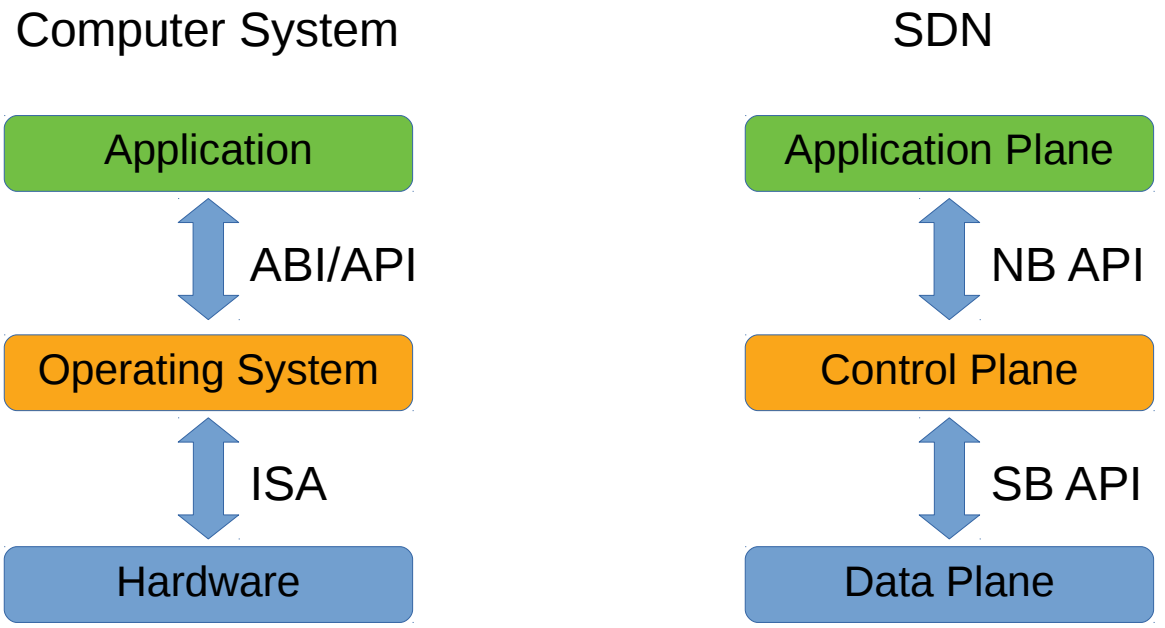
Data Plane

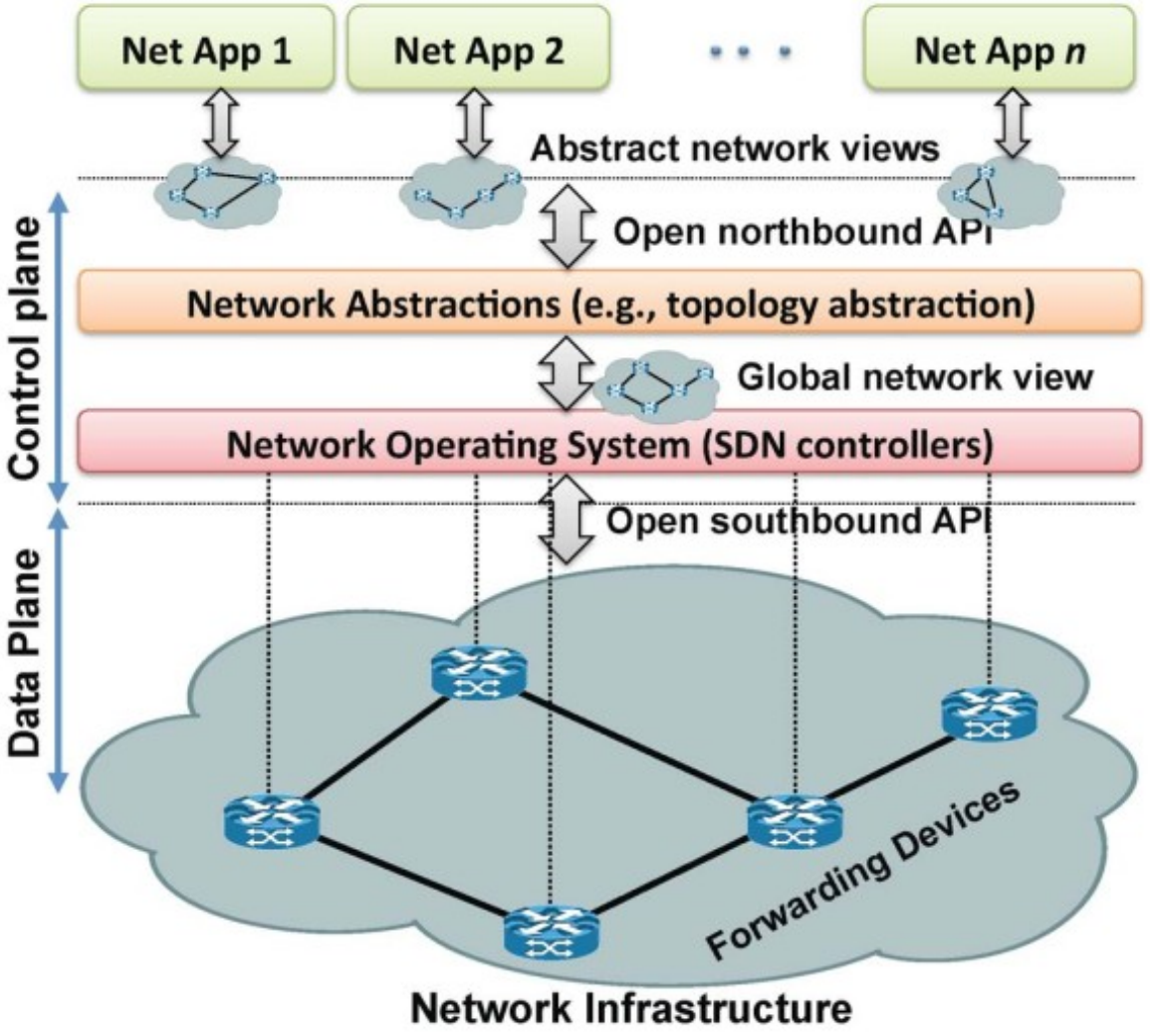
# SDN


Management Plane
NB API
Control Plane
SB API
Data Plane
Mobile Network
National or Global ISP
Local or Regional ISP
Home Network
Enterprise Network
SDN Controller
SDN Controller
SDN Controller
Network Engineer
Kurose and Ross (2017)

# SDN


Application Plane
NB API
Control Plane
SB API
Data Plane
Mobile Network
National or Global ISP
SDN Controller
Local or Regional ISP
SDN Controller
Home Network
SDN Controller
Enterprise Network
Kurose and Ross (2017)


Computer System
Application
ABI/API
Operating System
ISA
Hardware
SDN
Application Plane
NB API
Control Plane
SB API
Data Plane




Kreutz, et al. (2015)

# SDN is About Abstraction

- Separating control and data planes
  - Data plane: simple forwarding elements
  - Control plane: controller or NOS, basic/core services
  - Communications between those planes are through Southbound API, e.g. OpenFlow (packet handling), OVSDB, NETCONF, SNMP
- Application plane: apps that run on top of control plane
  - Can be anything that provides network services such as routing, mobility, firewall, IDS, etc. NFV
  - Force policies through control plane
  - Communicate with control plane via Northbound API, e.g. REST

# Network Targets

- Data center networks
- Enterprise/campus networks
- Wide area networks
  - Internet exchange points (IXPs)
- Mobile wireless networks
- Optical networks
- Internet of Things
- Home networks
- ...

# Experimenting SDN

- Simulation in network emulator
- Testbed with real SDN switches

# Data Plane

# Data Plane

- Hardware switches
  - Switches from Allied Telesis, Cisco, Dell, HP/Aruba, Pica8, and so on
  - Board: Zodiac FX for SDN experiment on your desk or for home networks
  - Cheap switches e.g. TP-Link WR1043ND flashed with OpenWrt based on Pantou/CPqD
- Software switches: Open vSwitch, LINC, ...

# Control Plane

# SDN Controller

- A middle layer
- Network Operating System (NOS)
- Basic/core/key services (apps actually)
  - Topology discovery
  - Inventory
  - Statistics
  - Host tracking

# POX

- POX is an SDN controller written in Python
  - Python 2.7
  - OpenFlow 1.0
  - Open vSwitch/Nicira extensions
- Popular for learning SDN
- Boot up with pox.py and load components that are specified in options
  - Components: supports, network applications
- POX Python API
- Site: github.com/noxrepo/pox

# Ryu

- A component-based SDN Framework: controller+
- Includes components and well-defined API to make easy for developers to create new network management and control applications
- OpenFlow, NetConf, OF-config, etc.
  - Supports OpenFlow 1.0, 1.1, 1.2, 1.3, 1.4, 1.5 and Nicira extensions
  - Can be integrated with OpenStack
- Developed by Nippon Telegraph and Telephone (NTT)
- Used as the base SDN controller/framework by several projects such RouteFlow and Faucet
- Site: osrg.github.io/ryu/

# SDN Controllers

| | NOX | POX | Ryu | Floodlight | ODL |
|---|---|---|---|---|---|
| **Language** | C++ | Python | Python | Java | Java |
| **Performance** | Fast | Slow | Slow | Fast | Fast |
| **Distributed** | No | No | Yes | Yes | Yes |
| **OpenFlow** | 1.0 (CpqD: 1.1, 1.2, 1.3) | 1.0 | 1.0, 1.1, 1.2, 1.3, 1.4 | 1.0 | 1.0, 1.3 |
| **Multi-tenant Clouds** | No | No | Yes | Yes | Yes |
| **Learning Curve** | Moderate | Easy | Moderate | Steep | Steep |

Feamster (2014)

# Application Plane

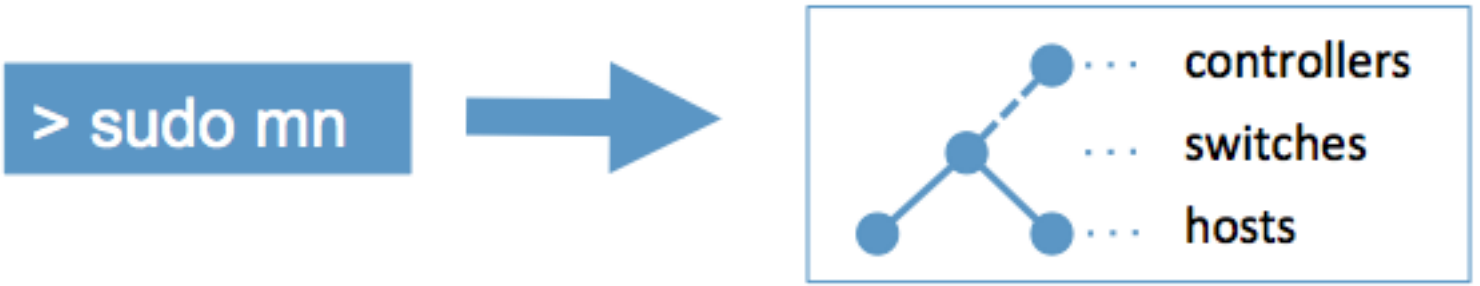
# Application Plane

- A layer on top of controller/control plane
- Communicate to control plane via Northbound API, unfortunately there is no standard. Some controllers provide programming language-specific API or REST
- Example of network apps: firewall, load balancing, monitoring, intrusion detection system, mobility manager, QoS, etc.

# Running SDN in Network Emulator

# Mininet

- A network emulator for experimenting SDN
- Provides realistic virtual network consisting of virtual hosts, switches, links, and controllers with real kernel, switch, and application code
- Batteries included
  - OpenFlow, Open vSwitch, POX, Wireshark dissector
- Built using Python utilizing several technologies to provide virtual environment: container-like node, software switch
- Command line and Python API
- Site: mininet.org



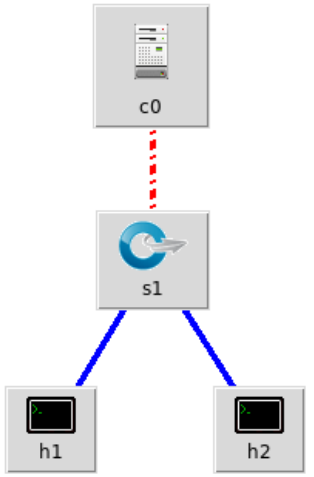

mininet.org

# Basic Steps

- Download Mininet VM
- Boot VM with VirtualBox or QEMU/KVM
- Execute “sudo mn”

# Interacting with Mininet

- Default topology: minimal, consists of one OpenFlow kernel switch connected to two hosts, plus OpenFlow reference controller
- Commands: help, nodes, net, dump
- Run commands
  - [node] ifconfig -a
  - [node] ps -a
  - pingall
  - [node] python -m SimpleHTTPServer 80 &
  - [node] kill %python
  - xterm [node]



# Mininet

- Regression tests
  - sudo mn --test pingall --topo single,3
  - sudo mn --test iperf
- Link variations
  - sudo mn --link tc,bw=10,delay=10ms
  - Bandwidth 10 Mbps, delay of each link is 10 ms
- Changing topology
  - CLI: sudo mn --topo linear,4
  - Custom topology

# Custom Topology (1)

- Adding “topos” dict with a key/value pair to generate topology. Pass in “topo=mytopo” parameter

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )
```

# Custom Topology (2)

```
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

# Custom Topology (3)

$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall

# Misc.

- Python interpreter: py h1.IP()
- Link up/down: link s1 h1 down/up

# Python API Examples

- See directory mininet/examples

# Using Remote SDN Controller

- POX will be used and it runs on the same VM, but outside Mininet
- Running POX

  $ cd ~/pox

  $ ./pox.py forwarding.l2_learning

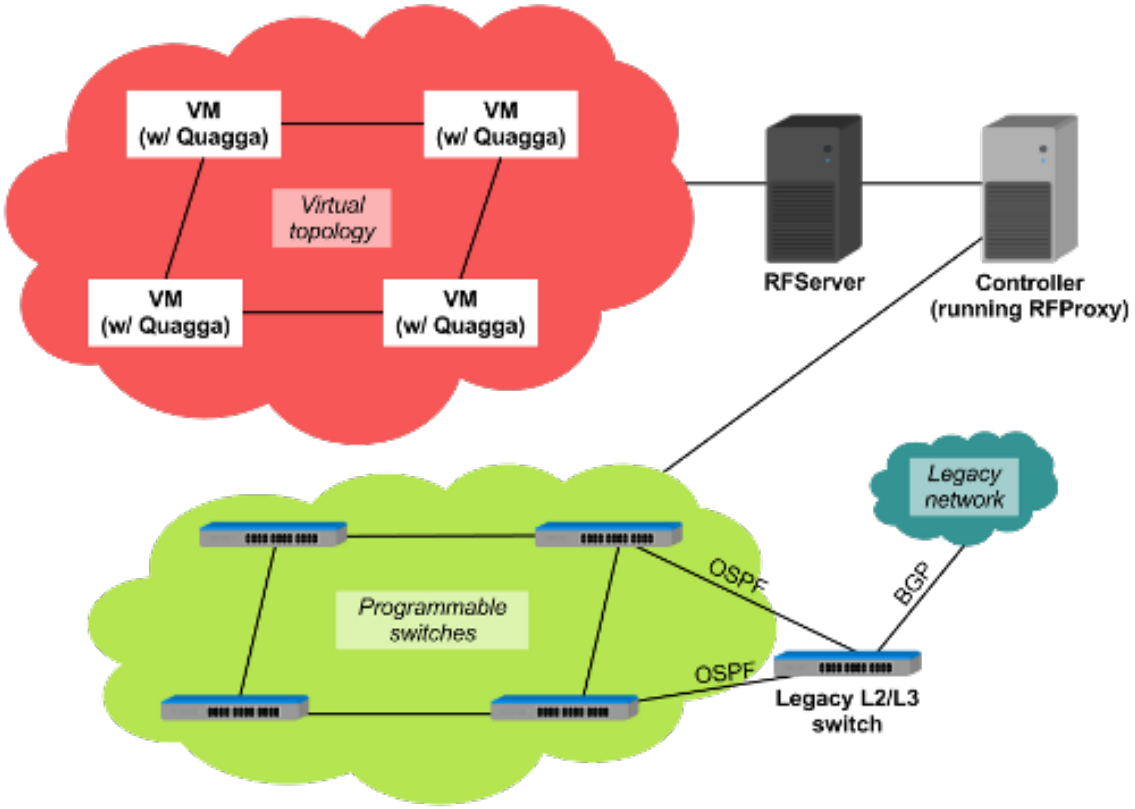
  $ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
- See in pox dir: forwarding.l2_pairs, info.packet_dump, samples.pretty_log, and log.level --DEBUG

# SDN Apps

- forwarding.l2_learning
  - Making OpenFlow switches like Ethernet learning switches
  - Learning MAC addresses and matches all fields in the packet header
- openflow.discovery
  - To discover switches and interconnected links
- host_ tracker
  - Tracking hosts in the network

# Projects

# RouteFlow (1)

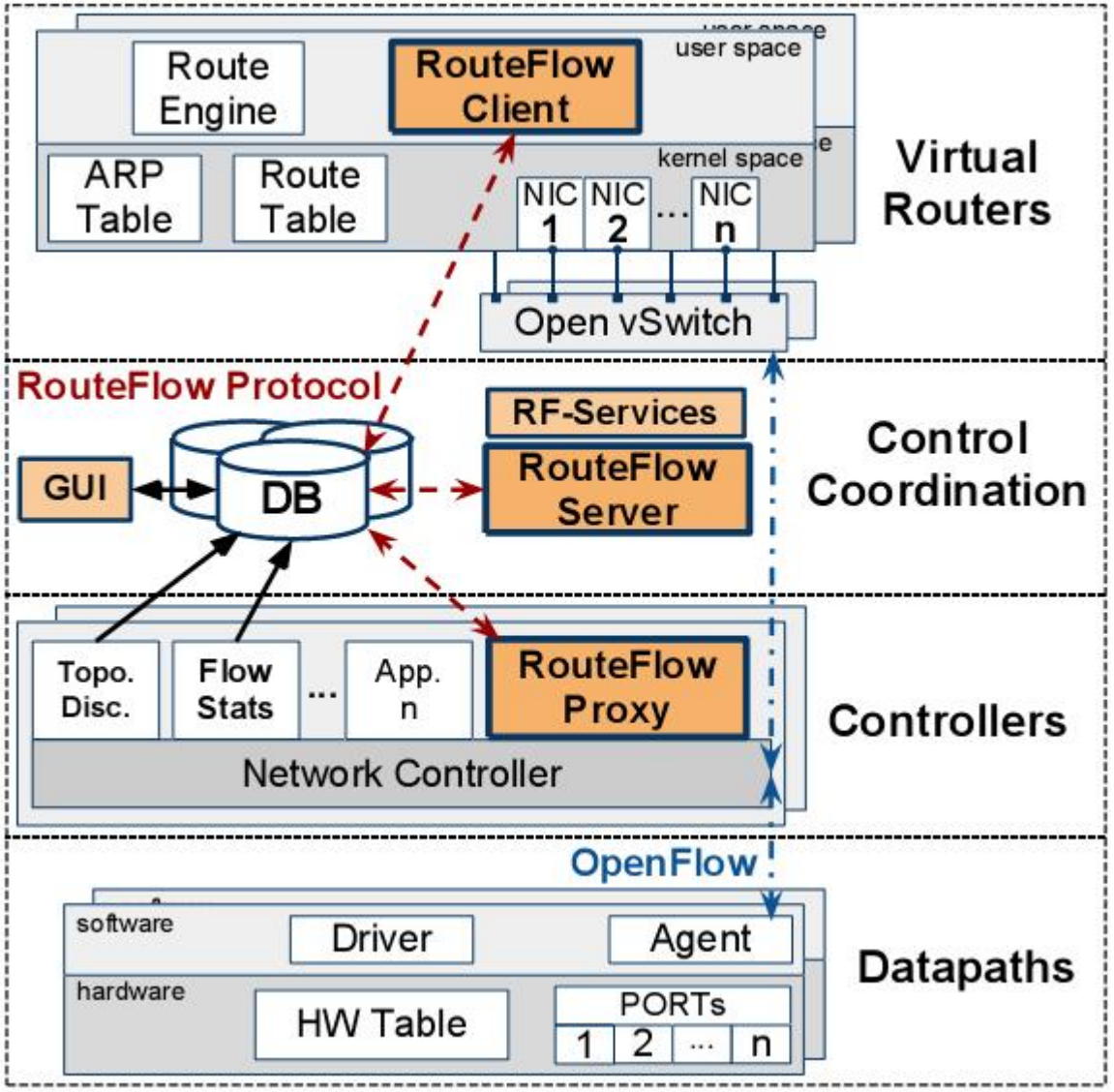



https://routeflow.github.io/RouteFlow/

# RouteFlow (2)

- Providing virtualized IP routing services over OpenFlow infrastructure
  - IP-Routing-as-a-Service model of networking
  - Hybrid networking (traditional-SDN)
  - Alternative path to migrate legacy IP deployments to SDN
  - Framework for different flavours of network virtualization such as logical routers, router aggregation
  - Simplify intra- and inter-domain routing interoperable with legacy equipments (non-SDN/OpenFlow)
- SDN controller: POX or Ryu
- Site: routeflow.github.io/RouteFlow/

# RouteFlow (3)

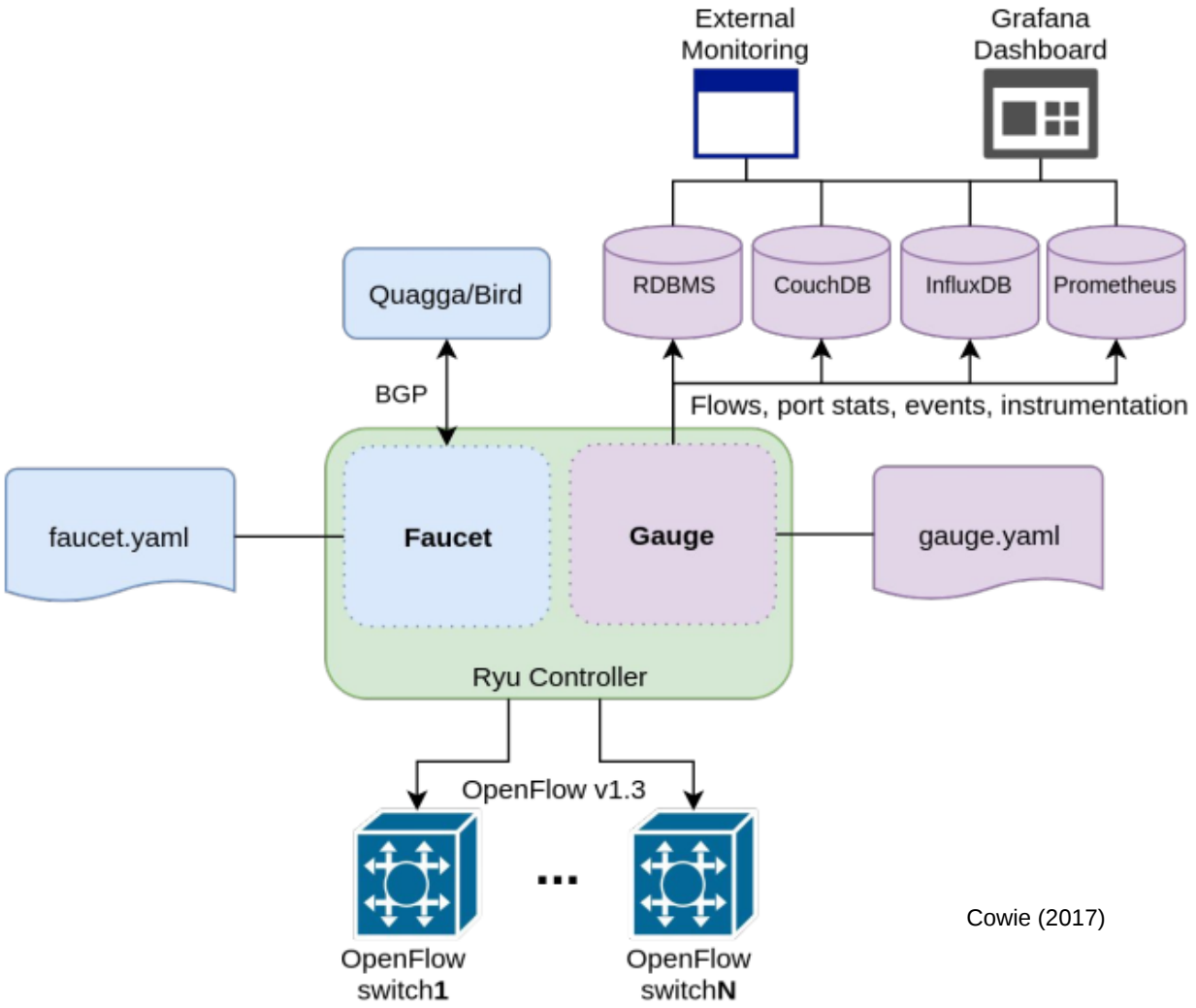



https://sites.google.com/site/routeflow/

# Faucet

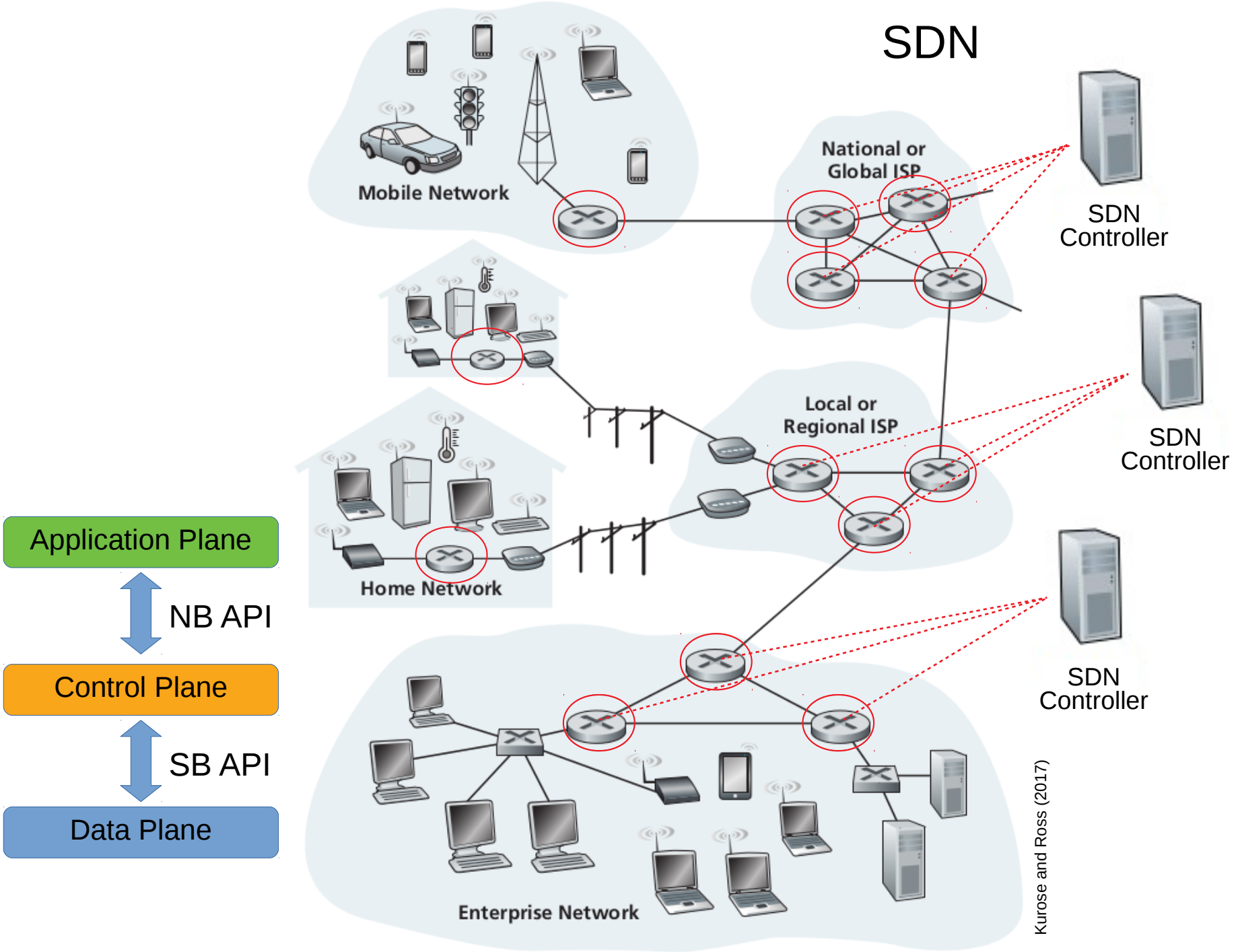
- Lightweight SDN controller based on Ryu
- Targeted for enterprise networks
- OpenFlow 1.3, multi-vendor (single packet processing), supports Layer 2 and 3
- Policy driven approach for extensibility
- Monitoring and instrumentation
- Well-tested
- Production ready
- Installs in < 30 seconds
- Site: faucet.nz

# Faucet Architecture

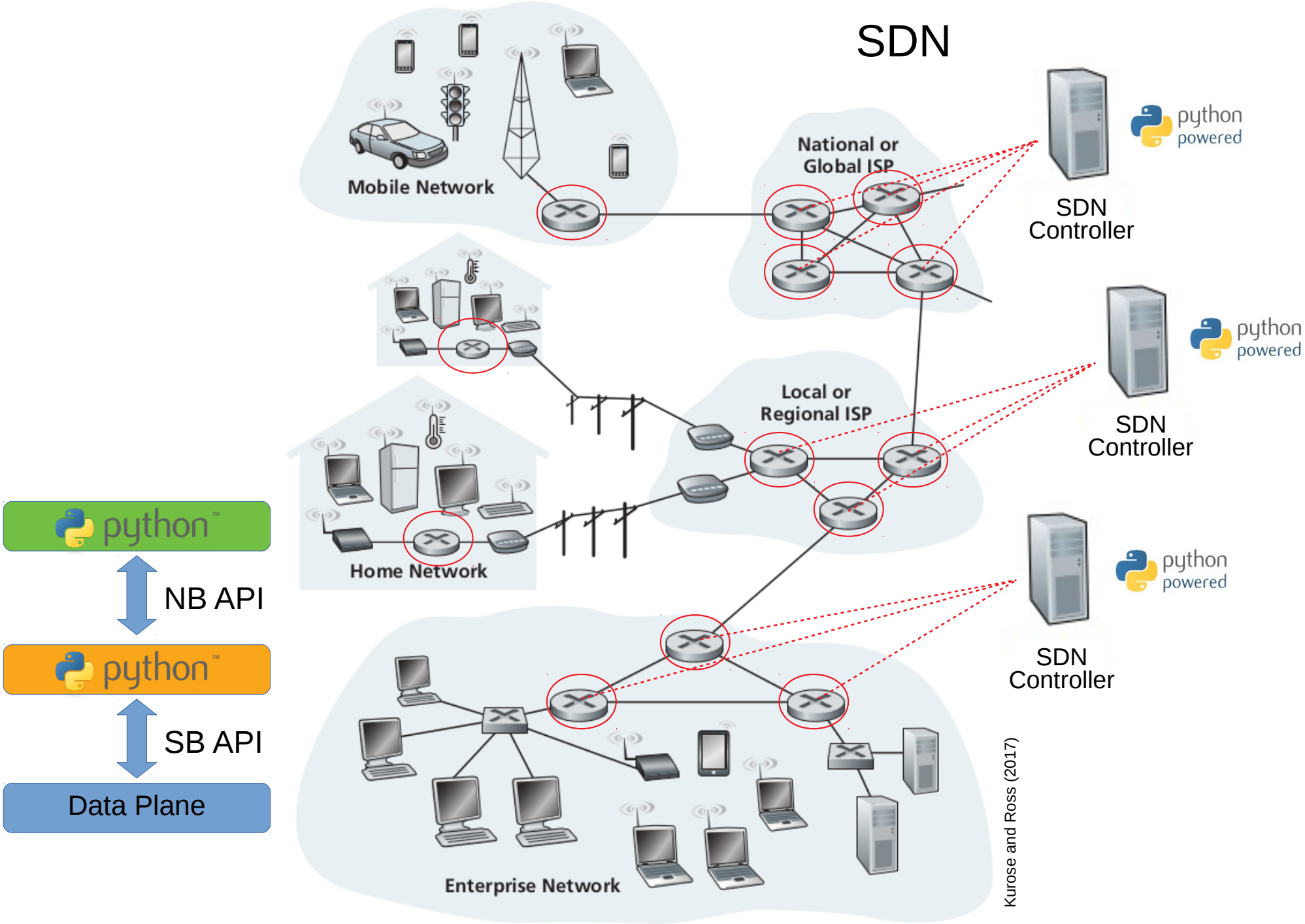



Cowie (2017)

# SDN


Application Plane
NB API
Control Plane
SB API
Data Plane
Mobile Network
National or Global ISP
SDN Controller
Local or Regional ISP
SDN Controller
Home Network
SDN Controller
Enterprise Network
Kurose and Ross (2017)

# SDN


Mobile Network
National or Global ISP
Local or Regional ISP
Home Network
Enterprise Network
SDN Controller
SDN Controller
SDN Controller
python powered
python powered
python powered
python
NB API
python
SB API
Data Plane
Kurose and Ross (2017)

# Summary

# Python-Defined Networking

- Python can be used to define networking
- The lightweight feature and easy-to-moderate learning curve makes Python-based SDN softwares interesting to be learned and developed
- Controller is really crucial in SDN. With Python-based one, it will give a platform for developing Python-based network applications
- Projects such as RouteFlow and Faucet are really making Python-defined networks

# Reading List


[1] D. Kreutz, F.M.V. Ramos, P.E. Veríssimo, C.E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: a comprehensive survey,” Proc. IEEE, vol. 103, no. 1, pp. 14-76, January 2015.

[2] R. Jain, “Introduction to software-defined networking (SDN),” CSE570, 2015.

[3] B. Cowie, “Software-defined enterprise networks,” OpenStack Summit Sydney, 2017.

# How SDN Works

- Packet flow:
  - Packet arrive at forwarding device, parse packet header. If it knows how to take action, or query to the SDN controller
  - Network app determines what action to do, and send it to controller, then the controller send the action to the forwarding devices
  - Device per device does the sending of packet-in
  - There is a cache of the flow entries, until expired
  - Fast-path if there is a cache of the rule, action in each forwarding device
- D. Mahler, “Introduction to SDN (Software-defined Networking),” November 2014, [Online]. Available: https://youtube.com/watch?v=DiChnu_PAzA