department of computer science
faculty of sciences

Internet Programming

Security

Introduction

Security Issues in Internet Applications

- A distributed application can run inside a LAN
  - Only a few users have access to the application
  - Network infrastructures are handled by one organization
  - Security is not a huge threat (in general)

- When you deploy an application over the Internet
  - You cannot control who your users are
  - You cannot control the network infrastructures
  - There may be attacks to your application
  - You must address the security issues!

What is Security? [1/2]

- **Confidentiality**
  - Prevent unauthorized disclosure of the information
  - An attacker cannot “spy” on you
- **Integrity**
  - Prevent unauthorized modification of the information
  - An attacker cannot modify your information
- **Authentication**
  - Prove your identity to the system
  - An attacker cannot pretend to be an authorized user
- **Authorization**
  - Define what each user is allowed to do

What is Security? [2/2]

- **Non Repudiation**
  - To prevent false denial of a contract
  - A user cannot pretend he did not sign some statement
- **Auditing**
  - To securely record evidence of performed actions
- **Availability**
  - Guarantee access to the information
  - An attacker cannot prevent you from accessing your information
- **Fault Tolerance**
  - To provide some degree of (correct) service despite failures or attacks

Basic Security Mechanisms

Encryption / Decryption

- Encryption provides **confidentiality**
- **Symmetric** key encryption (e.g., DES)
  - The **same key** is used for **encryption** and **decryption**
  - But how do you distribute the key to your partners?
    - You need to communicate over a secure channel!
    - E.g., give it by hand, by post, by other means
- **Asymmetric** key encryption (e.g., RSA)
  - Keys are created in pairs
    - **Public key**: You can announce it to everyone → Used to ENCRYPT
    - **Private key**: You should keep it strictly for yourself → Used to DECRYPT
  - You encrypt your messages with the recipient's public key
- Symmetric encryption/decryption is usually much faster than asymmetric

Digital Signatures

- Digital signatures provide **integrity** and **authentication**
- You attach a signature to each message
  - **Only you** can attach a correct signature to a message
  - An attacker can modify the message but cannot forge a correct signature to pretend the message comes from you
- Digital signatures are usually realized by asymmetric keys
  - The same pairs of keys as for asymmetric encryption!
  - You **sign** the message → **Encrypt it with your private key**
  - The recipient can **check** the signature without knowing the secret key
  - It is **"impossible"** (read: **computationally infeasible**) to create a correct signature without knowing the secret key
- If you want both integrity, authentication and confidentiality, **use both encryption and signatures**!

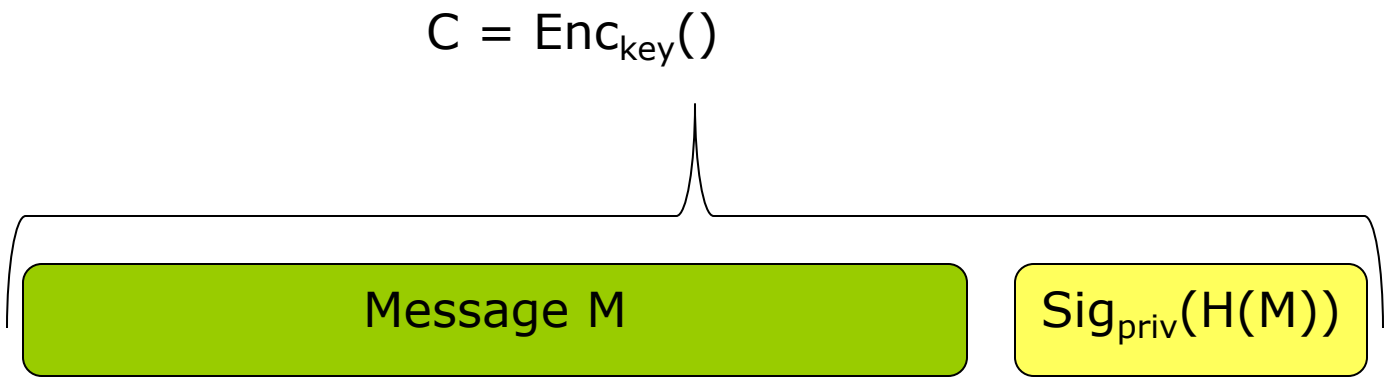
Hash Functions

- **Cryptographic hash functions** provide **integrity** as well

- Cryptographic hash functions are the same as signatures, but without a key
  - You compute a signature (or hash) of your message
  - You transmit it in a secure way
  - It is very hard to create another message whose hash will be the same
  - Anyone can compute the hash of the received message and check whether it matches the expected value

```
% shasum  -a 256 slides.tex
b08abd37753c0a39a3764a4d1bc8b2ce192751de961160140e5c5a66e2d7afb8
% echo "." >> slides.tex
% shasum  -a 256 slides.tex
219210b2152fc020ac33d4b33704112dfa5f22c7625268698fd39c7428913d39
```

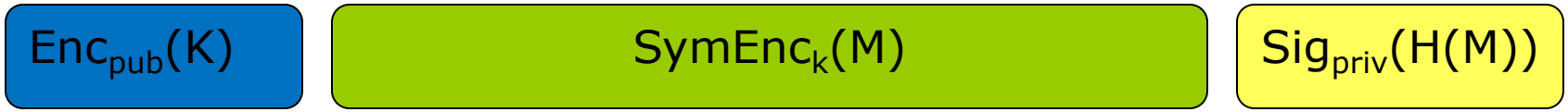
Combination: **privacy**, **integrity** and **authentication**



1. Public key must be distributed
2. Symmetric key must be exchanged

- M, $Sig_{priv}(H(M)) = Dec_{key}(C)$
- H(M)
- $Ver_{pub}(Sig_{priv}(H(M))) == H(M)$

Alternative: **privacy**, **integrity** and **authentication**



For sending

- Compute the hash of message, *i.e., H(*M), and sign it with your private key (yellow)
- Encrypt message *M* with symmetric key *k (green)*
- Encrypt the symmetric key *k* with the public key (blue)

For receiving

- Decrypt key *k*, with private key
- Decrypt ciphertext of *M* with symmetric key *k*
- Compute hash of M and verify that signature using public key are equal.

Secure Protocols

- Basic security tools are not enough
  - How do you use them?
  - How do you react when something happens?
- Secure protocols define how basic tools are used
  - They provide higher levels of security by merging the strengths of several basic security tools
  - They provide simple security mechanisms
  - The provide standards
- Examples:
  - **PGP**: **P**retty **G**ood **P**rivacy
    - Encryption / decryption / signatures
  - **SSL**: **S**ecure **S**ocket **L**ayer
  - **SSH**: **S**ecure **SH**ell
  - **HTTPS**: Secure HTTP = HTTP over SSL rather than normal sockets

GPG

GPG

- GPG: **G**NU **P**rivacy **G**uard

- GPG is primarily meant for securing email
  - But you can also use it to encrypt / decrypt / sign any message
  - It is a free alternative to PGP
  - It is available at **http://www.gnupg.org/**

- GPG allows you to
  - Create public/private key pairs
  - Encrypt/decrypt messages
  - Sign/check messages

GPG Usage

- To create your own public/private key pair:

```
gpg --gen-key
```

- You will be asked several questions: which kind of key you want, which size, which expiration date, a username, your email address, etc.
  - Unless you have a good reason to do so, keep the default values

- Then it will ask you a passphrase
  - Type any passphrase
  - Your private key will be encrypted with this passphrase before being stored on disk
  - GPG will ask you for your passphrase each time it needs to access your private key

Disseminating your Public Key

- You can give your public key to anyone
  - In particular, to anyone who may want to communicate with you using GPG
  - **Never give your private key to anyone!**

- You can get your public key with:

```
gpg --export -armor <Your_Name>
```

- You can import somebody else's public key to your "keyring" with:

```
gpg --import <filename>
```

- You can list the keys contained in your keyring:

```
gpg --list-keys
```

Encryption / Decryption with GPG

- To **encrypt** a file, you must specify the recipient of your message
  - The message will be encrypted with the recipient's public key
  - Of course, the recipient's public key must be in your keyring

```
gpg --encrypt -r <recipient> < <file_to_encrypt>
```

- To **decrypt** a file
  - GPG will use your private key to decrypt a file sent to you
  - So you will be asked to type your passphrase

```
gpg --decrypt < <file_to_decrypt>
```

Signatures with GPG

- To sign a message:

```
gpg --output <output_file>  --clearsign <document>
```

  - You will be asked for your passphrase

- To check a signature

```
gpg --verify <document>
```

- You can encrypt and sign a document

```
gpg --armor --sign --encrypt -r spyros@cs.vu.nl < document
```

- To decrypt and check:

```
gpg --decrypt  <encrypted_and_signed_document>
```

Public Key Authentication Problem

- **PROBLEM!**
  - How do you make sure a public key actually belongs to your partner?
  - An attacker may create a key pair and tell you that the public key belongs to your partner
  - He can then successfully intercept your messages to your partner

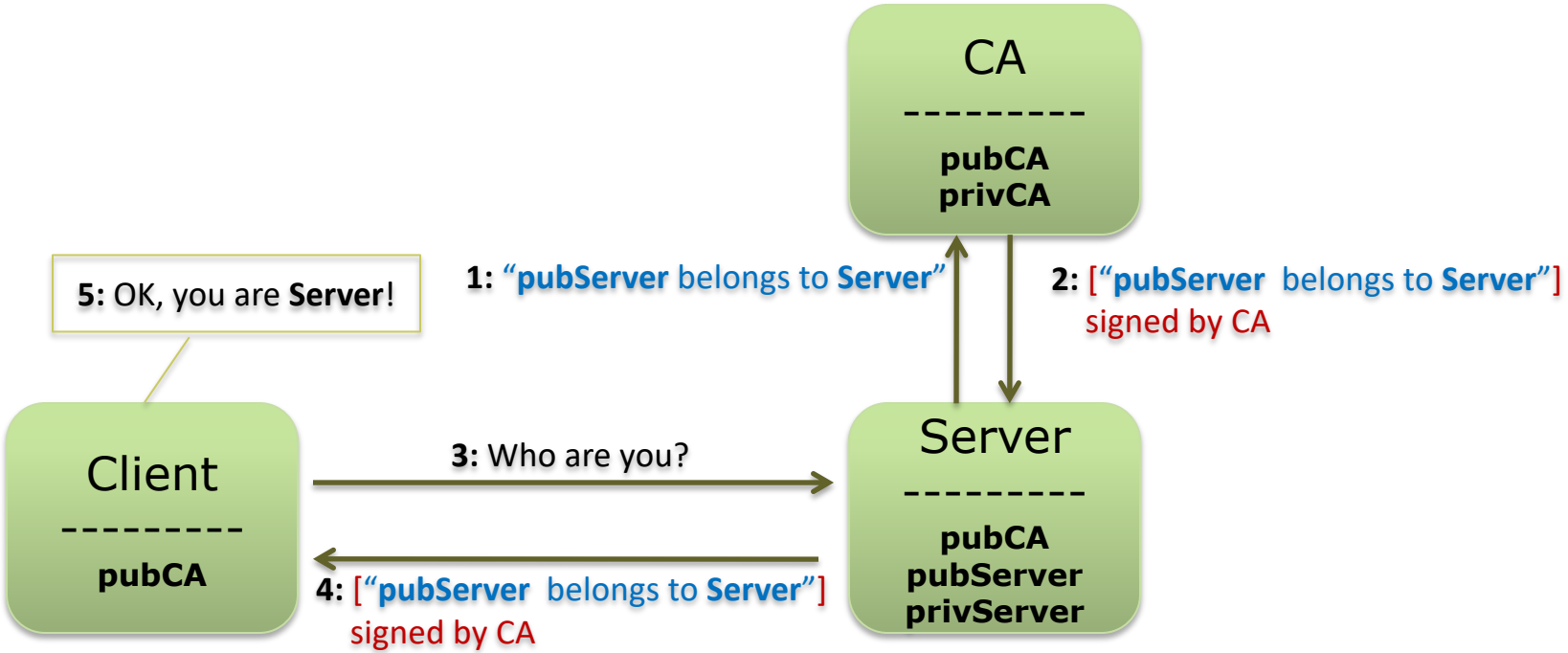
- PKI: **P**ublic **K**ey **I**nfrastructure
  - Provides **authentication**

Public Key Infrastructure

- A **Certification Authority (CA)** authenticates public keys
  - Everybody trusts the CA
  - Everybody knows the public key of the CA
  - The CA issues messages saying

    "I certify that key X belongs to person Y"
  - The message is signed by the CA, so everybody can check that it has been issued indeed by the CA

Public Key Infrastructure



- Steps (1) and (2) are done only once!
- Clients check the validity of messages, without bothering the CA

CA delegation

- Often, one public authority is not enough
- You want **delegation**: I will be the new CA for cs.vu.nl
  - The “root CA” certifies the “local CA”
  - The “local CA” can issue certificates
- A key is authenticated if there is a **chain of certifications** from the root CA to the key

SSL

SSL

- SSL: **S**ecure **S**ocket **L**ayer

- SSL is a communication protocol
  - It provides a higher layer of abstraction than normal sockets
    - Server authentication
    - Client authentication
    - Encryption
  - Secure sockets
  - SSL uses normal sockets as a base, and adds lots of crypto to it

- There are many implementations of SSL
  - There are multiple open-source implementations: **OpenSSL, LibreSSL, GNUtls**
  - It is quite difficult to use!

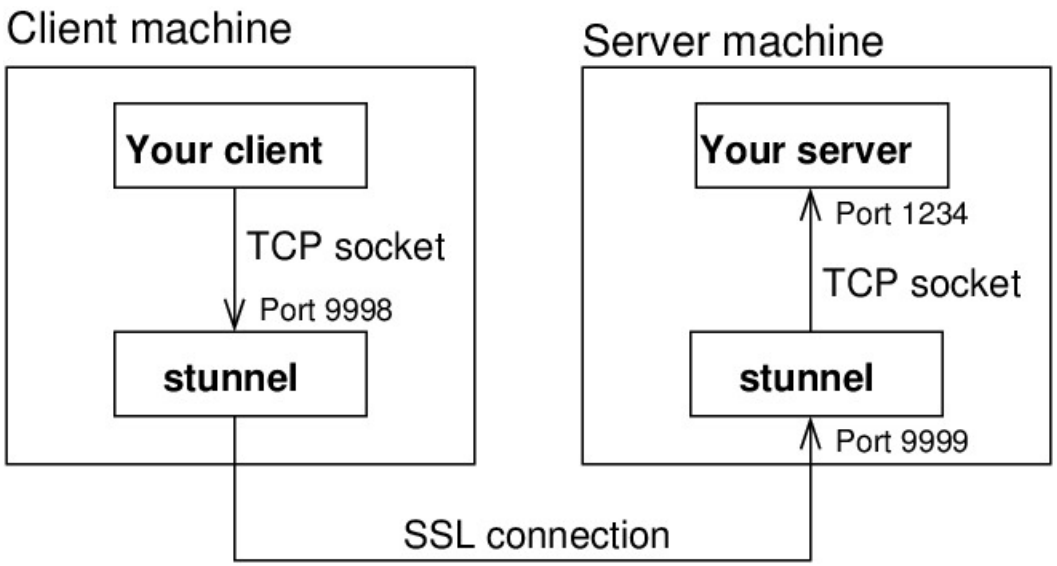
Bird-Eye View of OpenSSL

- You start by initializing SSL
  - Get yourself a key pair
  - Get a CA's certification for your key pair
  - Initialize the library
  - Create an "SSL context"
  - Load your key in the context, define the list of CAs that you trust
- Create a normal TCP connection
- Create an SSL connection (which uses the TCP connection)
  - Attach the SSL connection to the TCP connection
  - A number of checks are realized (e.g., the server key must be certified)
  - Use the SSL connection to transfer data
- Close the SSL connection and the TCP socket

stunnel: An SSL Tunnel

- What if you want to use SSL with an existing application?
  - **stunnel** allows you to use SSL without messing with the original code
  - Available from **http://www.stunnel.org/**

- stunnel creates a daemon which **converts regular TCP connections to SSL** and vice-versa



SSH tunnel

- You can use **ssh** in a similar way:

```
ssh -f -N -L localhost:9090:localhost:3500 acropolis.cs.vu.nl
```

  - **-f:** work in the background
  - **-N:** do Not execute a task, just establish a tunnel
  - **-L:** the tunnel origin and destination address

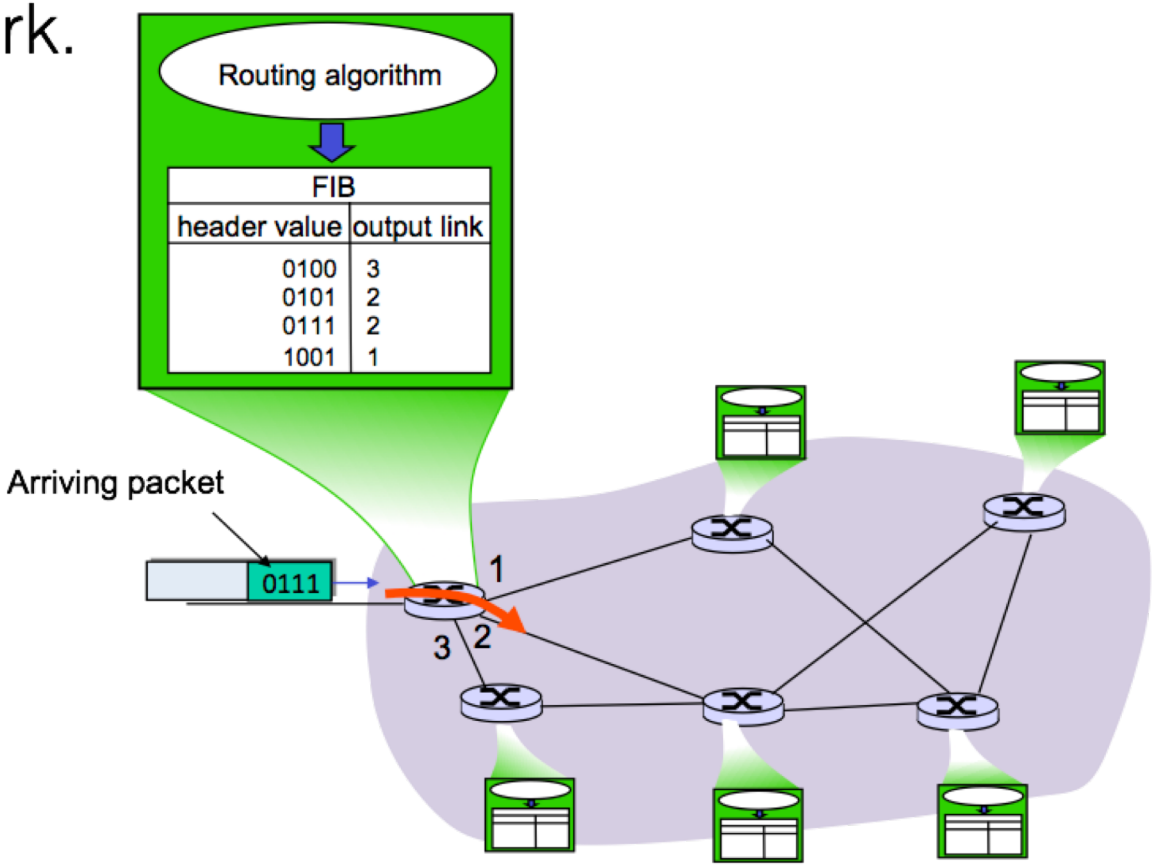
- Then, by accessing **localhost:9090**, ssh automatically **tunnels** the TCP connection to **localhost:3500** on **acropolis.cs.vu.nl**

SDN
Software Defined Networking*

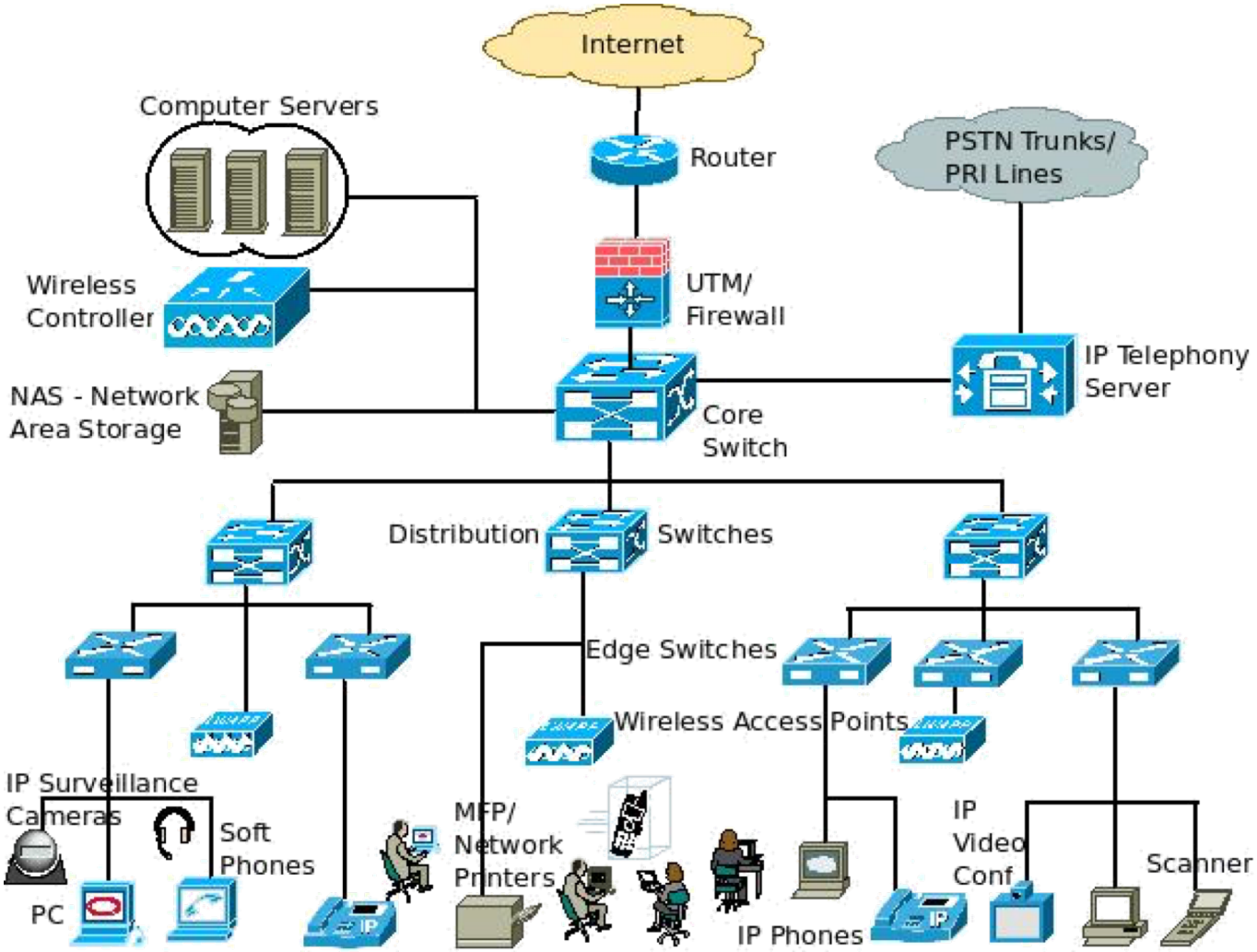
*Slides are based on presentation of duke university

Packet Forwarding

Packet forwarding Is the process of getting packets from source to destination in a network.



Limitations of Current Networks



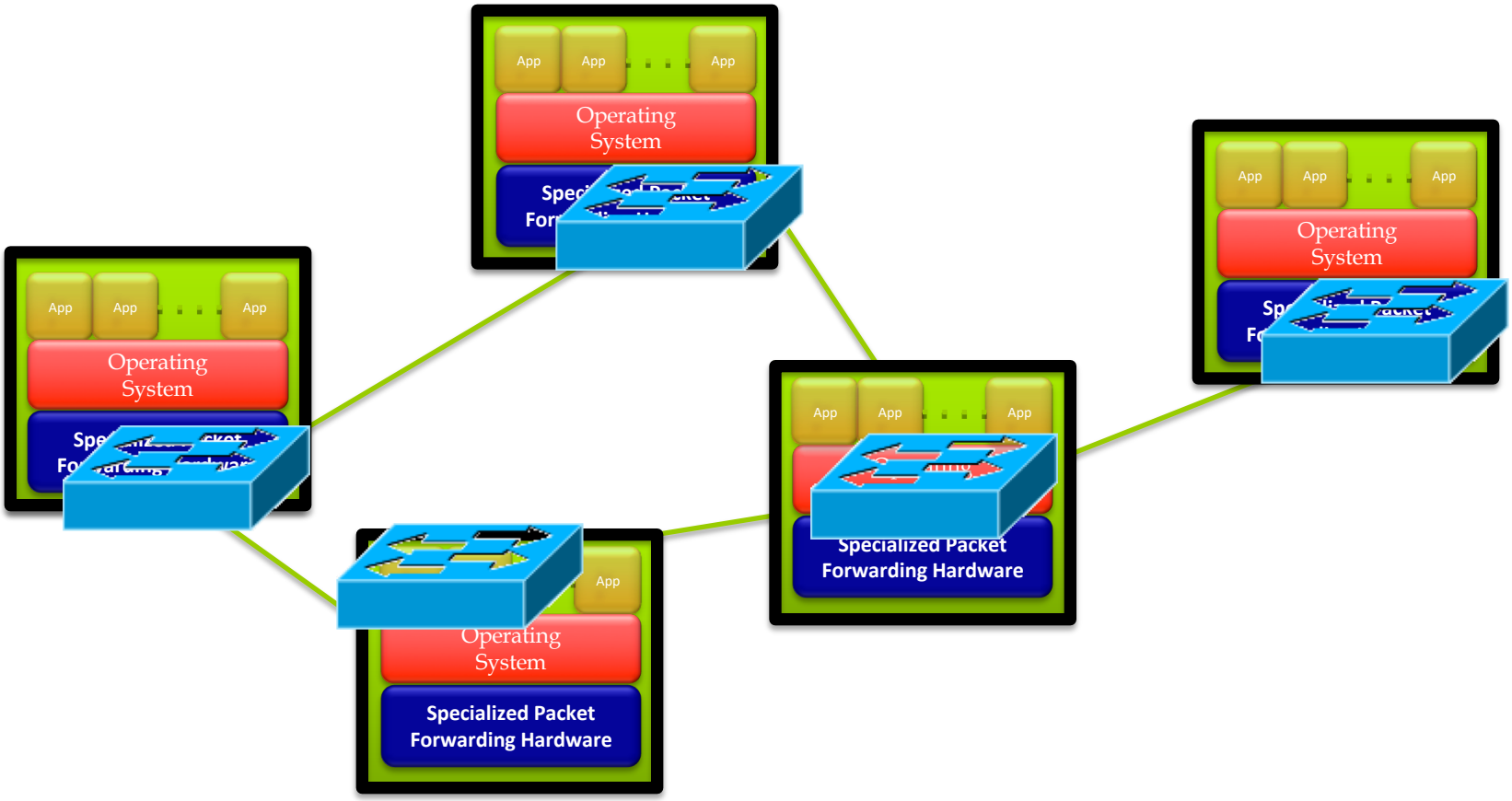
- **Enterprise networks are difficult to manage**
- **“New control requirements have arisen”:**
  - Greater scale
  - Migration of VMS
- **How to easily configure huge networks?**

Limitations of Current Networks

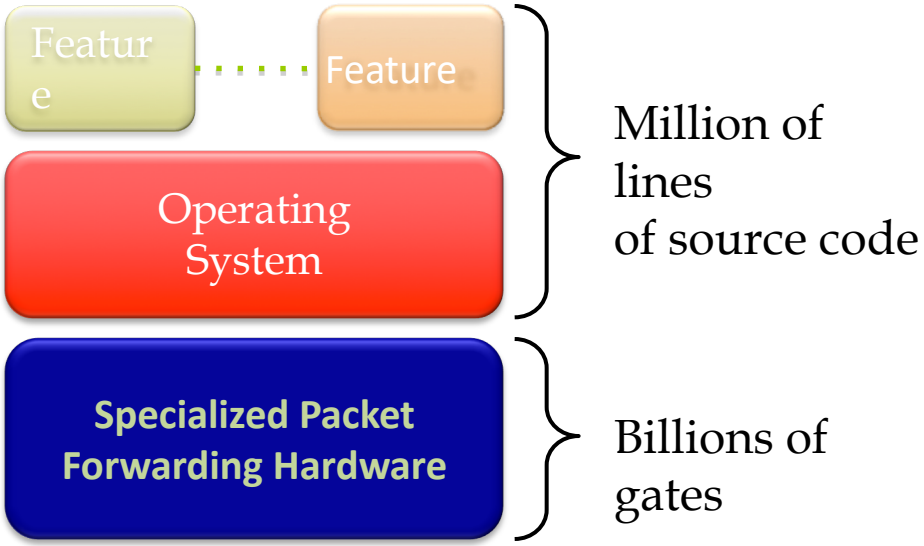
- **Enterprise networks are difficult to manage**
- **"New control requirements have arisen":**
  - Greater scale
  - Migration of VMs
- **How to easily configure huge networks?**

Limitations of Current Networks

- **Old ways to configure a network**



Limitations of Current Networks



Many complex functions baked into infrastructure

OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, …

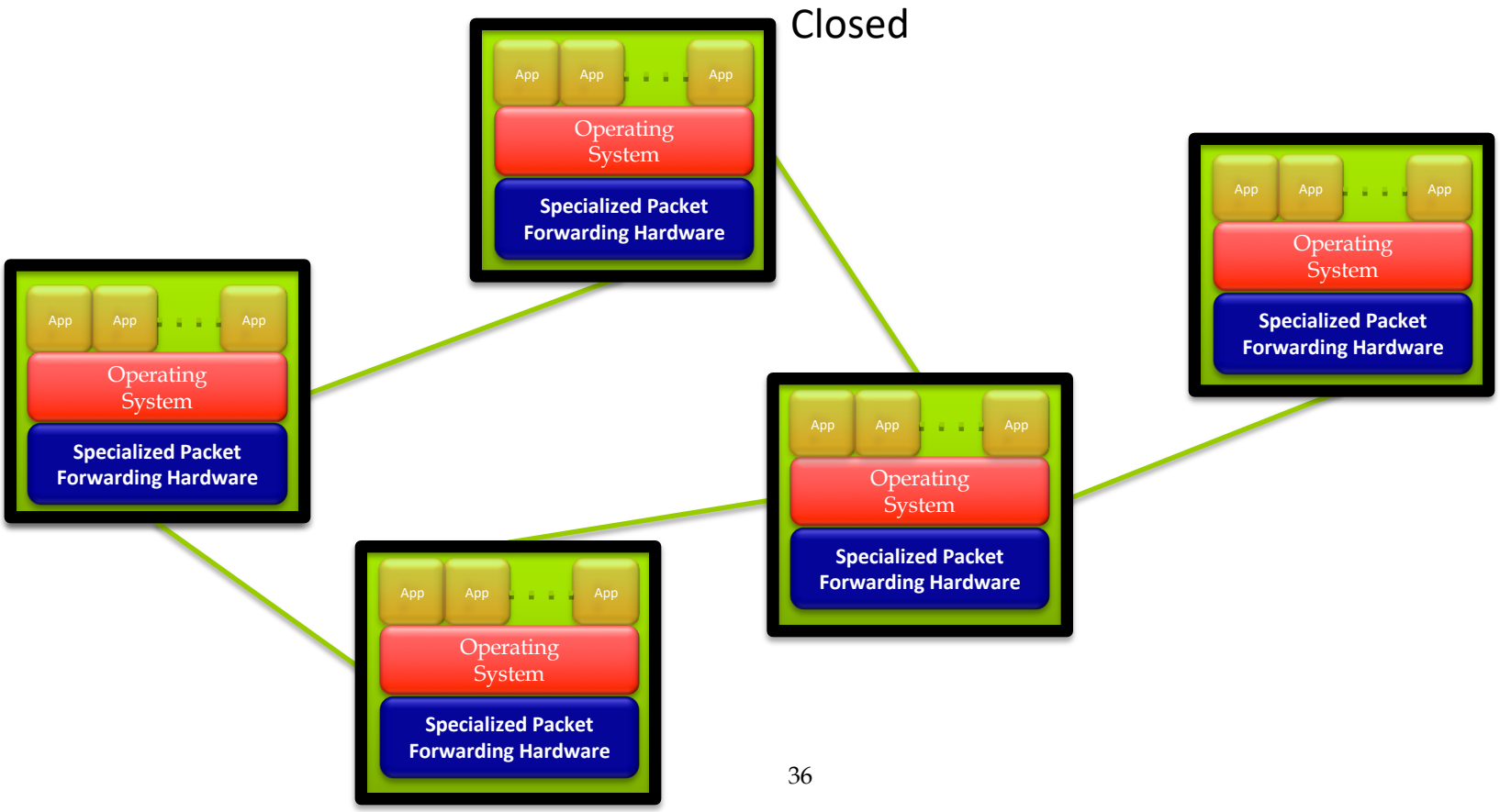
Cannot dynamically change according to network conditions

Limitations of Current Networks

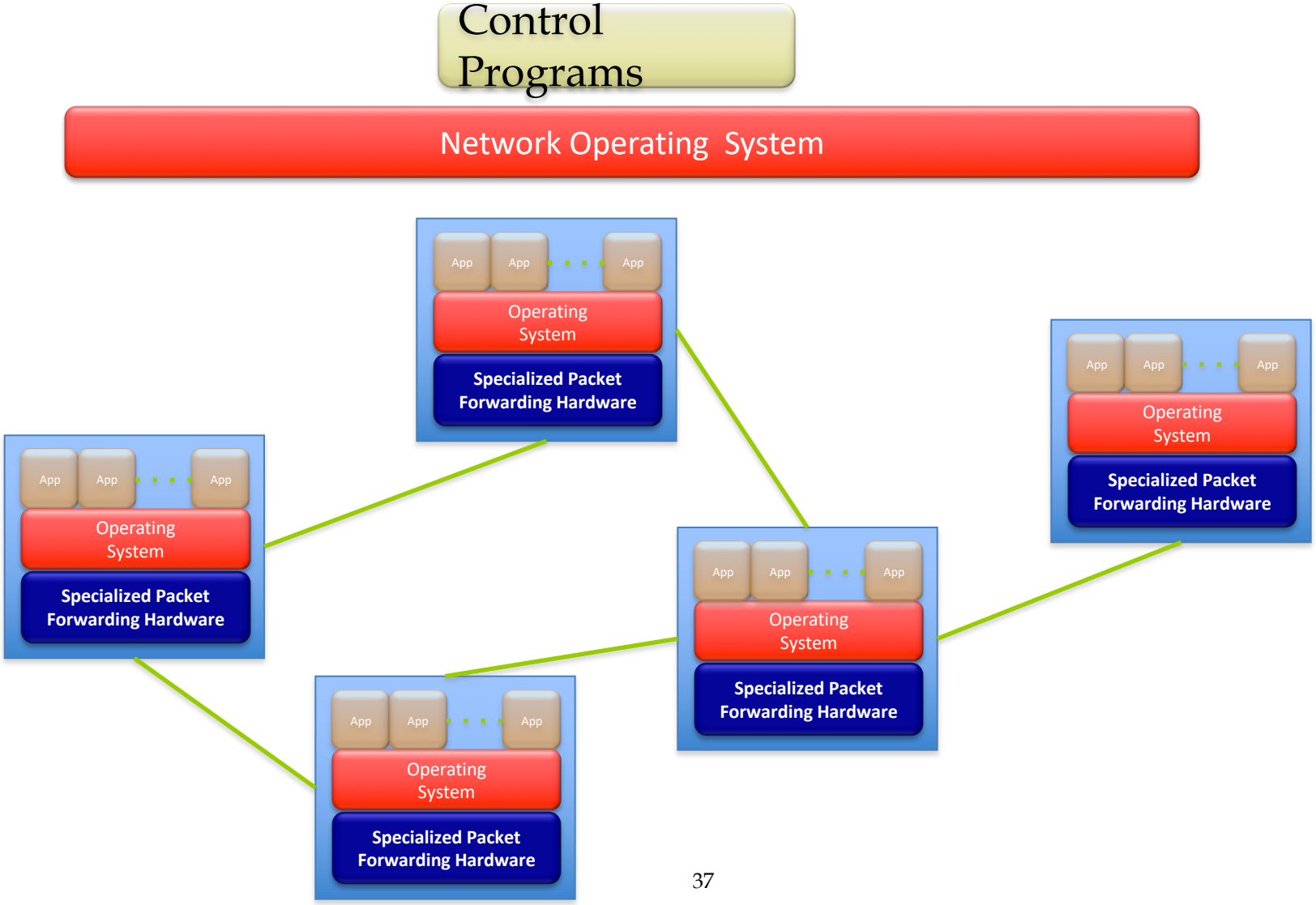
- **No control plane abstraction for the whole network!**
- **It's like old times – when there was no OS…**

Wilkes with the EDSAC, 1949

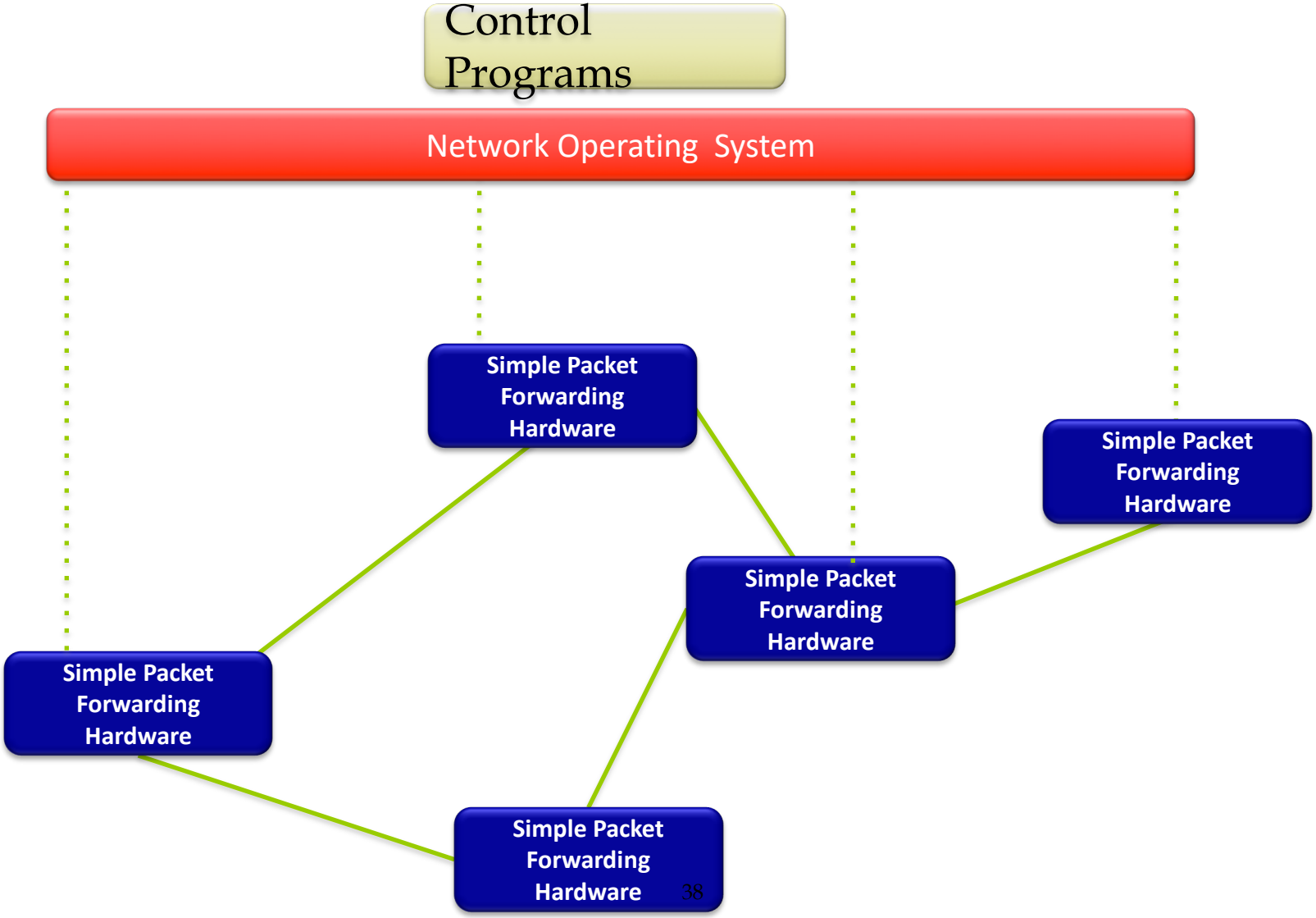
Idea: An OS for Networks



Idea: An OS for Networks



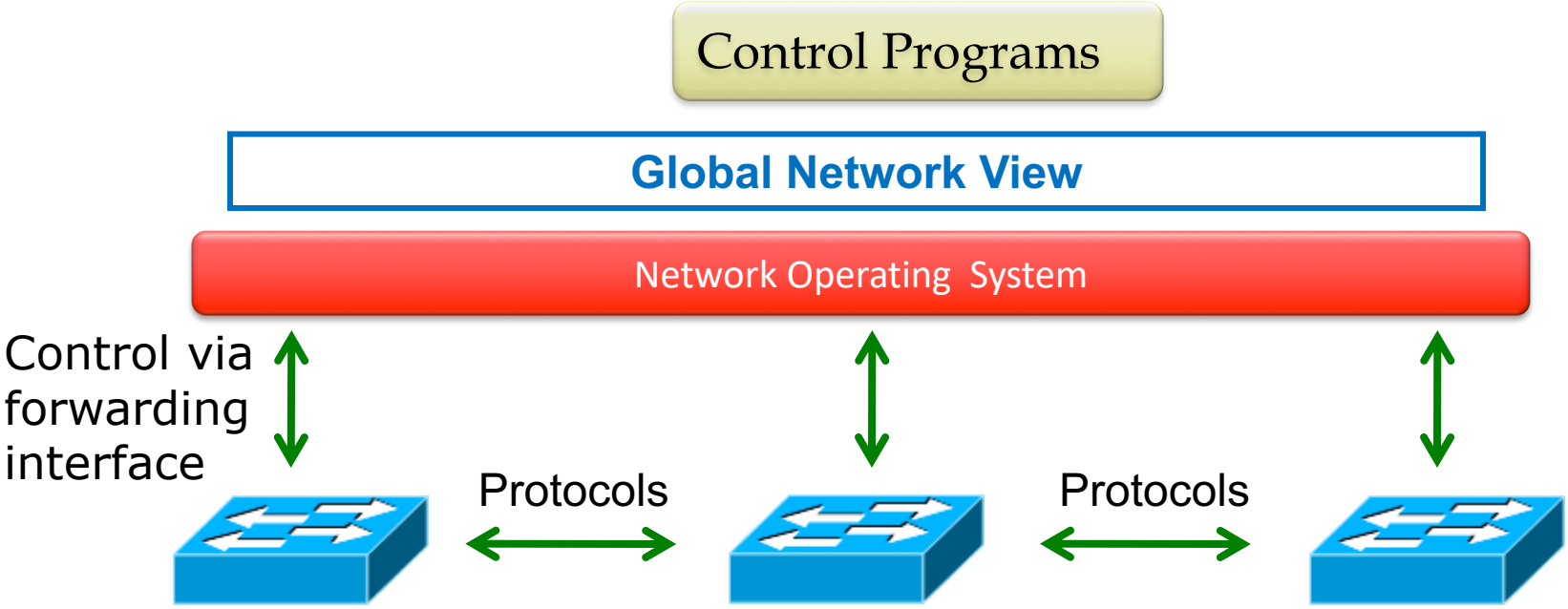
Idea: An OS for Networks



Idea: An OS for Networks

- **"NOX: Towards an Operating System for Networks"**

Software-Defined Networking (SDN)



39

Software Defined Networking

- **No longer designing distributed control protocols**
- **Much easier to write, verify, maintain, …**
  - An interface for programming
- **NOS serves as fundamental control block**
  - With a global view of network

Software Defined Networking

- **Examples**
  - Ethane: network-wide access-control
  - Power Management

Software Defined Networking

- **Questions:**
  - How to obtain global information?
  - What are the configurations?
  - How to implement?
  - How is the scalability?
  - How does it really work?

Outline

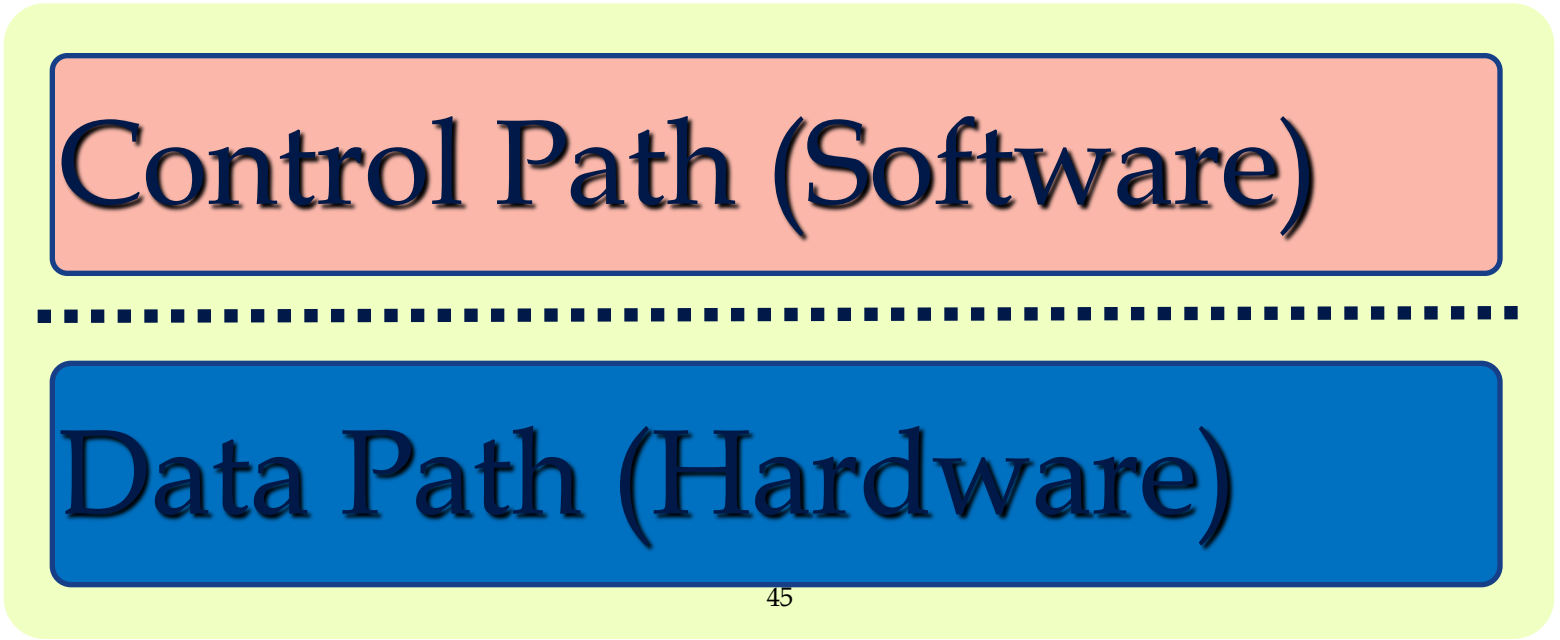
- **What is SDN?**
  - Limitations of current networks
  - The idea of Network OS
- **What is OpenFlow?**
  - How it helps SDN
- **The current status & the future of SDN**
- **Conclusions**

OpenFlow

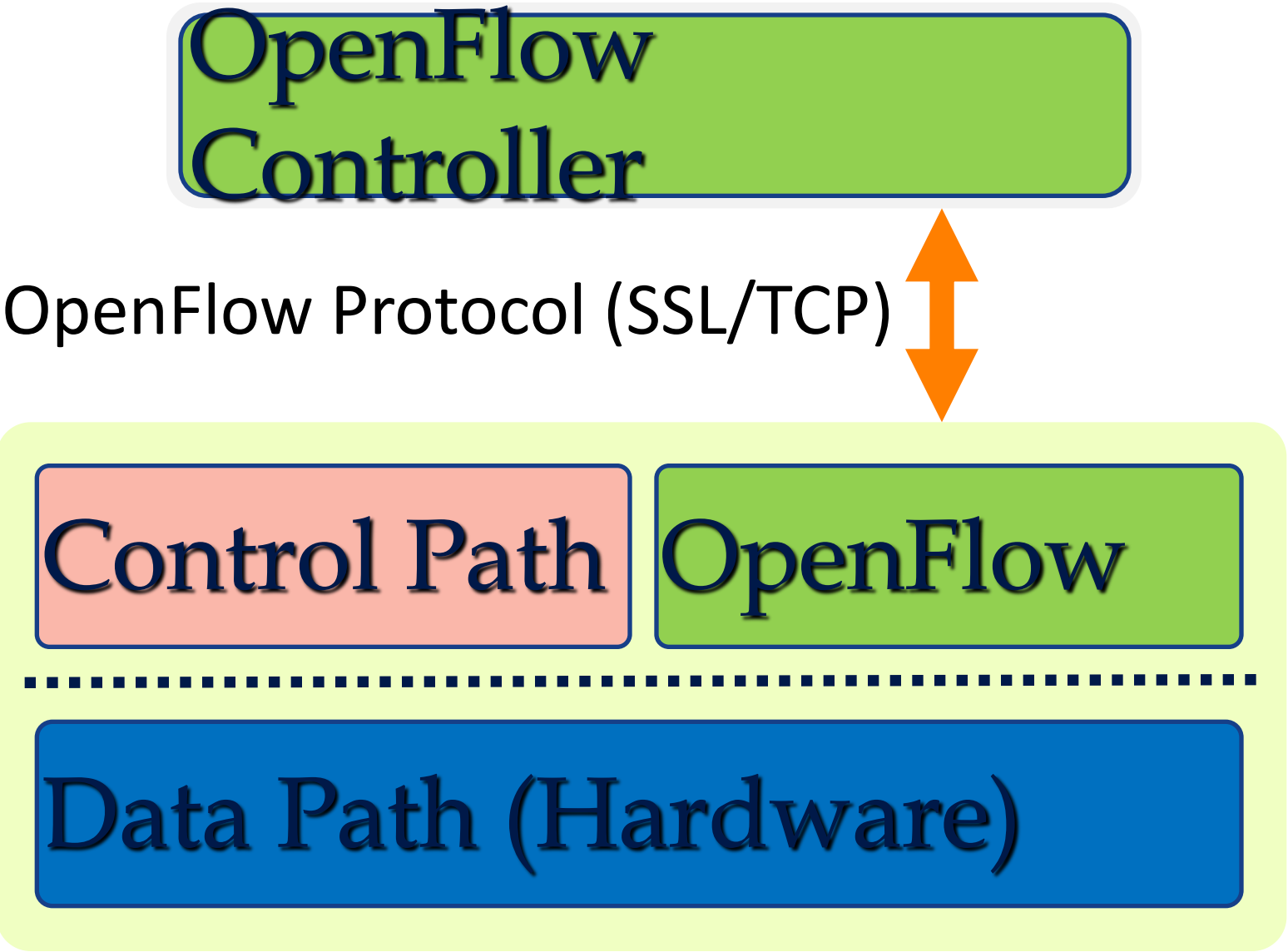
- **“OpenFlow: Enabling Innovation in Campus Networks”**

- **Like hardware drivers**
  - interface between switches and Network OS

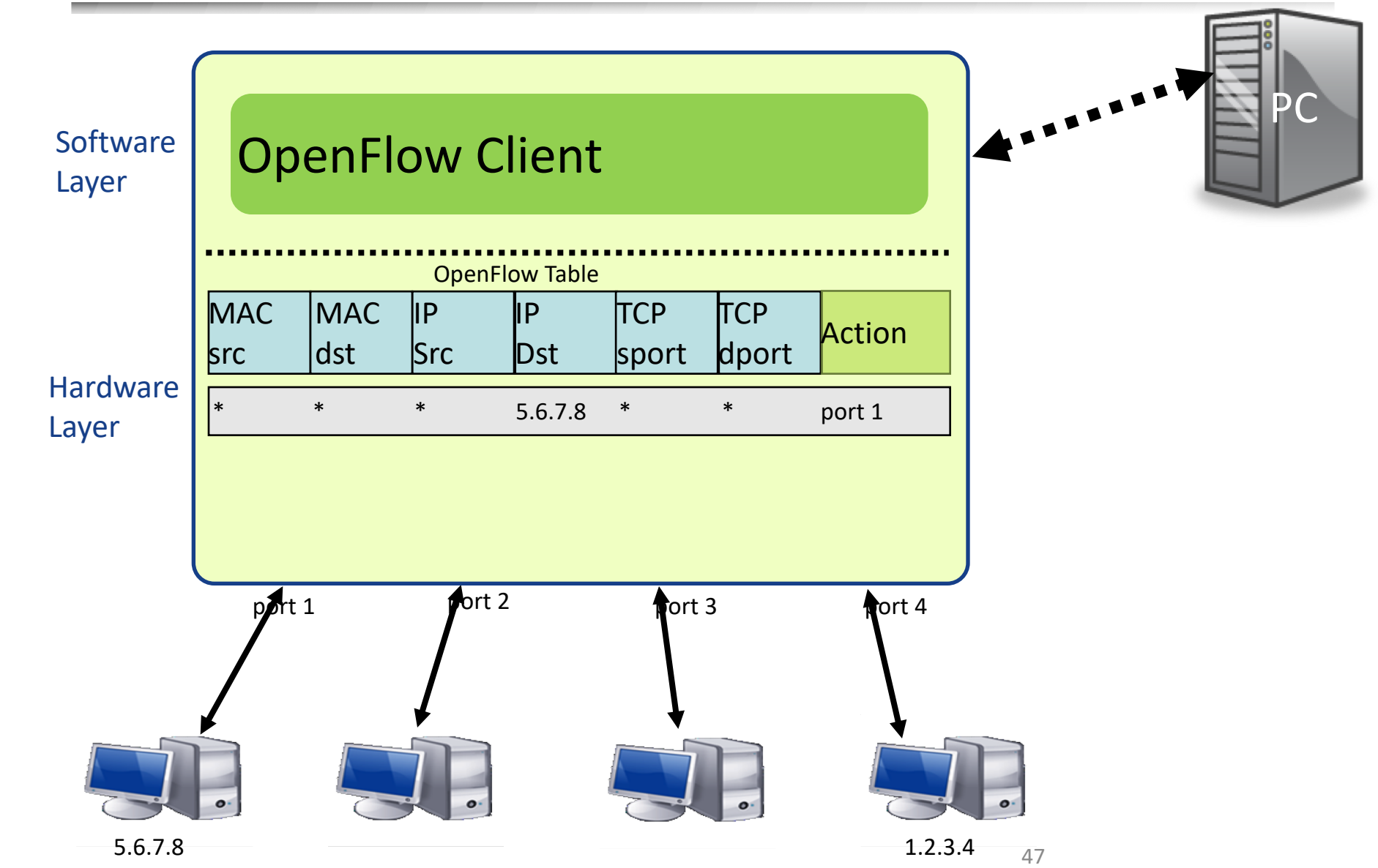
OpenFlow



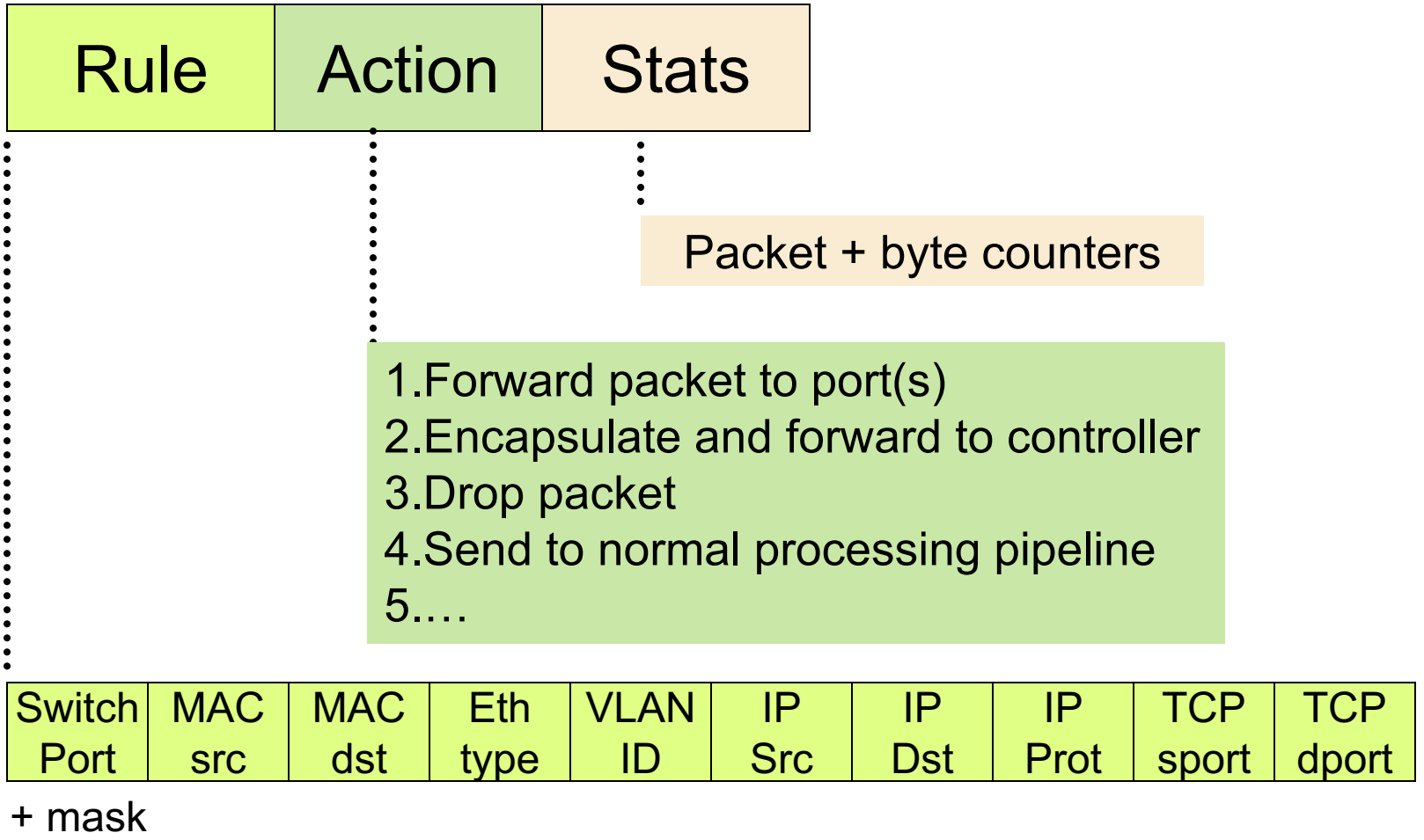
OpenFlow


OpenFlow Controller
OpenFlow Protocol (SSL/TCP)
Control Path
OpenFlow
Data Path (Hardware)

OpenFlow Switching



OpenFlow Table Entry



OpenFlow Examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Routing

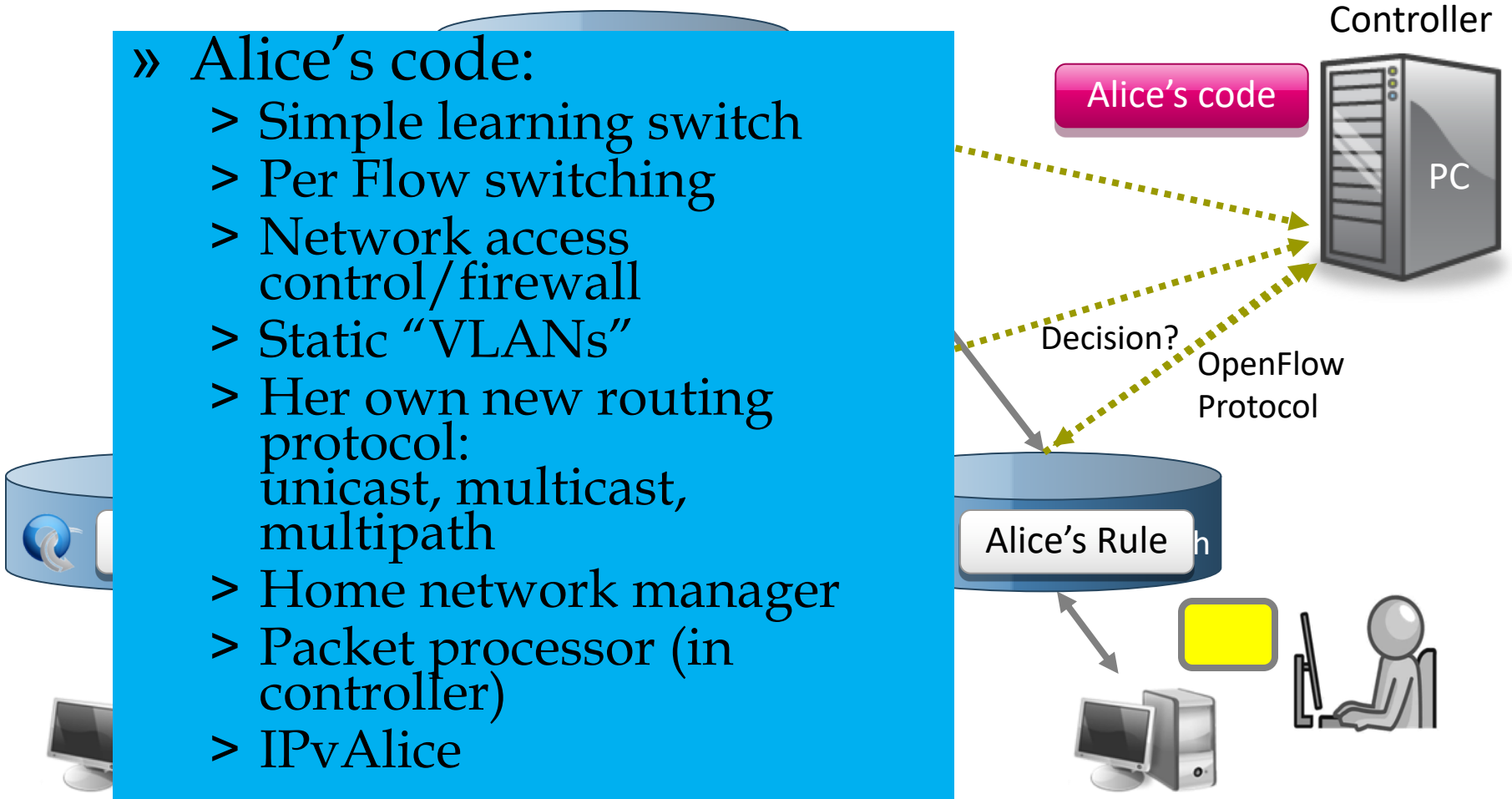
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

OpenFlow Usage

- Alice's code:
  - Simple learning switch
  - Per Flow switching
  - Network access control/firewall
  - Static "VLANs"
  - Her own new routing protocol: unicast, multicast, multipath
  - Home network manager
  - Packet processor (in controller)
  - IPvAlice



OpenFlow

- Standard way to control flow-tables in commercial switches and routers
- Just need to update firmware
- Essential to the implementation of SDN

Centralized/Distributed Control

- **"Onix: A Distributed Control Platform for Large-scale Production Networks"**