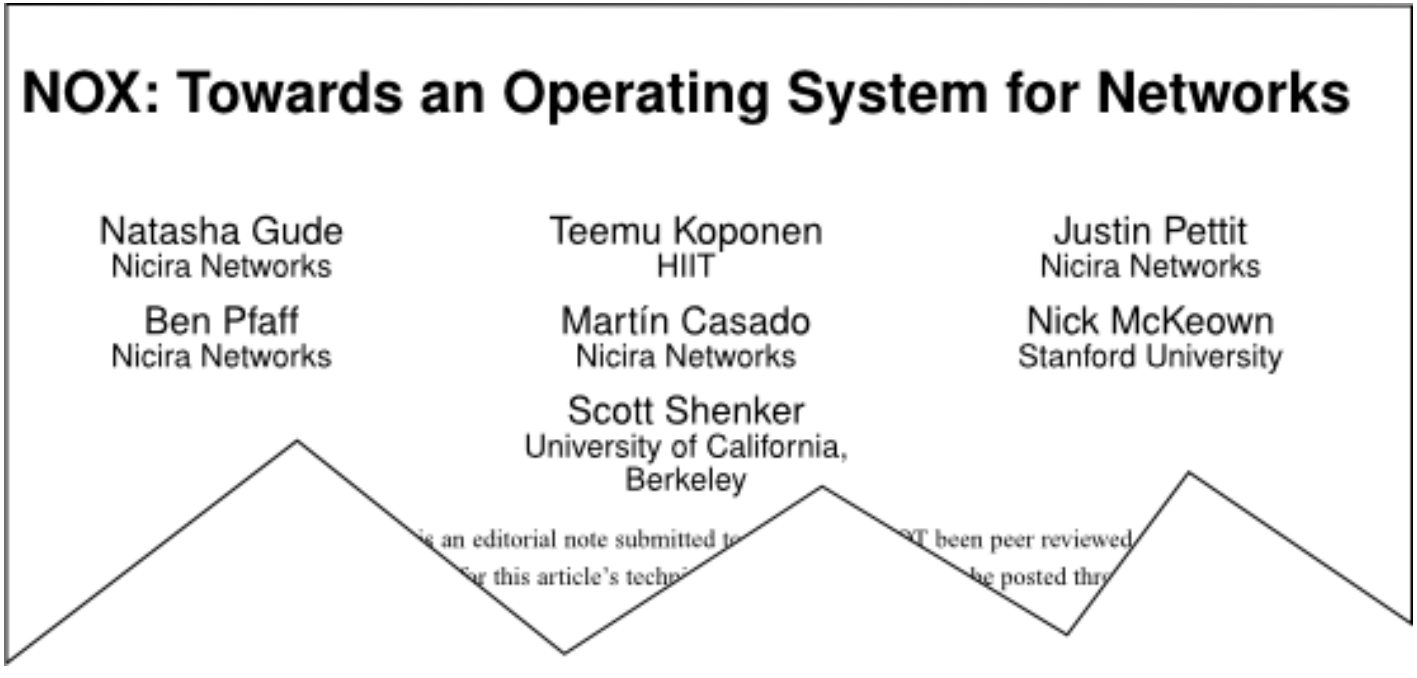
# Applying Operating System Principles to SDN Controller Design

Matthew Monaco Oliver Michel Eric Keller

University of Colorado, Boulder

HotNets-XII
November 21, 2013

University of Colorado **Boulder**



NOX: Towards an Operating System for Networks

Natasha Gude
Nicira Networks

Teemu Koponen
HIIT

Justin Pettit
Nicira Networks

Ben Pfaff
Nicira Networks

Martín Casado
Nicira Networks

Nick McKeown
Stanford University

Scott Shenker
University of California, Berkeley

*What we clearly need is an "operating system" for networks. . .*

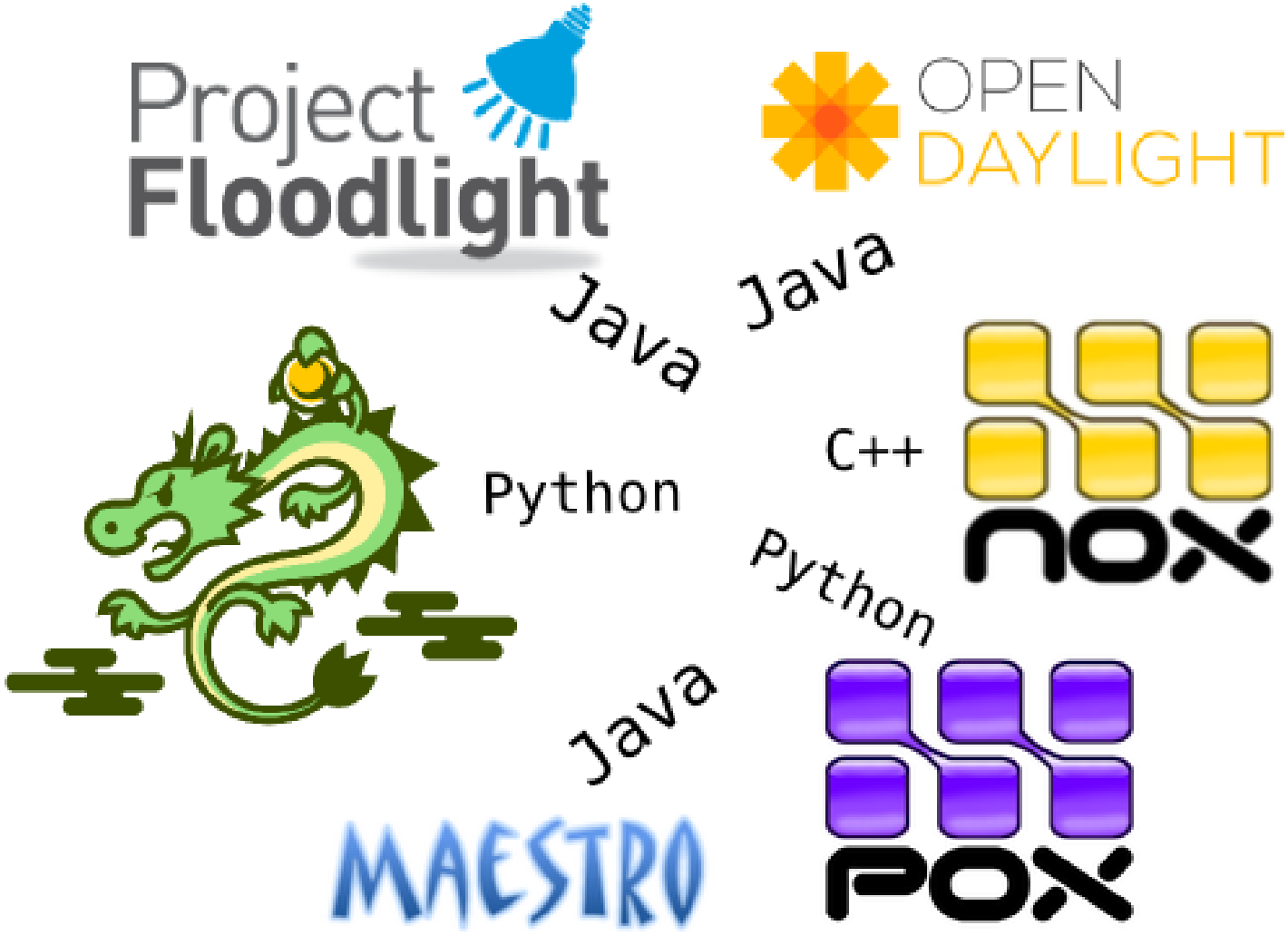
Project Floodlight

OPEN DAYLIGHT

nox

MAESTRO

POX


Project Floodlight
OPEN DAYLIGHT
Java
Java
C++
Python
Python
Java
MAESTRO
nox
POX

```
git clone
eclipse
```



Java

Java

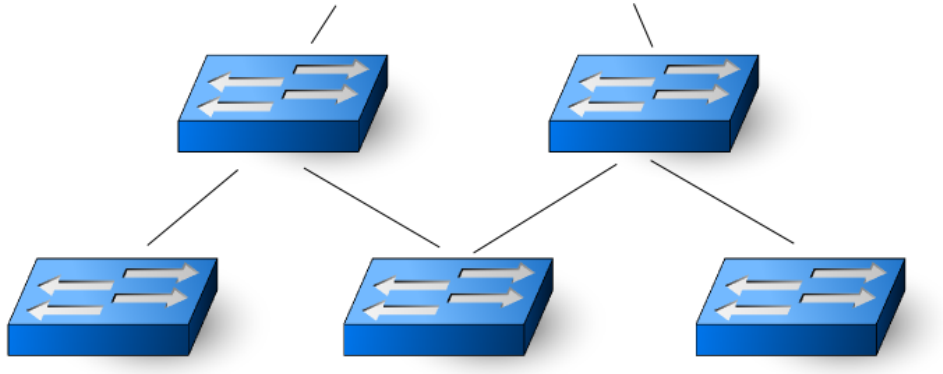
Python

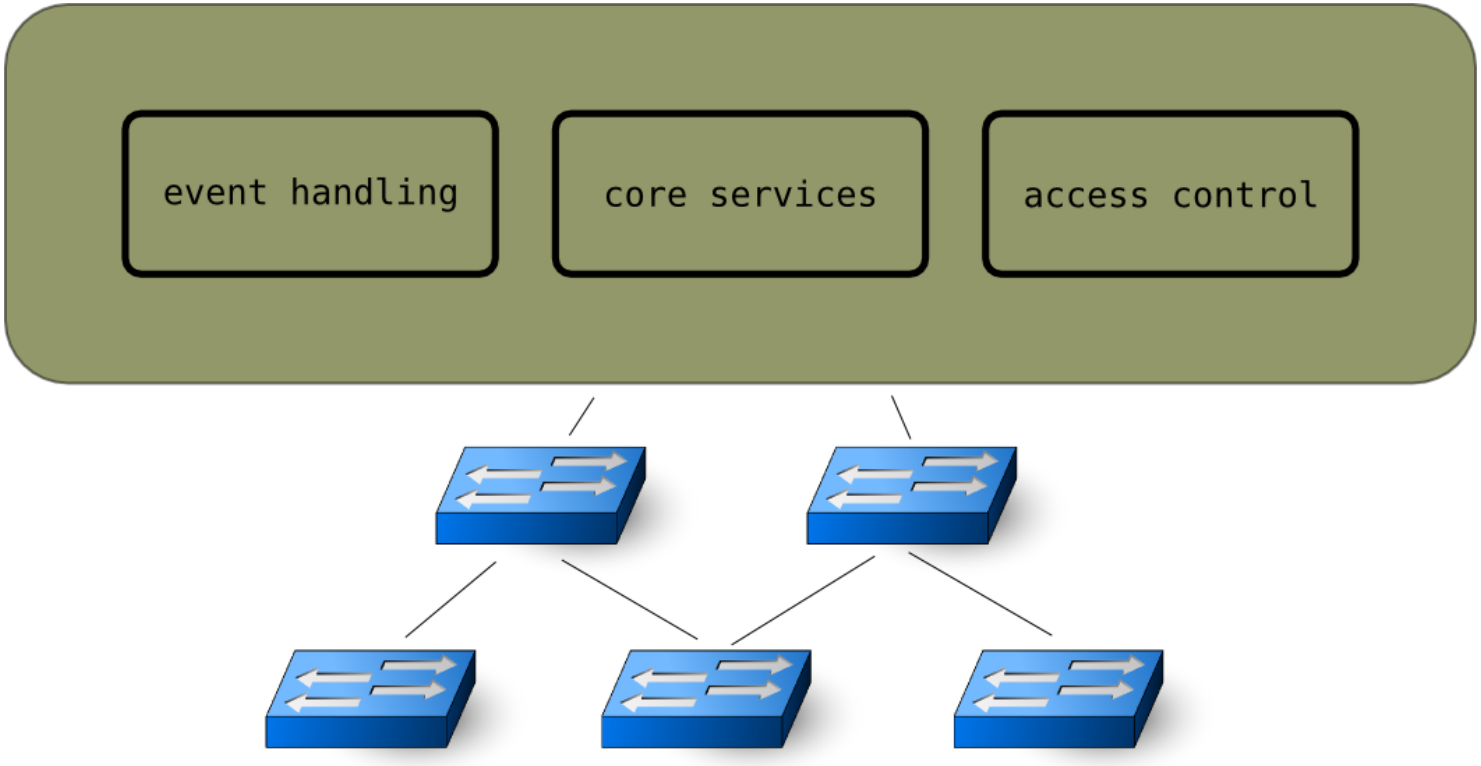
C++


Python

Java


# Monolithic




event handling
core services
access control

# Traditional Operating Systems

# Distributions

- Technically sound

# Distributions

- Technically sound
- Distributions

# Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users

# Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users
- Networks

# Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users
- Networks
  - Datacenters, Universities, Enterprises
  - Small business, Home Users, Hobbyists

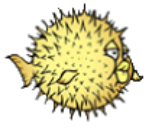
# Software Projects












How do administrators and developers interact with traditional OSes?

# How do administrators and developers interact with traditional OSes?

```
$
```

# How do administrators and developers interact with traditional OSes?

```
$
```

```
$ grep
$ find
$ sed
$ sort
$ cat
```

# Detective Work

```
$ cd /var/log
$ grep some_problem syslog
```

## Performance Analysis

```
$ ps -A | sort -k3nr | head -10
$ kill -TERM 12345
```

# Simple Automation

```
#!/bin/bash

procs=(
    ps -A | sort -k3nr | head -$1 | tr -s ' ' \
    | cut -d' ' -f9
)

for p in "${procs[@]}"; do
    printf "%d is misbehaving\n" "$p" >&2
    kill -TERM "$p"
    sleep 3
    kill -KILL "$p"
done
```

## Large Applications

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    puts("Hello, World!");
    return EXIT_SUCCESS;
}
```

```
$ ./configure
$ make
$ sudo make install
```

What can we learn from a traditional operating system?

What can we **use** from a traditional operating system?

Integrate more tightly with Linux

Integrate more tightly with Linux

Use off-the-shelf technologies

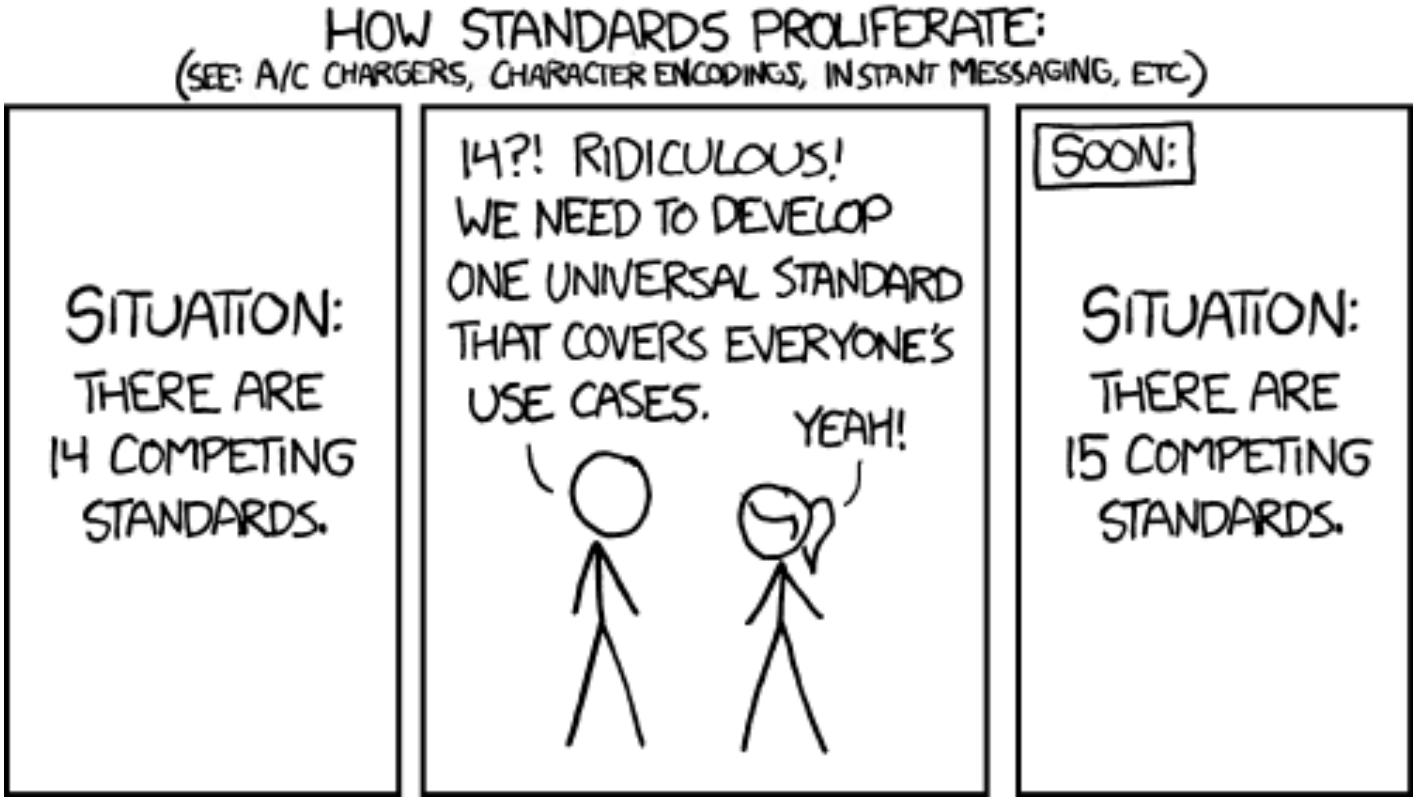
Integrate more tightly with Linux

Use off-the-shelf technologies

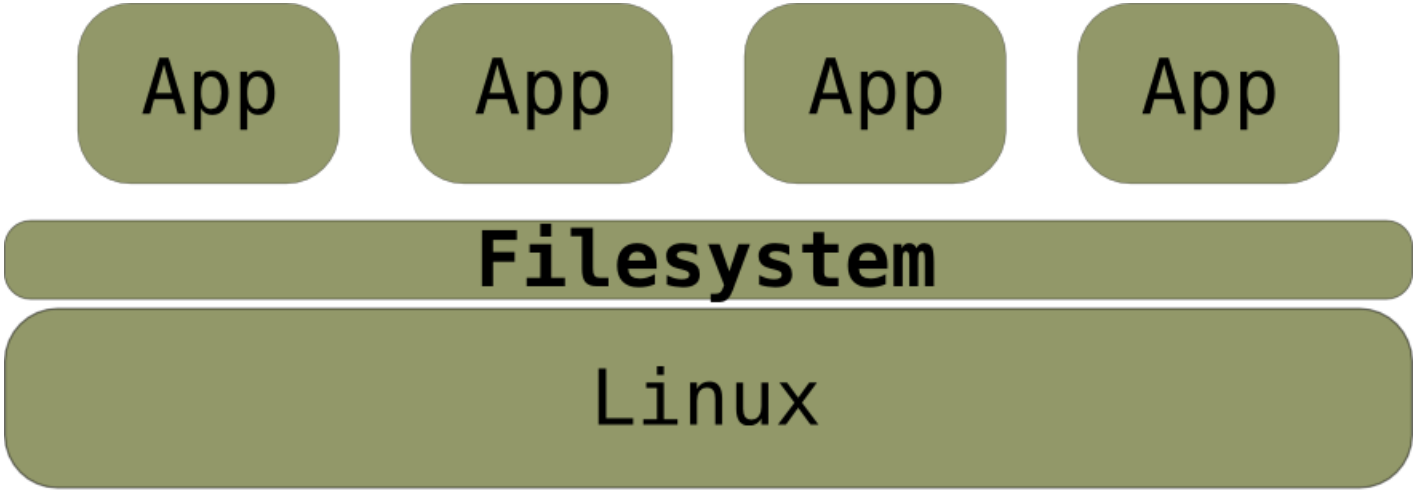
Encourage active ecosystem

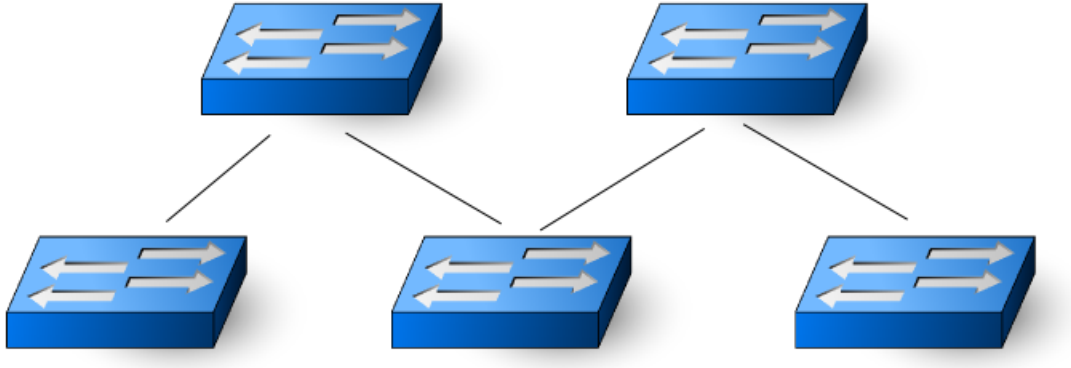
# Yanc: Yet Another Network Controller

# Obligatory XKCD

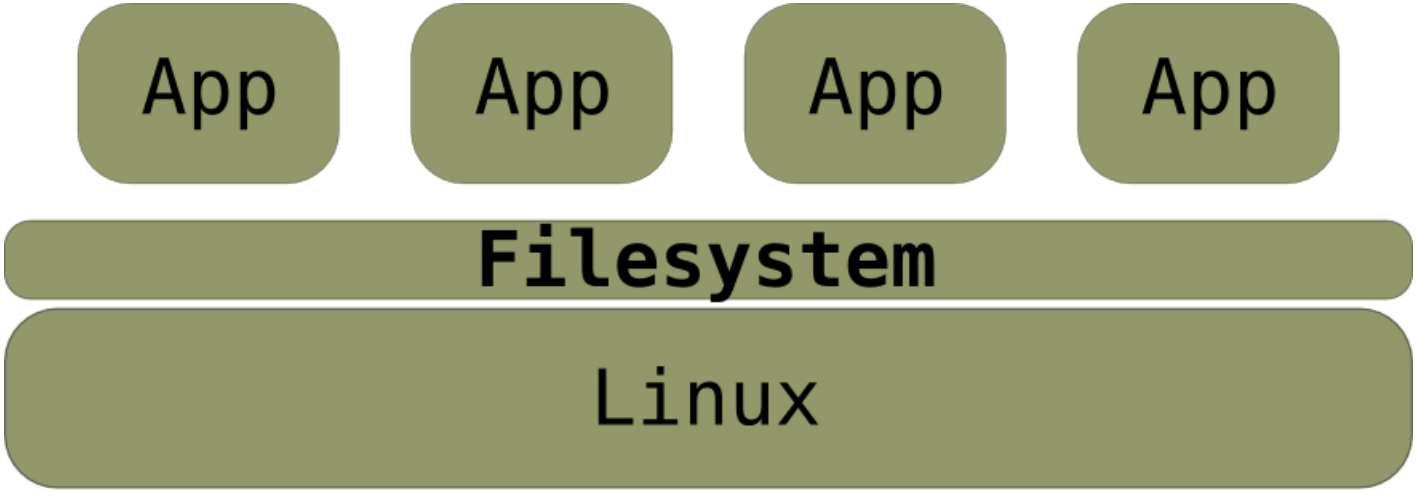


# Yanc: Yet Another Network Controller

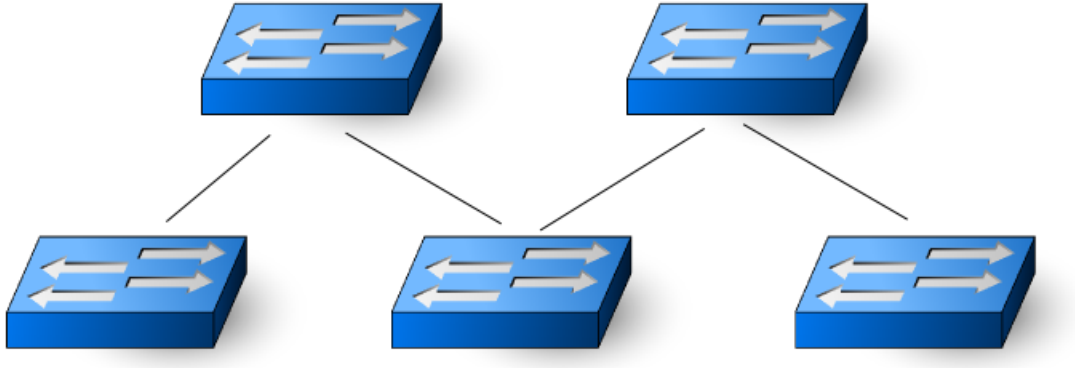


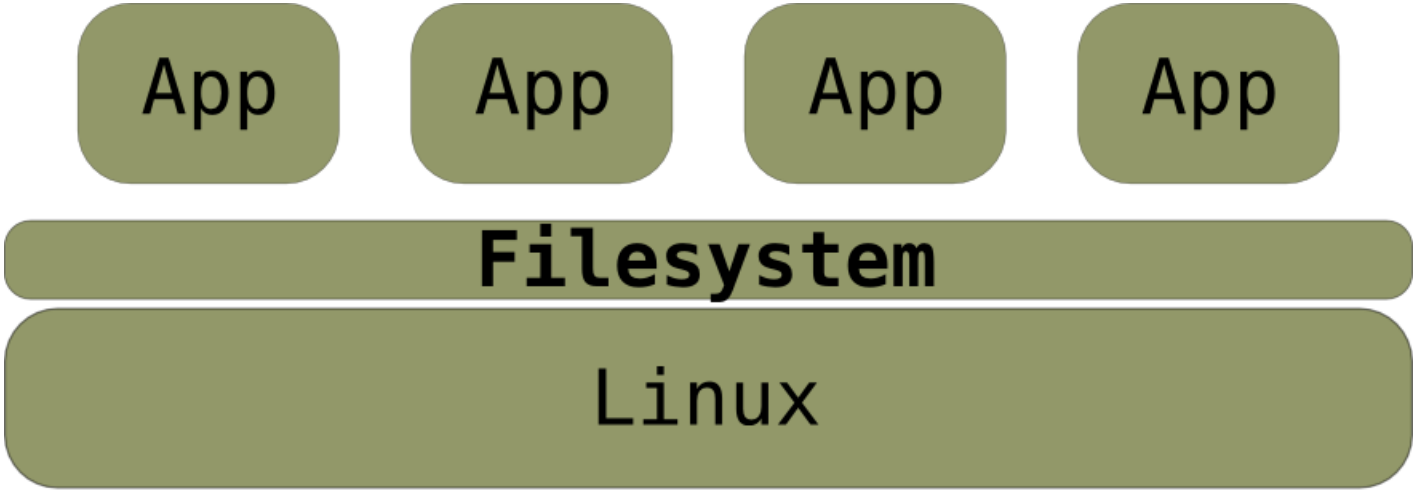
# Yanc: Yet Another Network Controller




Everything is a file

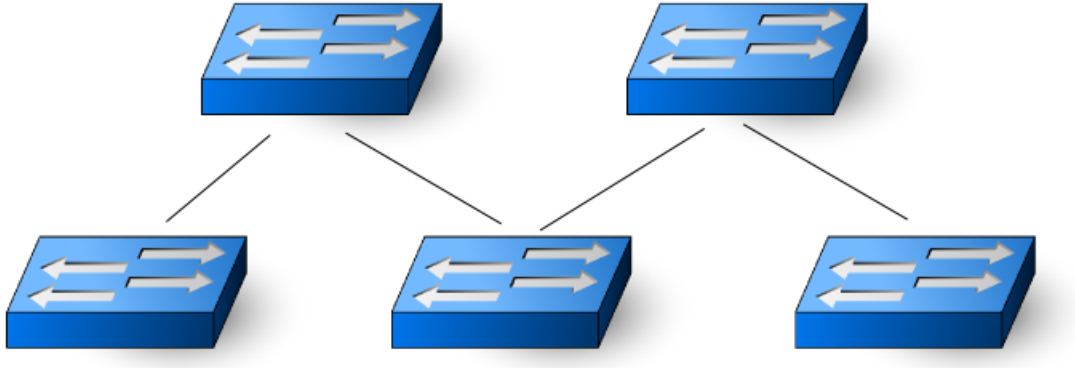


# Yanc: Yet Another Network Controller



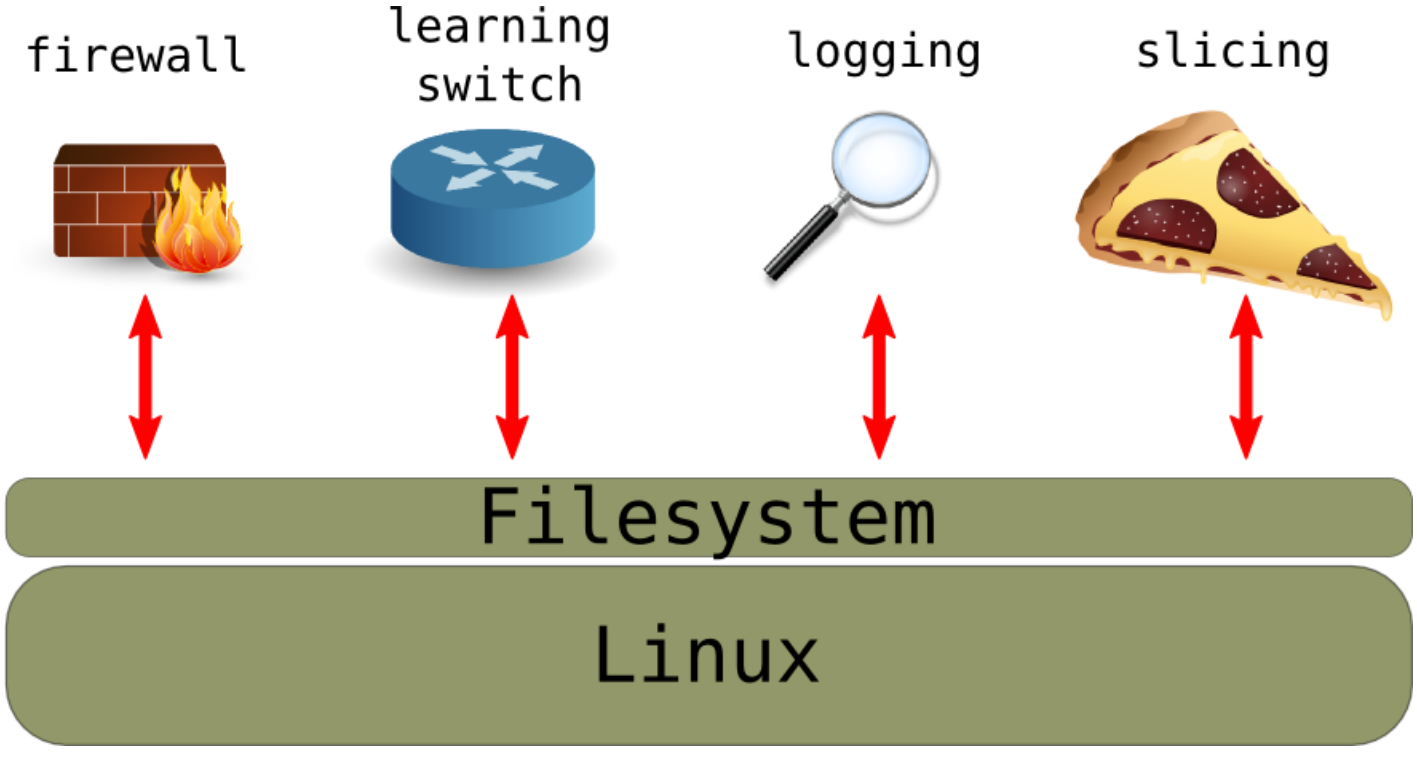

Everything is a file

Procfs, Sysfs

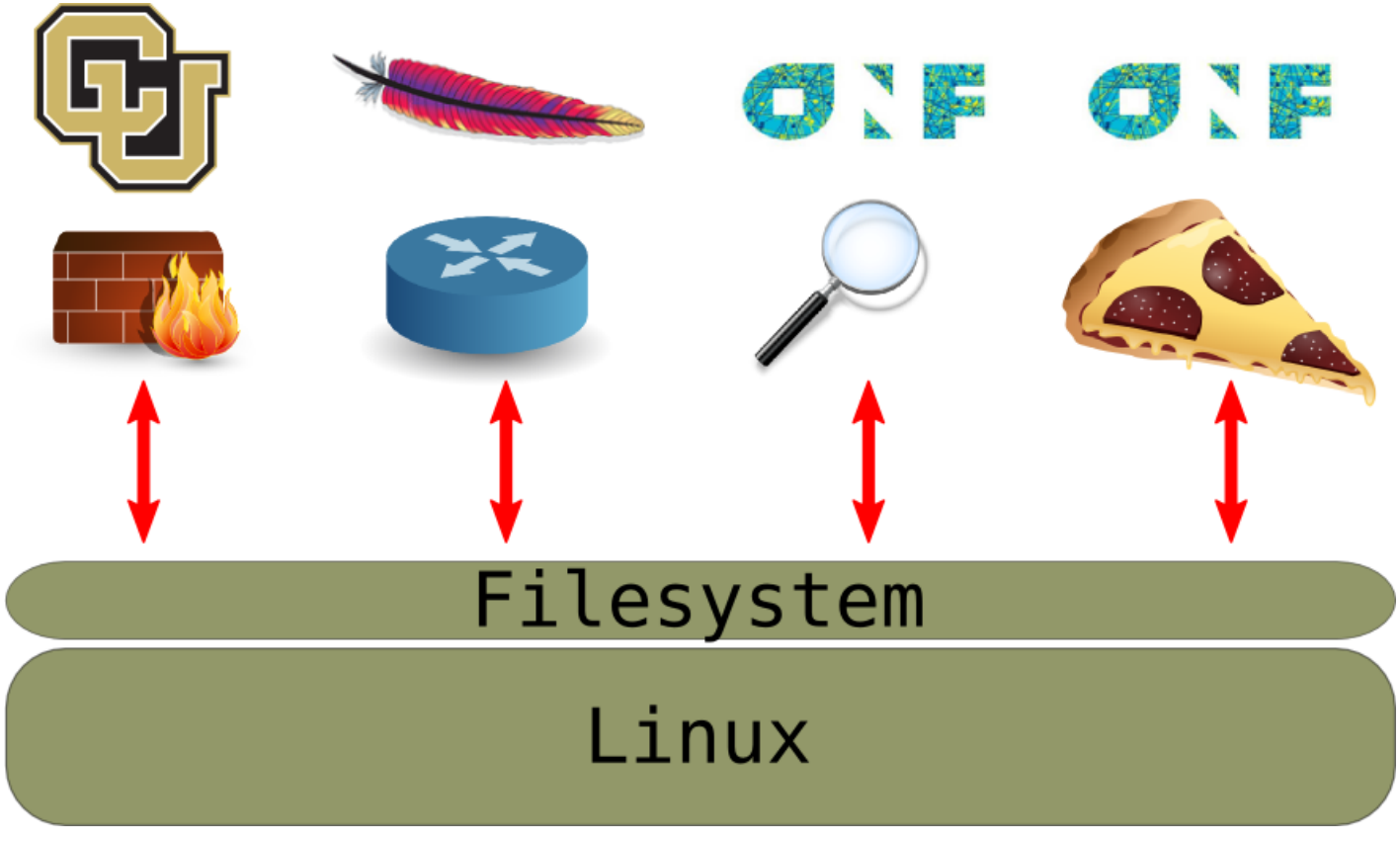


# Why a filesystem?

# Logically Distinct Applications



# Independent Development



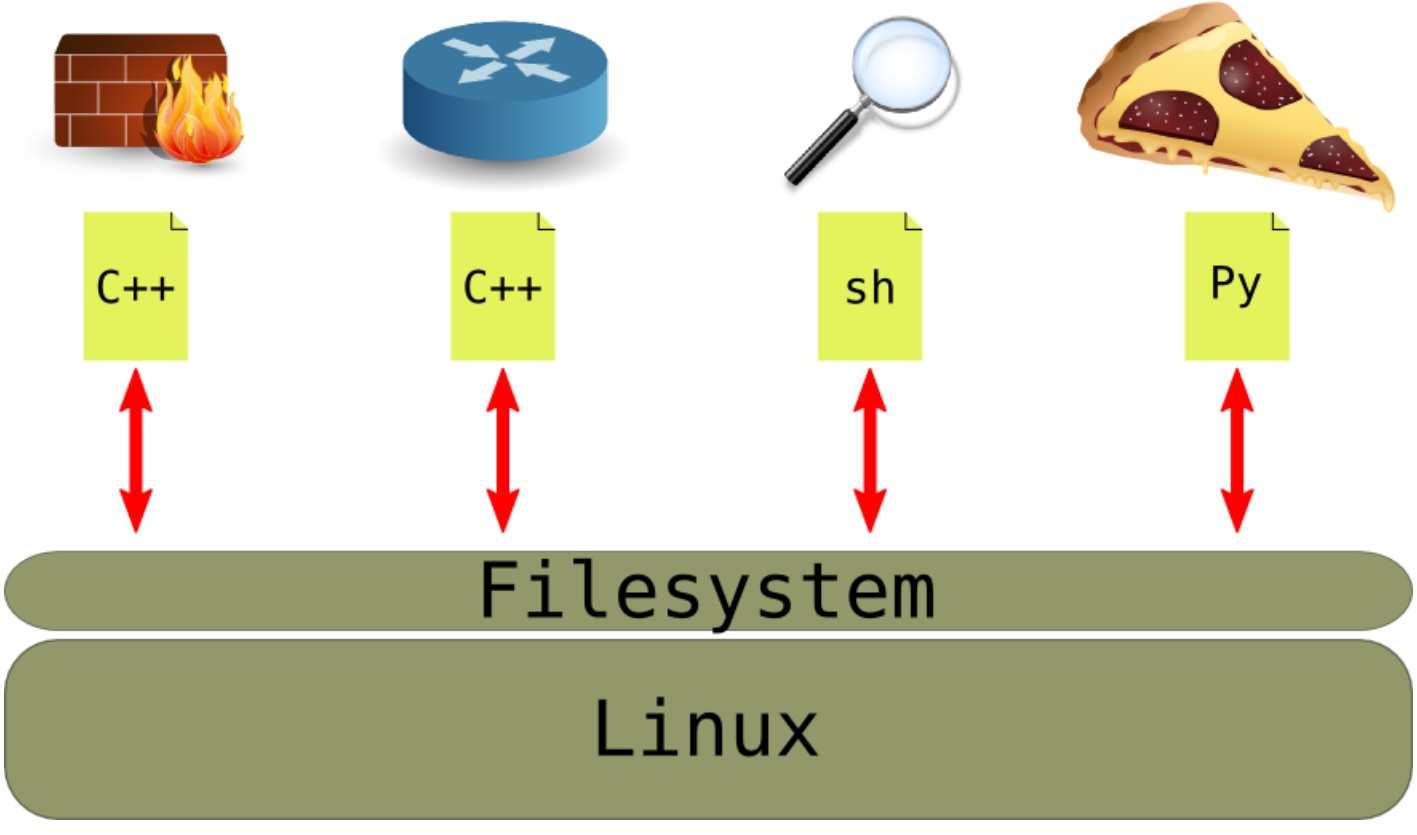
## Independent Development

```
# apt-get install yanc-learning-switch
```

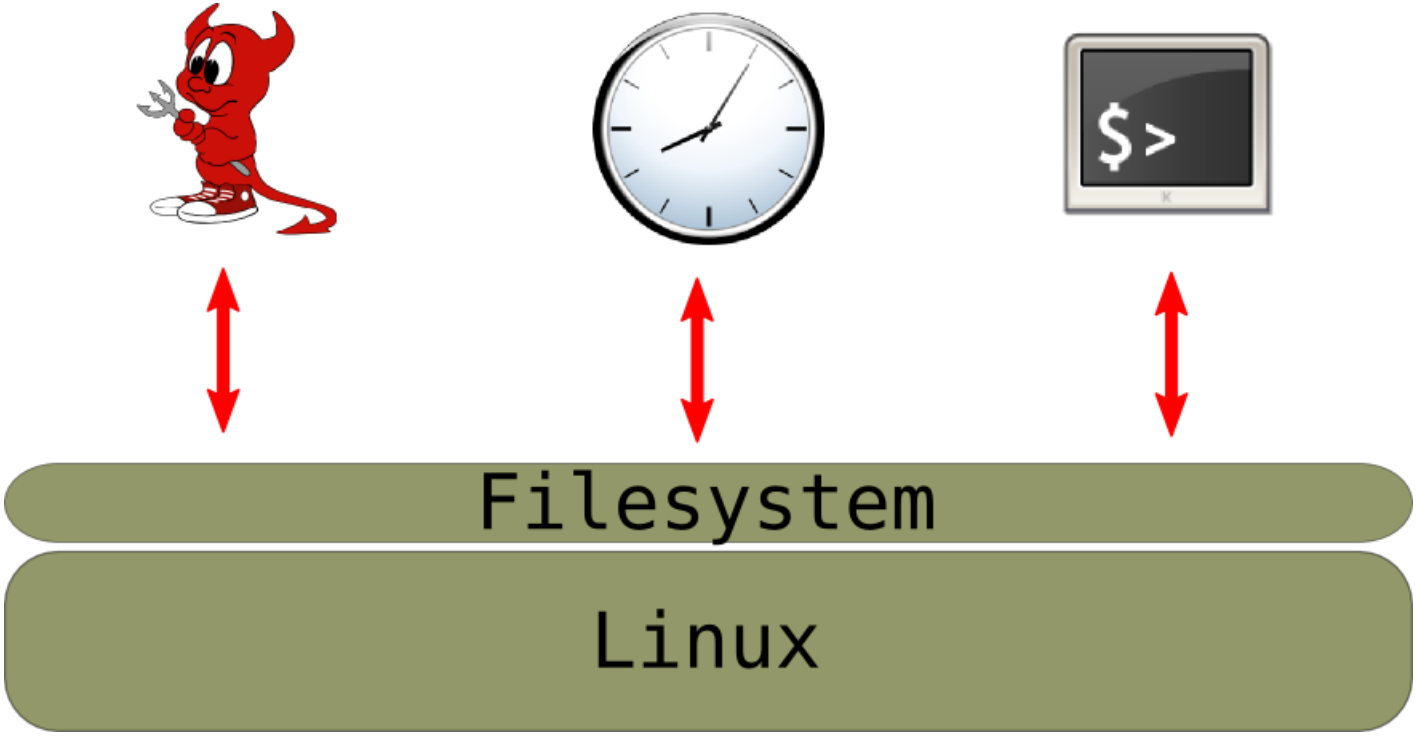
```
# apt-get install yanc-router
```

```
$ git clone git@github.com/mmonaco/yanc-firewall && cd
    yanc-firewall
$ make
# make install
```

# Any Programming Language



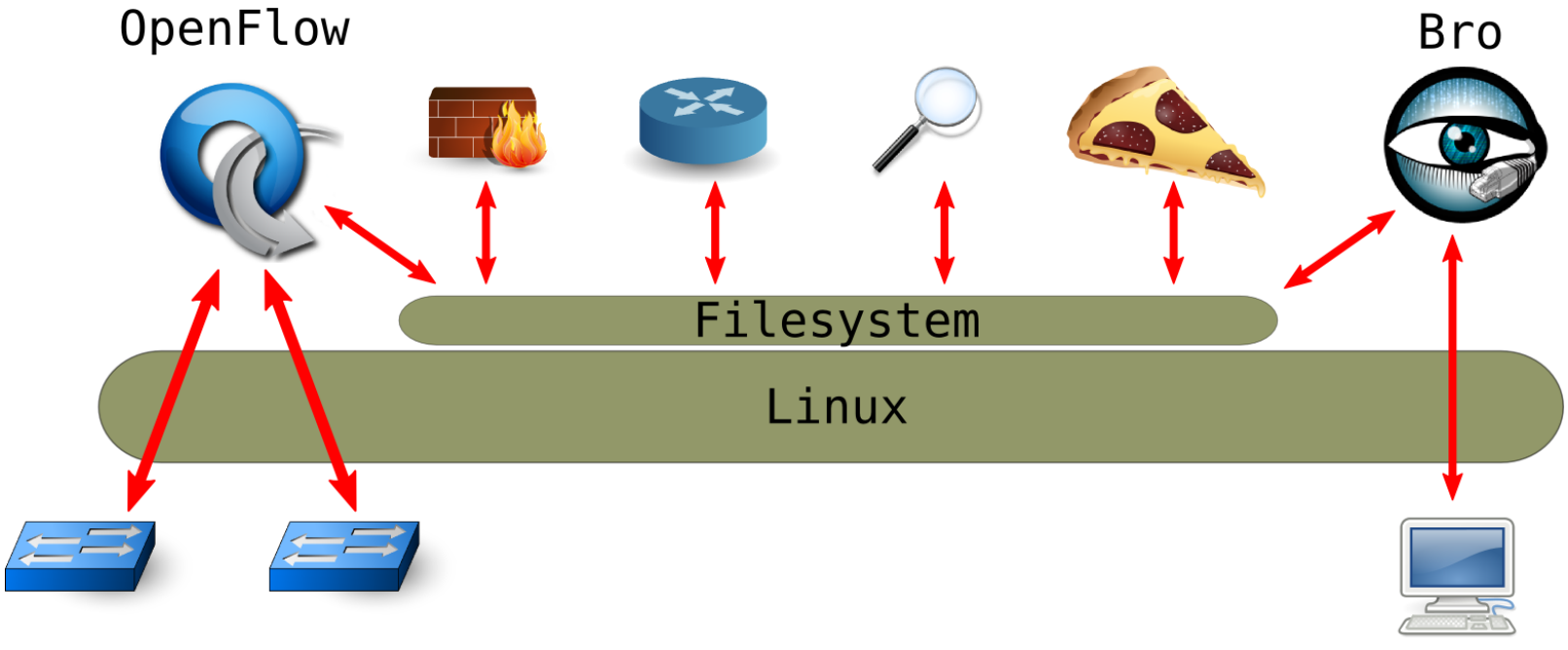
# Any Form



## Other Technologies

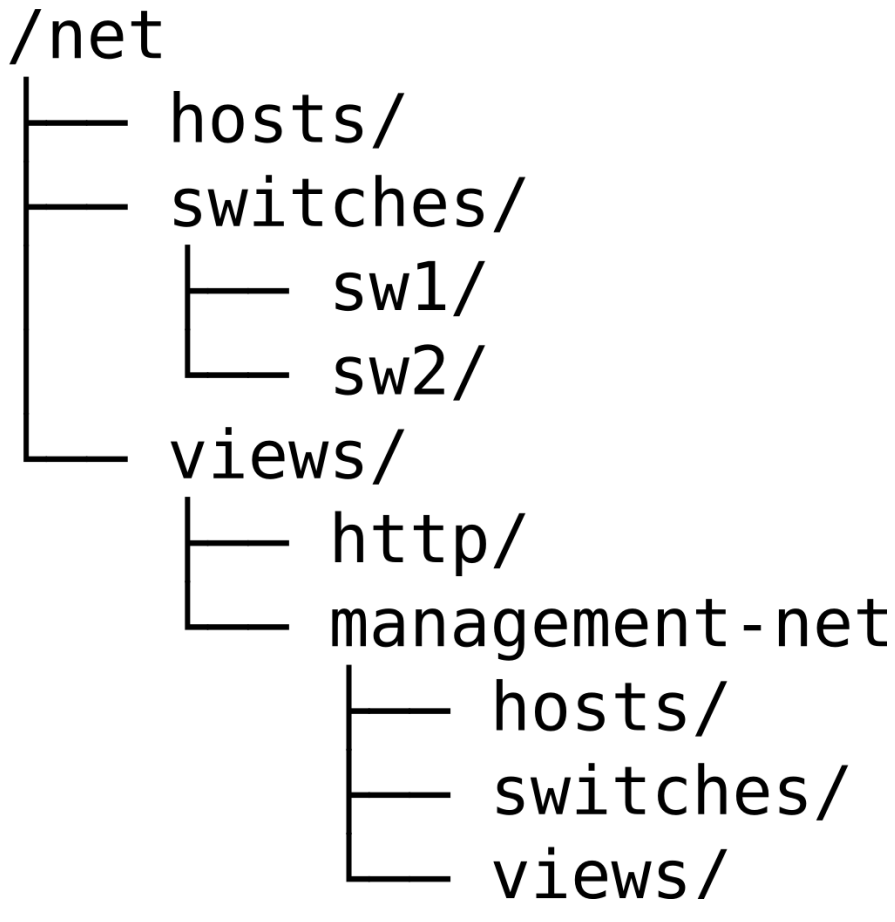
- Inotify
- File permissions and ACLs
- Namespaces and CGroups
- Layered Filesystems

# Decouple from Hardware

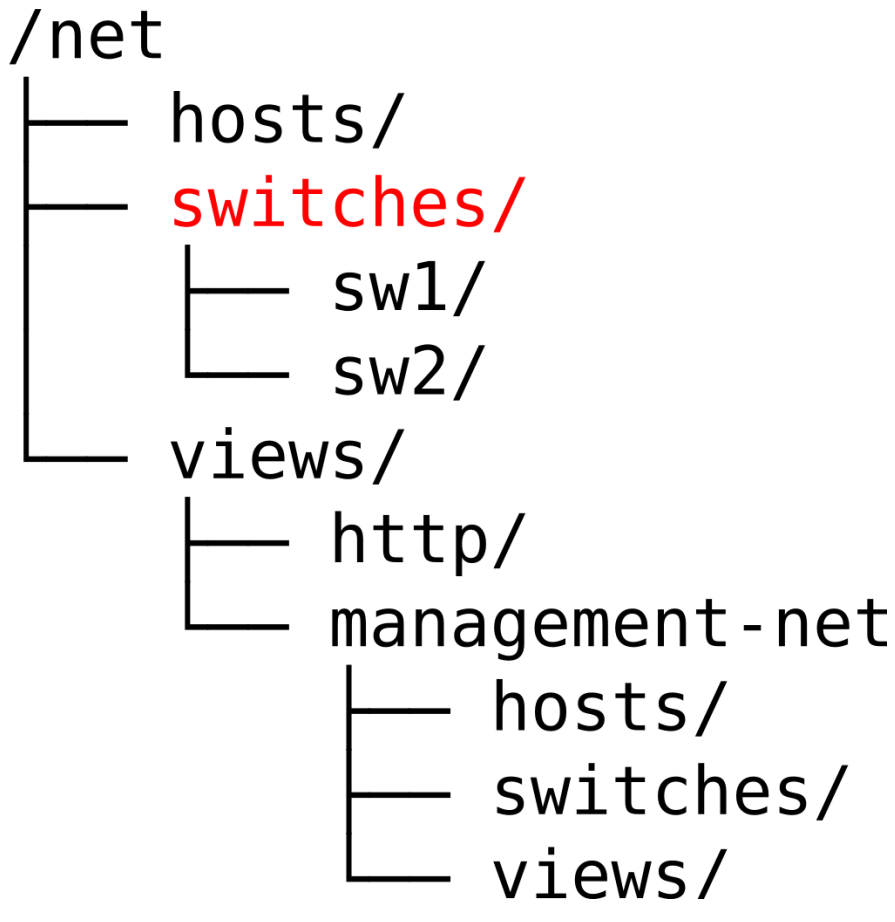


# The Yanc Filesystem

## Root Directory



## Root Directory



## Switch Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        ├── flows/
        ├── packet_in/
        ├── packet_out/
        ├── ports/
        ├── actions
        ├── capabilities
        ├── id
        └── n_buffers
```

## Switch Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        ├── flows/
        ├── packet_in/
        ├── packet_out/
        ├── ports/
        ├── actions
        ├── capabilities
        ├── id
        └── n_buffers
```

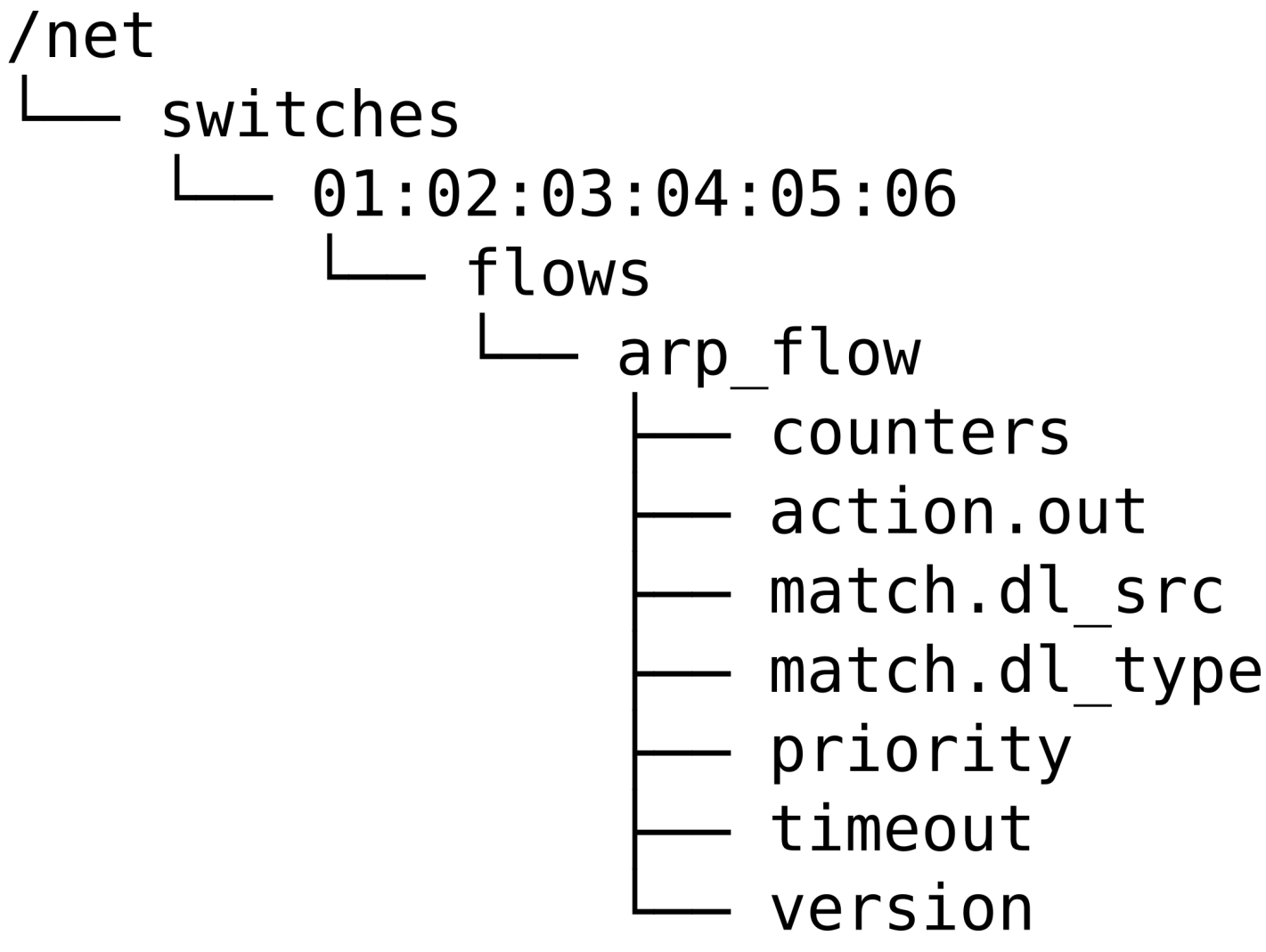
# Port Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        └── ports
            └── LOCAL
                ├── config.port_down
                ├── hw_addr
                ├── peer -> /dev/null
                ├── port_no
                ├── stats.rx_bytes
                └── stats.rx_packets
```

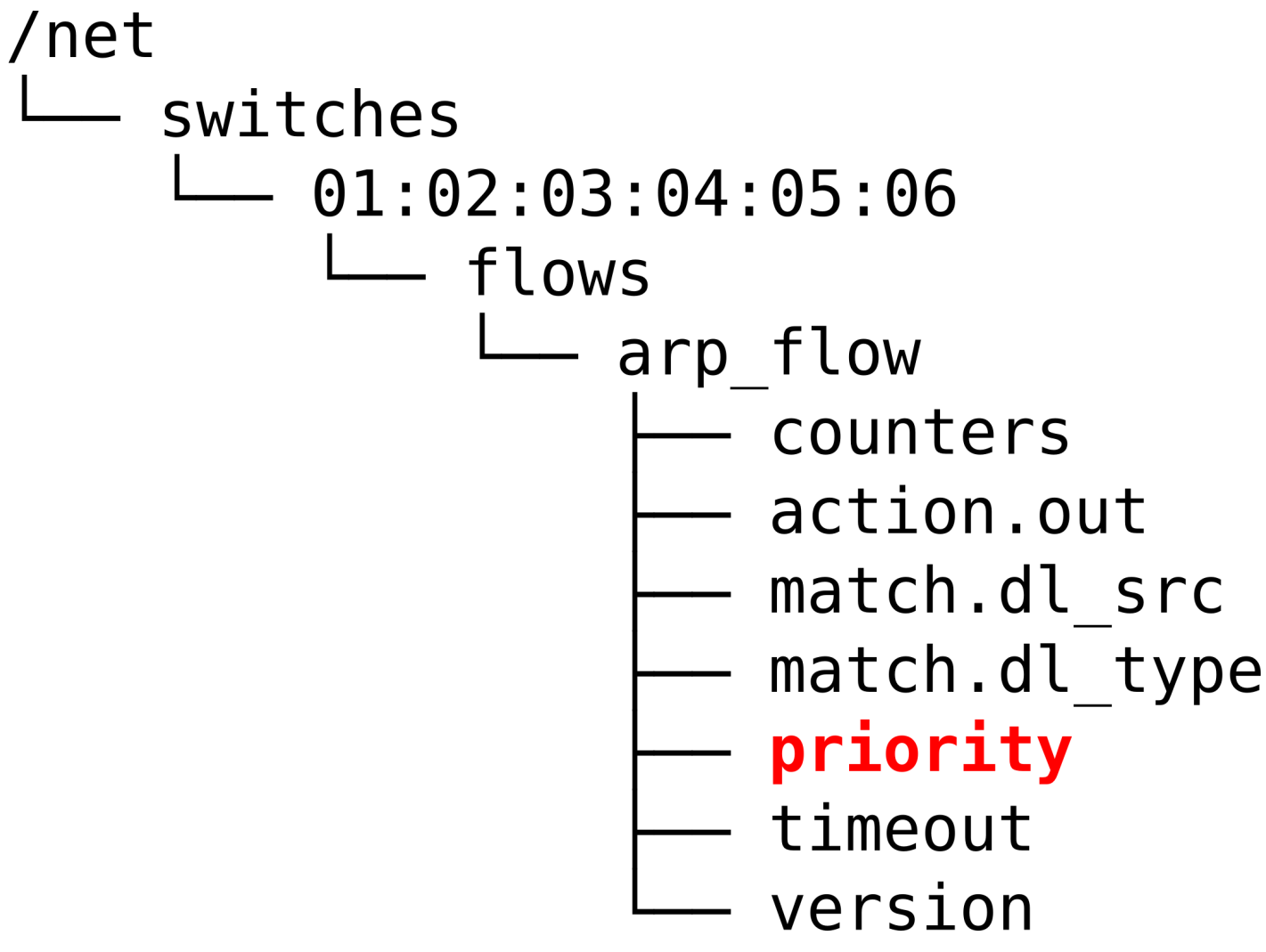
# Port Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        └── ports
            └── LOCAL
                ├── config.port_down
                ├── hw_addr
                ├── peer -> /dev/null
                ├── port_no
                ├── stats.rx_bytes
                └── stats.rx_packets
```

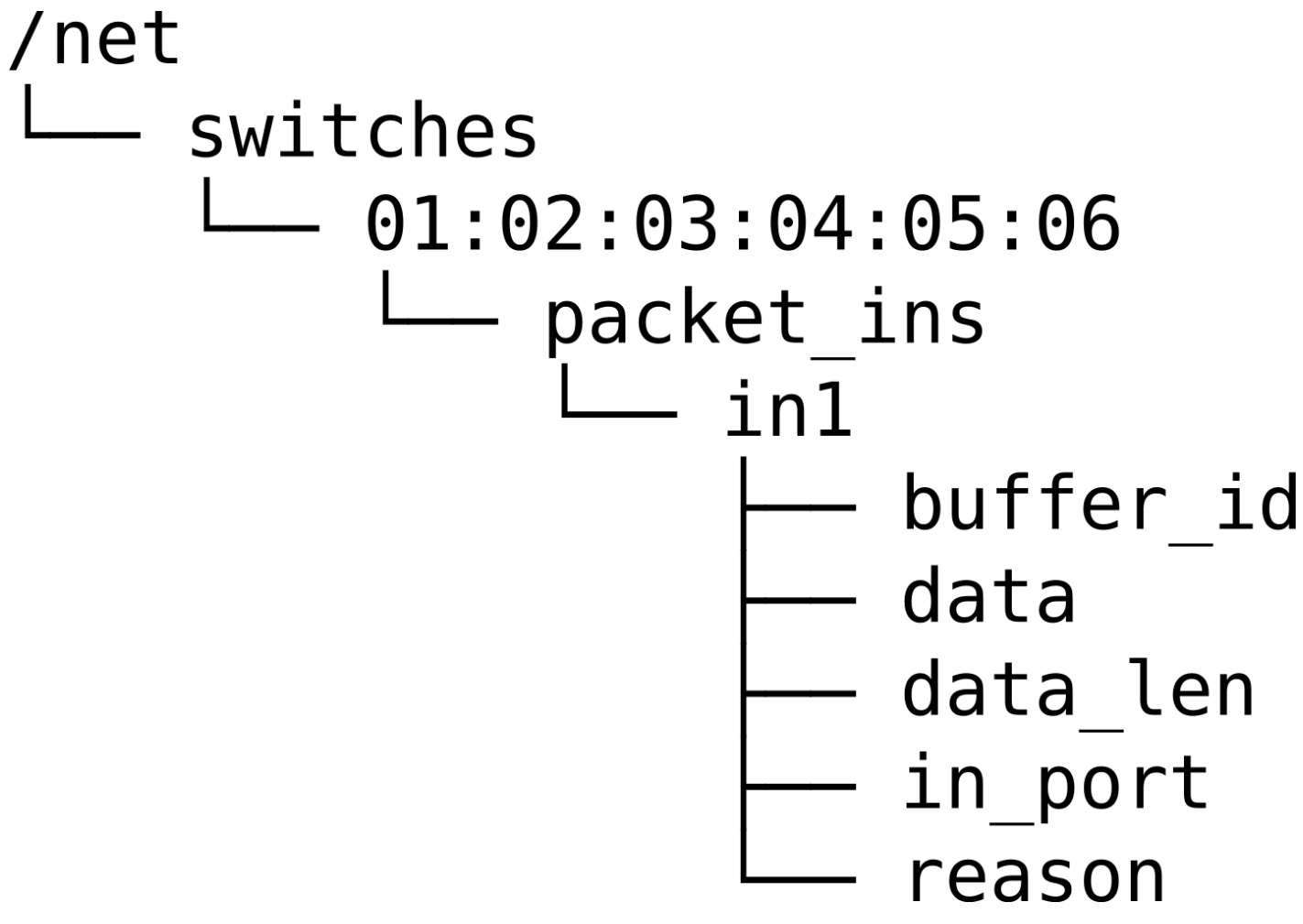
## Flow Entry Directory



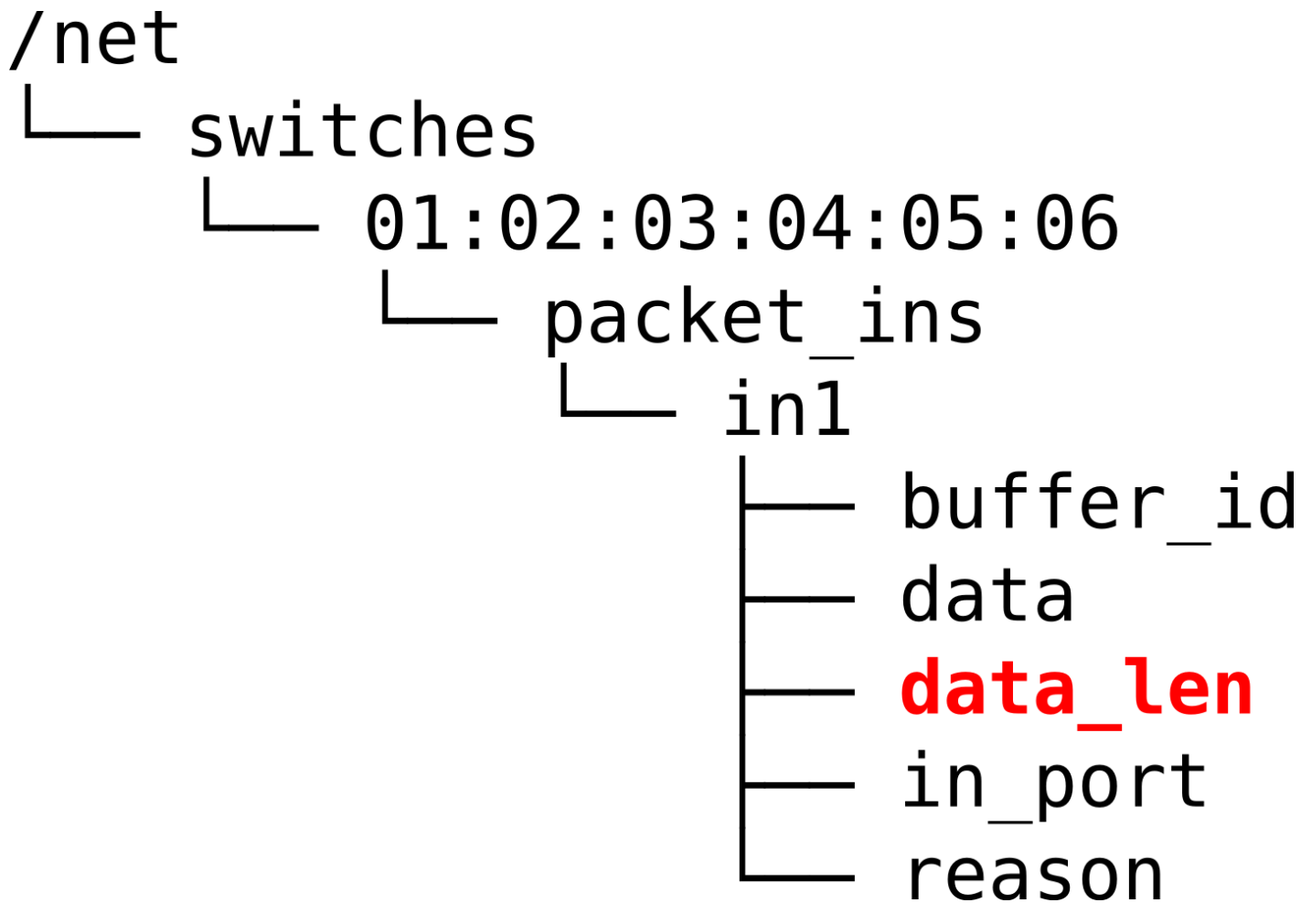
## Flow Entry Directory



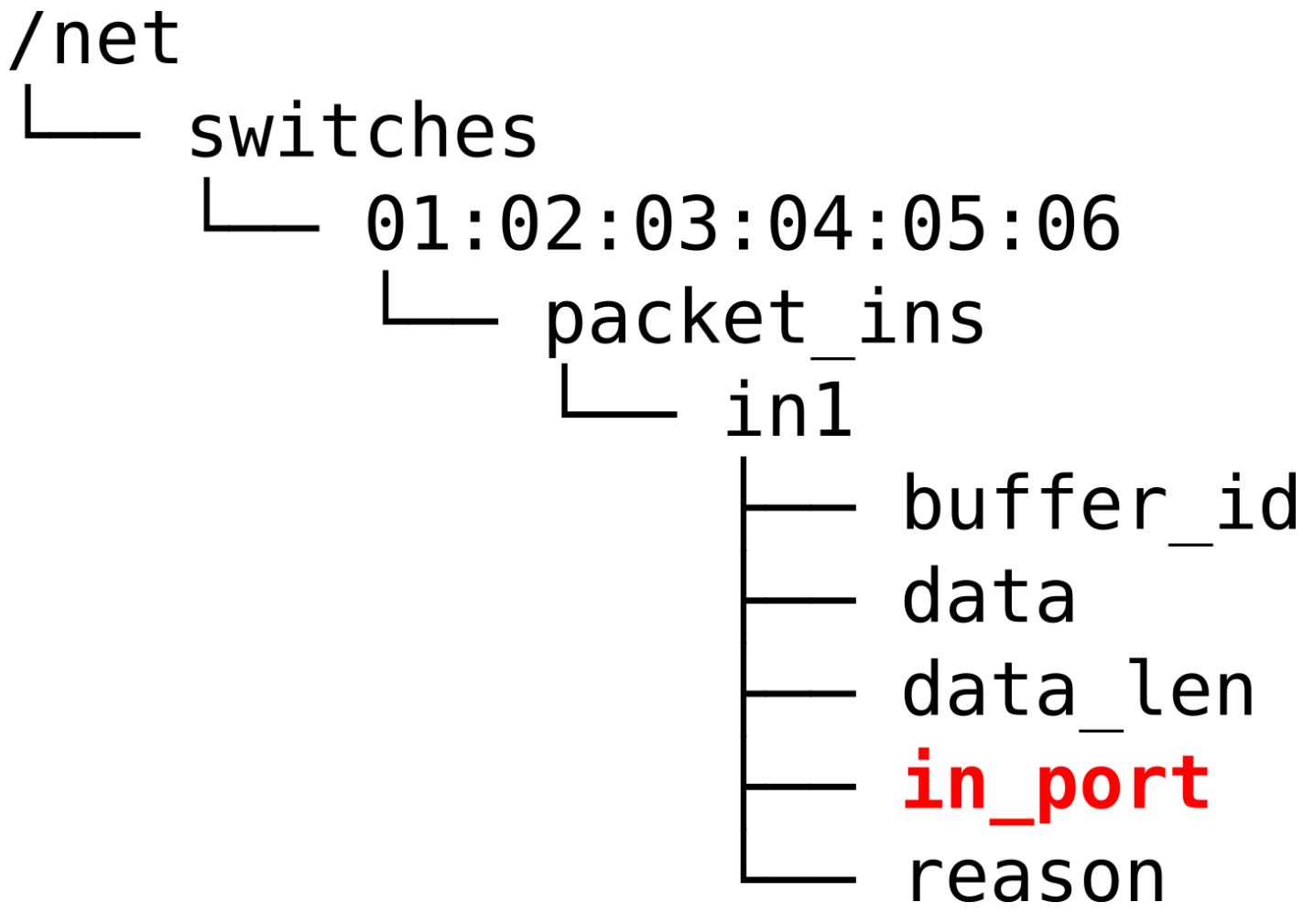
## Packet In Directory



## Packet In Directory



## Packet In Directory



## Packet Out Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        └── packet_outs
            └── out1
                ├── action.out_port
                ├── buffer_id
                ├── data
                └── state
```

## Packet Out Directory

```
/net
└── switches
    └── 01:02:03:04:05:06
        └── packet_outs
            └── out1
                ├── action.out_port
                ├── buffer_id
                ├── data
                └── state
```

# Using the Filesystem

```
$ echo 1 > port_1.port_down
```

```
$ echo 1 > port_1.port_down
```

```
$ cd switches/00:01:02:0a:0b:0c/flows
$ mkdir my_flow_entry
$ ls -1 my_flow_entry
counters
priority
timeout
version
```

```
$ find /net -name tp.dst -exec grep 22 {} +
```

```
$ find /net -name tp.dst -exec grep 22 {} +
```

```
#!/bin/bash
flowdir=/net/switches/"$1"/flows/"$2"
mkdir "$flowdir"
echo ff:ff:ff:ff:ff:ff > "$flowdir"/match.dl_dst
echo 0x0806 > "$flowdir"/match.dl_type
echo FLOOD > "$flowdir"/action.out
```

## Libraries

```
#!/usr/bin/env python3

def new_switch(id, n_tables=1):
    pass

def write_flow(switch, matches=[], actions=[]):
    pass
```

```
#ifndef _YANC_H_
#define _YANC_H_

int new_switch(uint64_t, uint8_t);
int write_flow(const char* path, match_t*, action_t*);

#endif/*_YANC_H_*/
```

- Filesystem - C, FUSE

- Filesystem - C, FUSE
- OpenFlow - C++

- Filesystem - C, FUSE
- OpenFlow - C++
- Discovery - Python

- Filesystem - C, FUSE
- OpenFlow - C++
- Discovery - Python
- Static Flow Pusher - Bash

# Ongoing Work

Composition

Distribution

Applications

## Composition

- Independent development

## Composition

- Independent development
- Event handling

## Composition

- Independent development
- Event handling
- Synthesize configuration

## Composition

- Independent development
- Event handling
- Synthesize configuration
- /usr/share/yanc.d/
- /etc/yanc.d/

## Composition

- Independent development
- Event handling
- Synthesize configuration
- /usr/share/yanc.d/
- /etc/yanc.d/
- Ship with configuration

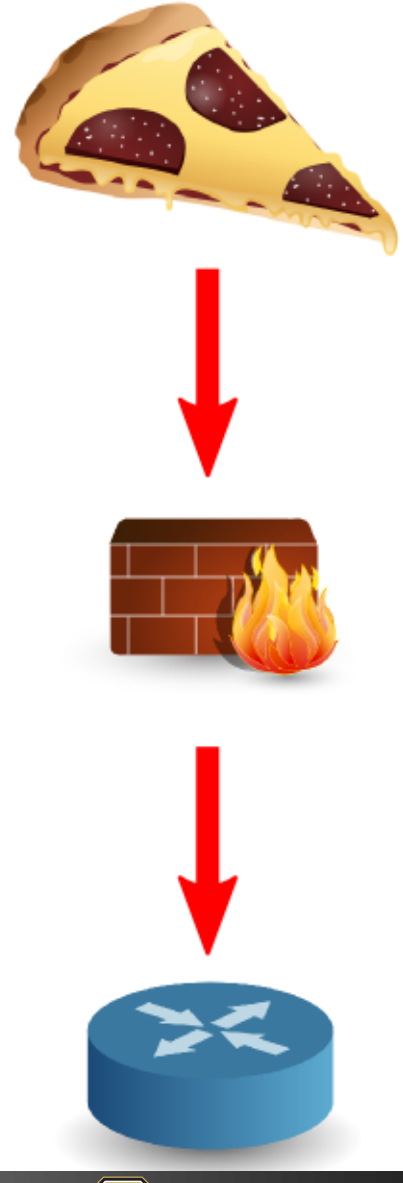
## Composition

- Independent development
- Event handling
- Synthesize configuration
- /usr/share/yanc.d/
- /etc/yanc.d/
- Ship with configuration
- Administrator controlled

## Composition

```
# /etc/yanc.d/

<event> slice
<event> firewall
<event> router
```



## Distribution

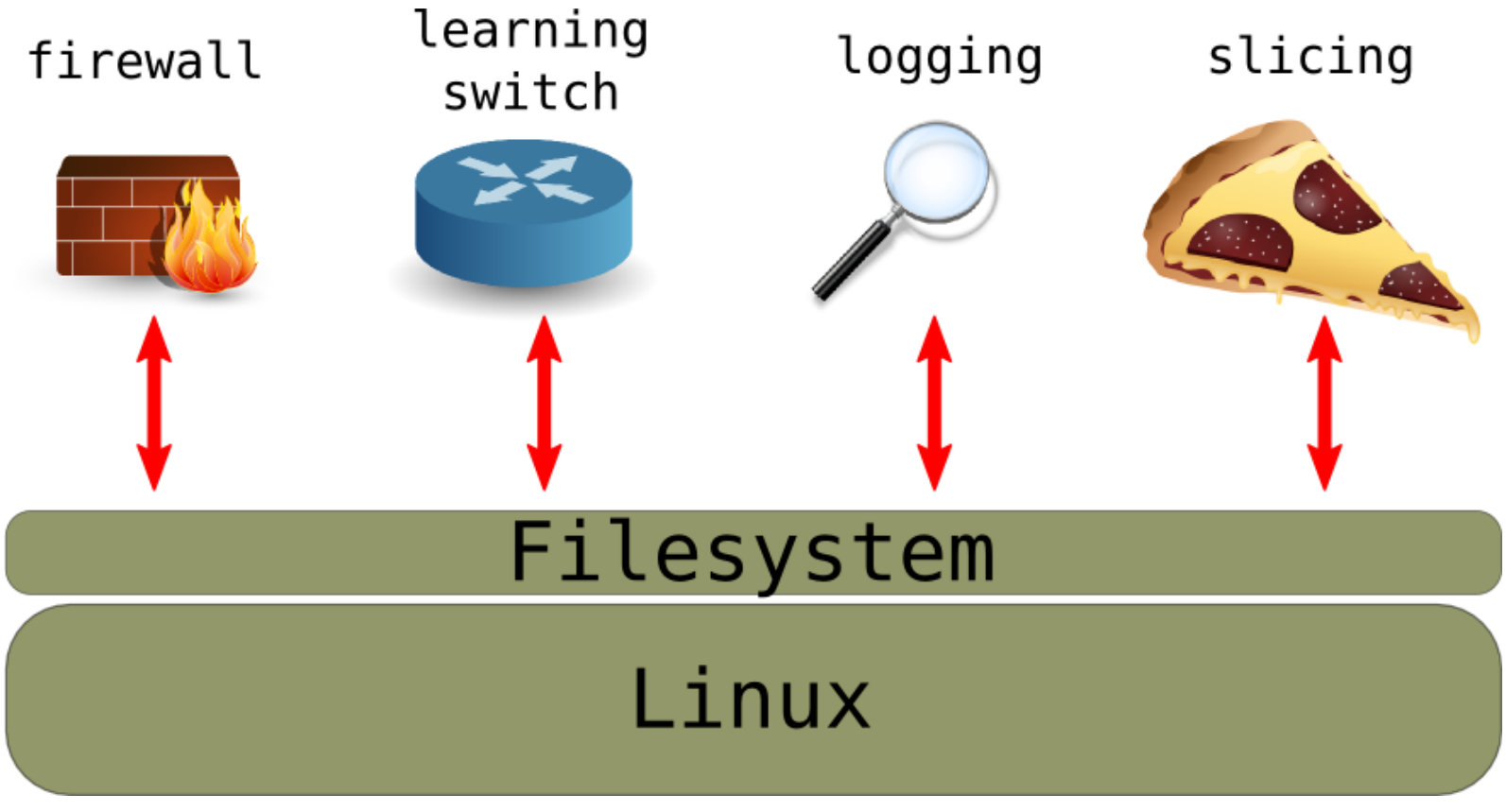
Latency

Fault Tolerance

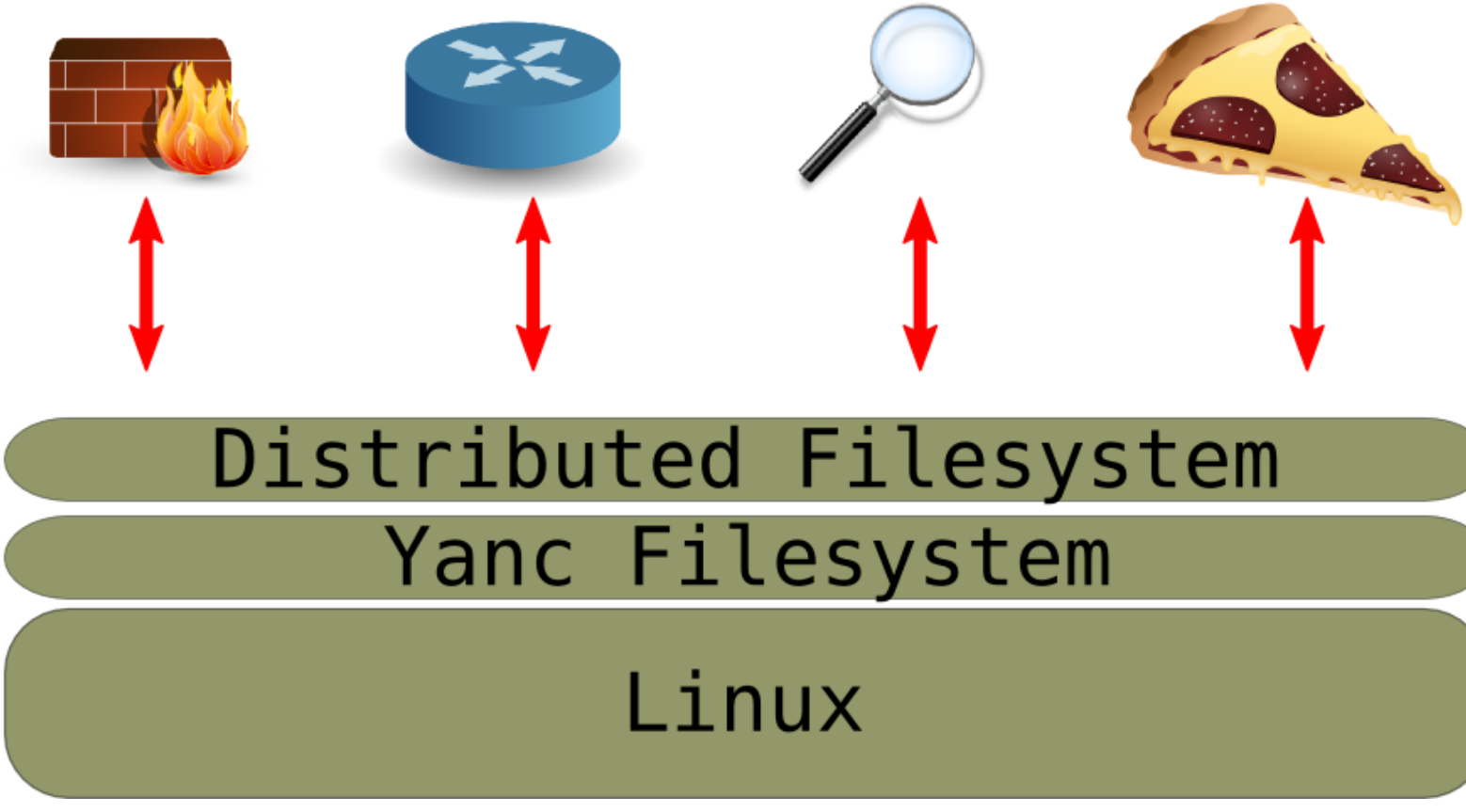
Administration

Different Requirements

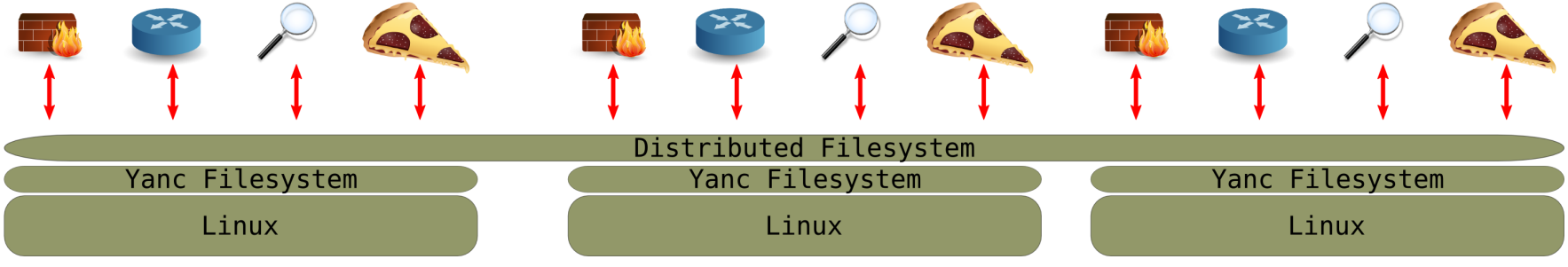
## Distribution



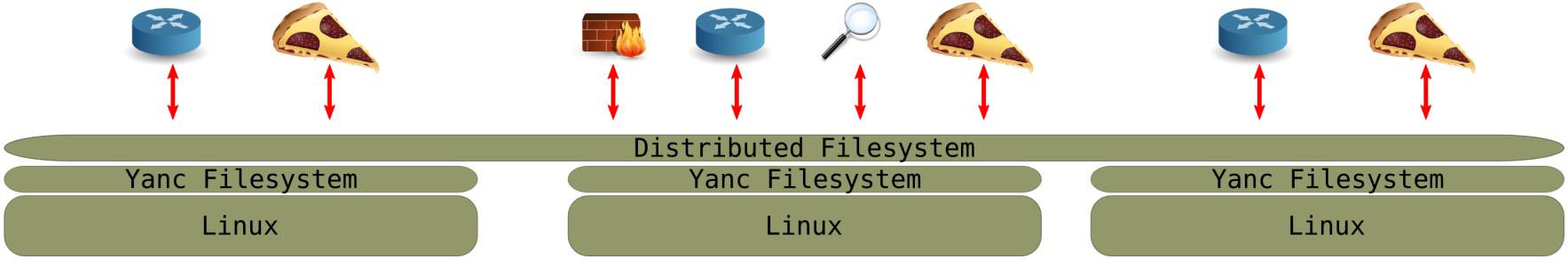
## Distribution

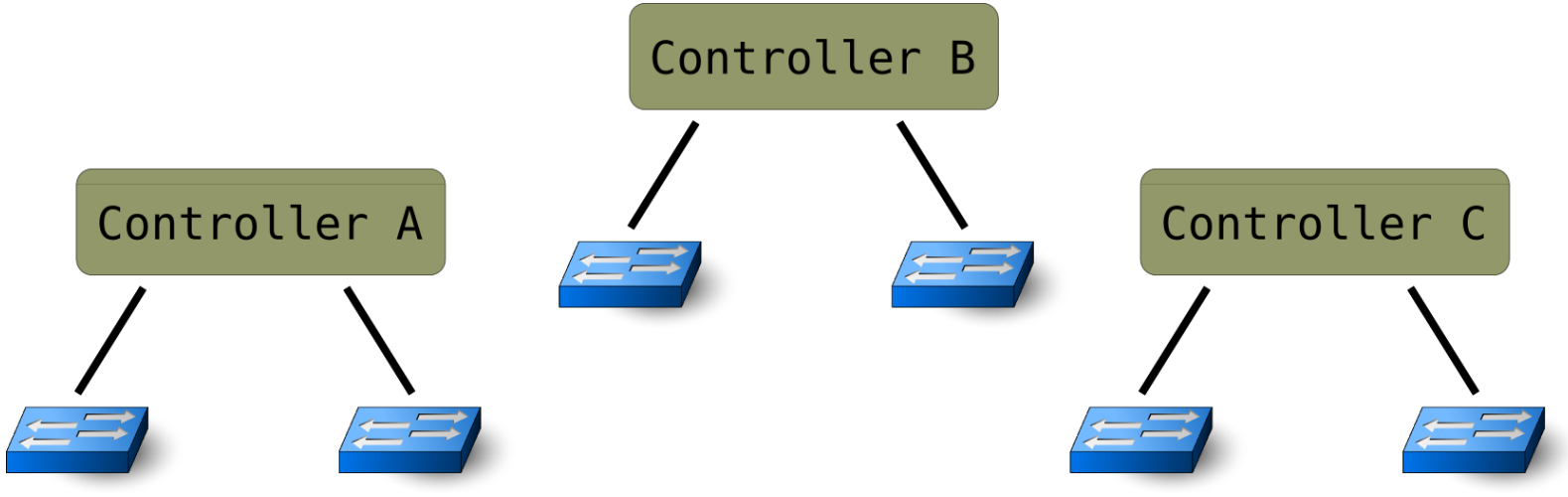


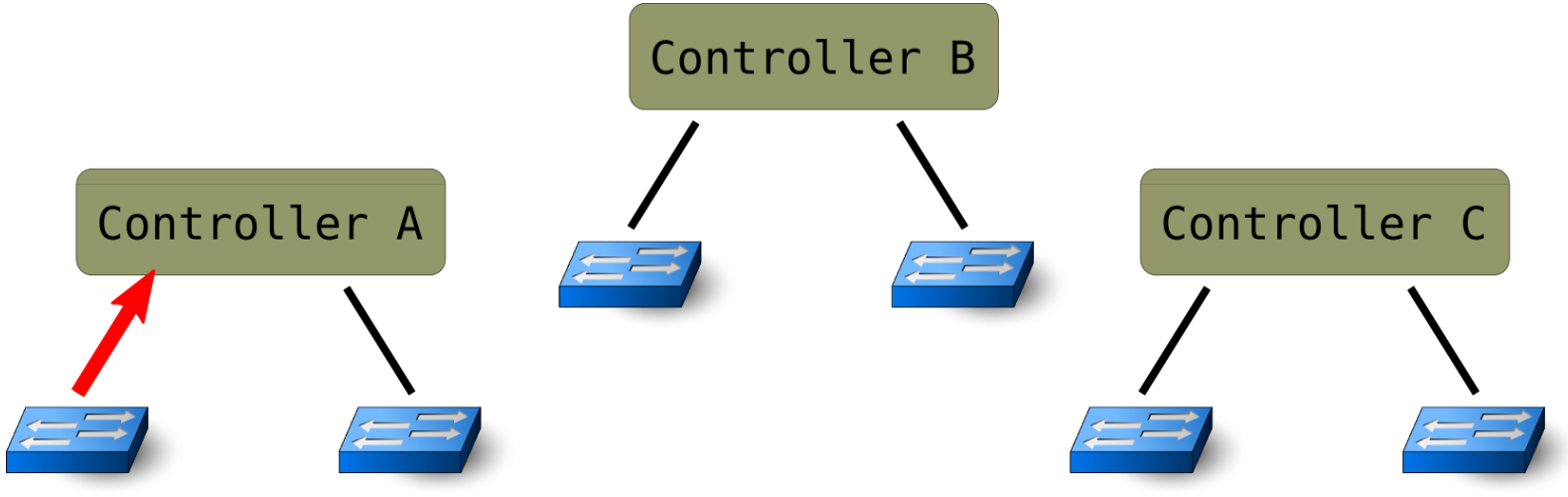
# Distribution

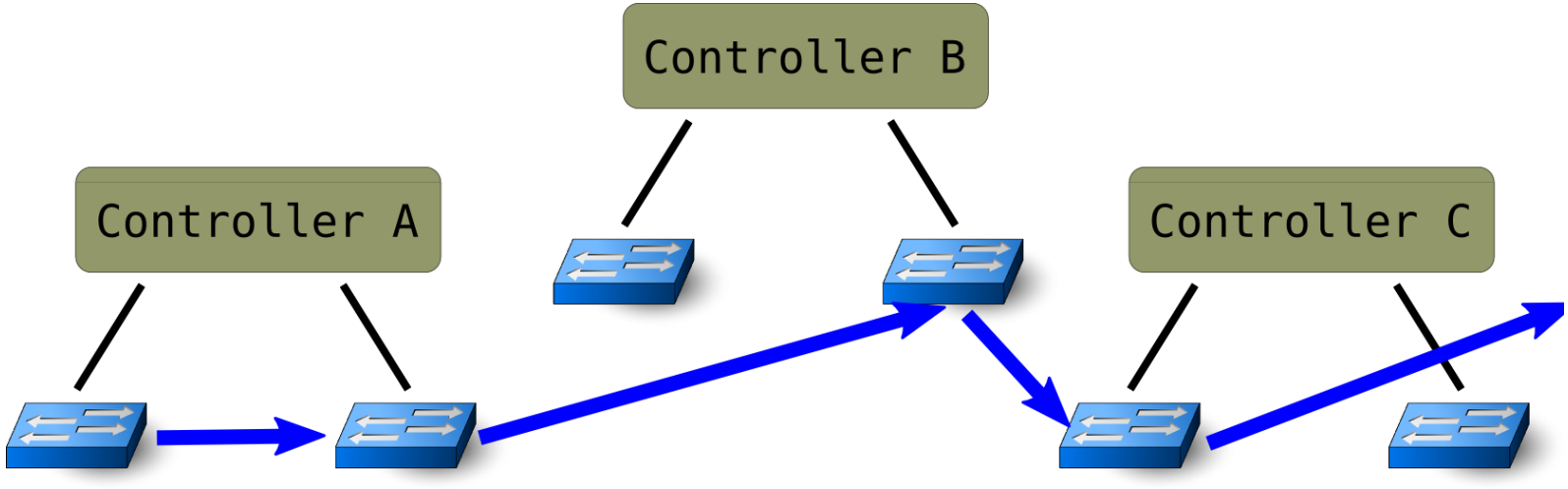


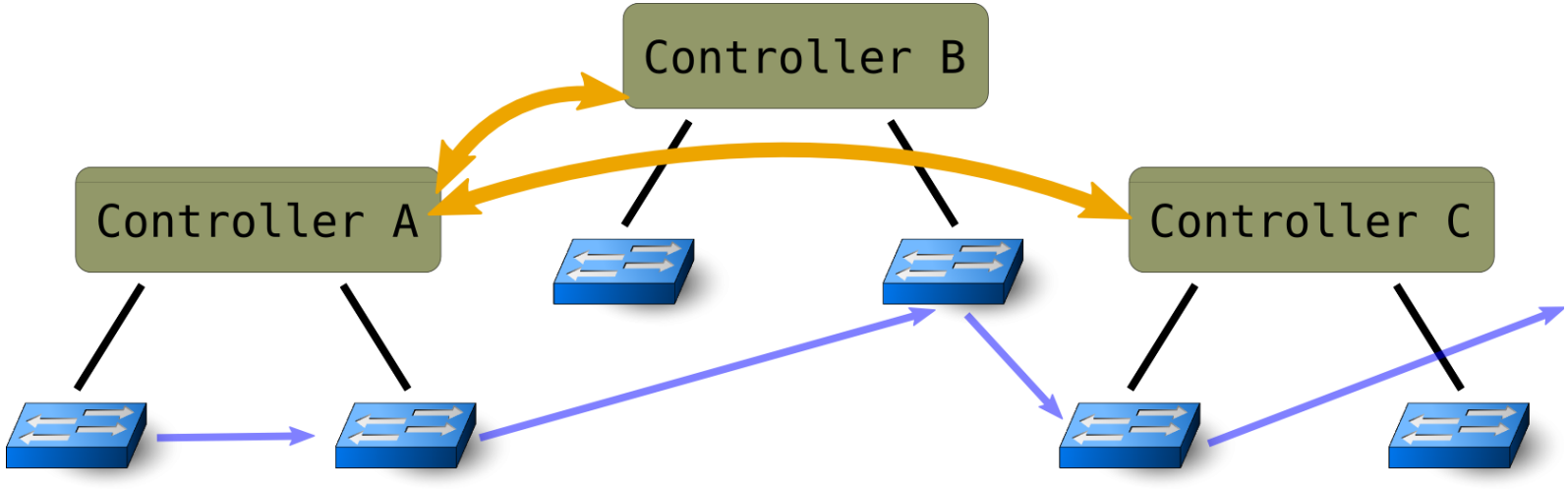
# Distribution

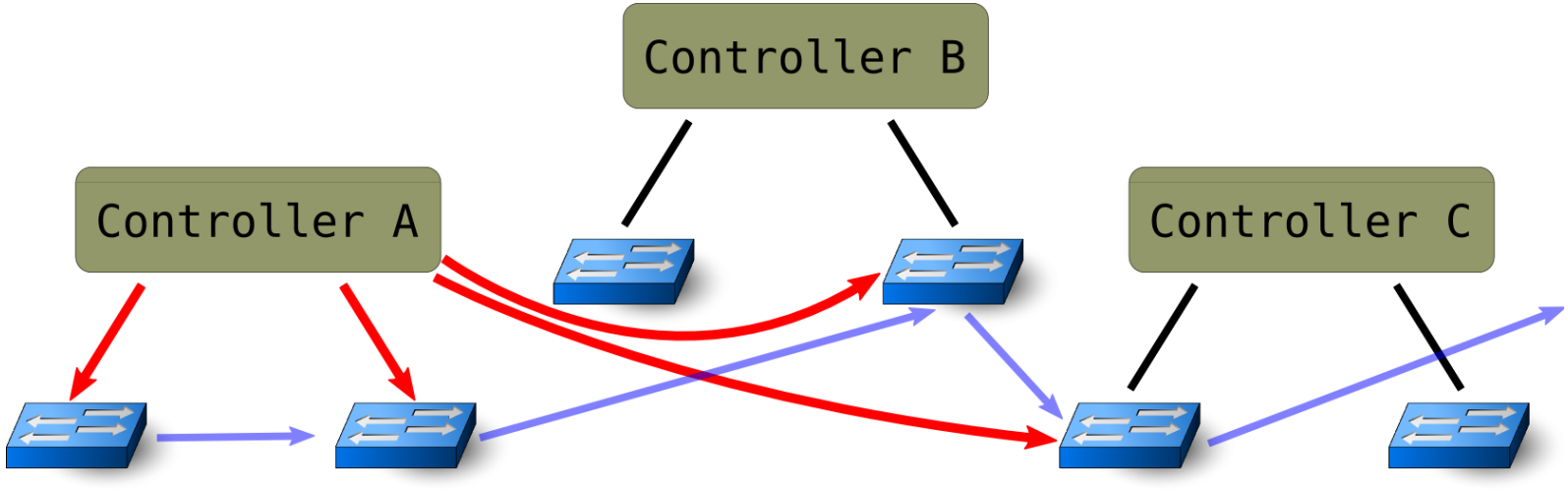


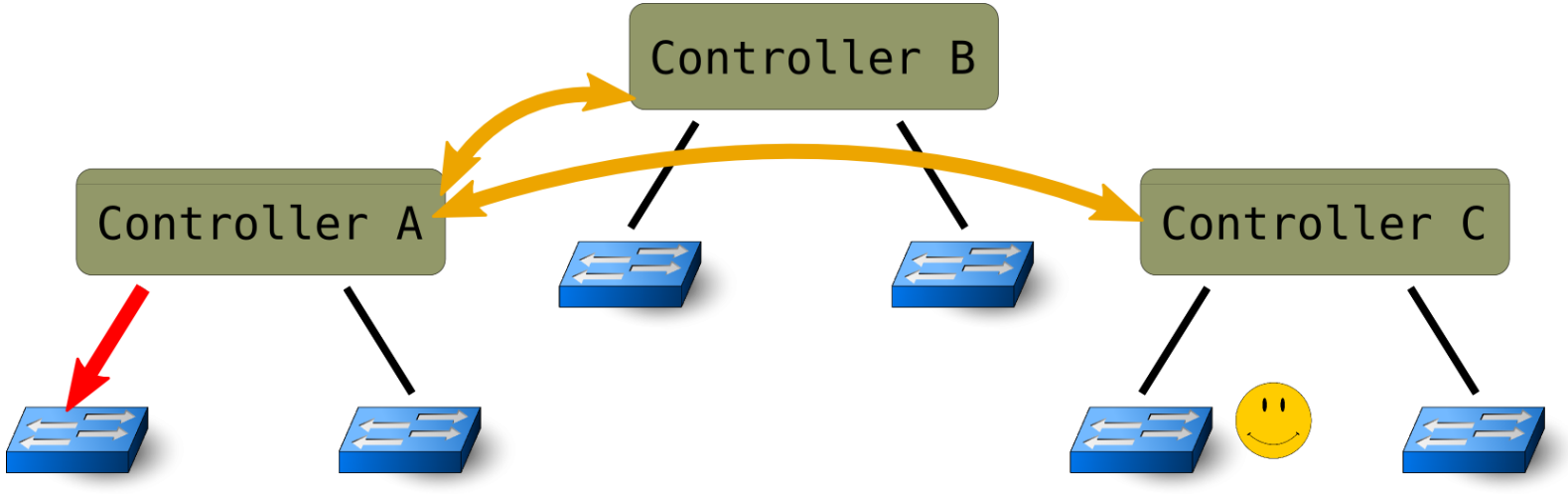

Controller B
Controller A
Controller C


Controller B
Controller A
Controller C


Controller B
Controller A
Controller C


Controller B
Controller A
Controller C


Controller B
Controller A
Controller C


Controller B
Controller A
Controller C

# Defined a Universal Interface for Network Control

Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes

Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes

Implemented Interface

Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes

Implemented Interface

Leveraging Existing OS Technologies

Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes

Implemented Interface

Leveraging Existing OS Technologies

Built Functional Applications on Top of Interface

Thank You!