Applying Operating System Principles to SDN Controller Design

Oliver Michel, Matthew Monaco, Eric Keller

University of Colorado Boulder

Invited Talk University of Illinois at Urbana-Champaign
April 11th, 2014.

"What we clearly need is an "operating system" for networks, one that provides a uniform and centralized programmatic interface to the entire network."

[Gude et.al.'08]

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

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Performance

Keywords

Architecture, Management, Network, Security

1 Introduction

As anyone who has operated a large network can attest, enterprise networks are difficult to manage. That they have

interface to the entire network.[1] Analogous to the read and write access to various resources provided by computer operating systems, a network operating system provides the ability to *observe* and *control* a network.

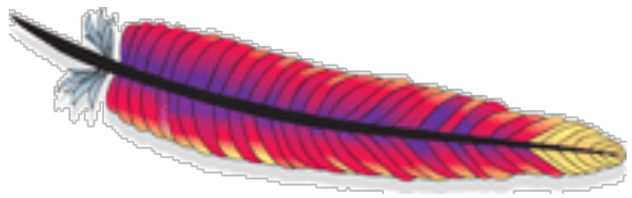
A network operating system does not manage the network itself; it merely provides a programmatic interface. *Applications* implemented on top of the network operating system perform the actual management tasks.[2] The programmatic interface should be general enough to support a broad spectrum of network management applications.

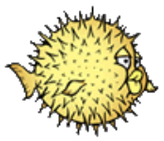
Such a network operating system represents two major conceptual departures from the status quo. First, the network operating system presents programs with a *centralized* programming model[3]; programs are written as if the entire network were present on a single machine (*i.e.*, one would use Dijkstra to compute shortest paths, not Bellman-Ford). This

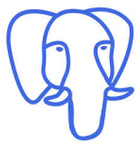
Extend an existing operating system and its user space software ecosystem in order to serve as a practical network operating system

Distributions

Software Projects







```
$
```

```
$
```

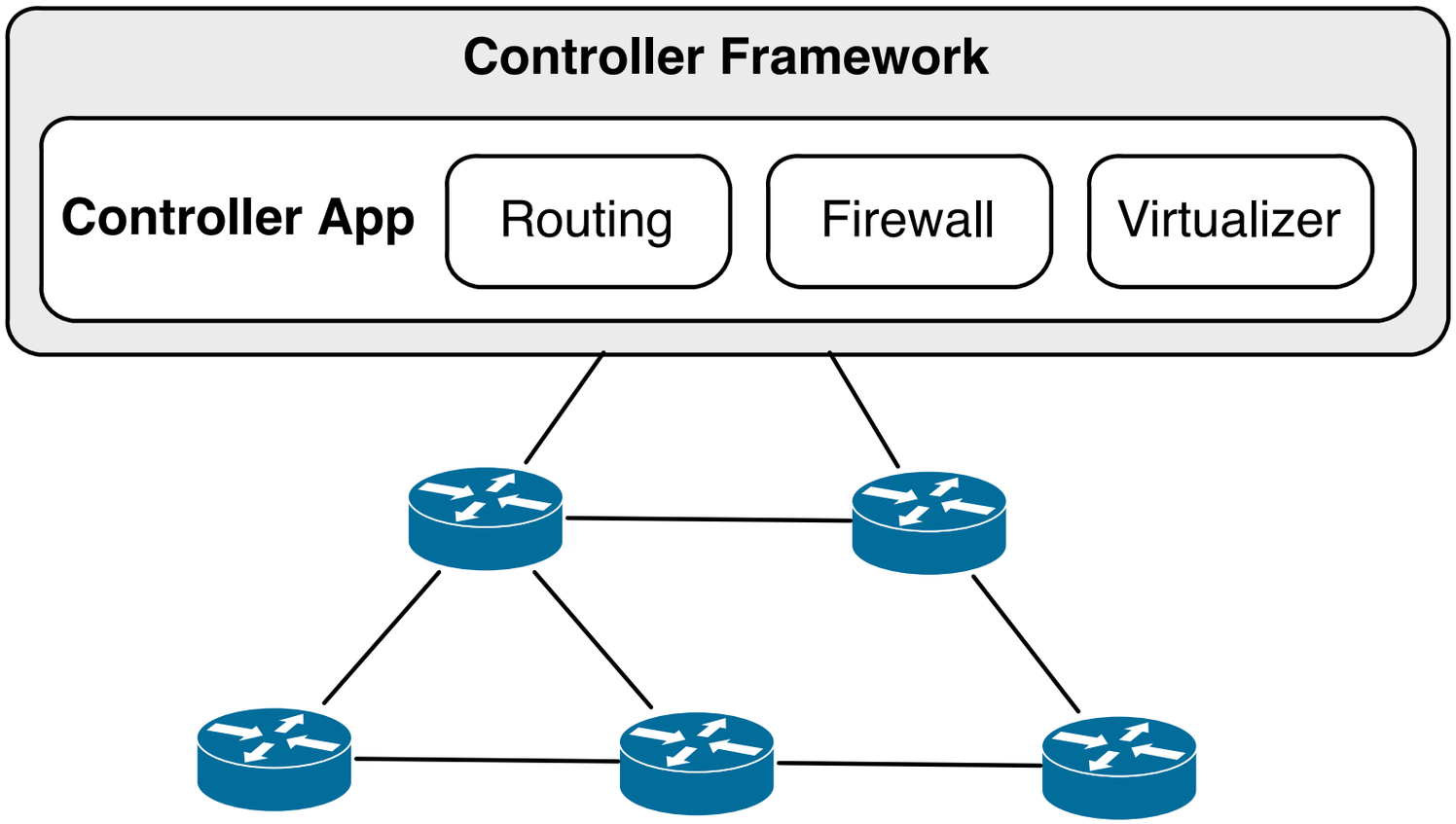
```
$ grep
$ find
$ sed
$ sort
$ cat
```

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

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

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

```
#!/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
```


Controller Framework
Controller App
Routing
Firewall
Virtualizer


nox


nox


pox


nox


pox


nox


pox


OPEN
DAYLIGHT


nox


pox


OPEN
DAYLIGHT


Project
Floodlight


nox


pox


OPEN
DAYLIGHT


Project
Floodlight


Trema

C++

C++

Python

C++

Python

Python

Trema

C++

Python

Python

Java

Trema

C++

Python

Python

Java

Java

Trema

C++

Python

Python

Java

Java

Trema Ruby/C

yanc
yet another network controller

yanc
yet another network controller

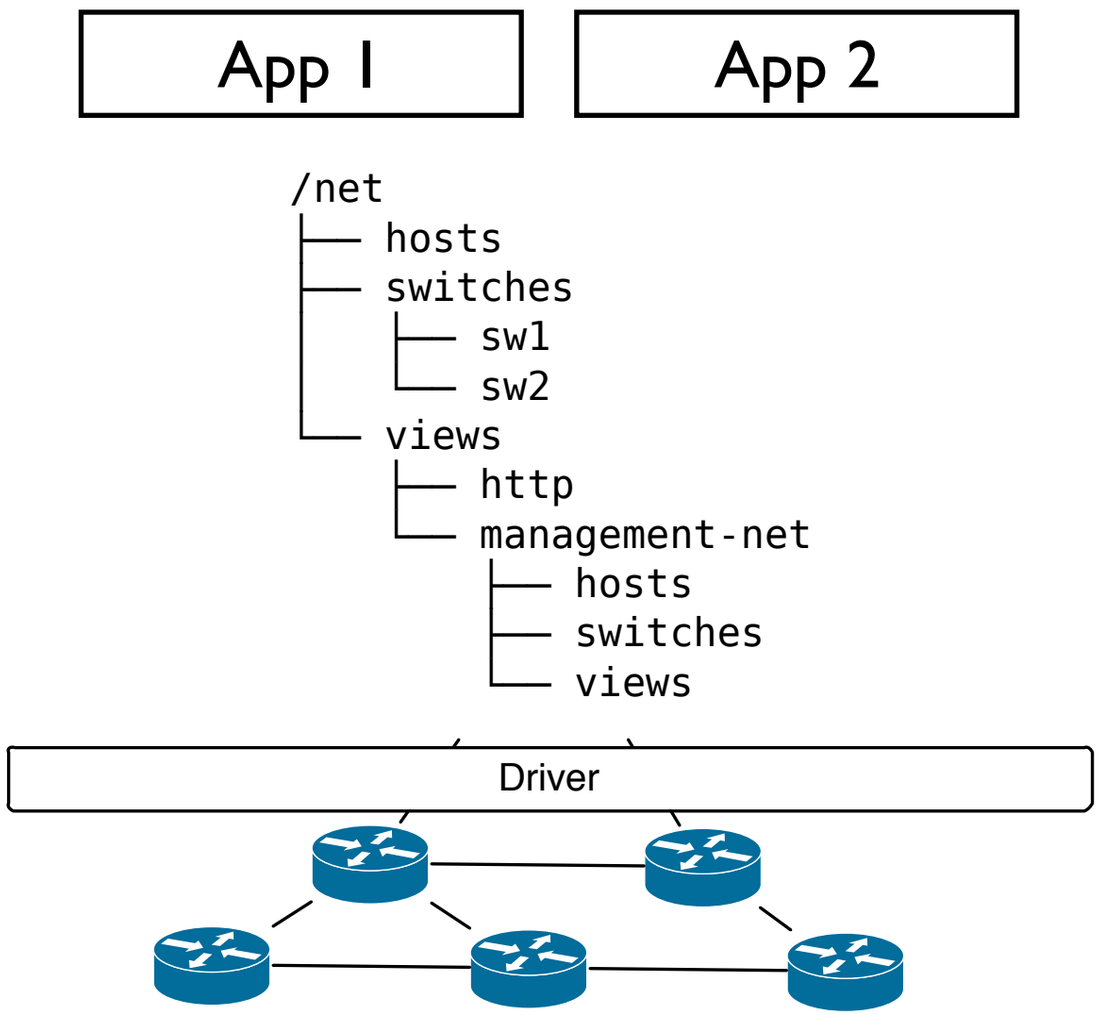
```
/net
├── hosts
├── switches
│   ├── sw1
│   └── sw2
└── views
    ├── http
    └── management-net
        ├── hosts
        ├── switches
        └── views
```

yanc
yet another network controller

App 1 | App 2

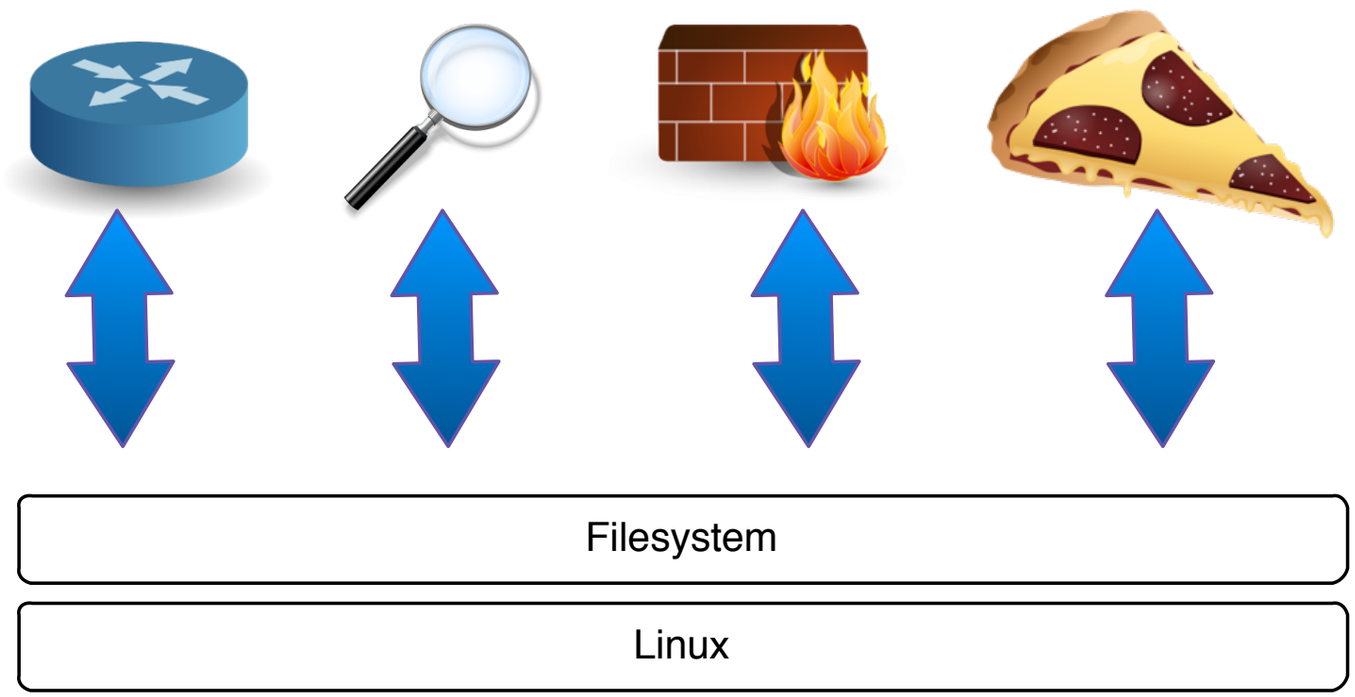
```
/net
├── hosts
├── switches
│   ├── sw1
│   └── sw2
└── views
    ├── http
    └── management-net
        ├── hosts
        ├── switches
        └── views
```

yanc
yet another network controller



Why a filesystem?

Logically Distinct Applications

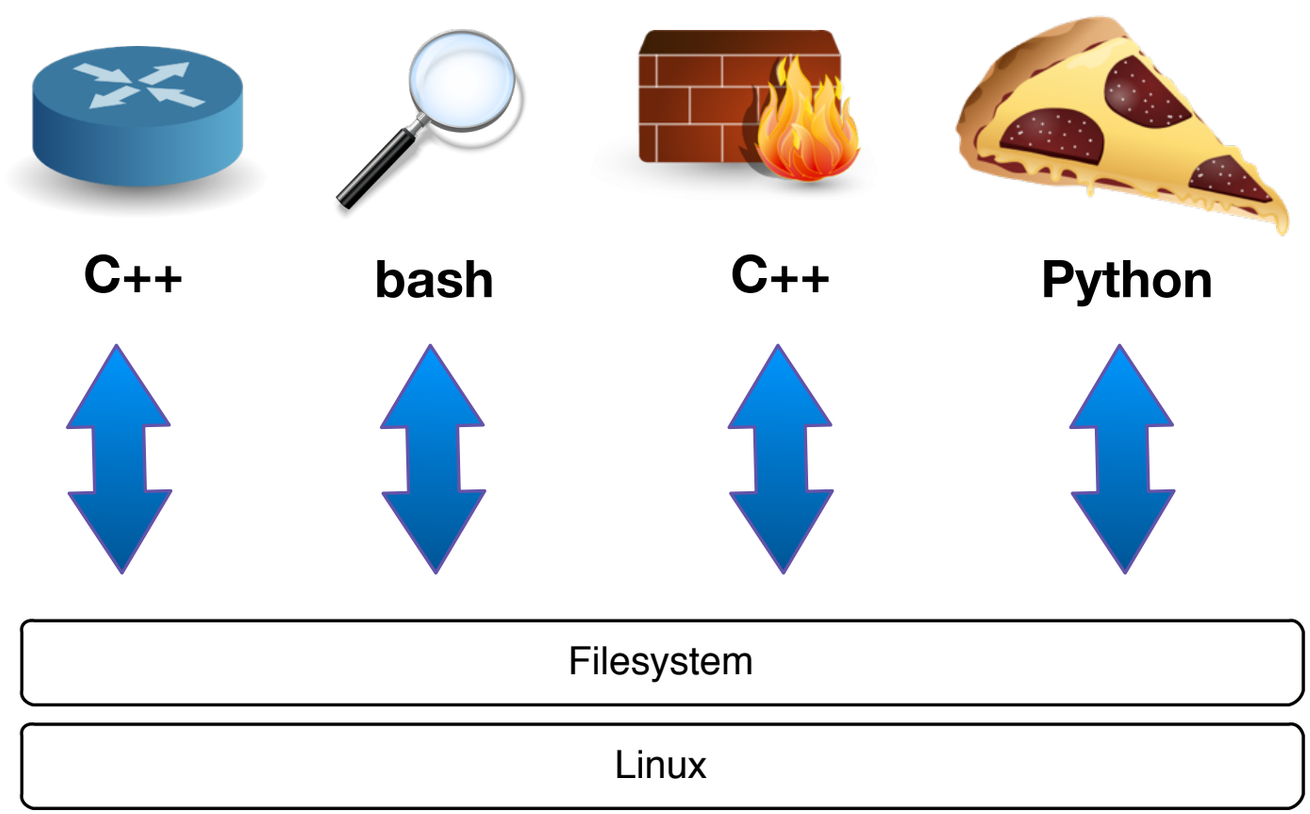


Independent Development

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

```
$ git clone git@github.com/yanc/yanc-fw
$ cd yanc-fw
$ make
# make install
```

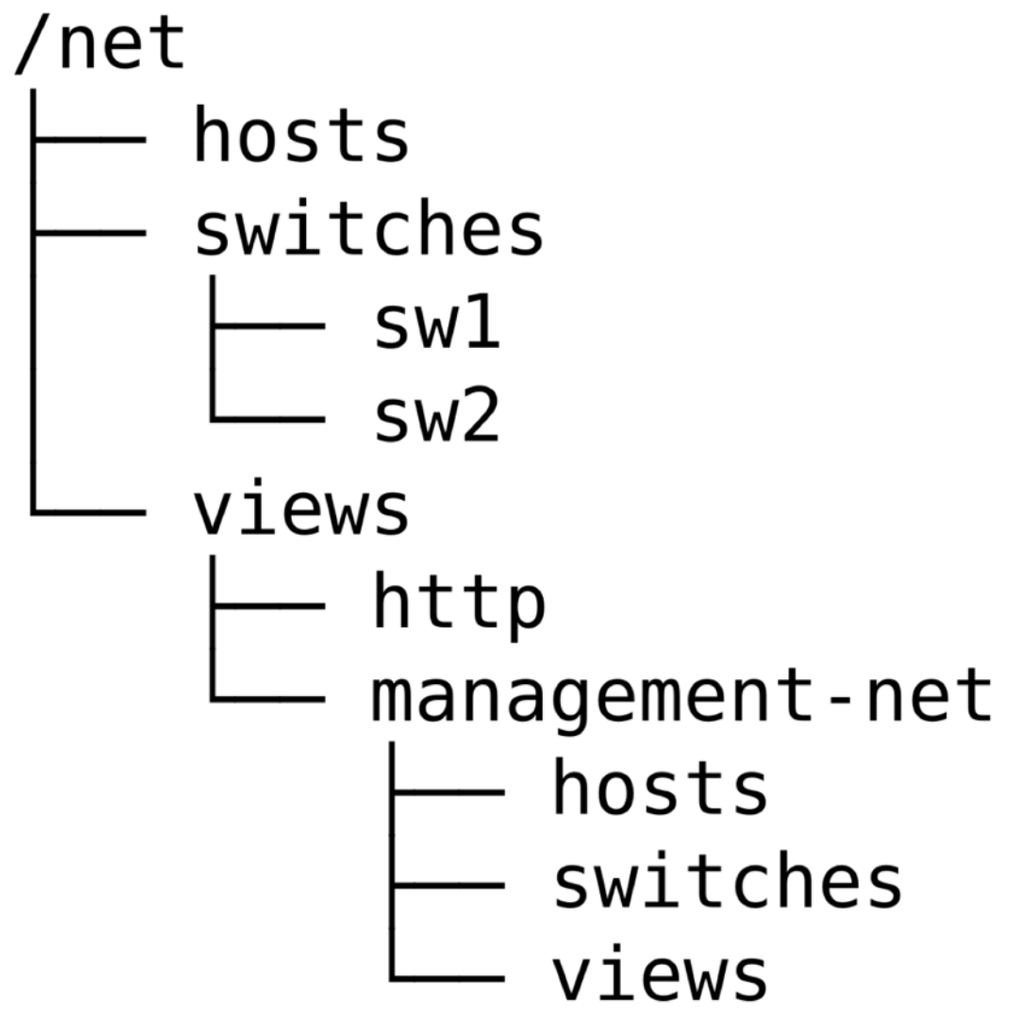
Any Programming Language



Yanc Components

- The Yanc Filesystem
- Hardware Decoupling
- Operating System Integration
- Dynamic Application Composition
- Distribution

The Yanc Filesystem



The Yanc Filesystem

```
01:02:03:04:05:06/
├── flows/
├── packet_in/
├── packet_out/
├── ports/
├── actions
├── capabilities
├── datapath_id
├── flags
├── miss_send_len
├── n_buffers
└── n_tables
```

The Yanc Filesystem

```
port_01/
├── config.flood
├── config.fwd
├── config.packet_in
├── config.port_down
├── config.recv
├── config.recv_stp
├── config.stp
├── hw_addr
├── peer -> /dev/null
├── port_no
├── state.link_up
├── state.stp_forward
├── state.stp_learn
├── state.stp_listen
├── stats.collisions
├── stats.rx_bytes
├── stats.rx_crc_err
├── stats.rx_dropped
├── stats.rx_errors
├── stats.rx_frame_err
├── stats.rx_over_err
├── stats.rx_packets
├── stats.tx_bytes
├── stats.tx_dropped
├── stats.tx_errors
└── stats.tx_packets
```

The Yanc Filesystem

```
in
├── buffer_id
├── data
├── data_len
├── in_port
└── reason
```

```
out
├── action.out_port
├── buffer_id
├── data
├── in_port
└── state
```

The Yanc Filesystem

```
arp_flow
├── counters/
├── match.dl_type
├── match.dl_src
├── action.out
├── priority
├── timeout
└── version
```

The Yanc Filesystem

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

```
$ mkdir -p switches/00:01:02:0a:0b:0c/flows/my_flow
```

The Yanc Filesystem

```
$ find /net -name tp_dst -exec grep 22 {} +
```

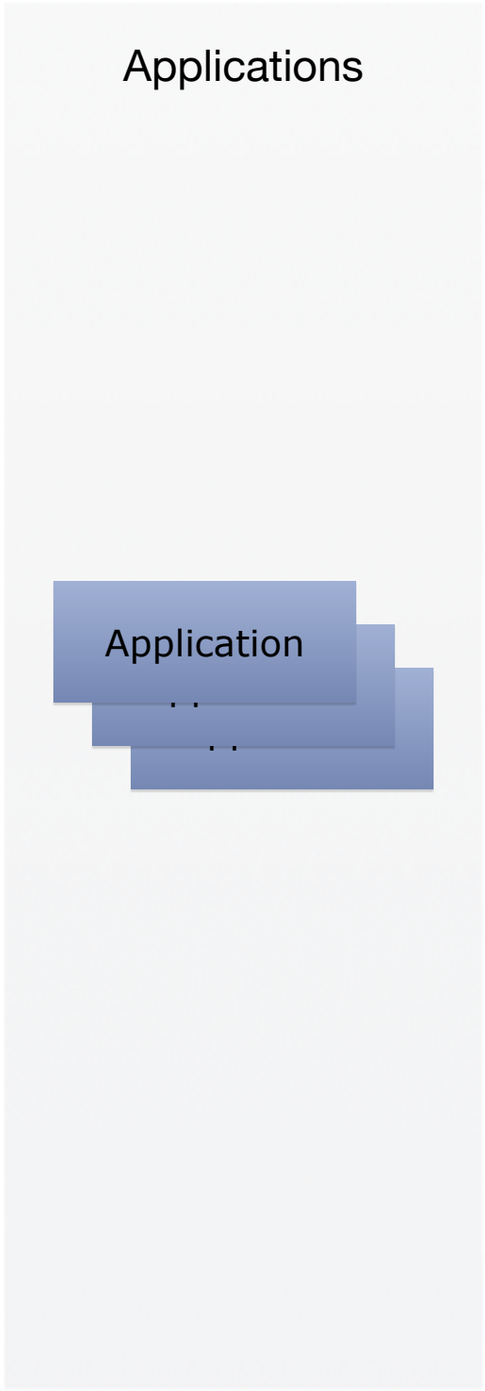
```
#!/bin/bash
flowdir=/net/switches/"$1"/flows/"$2"
mkdir "$flowdir"
echo ff:ff:ff:ff:ff:ff > "$flowdir"/match.dl_dst
echo 0x0806 > "$flowdir"/match.dl_type
echo FLOOD > "$flowdir"/action.out
```

Hardware Decoupling and Middleboxes

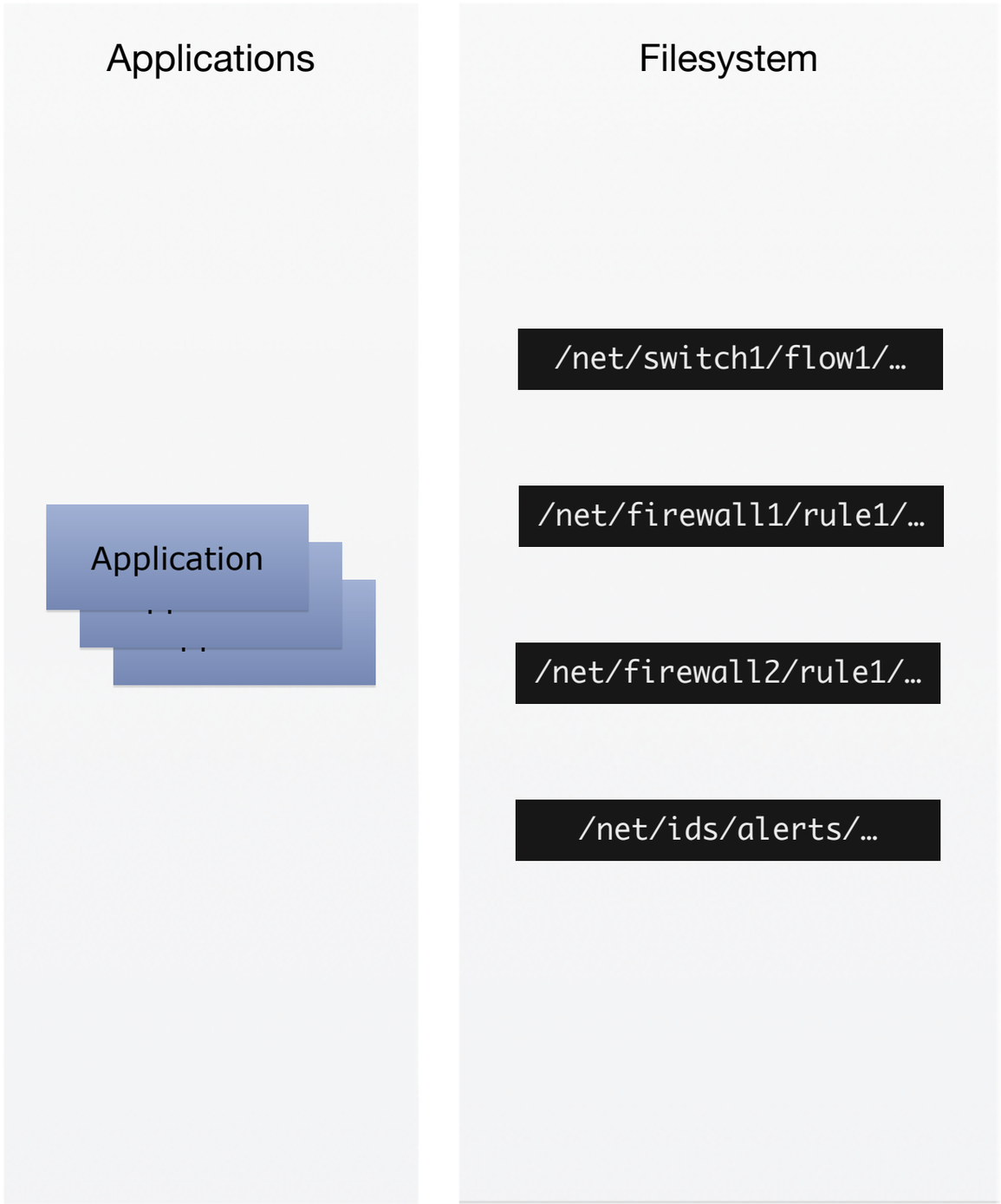
- Support for different physical devices and device classes
  - Device Drivers
  - Schema-based filesystem

Device Drivers

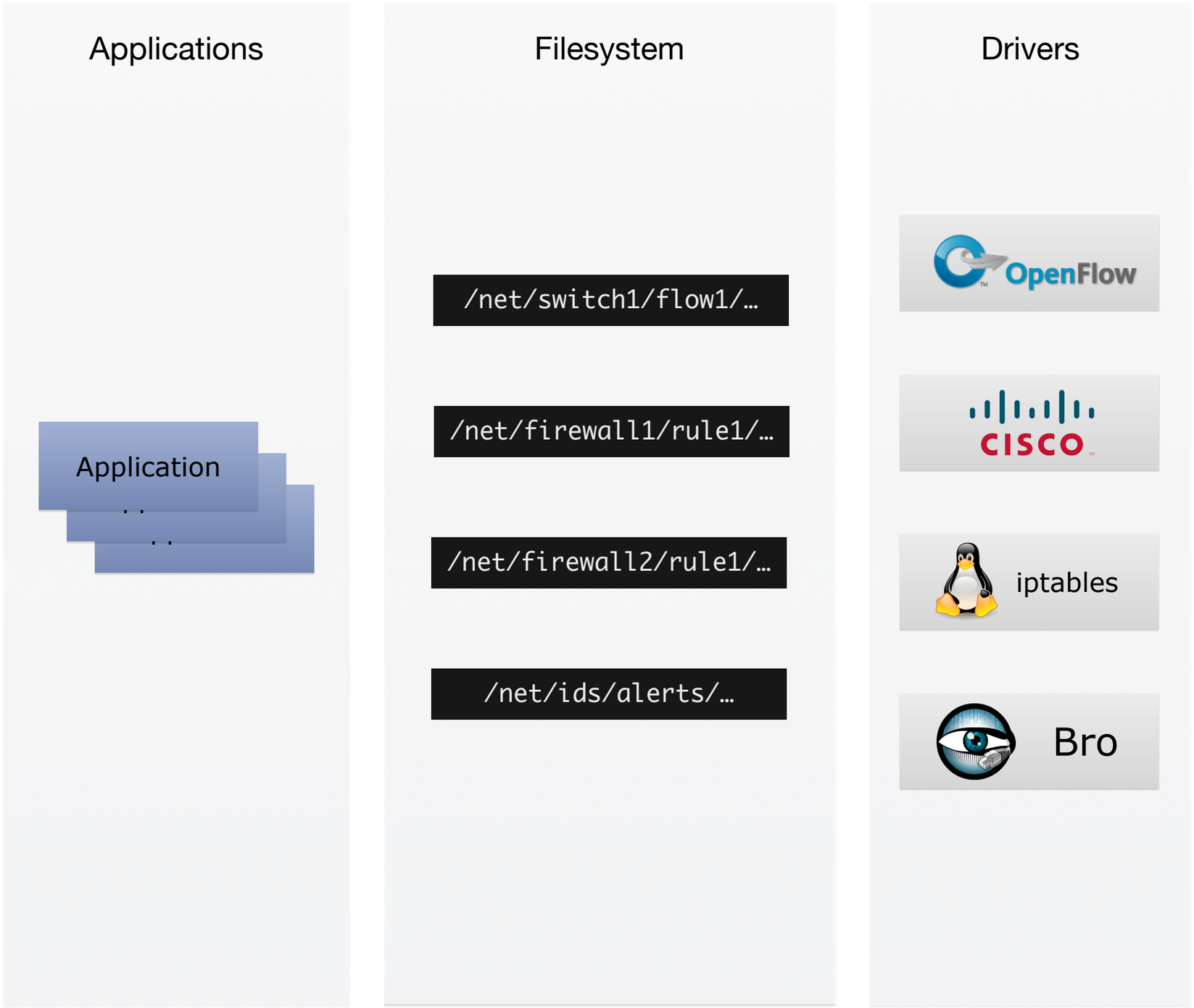
Device Drivers



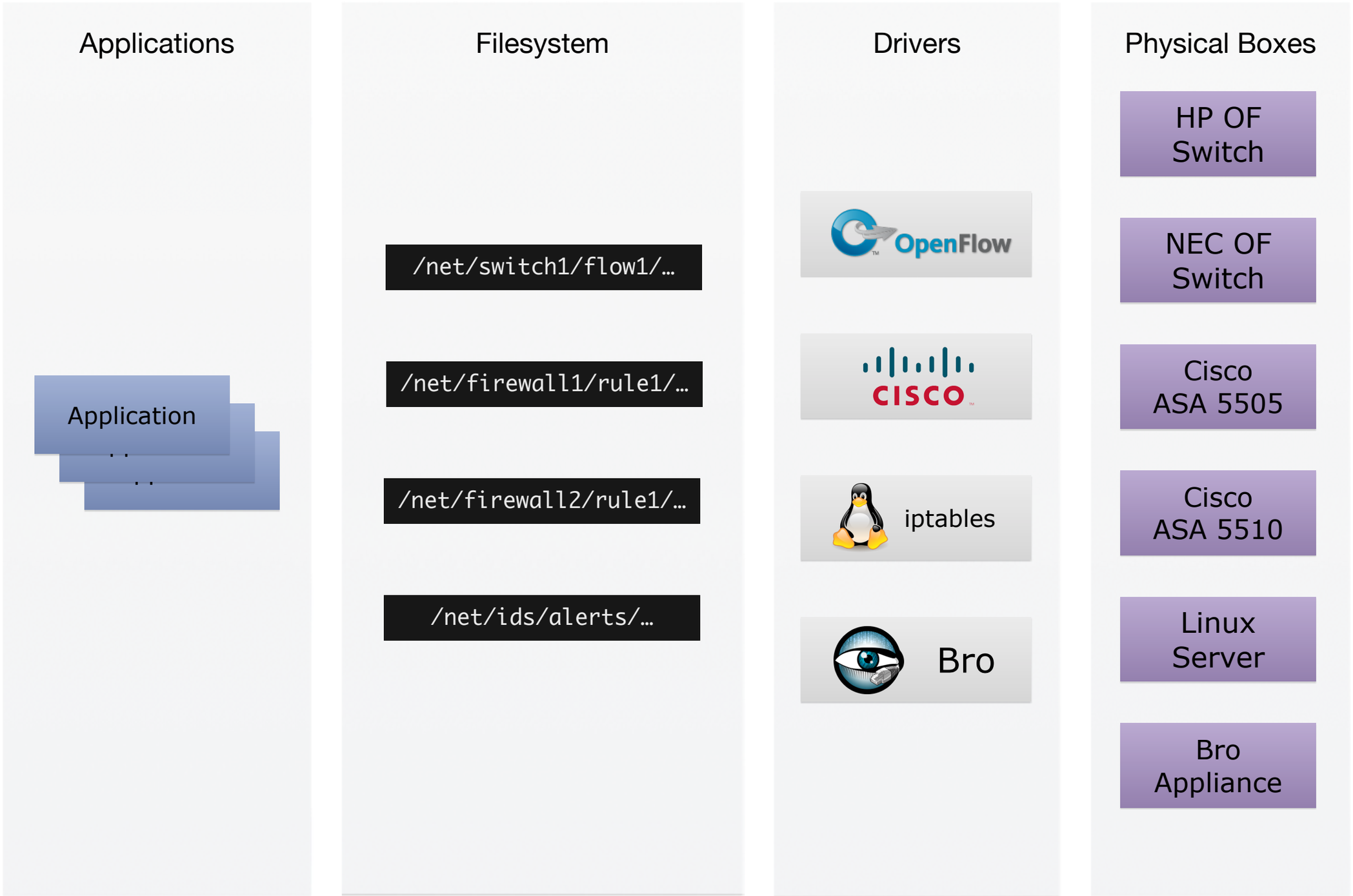
Device Drivers



Device Drivers



Device Drivers



Schema-based Filesystem

```
nodes {
  switch-* {
    u64 id;
    ports/* {
      ln peer;
    }
  }

  host-* {
    eth mac_addr;
    str hostname;
    ln  port-*;
  }
}
```

```
/net
└── nodes
    ├── host-1
    │   ├── port-1 -> ../switch-1/ports/port-1
    │   ├── hostname
    │   └── mac_addr
    └── switch-1
        ├── ports
        │   └── port-1
        │       └── peer -> ../../../host-1
        └── id
```

Operating System Integration

- Integrate tightly with Linux
- Use off-the-shelf technologies
- Encourage active ecosystem

Operating System Integration

What can we use from a traditional operating system?

Operating System Integration

What can we use from a traditional operating system?

Operating System Integration

What can we use from a traditional operating system?

- Inotify
- File Permissions and ACLs
- Namespaces and Control Groups
- Layered Filesystem

Application Composition

- Today's controllers show mostly monolithic design
- Distinct applications must be composed

Composition in Yanc

- Frenetic/Pyretic introduced Parallel/Sequential Composition
  - per packet abstraction
  - compiled into single application
- Yanc controls read/writes to the network state
- Yanc provides dynamic composition

Motivating Example

Motivating Example

Fast Router

Motivating Example

Fast Router

Route Optimizer

Motivating Example

Fast Router

Route Optimizer

Link Tapper

Motivating Example

Fast Router

Route Optimizer

Link Tapper

Motivating Example

Fast Router

Route Optimizer

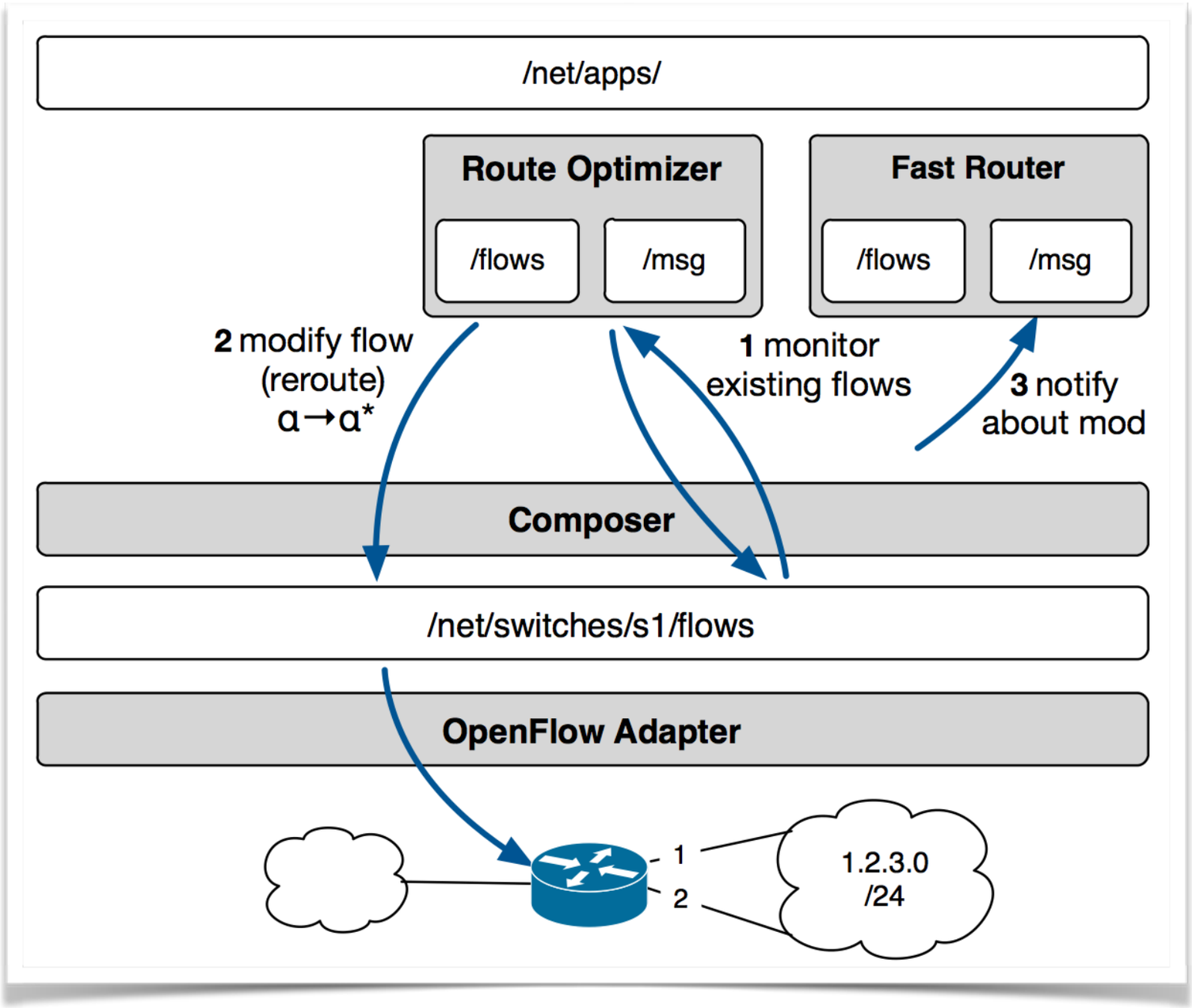
Link Tapper

Motivating Example

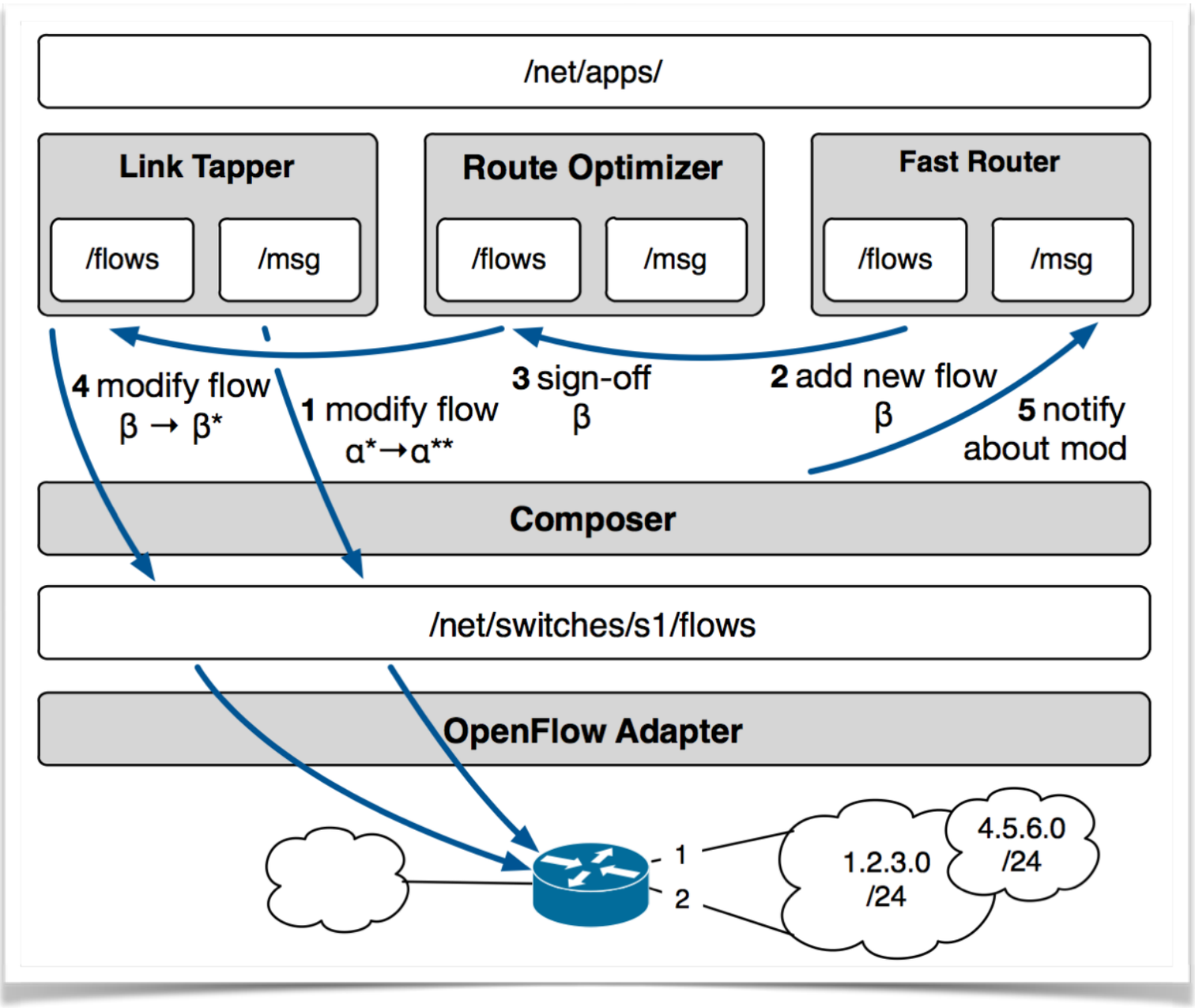
Motivating Example

Motivating Example

Optimizing a flow



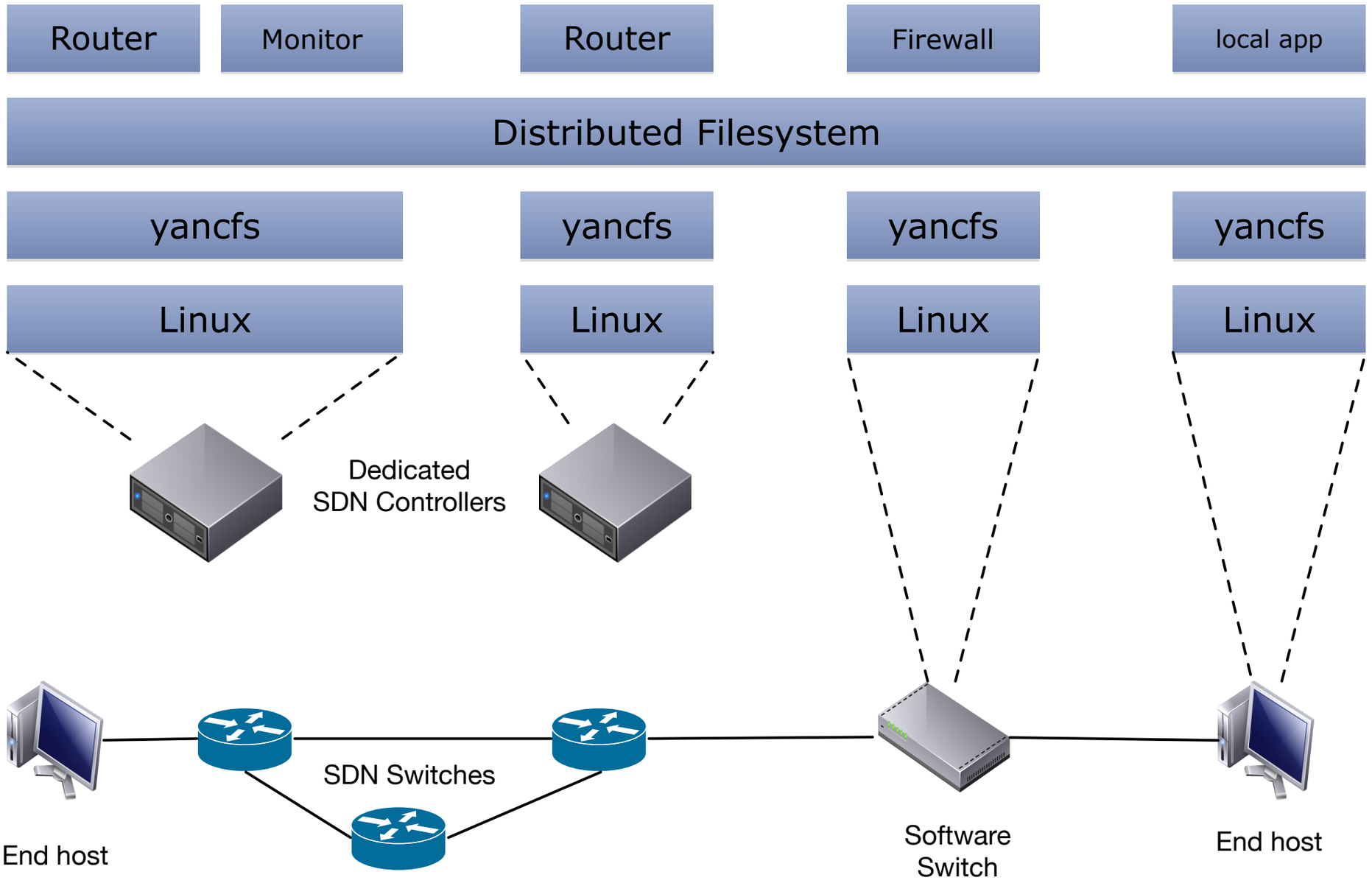
Adding a Linktapper



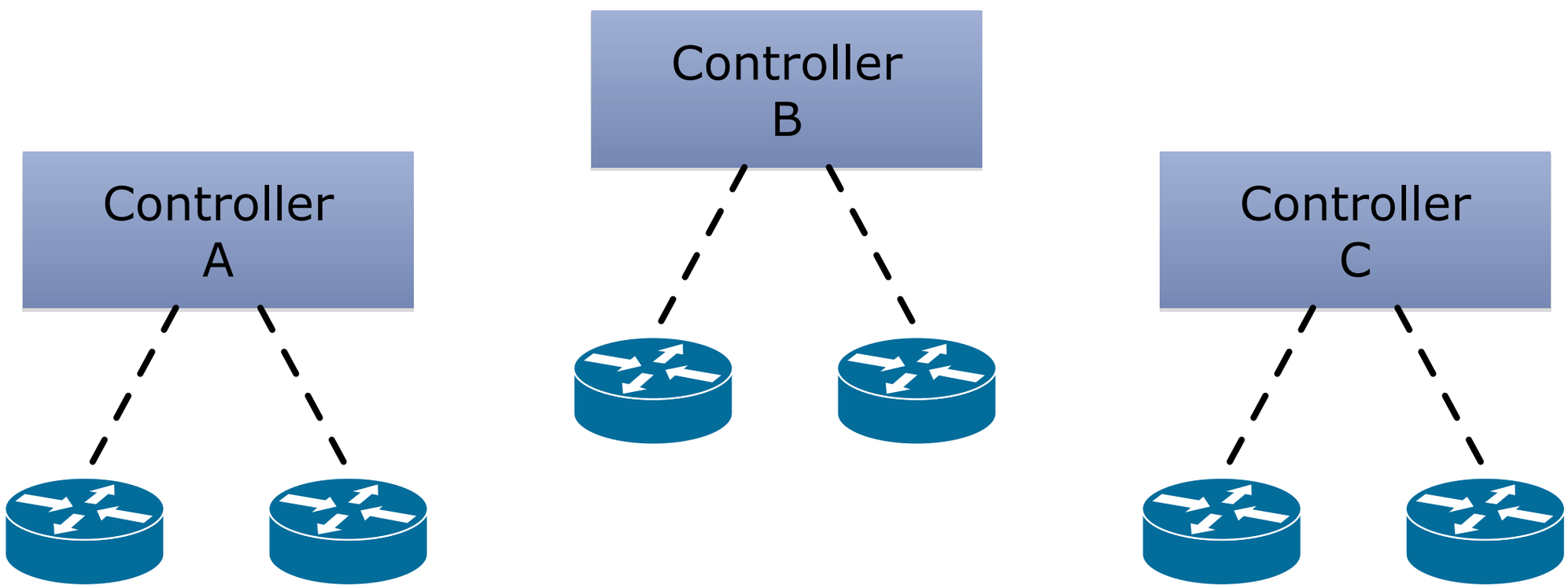
Distribution

- Filesystem representation allows layering
  - Distributed filesystem on top of yancfs
- Yanc may run on multiple servers or even switches

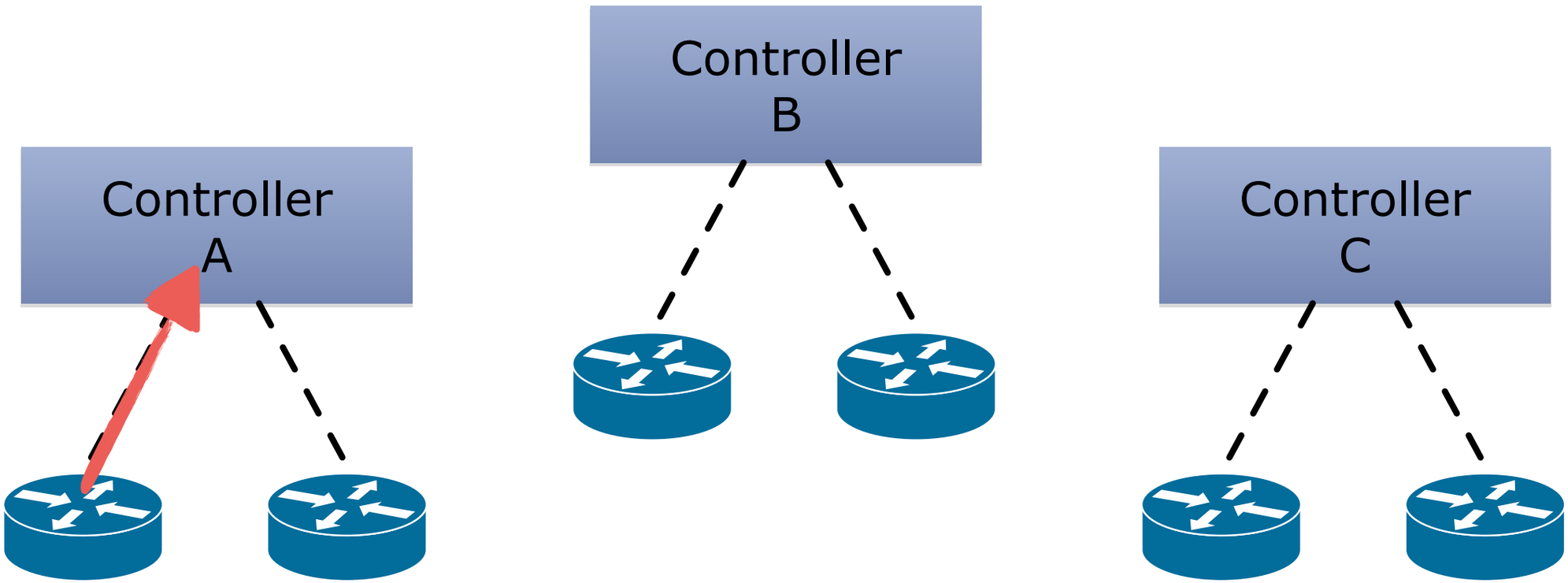
Layered Filesystem



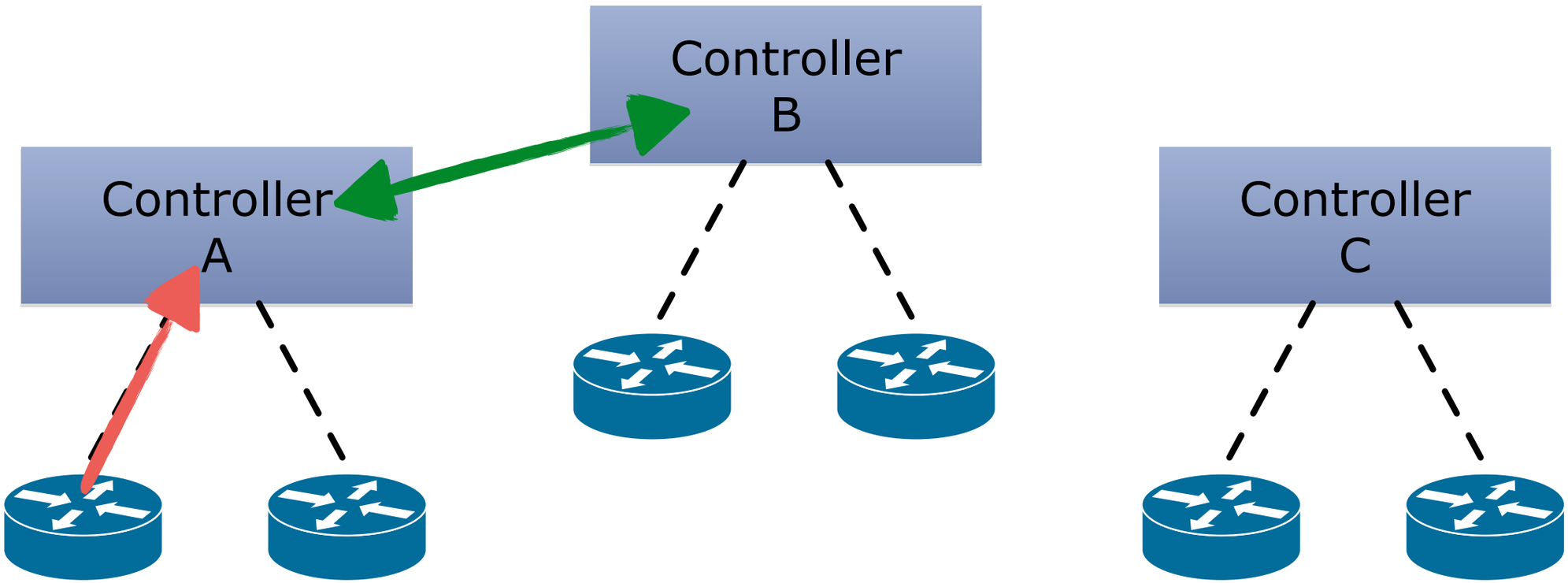
Locking



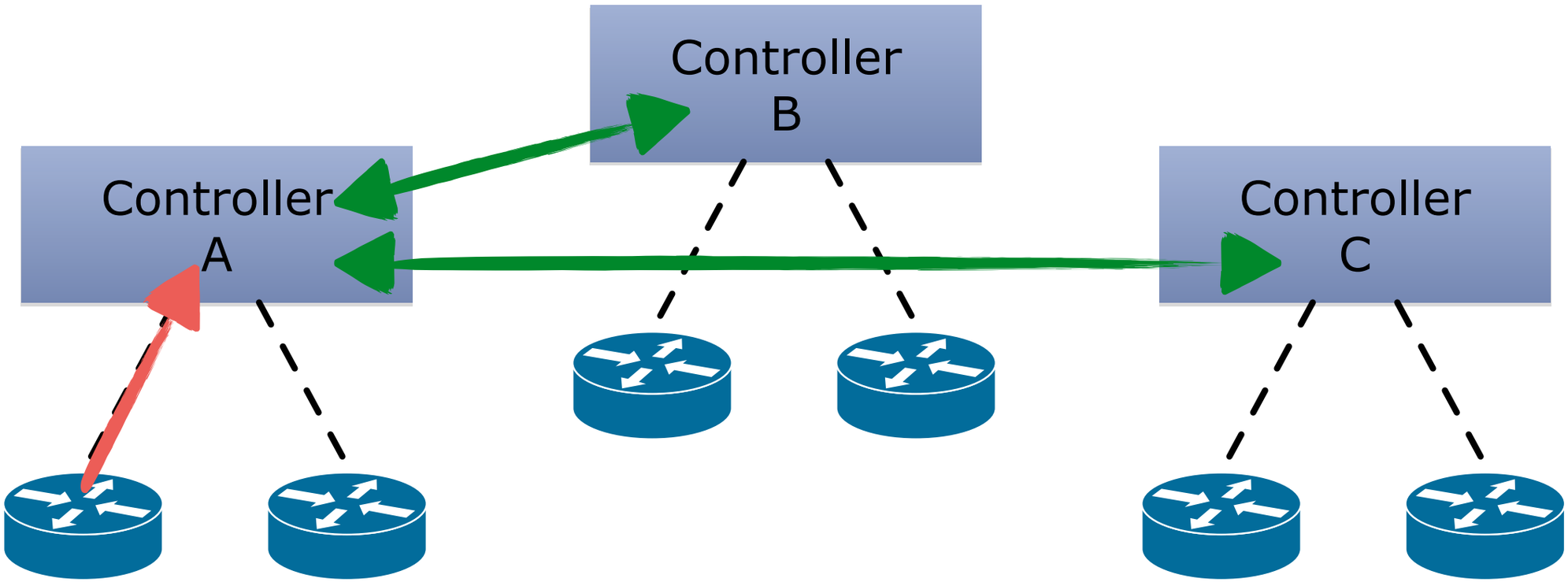
Locking



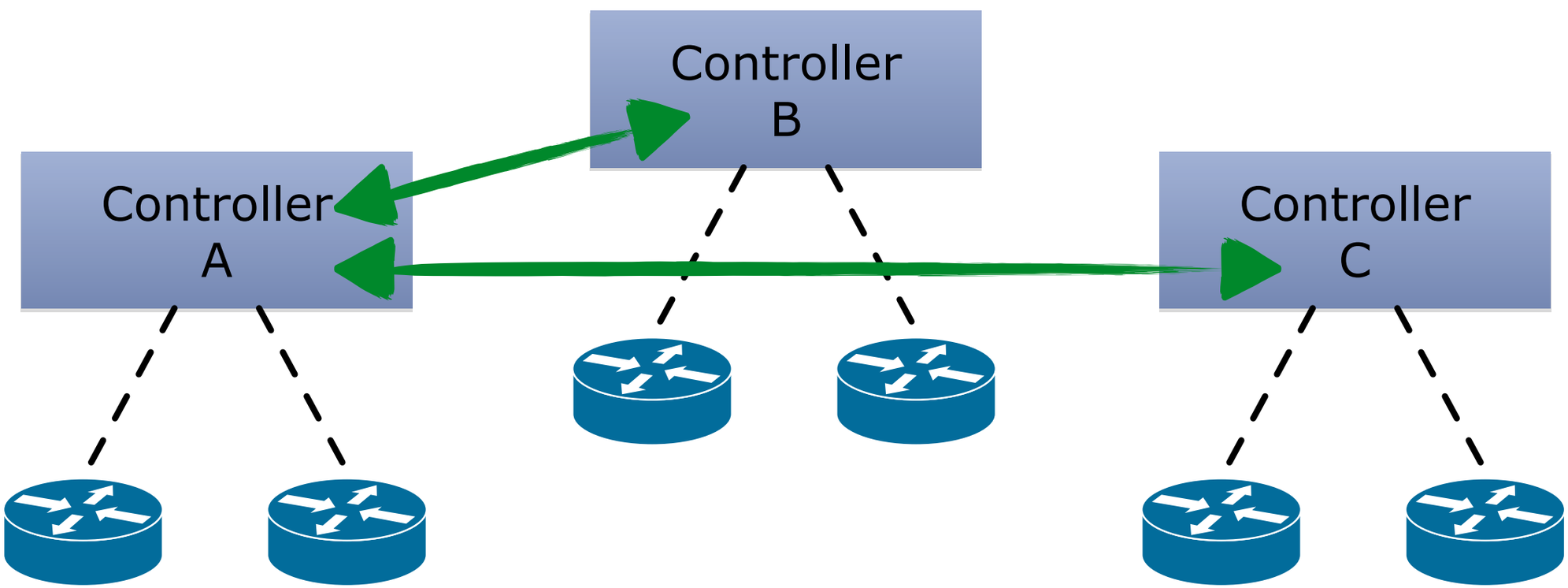
Locking



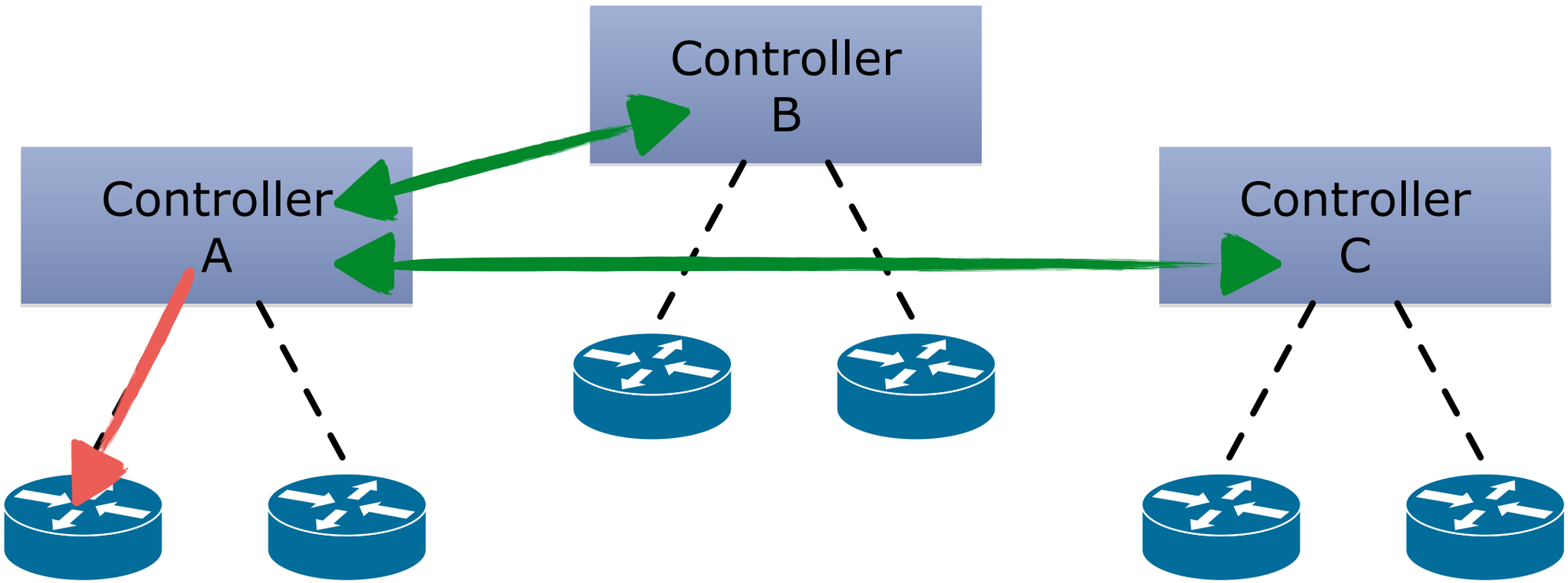
Locking



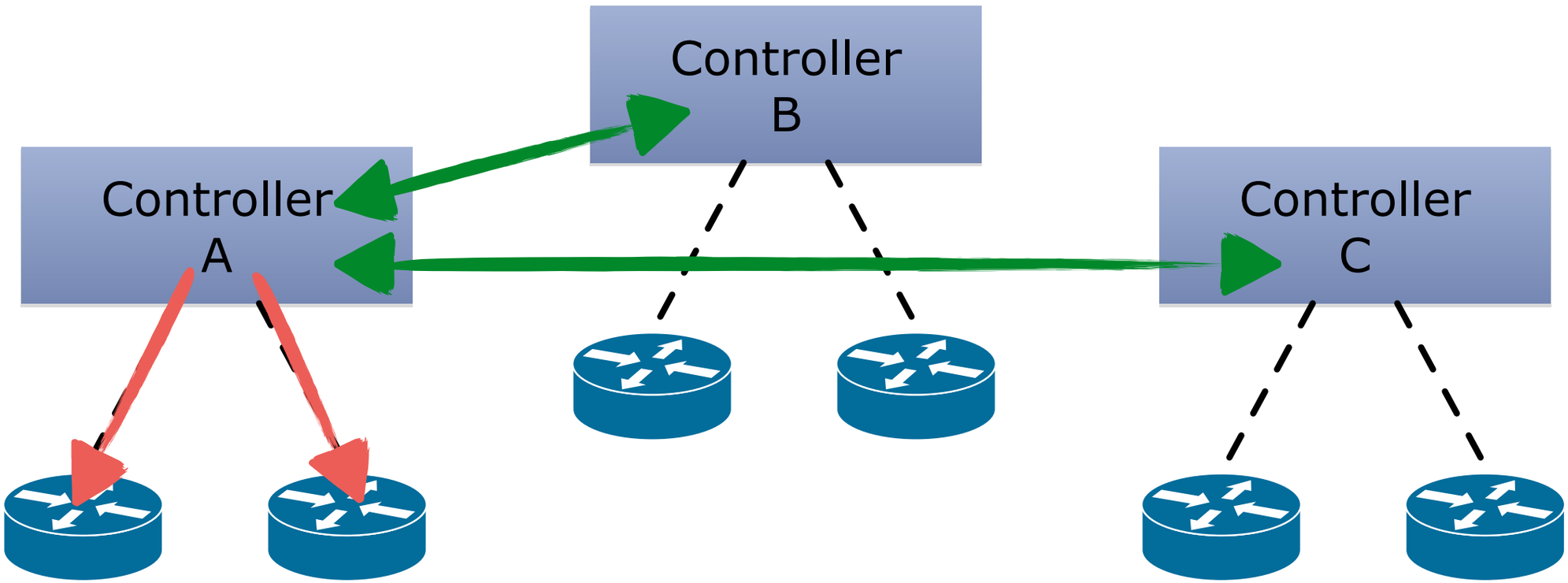
Locking



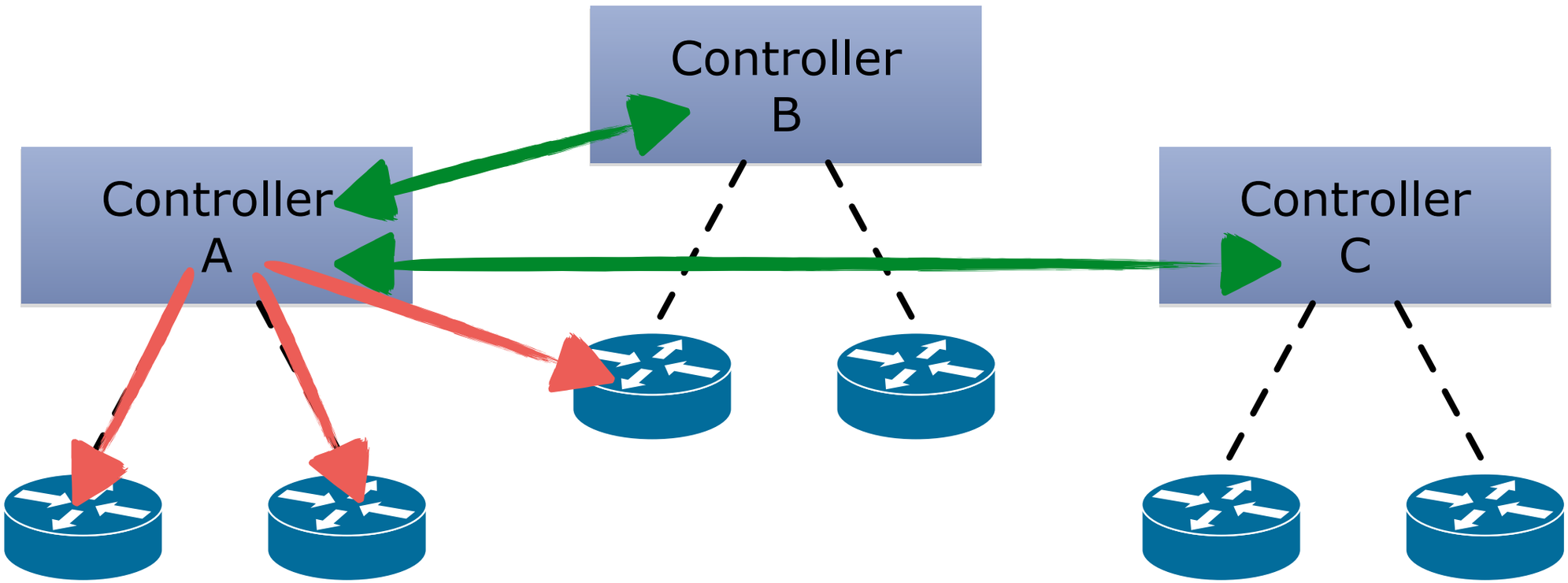
Locking



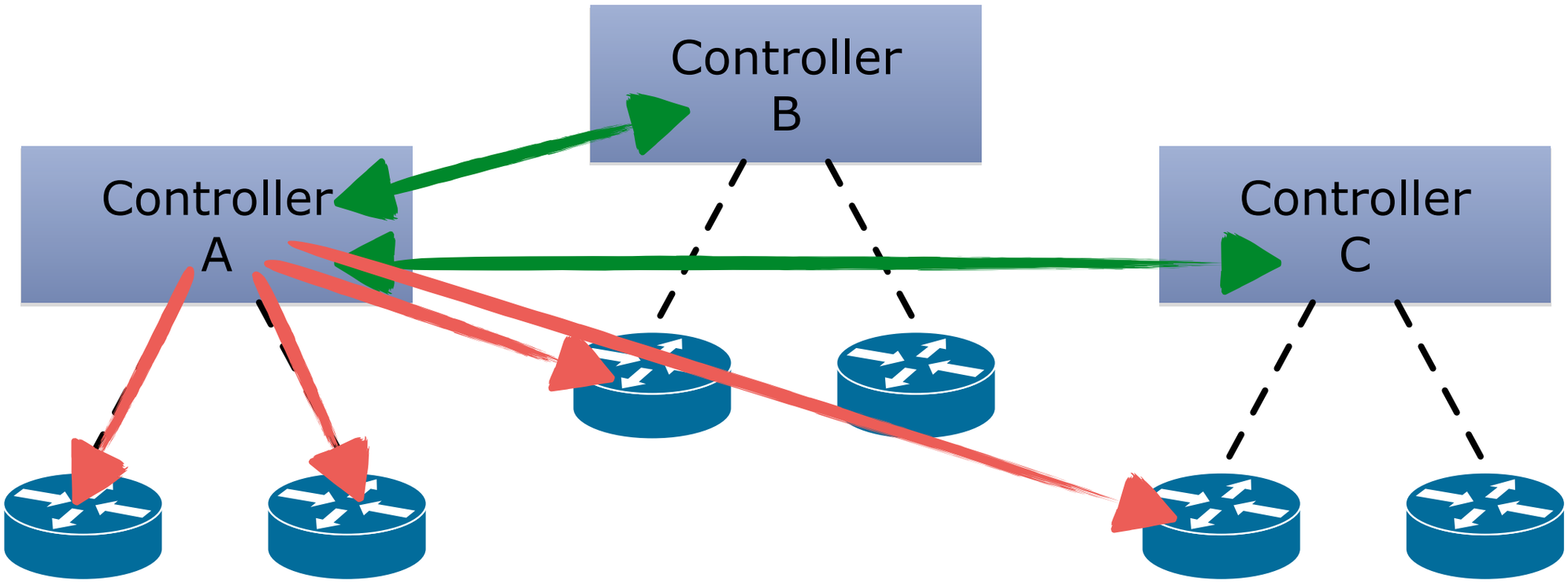
Locking



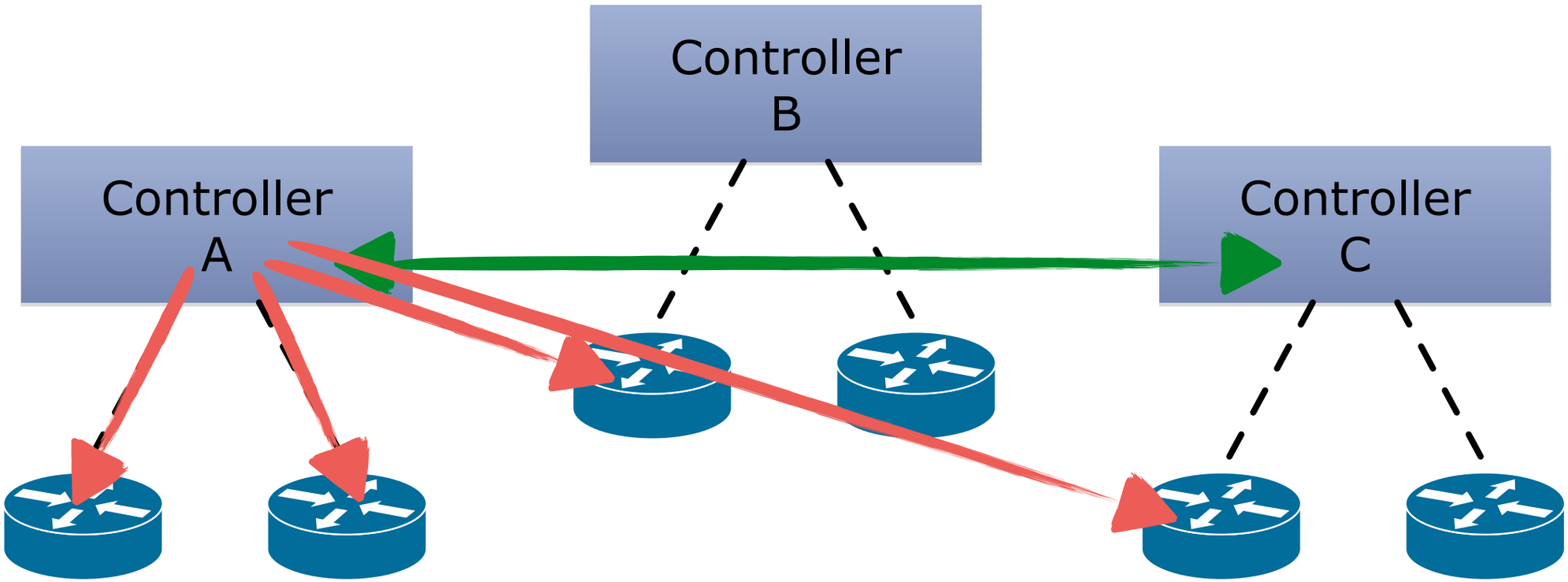
Locking



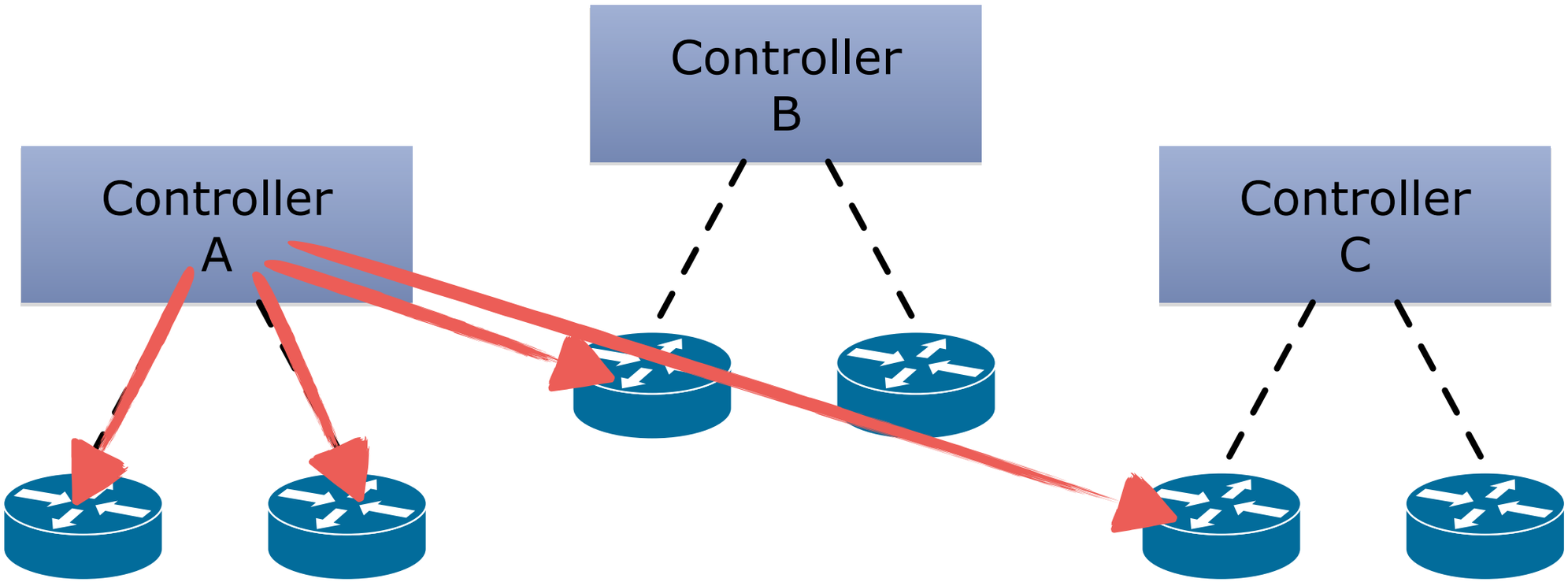
Locking



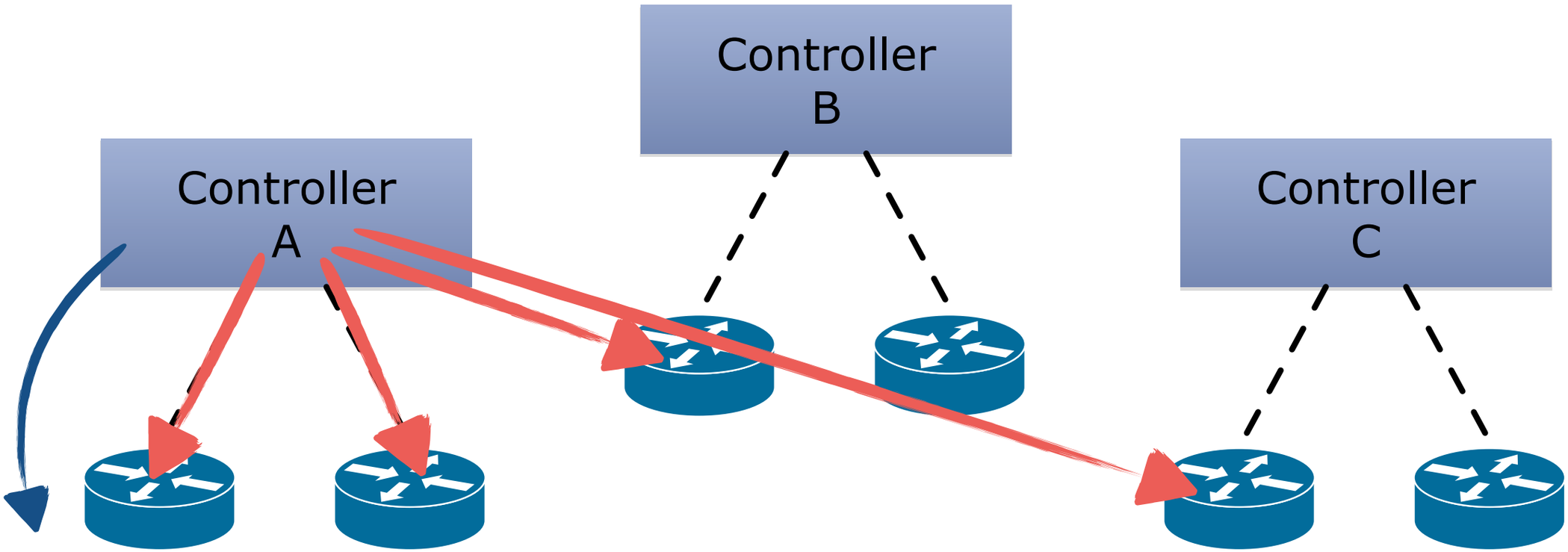
Locking



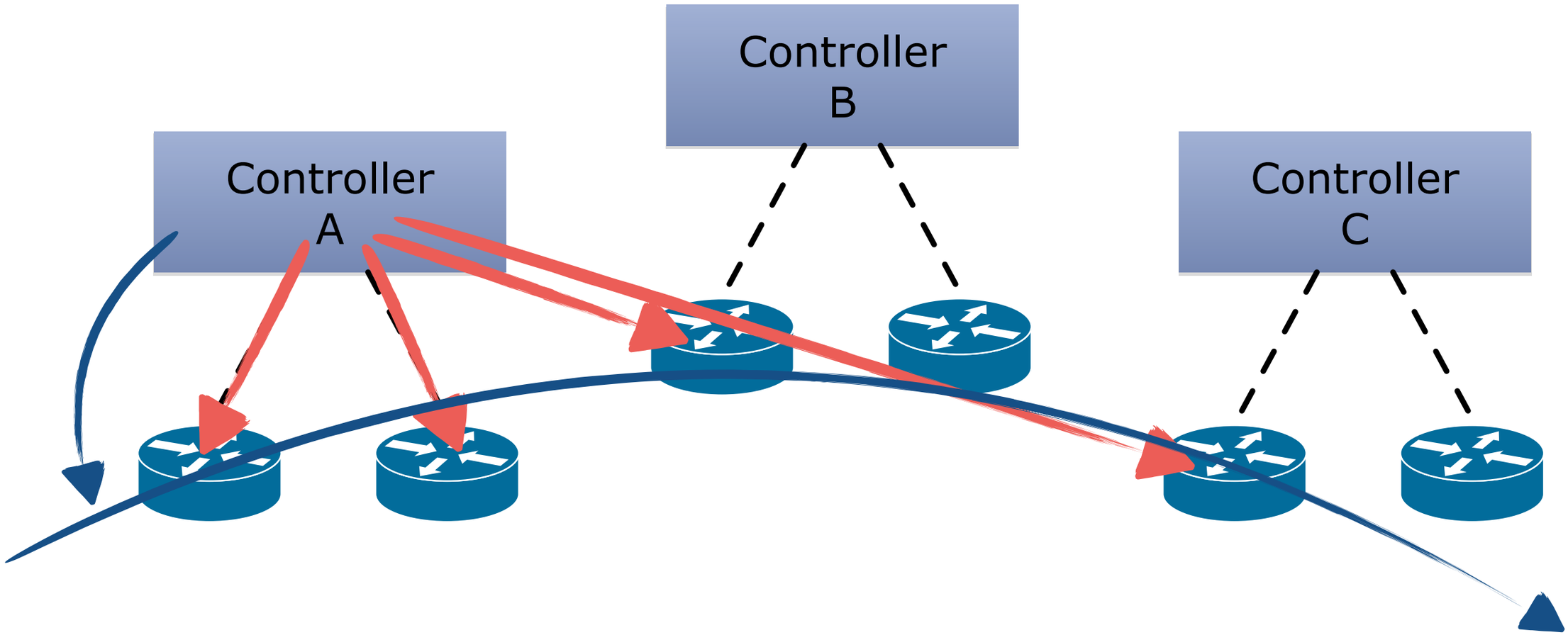
Locking



Locking



Locking



Prototype

- Filesystem implemented using FUSE (C)
- OpenFlow Driver (C++)
- Applications (Python and Ruby)
  - Topology Discovery
  - Learning Switch
  - Fast Router
  - Route Optimizer
  - Link Tapper
- several Bash management scripts and static flow pusher

Conclusion

- 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!

Matthew Monaco, Oliver Michel, and Eric Keller. Applying operating system principles to SDN controller design. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks (HotNets-XII)*. 2013.

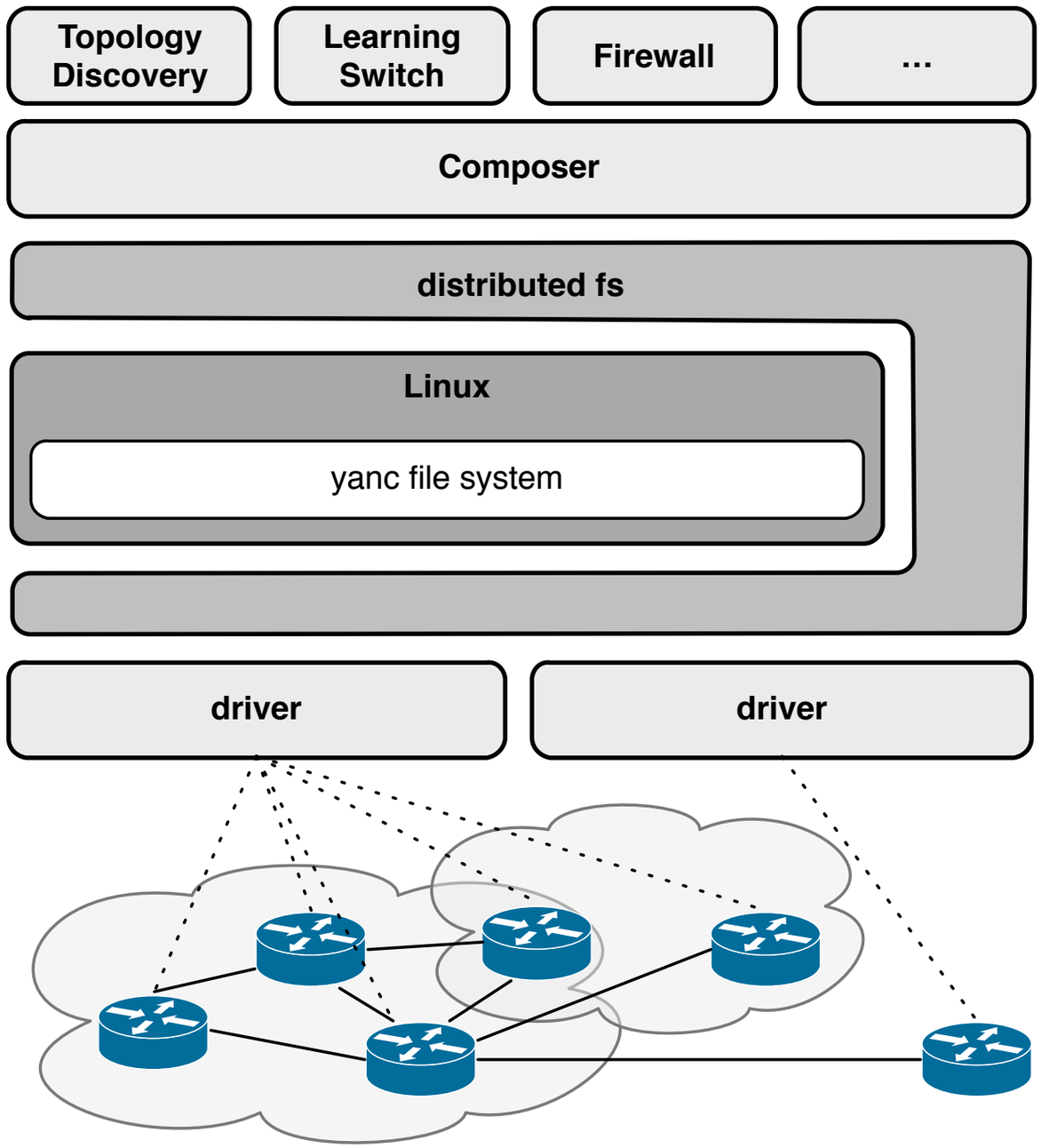
Backup Slides

Programmability

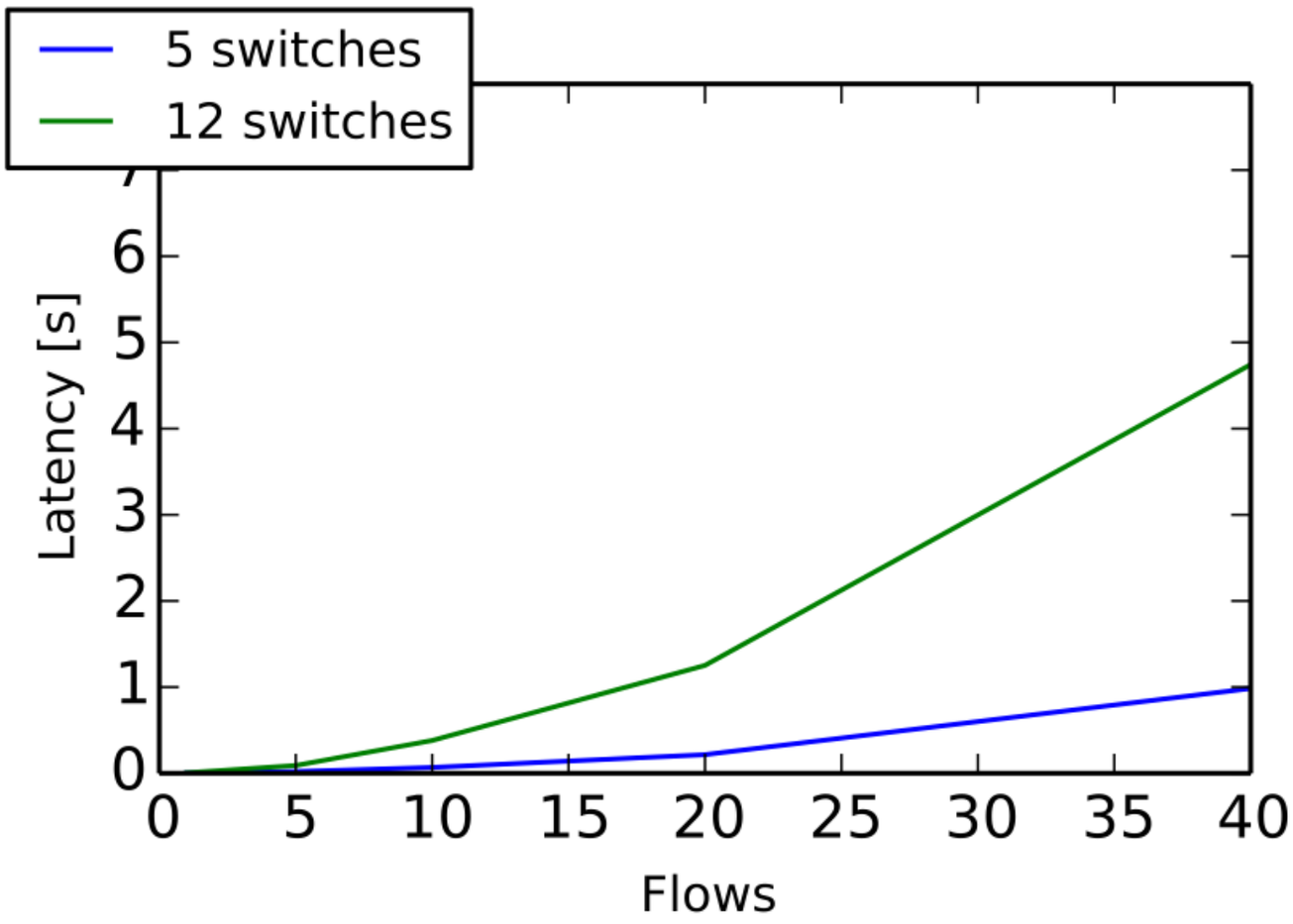
```
#!/usr/bin/env python3

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_*/
```

Architecture



Performance



file system	op	ops/s
yancfs	mkdir	10245
yancfs	rmdir	11192
tmpfs	mkdir	721877
tmpfs	rmdir	782249

Shared Library