Towards Correct Network Virtualization

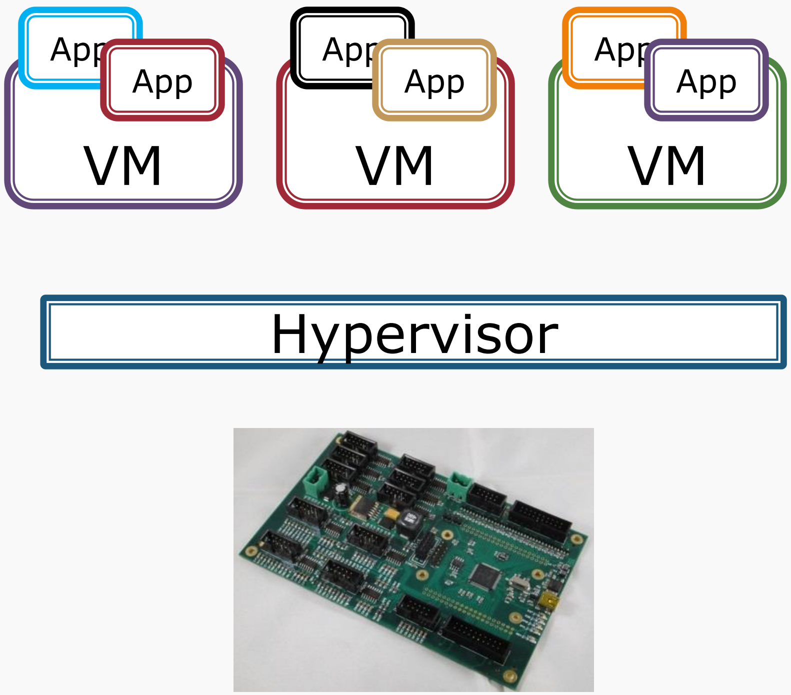
Soudeh Ghorbani

Brighten Godfrey

UIUC

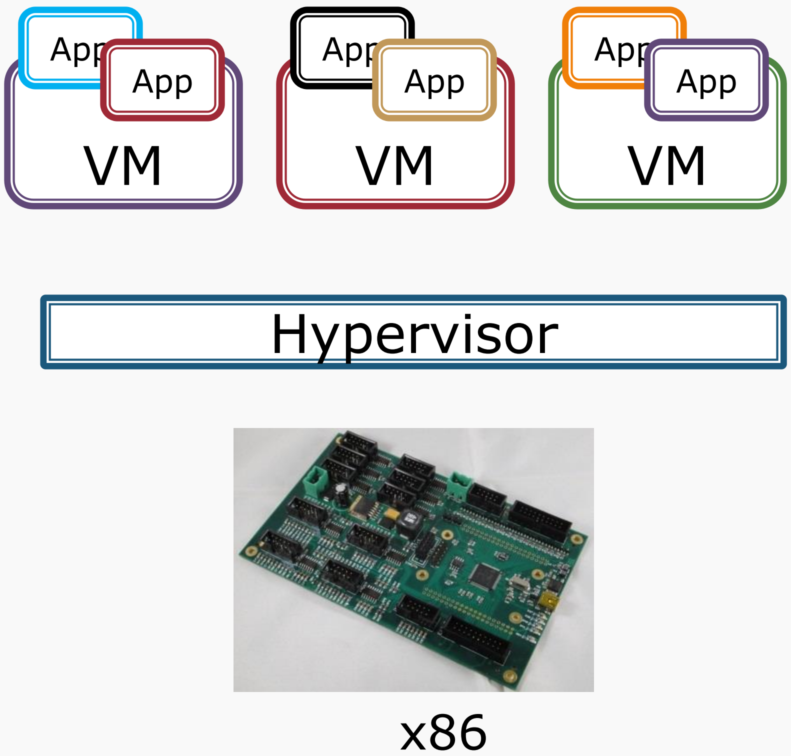
HotSDN 2014

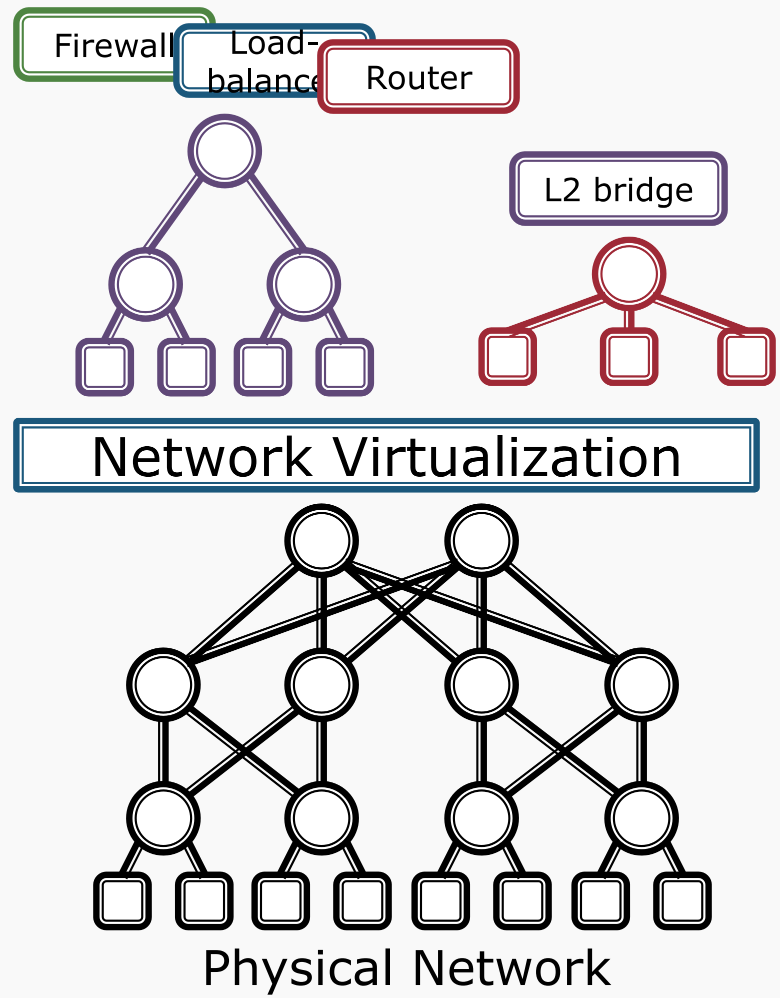
Virtualization



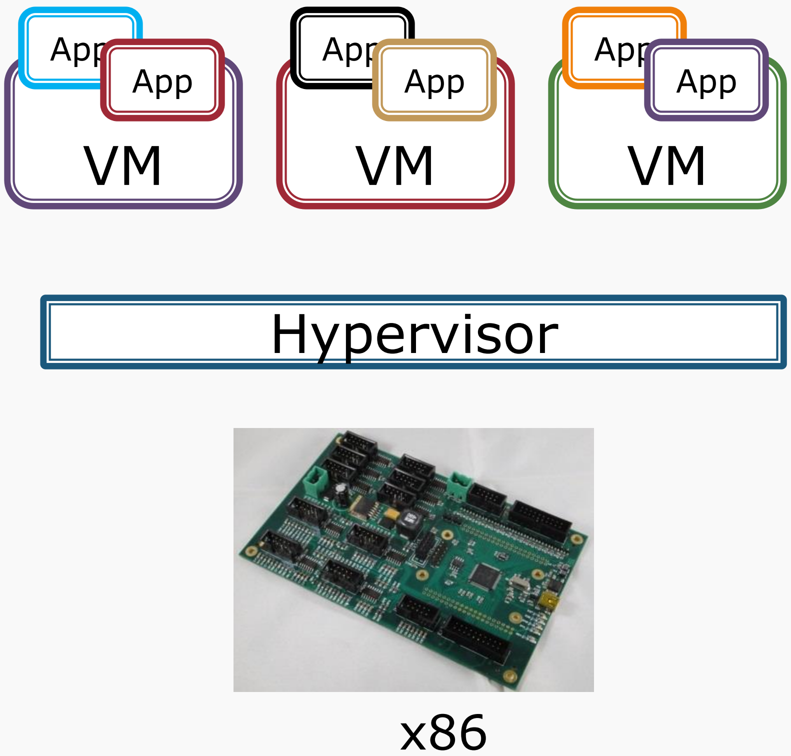
x86

Virtualization





Virtualization



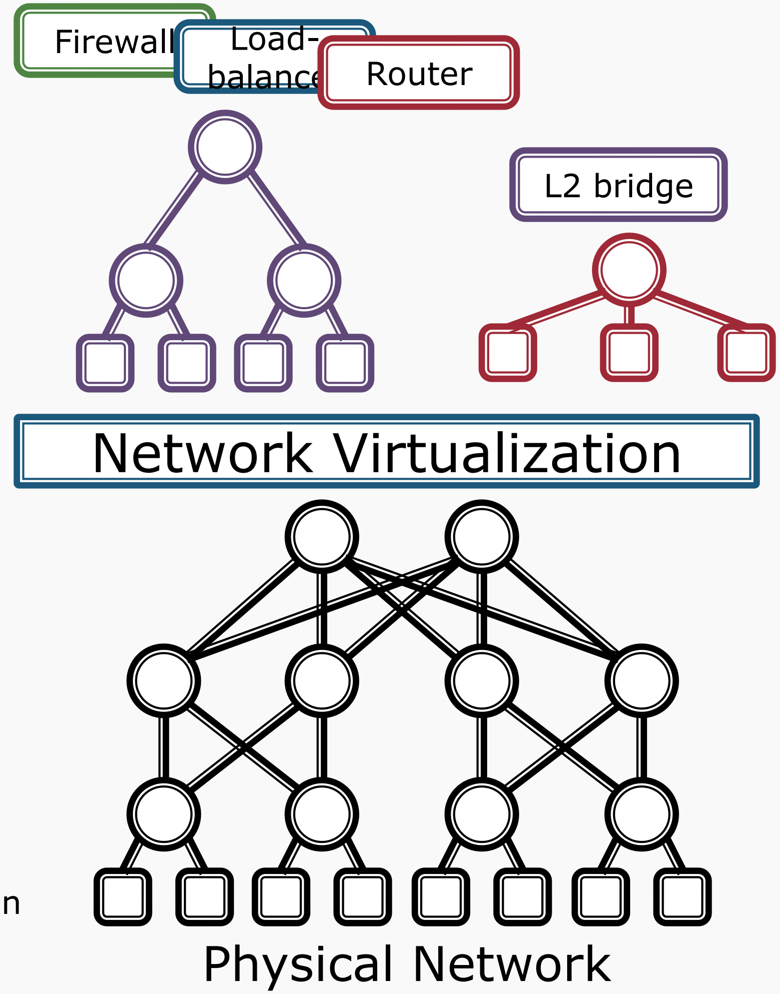
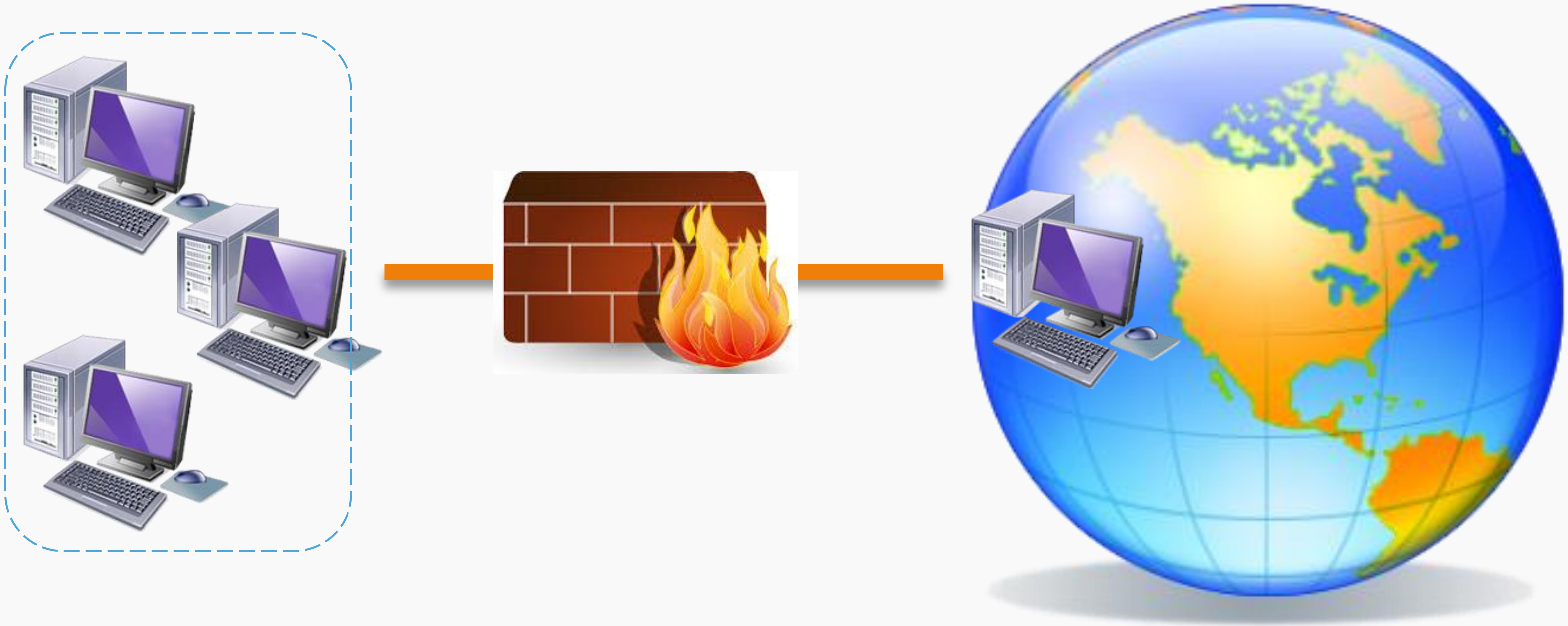


Diagram inspired by Teemu Koponen's NSDI 2014 talk on "*Network Virtualization in Multi-tenant Datacenters*".

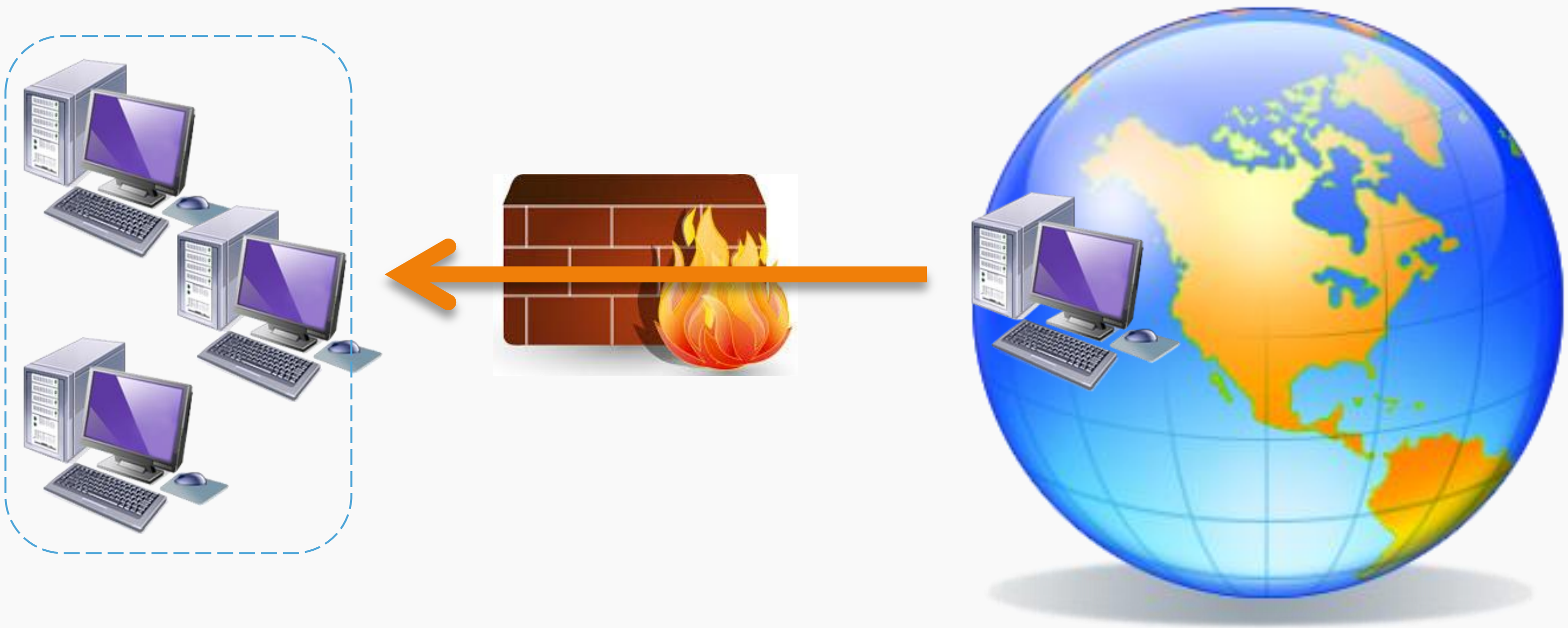
Is the physical implementation a faithful reproduction of the virtual network?

Virtual firewall



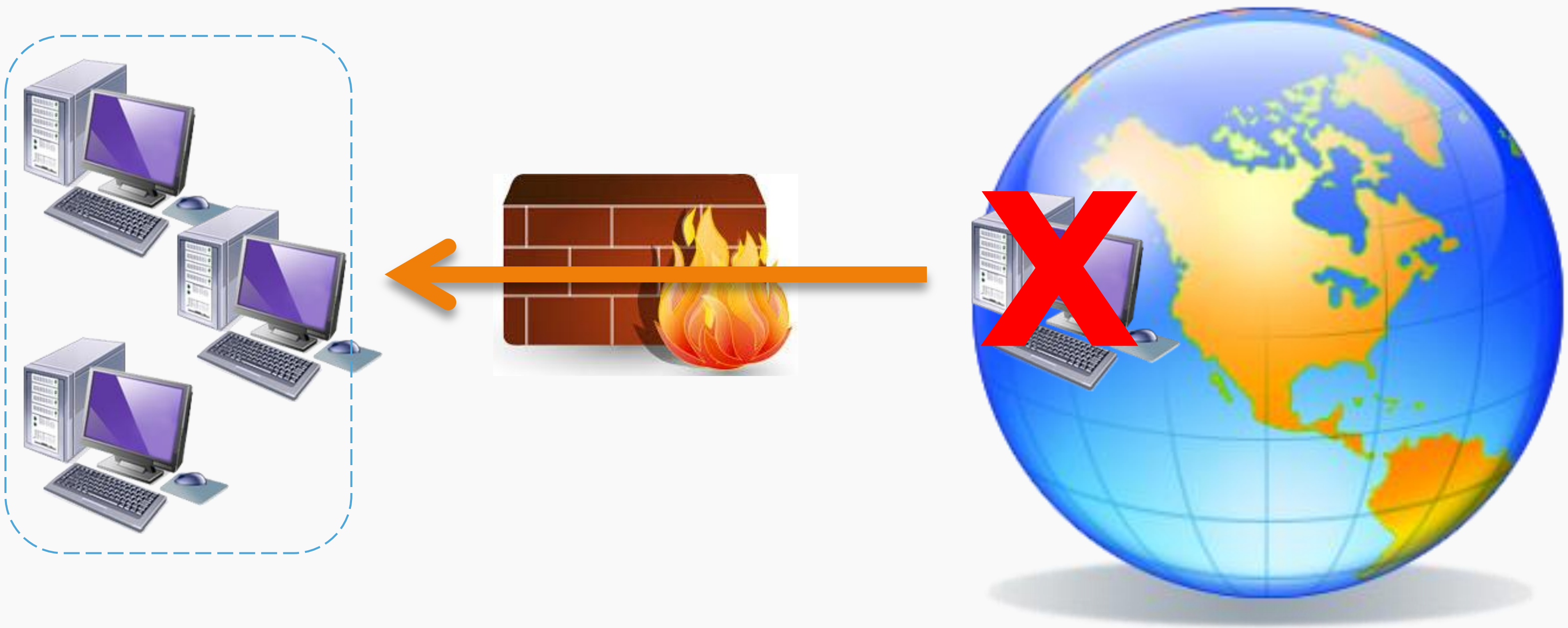
Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.

Virtual firewall



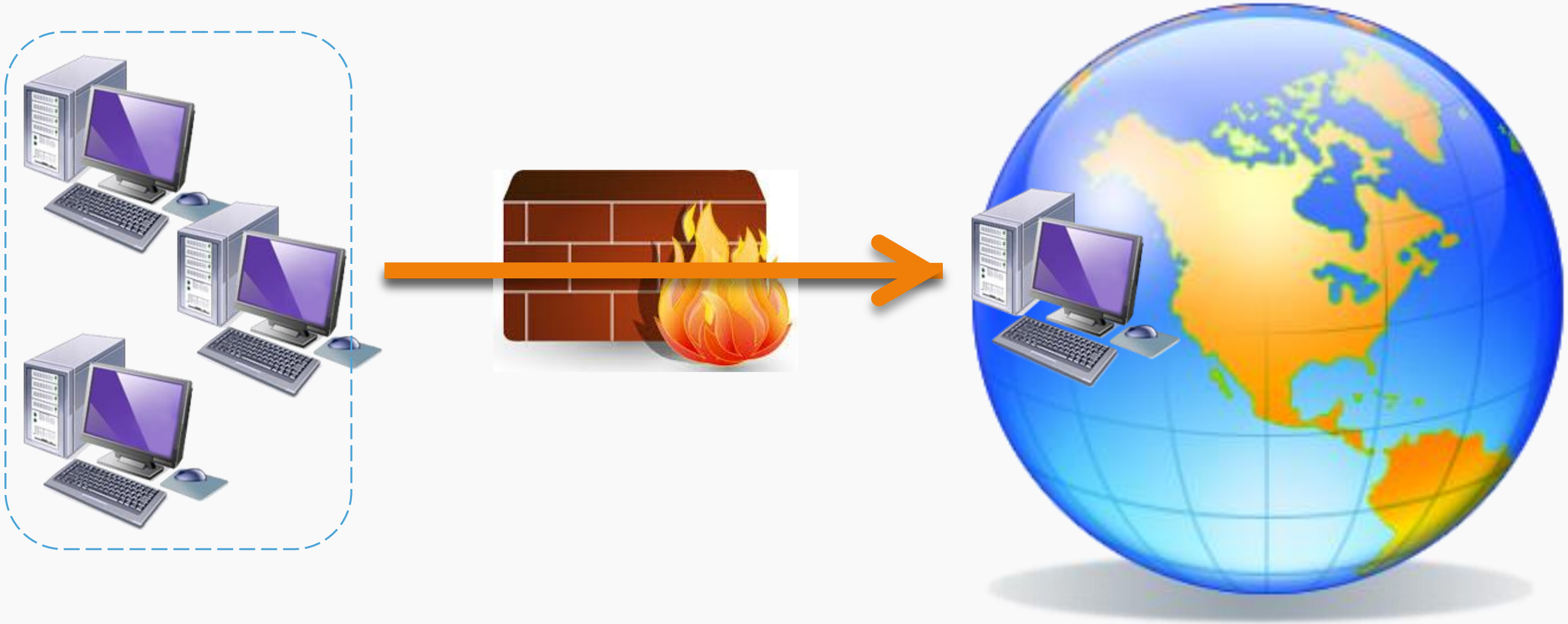
Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.

Virtual firewall



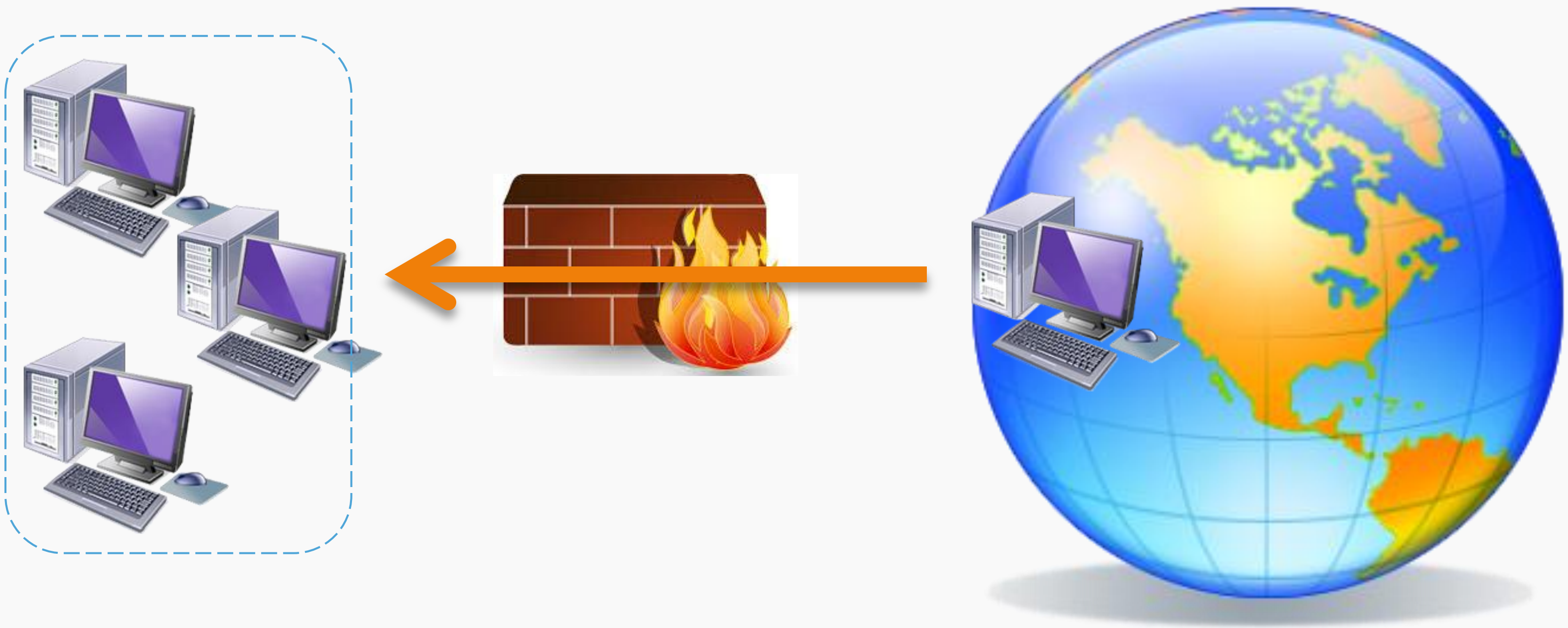
Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.

Virtual firewall



Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.

Virtual firewall



Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0	*	Send to controller

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0	*	Send to controller

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0	*	Send to controller

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0	*	Send to controller

(Part of the) Firewall Controller App

```
switch(msg.getType()) {
   case PACKET_IN:
      if ( internal.contains(msg.srcMAC()) ) {
         whitelisted[msg.dstMAC()][msg.srcMACA()] = true;
       }else {
          if (whitelisted[msg.srcMAC()][msg.dstMAC()] ){
             whitelist(sw, msg);
          }else{
              blacklist(sw, msg);
          }
      }
```

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0		

Packet-in from an internal client? Save state: dst server is allowed to send back.

(Part of the) Firewall Controller App

```
swi
   case PACKET_IN:
      if ( internal.contains(msg.srcMAC()) ) {
         whitelisted[msg.dstMAC()][msg.srcMACA()] = true;
      }else {
         if (whitelisted[msg.srcMAC()][msg.dstMAC()] ){
            whitelist(sw, msg);
         }else{
             blacklist(sw, msg);
         }
      }
```

Virtual firewall app

Firewall Switch

Prio rity	Flow	Action
10	srcip=130.126.*.*	Send to controller, fwd(1)
0	*	Send to controller

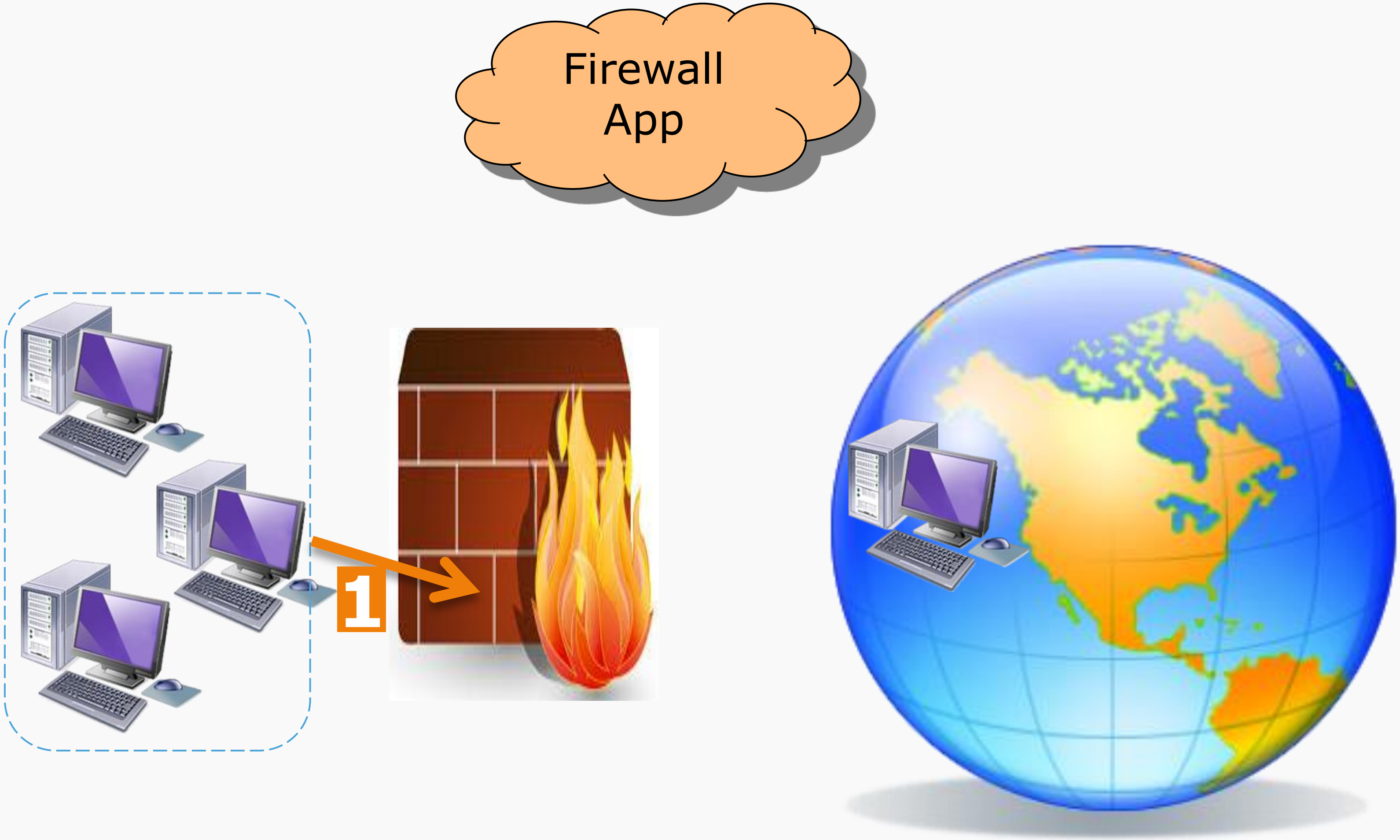
(Part of the) Firewall Controller App

```
switch(msg.getType()) {
   case PACKET_IN:
       if ( internal.contains(msg.srcMAC()) ) {
           whitelisted[msg.dstMAC()][msg.srcMACA()] = true;
        }else {
           if (whitelisted[msg.srcMAC()][msg.dstMAC()] )
               whitelist(sw, msg);
           }else{
               blacklist(sw, msg);
           }
       }
```

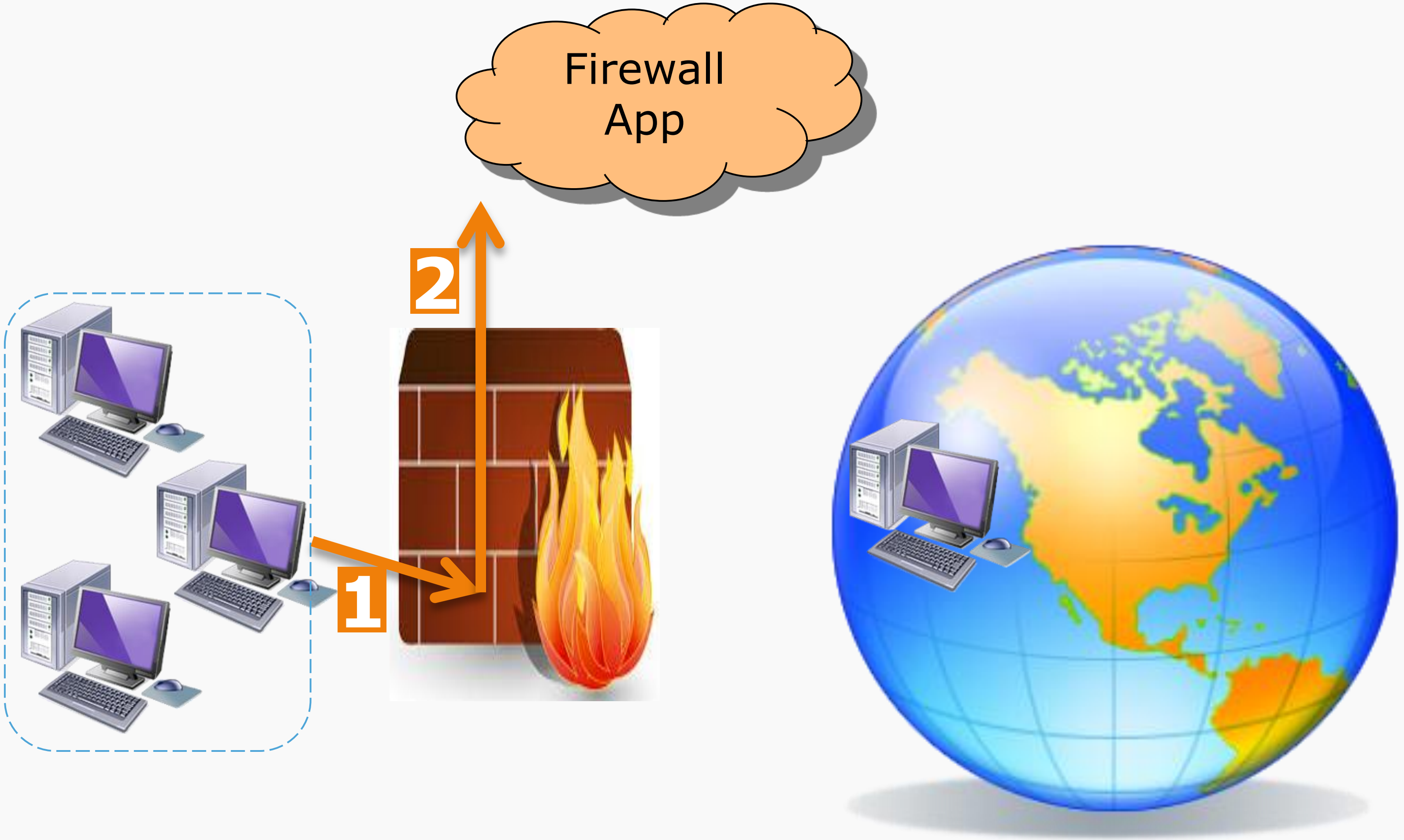
Packet-in from an external server?

- If the server is allowed to send, install rules to allow bidirectional traffic.
- Else, blacklist the external server.

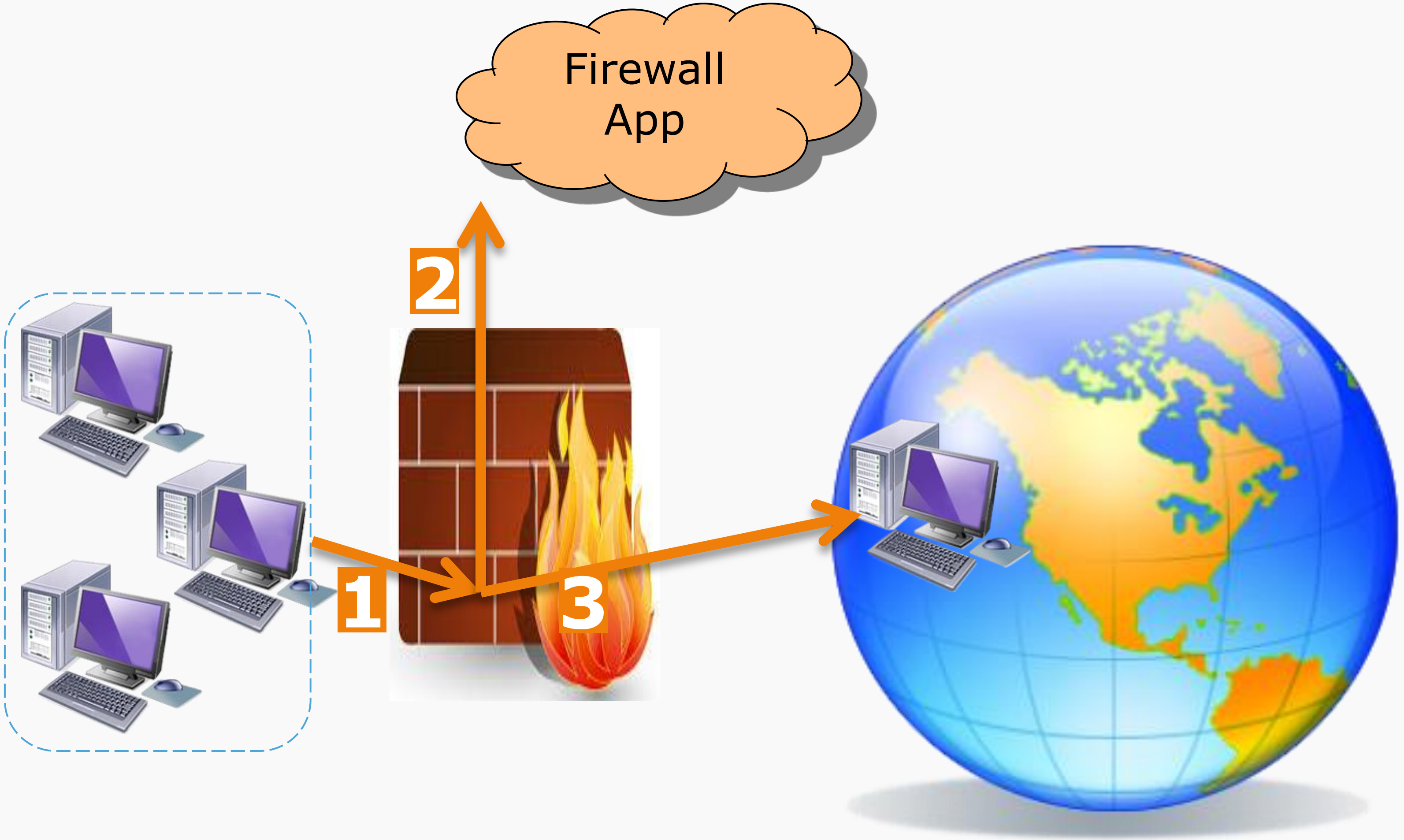
Virtual firewall



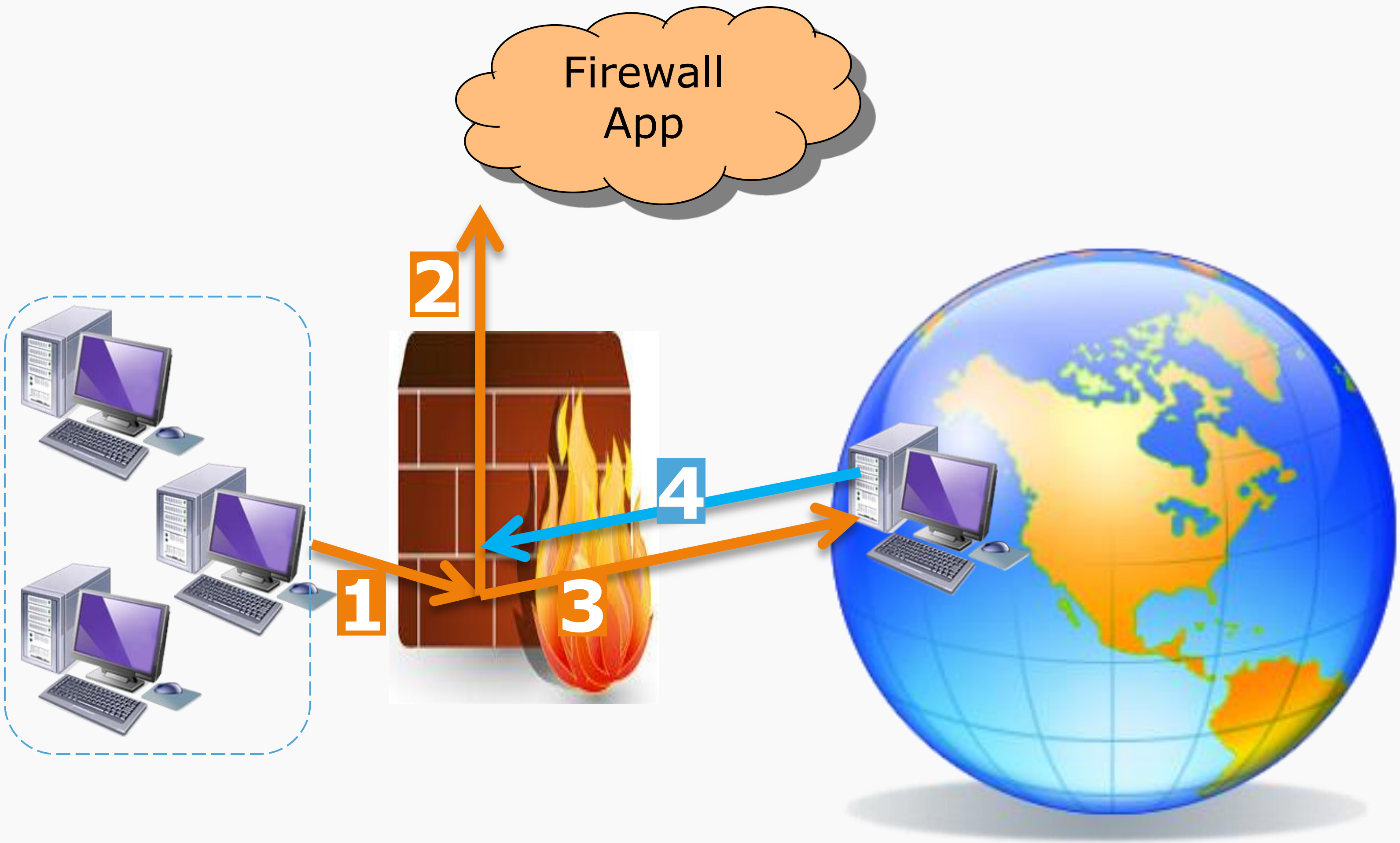
Virtual firewall



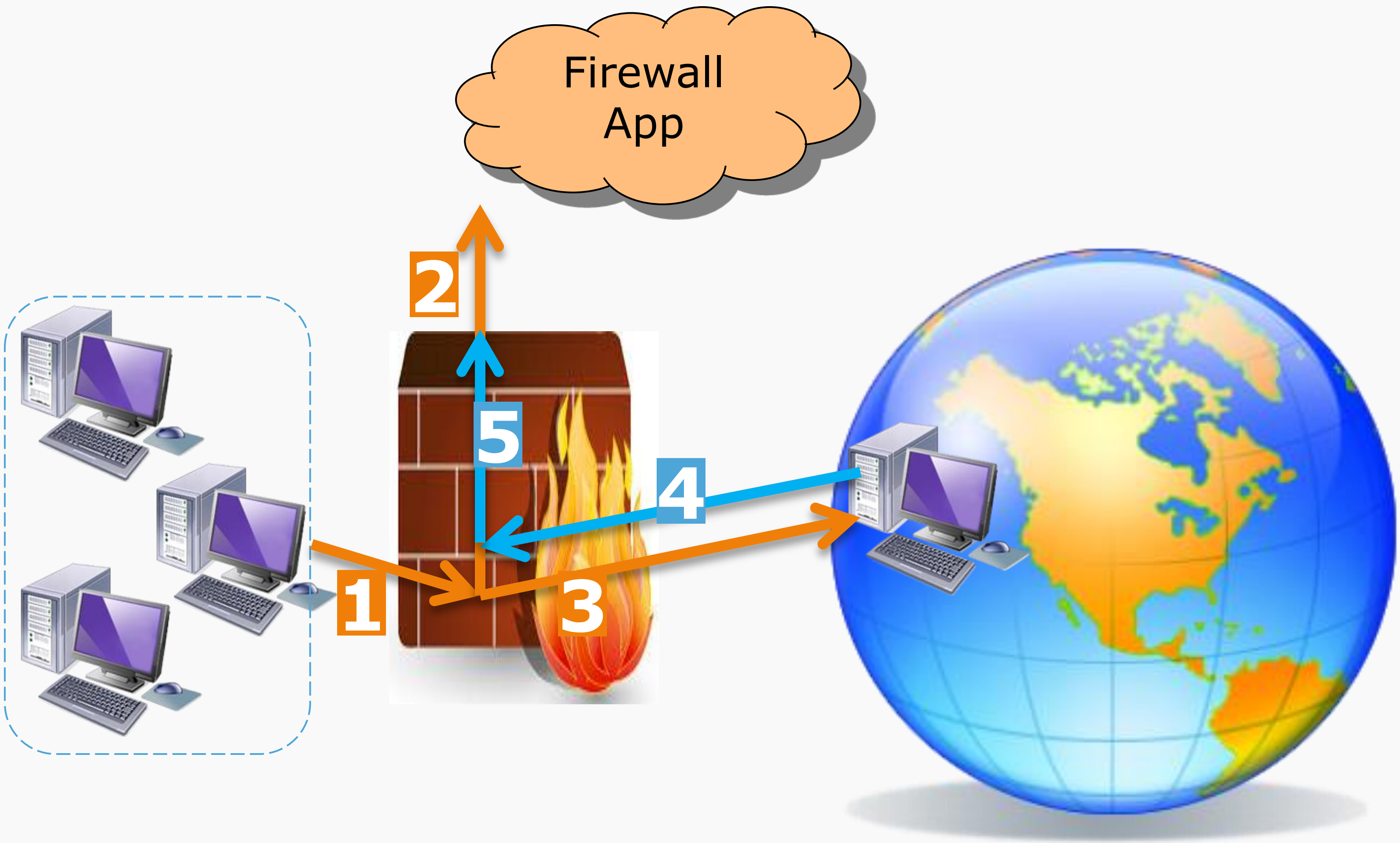
Virtual firewall



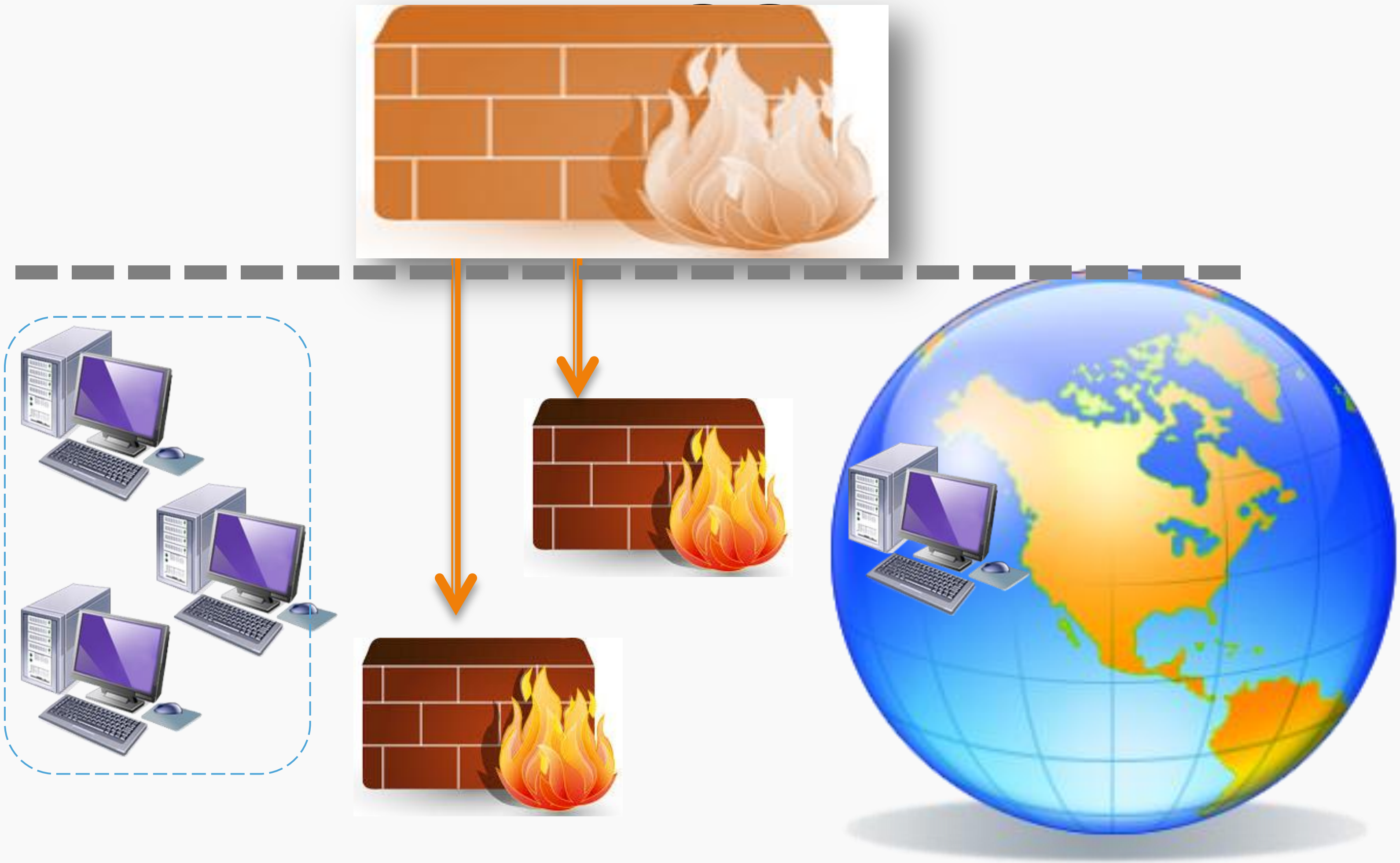
Virtual firewall



Virtual firewall

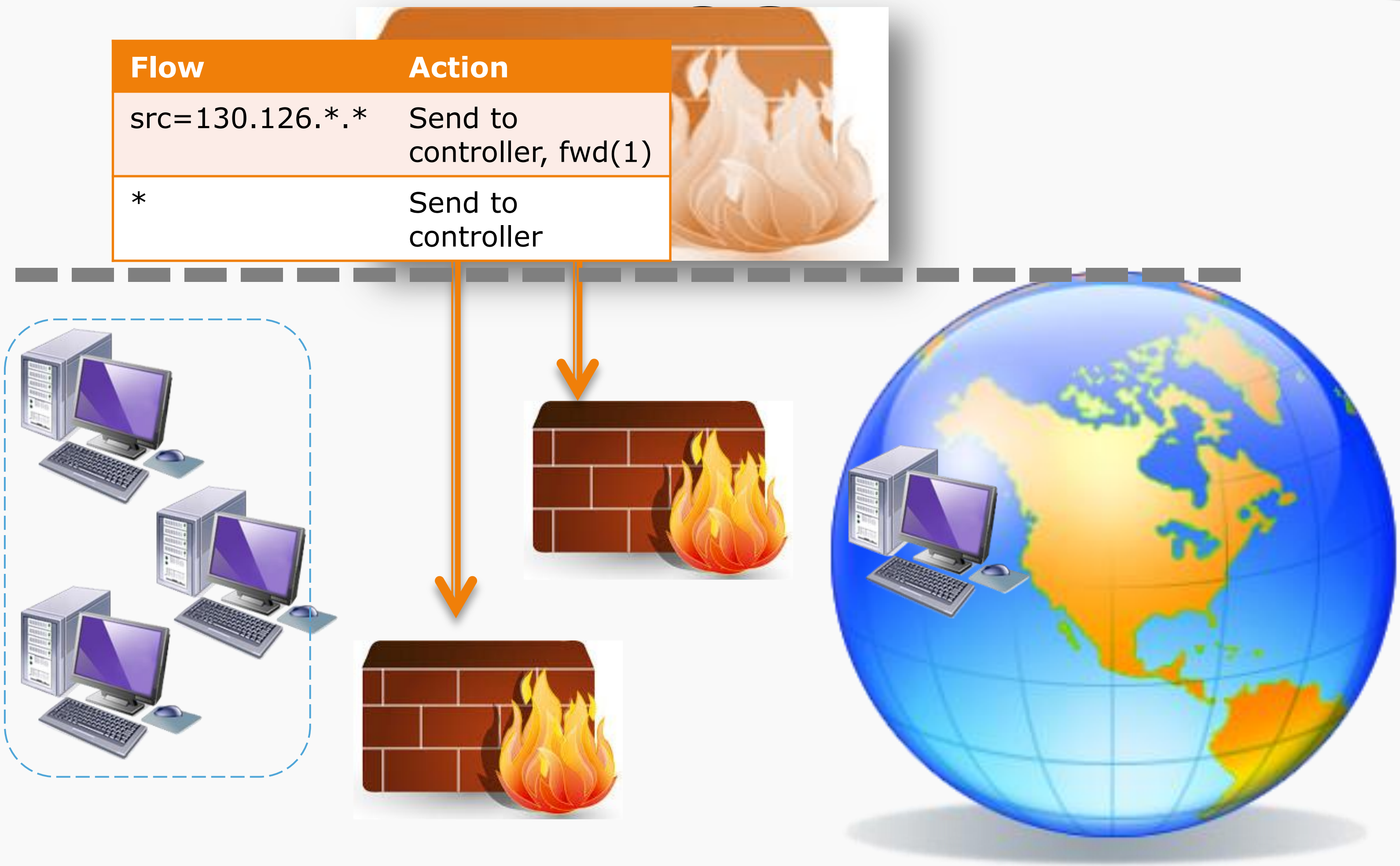


Firewall + virtualization = bug

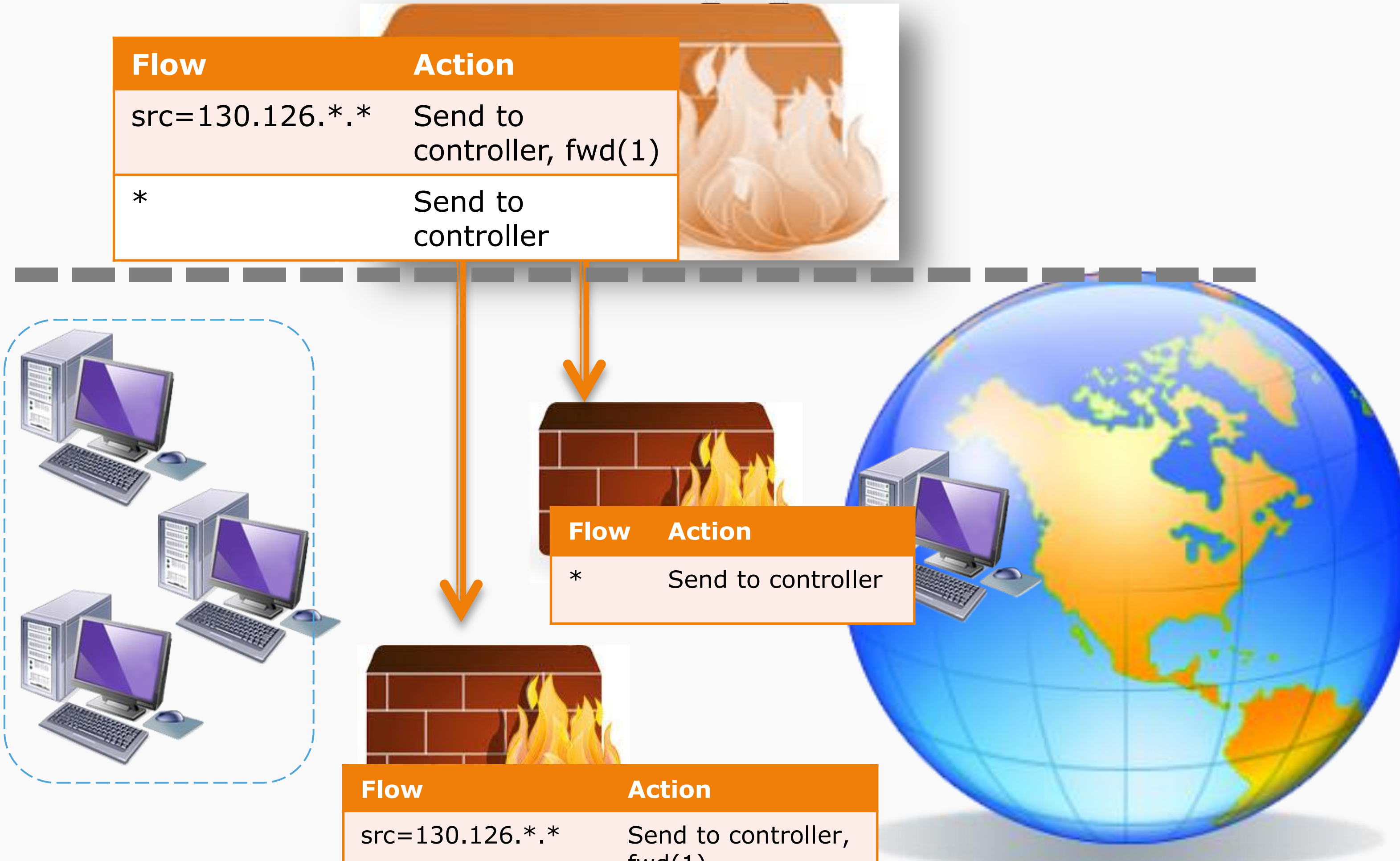


Firewall + virtualization = bug

Flow	Action
src=130.126.*.*	Send to controller, fwd(1)
*	Send to controller



Firewall + virtualization = bug

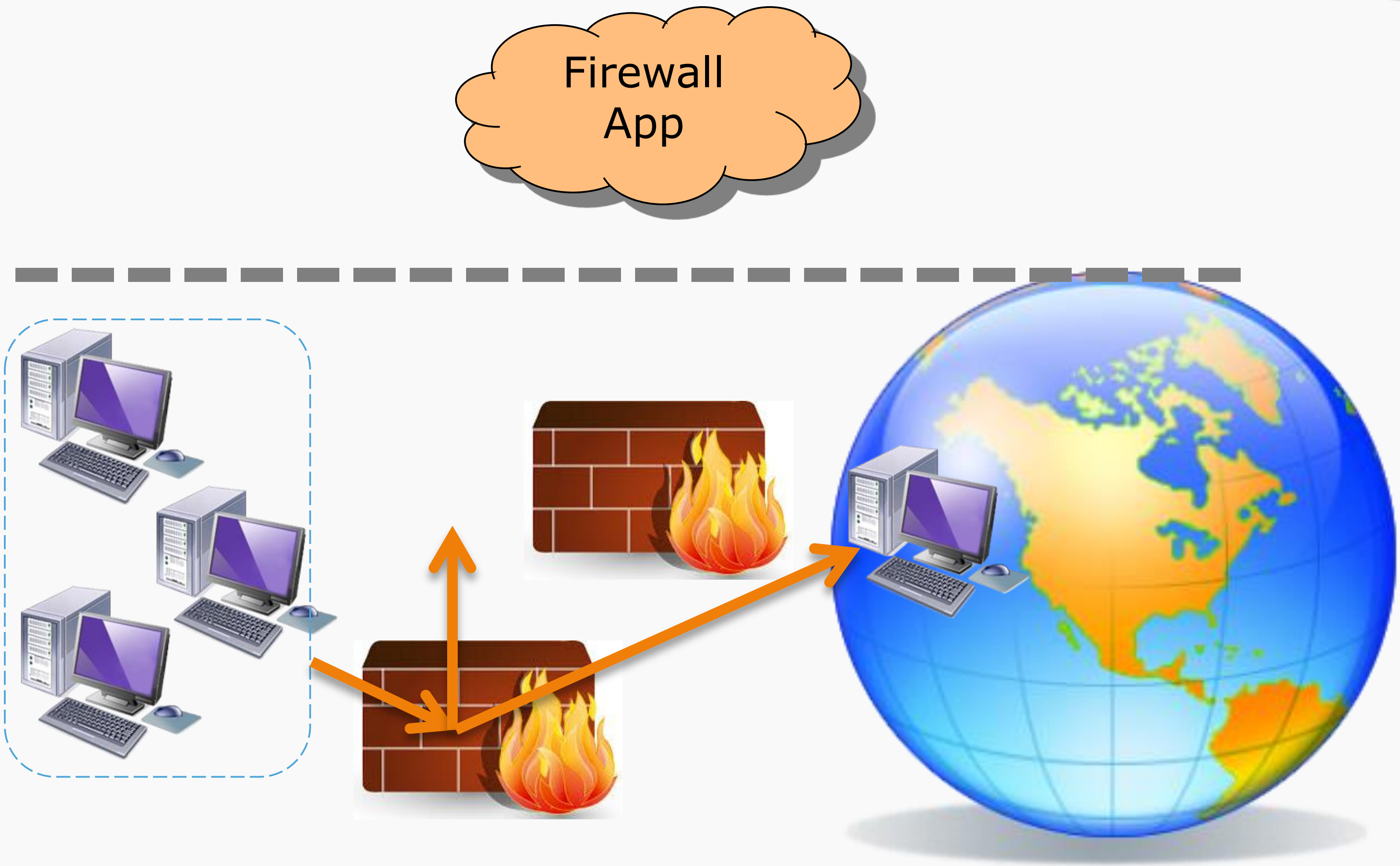


Flow	Action
src=130.126.*.*	Send to controller, fwd(1)
*	Send to controller

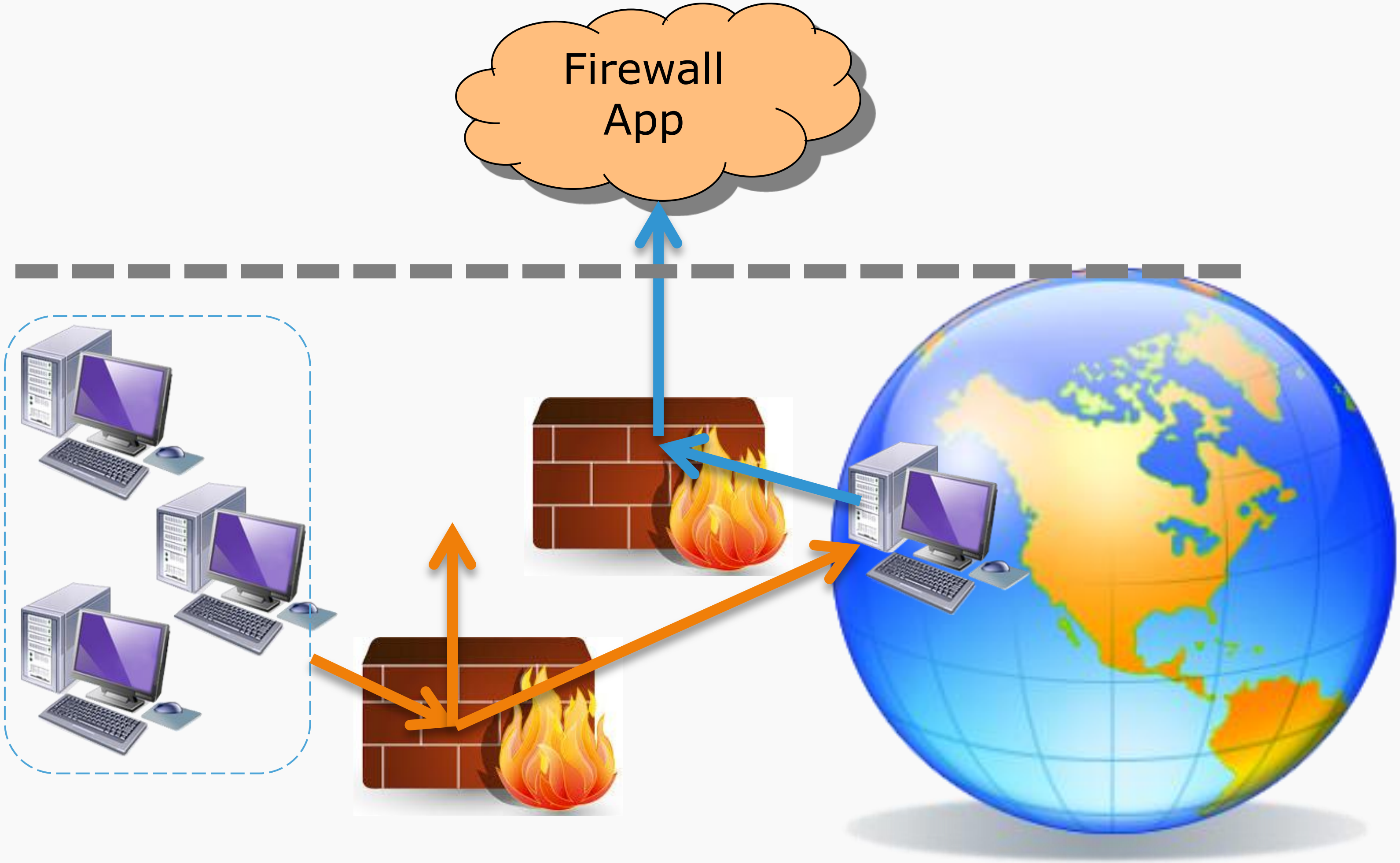
Flow	Action
*	Send to controller

Flow	Action
src=130.126.*.*	Send to controller, fwd(1)

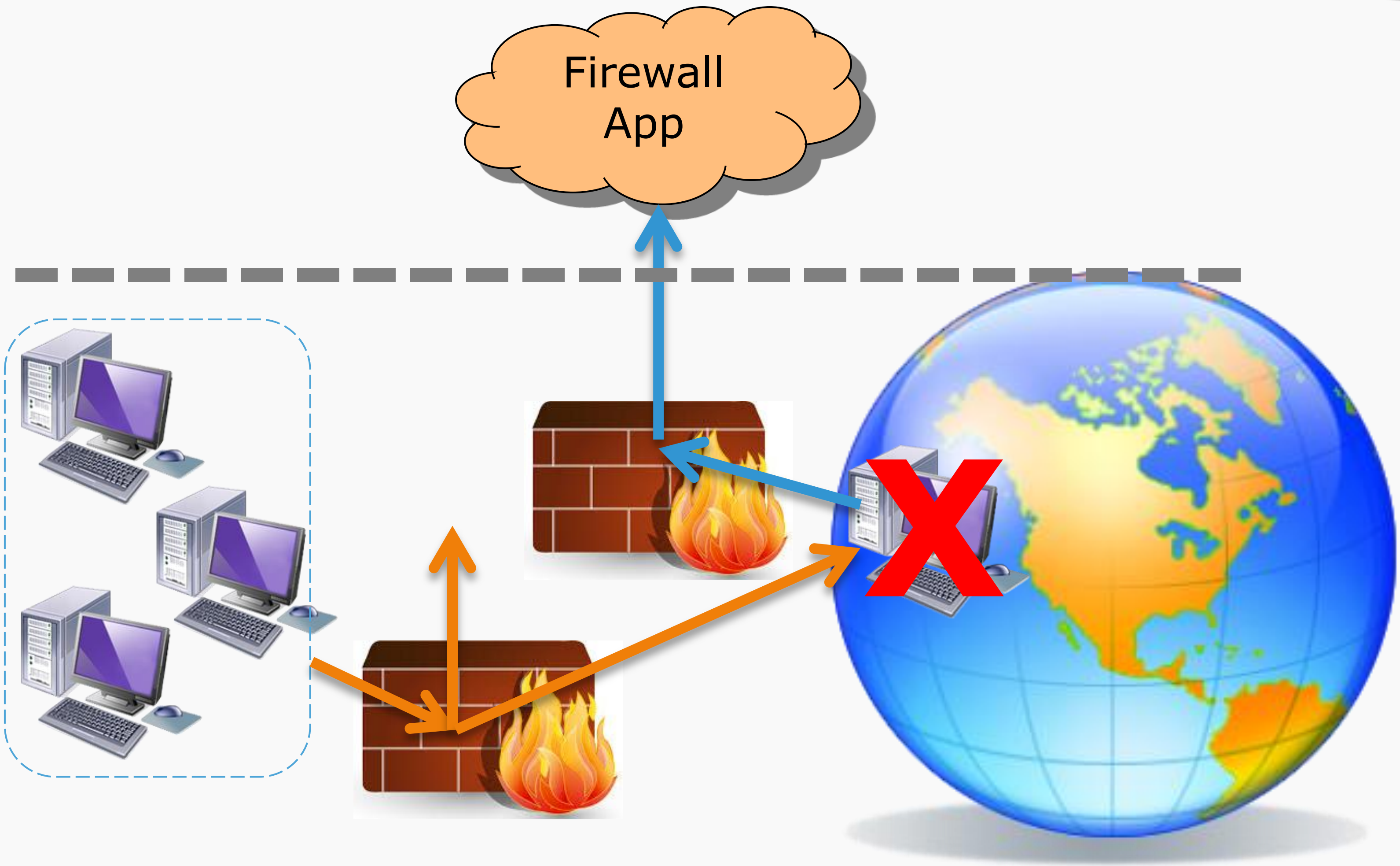
Firewall + virtualization = bug



Firewall + virtualization = bug



Firewall + virtualization = bug



Network virtualization: What could go wrong?

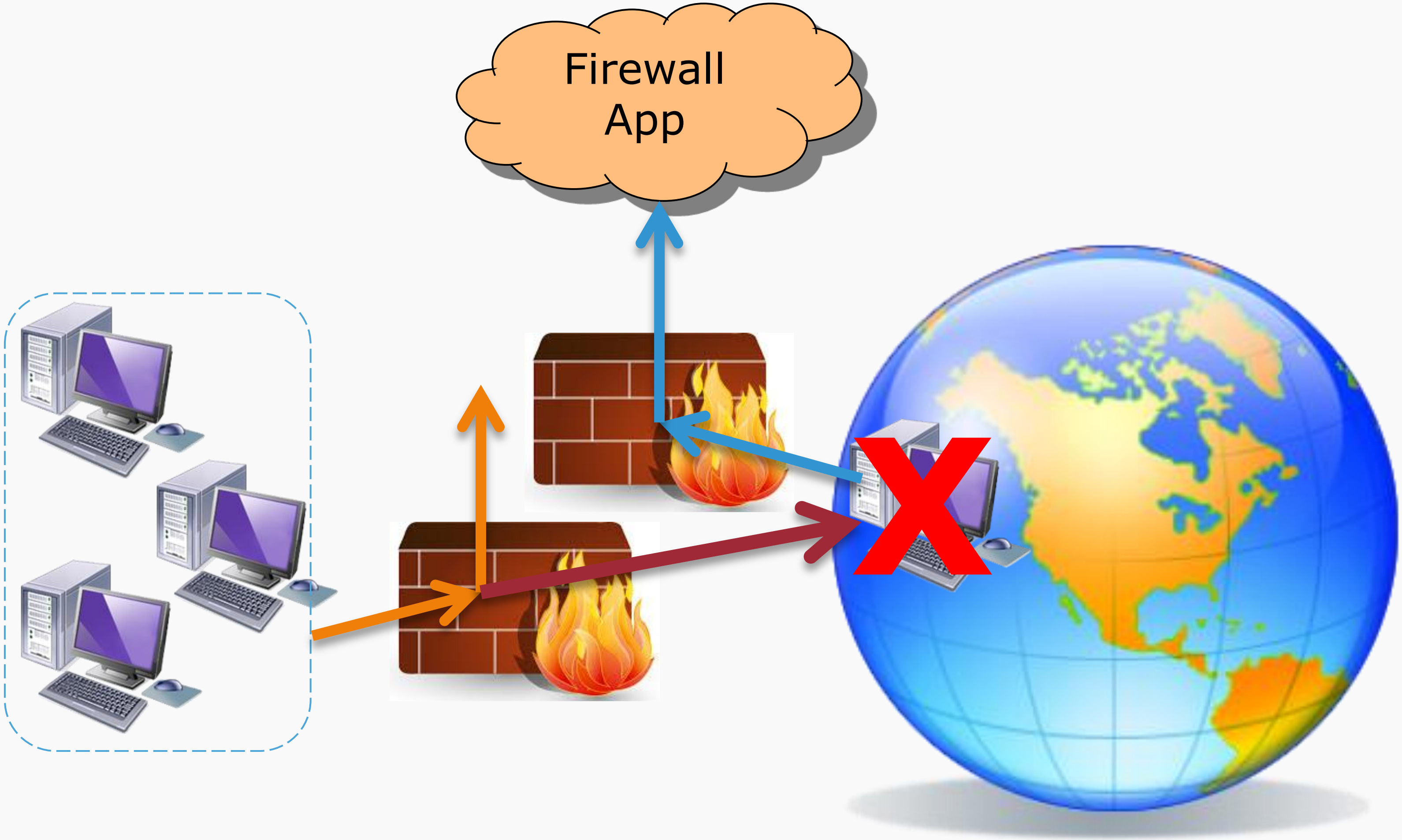
App	Virtualization technique	Incorrect-behavior
Stateful firewall	One-to-many mapping	Blacklisting the legitimate hosts
NAT	One-to-many mapping	Dropping requested packets
Load-balancer	One-to-many mapping	Overloading some servers and leaving some underutilized
Firewall & router	Many-to-one mapping	Blacklisting the legitimate hosts

Related work

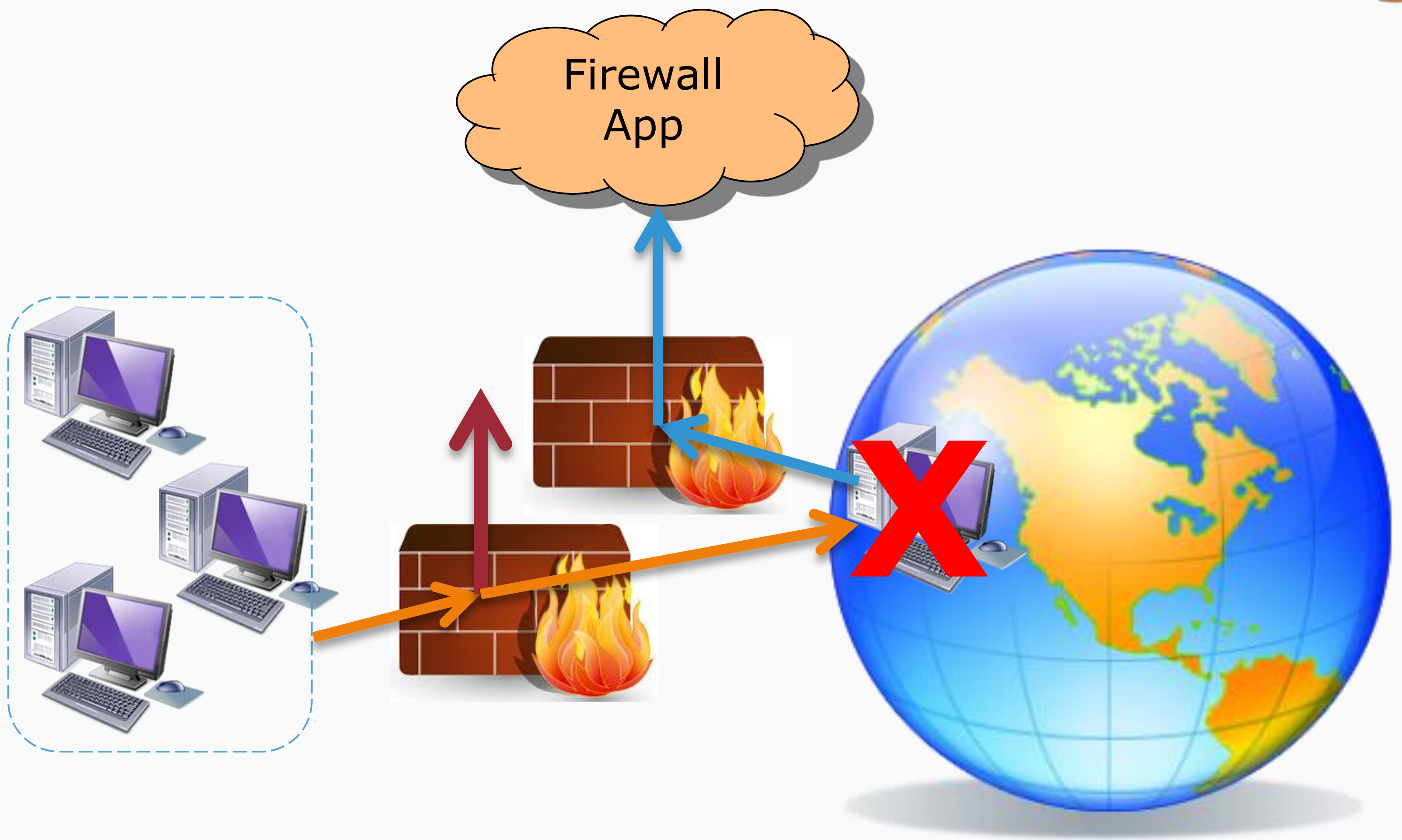
- Incorrect behavior caused by moving, observed in:
    1. *"LIME: Transparent, Live Migration of a Software-Defined Network"*, Soudeh Ghorbani, Cole Schlesinger, Matthew Monaco, Eric Keller, Matthew Caesar, Jennifer Rexford, David Walker, under submission.
    2. *"OpenNF: Enabling Innovation in Network Function Control"*, Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, Aditya Akella, SIGCOMM 2014.
- These existing solutions are:
    - Only a short-term fix while virtual network is being moved.
    - Infeasible when incorrect behavior is permanent rather than transient.

Root-cause of the incorrect behavior

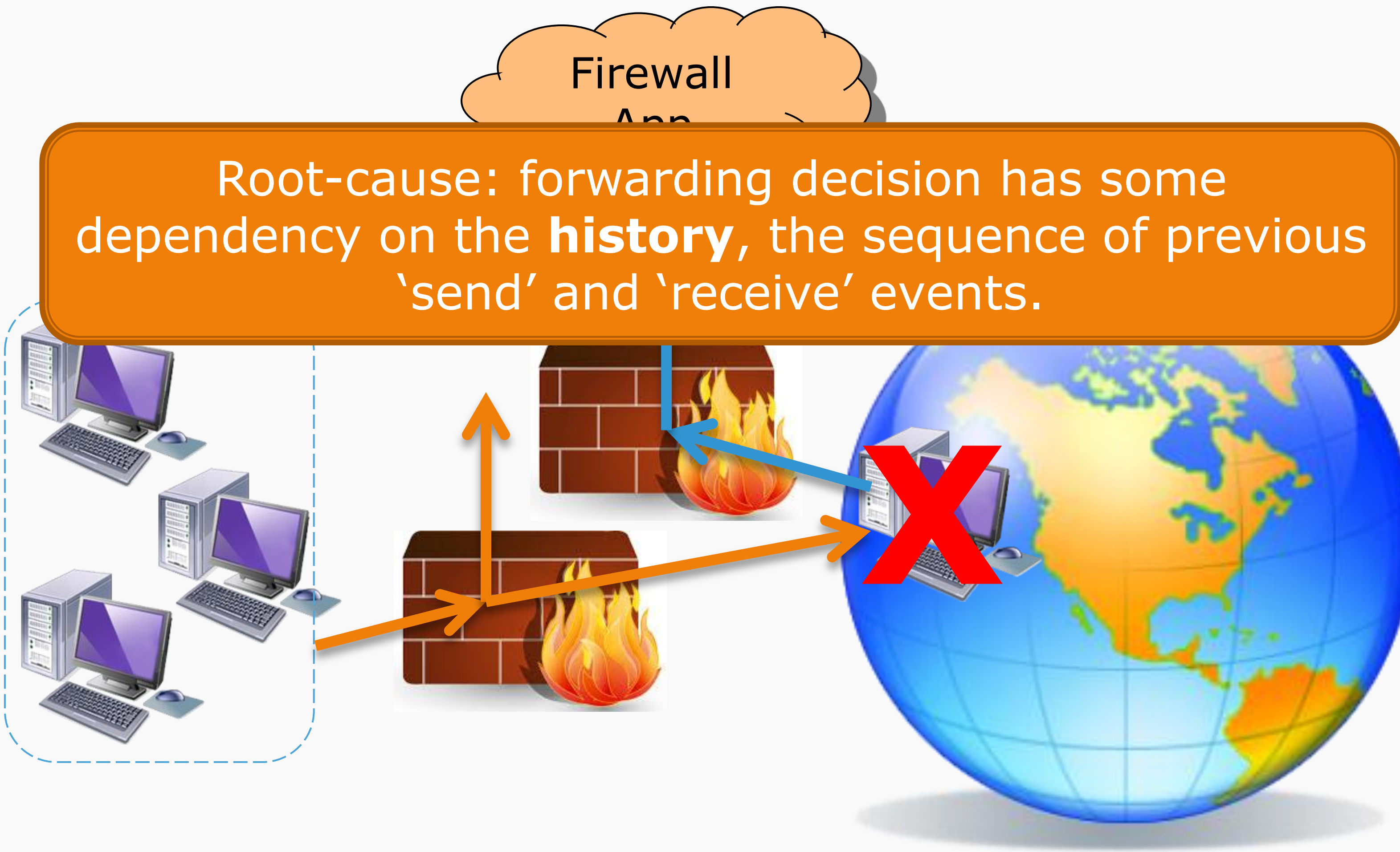
Firewall + virtualization = bug



Firewall + virtualization = bug



Firewall + virtualization = bug



Root-cause: forwarding decision has some dependency on the **history**, the sequence of previous 'send' and 'receive' events.

Who programs the network?

- The entities that can make or influence the forwarding decisions:
  - **Controller**
  - **Switch**: random forwarding like ECMP
  - **Data packet**: indirectly through local state, e.g., idle-timers

Who programs the network?

- The entities that can make or influence the forwarding decisions:
  - **Controller**
  - **Switch**: random forwarding like ECMP
  - **Data packet**: indirectly through local state, e.g., idle-timers

Can existing correctness definitions detect the incorrect behavior?

Correctness conditions:

1. Per-packet/flow consistency: prevents loops, black-holes,…

   Consensus Routing [NSDI'08], Consistent Updates [SIGCOMM'12]

2. Congestion freedom

   zUpdates [SIGCOMM'13], SWAN [SIGCOMM'13], On Consistent Updates in Software-Defined Networks [HotNets'13]

Can existing correctness definitions detect the incorrect behavior?

1 **None** of these conditions were violated in our examples!

1. Per-packet/flow consistency: prevents loops, black-holes,…
   Consensus Routing [NSDI’08], Consistent Updates [SIGCOMM’12]

2. Congestion freedom
   zUpdates [SIGCOMM’13], SWAN [SIGCOMM’13], On Consistent Updates in Software-Defined Networks [HotNets’13]

Can existing correctness definitions detect the incorrect behavior?

1 **None** of these conditions were violated in our examples!

2 "Correctness is what users want."
Leslie Lamport

Co

1. Per packet/flow consistency: prevents loops,

es

[SIGCOMM'12]

2. Congestion freedom

zUpdates [SIGCOMM'13], SWAN [SIGCOMM'13], On Consistent Updates in Software-Defined Networks [HotNets'13]

Can existing correctness definitions detect the incorrect behavior?

1 **None** of these conditions were violated in our examples!

2 "Correctness is what users want."
Leslie Lamport

[SIGCOMM'12]

3 Techniques designed to preserve those correctness conditions could **break the otherwise correct behavior**.

On Consistent Updates in Software-Defined Networks [HotNets'13]

Can existing correctness definitions detect the incorrect behavior?

1 **None** of these conditions were violated in our examples!

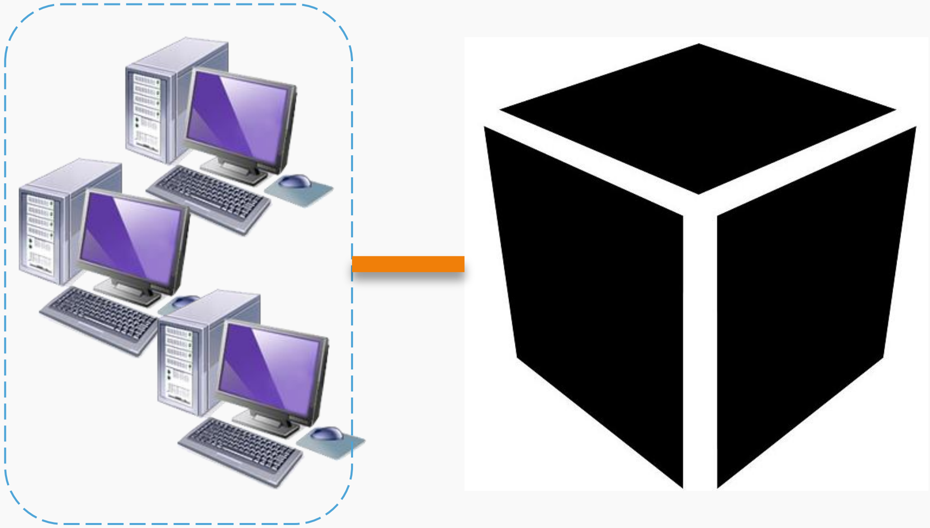
2 “Correctness is what users want.”
Leslie Lamport

[SIGCOMM’12]

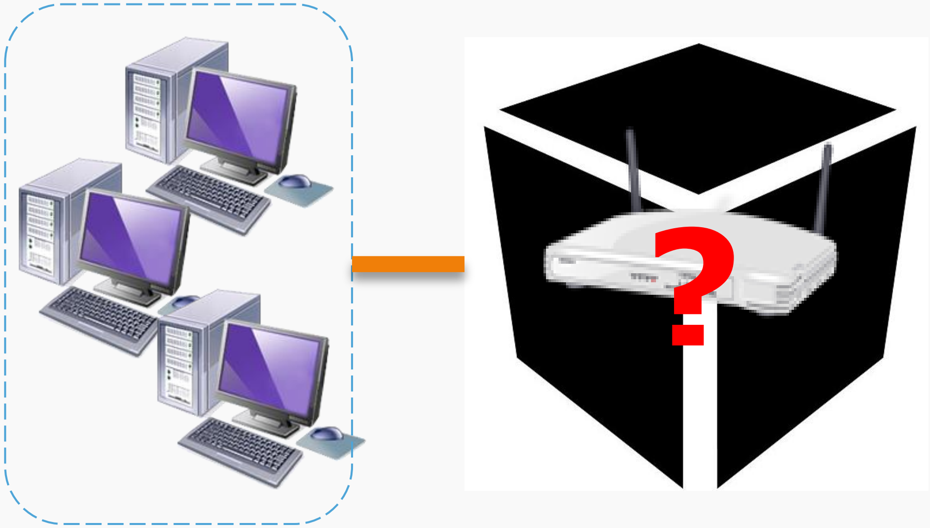
3 Techniques designed to preserve those correctness conditions could **break the otherwise correct behavior**.

4 We need **new definitions of correctness** and **new techniques** to achieve those.

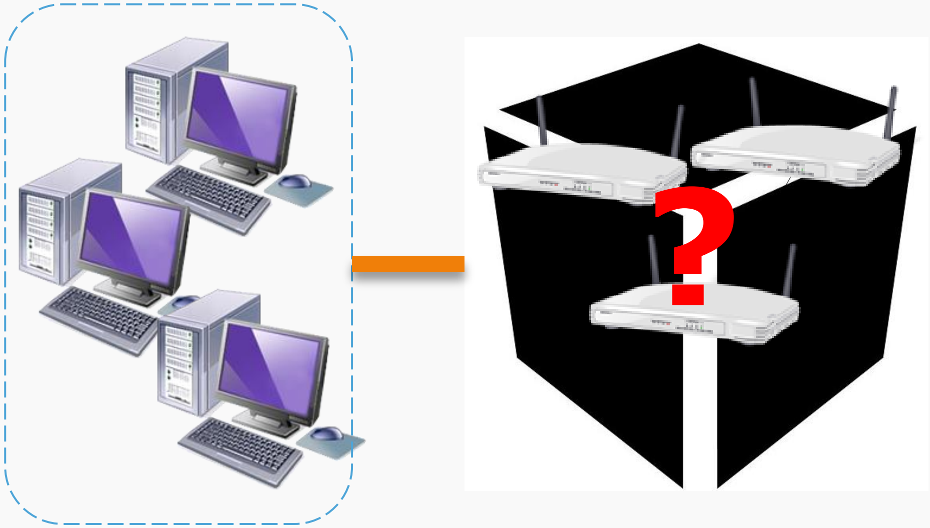
A new correctness condition: End-to-end correctness



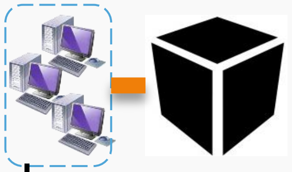
A new correctness condition: End-to-end correctness



A new correctness condition: End-to-end correctness

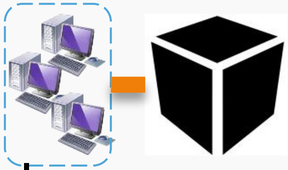


A new correctness condition: End-to-end correctness



- A mapping of a logical network L to a physical network P is said to be **end-to-end correct** *iff* $Pr_L[E] \approx Pr_P[E]$ where E is the partially ordered set of 'send' and 'receive' events.

A new correctness condition: End-to-end correctness



- A mapping of a logical network L to a physical network P is said to be **end-to-end correct** *iff* $Pr_L[E] \approx Pr_P[E]$ where E is the partially ordered set of 'send' and 'receive' events.

- Key features:
  - distinguishes between events that happen **always**, **sometimes**, and **never**.

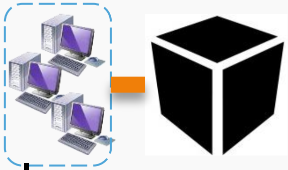
A new correctness condition: End-to-end correctness

- A mapping of a logical network L to a physical network P is said to be **end-to-end correct** *iff* $Pr_L[E] \approx Pr_P[E]$ where E is the partially ordered set of 'send' and 'receive' events.

- Key features:
  - distinguishes between events that happen **always**, **sometimes**, and **never**.

  - permissive of the **differences in packet loss or timing** that do not affect correctness.

A new correctness condition: End-to-end correctness



- A mapping of a logical network L to a physical network P is said to be **end-to-end correct** *iff* $Pr_L[E] \approx Pr_P[E]$ where E is the partially ordered set of 'send' and 'receive' events.

- Key features:
  - distinguishes between events that happen **always**, **sometimes**, and **never**.
  - permissive of the **differences in packet loss or timing** that do not affect correctness.
  - permissive of the **legitimate differences in orderings** of events.

So far:

1 We identified the **problem:** incorrect application-level behavior under the existing virtualization techniques.

2 We identified its **root-cause:** dependence on the history.

3 We developed an analytical **framework** to reason about the problem.

Research Vision:

Developing a general **algorithm.** 4

Proving its **correctness.** 5

Developing a correct virtualization **System.** 6

Thanks!

Questions?