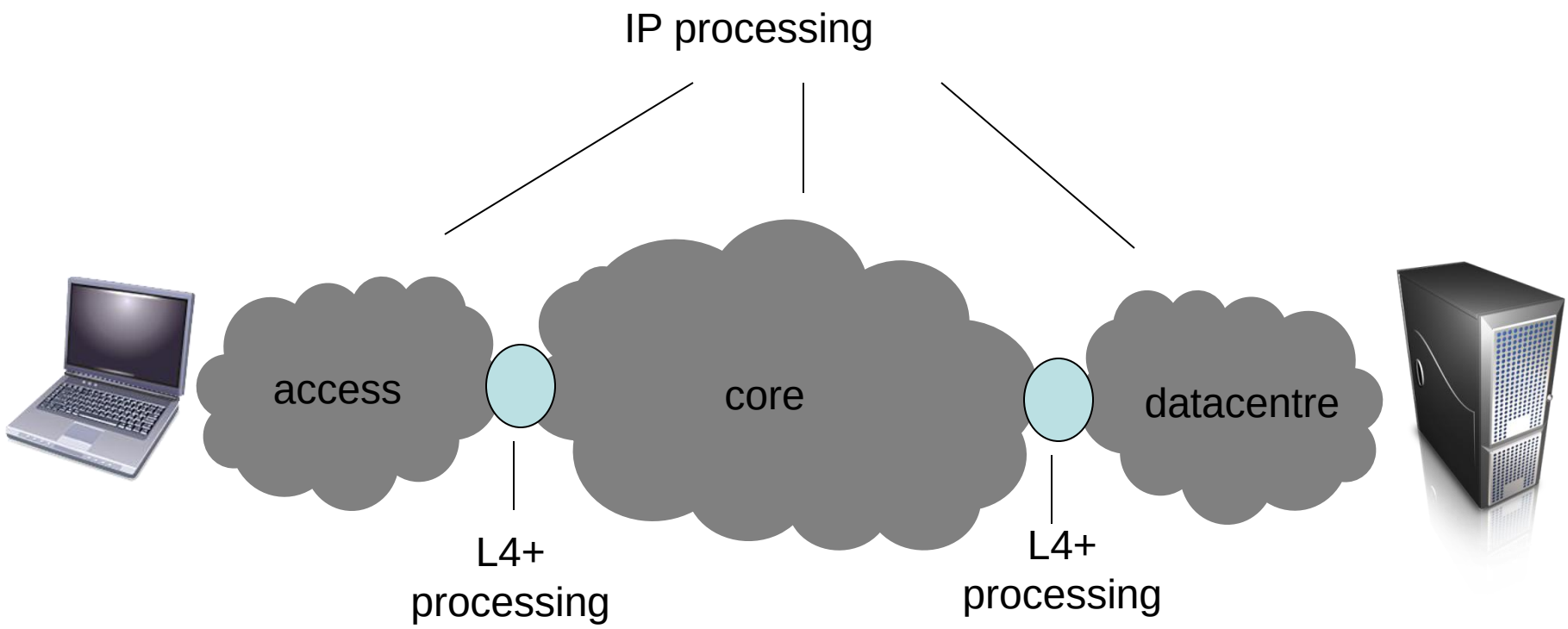
Flow Processing and the Rise of Commodity Network Hardware

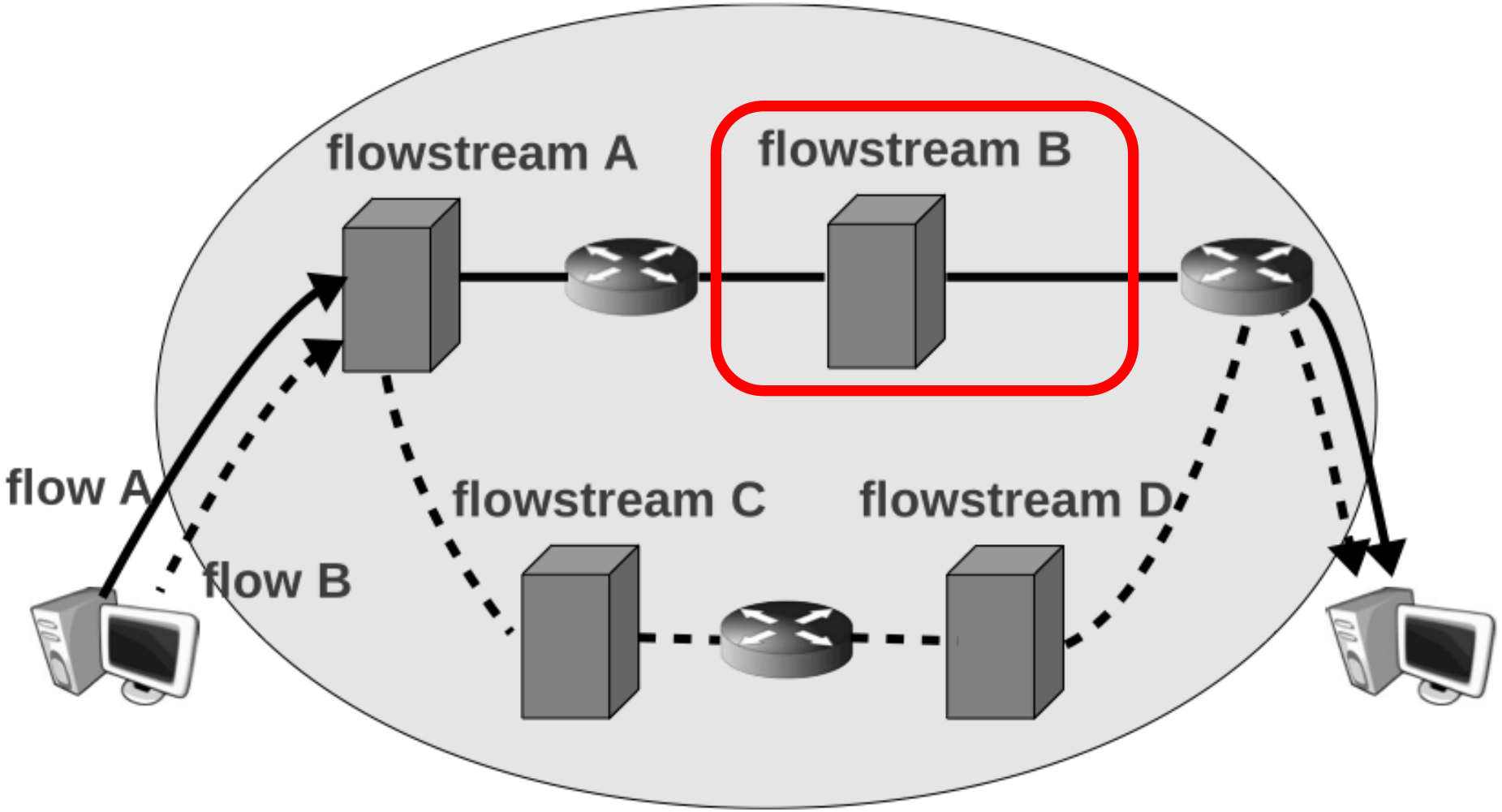
felipe.huici@neclab.eu

A Segmented Internet



It already looks somewhat like this, but the L4+ processing is more distributed.

Empowering both the ends and the middle



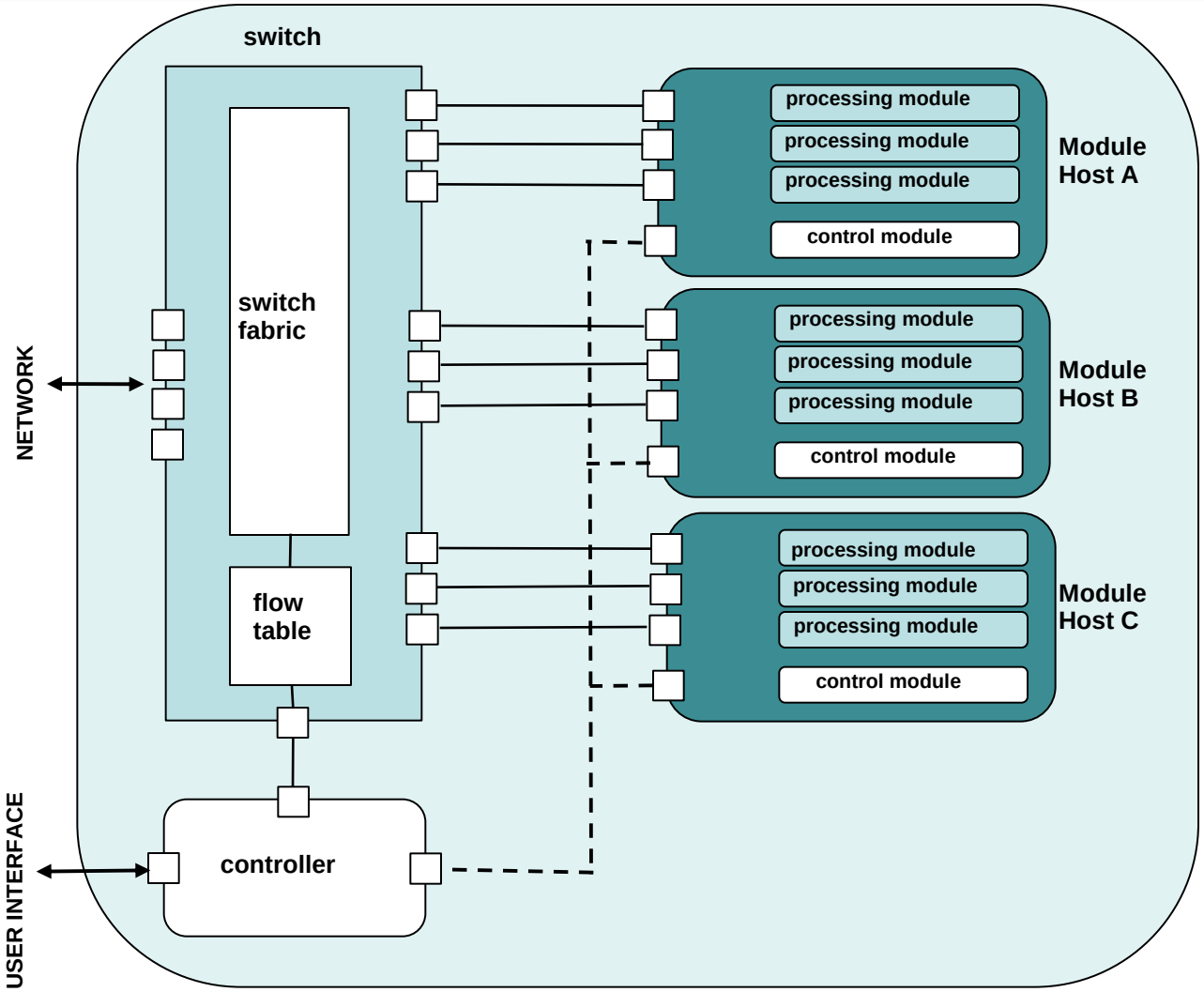
Outline

- **Flowstream - Flow Processing Platform**
- **ClickOS – Tiny virtual machine for flow processing**

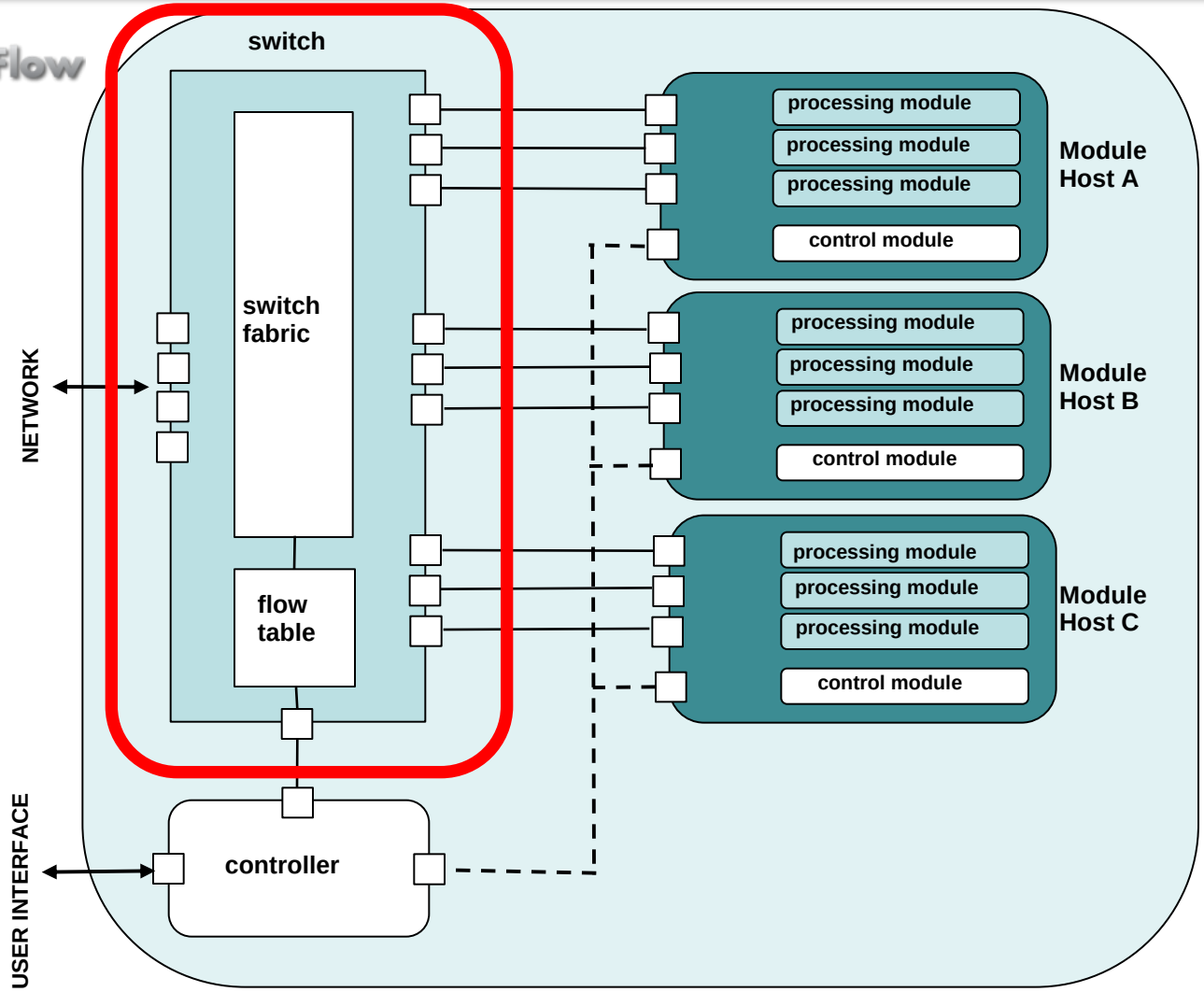
A Platform for Change

- **Those L4+ platforms need to be more general that today's middleboxes.**
  - More open.
  - More upgradable, as new apps arrive.
  - Aggregate functionality, so it's manageable.
  - Cheap and scalable.
    - » Exploit commoditisation of 1U severs and switches.
    - » Change the middlebox market to one of software rather than appliances.
    - » Reduce operator costs.
    - » Increase reliability.
    - » Enable innovation.

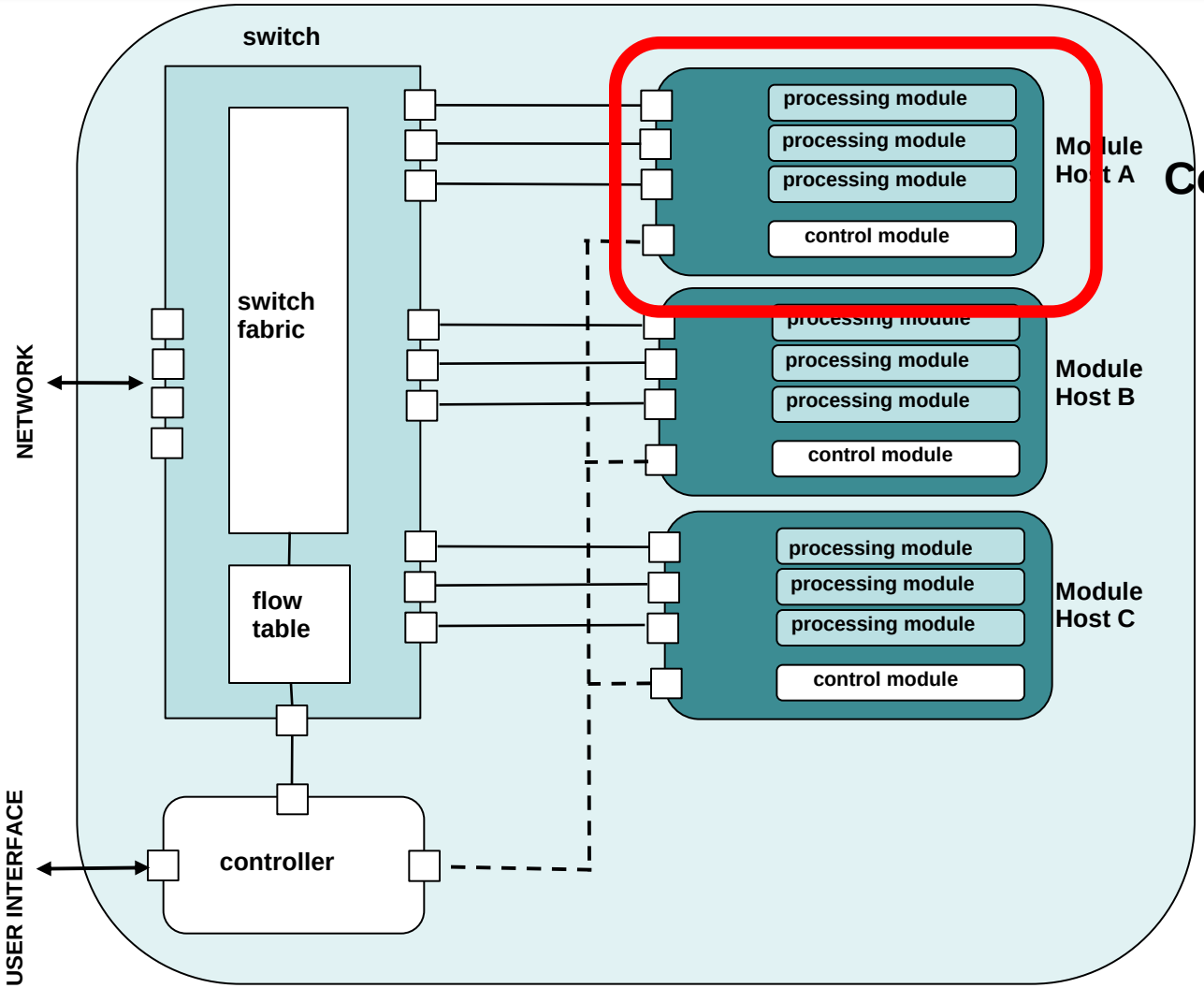
Flowstream Platform



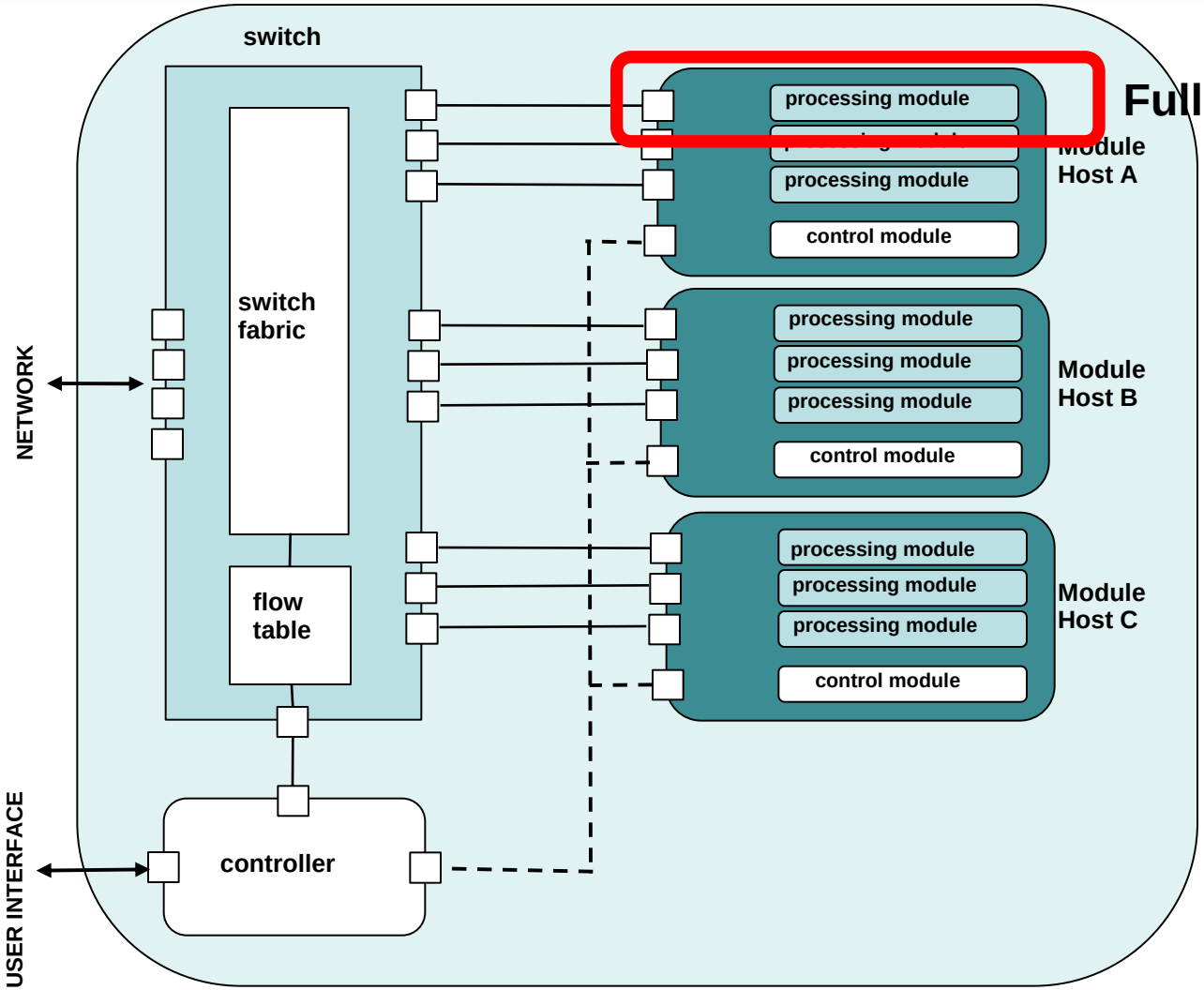
Flowstream: components



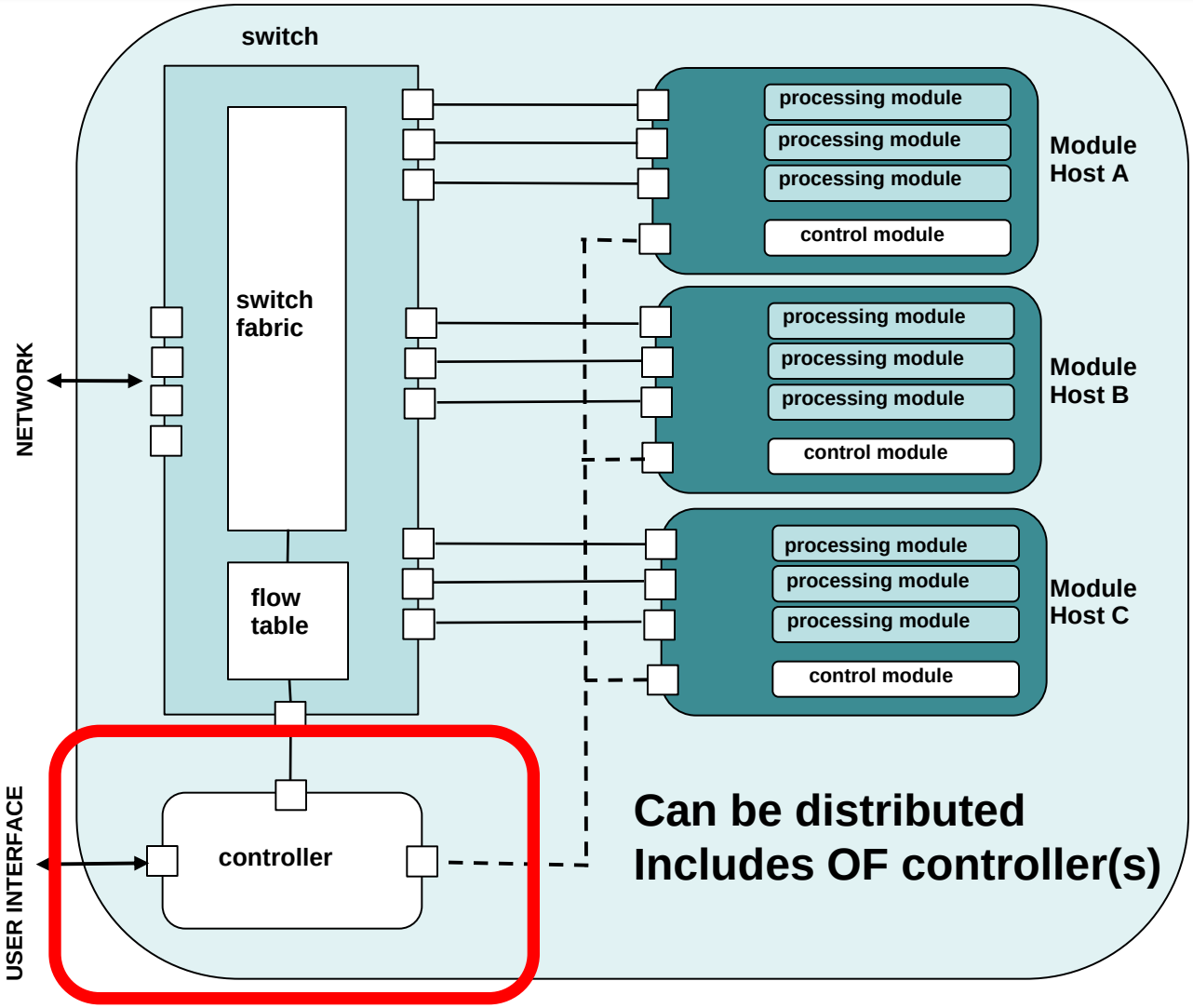
Flowstream: components



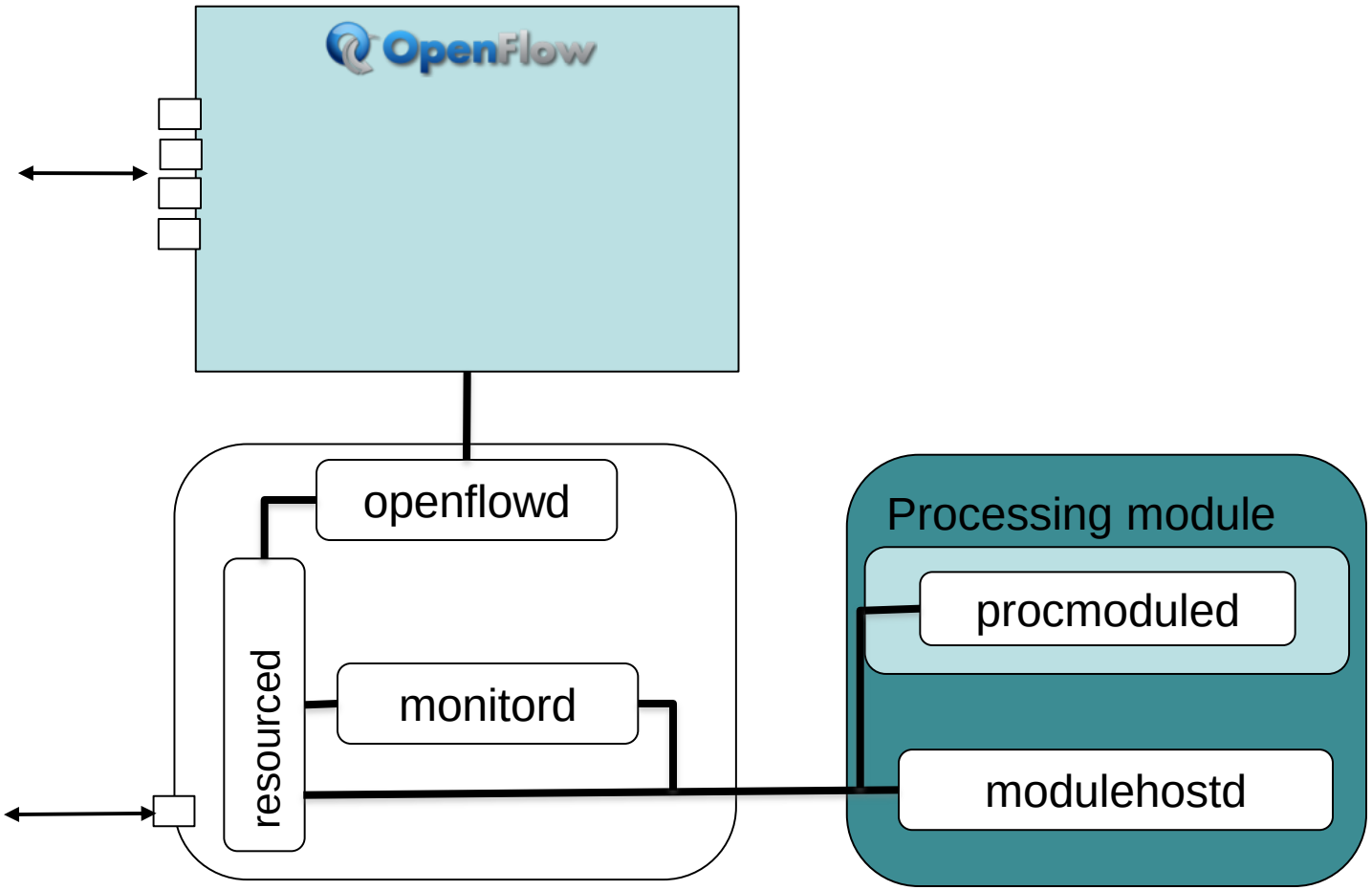
Flowstream: components



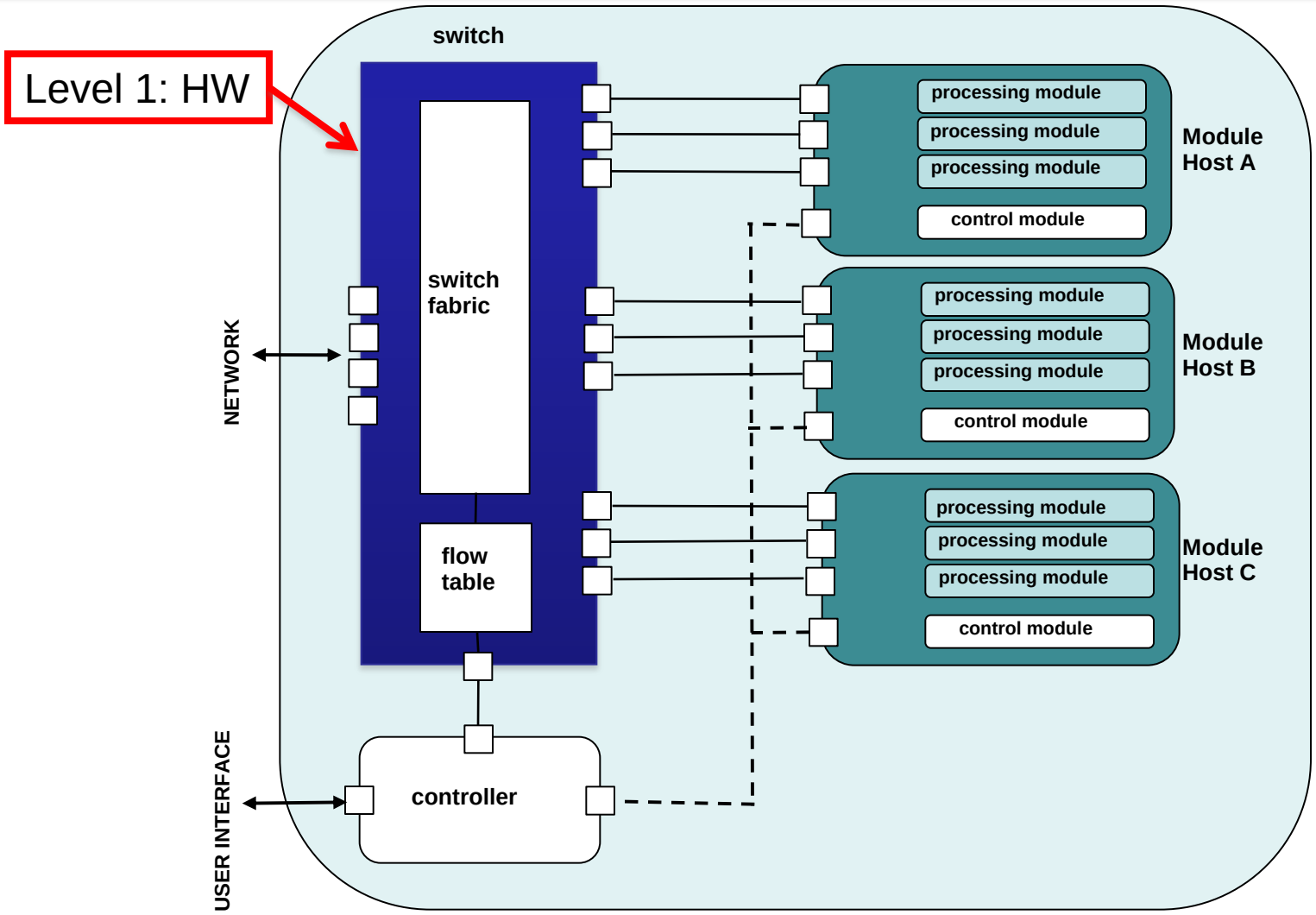
Flowstream: components



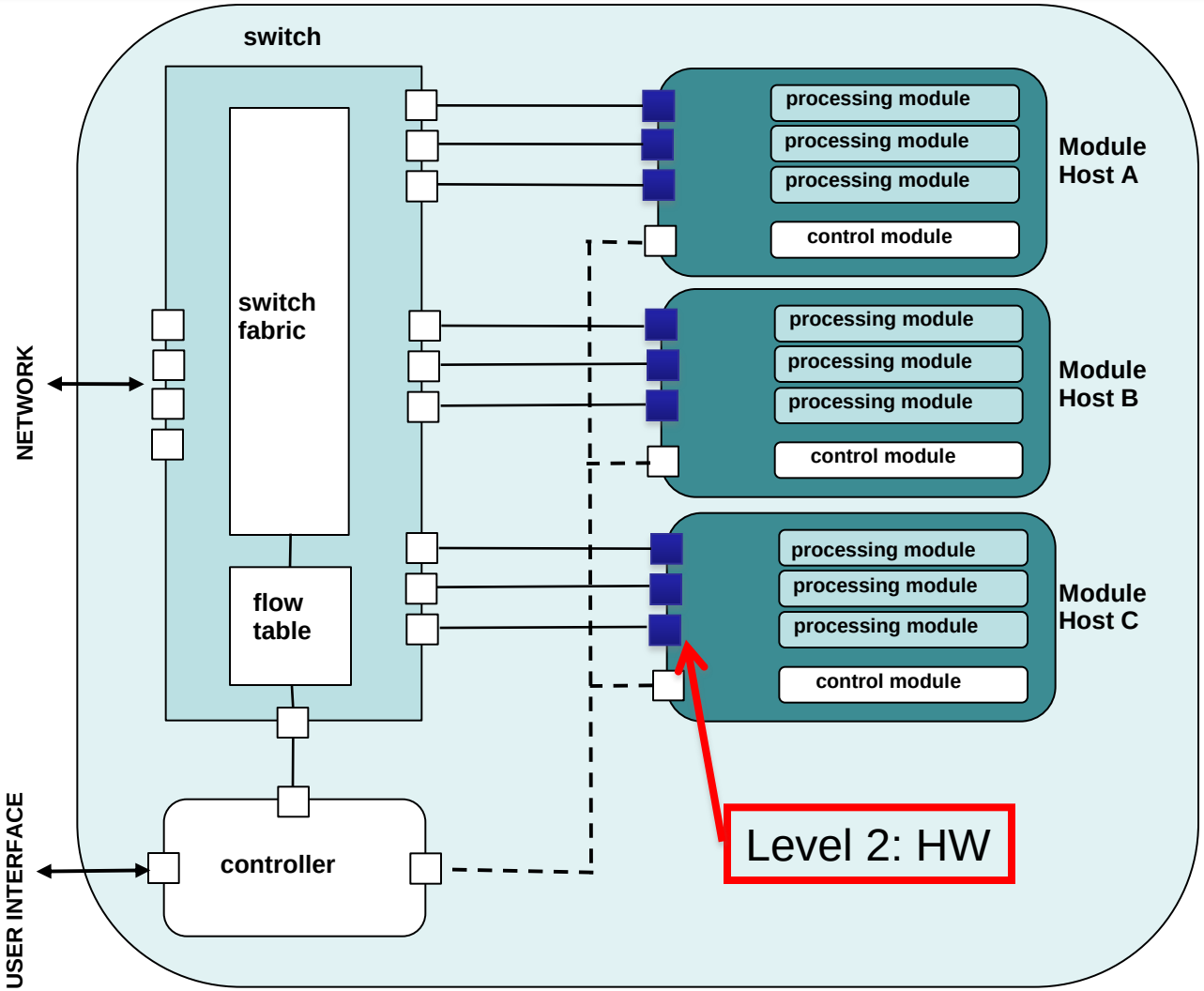
Components: Controller



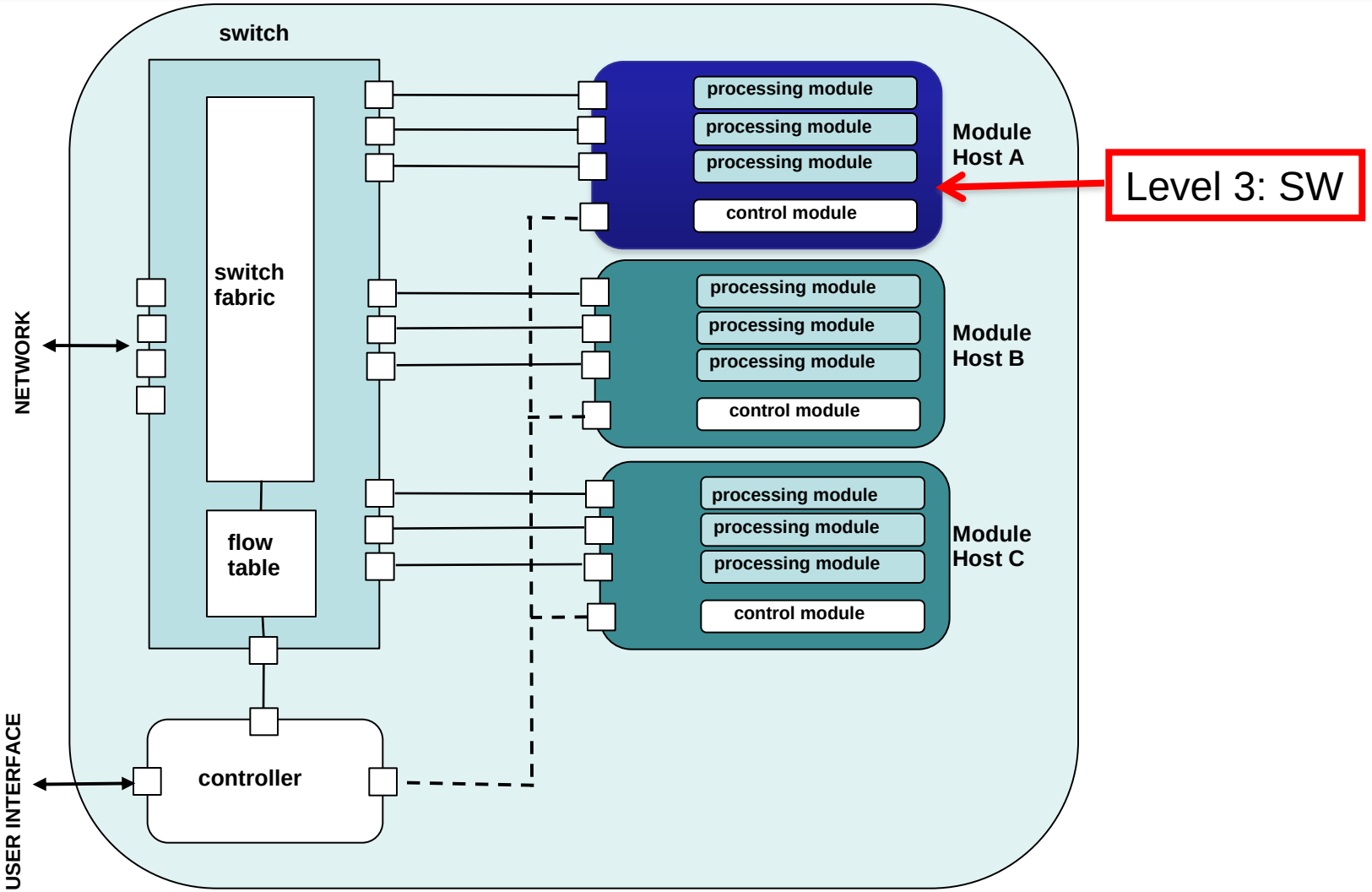
Classification



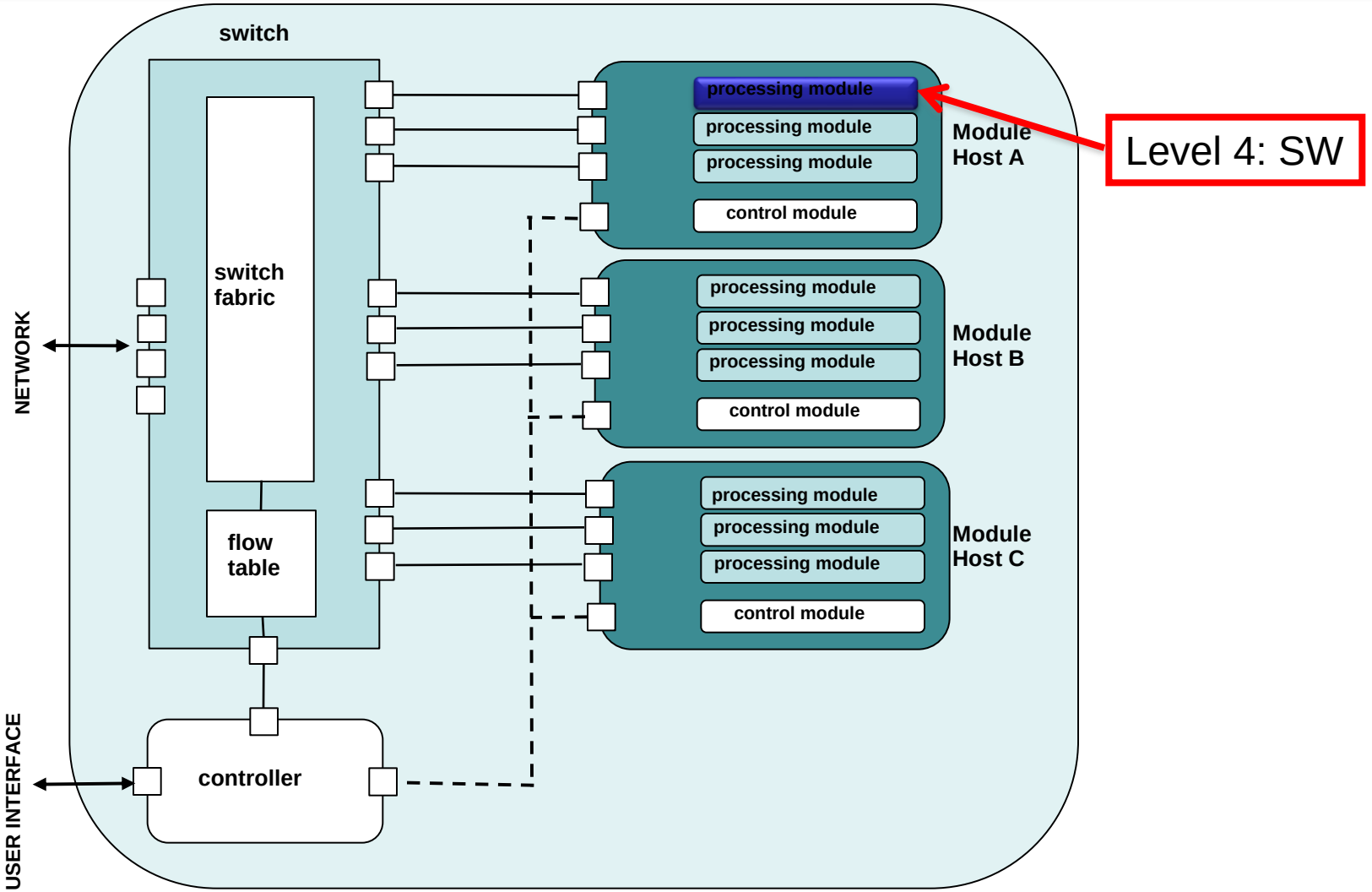
Classification



Classification

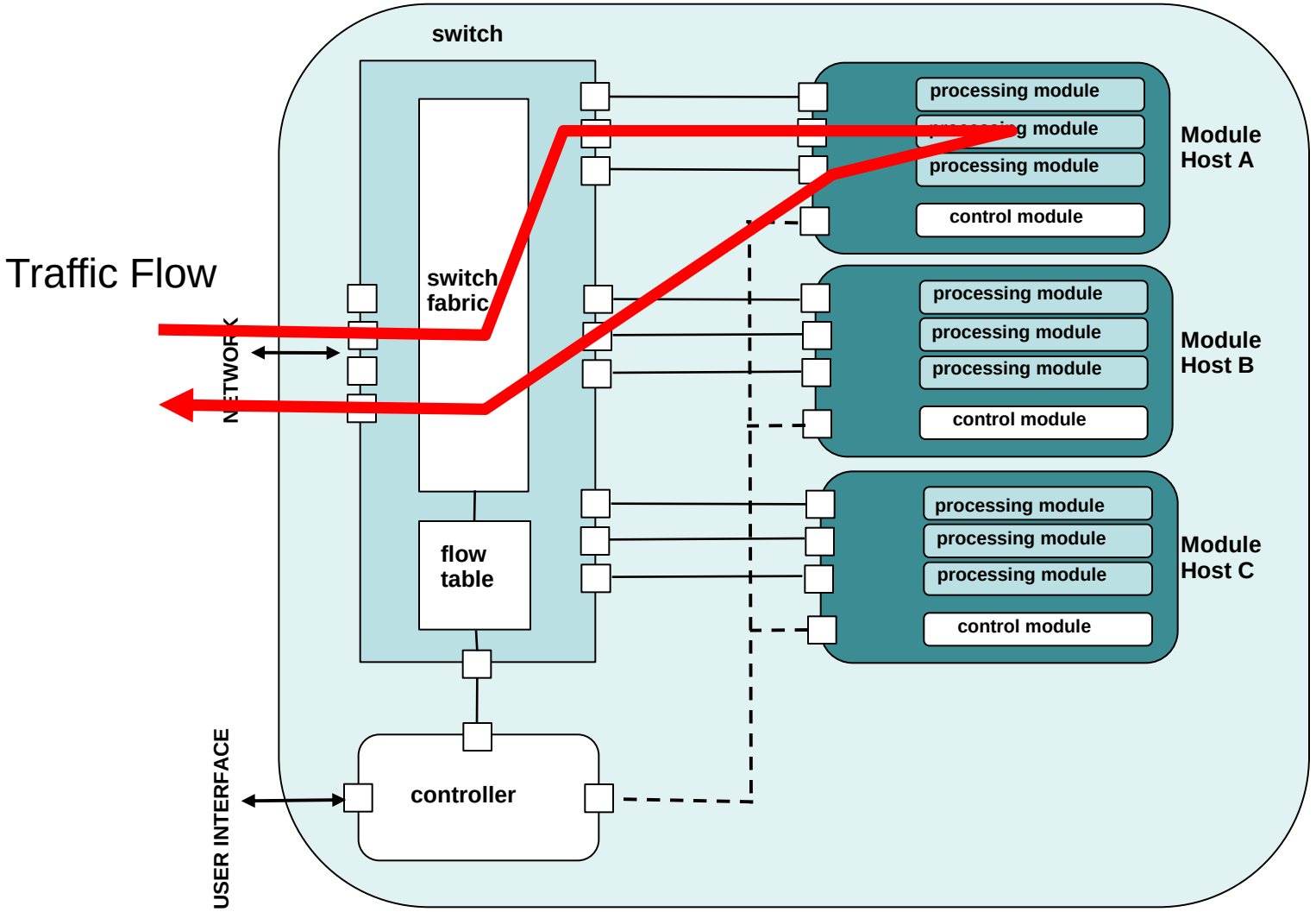


Classification



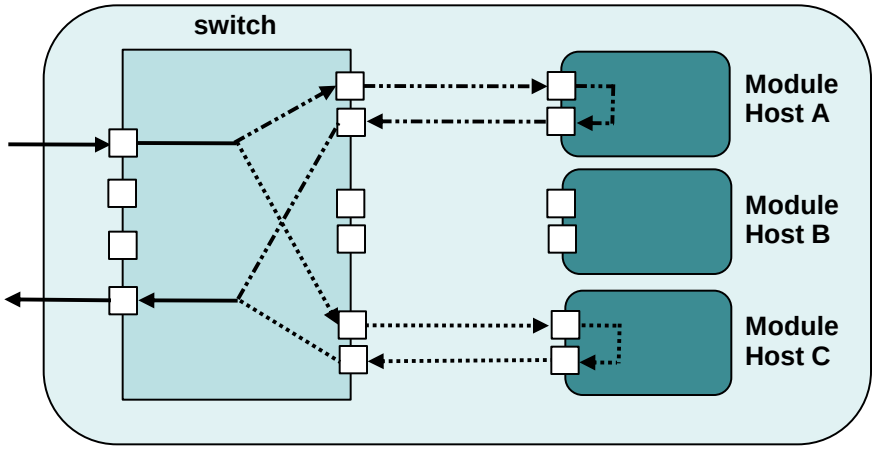
Data Paths

- Basic

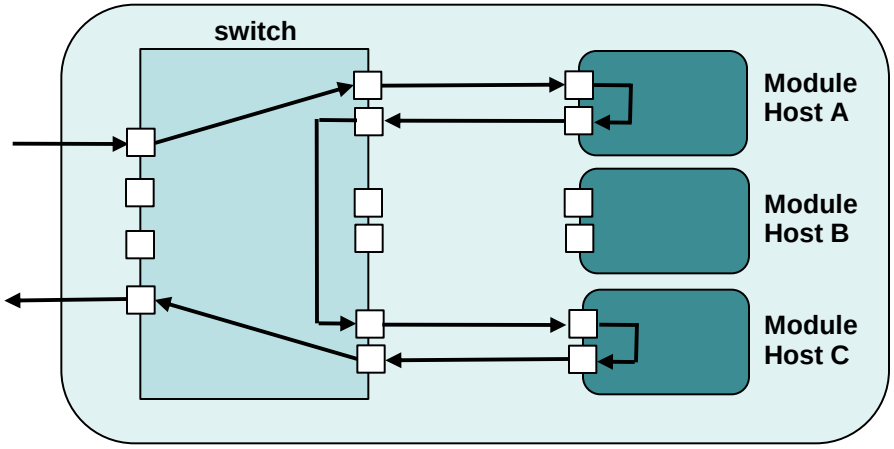


Data Paths

- Parallel Processing

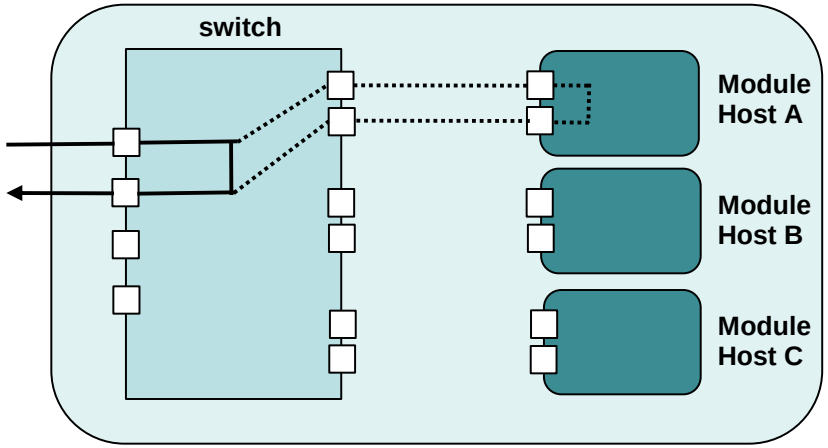


- Serial Processing

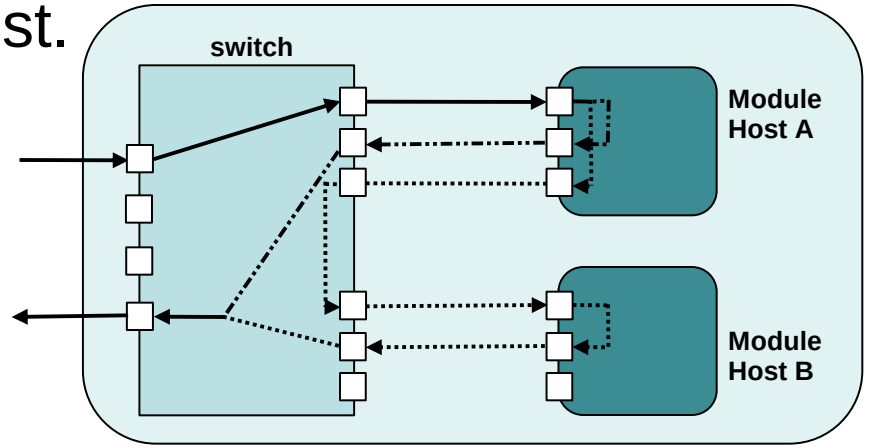


Data Paths

- Offload processing to the switch.

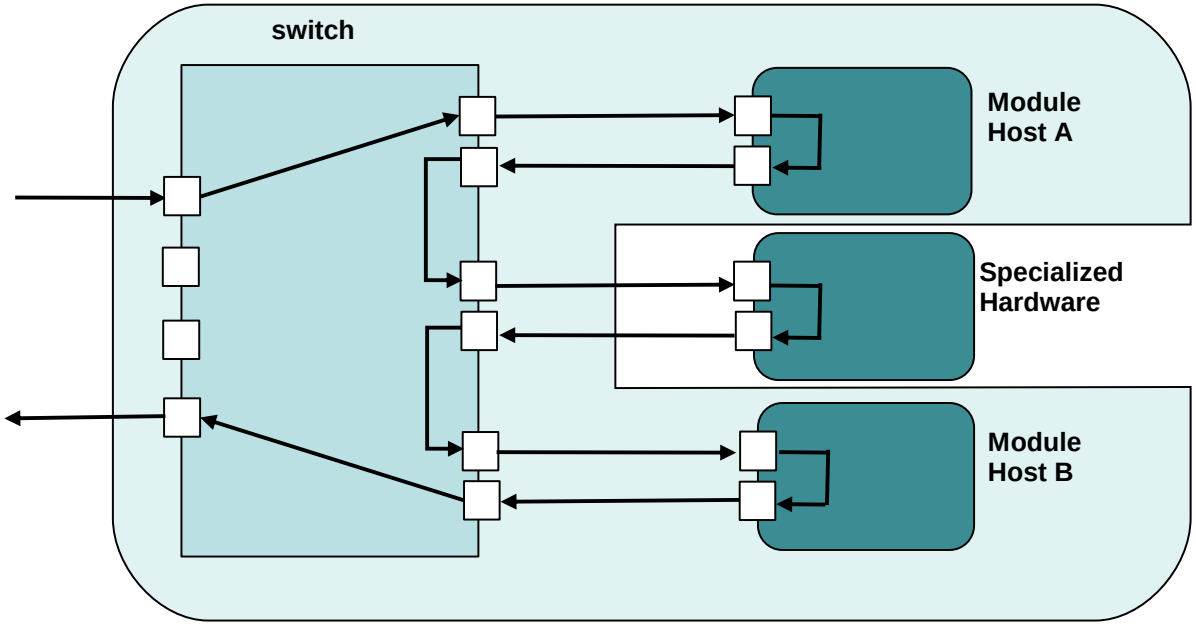


- Offload processing to another host.



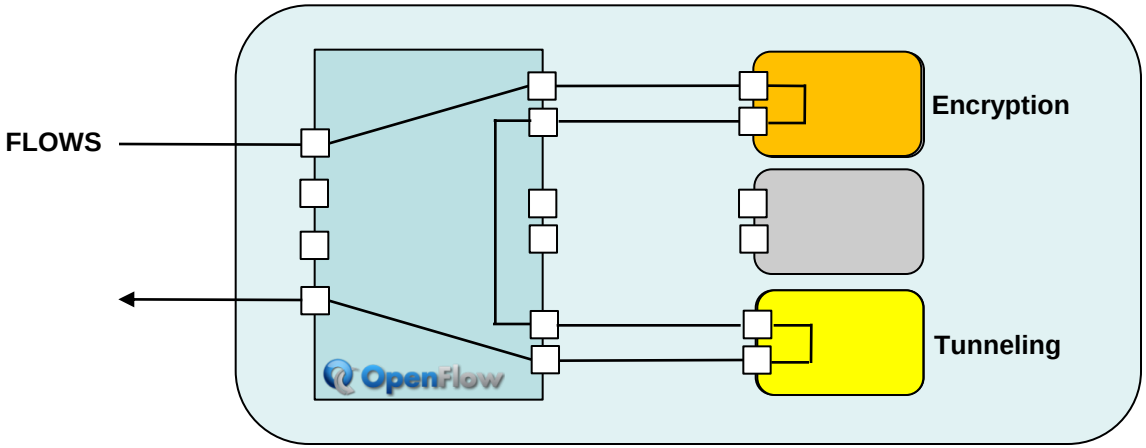
Data Paths

- Third party middleboxes
  - Operators invested heavily already.
  - Some hardware is specialised.



Flowstream: Sample Usage

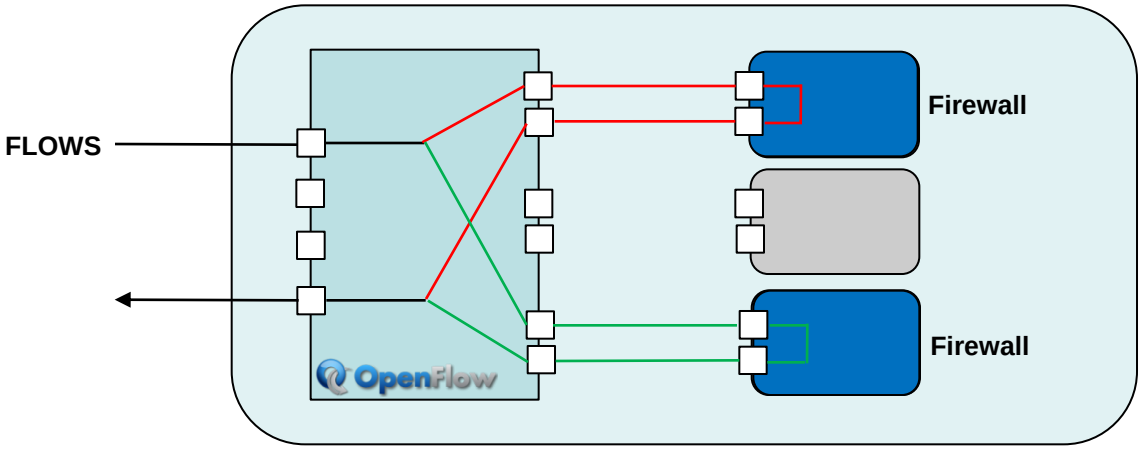
Serial processing: VPN Tunneling



- **Serial processing (VPN tunnel)**
  - Offloads expensive encryption processing to a single module host

Flowstream: Sample Usage

Load Balancing Scenario: Firewalls

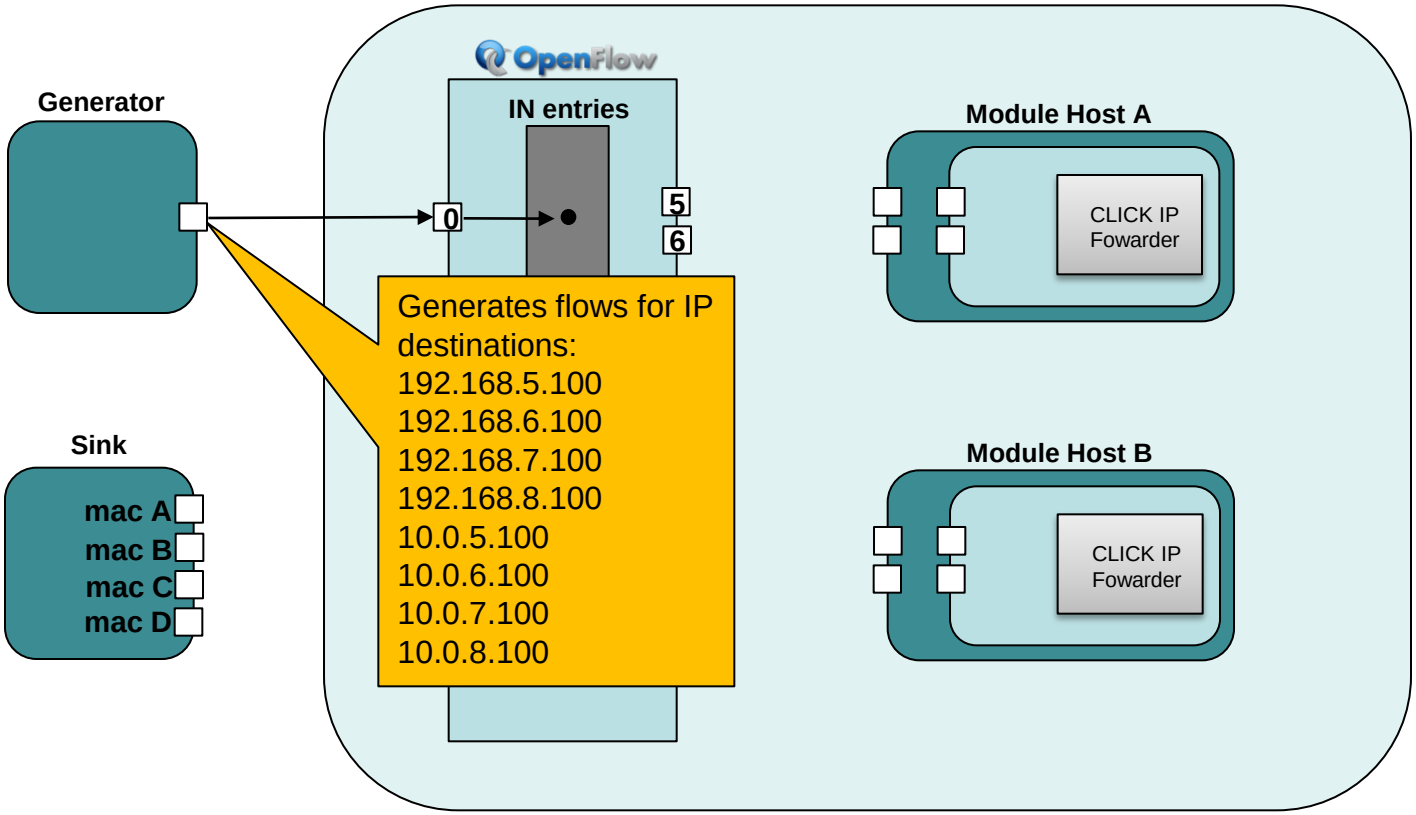


- The controller splits flows (for instance based on IP prefix) to two different module hosts
- Flows are processed in parallel
- Flows are then merged back together at OpenFlow switch

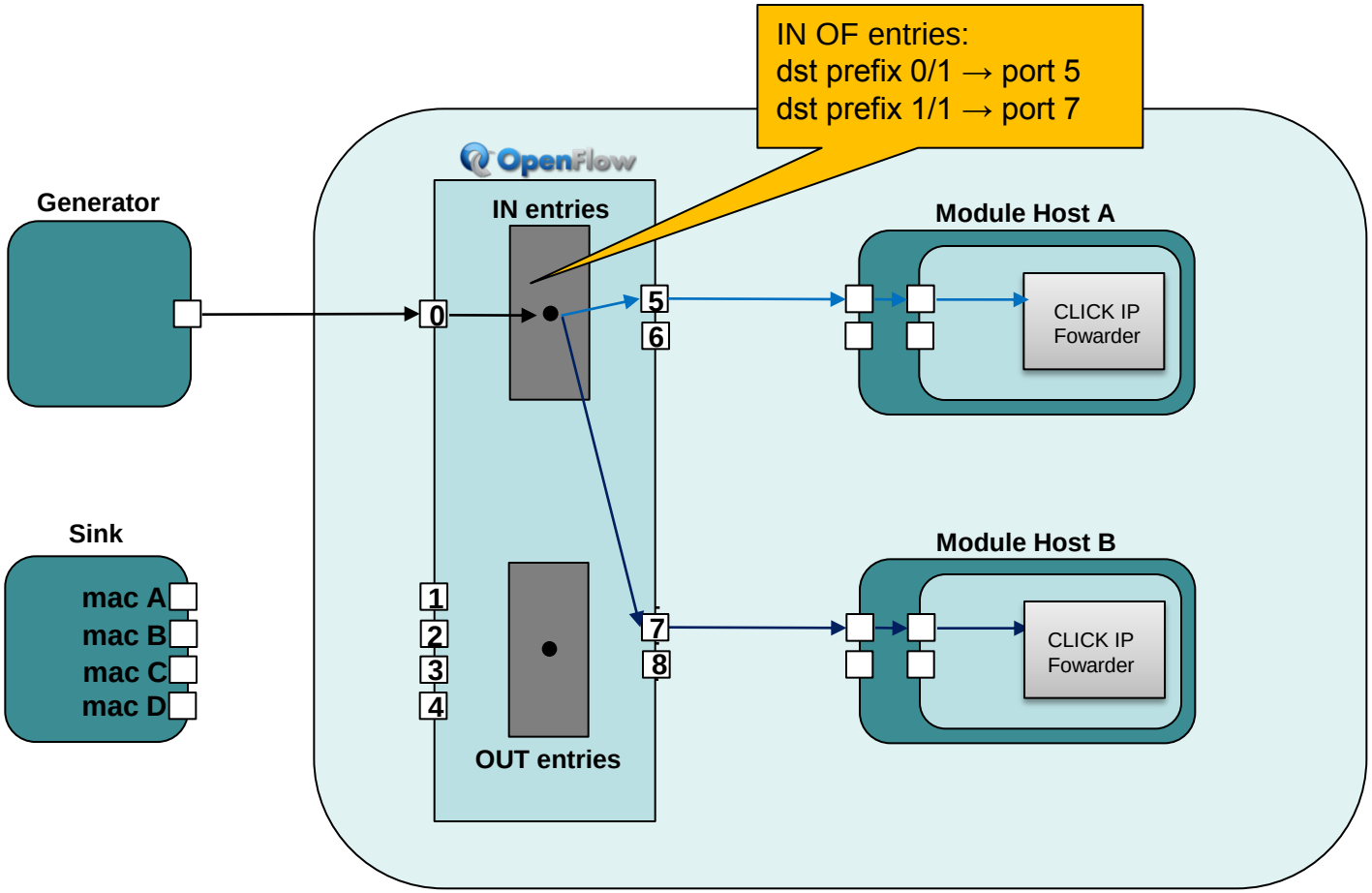
Proof-of-Concept: Load-Balancing IP Router

- **Uses**
  - Xen dom0 as module host
  - Linux guest domains with Click as processing modules
- **Destination-IP based load balancing, e.g.**
  - Prefix 0/1 → module host 1
  - Prefix 1/1 → module host 2
- **IP forwarding table can also be split across servers**
  - i.e., module host 1 only needs entries for 0/1 prefix

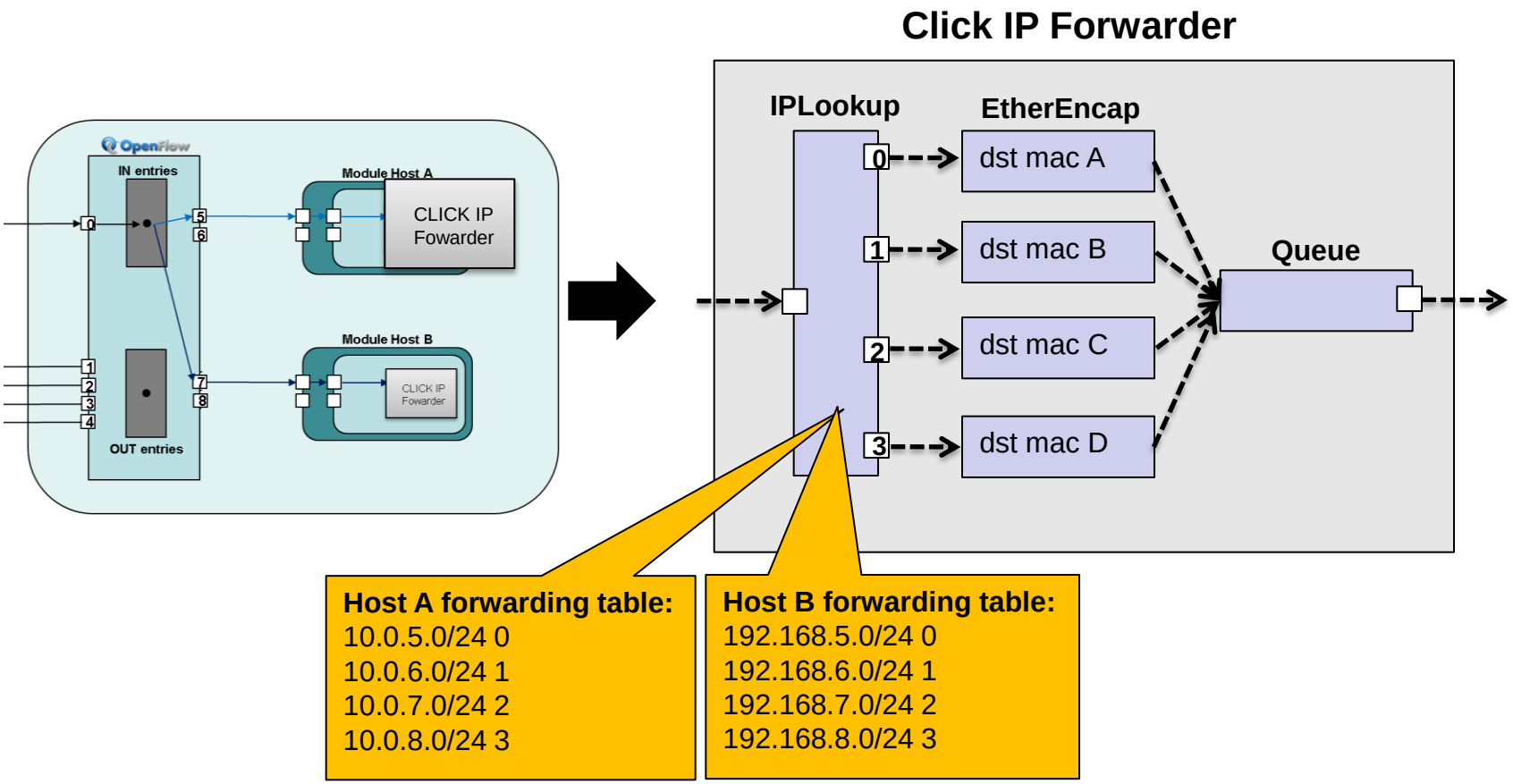
Proof-of-Concept: Load-Balancing IP Router



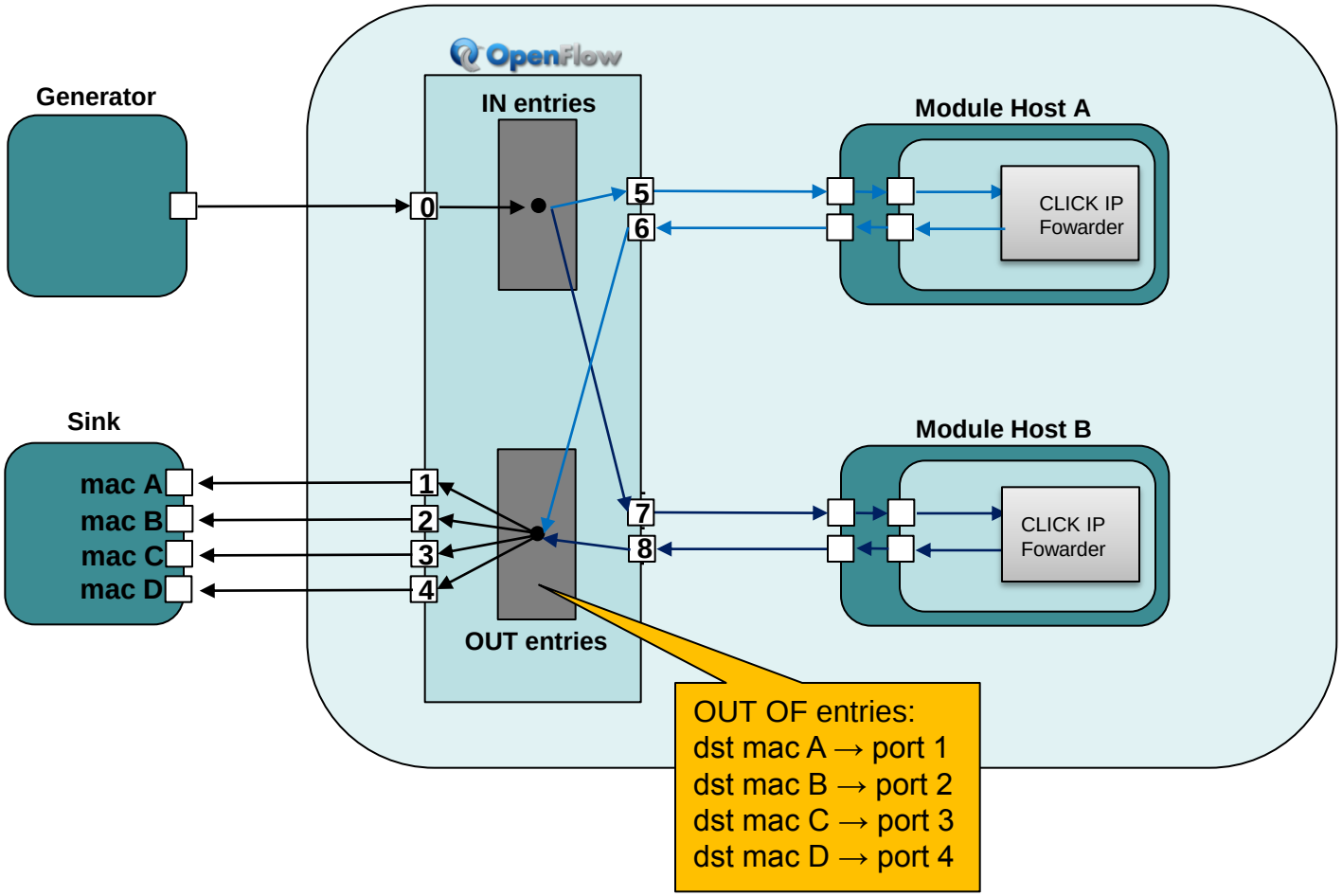
Proof-of-Concept: Load-Balancing IP Router



Proof-of-Concept: Load-Balancing IP Router



Proof-of-Concept: Load-Balancing IP Router



Input Format

```
<allocationdescription id="felipeiprouter" type="fastiprouter" subtype="prefixlb“>

 <portmacmappings>
  <portmacmapping portid="0" mac="00:15:17:15:5D:20" switchport="1"/>
  <portmacmapping portid="1" mac="00:15:17:15:5D:21" switchport=“2"/>
  <portmacmapping portid="2" mac="00:15:17:15:5B:C2" switchport=“3"/>
  <portmacmapping portid="3" mac="00:15:17:15:5B:C3" switchport=“4"/>
 </portmacmappings>

 <pm id="0">
  <prefix addr="128.0.0.0" mask="128.0.0.0"/>
  <forwardingtable id="0">
   <forwardingentry addr="192.168.5.0" mask="24" outport="0"/>
   <forwardingentry addr="192.168.6.0" mask="24" outport="1"/>
   <forwardingentry addr="192.168.7.0" mask="24" outport="2"/>
   <forwardingentry addr="192.168.8.0" mask="24" outport="3"/>
  </forwardingtable>
 </pm>

 <pm id="1">
  <prefix addr="0.0.0.0" mask="128.0.0.0"/>
  <forwardingtable id="0">
   <forwardingentry addr="10.0.5.0" mask="24" outport="0"/>
   <forwardingentry addr="10.0.6.0" mask="24" outport="1"/>
   <forwardingentry addr="10.0.7.0" mask="24" outport="2"/>
   <forwardingentry addr="10.0.8.0" mask="24" outport="3"/>
  </forwardingtable>
 </pm>

</allocationdescription>
```

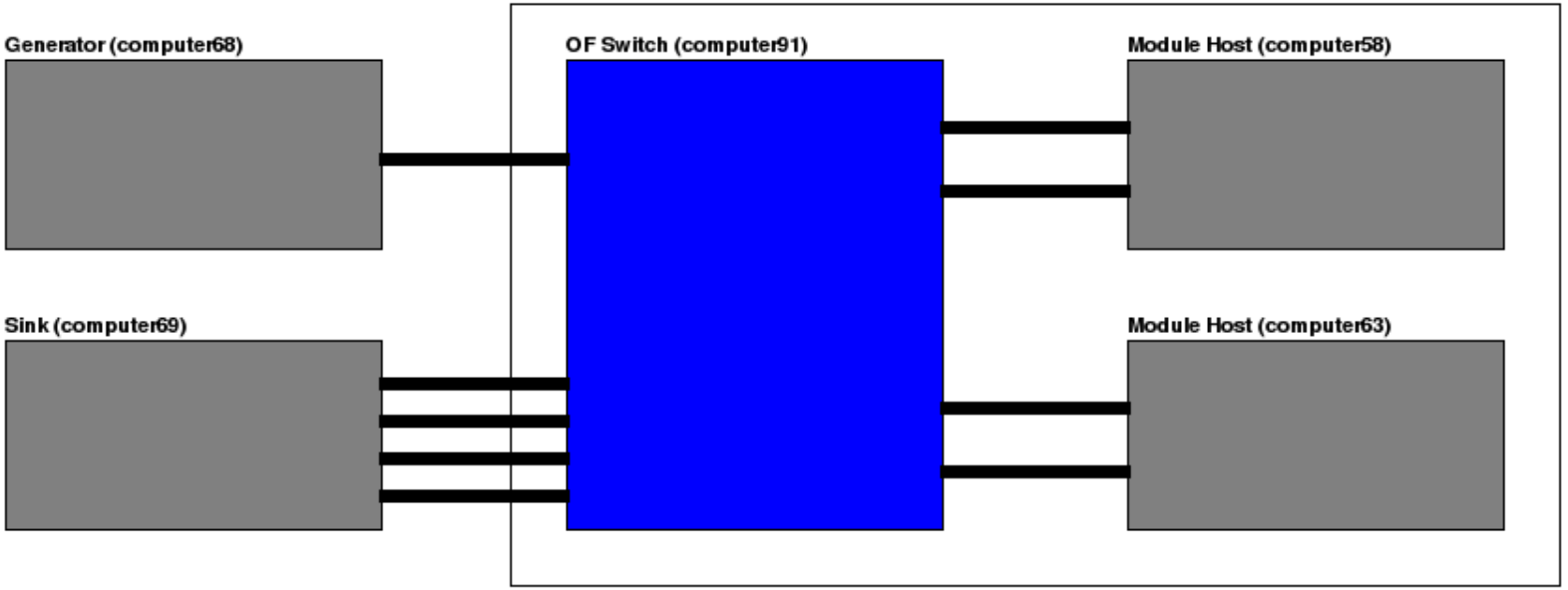
IP router load balanced by IP prefix

mappings of platform ports to external MAC addresses

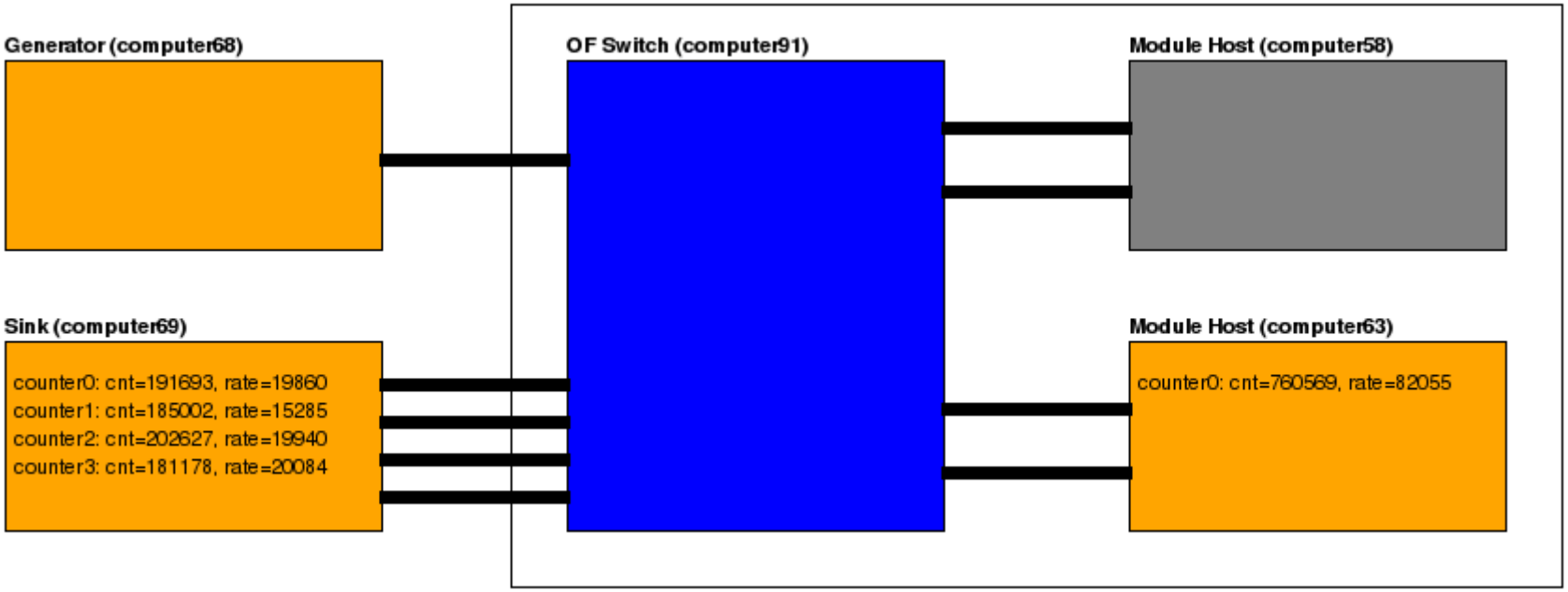
Forwarding specification for first host (module host A)

Forwarding specification for second host (module host B)

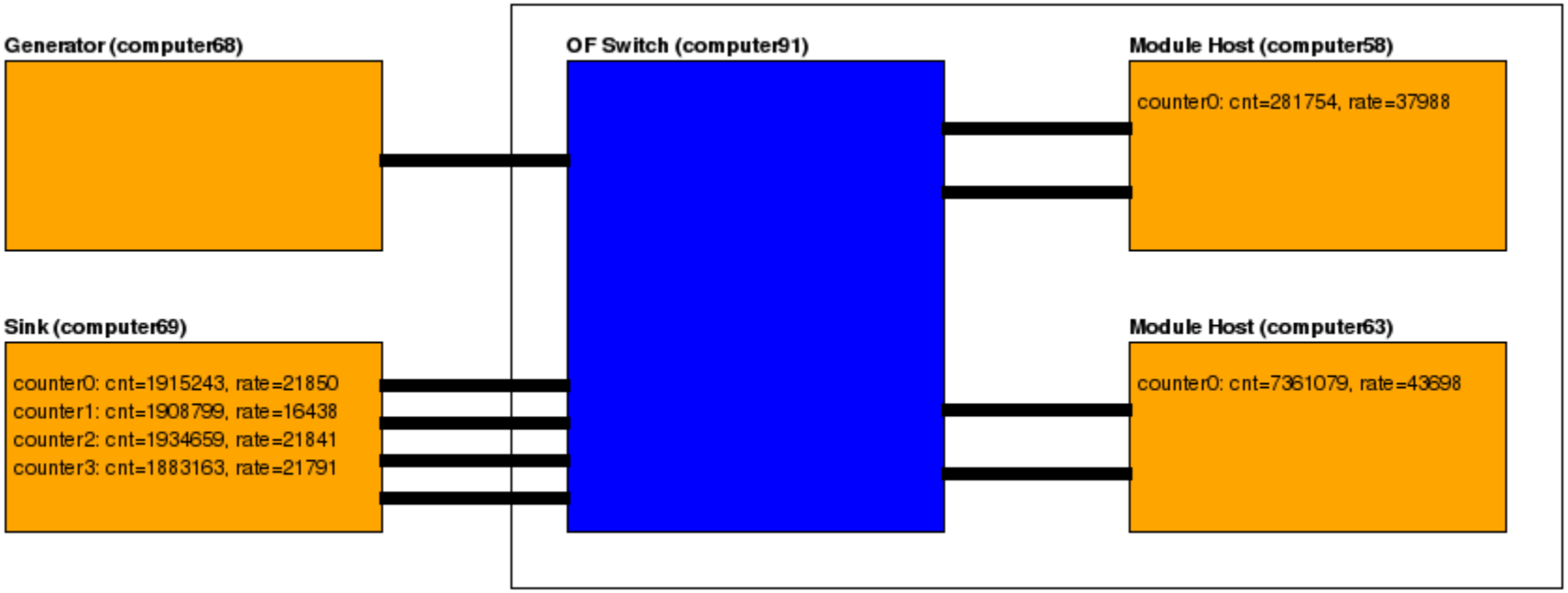
IP Router (idle)



IP Router (one computer)



IP Router (two computers)



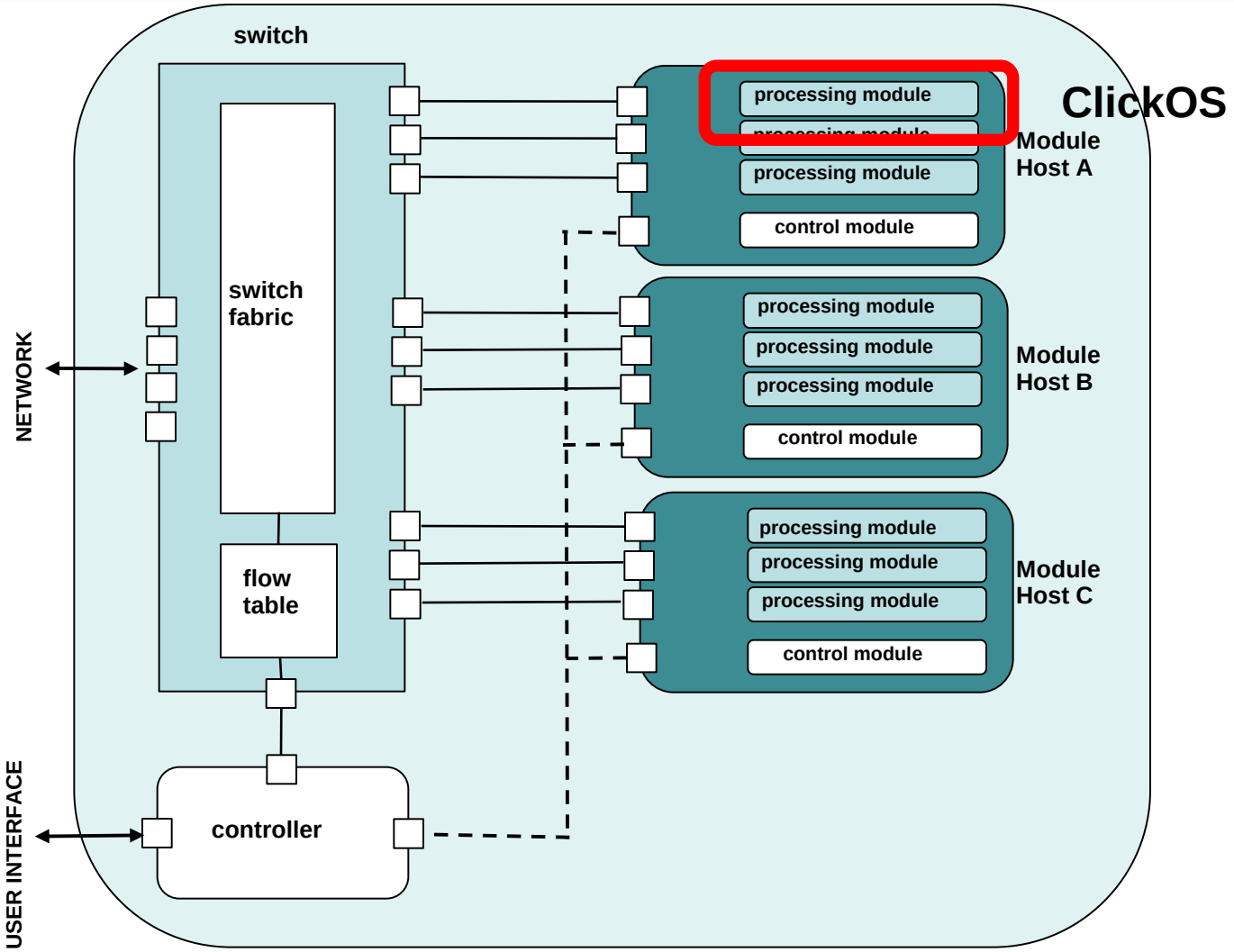
Implementation Status and Future Work

- **Currently working on**
  - Resource allocation algorithm
  - Other types of processing (e.g., carrier-grade NAT)
  - Flow migration algorithms and primitives
  - Integration with different kinds of module hosts, processing modules
    - » ClickOS…

Outline

- Flowstream - Flow Processing Platform
- **ClickOS – Tiny virtual machine for flow processing**

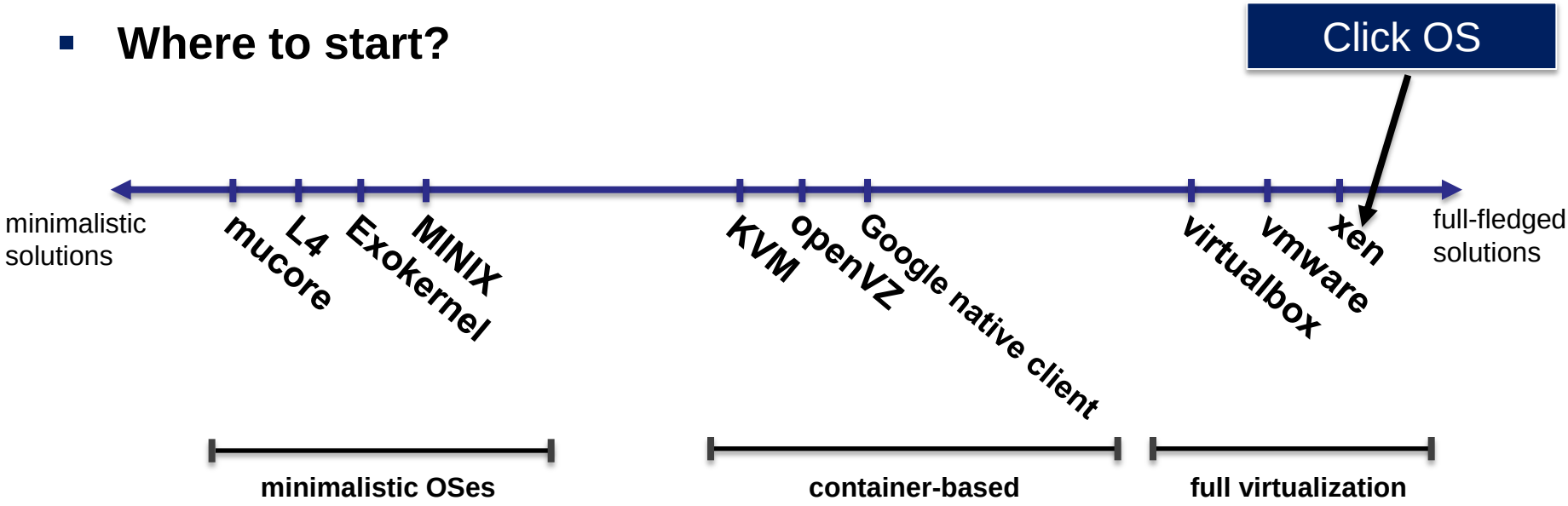
Flowstream: components



Module Host / Processing Module

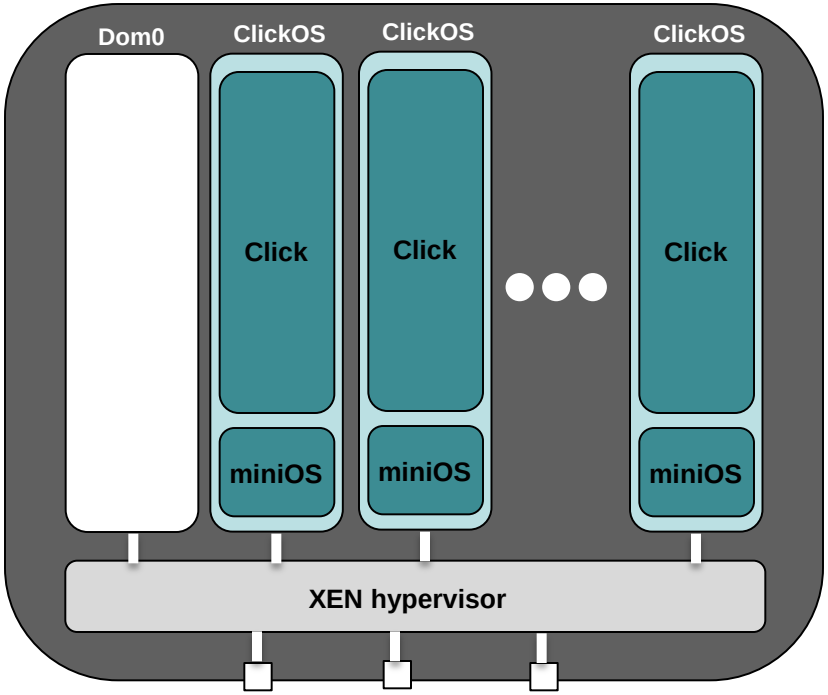
- **Requirements**
  - Flexible processing
  - Good performance
  - Isolation
  - Support for potentially many “domains”
    - » So ideally would be like “small” domains (i.e., not full OSes)

- **Where to start?**



ClickOS Overview

- **Runs as a guest domain on XEN**
- **Is based on mini OS, a minimalistic OS created for XEN stub domains**
  - Stub domains were originally intended to run (perhaps untrusted) drivers
  - Mini OS + Click = Click OS



Advantages

- **Click OS takes advantage of features provided by XEN...**
  - Domain isolation
  - Driver support
  - Performance features (e.g., device passthrough)
- **…but is still small**
  - Click OS image is only 2.5 MB with most Click elements compiled
- **Takes advantage of the processing flexibility provided by Click**

A One-Slide Click Primer

- **Modular architecture for network processing**
- **Based around the concept of "elements"**
- **Elements are connected in a configuration file**
- **A configuration is installed via a command line executable**
  - (e.g., click-install router.click)
- **An element**
  - Can be configured with parameters
  - Can expose read and write variables available via sockets or the /proc system under Linux

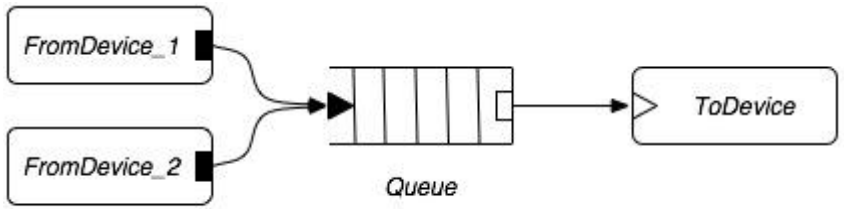
ELEMENT HANDLERS

length (read-only)
Returns the current number of packets in the queue.
highwater_length (read-only)
Returns the maximum number of packets that have ever been in the queue at once.
capacity (read/write)
Returns or sets the queue's capacity.
drops (read-only)
Returns the number of packets dropped by the queue so far.
reset_counts (write-only)
When written, resets the `drops` and `highwater_length` counters.
reset (write-only)
When written, drops all packets in the queue.

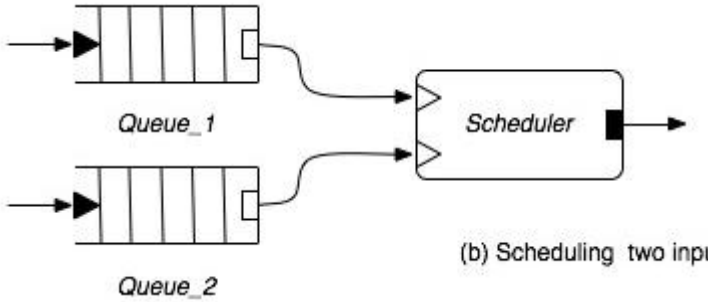
SYNOPSIS

Queue
Queue(CAPACITY)

Ports: 1 input, 1-2 outputs
Package: standard (core)



(a) Merging two input flows



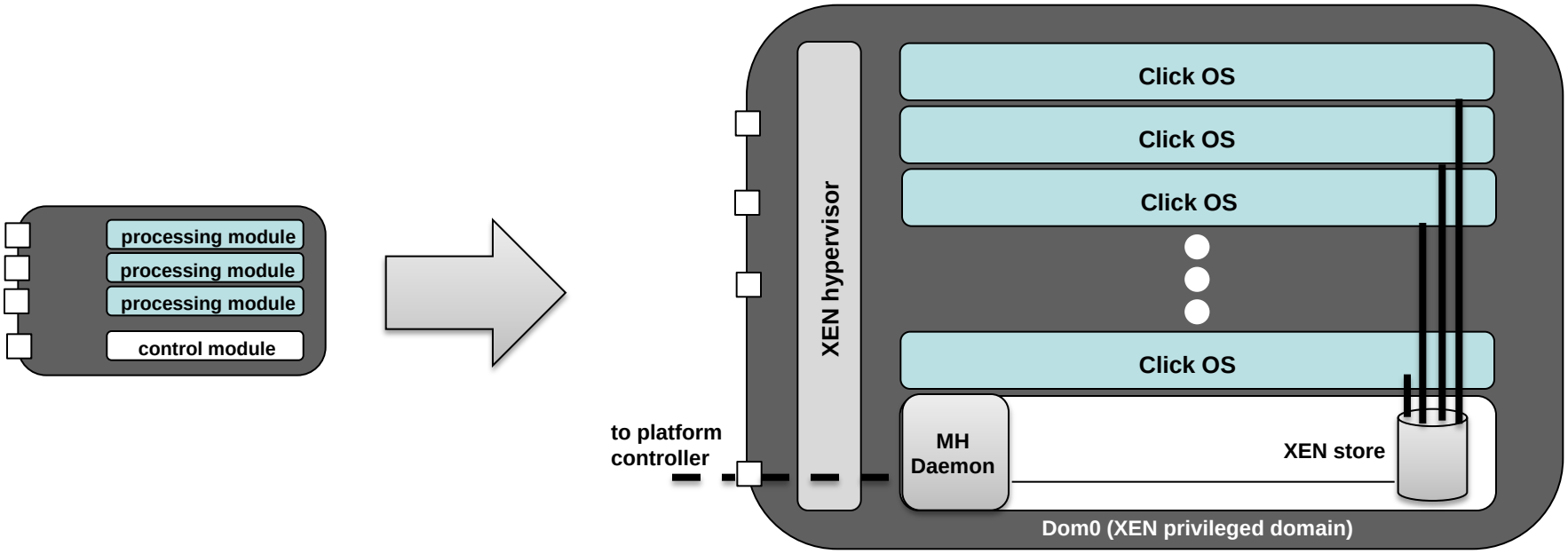
(b) Scheduling two input flows

Click OS – Control Plane

- **Click provides mechanisms to**
  - Install configurations (through the command line)
  - Interact with elements
    - » By providing read and write handlers
- **ClickOS must provide equivalent mechanisms, even though the commands will come from dom0**

Click OS – Control Plane

- **Use the XEN Store for control plane messaging**
  - XenStore is an information storage space shared between domains
  - Similar in spirit to procfs
    - » e.g., /local/domain/0/memory/target
  - Each domain gets its own path in the store



Click OS – Control Plane

- **Modify Click's main driver loop**
  - To check element handlers
  - To implement dynamic reconfiguration
  - Only need to check for changes from time to time, not at every loop execution
- **Example: Click OS configuration installation**
  1. Platform controller tells module host daemon to install a configuration
  2. Module host daemon writes the configuration to the relevant entry in the XEN store (e.g., /local/domain/<domID>/clickos/config)
  3. Remove configuration by writing empty string

ClickOS Startup

```
net TX ring size 256
net RX ring size 256
netfront_add_rx_handler(): list head initialized ('OS netfront RX handler (lwip)').
backend at /local/domain/0/backend/vif/1/0
mac is 00:0c:76:6b:f3:13
reading backend values from xenbus...
xenbus_read for IP.. (ip ptr = 0x19ff78)
xenbus_read done, ip=(null)
*************************
NODEDEV 00:0c:76:6b:f3:13
[main] IP 0 netmask 0 gateway 0.
[main] TCP/IP bringup begins.
Thread "tcpip_thread": pointer: 0x840041e0, stack: 0x2b0000
[tcpip_thread] TCP/IP bringup ends.
[main] Network is ready.
Thread "pcifront": pointer: 0x84004650, stack: 0x2c0000
"main"
[ClickOS] ====================================
[ClickOS] click-os.cc::main Starting click
[ClickOS] ====================================
Before click export elemetns
After click export elements
[ClickOS] click-os.cc::main pathname=/control/click/startup_config
[ClickOS] click-os.cc::main path=control/click/startup_config
pcifront_watches: waiting for backend path to appear device/pci/0/backend
[ClickOS] click-os.cc::main init xenbus watch 0x0
[ClickOS] click-os.cc::main init xenbus watch 0x0, path=control/click/startup_config
[ClickOS] click-os.cc::main xenbus read -> wait [mgs=ENOENT]
[ClickOS] click-os.cc::main init xenbus watch 0x0
```

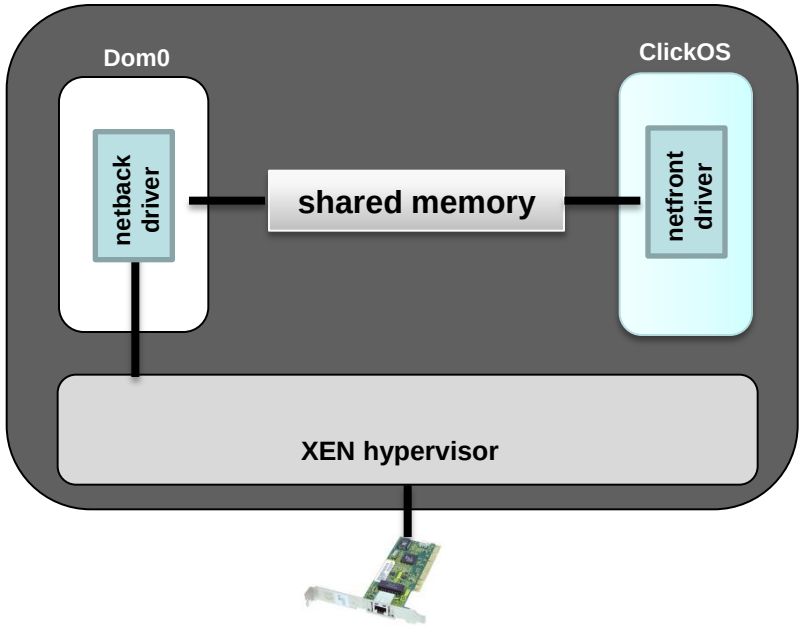
~2.5 secs

ClickOS Configuration Installation

```
FromClickOS(eth1) -> Print -> Discard;]
[ClickOS] click-os.cc::main Click config in:
[FromClickOS(eth1) -> Print -> Discard;]
[ClickOS] click-os.cc::main Parsing configuration
[ClickOS] click-os.cc::parse_configuration ...
[ClickOS] click-os.cc::parse_configuration: Parsed router 0x19ff40
click-os.cc::parse_elemensts: Router elements read:
 [InfiniteSource
ZeroDiscard
Discard
FromClickOS
Print
Error
SimpleQueue
ZeroSource
ToClickOS
]
FromClickOS::configure(): conf=eth1
get_net_dev()
NODEDEV 00:0c:76:6b:f3:13
LWIP_C 00:0c:76:6b:f3:13
FromClickOS: Calling netfront_add_rx_handler()...
netfront_add_rx_handler(): Added handler 'FromClickOS netfront handler'.
netfront - Registered RX handlers:
        OS netfront RX handler (lwip)
        FromClickOS netfront handler
[ClickOS] click-os.cc::parse_elemensts: Click router init OK
[ClickOS] click-os.cc::main Router instance created.
[ClickOS] click-os.cc::main router->use()
[ClickOS] click-os.cc::main Activating router...
[ClickOS] click-os.cc::main Starting driver thread...
CLICKOS
```

Click OS – Data Plane

- **I/O in Click**
  - Click relies on the elements PollDevice/FromDevice and ToDevice for interacting with interfaces
- **I/O in XEN**
  - Achieved through a “split” driver:
    - » Netback: portion in the physical domain handling the actual network device.
    - » Netfront: portion in the guest domain acting as a proxy to the netback.



Click OS – Data Plane

- **In Click OS, need networking elements that talk to XEN's netfront**
  - FromClickOS: receives packets from the netfront driver
  - ToClickOS: sends packets to the netfront driver

Multiplying ClickOS

- **ClickOS images are small (~ 2.5MB)**
  - As opposed to regular guest domains (e.g., Linux-based ones), we can potentially run many of them
- **To do so, had to modify the dom0 kernel to increase the number of IRQs**
- **Created 1,010 ClickOS domains**

```
successfully created vm 0
successfully created vm 1
successfully created vm 2

...

successfully created vm 1010
error while creating vm 1011
libxl: error: libxl.c:142:libxl_domain_rename create xs transaction for domain (re)name: Invalid argument
cannot make domain: -3
```

Future Work

- **Currently cleaning-up the control plane mechanisms**
- **Beginning to run performance experiments**
  - Different number of concurrent ClickOS images
  - Different kinds of processing
  - Different traffic loads
  - Using (vs not) hypervisor pass-throughs
  - Using (vs not) hardware multi-queues, VMDq, SR-IOV, etc
  - 1Gb vs 10Gb cards
  - [Your variable here…]