CICN
Community
Information-Centric Networking

Tutorial at ACM SIGCOMM ICN, Berlin, Germany
26th of September 2017

Tutorial agenda

- Project overview
- Vector Packet Processing
- vICN: automation of virtual ICN network deployment
- The consumer/producer socket API with applications to HTTP

CICN project overview

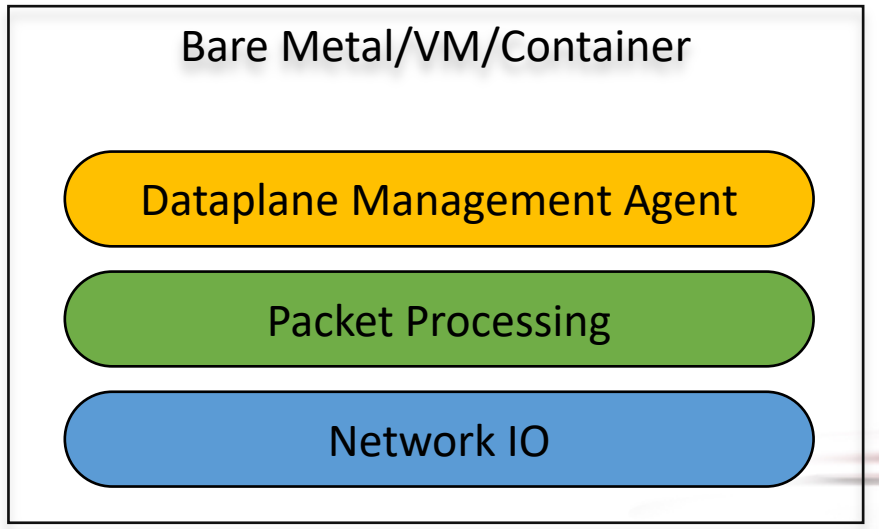
- CCNx Internet documents are specified at the ICNRG define the architecture.
- The rest is just software development, testing and experimentation.
- Focus on VPP and application development:
  - Vector Packet Processing as the Universal Data Plane for vRouting and vSwitching
  - vICN automation of virtual networks deployment
  - The Consumer/Producer Socket API and HTTP

What is FD.io (pronounced “fido”)?

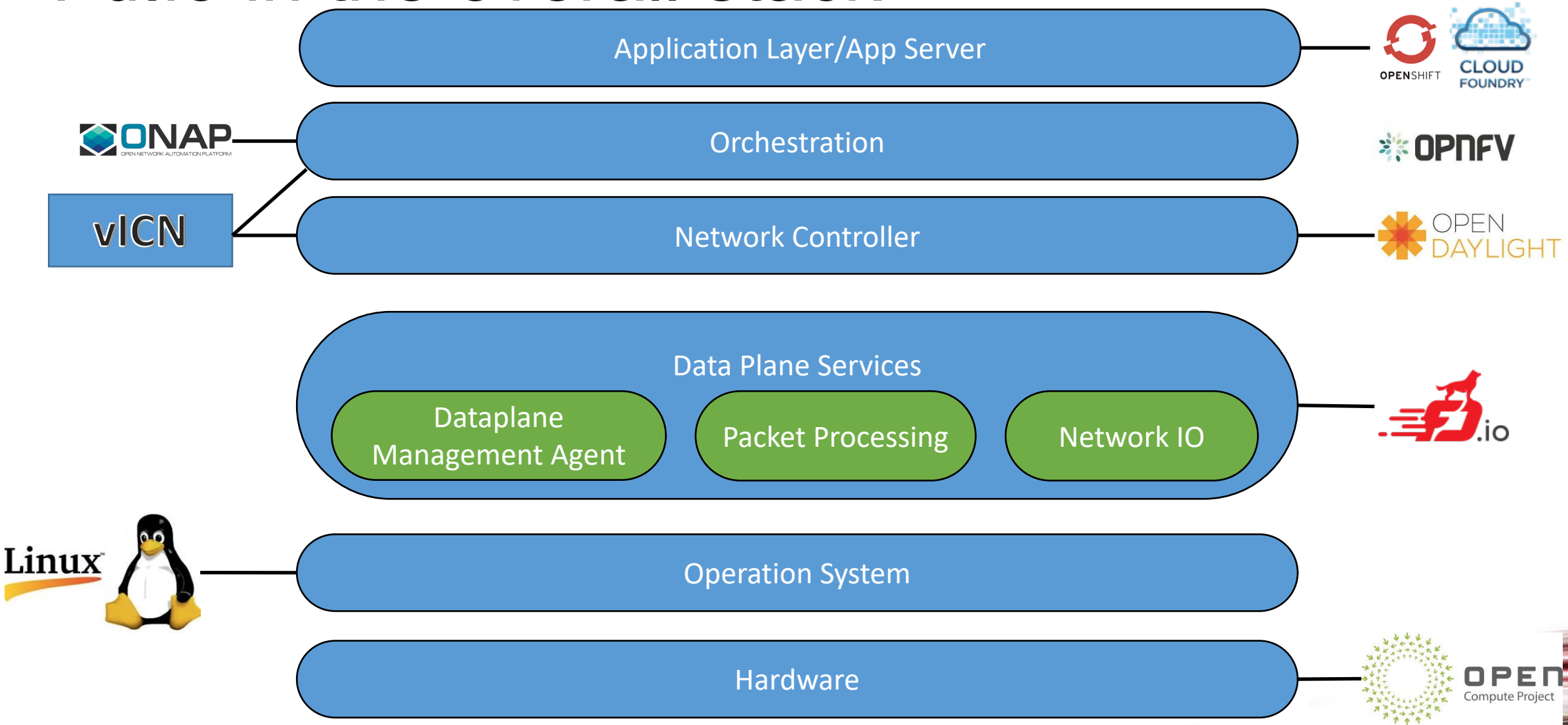
FD.io: The Universal Dataplane

- Project at Linux Foundation
  - Multi-party
  - Multi-project
- Software Dataplane
  - High throughput
  - Low Latency
  - Feature Rich
  - Resource Efficient
  - Bare Metal/VM/Container
  - Multiplatform intel ARM Power

- Fd.io Scope:
  - **Network IO -** NIC/vNIC <-> cores/threads
  - **Packet Processing –** Classify/Transform/Prioritize/Forward/Terminate
  - **Dataplane Management Agents -** ControlPlane



Fd.io in the overall stack



Multiparty: Broad Membership

Service Providers

AT&T

COMCAST

Network Vendors

CISCO

ERICSSON

HUAWEI

ZTE

Metaswitch Networks

BROCADE

Chip Vendors

intel

CAVIUM NETWORKS

Integrators

redhat.

INOCYBE
OPENDAYLIGHTDISTRIBUTION

Multiparty: Broad Contribution










Yandex


Qiniu











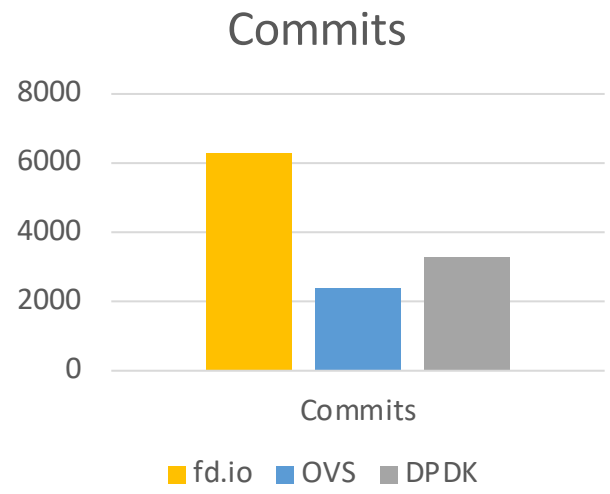


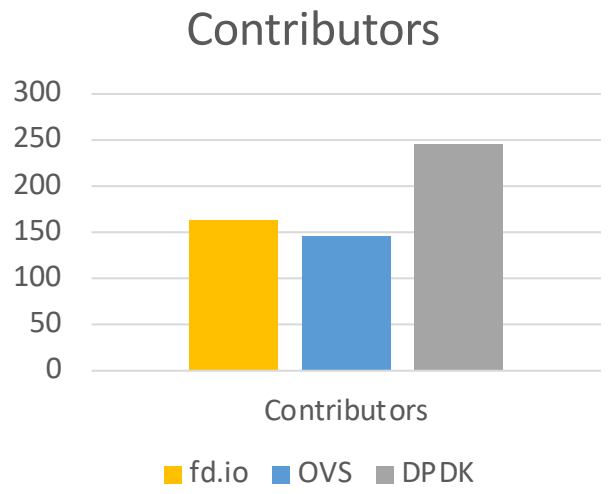
Universitat Politècnica de Catalunya (UPC)

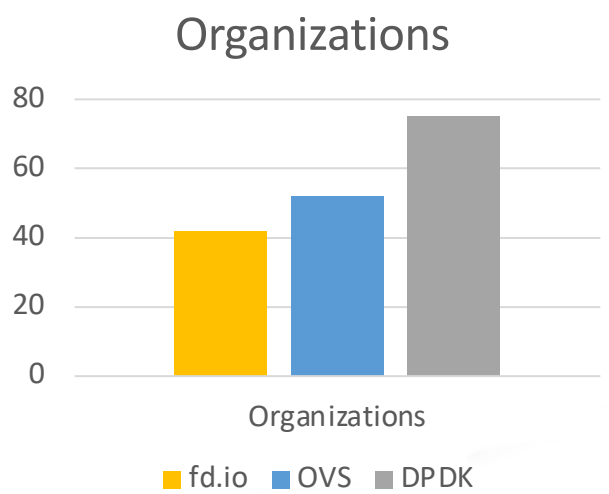
Code Activity

- In the period since its inception, fd.io has more commits than OVS and DPDK combined, and more contributors than OVS

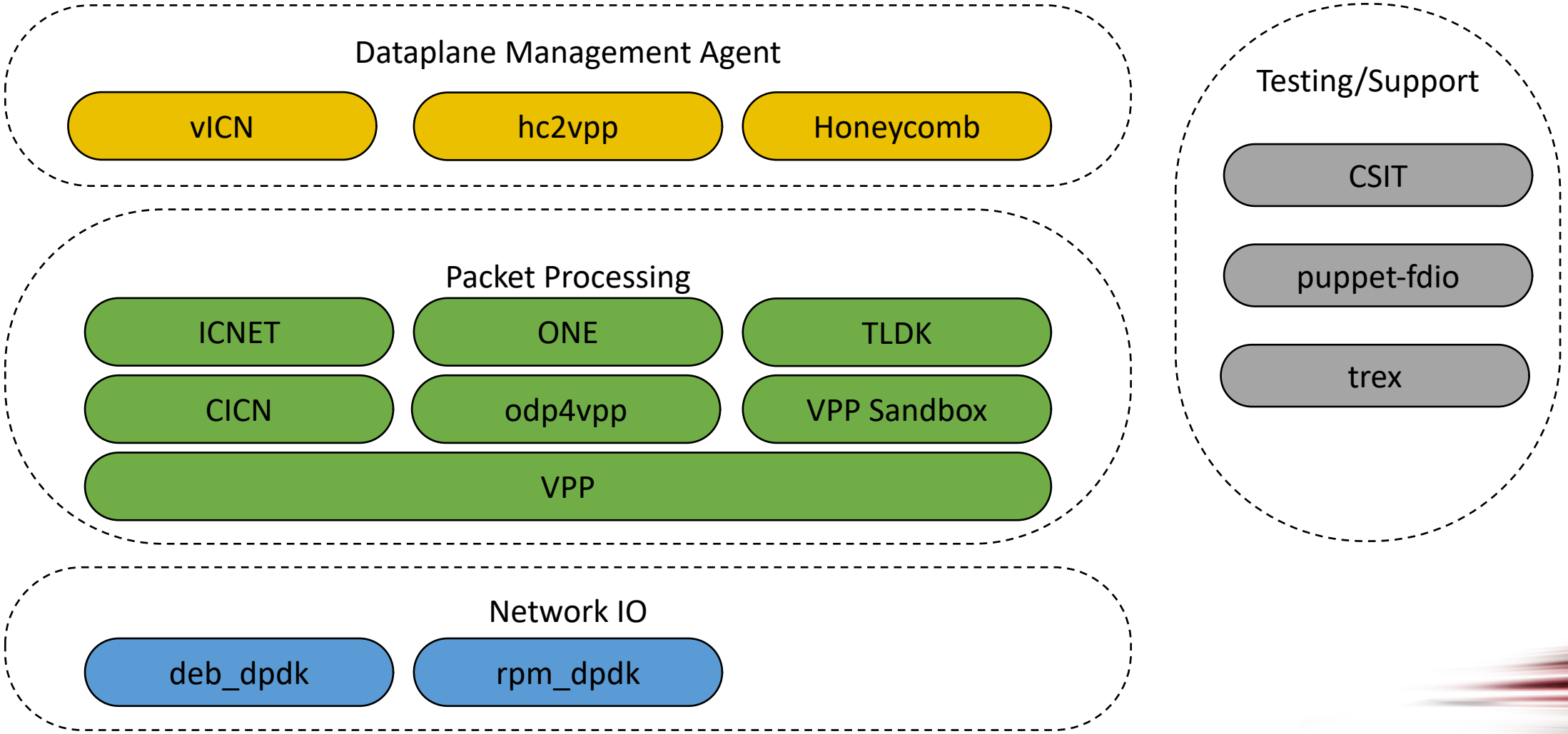
2016-02-11 to 2017-04-03	Fd.io	OVS	DPDK
Commits	6283	2395	3289
Contributors	163	146	245
Organizations	42	52	78



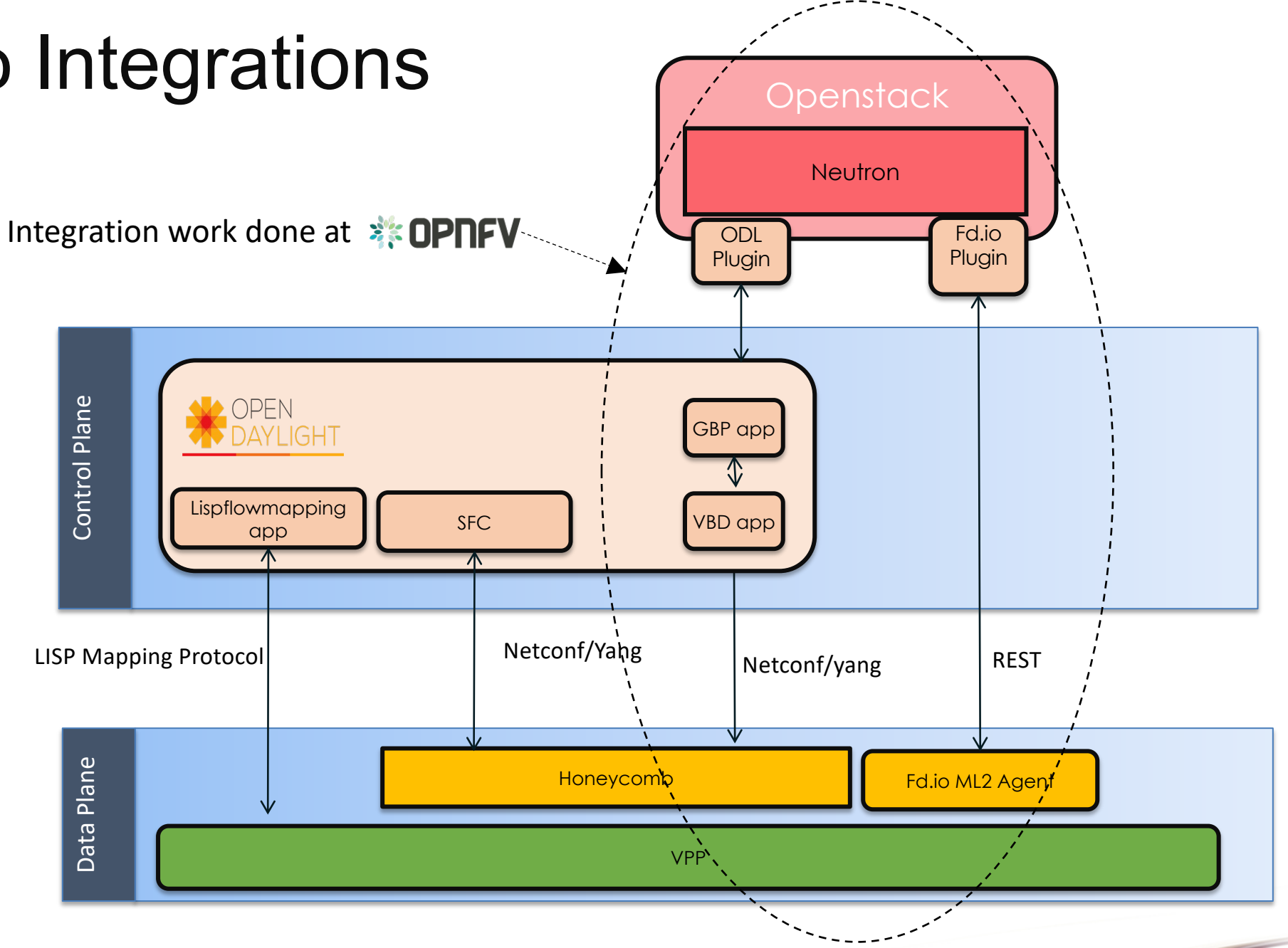




Multiproject: Fd.io Projects

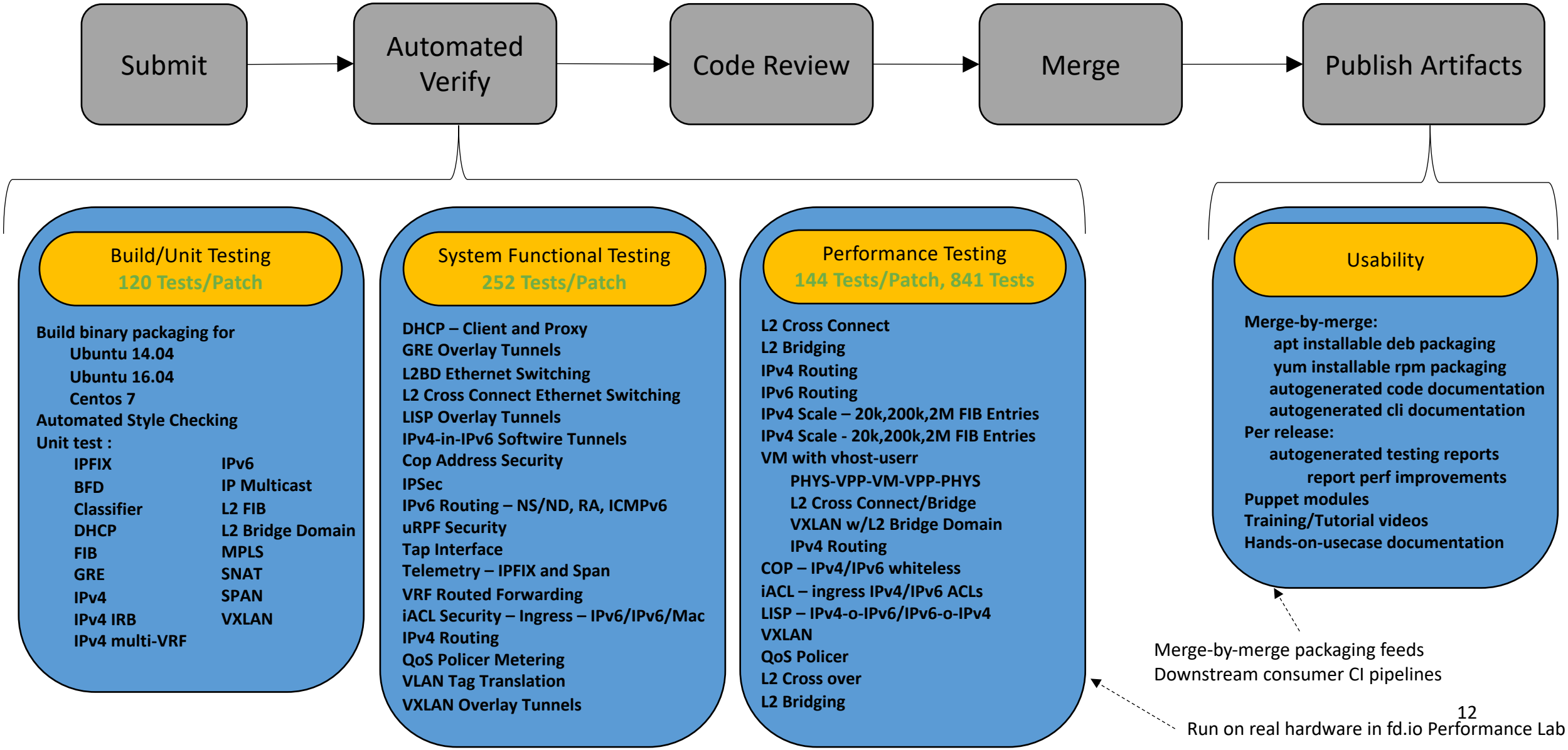


Fd.io Integrations

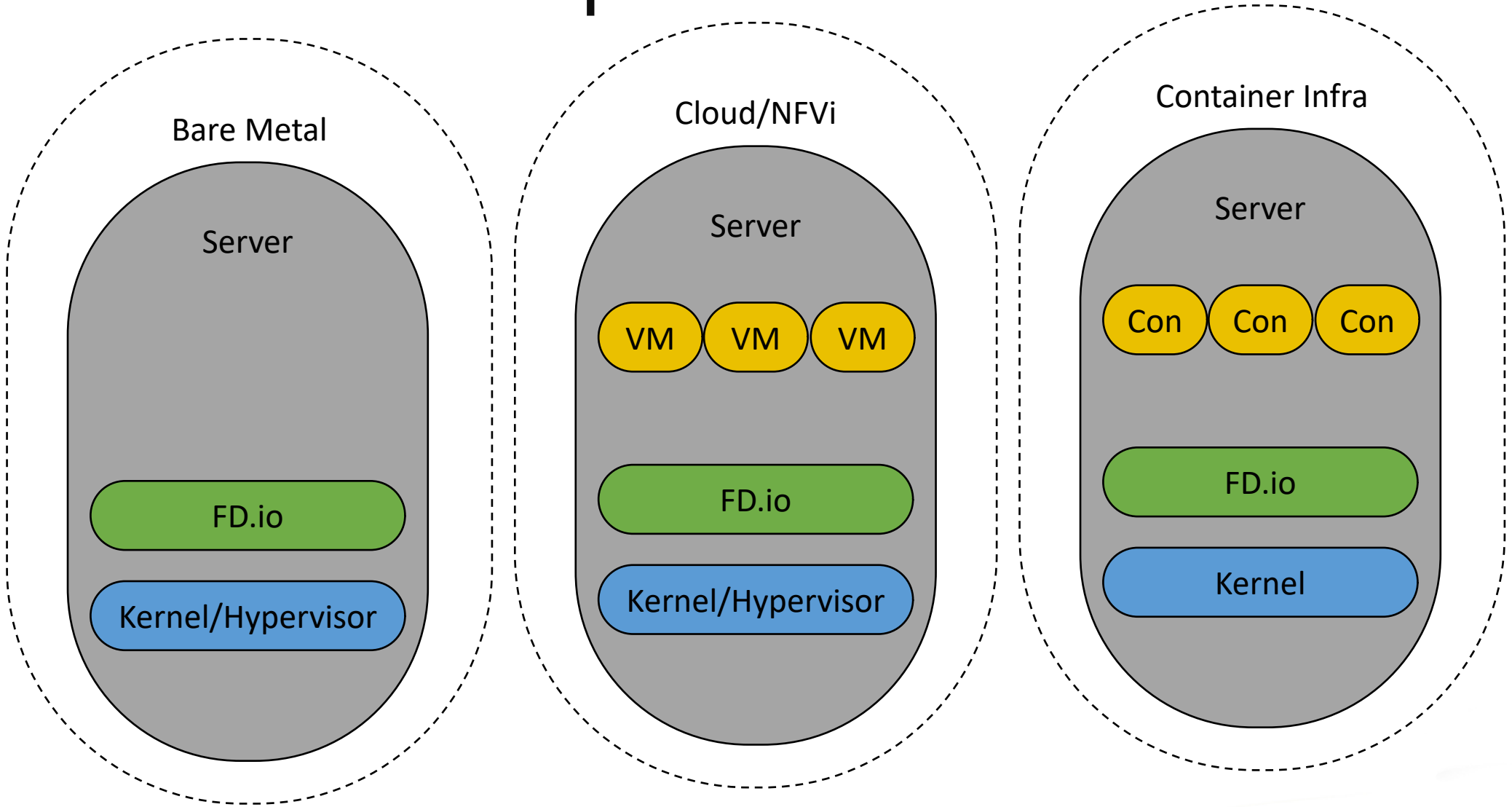


Continuous Quality, Performance, Usability

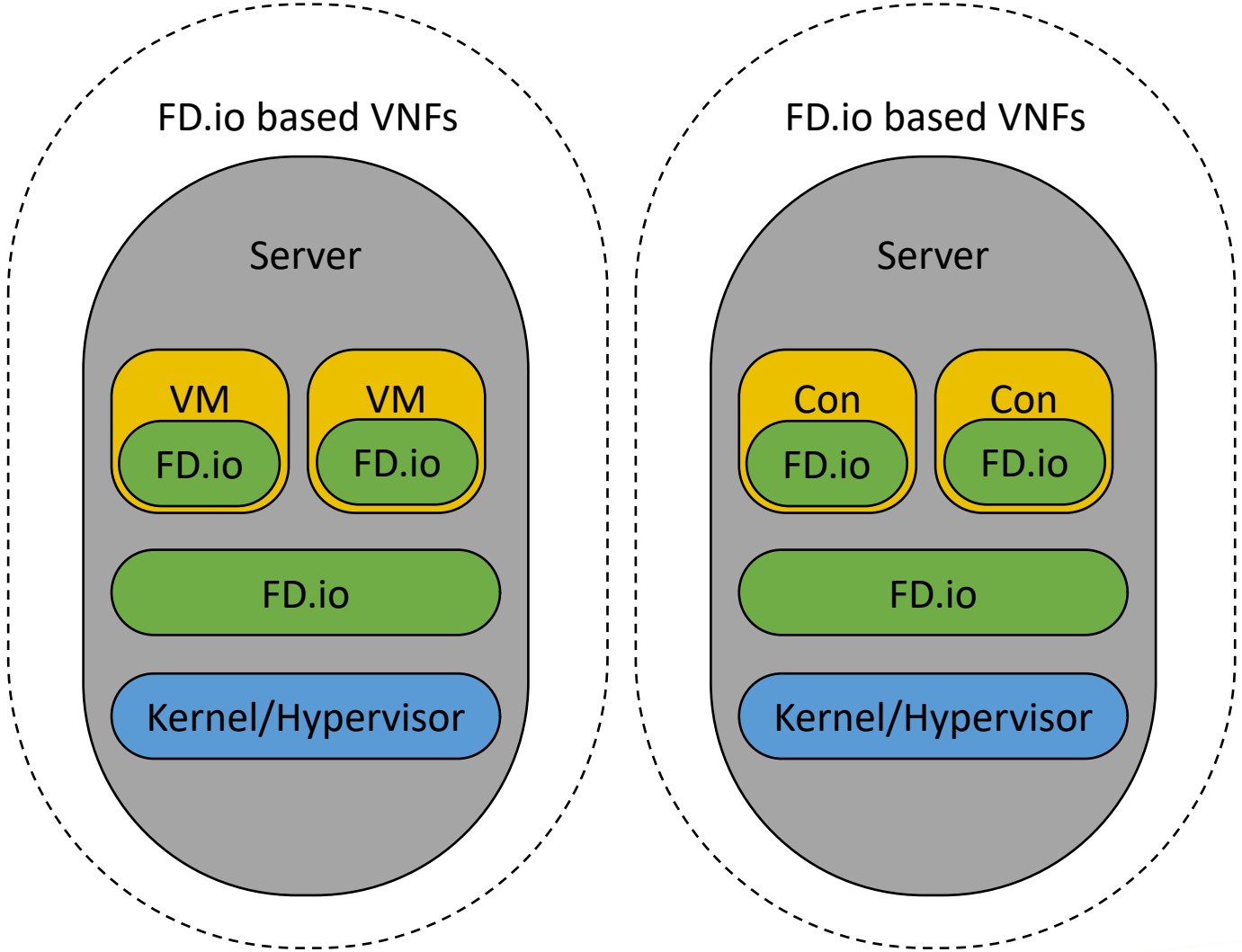
Built into the development process – patch by patch



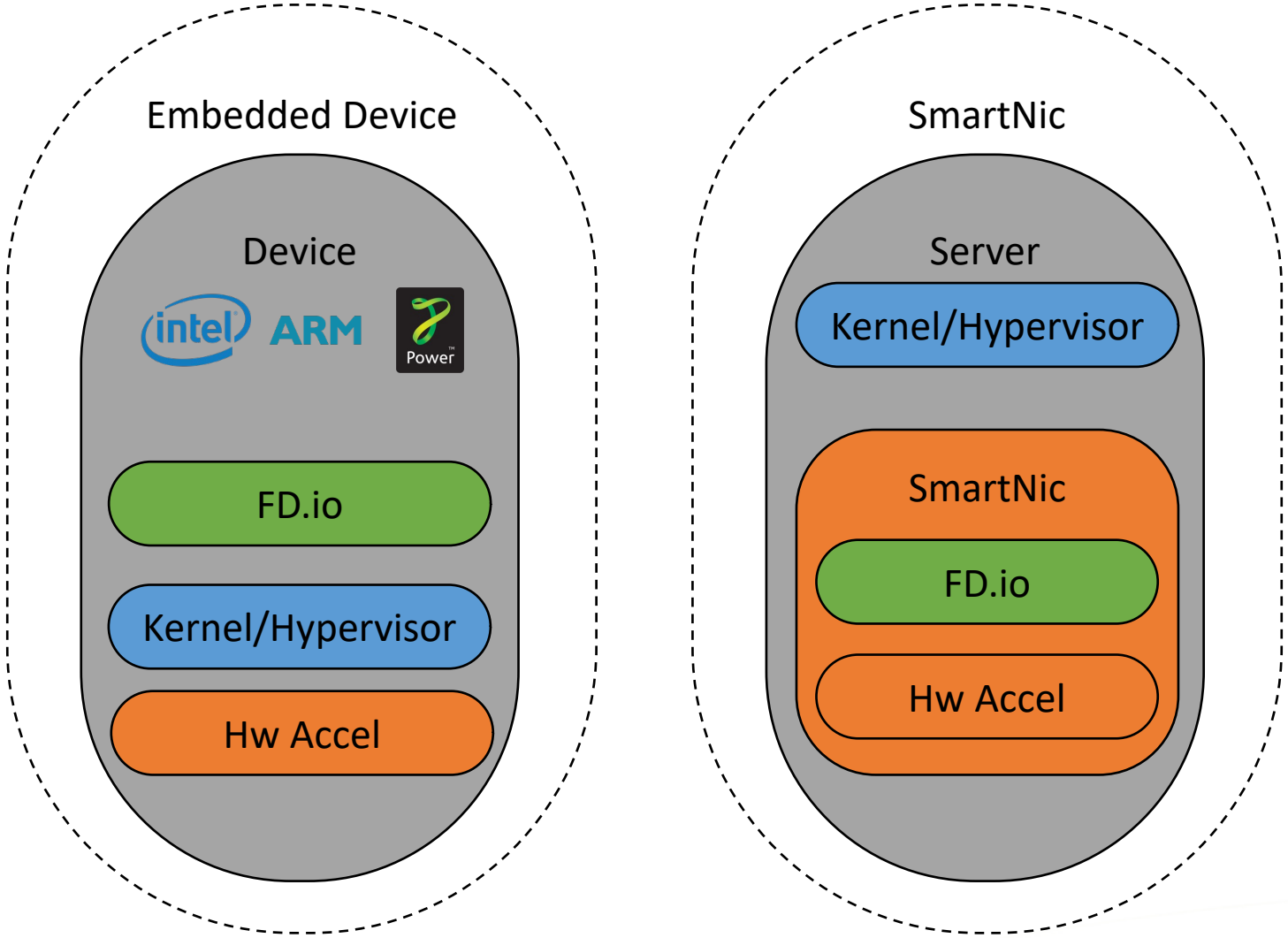
Universal Dataplane: Infrastructure



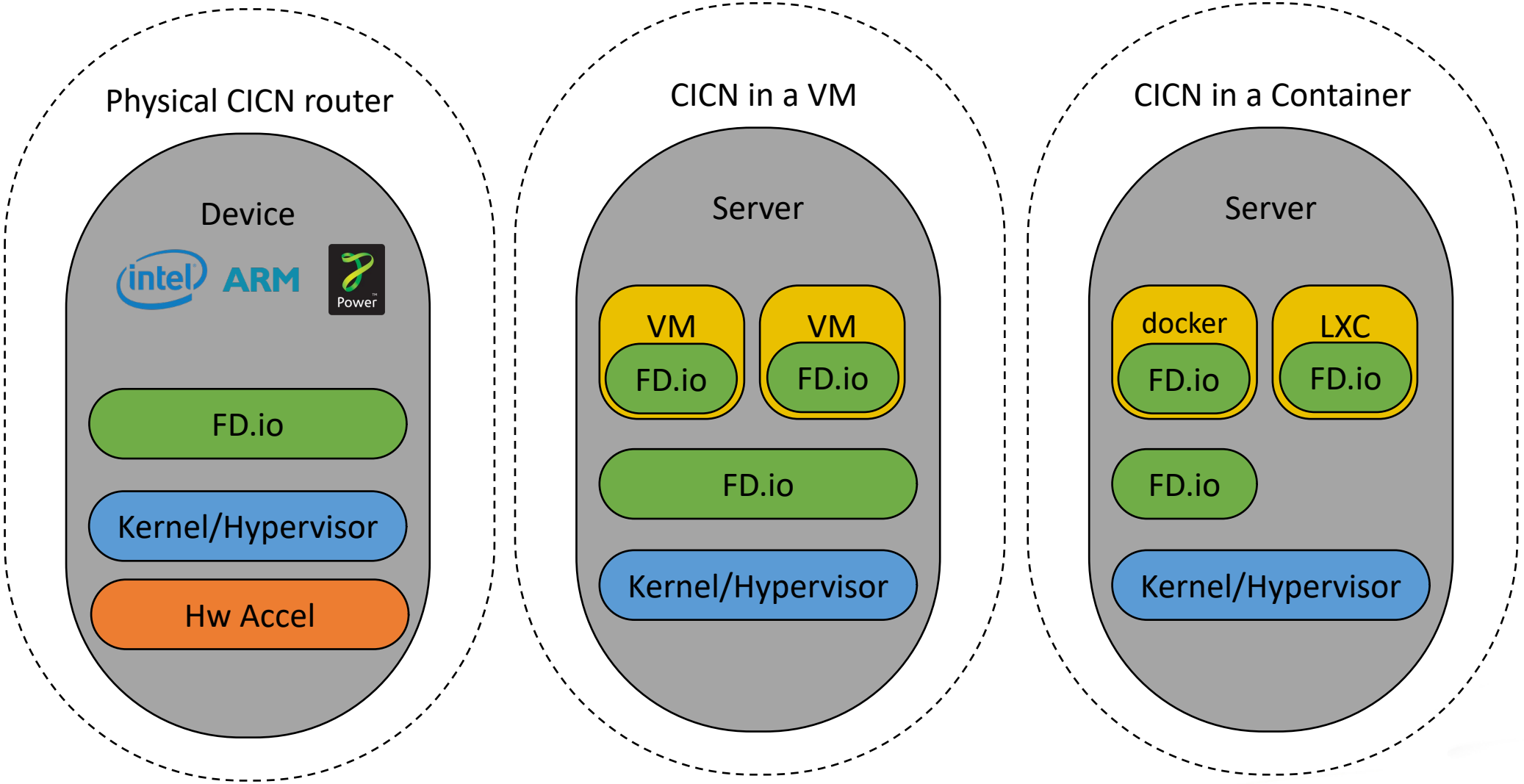
Universal Dataplane: VNFs



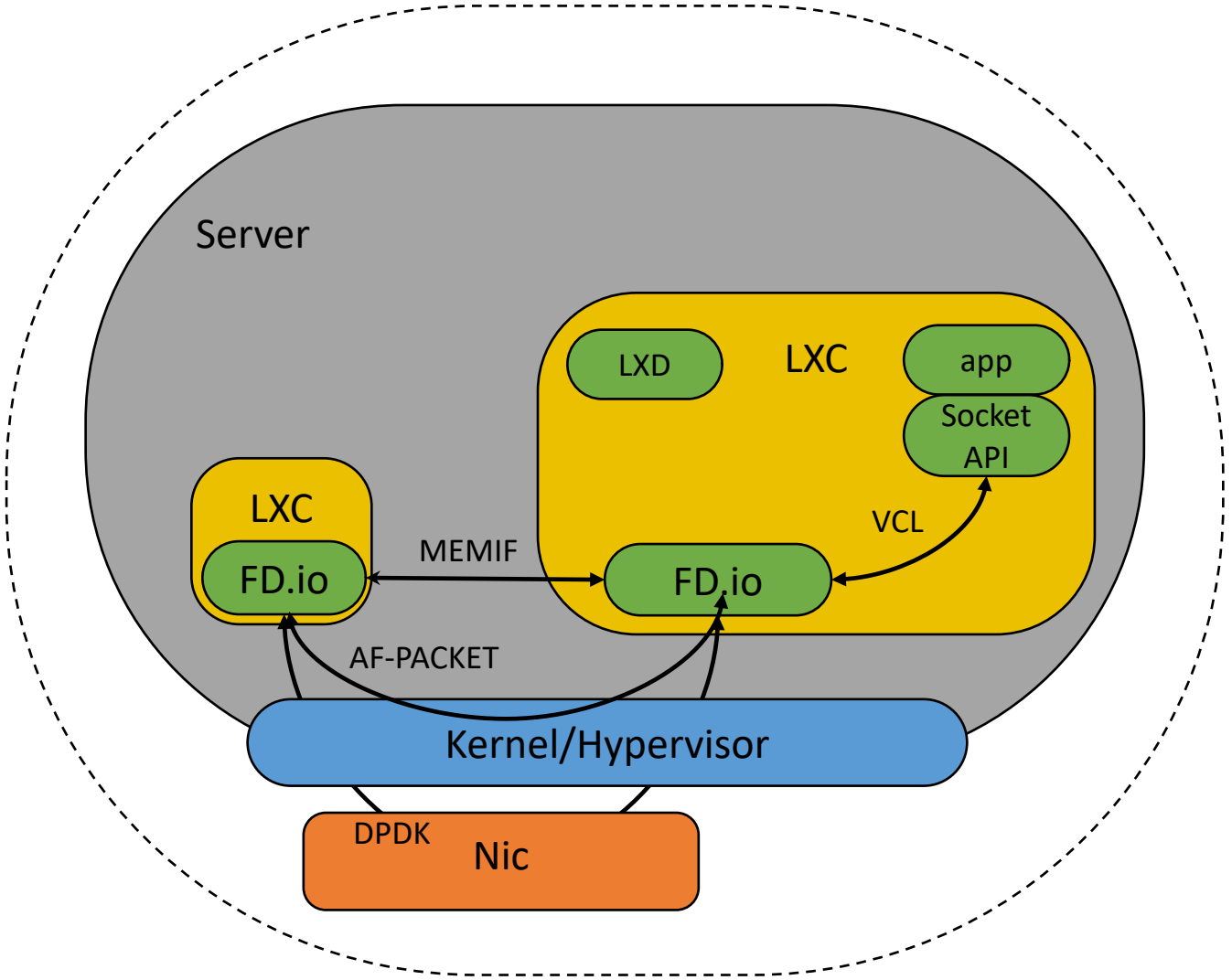
Universal Dataplane: Embedded



Universal Dataplane: CICN Example



Universal Dataplane: communication/API



Consumer/Producer Socket API

- Segmentation/Naming
- Manifest management
- Reassembly
- Flow and Congestion Control

Existing drivers for links

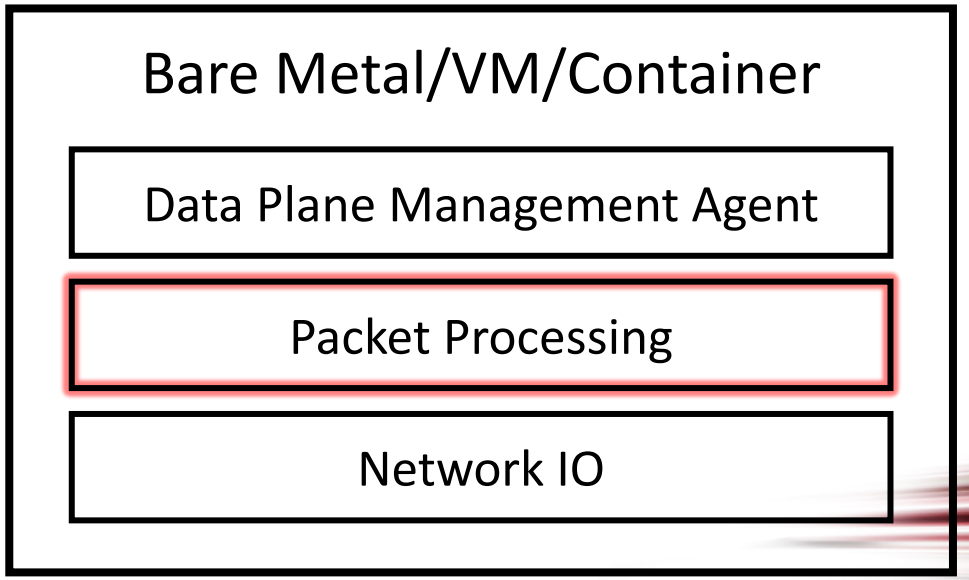
- DPDK
- AF-PACKET
- MEMIF (SHARED MEMORY)

What is Vector Packet Processing?

An open-source software that provides out-of-the-box production quality switch/router functionality running under commodity CPUs

- High Throughput
  - 14+ Mpps per core
- Multiplatform 

- Feature rich
  - L2, L3, L4, local and remote programmability
- Modular and Extensible
  - Through plugins



Why VPP?

- NFV goals
  - Software flexibility without giving up to hardware level performance
- What about existing solutions?
  - Linux Kernel
    - Too slow for high throughput
    - Evolve slowly
  - Click
    - In principle similar to VPP, no V(ector)

CICN distribution

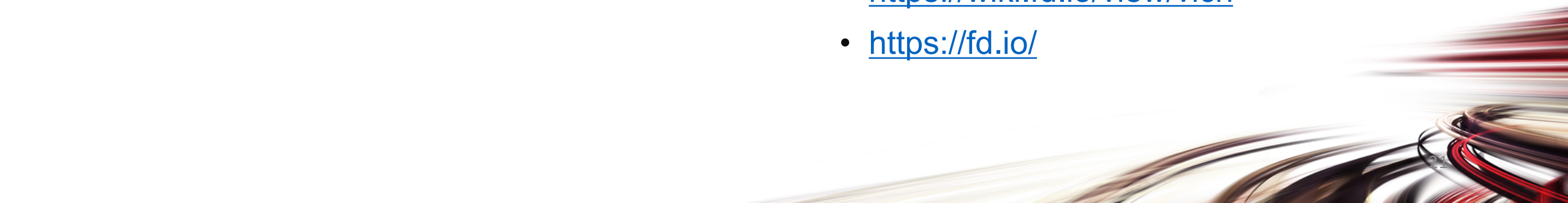
- Core libraries
  - Consumer/Producer Socket API, CCNx libs, PARC C libraries
- Server and Router
  - VPP cicn plugin for Ubuntu 16, CentOS 7
  - HTTP video server, Apache Traffic Server Plugin coming soon
- Client
  - Metis Forwarder
  - VIPER MPEG-DASH video player
  - Android 7/8, MacOS X 10.12, iOS 10/11, Ubuntu 16, CentOS 7
  - Soon Apple Store and Google Play
- vICN
  - intent-based networking
  - model driven programmable framework
  - monitoring and streaming for BigData support (PNDA.io)

Opportunities to Contribute

- Forwarding strategies
- Mobility management
- Hardware Accelerators
- vICN, configuration/management/control
- Consumer/Producer Socket API
- Reliable Transport
- Instrumentation tools
- HTTP integration

We invite you to Participate in fd.io

- Get the Code, Build the Code, Run the Code, install from binaries
- from binary packages
- Read/Watch the Tutorials
- Join the Mailing Lists
- Join the IRC Channels
- Explore the wiki
- Join fd.io as a member
- https://wiki.fd.io/view/cicn
- https://wiki.fd.io/view/vicn
- https://fd.io/




?
Q
&
A
!

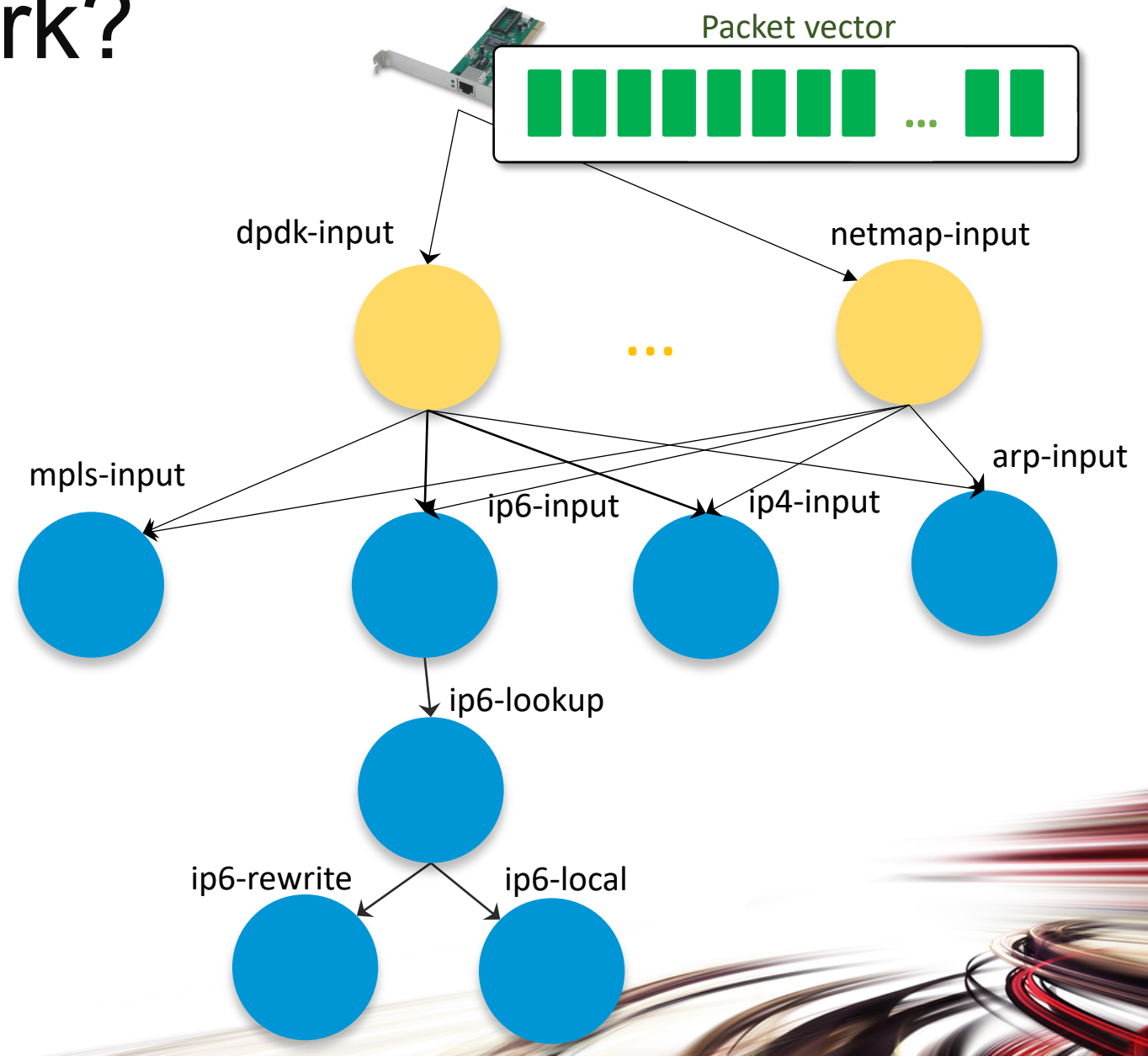
Vector Packet Processing for ICN

Alberto Compagno

Tutorial at ACM SIGCOMM ICN, Berlin, Germany
26th of September 2017

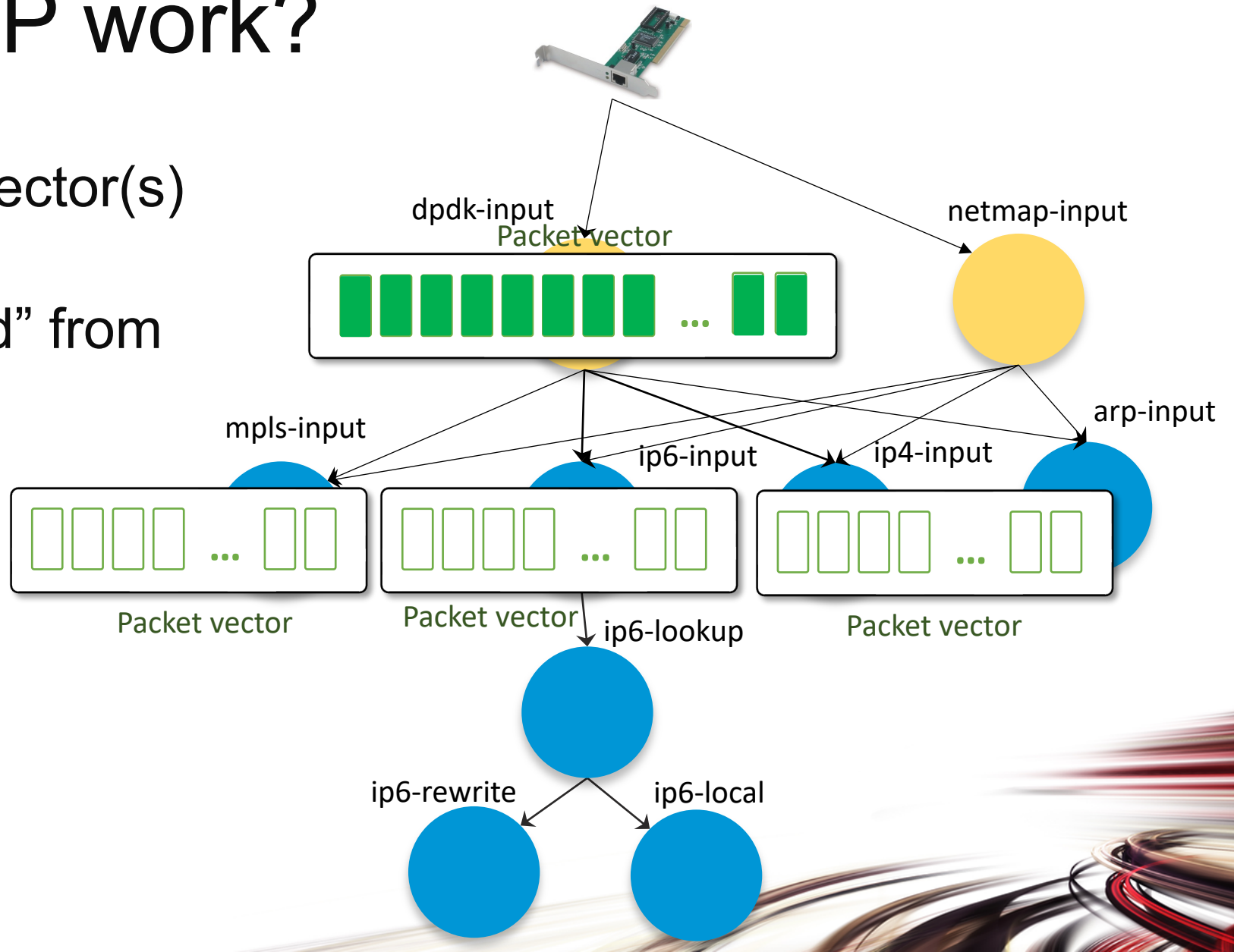
How does VPP work?

- VPP is a ‘packet processing graph’
- Nodes are
  - Small
  - Loosely coupled
- VPP processes vectors of packets
  - Passed from node to node



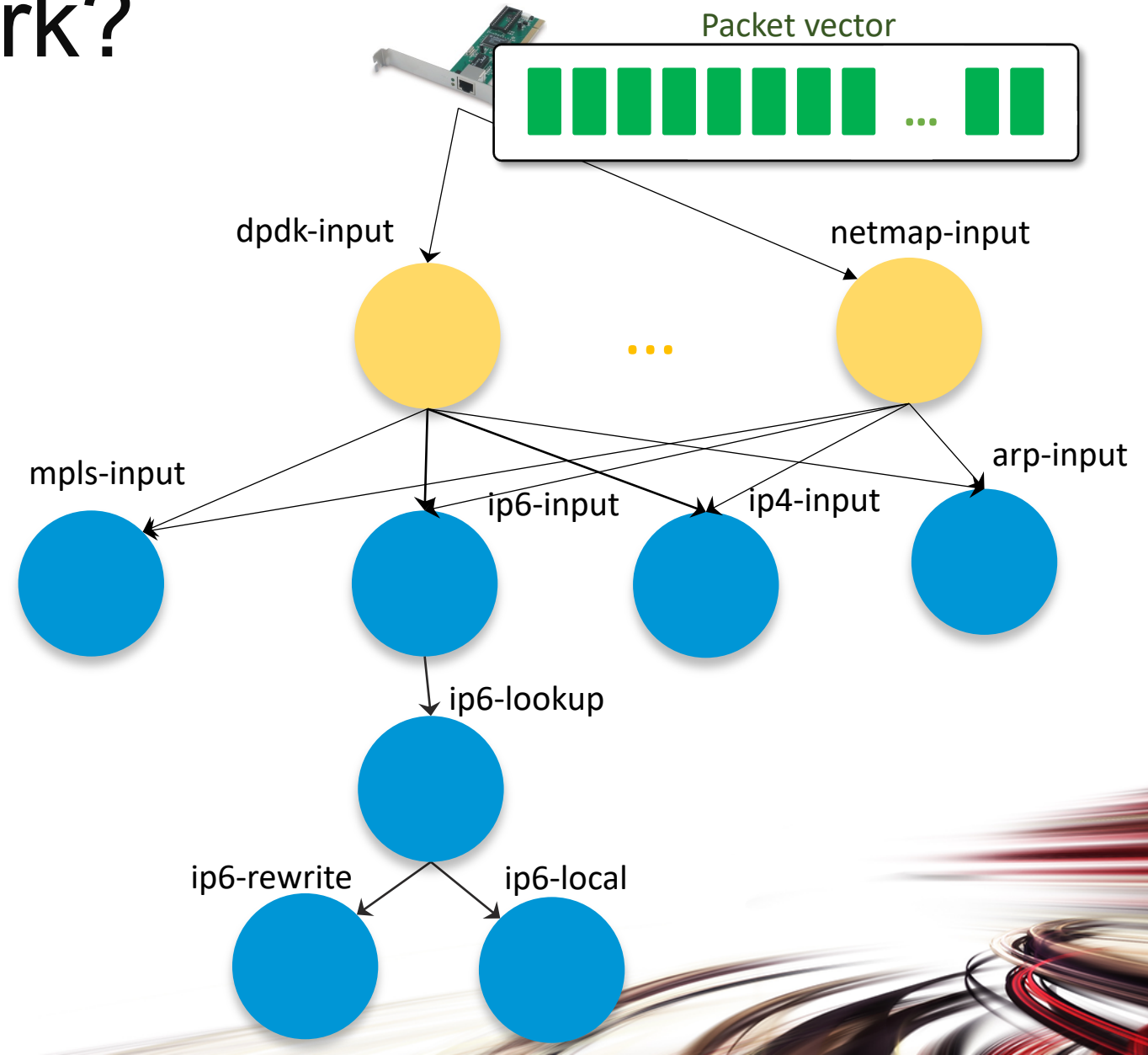
How does VPP work?

- Each node has its vector(s)
- Packets are “passed” from vector to vector



How does VPP work?

- Three types of nodes
  - Input
  - Internal
  - Process



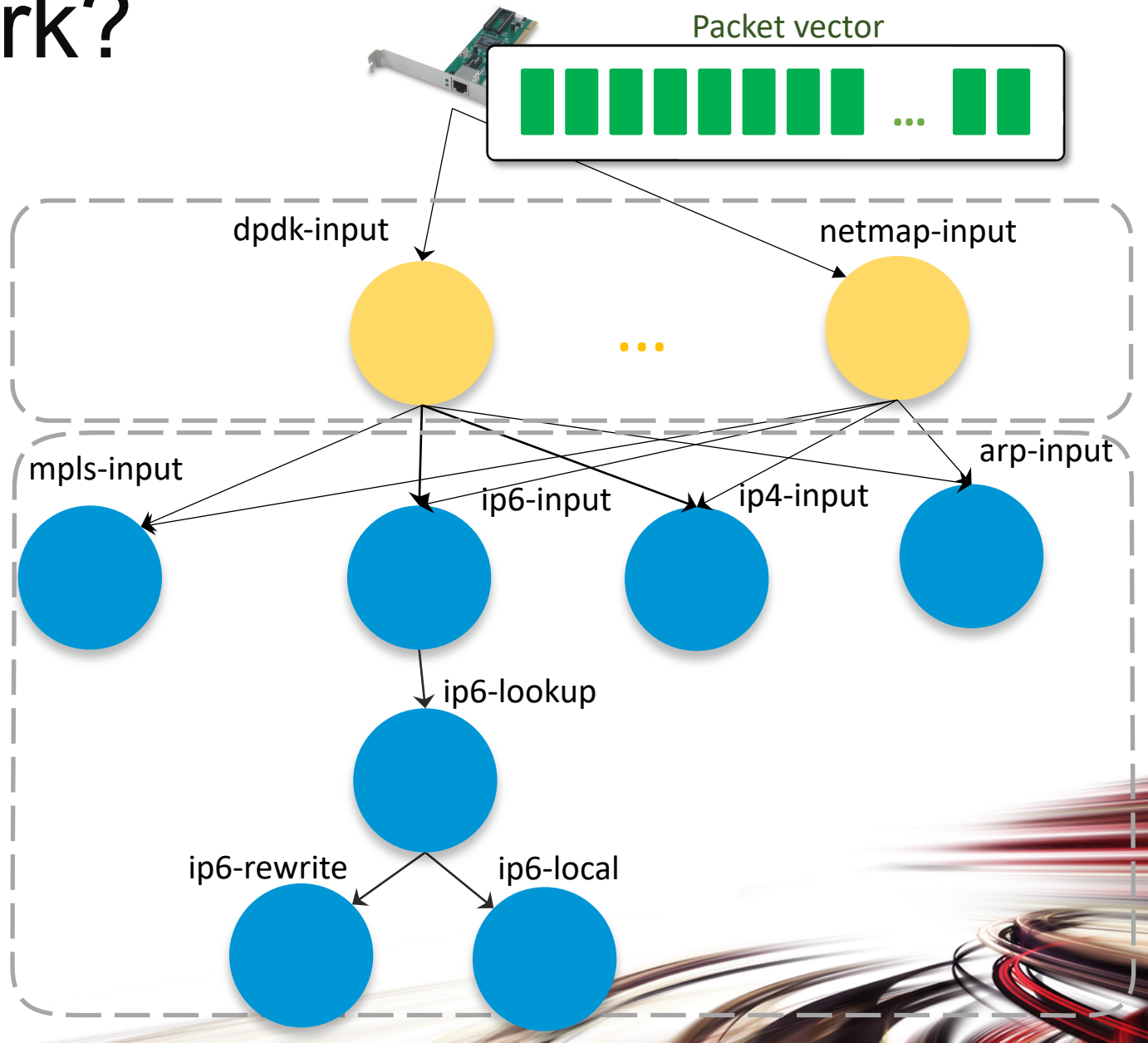
How does VPP work?

Input nodes

- Read packets from RX buffer
- Create the packet vector

Internal nodes

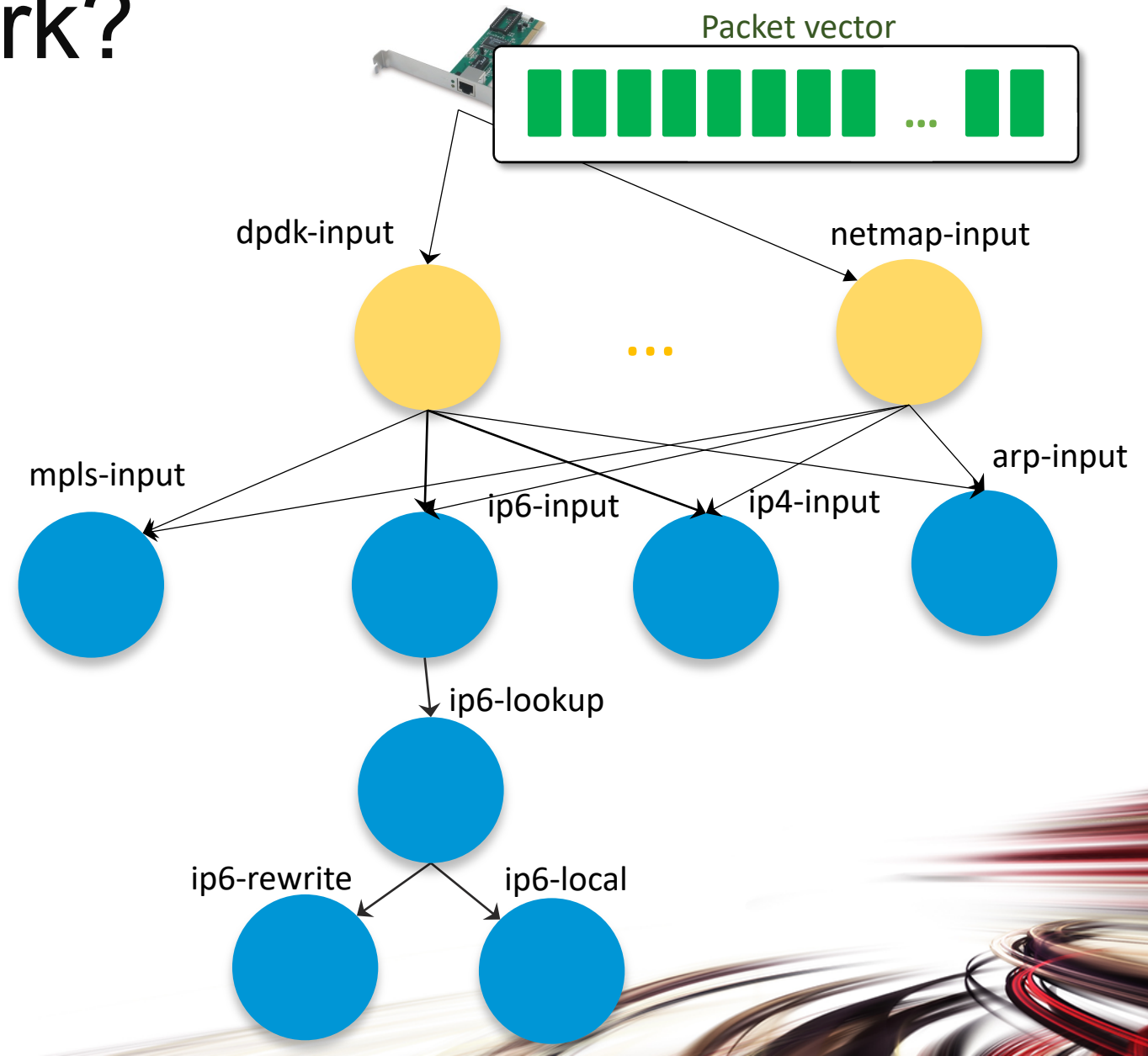
- Process packets
- Called from other nodes
- Can be leaf (drop or TX)



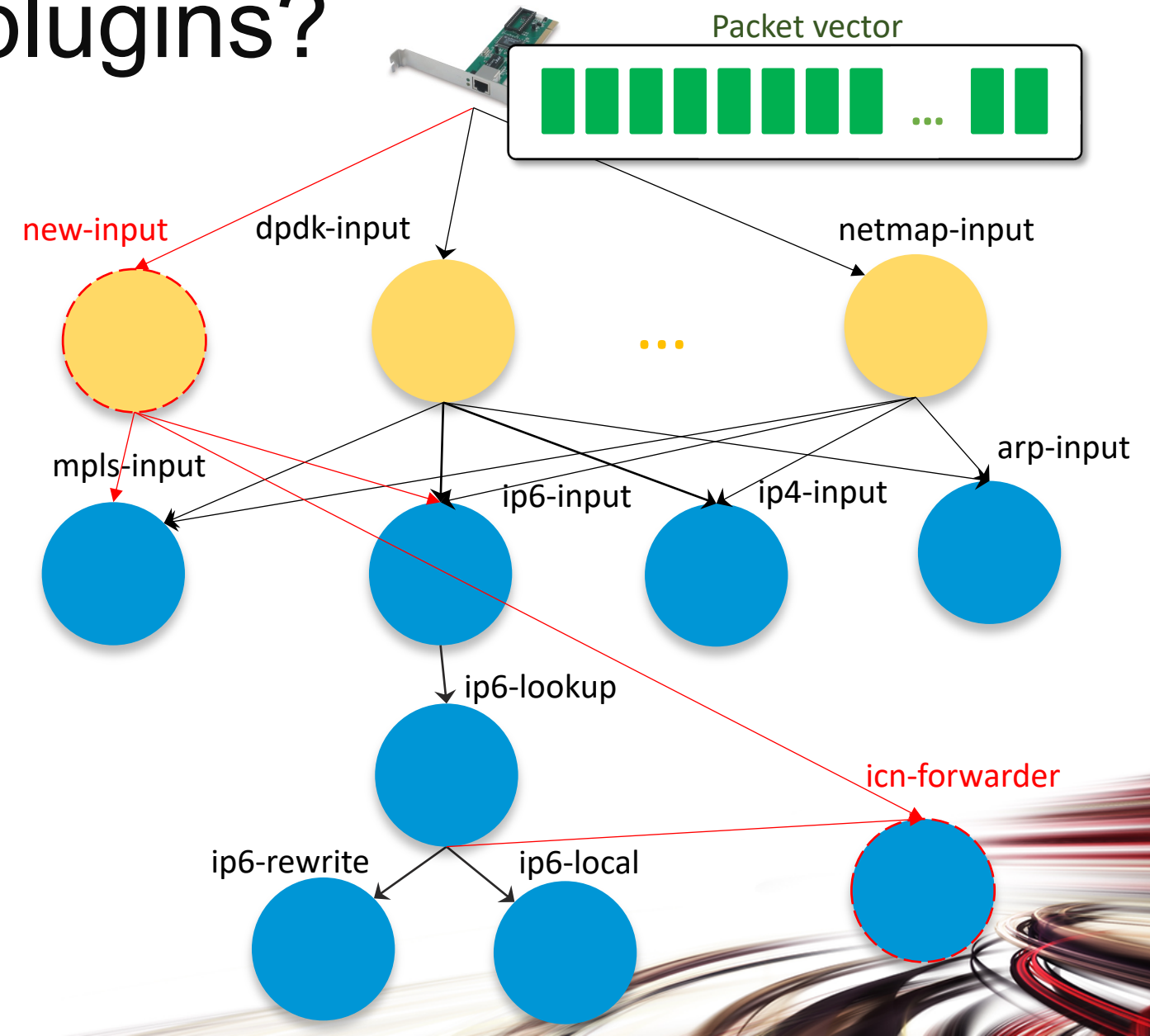
How does VPP work?

Process nodes

- Not part of the processing graph
- Run in background
- React to timer/event



Extend VPP with plugins?



Plugins are first class citizen

They can:

- Add nodes
- Add api
- Rearrange the graph

How does VPP accelerate packet processing?

Accelerating packet processing

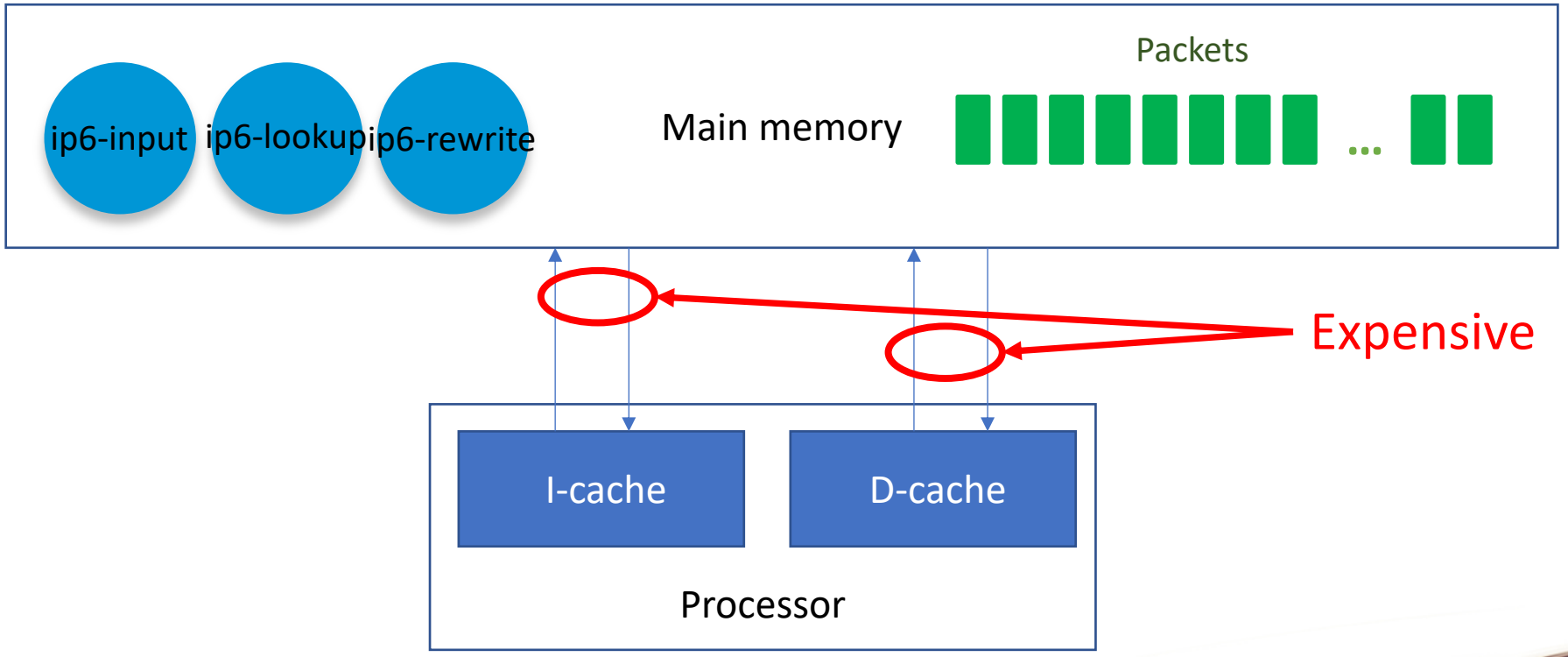
- Kernel bypass
- Code Design (Multi-loop, Branch prediction, Function flattening, Lock-free structures, Numa aware)
- Reduce cache misses

Reduce cache misses – Why?

- 14 Mpps on 3.5GHz CPU = 250 cycles/packet

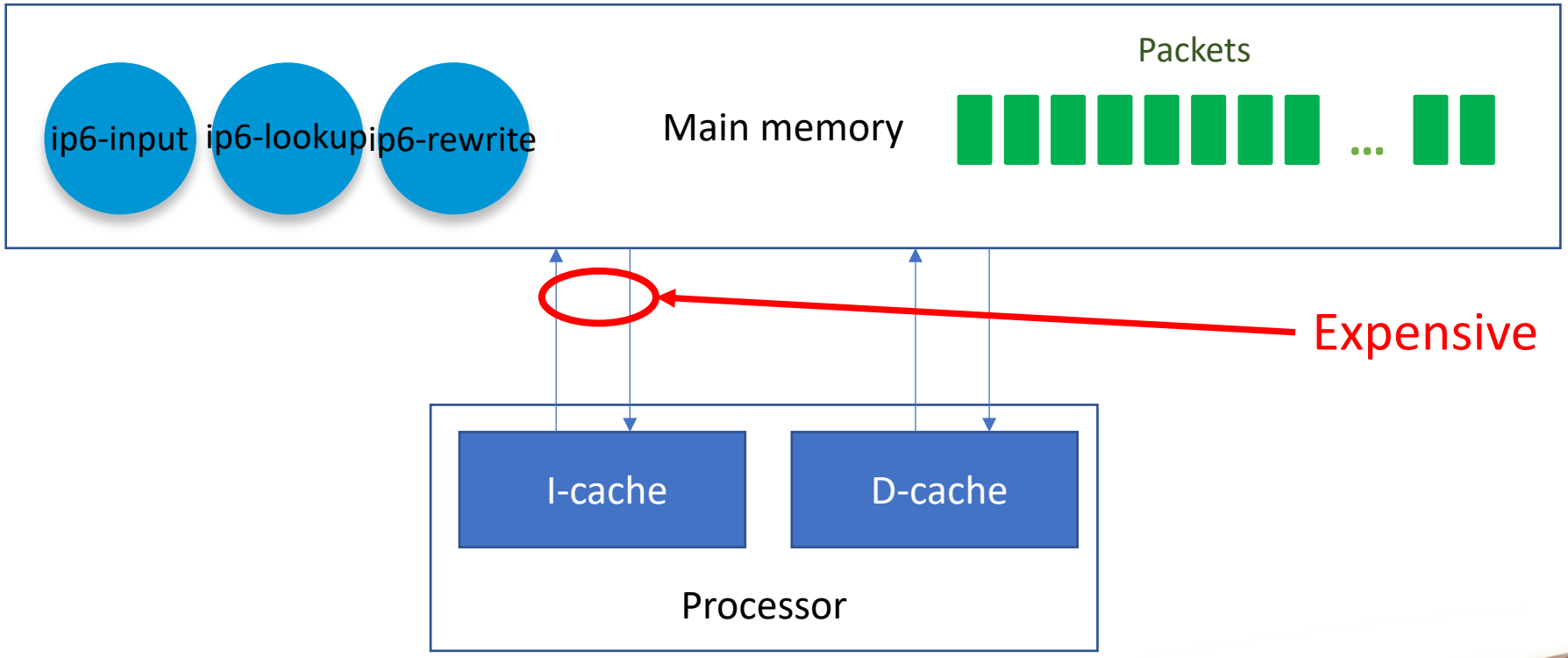
- Cache hit:
  - ~2-30 cycles
- Cache miss (main memory)
  - ~140 cycles

Reduce cache misses



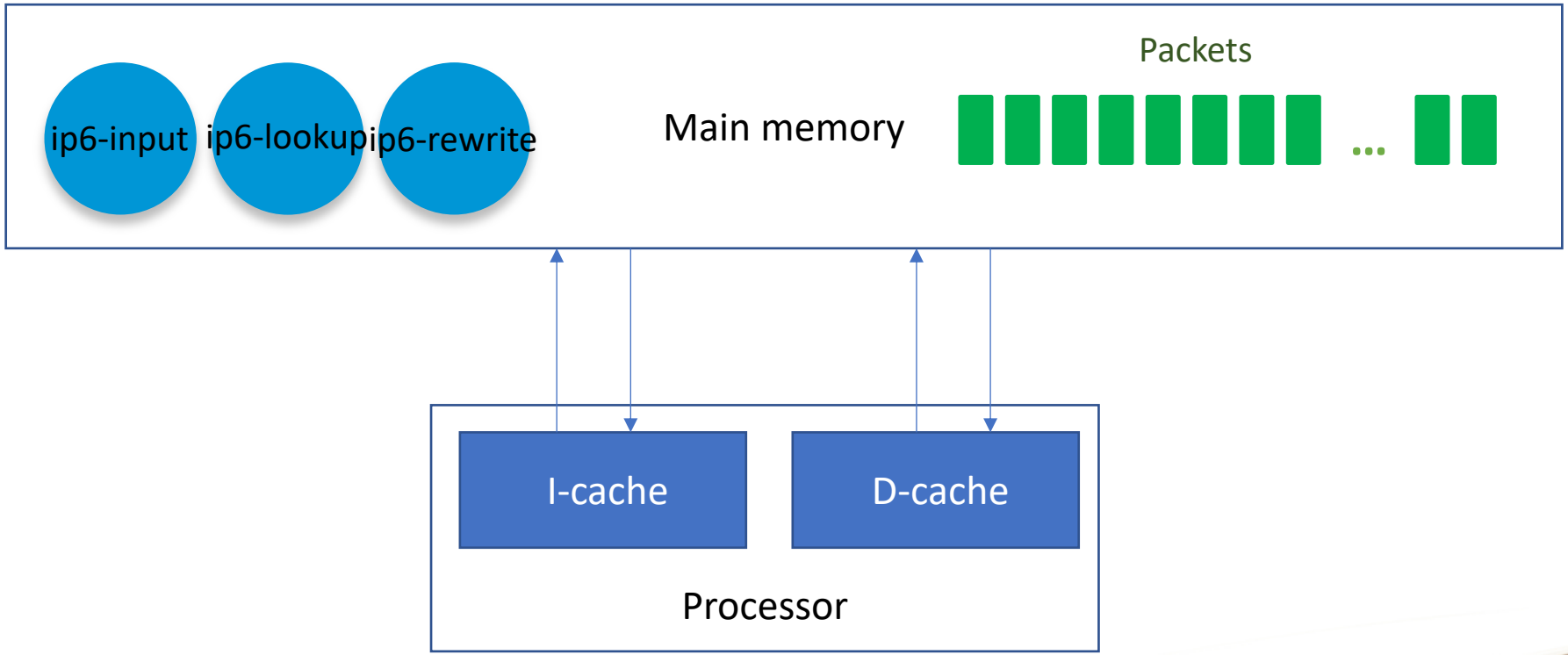
Reduce cache misses – I-cache

Let’s compare scalar processing with vector processing



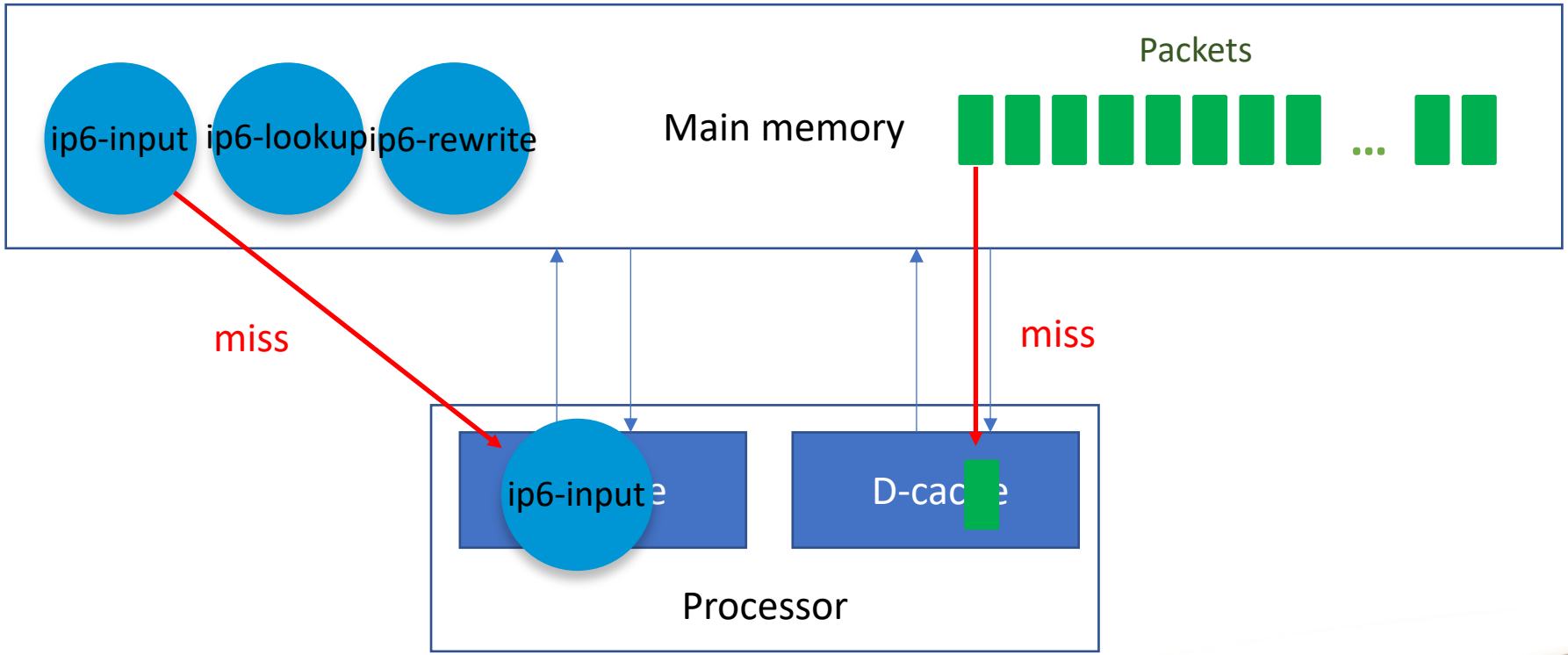
Scalar Packet Processing

- Process one packet at a time



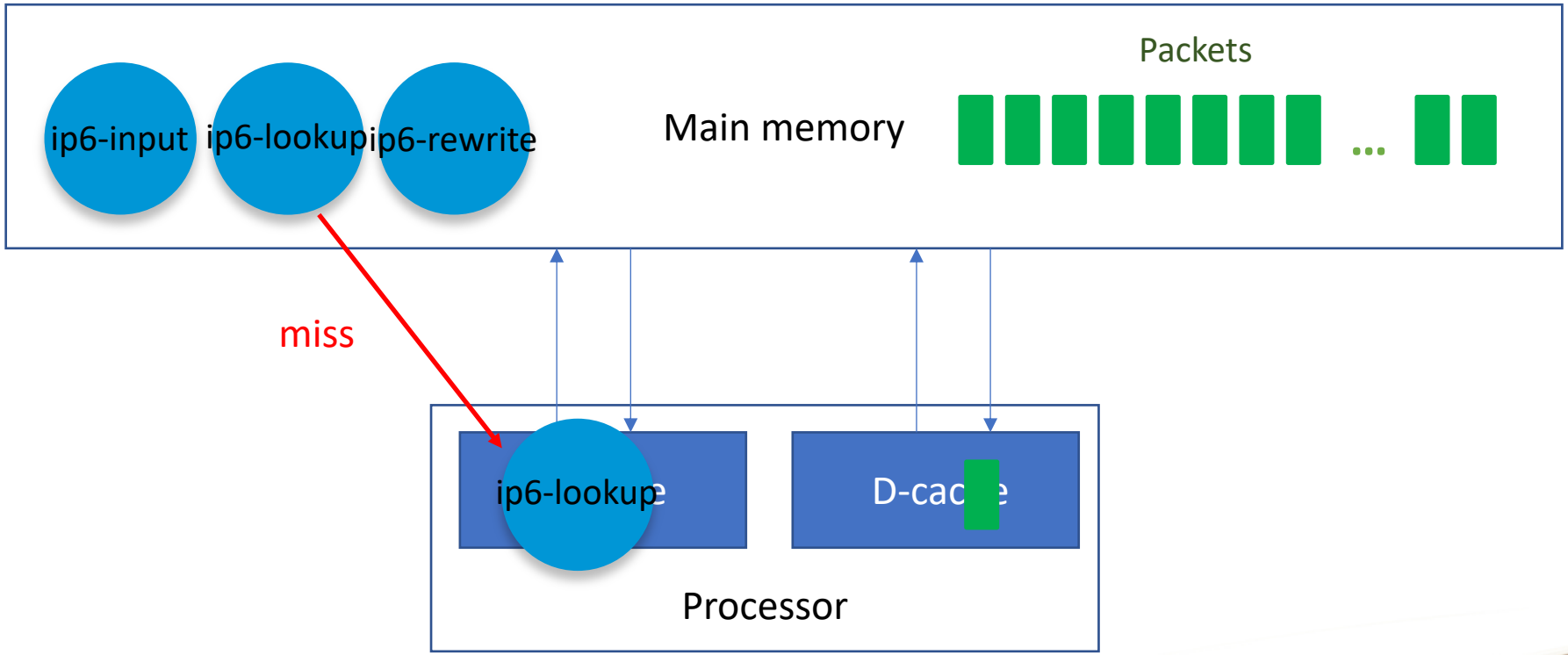
Scalar Packet Processing

- Process one packet at a time



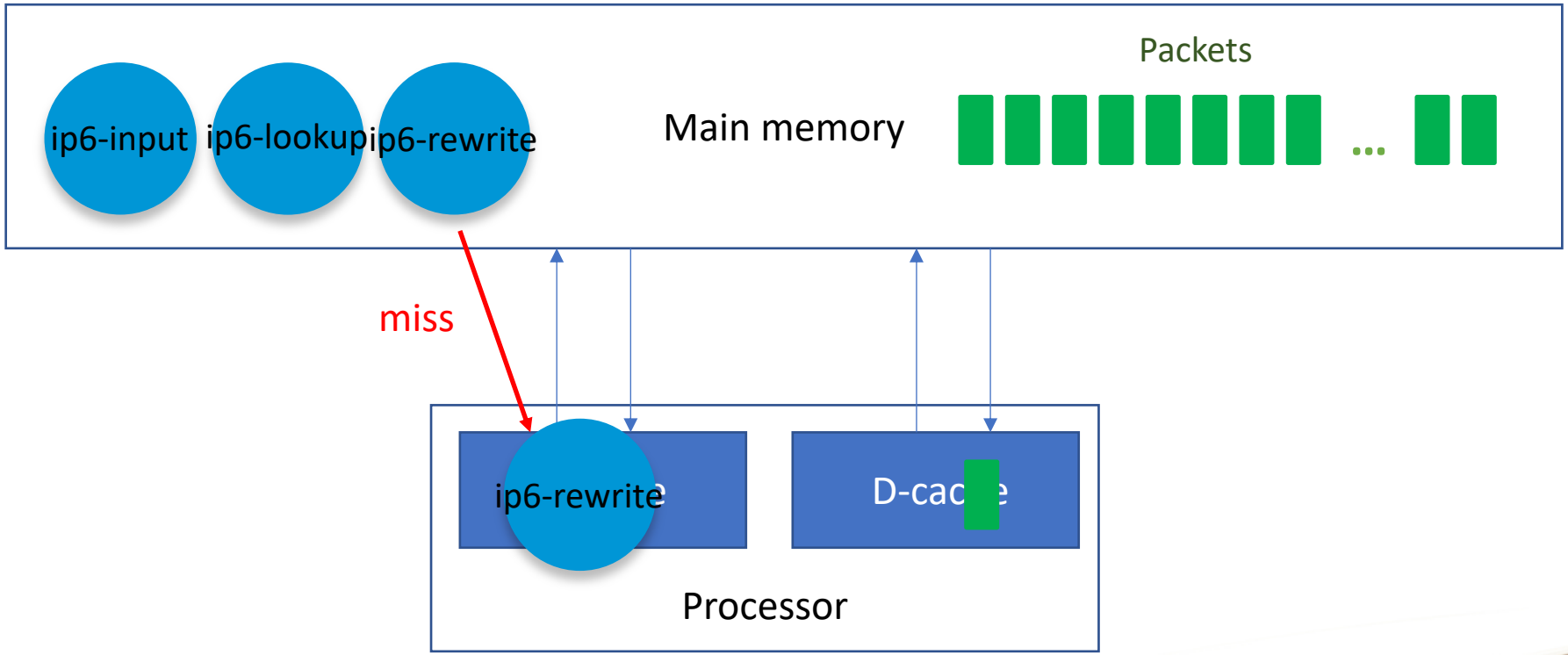
Scalar Packet Processing

- Process one packet at a time



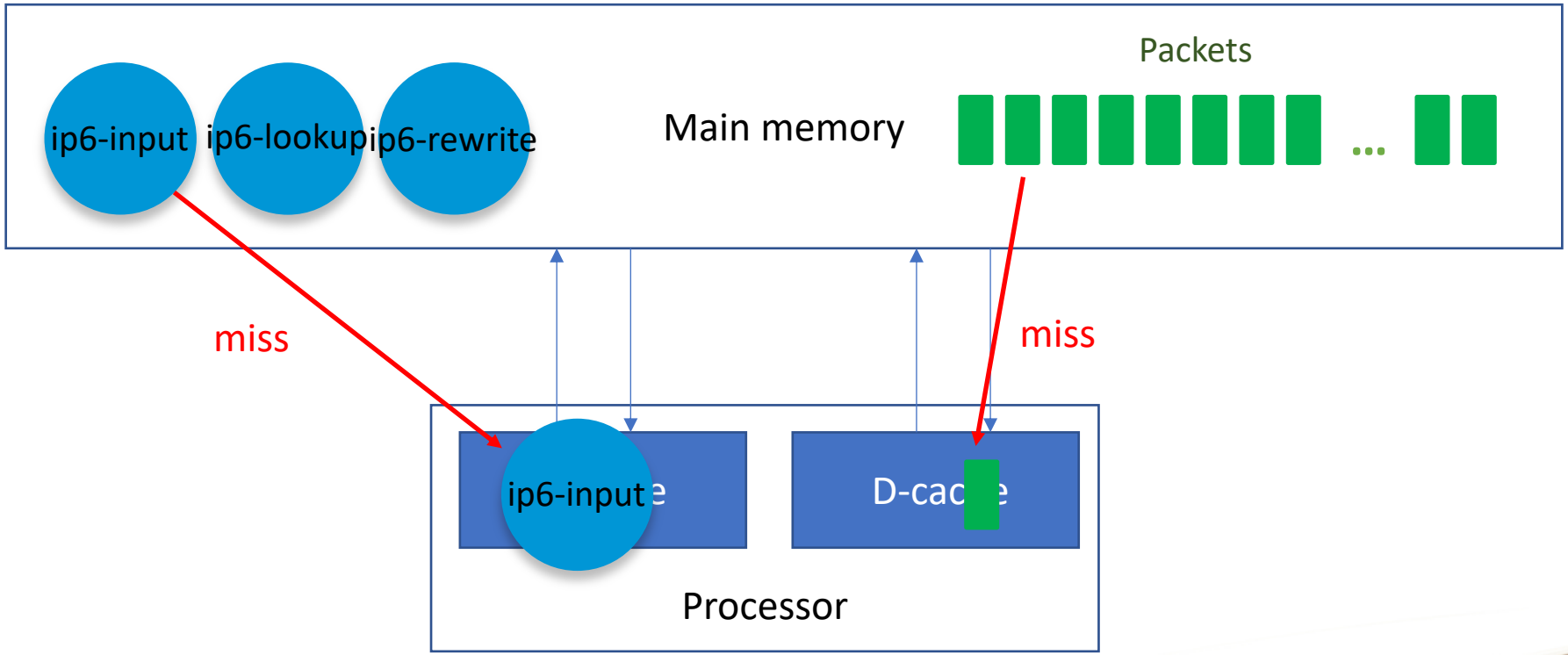
Scalar Packet Processing

- Process one packet at a time



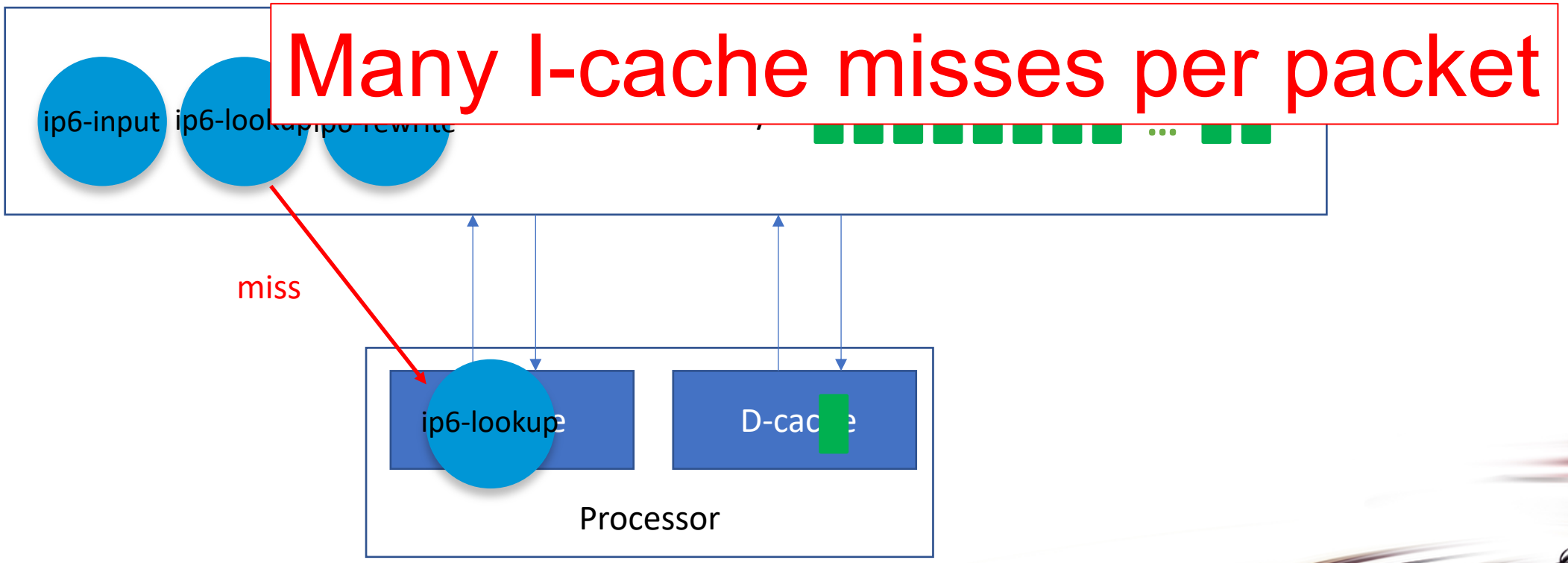
Scalar Packet Processing

- Process one packet at a time



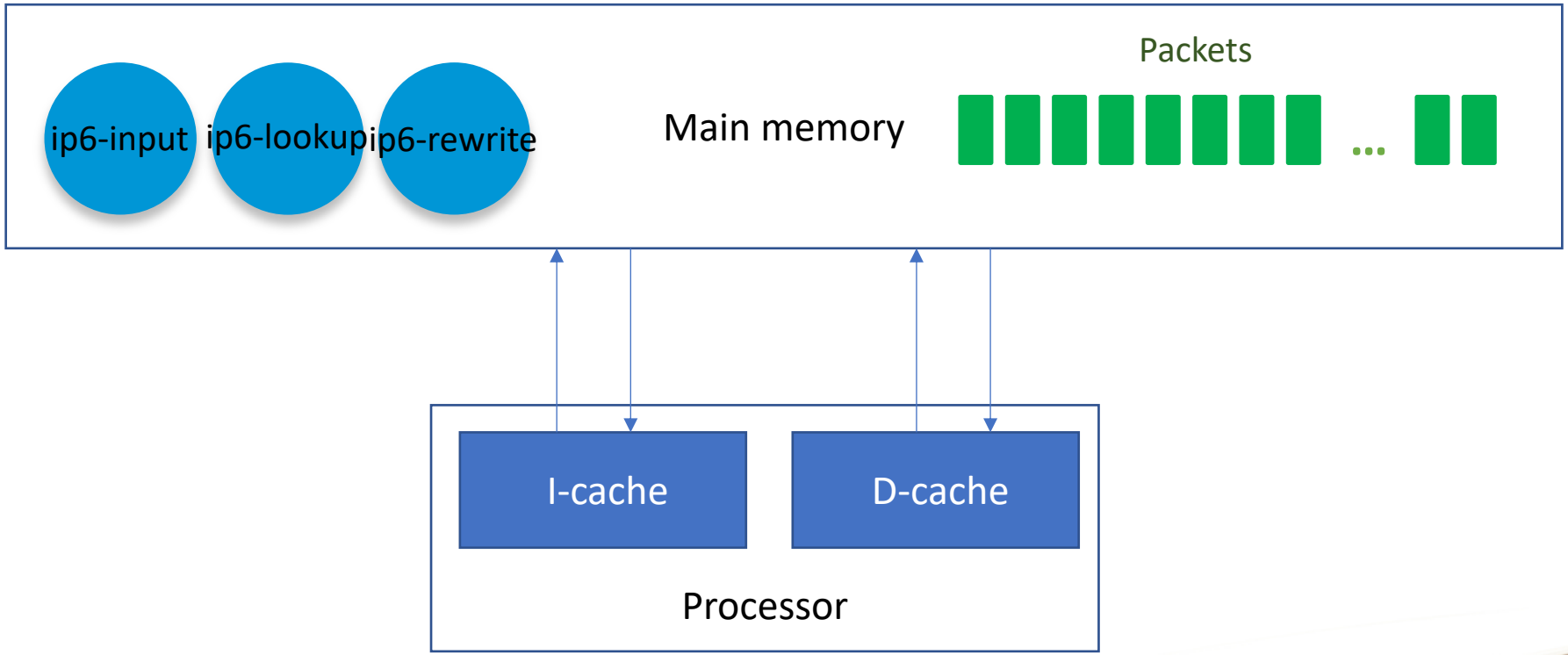
Scalar Packet Processing

- Process one packet at a time



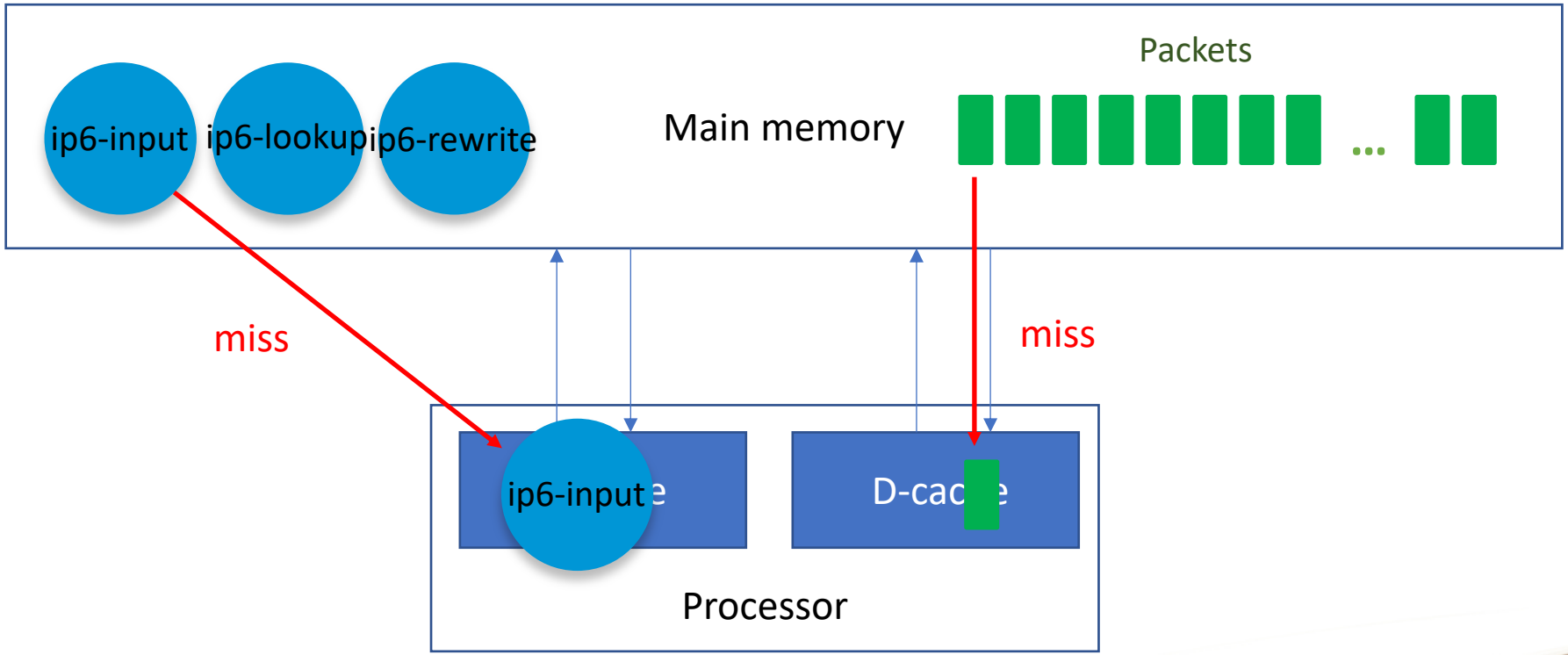
Vector Packet Processing

- Every node process the full packet vector



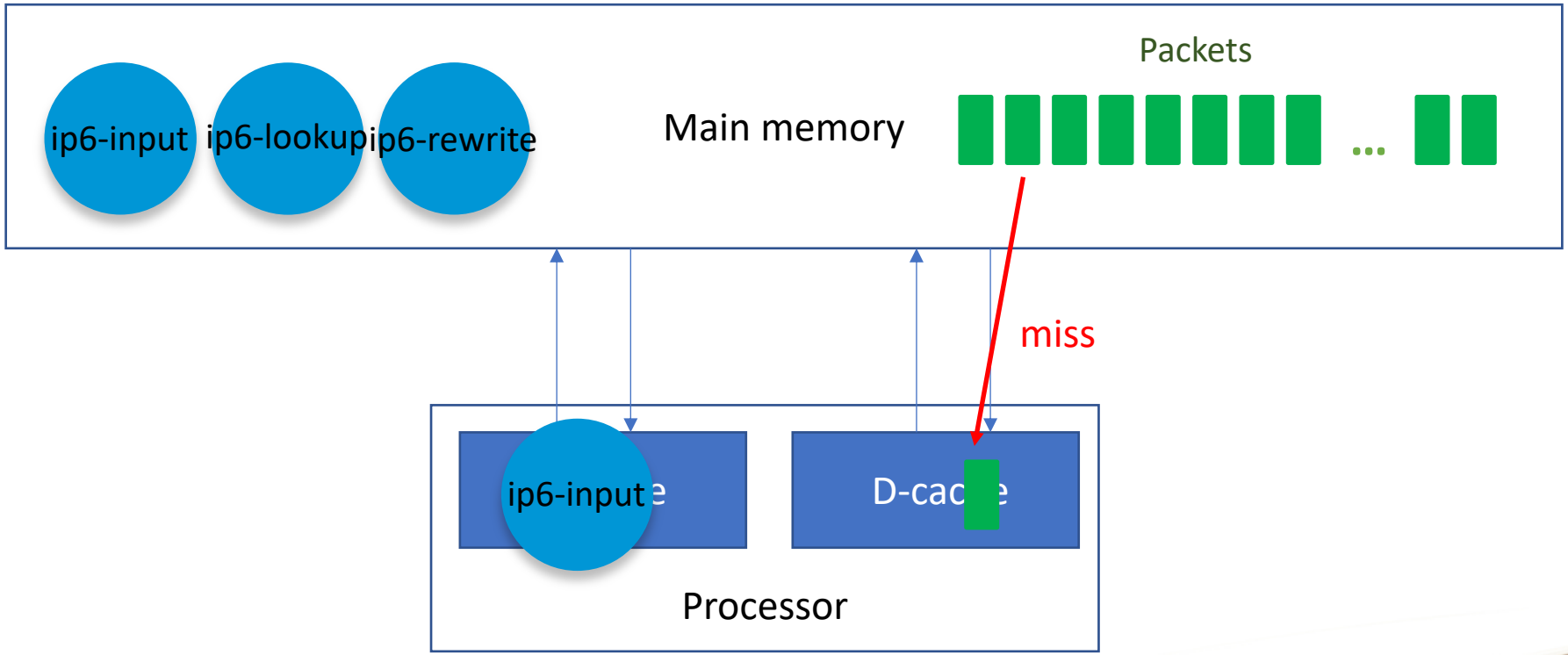
Vector Packet Processing

- Every node process the full packet vector



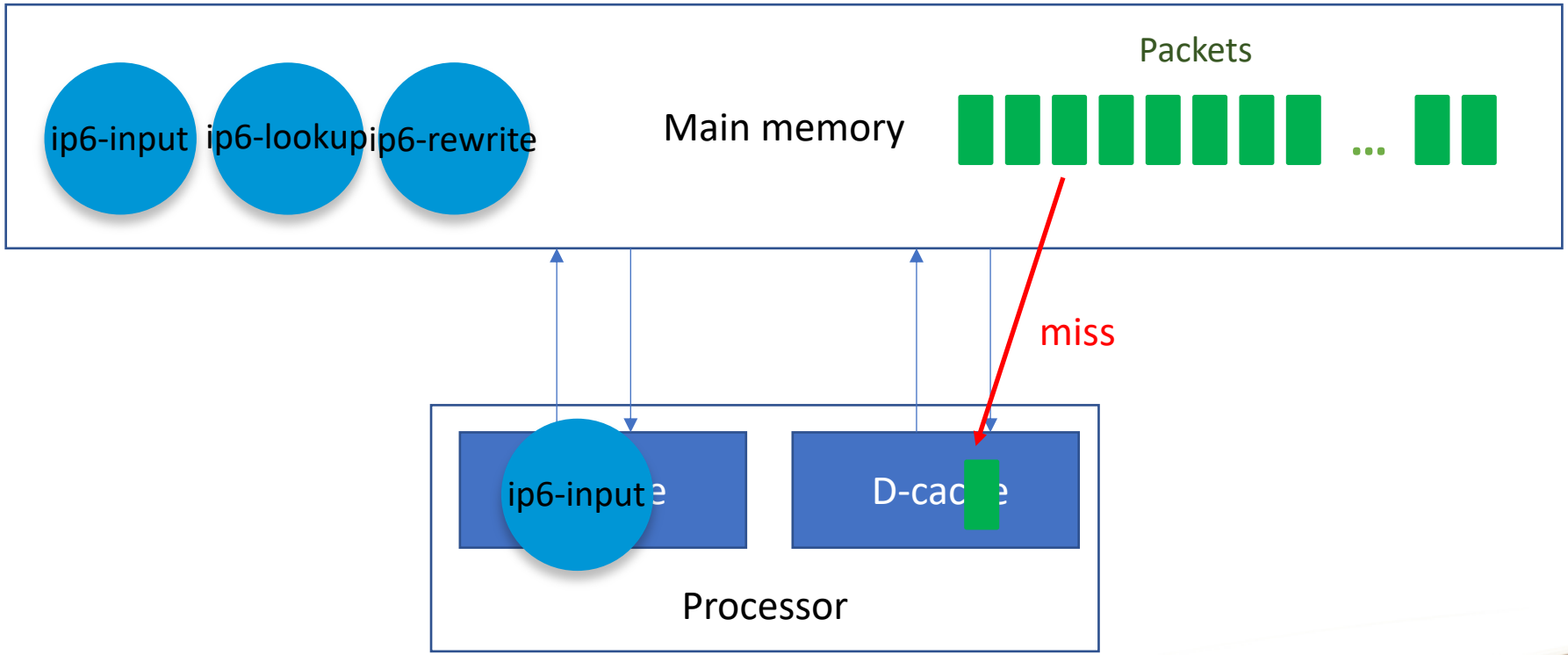
Vector Packet Processing

- Every node process the full packet vector



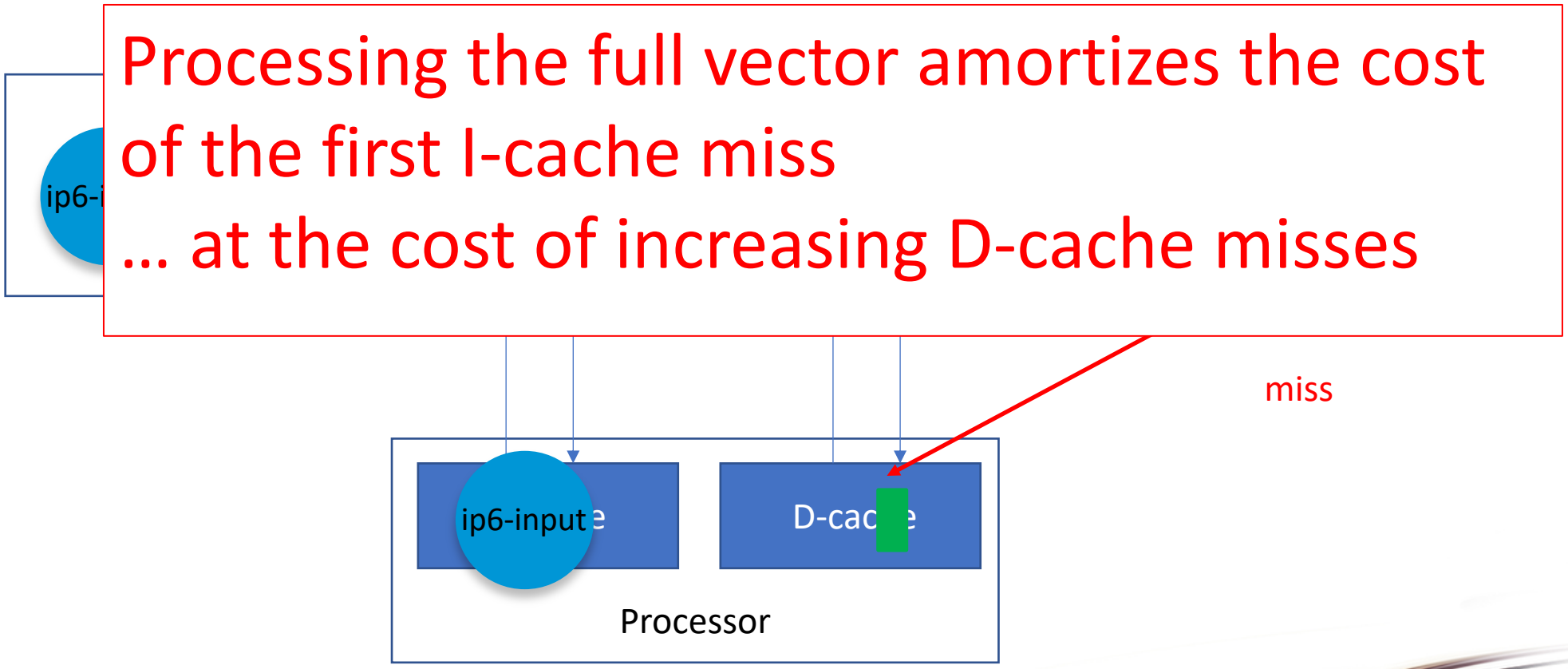
Vector Packet Processing

- Every node process the full packet vector



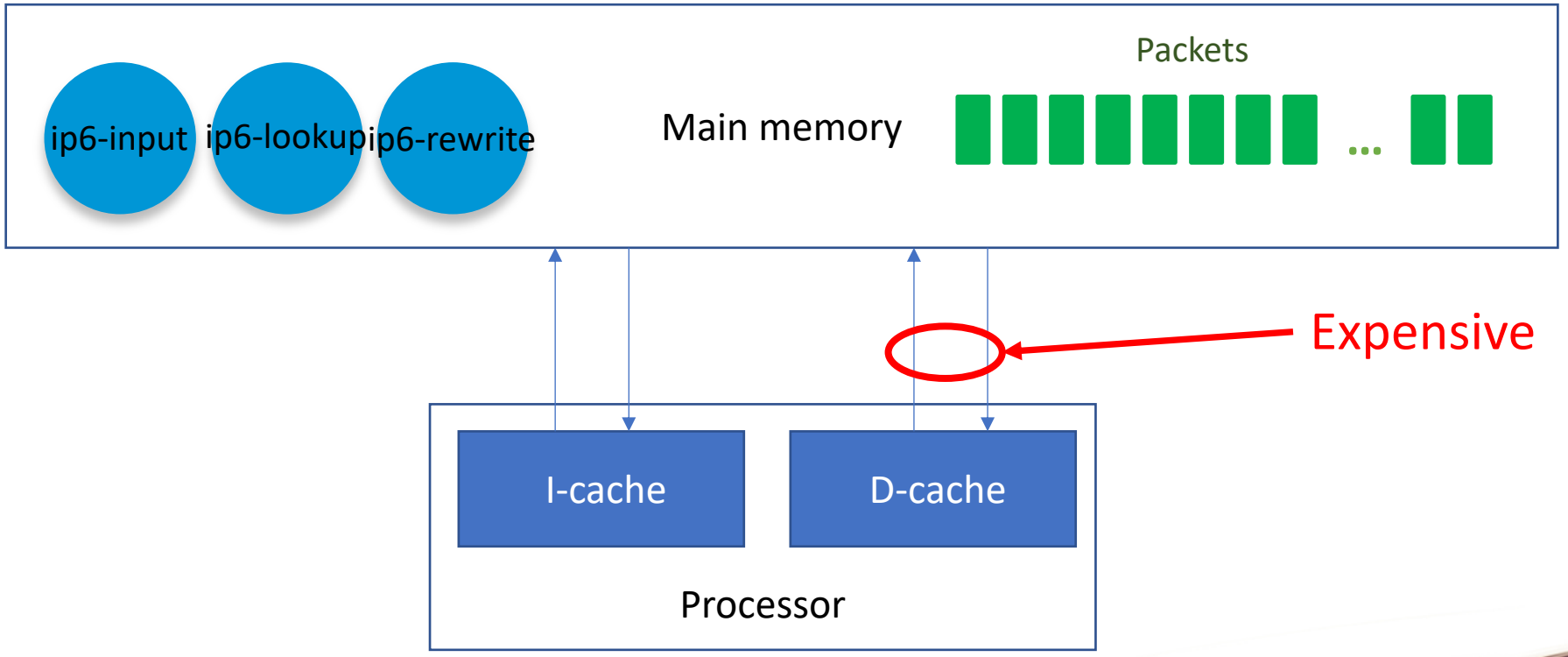
Vector Packet Processing

- Every node process the full packet vector



Reduce cache miss – D-cache

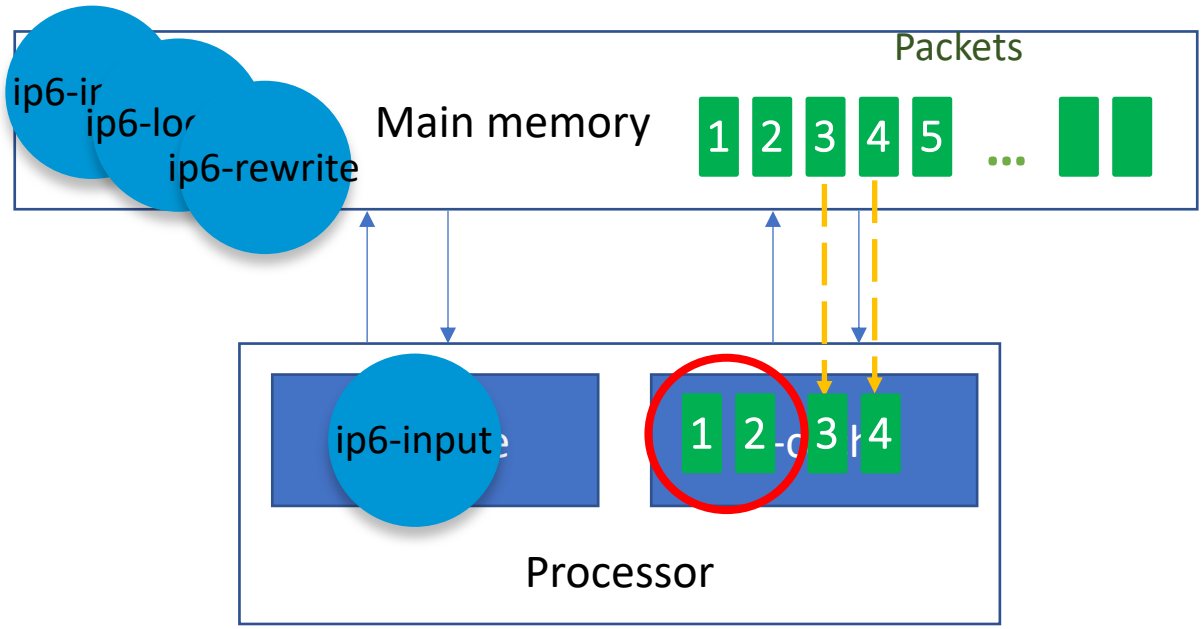
VPP pre-fetches data into D-cache



Reduce cache miss – D-cache

Example: Processing packet 1 & 2

Might have a cache miss for packet 1 & 2



VPP node pseudocode

while packets in vector

while 4 or more packets

PREFETCH #3 and #4

PROCESS #1 and #2

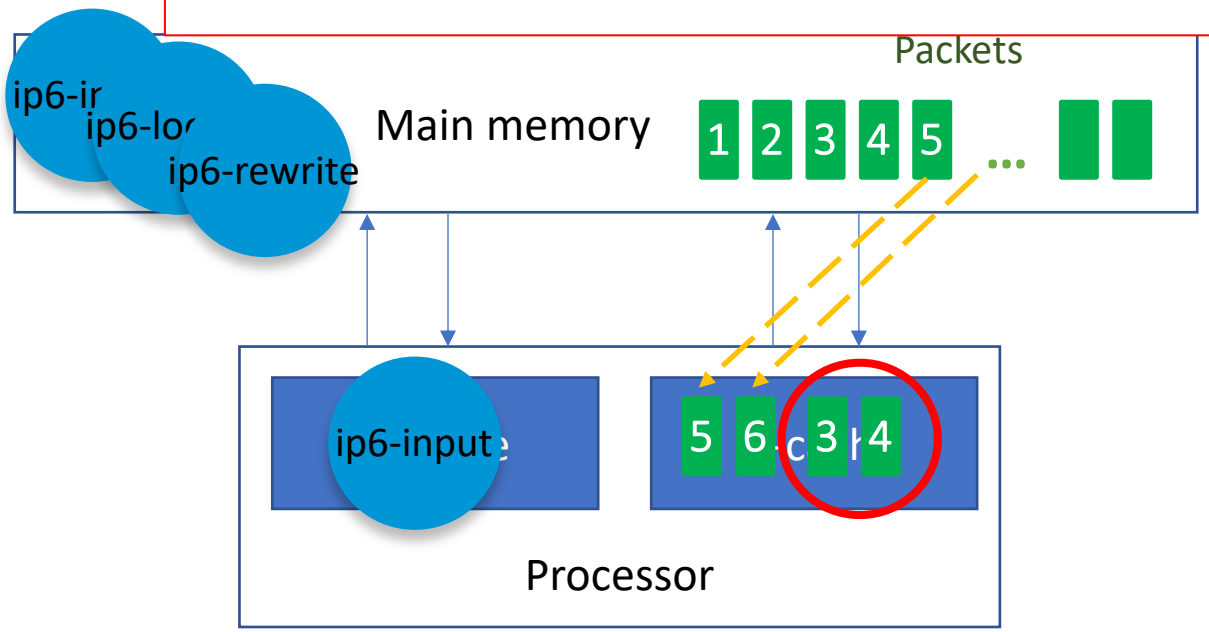
while any packets

<as above but single packet>

Reduce cache miss – D-cache

Example: Processing packet 3 & 4

The cost of the first D-cache miss is amortized by the subsequent D-cache hits.



while 4 or more packets

PREFETCH #5 and #6

PROCESS #3 and #4

while any packets

<as above but single packet>

Hands on VPP!

VPP documentation

- Wiki

https://wiki.fd.io/view/VPP

- Doxygen

https://docs.fd.io/vpp/17.04/

Download VPP (v17.04)

- Clone the source code from git

git clone https://gerrit.fd.io/r/vpp

- Or install it from .deb pkg (rpm for Centos available too)

… see wiki

Configure and Start VPP

- VPP configuration file

emacs /etc/vpp/startup.conf

- Start vpp

sudo vpp -c /etc/vpp/startup.conf

VPP Command Line Interface

- To start a shell:

```
# vppctl
```

- To run one command:

```
# vppctl <command>
```

VPP Command Line Interface

- A bunch of useful commands:
  - ?
  - show
  - set

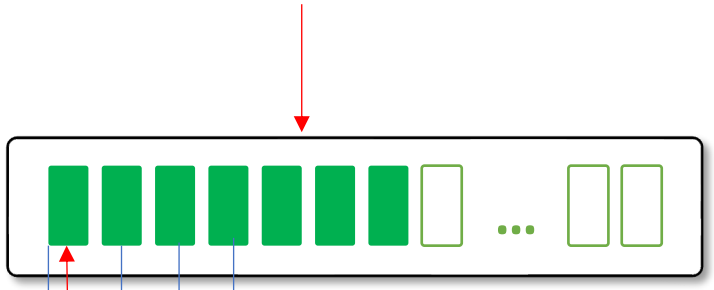
Create your own plugin

Outline

- VPP structures
- Design & Implement your node(s)
- Insert your node(s) in the vlib_graph
- Compile and install your plugin

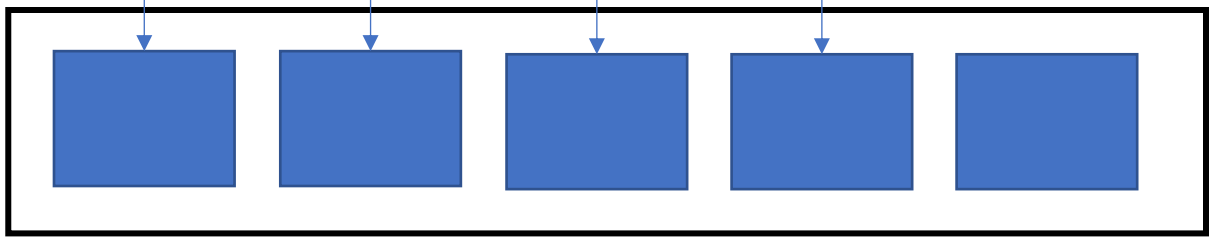
VPP structures

The vector of packets is called FRAME



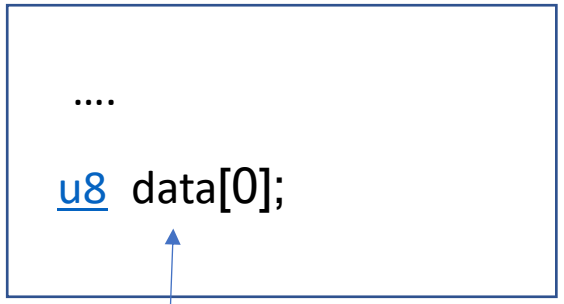
Each element is called VECTOR

A vector is an index to a vlib_buffer_t



Memory holding vlib_buffer_t objects

vlib_buffer_t



Pointer to packet data
(DMA memory)

Outline

- VPP structures
- Design & Implement your node(s)
- Inserting your node(s) in the vlib_graph
- Compiling and installing your plugin

Design & Implement your node(s)

- Your node should follow VPP style
  - Multi-loop, Branch prediction, Function flattening, Lock-free structures
- A node must implement a processing function that
  - “Moves vectors” from your node’s frame to the next node’s frame
  - Processes packets as YOU want
- Add whatever else you need
  - Supporting Functions, macros, variables, etc.. (C code)

Register your node(s) to VPP

- Each node must be registered to VPP through VLIB_REGISTER_NODE macro

```
#define VLIB_REGISTER_NODE (  x,
                              ...
                            )
```

Value:

```
__VA_ARGS__ vlib_node_registration_t x;                                  \
static void __vlib_add_node_registration_##x (void)                      \
    __attribute__((__constructor__)) ;                                   \
static void __vlib_add_node_registration_##x (void)                      \
{                                                                        \
    vlib_main_t * vm = vlib_get_main();                                  \
    x.next_registration = vm->node_main.node_registrations;              \
    vm->node_main.node_registrations = &x;                               \
}                                                                        \
__VA_ARGS__ vlib_node_registration_t x
```

Definition at line **143** of file **node.h**.

```
typedef struct _vlib_node_registration
{
  /* Vector processing function for this node. */
  vlib_node_function_t *function;

  /* Node name. */
  char *name;

  /* Name of sibling (if applicable). */
  char *sibling_of;

  /* Node index filled in by registration. */
  u32 index;

  /* Type of this node. */
  vlib_node_type_t type;

  /* Error strings indexed by error code for this node. */
  char **error_strings;
```

Example: Cicn plugin

```
VLIB_REGISTER_NODE (icnfwd_node) =
{
  .function = icnfwd_node_fn,
  .name = "icnfwd",
  .vector_size = sizeof (u32),
  .runtime_data_bytes = sizeof (icnfwd_runtime_t),
  .format_trace = icnfwd_format_trace,
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = ARRAY_LEN (icnfwd_error_strings),
  .error_strings = icnfwd_error_strings,
  .n_next_nodes = ICNFWD_N_NEXT,
  .next_nodes = {
    [ICNFWD_NEXT_LOOKUP] = "ip4-lookup",
    [ICNFWD_NEXT_ERROR_DROP] = "error-drop",
  }
,};
```

Node processing function

Name of the node

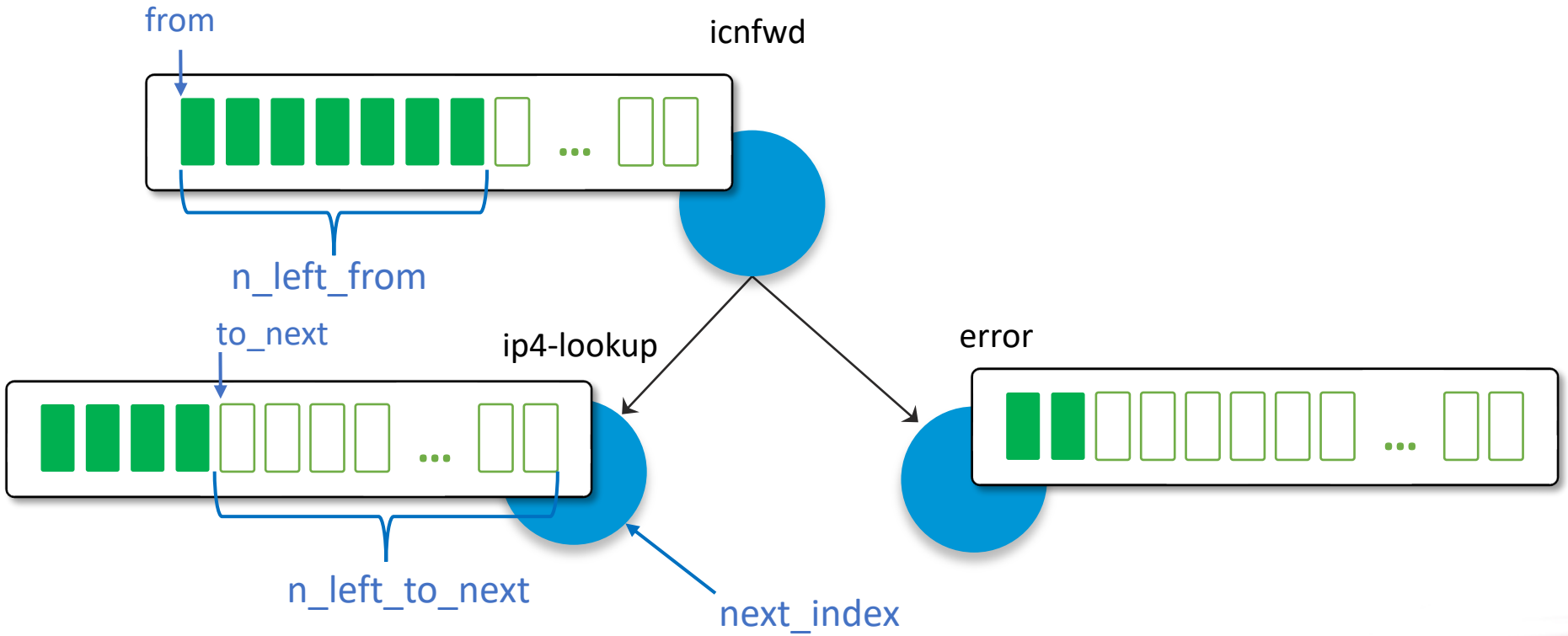
Runtime structure
You need to initialize it by yourself

Type of node

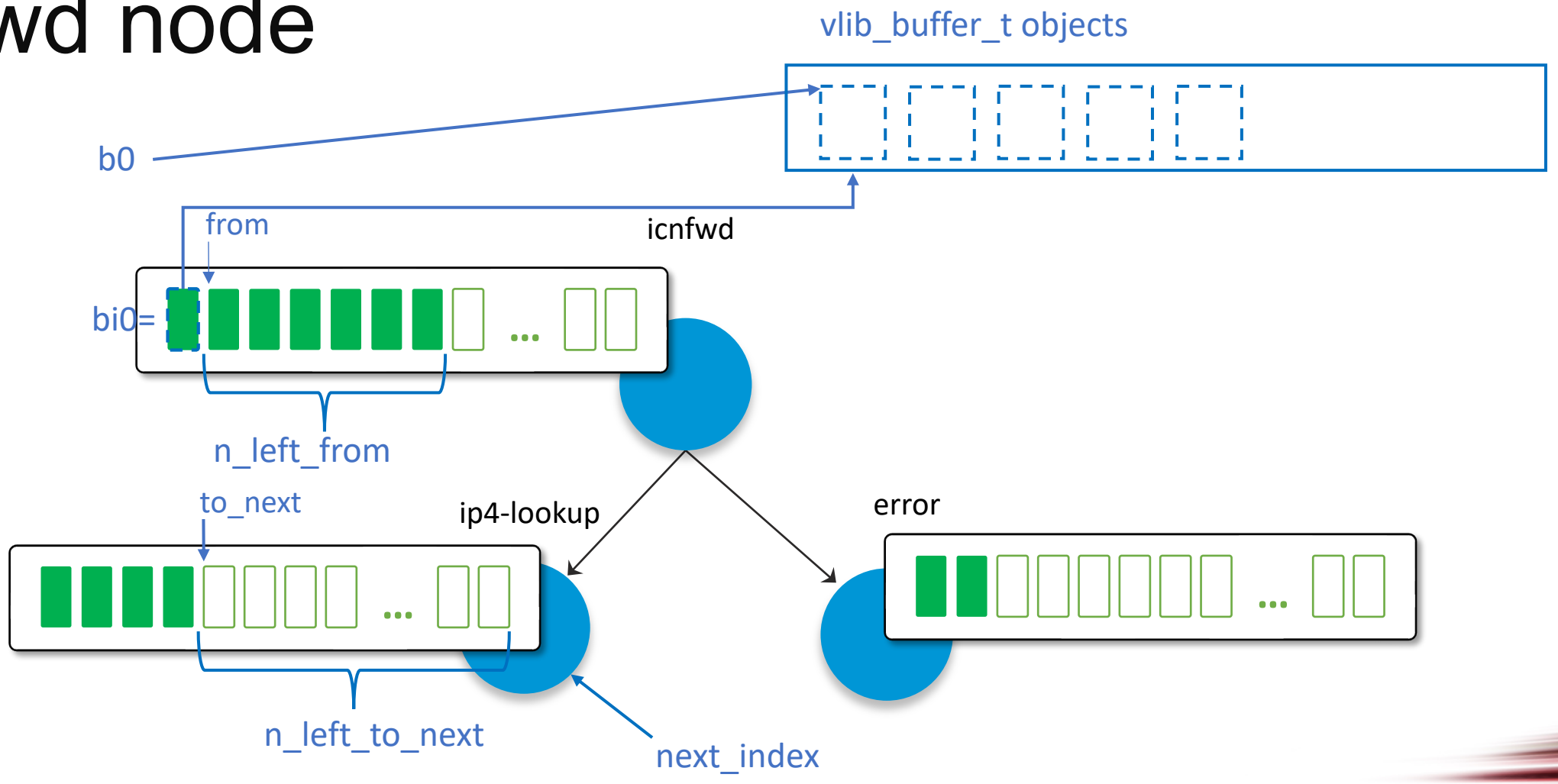
Next nodes in the Vpp graph

Let's take a look to icnfwd_node_fn

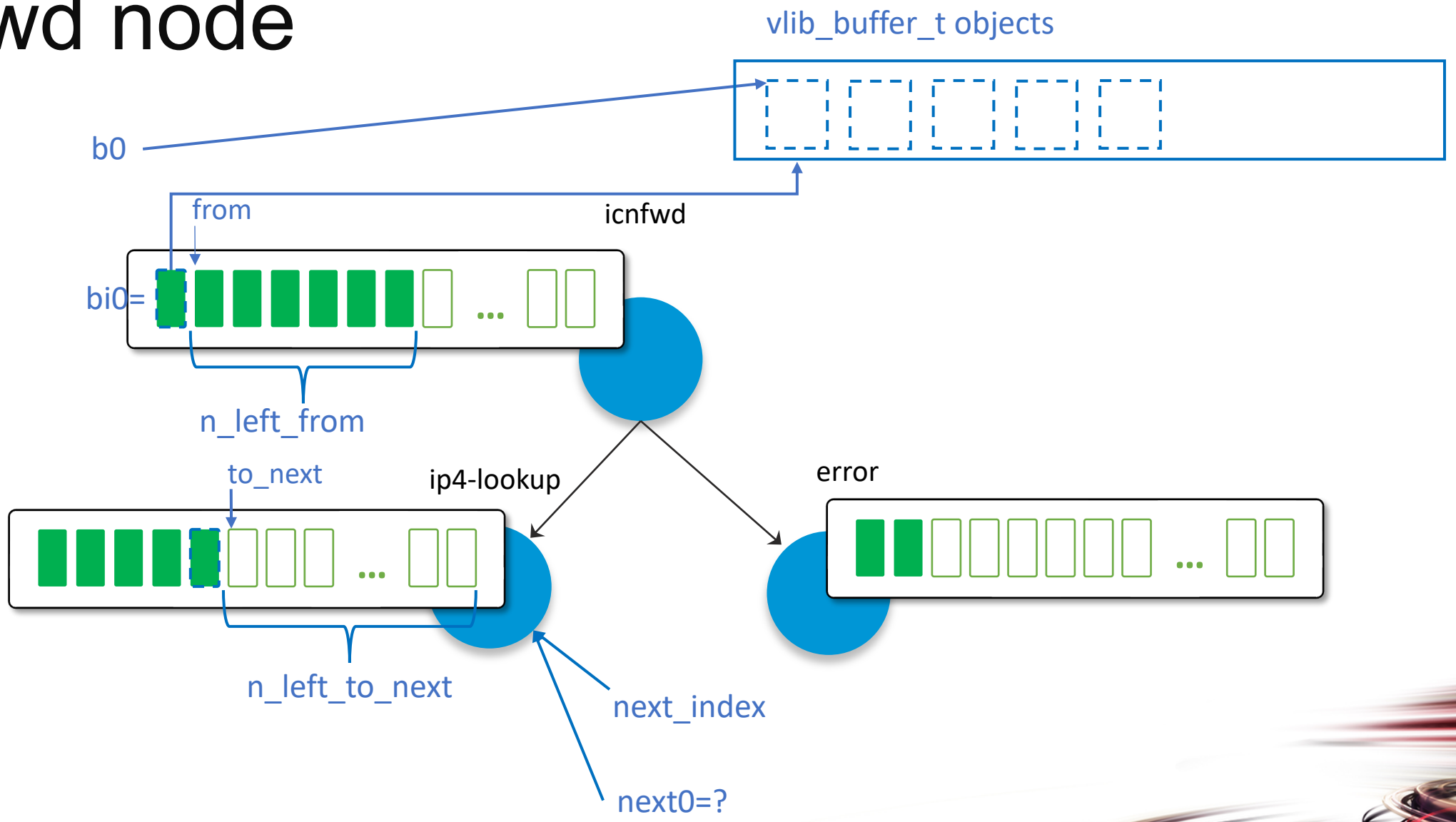
icnfwd node


from
icnfwd
n_left_from
to_next
ip4-lookup
error
n_left_to_next
next_index
next0=?
Hold the actual next node id

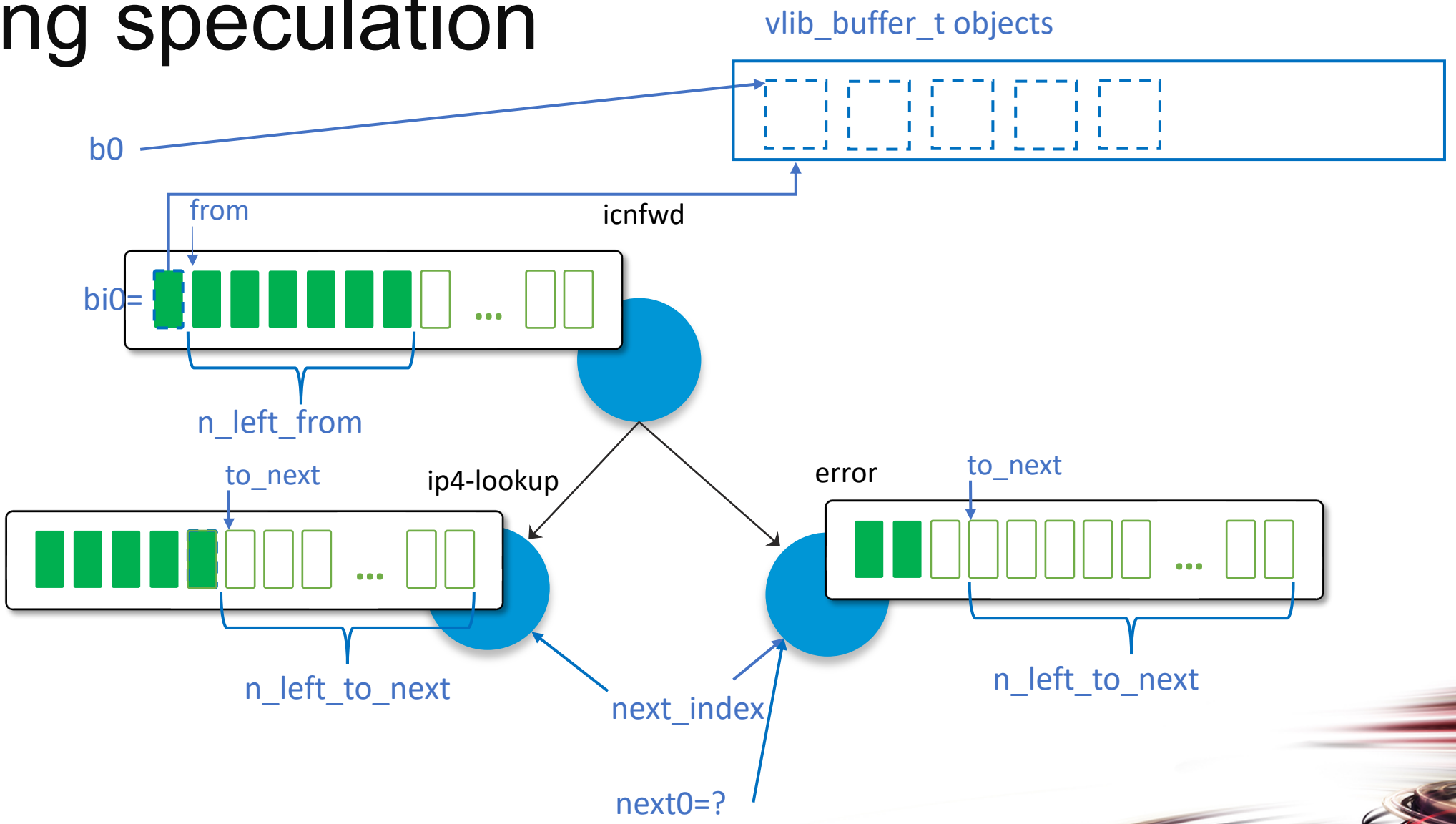
icnfwd node


vlib_buffer_t objects
b0
from
icnfwd
bi0=
n_left_from
to_next
ip4-lookup
error
n_left_to_next
next_index
next0=?

icnfwd node


vlib_buffer_t objects
b0
from
icnfwd
bi0=
n_left_from
to_next
ip4-lookup
error
n_left_to_next
next_index
next0=?

Wrong speculation



Example: Cicn plugin

```
VLIB_REGISTER_NODE (icnfwd_node) =
{
  .function = icnfwd_node_fn,
  .name = "icnfwd",
  .vector_size = sizeof (u32),
  .runtime_data_bytes = sizeof (icnfwd_runtime_t),
  .format_trace = icnfwd_format_trace,
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = ARRAY_LEN (icnfwd_error_strings),
  .error_strings = icnfwd_error_strings,
  .n_next_nodes = ICNFWD_N_NEXT,
  .next_nodes = {
    [ICNFWD_NEXT_LOOKUP] = "ip4-lookup",
    [ICNFWD_NEXT_ERROR_DROP] = "error-drop",
  }
,};
```

← Errors handling (counters)

Other important macros

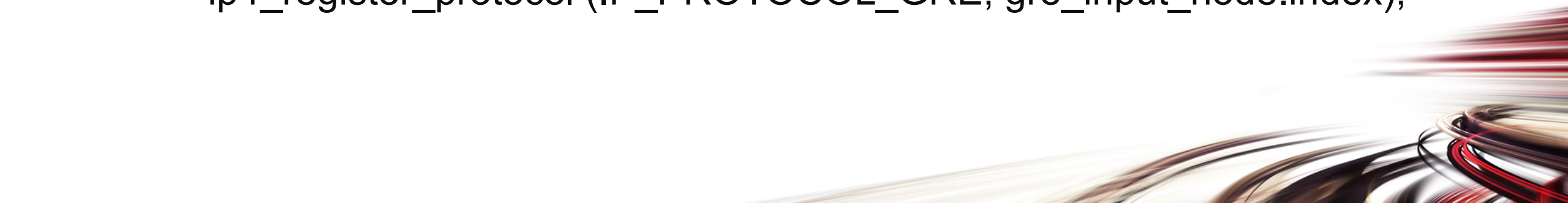
- VPP_INIT_FUNCTION
  - Function that is called during VPP initialization
- VPP_REGISTER_PLUGIN
  - Required to guarantee that your plugin is actually a VPP plugin
    …and not a library copied by mistake in /usr/lib/vpp_plugins

Outline

- VPP structures
- Design & Implement your node(s)
- Insert your node(s) in the vlib_graph
- Compile and install your plugin

Insert your node to VPP graph

1. direct all the packets from one interface
   - vnet_hw_interface_rx_redirect_to_node (vnet_main, hw_if_index, my_graph_node.index /* redirect to my_graph_node */);
2. capture packets with a particular ethertype
   - ethernet_register_input_type (vm, ETHERNET_TYPE_CDP, cdp_input_node.index);
3. for-us packet for new protocol on top of IP
   - ip4_register_protocol (IP_PROTOCOL_GRE, gre_input_node.index);



Insert your node to VPP graph

4. ip-for-us packet sent to a specific UDP port
   - udp_register_dst_port (vm, UDP_DST_PORT_vxlan, vxlan_input_node.index, 1 /* is_ip4 */);

5. direct all packets from one ip prefix
   - Create your own Data Path Object (i.e. result of a FIB lookup)

Outline

- VPP structures
- Design & Implement your node(s)
- Insert your node(s) in the vlib_graph
- Compile and install your plugin

Compiling your plugin

- VPP provides Automake/Autoconf examples
  - Install vpp-dev and move to /usr/share/doc/vpp/examples

- Adapting Makefile.am and sample.am is trivial

- Compile cicn-plugin:

```
$ cd cicn-plugin
$ autoreconf -i -f
$ mkdir -p build
$ cd build
$ ../configure --with-plugin-toolkit
OR, to omit UT code
$ ../configure --with-plugin-toolkit  --without-cicn-test
$ make
$ sudo make install
```

vICN: configuration, management and control of an virtual ICN network

Marcel Enguehard

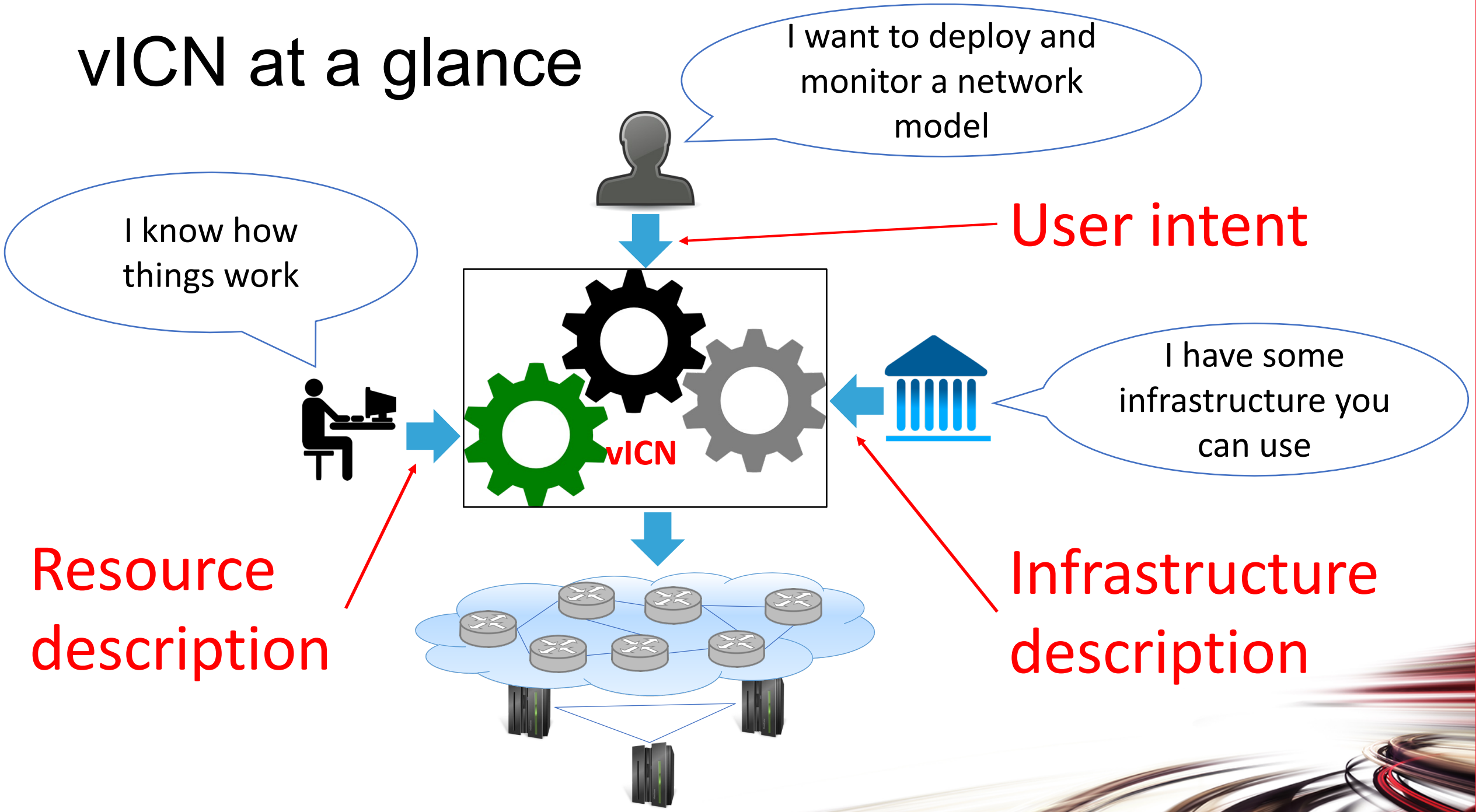
ACM ICN Conference – CICN tutorial

September 26th 2017

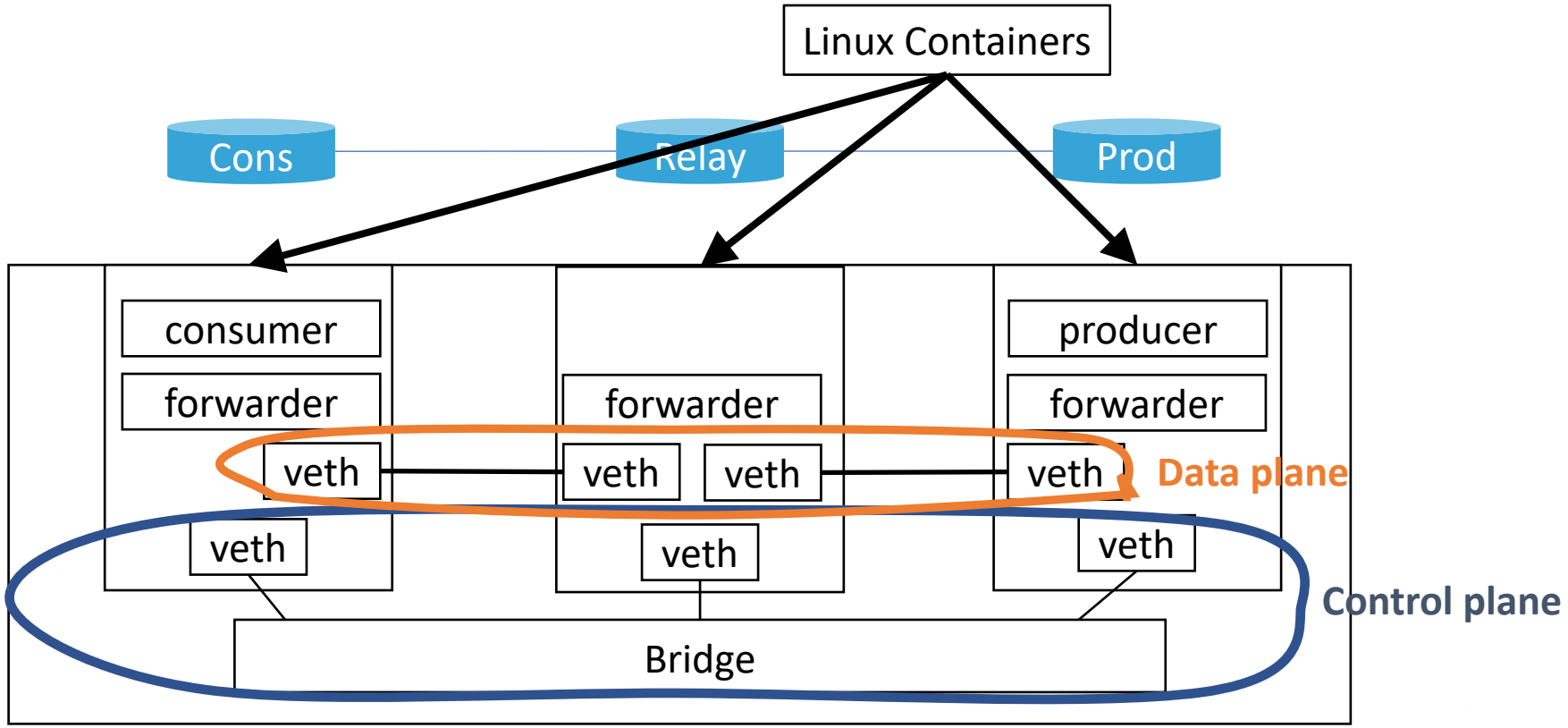
What is vICN

- Unified framework for network deployment, management and monitoring
- Integrates all the tools of the CICN fd.io suite
- Provides an API to easily to bootstrap ICN deployments and get meaningful telemetry out of it

vICN at a glance



Example vICN topology



vICN resources

Class

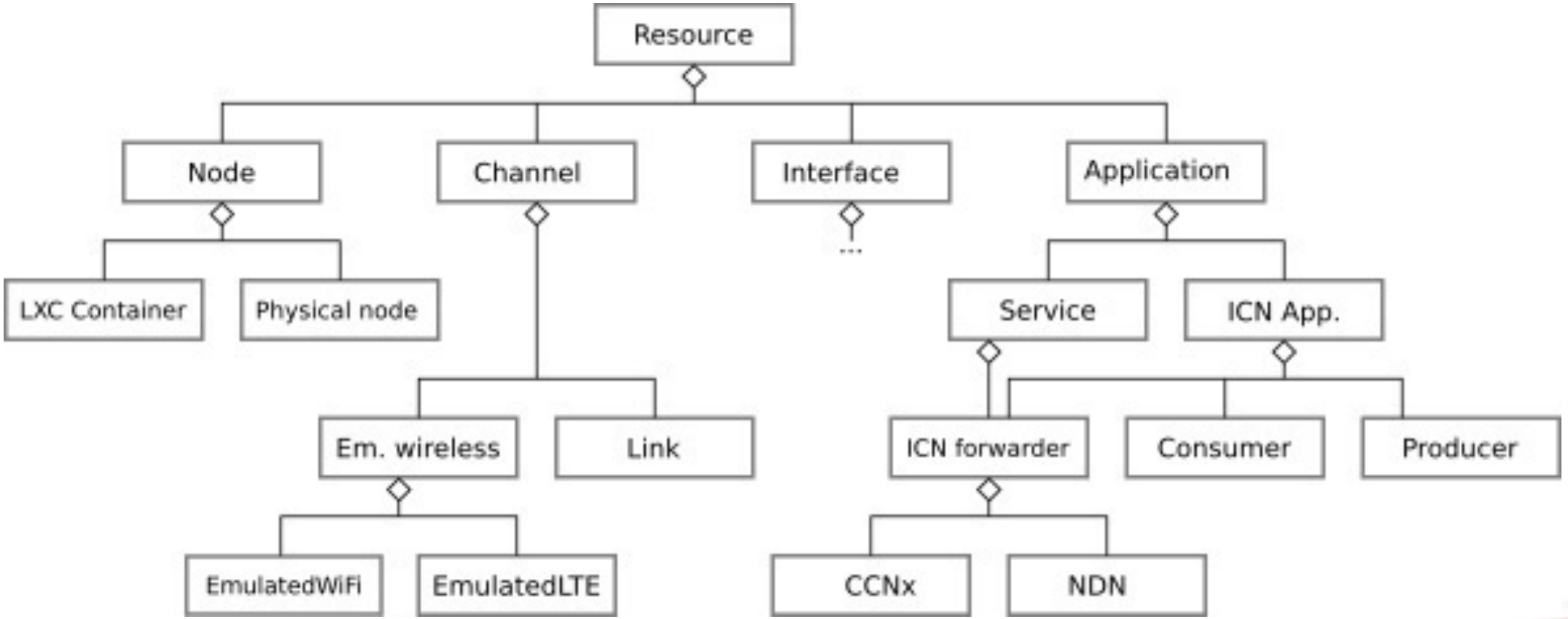
- Virtual representation of deployment element
- Node, forwarder, application, link, etc.
- Described by *attributes*

Members

Example resource: forwarder

- Represents an ICN forwarder
- Attributes:
  - node
  - cache_size
  - cache_policy (e.g., LRU)
  - log_file
  - etc.

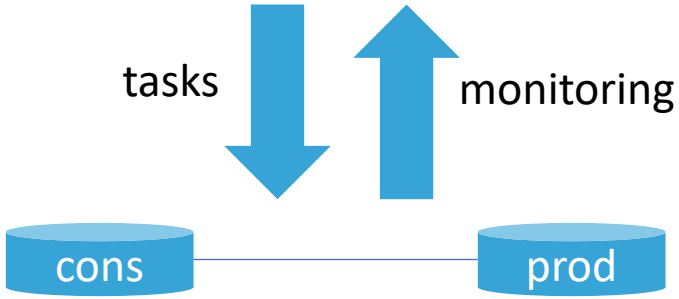
Resource hierarchy



How does it work?

- Intent based-framework
- Object-based model
- State reconciliation between model and deployment

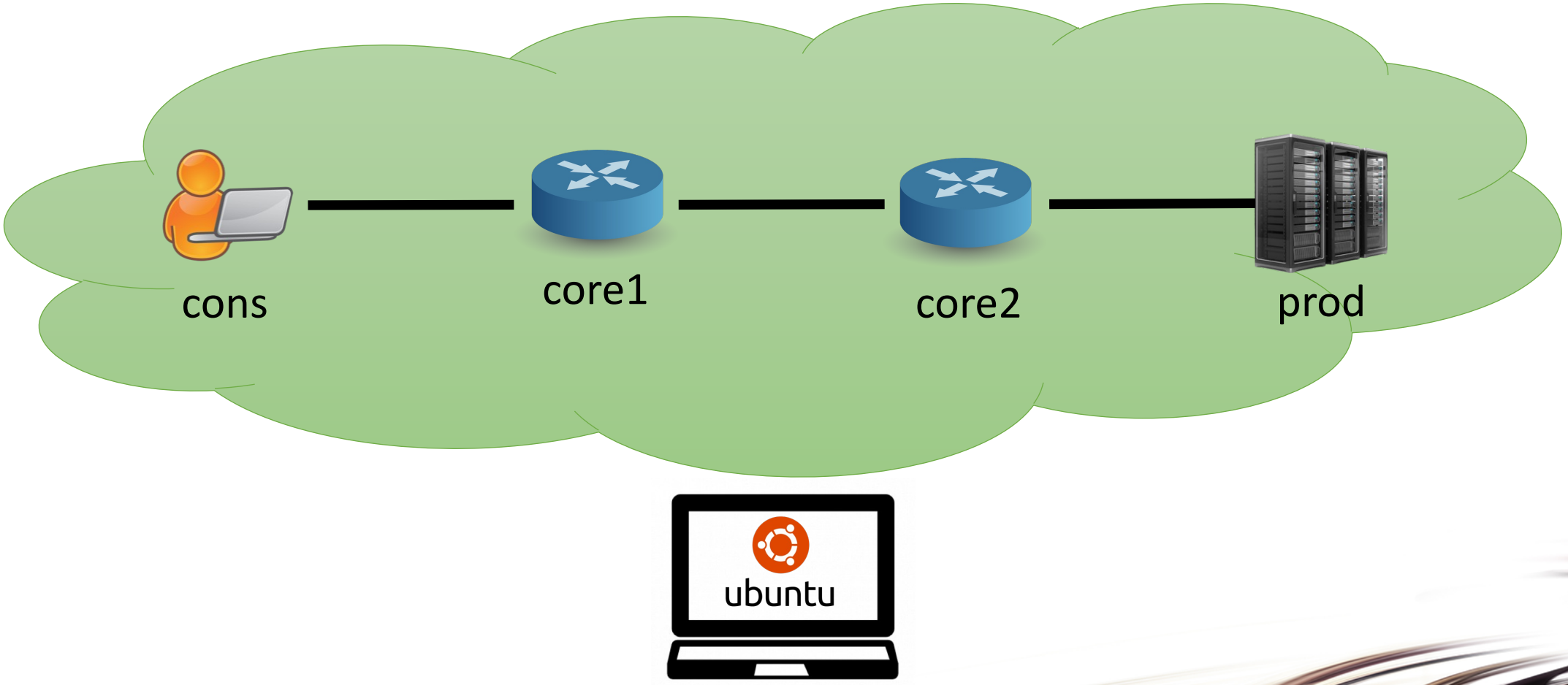
```
cons = LxcContainer()
prod = LxcContainer()
link = Link(src=cons, dst=prod)
```



vICN functionalities

- Multithreaded deployment of network models
- SDN controller for IPv4, IPv6, and ICN
- Wireless links emulation
- Connection of real devices
- Built-in monitoring through Python model

Our example deployment



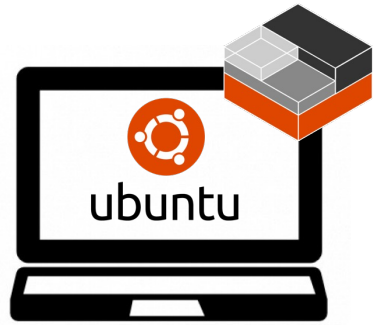
Network model deployment

Network model declaration

- JSON file containing list of resources
- Resources complemented with “key” attributes
- Intent-based declaration: descriptive approach (not imperative)

Physical resources

```
{
   "type": "Physical",
   "name": "server",
   "hostname": "localhost"
},
{
   "type": "LxcImage",
   "name": "cicn-image",
   "node": "server",
   "image": "ubuntu1604-cicnsuite-rc3"
}
```

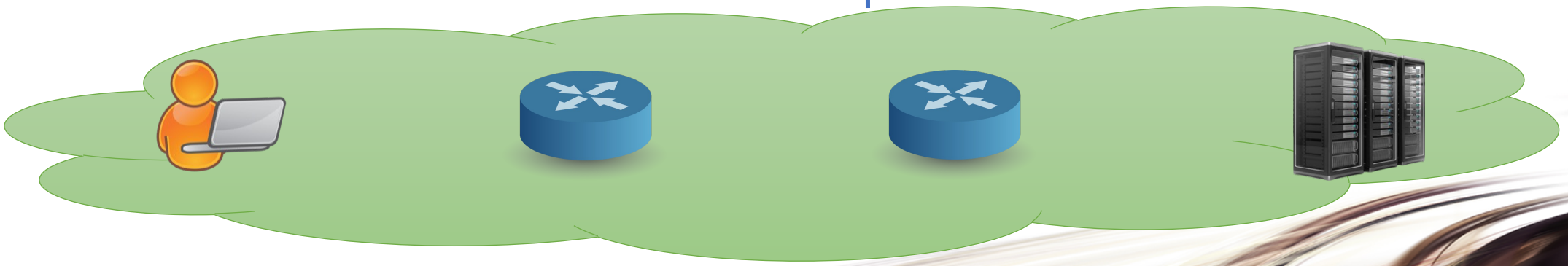


Nodes

Refer topology previous resources

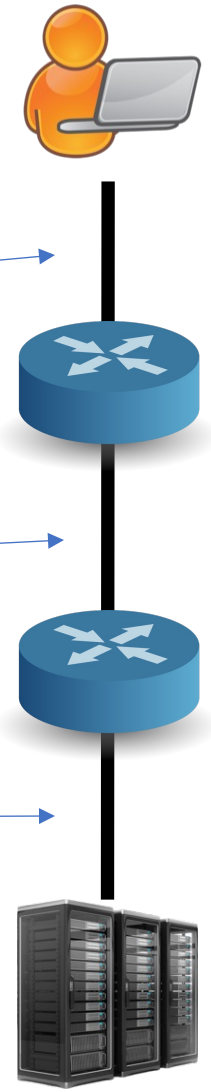
```
{
        "type" : "LxcContainer",
        "image": "cicn-image",
        "name" : "cons",
        "groups": [ "virtual" ],
        "node" : "server"
},
{
        "type" : "LxcContainer",
        "image": "cicn-image",
        "name" : "prod",
        "groups": [ "virtual" ],
        "node" : "server"
},
```

```
{
        "type" : "LxcContainer",
        "image": "cicn-image",
        "name" :  "core1",
        "groups": [ "virtual" ],
        "node" : "server"
},
{
        "type" : "LxcContainer",
        "image": "cicn-image",
        "name" : "core2",
        "groups": [ "virtual" ],
        "node" : "server"
},
```



Links

```
{
    "type": "Link",
    "src_node": "cons",
    "dst_node": "core1",
    "groups": [ "virtual" ]
},
{
    "type": "Link",
    "src_node": "core1",
    "dst_node": "core2",
    "groups": [ "virtual" ]
},
{
    "type": "Link",
    "src_node": "core2",
    "dst_node": "prod",
    "groups": [ "virtual" ]
},
```



IP networking on topology

```
{
        "type": "CentralIP",
        "ip4_data_prefix": "192.168.19.0/24",
        "ip6_data_prefix": "9001::/16",
        "ip_routing_strategy": "spt",
        "groups": [
                "virtual"
         ]
}
```

IPv6 addresses are attributed by /64

Defines objects on which to act

`CentralIP` is similar to an SDN controller that assigns addresses and sets up the routing in the network:

```
CentralIP = (Ipv4Assignment | Ipv6Assignment) > IPRoutes
```

ICN forwarders

```
{
        "type": "MetisForwarder",
        "cache_size": 0,
        "node": "cons"
},
{
        "type": "MetisForwarder",
        "cache_size": 2000,
        "node": "core1"
},
{
        "type": "MetisForwarder",
        "cache_size": 0,
        "node": "core2"
},
{
        "type": "MetisForwarder",
        "cache_size": 0,
        "node": "prod"
},
```

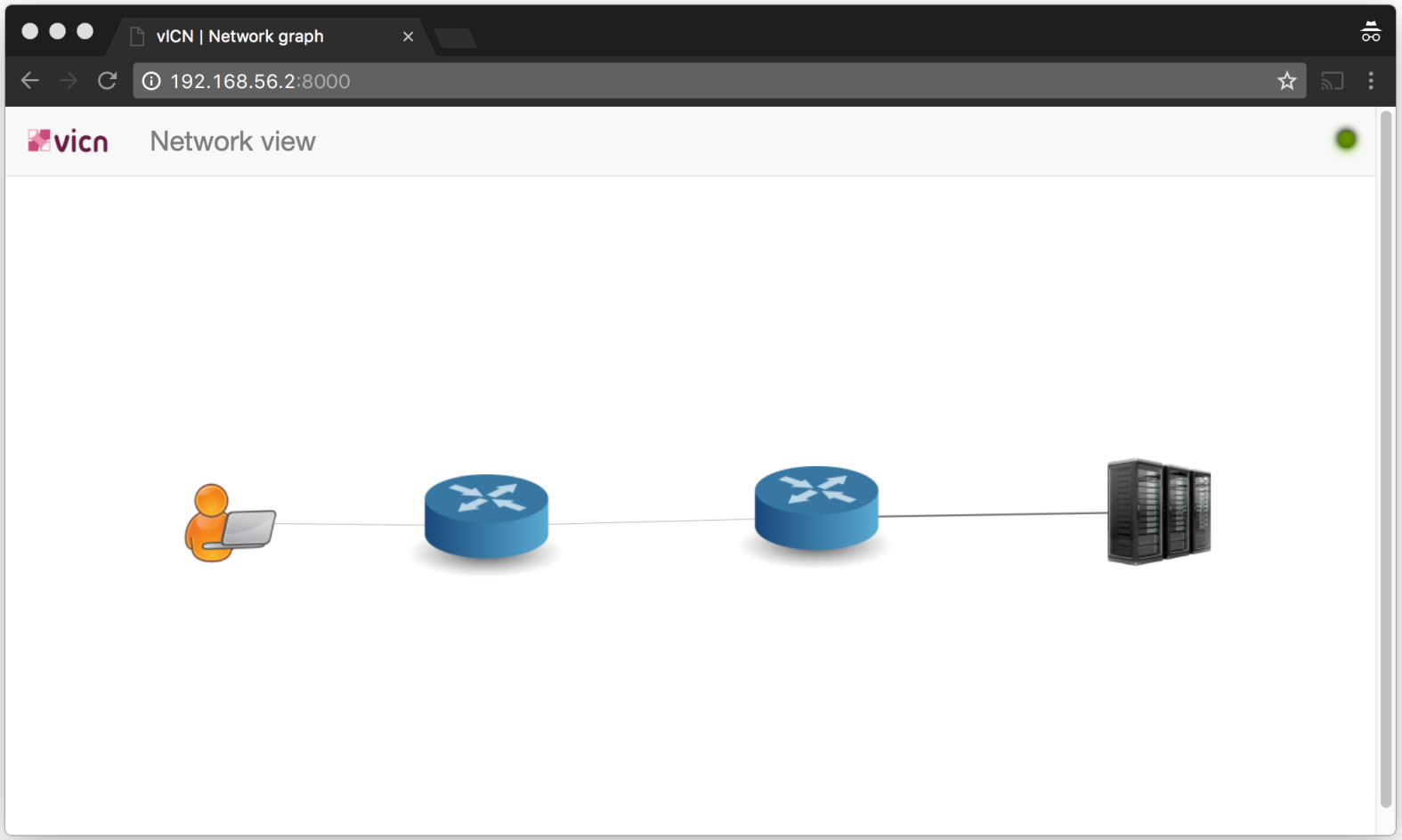
```
{
        "type": "WebServer",
        "prefixes": [
                "/webserver"
        ],
        "node": "prod"
},
{
        "type": "CentralICN",
        "groups": [ "virtual" ],
        "face_protocol": "udp4"
}
```

Like `CentralIP`

ether, udp4, udp6, tcp4, tcp6

GUI

```
{
      "type": "GUI",
      "groups": ["virtual"]
},
```



Launching vicn

```
cicn@cicn-VirtualBox:~/vicn$ sudo vicn/bin/vicn.py -s
examples/tutorial/tutorial06-acm-icn17.json

[…]

2017-09-21 17:48:15,023 - vicn.core.task - INFO - Scheduling task <Task[apy]
partial<_task_resource_update>> for resource <UUID MetisForwarder-MPDRB>
2017-09-21 17:48:15,024 - vicn.core.resource_mgr - INFO - Resource <UUID
MetisForwarder-MPDRB> is marked as CLEAN (99/104)
2017-09-21 17:48:15,146 - vicn.core.task - INFO - Scheduling task <Task[apy]
partial<_task_resource_update>> for resource <UUID MetisForwarder-NC33W>
2017-09-21 17:48:15,148 - vicn.core.resource_mgr - INFO - Resource <UUID
MetisForwarder-NC33W> is marked as CLEAN (100/104)
```

Traffic creation

Producer setup:

- producer-test

```
producer-test -D ccnx:/webserver
```

- Webserver

```
http-server -p $server_folder -l
http://webserver
```

Consumer setup

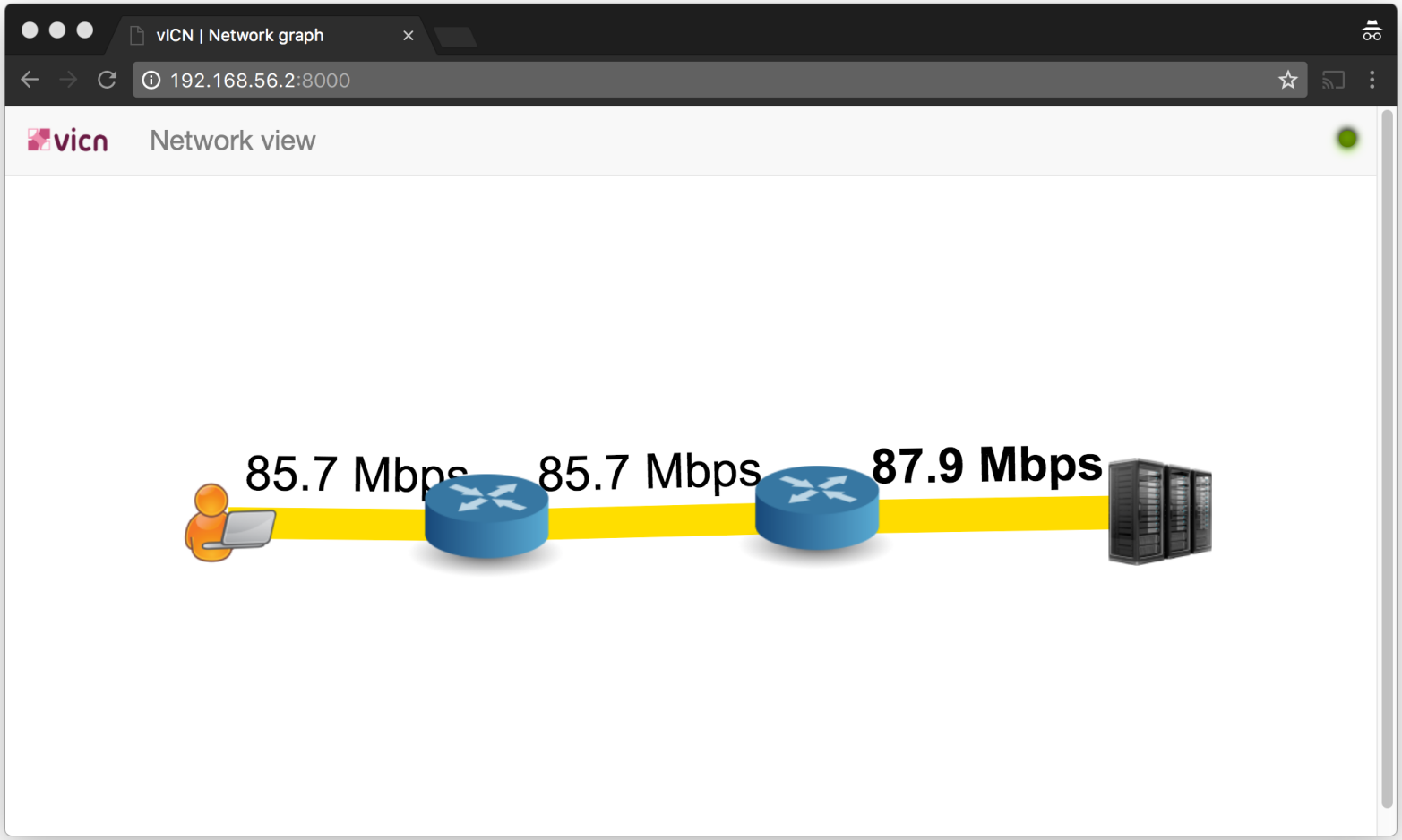
- consumer-test

```
consumer-test -D ccnx:/webserver
```

- iget

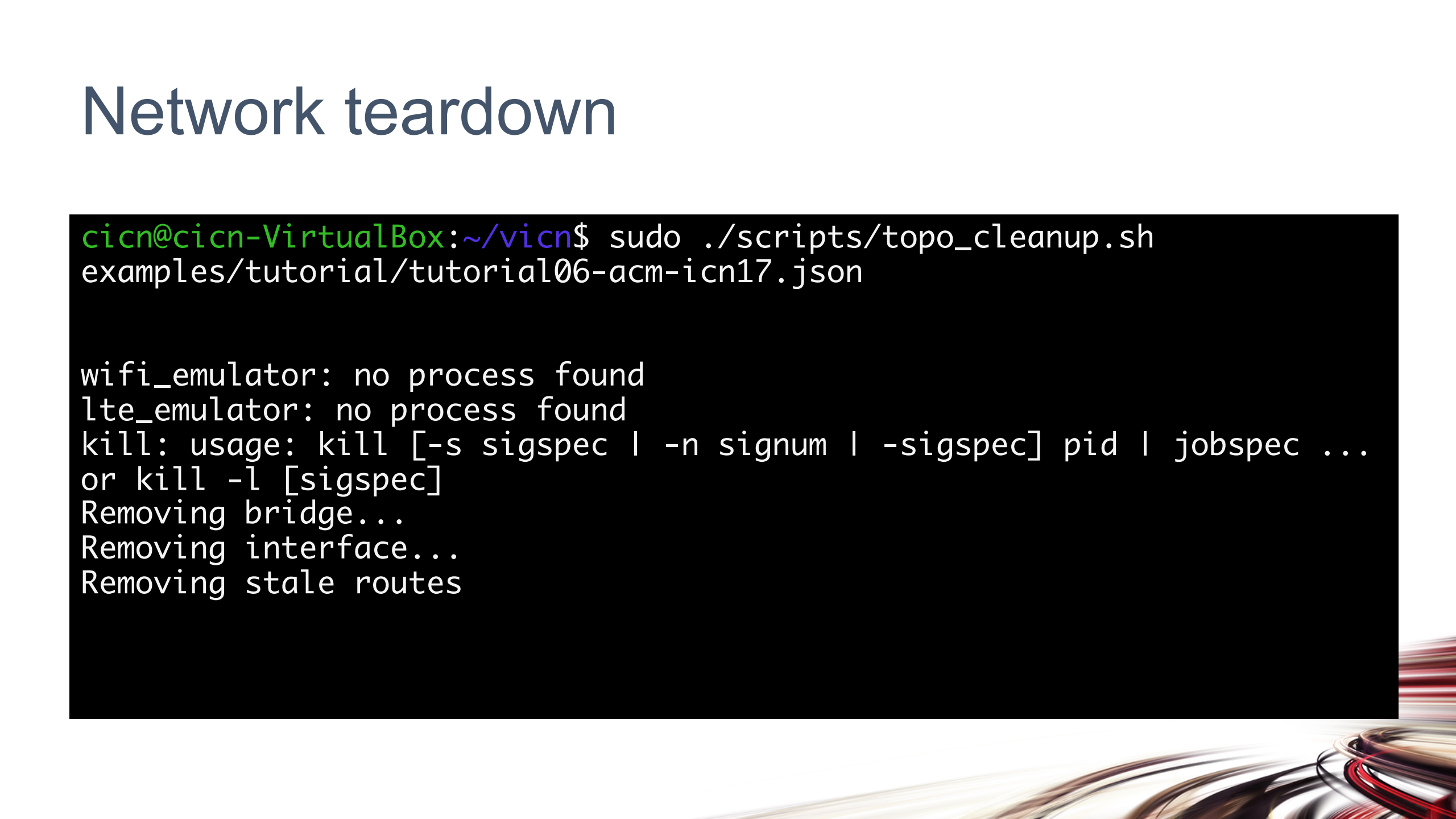
```
iget http://webserver/$filename
```

Traffic visualization on the GUI



Network teardown

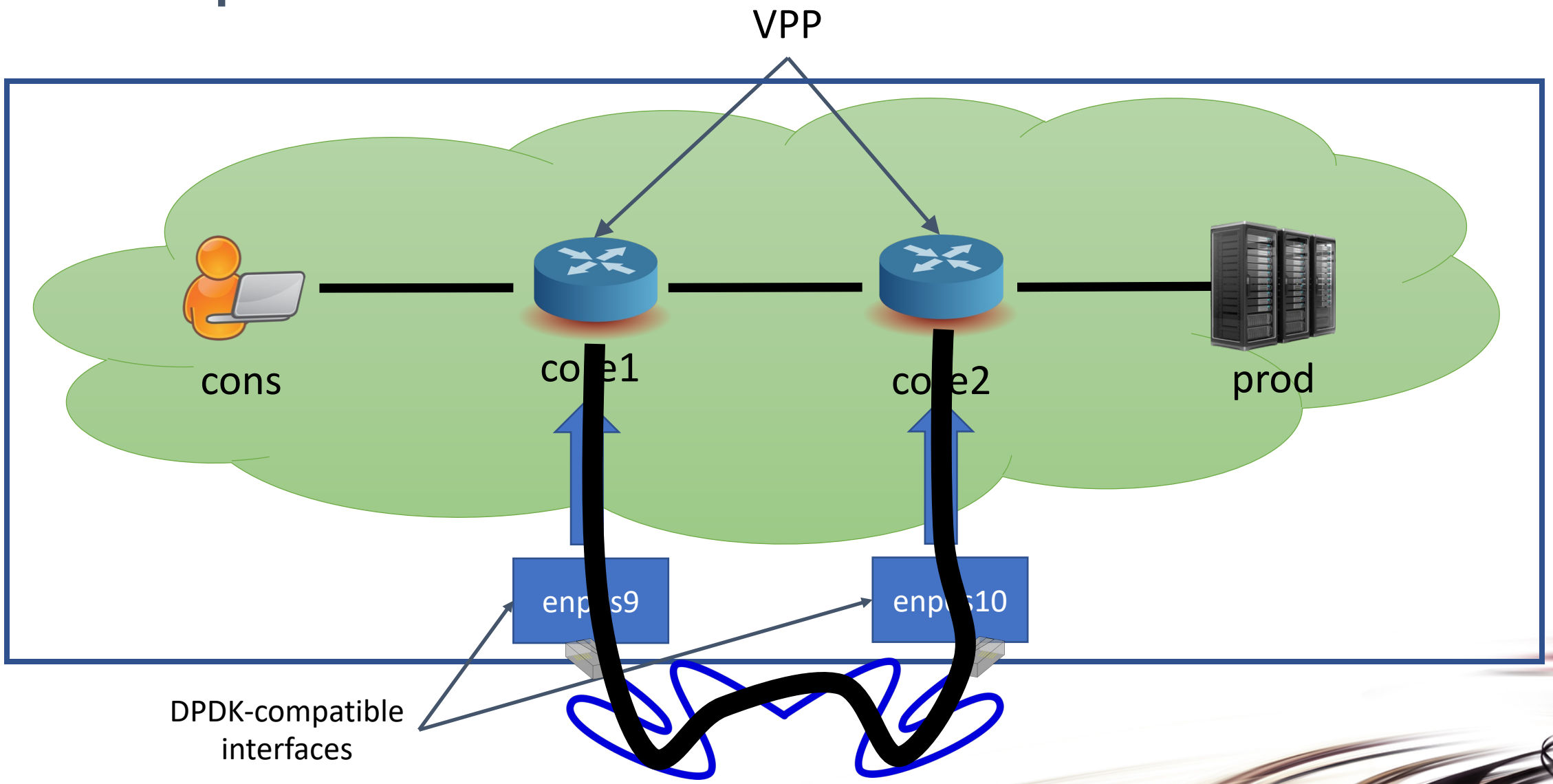
```
cicn@cicn-VirtualBox:~/vicn$ sudo ./scripts/topo_cleanup.sh
examples/tutorial/tutorial06-acm-icn17.json


wifi_emulator: no process found
lte_emulator: no process found
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ...
or kill -l [sigspec]
Removing bridge...
Removing interface...
Removing stale routes
```



VPP in vICN

- Objective: learn to setup vICN to use your Intel interfaces
- VPP running in container
- Uses DPDK and ZC-forwarding

Setup



Identifying the DPDK interfaces

Compare:

```
sudo lshw -c network -businfo
```

with http://dpdk.org/doc/nics

```
cicn@cicn-VirtualBox:~$ sudo lshw -c network -businfo
Bus info          Device      Class       Description
========================================================
pci@0000:00:03.0  enp0s3      network     82540EM Gigabit Ethernet Controller
pci@0000:00:08.0  enp0s8      network     82540EM Gigabit Ethernet Controller
pci@0000:00:09.0  enp0s9      network     82545EM Gigabit Ethernet Controller (Copper)
pci@0000:00:0a.0  enp0s10     network     82545EM Gigabit Ethernet Controller (Copper)
```

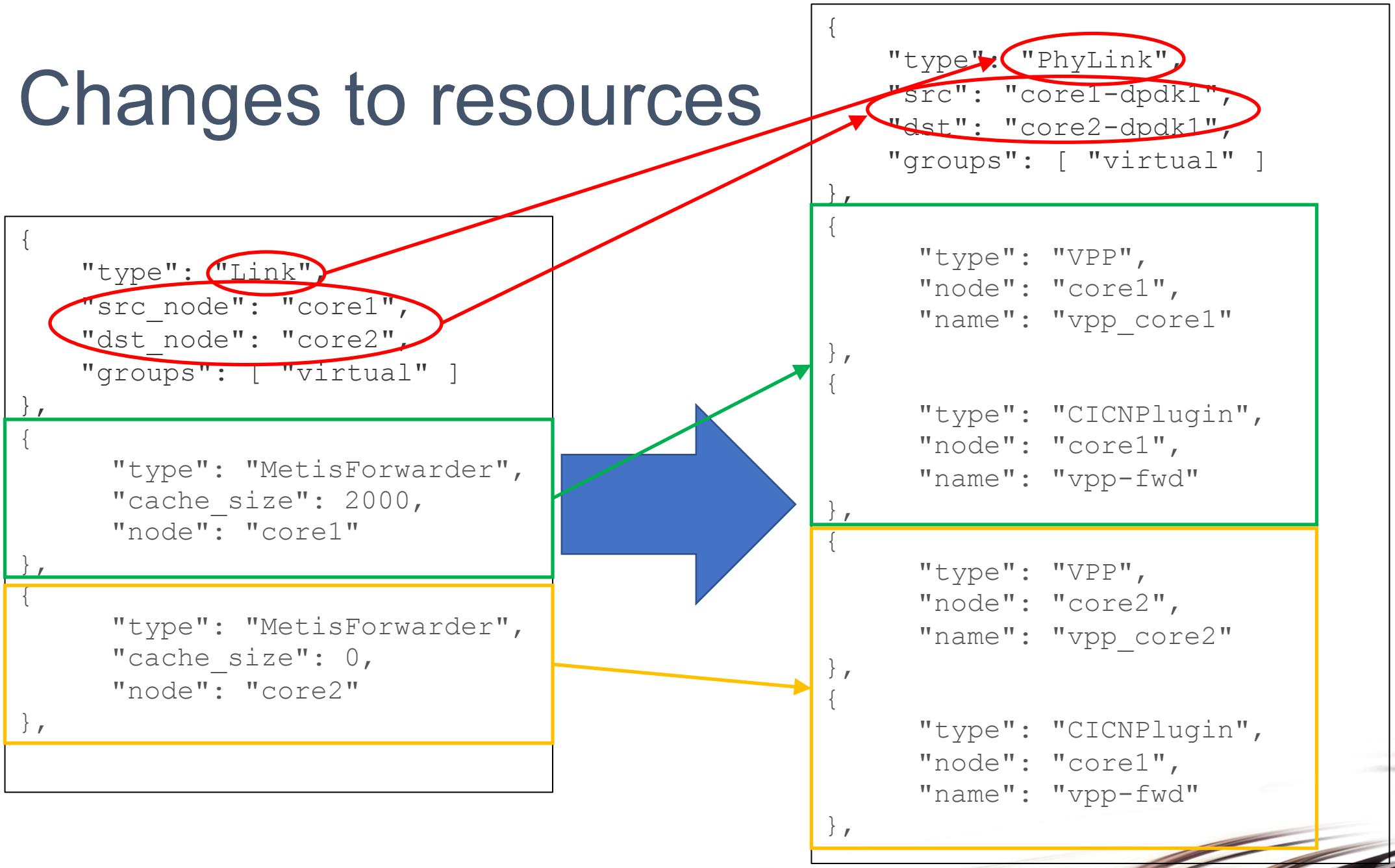
Declaring the DPDK Interfaces

```
{
        "type": "DpdkDevice",
        "name": "core1-dpdk1",
        "mac_address": "08:00:27:44:9a:38",
        "node": "core1",
        "device_name": "enp0s9",
        "pci_address": "0000:00:09.0"
},
```

```
{
        "type": "DpdkDevice",
        "name": "core2-dpdk1",
        "mac_address": "08:00:27:18:42:f2",
        "node": "core2",
        "device_name": "enp0s10",
        "pci_address": "0000:00:0a.0"
},
```

```
cicn@cicn-VirtualBox:~$ sudo lshw -c network -businfo
Bus info          Device      Class       Description
=========================================================
pci@0000:00:03.0  enp0s3      network     82540EM Gigabit Ethernet Controller
pci@0000:00:08.0  enp0s8      network     82540EM Gigabit Ethernet Controller
pci@0000:00:09.0  enp0s9      network     82545EM Gigabit Ethernet Controller (Copper)
pci@0000:00:0a.0  enp0s10     network     82545EM Gigabit Ethernet Controller (Copper)
```

Changes to resources



What is vICN actually doing?

- VPP-ready host
  - Install (if necessary) the DPDK driver and load it in the host kernel
  - Change driver for DPDK-compatible devices
  - Change number of hugepages for VPP
- VPP-ready container
  - Create a privileged container by changing its apparmor profile
  - Add DPDK-enabled interfaces to the container

What is vICN actually doing? (cont'd)

- Start VPP on the container
  - Create configuration file for VPP in the container
  - Start VPP
  - Set up IP forwarding
- Start CICN plugin in VPP
  - Enable CICN plugin
  - Set up ICN faces and routes

Launching vicn

```
cicn@cicn-VirtualBox:~/vicn$ sudo vicn/bin/vicn.py -s
examples/tutorial/tutorial06-acm-icn17-vpp.json

[…]

2017-09-21 17:48:15,023 - vicn.core.task - INFO - Scheduling task <Task[apy]
partial<_task_resource_update>> for resource <UUID MetisForwarder-MPDRB>
2017-09-21 17:48:15,024 - vicn.core.resource_mgr - INFO - Resource <UUID
MetisForwarder-MPDRB> is marked as CLEAN (99/104)
2017-09-21 17:48:15,146 - vicn.core.task - INFO - Scheduling task <Task[apy]
partial<_task_resource_update>> for resource <UUID MetisForwarder-NC33W>
2017-09-21 17:48:15,148 - vicn.core.resource_mgr - INFO - Resource <UUID
MetisForwarder-NC33W> is marked as CLEAN (100/104)
```

Traffic creation

Producer setup:

- producer-test

```
producer-test -D ccnx:/webserver
```

- Webserver

```
http-server -p $server_folder -l
http://webserver
```

Consumer setup

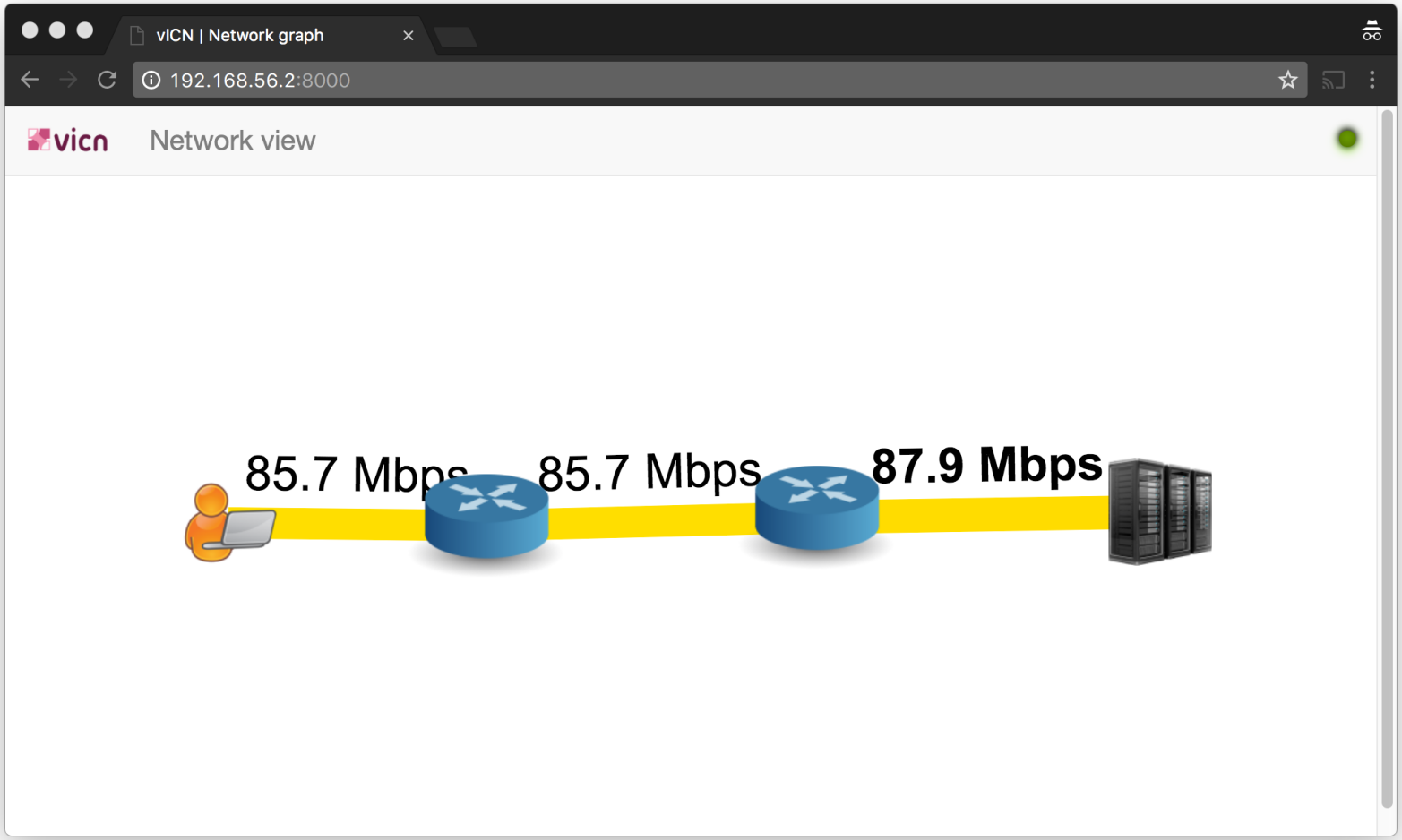
- consumer-test

```
consumer-test -D ccnx:/webserver
```

- iget

```
iget http://webserver/$filename
```

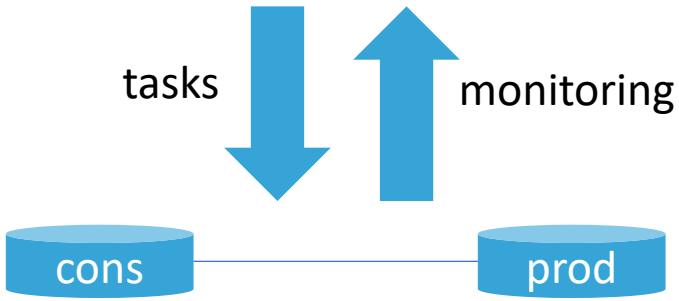
Traffic visualization on the GUI



Toward a new Python API

Use python objects instead of static JSON file

```
cons = LxcContainer()
prod = LxcContainer()
link = Link(src=cons, dst=prod)
```



More on vICN

- Demonstration session: new dynamic python API
- Thursday 10:50am: vICN paper presentation

Available tutorials

In `examples/tutorial/`:

- `tutorial01.json` → Simple topology
- `tutorial02-dumbell` → VPP
- `tutorial03-hetnets.json` → Wireless emulators
- `tutorial06-acm-icn17*` → Today's tutorial (soon)

https://wiki.fd.io/view/Vicn#Tutorials_overview

References

vICN wiki: https://wiki.fd.io/View/Vicn

vICN paper: http://conferences.sigcomm.org/acm-icn/2017/proceedings/icn17-26.pdf

vICN code: git clone -b vicn/master https://gerrit.fd.io/r/cicn vicn

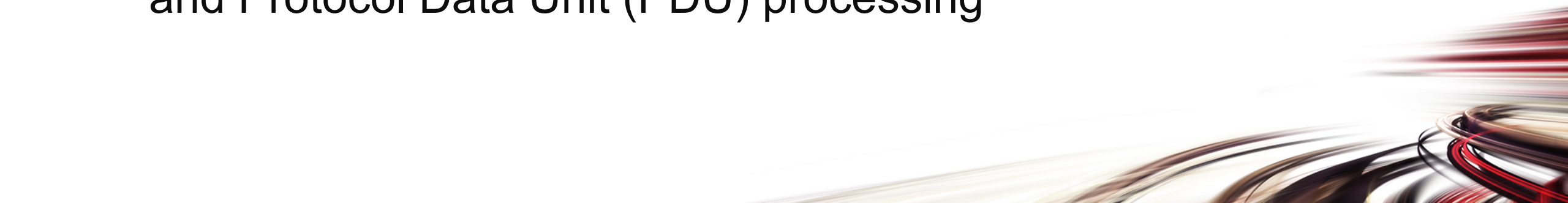
Libicnet: transport layer library for ICN

Mauro Sardara

Tutorial at ACM SIGCOMM ICN, Berlin, Germany
26th of September 2017

What is Libicnet?

- Library implementing a transport layer and exposing **socket API** to applications willing to communicate through an ICN protocol stack
- Relieves applications from the task of managing layer 4 problems, such as **segmentation and congestion control**
- Enhances the **separation** between Application Data Unit (ADU) and Protocol Data Unit (PDU) processing

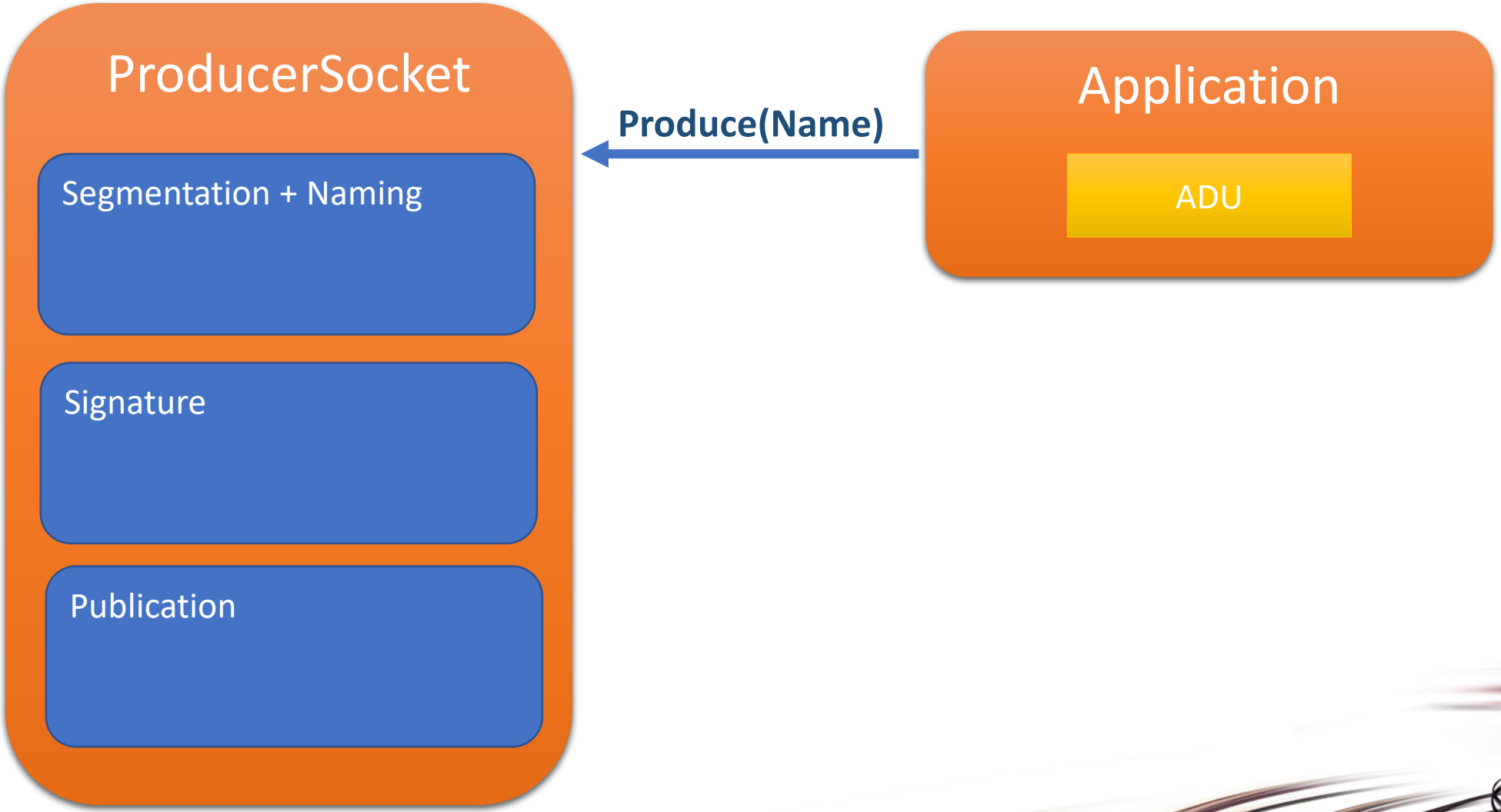


Core Elements

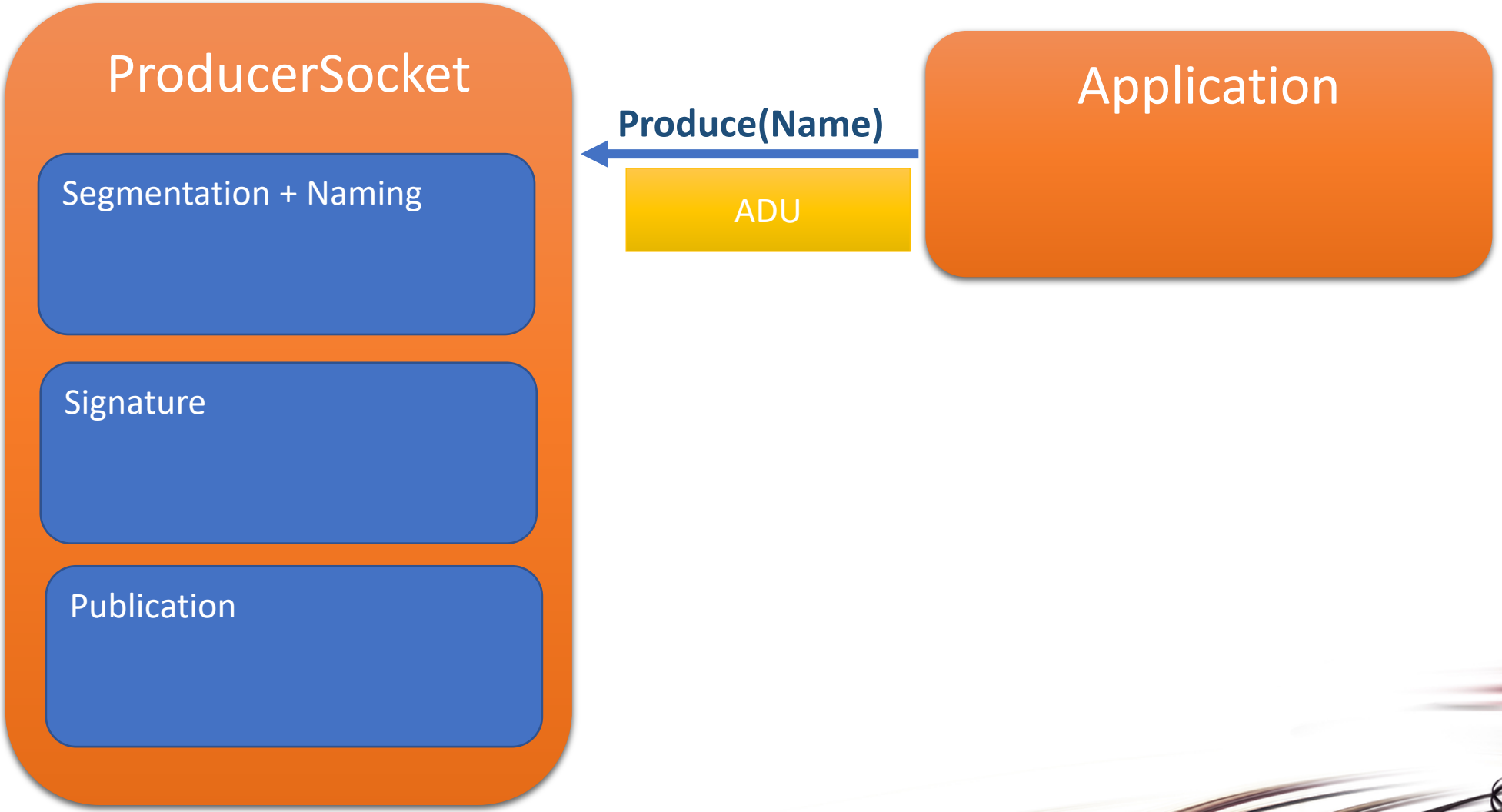
- ProducerSocket
  - ADU Segmentation and Naming → Layer 4 PDU (ICN Content Object)
  - L4 PDU Signature
  - L4 PDU Publication

- ConsumerSocket
  - Congestion control
  - L4 PDU Fetching
  - Signature verification
  - L4 PDU reassembly → ADU

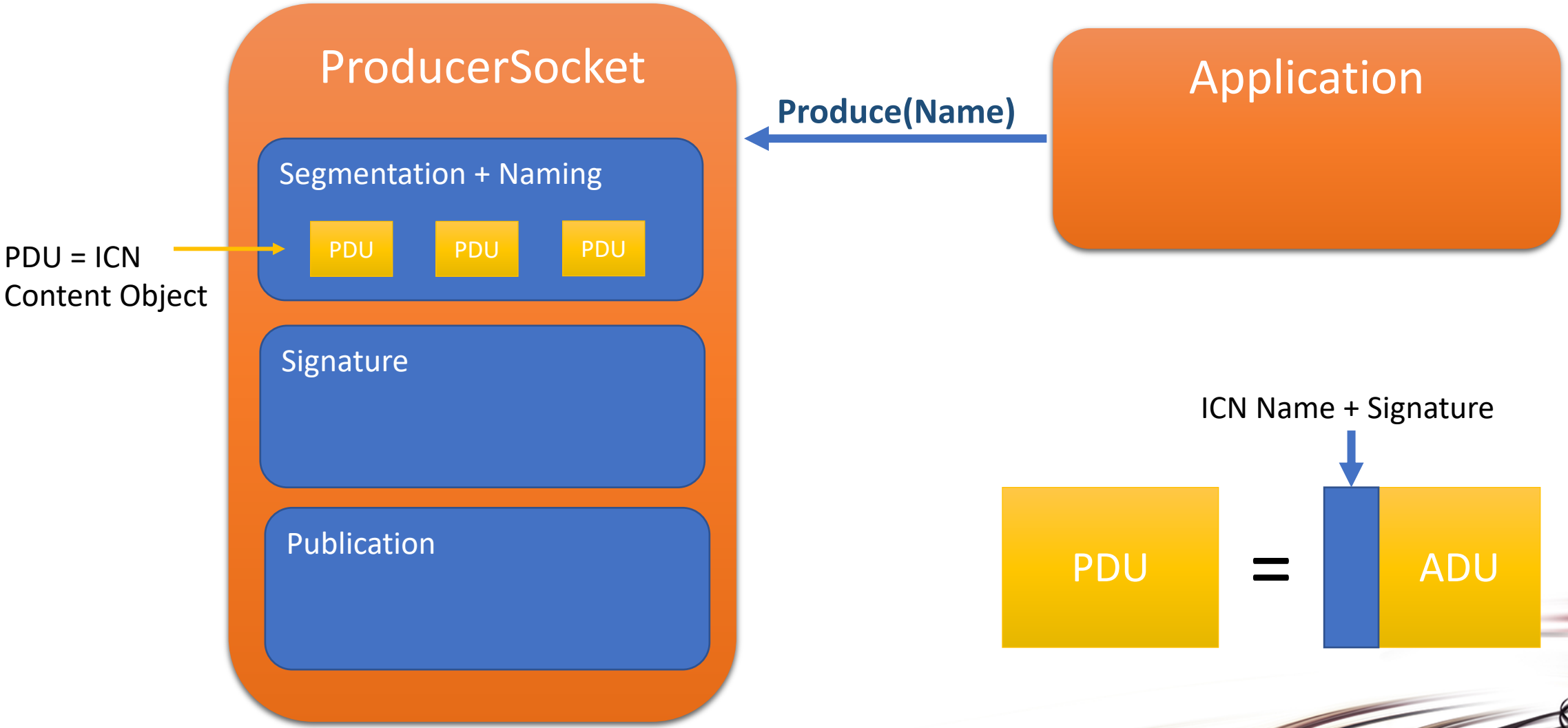
ProducerSocket



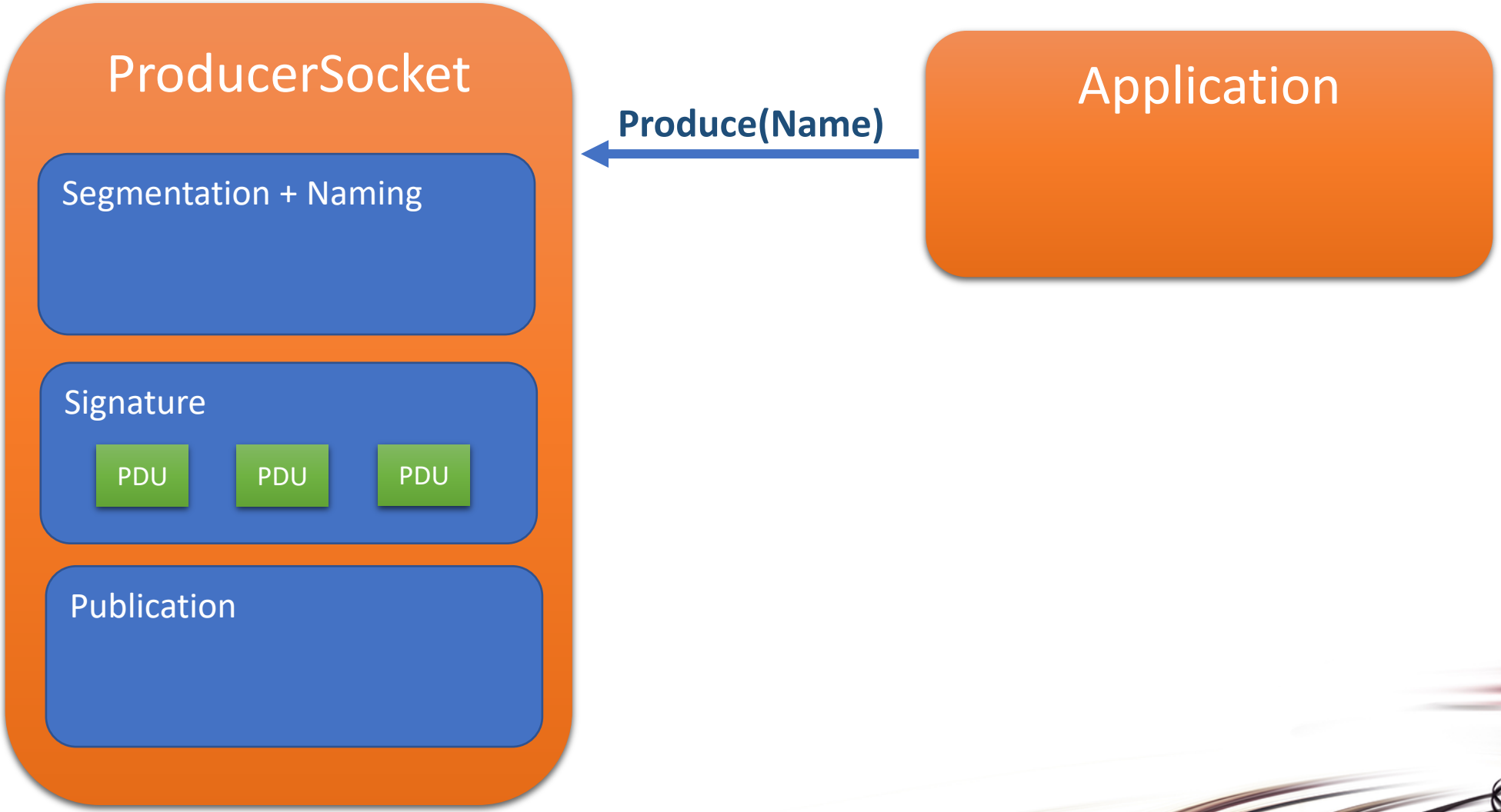
ProducerSocket



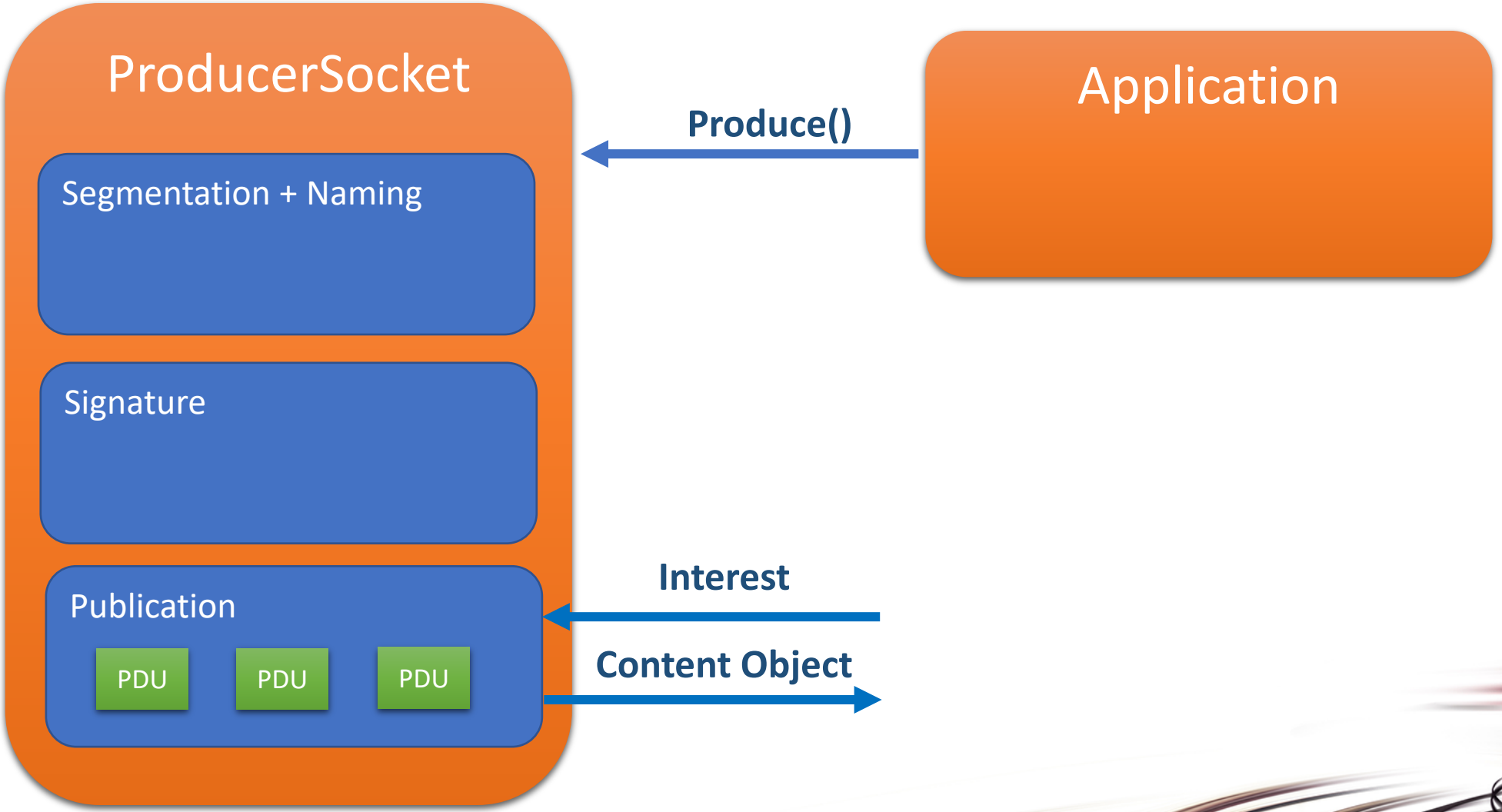
ProducerSocket



ProducerSocket

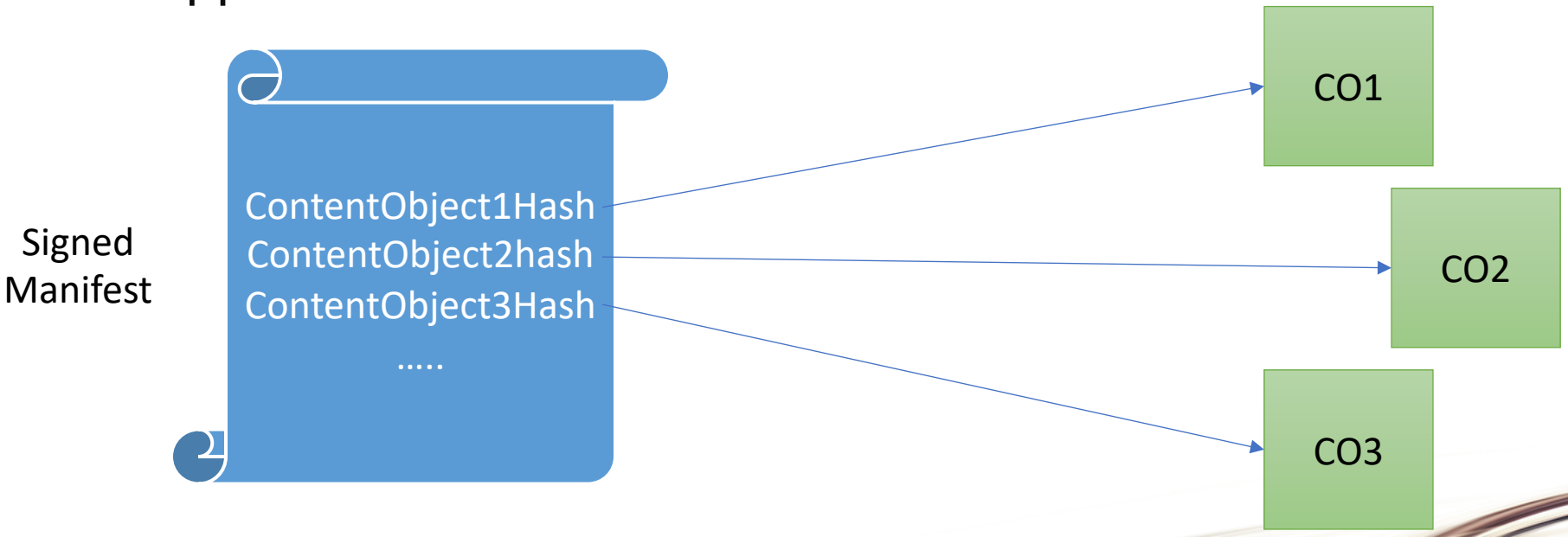


ProducerSocket

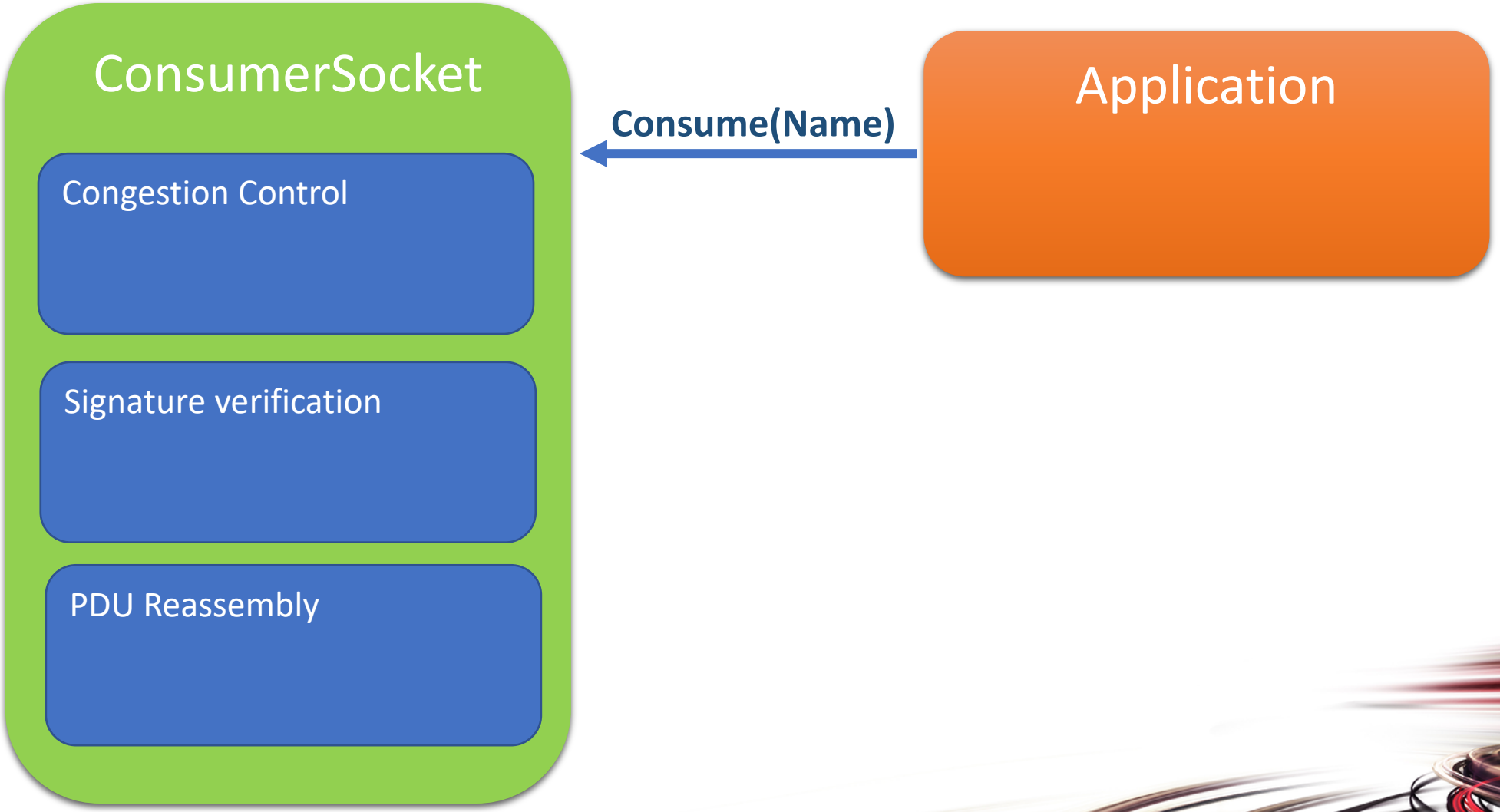


ProducerSocket

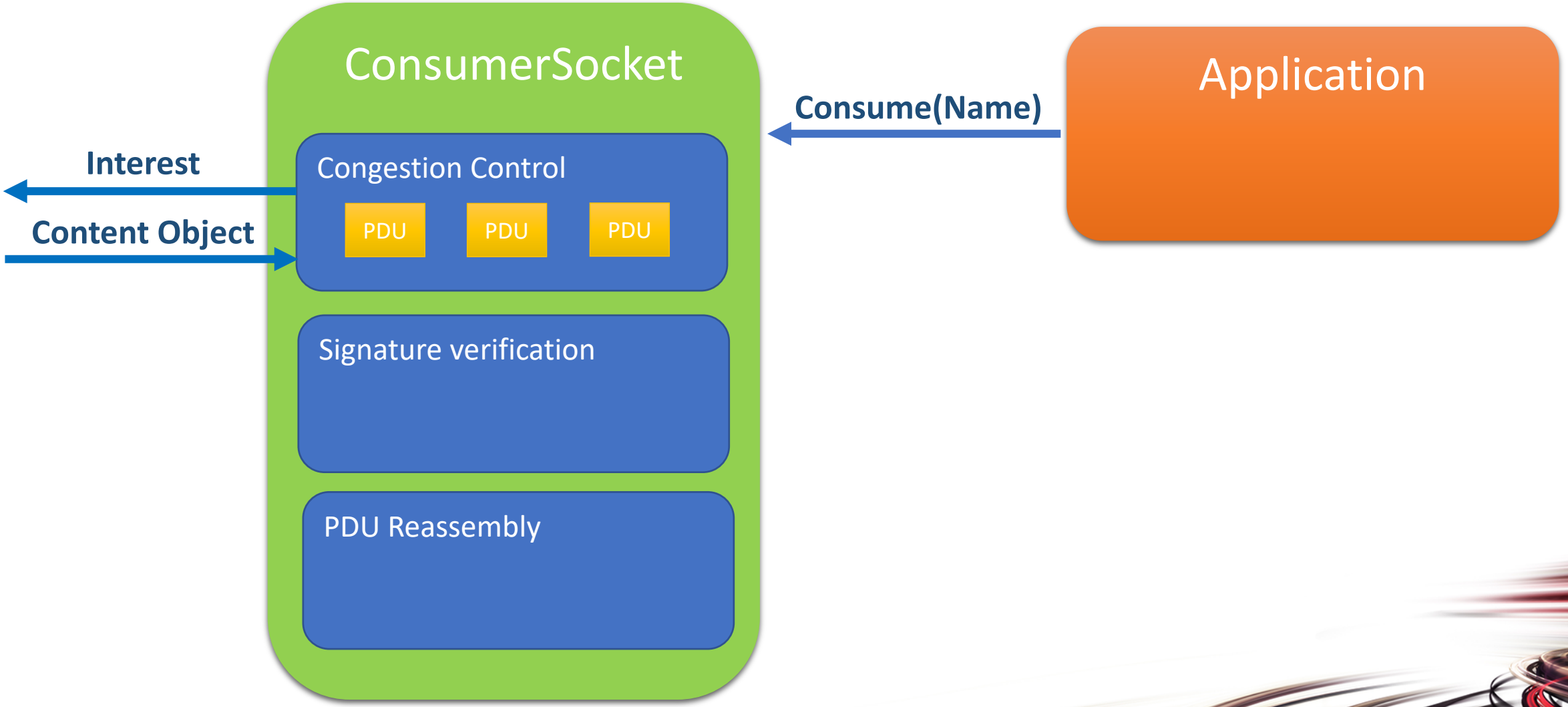
- Signature
  - The application has to provide the library with the information for signing the content objects
  - Signing every content object is computationally expensive: we provide support for manifest



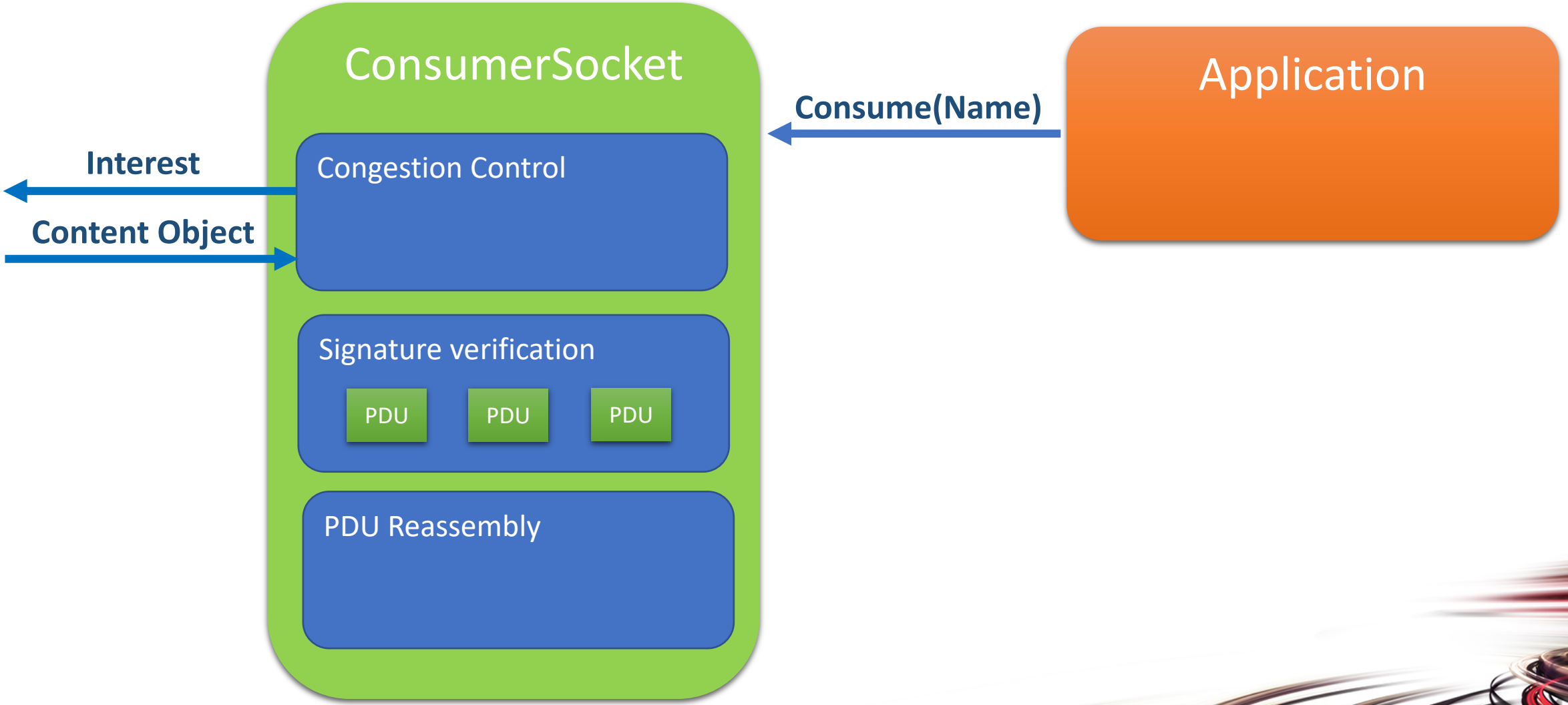
ConsumerSocket



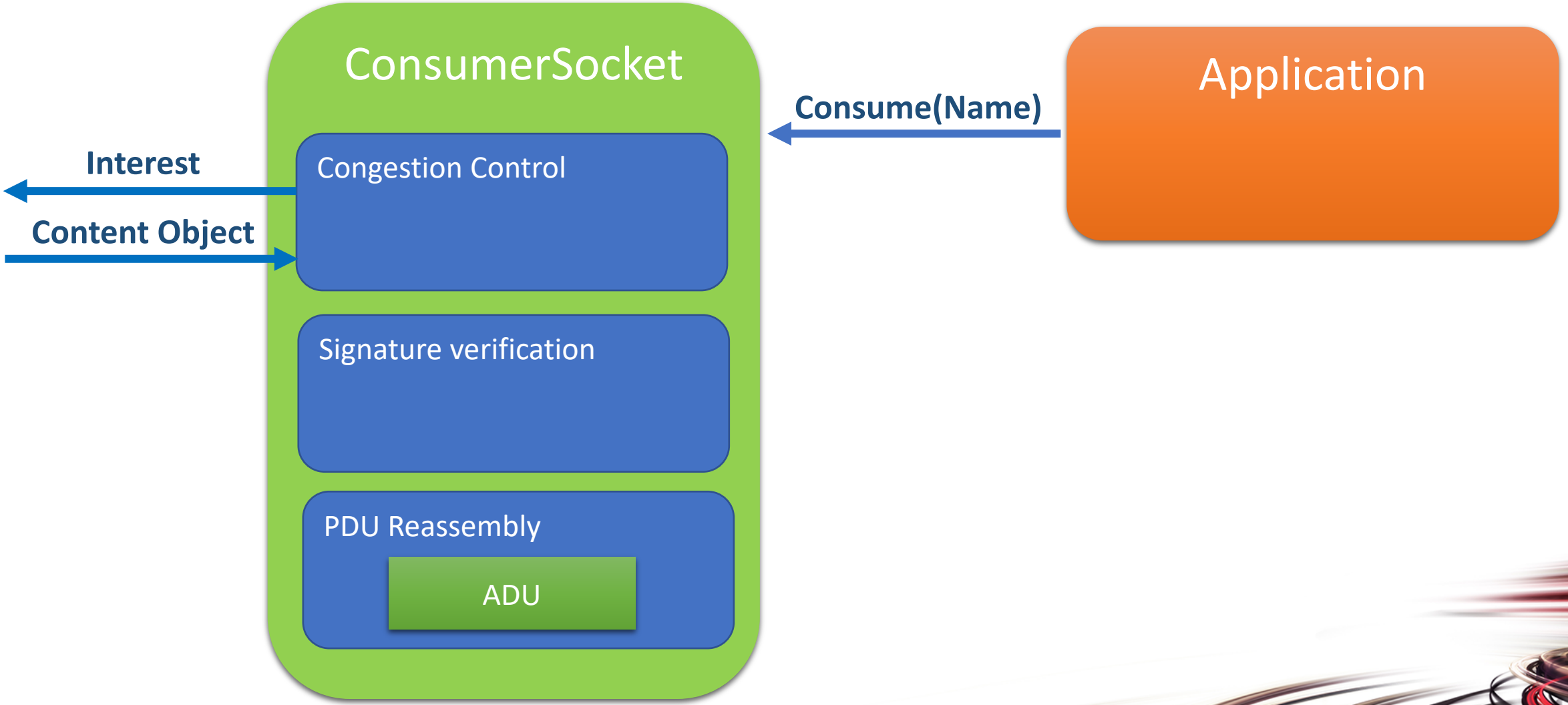
ConsumerSocket



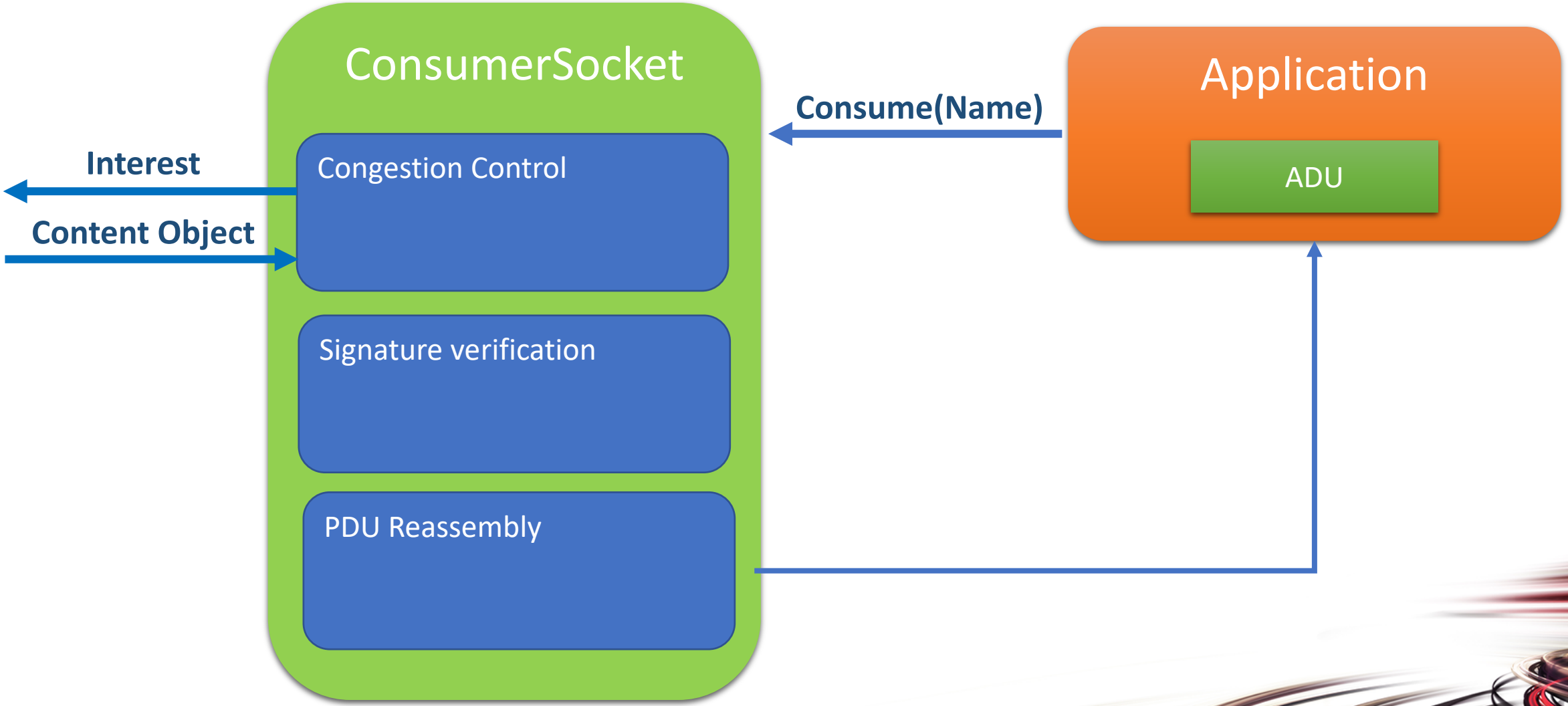
ConsumerSocket



ConsumerSocket



ConsumerSocket



ConsumerSocket

- Congestion Control
  - Application can choose among a set of algorithms: VEGAS, RAAQM[1], FIXED_WINDOW
  - Extension with new algorithms possible
- Signature
  - The application has to provide the library with the information for verifying the signature of the received content objects
  - As the producer case, verifying every content object is expensive: we verify just the manifest signature

[1] G. Carofiglio et al. "Multipath congestion control in content-centric networks,"
2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)

Hands on Libicnet!

Where to find Libicnet?

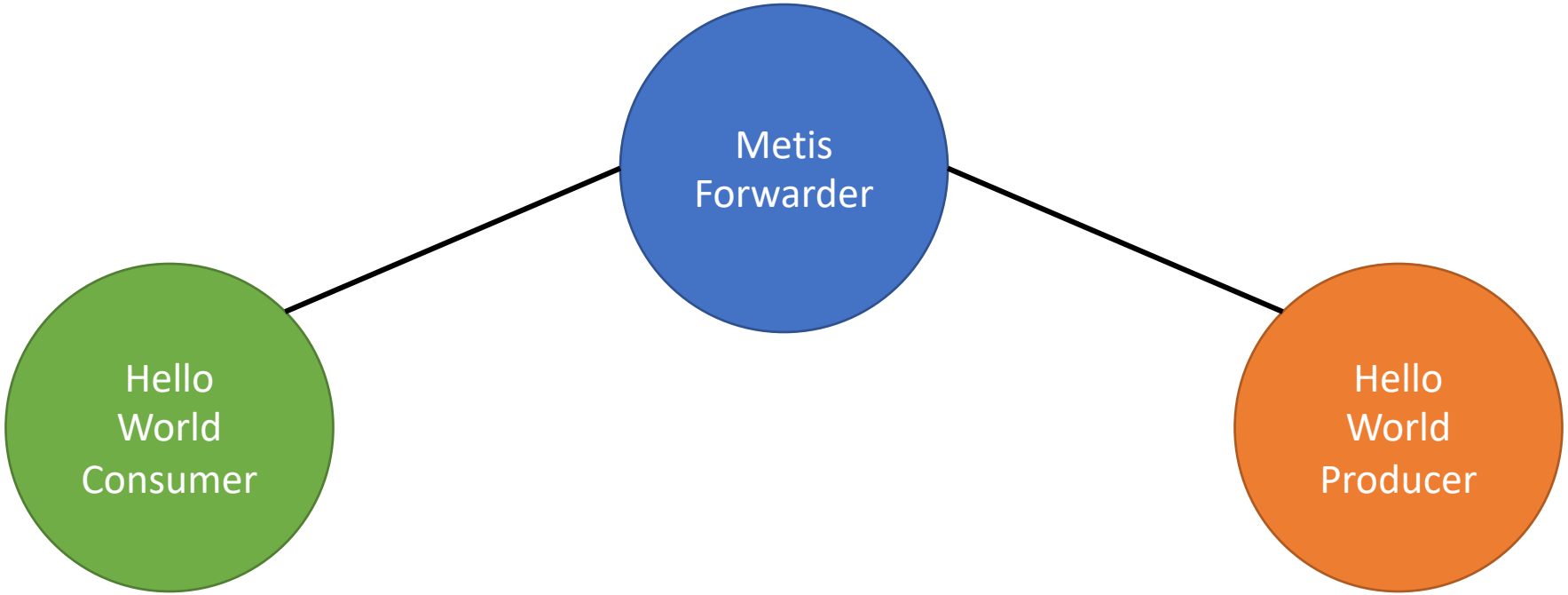
- Wiki page
  - https://wiki.fd.io/view/Libicnet

- Code
  - https://git.fd.io/cicn/log/?h=libicnet/master

Hello World Applications

- **We will see how building two trivial applications against Libicnet:**
  - Hello world Producer
    - It will produce a content of a certain size
  - Hello world Consumer
    - It will pull the content published by the producer

Topology


Metis
Forwarder
Hello
World
Consumer
Hello
World
Producer

Hello World Producer

```
#include <icnet/icnet_transport_socket_producer.h>
…
Name n("ccnx://helloworld");

ProducerSocket p_(n);


std::string content(10000, 'A');


p_.produce(n, (uint8_t *)content.data(), content.size());

p_.attach();

p_.serveForever();
```

Routable prefix

Naming, Segmentation, Signature, Publication

Local face forwarder-producer establishment

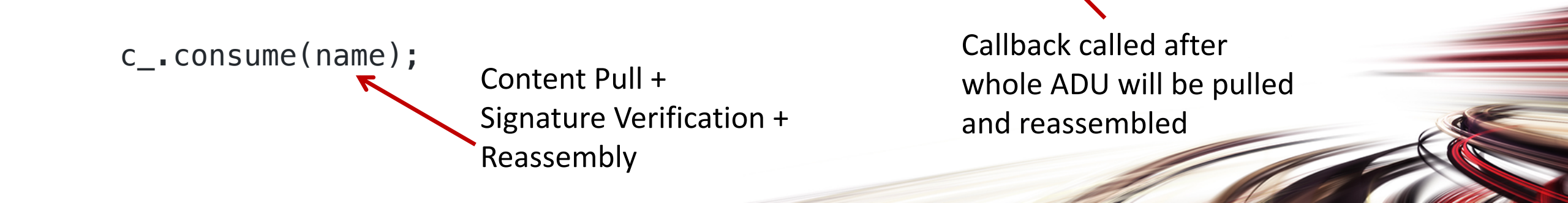
Hello World Consumer

```
#include <icnet/icnet_transport_socket_consumer.h>
…
Consumer c_(Name(), TransportProtocolAlgorithms::RAAQM);
c_.setSocketOption(GeneralTransportOptions::INTEREST_LIFETIME, 1001);
c_.setSocketOption(GeneralTransportOptions::MAX_INTEREST_RETX, 25);


c_.setSocketOption(ConsumerCallbacksOptions::CONTENT_RETRIEVED,
                   (ConsumerContentCallback) std::bind(&processContent,
                                                       std::placeholders::_1,
                                                       std::placeholders::_2));

Name name("ccnx://helloworld");

c_.consume(name);
```

Congestion control algorithm

Callback called after whole ADU will be pulled and reassembled

Content Pull + Signature Verification + Reassembly



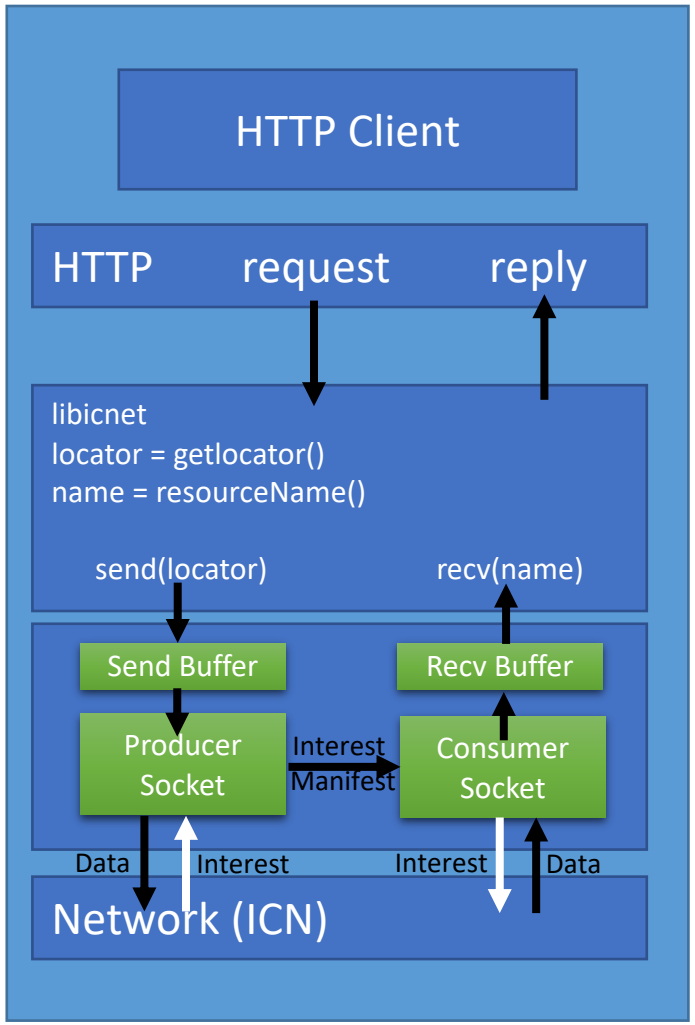
Callbacks

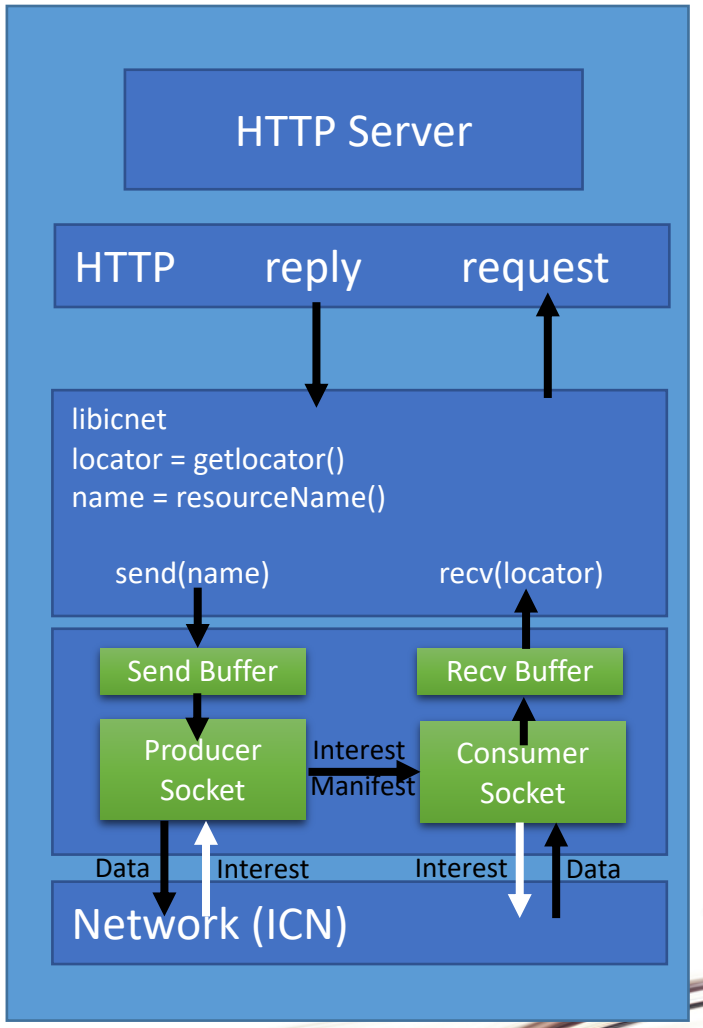
- The application can register into the library a set of callback allowing to directly handle events during the download/publication.

```
typedef enum {
  INTEREST_OUTPUT = 401,
  INTEREST_RETRANSMISSION = 402,
  INTEREST_EXPIRED = 403,
  INTEREST_SATISFIED = 404,
  CONTENT_OBJECT_INPUT = 411,
  MANIFEST_INPUT = 412,
  CONTENT_OBJECT_TO_VERIFY = 413,
  CONTENT_RETRIEVED = 414,
} ConsumerCallbacksOptions;
```

```
typedef enum {
  INTEREST_INPUT = 501,
  INTEREST_DROP = 502,
  CACHE_HIT = 506,
  CACHE_MISS = 508,
  NEW_CONTENT_OBJECT = 509,
  CONTENT_OBJECT_SIGN = 513,
  CONTENT_OBJECT_READY = 510,
  CONTENT_OBJECT_OUTPUT = 511,
} ProducerCallbacksOptions;
```

Advanced Example: HTTP support





Thank You!