INSANE Hands-on

Goal of the hands-on session

The **goal** of this hands-on session is to demonstrate that INSANE makes kernel-bypass acceleration **as simple and portable as** standard networking.

We will demonstrate:

- The ease of porting existing code to INSANE;
- The ease of switching among different kernel-bypass frameworks;
- The performance difference among different kernel-bypass frameworks;

Participants will leave with a working template to build their **own accelerated applications**.

Agenda of the hands-on session

Format:
Guided coding + live testing.

1. **Fundamentals of INSANE**

2. **Example application**

   Application: ping-pong app

   Baseline: The NATS middleware

3. **Environment Setup**

   Connect to remote resources (SSH)

   Explore provided tooling & example project

3. Porting the application to INSANE

Port the ping-pong application from NATS to INSANE

Observe performance with diverse network plugins

4. Wrap-Up & Q&A

INSANE Overview

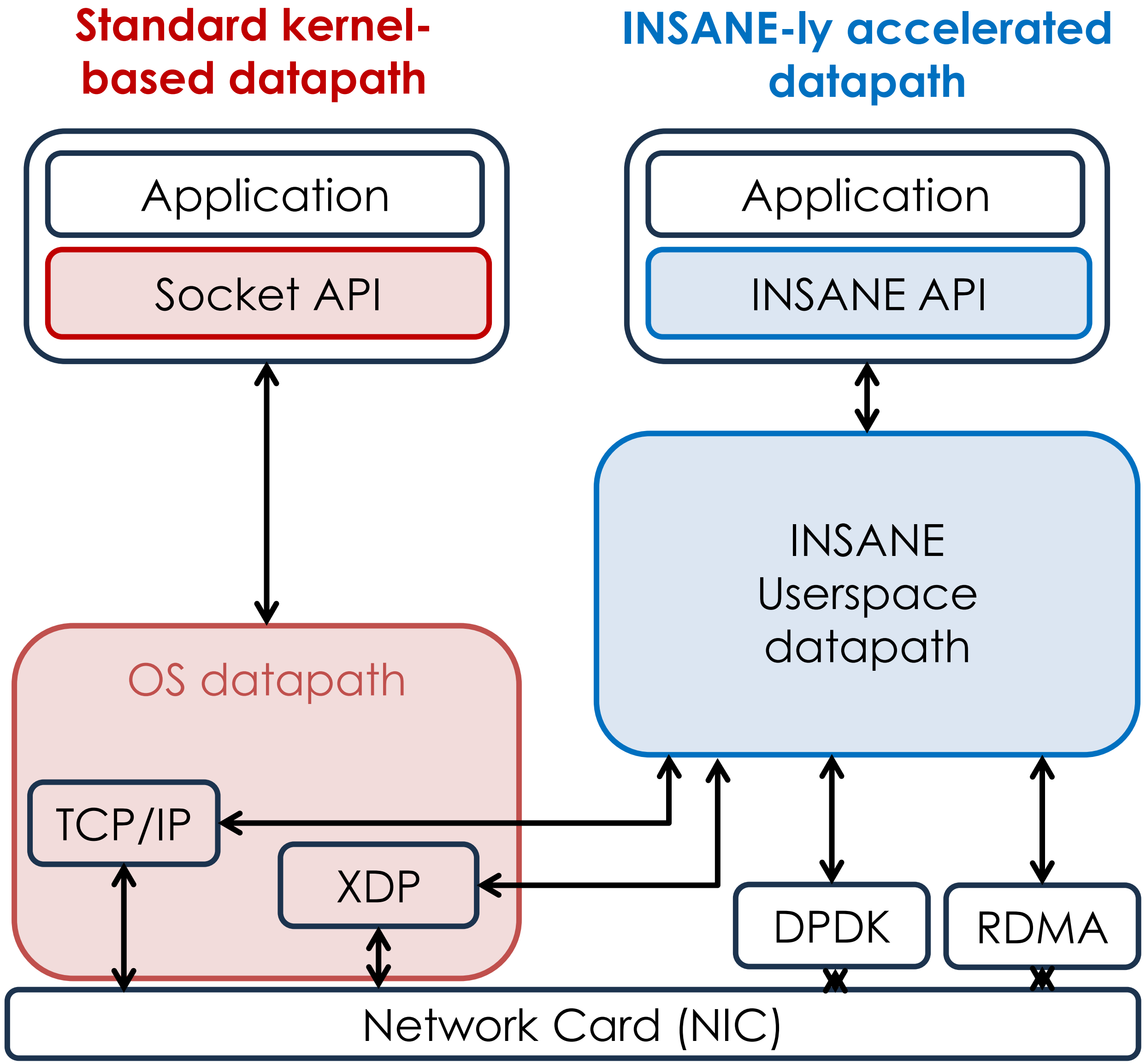
INSANE provides a general-purpose accelerated datapath:

1. Through an agnostic, high-level **API**,

2. Supported by a **userspace datapath**,

with the performance gains of OS bypass and the centralization advantages of a traditional OS kernel

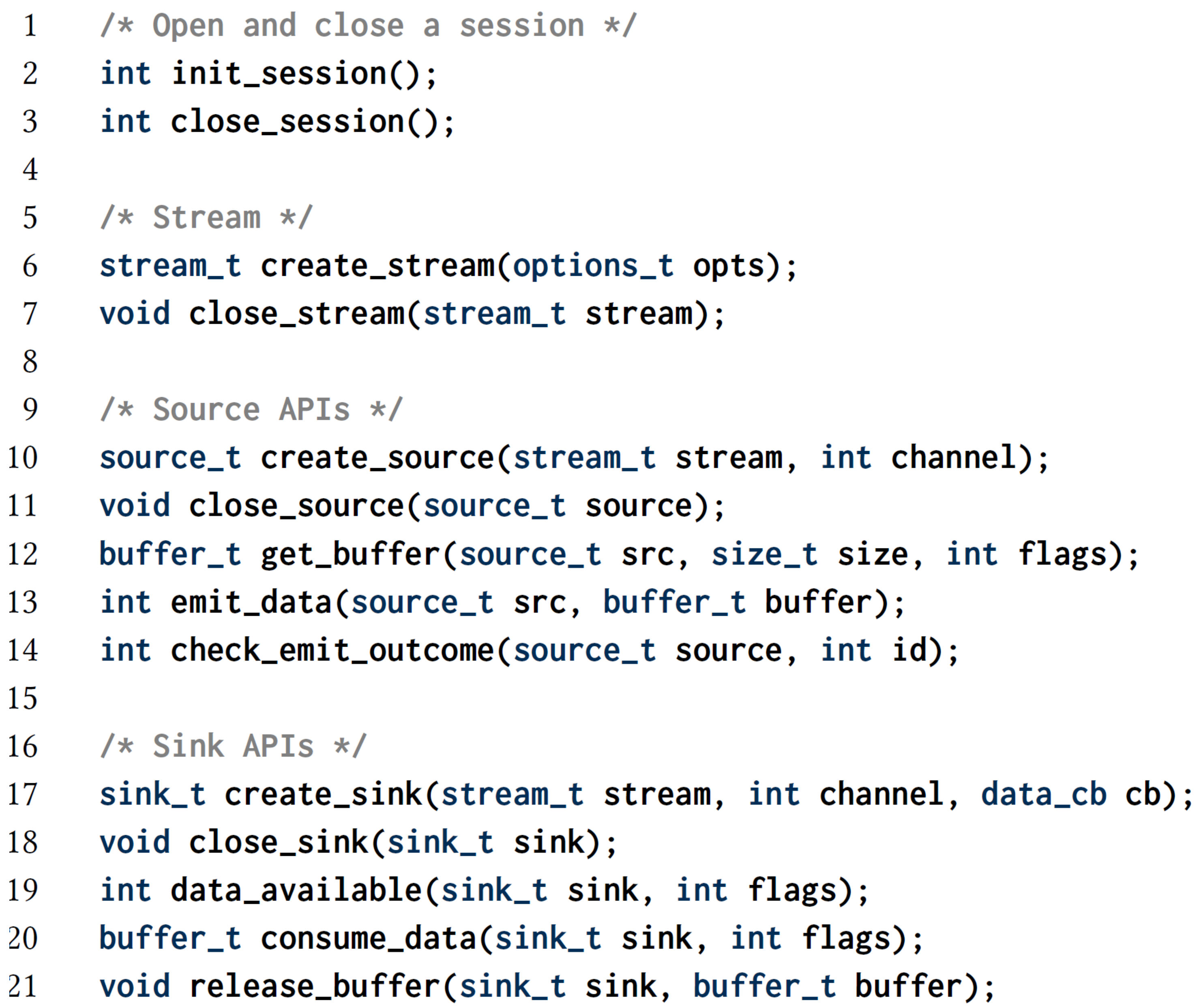
Open Source:
https://github.com/MMw-Unibo/INSANE



INSANE: The API

Currently available in

- C
- Python



```
/* Open and close a session */
int init_session();
int close_session();

/* Stream */
stream_t create_stream(options_t opts);
void close_stream(stream_t stream);

/* Source APIs */
source_t create_source(stream_t stream, int channel);
void close_source(source_t source);
buffer_t get_buffer(source_t src, size_t size, int flags);
int emit_data(source_t src, buffer_t buffer);
int check_emit_outcome(source_t source, int id);

/* Sink APIs */
sink_t create_sink(stream_t stream, int channel, data_cb cb);
void close_sink(sink_t sink);
int data_available(sink_t sink, int flags);
buffer_t consume_data(sink_t sink, int flags);
void release_buffer(sink_t sink, buffer_t buffer);
```

INSANE Quick Start: main concepts (1/3)

INSANE defines the following concepts

1. **Stream**

   A QoS domain. Communication is characterized by the same QoS policies. Will map to a specific network plugin.

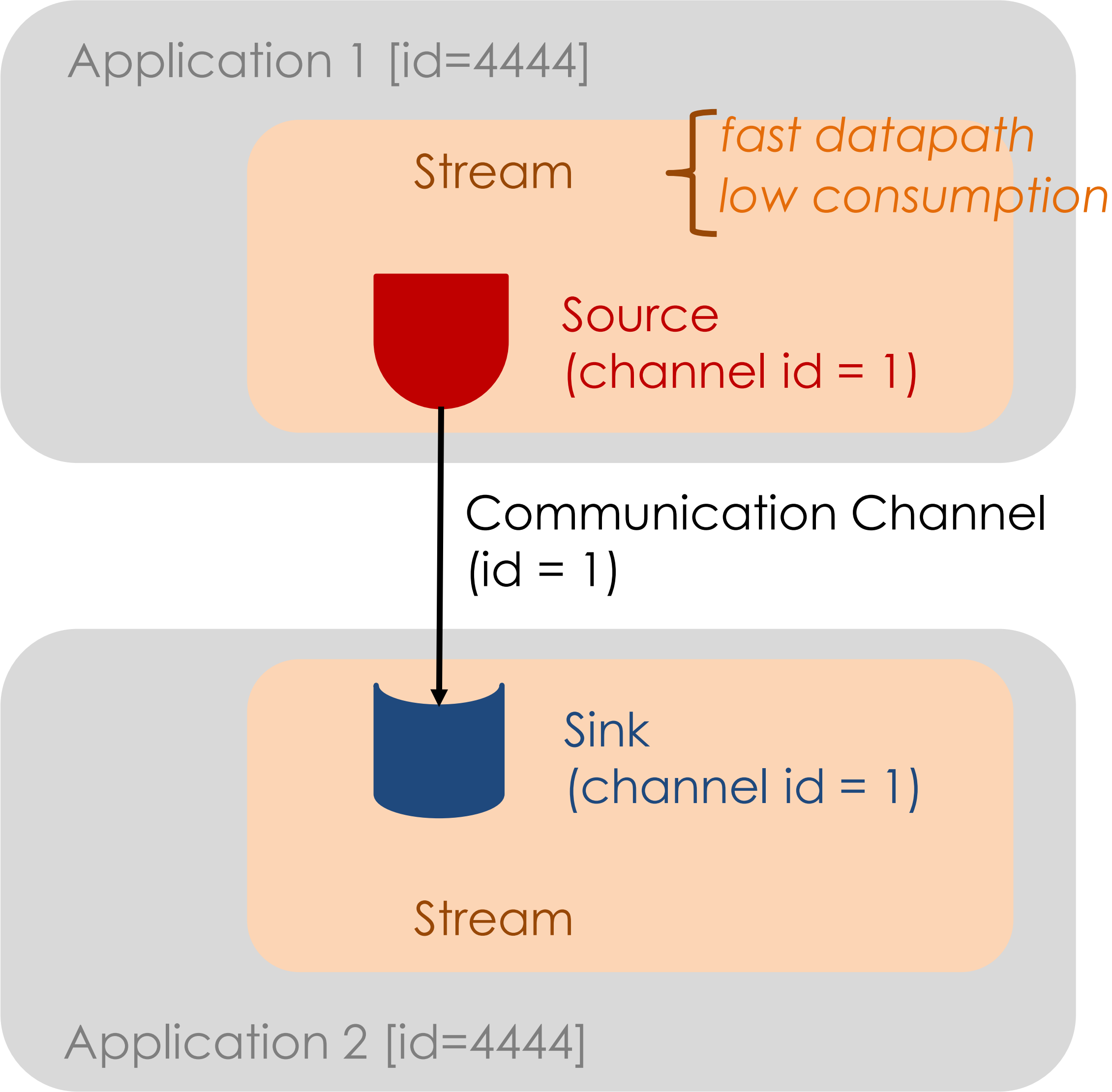
2. **Sink / Source**

   Receiver / Sender of INSANE buffers. Have a user-defined ID. Sink/Source with the same ID form a communication channel.

   Must be associated to a *stream*.

3. **Application ID**

   UDP/TCP port. Only apps with the same ID can communicate.



INSANE Quick Start: main concepts (2/3)

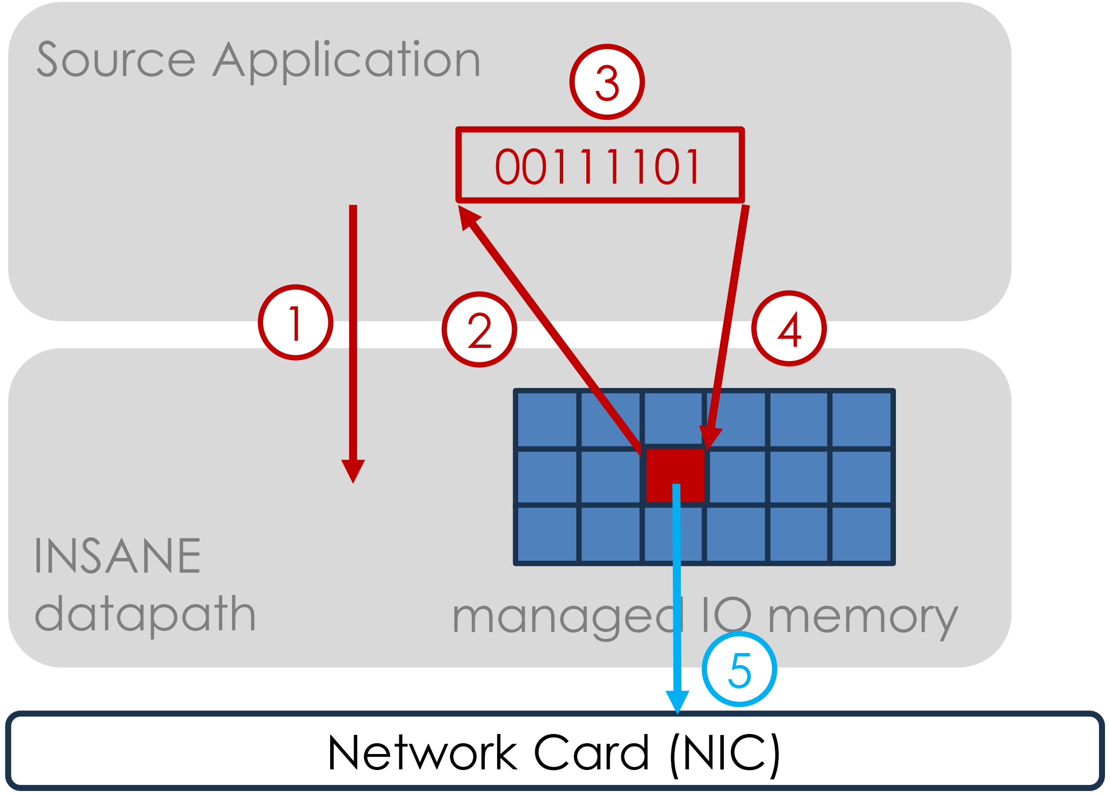
INSANE stream: A QoS domain. Communication is characterized by the same QoS policies. Will map to a specific network plugin.

QoS policies:

1. Reliability:	reliable / unreliable	Selects between TCP and UDP
2. Datapath:	default / fast	Kernel Bypass no / yes
3. Consumption:	polling / default	if datapath fast, software (DPDK, XDP) or hardware (RDMA)
4. Determinism:	default / deterministic	Packet scheduling policy. (Not implemented yet)

INSANE Quick Start: main concepts (3/3)

The INSANE API is **asynchronous** and lets apps access a DMA-capable heap for **zero-copy I/O**.



Clear **memory ownership semantic**: memory is allocated by INSANE, and applications borrow *memory slots* from it.

↓

Memory buffers must be required to send

Memory buffers must be released after receive

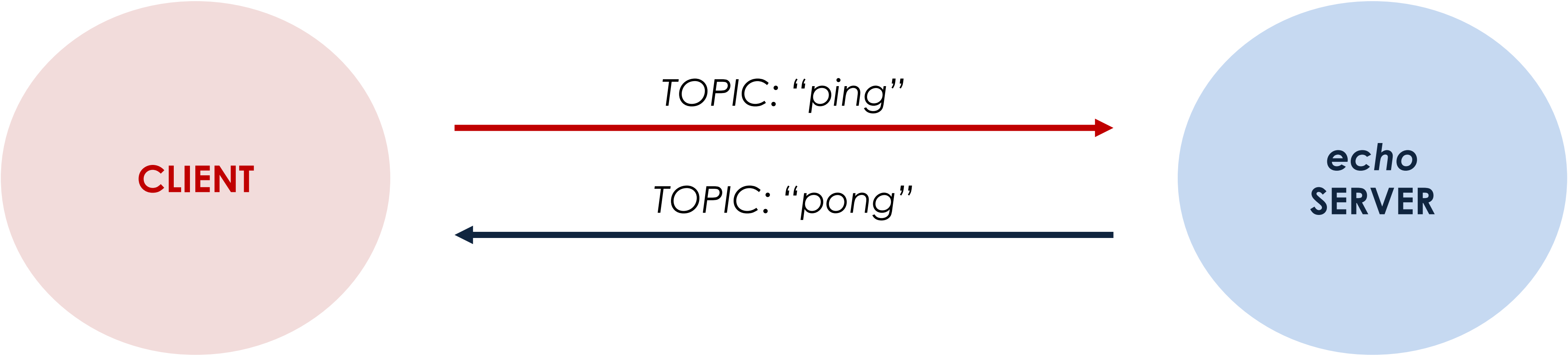
The NATS middleware

https://nats.io/

A lightweight publish/subscribe messaging system designed for **extreme simplicity**, scalability, and low operational overhead .

- Offers a **server-based, pub/sub** communication model.
- Very popular in edge and microservice architectures
- Easy to use and install, but relies on kernel-based networking
- Natively written in Go, but has Python bindings (and many others)

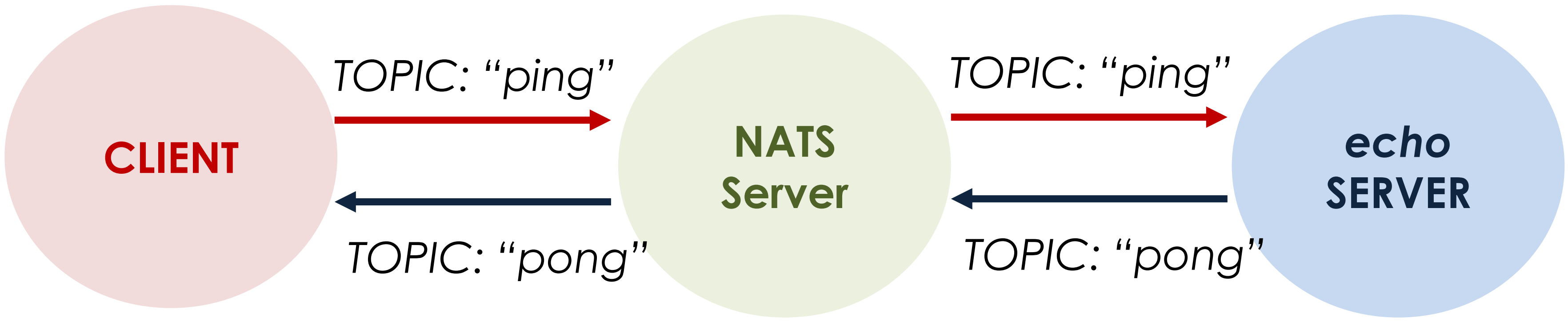
Example Application: a ping-pong test



- The client measures the *round-trip time*.
- The server and the client will be deployed on <u>two remote VMs</u>.
- The application is written in Python, in two versions:
  - using the NATS middleware → provided in the tutorial repository
  - using the INSANE middleware → we will create it step-by-step

Ping-pong application with NATS and with INSANE

NATS
centralized architecture



CLIENT

TOPIC: "ping"

TOPIC: "pong"

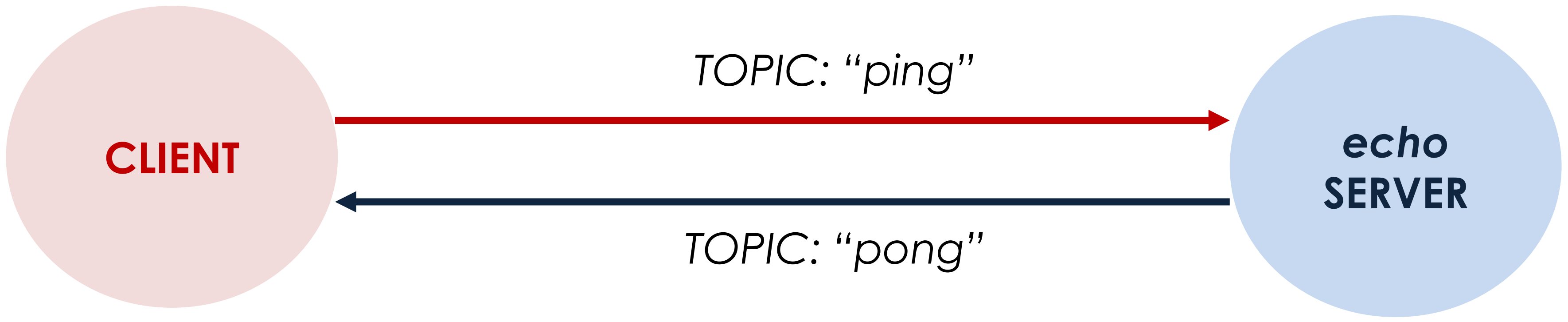
NATS Server

TOPIC: "ping"

TOPIC: "pong"

echo SERVER

INSANE
decentralized architecture



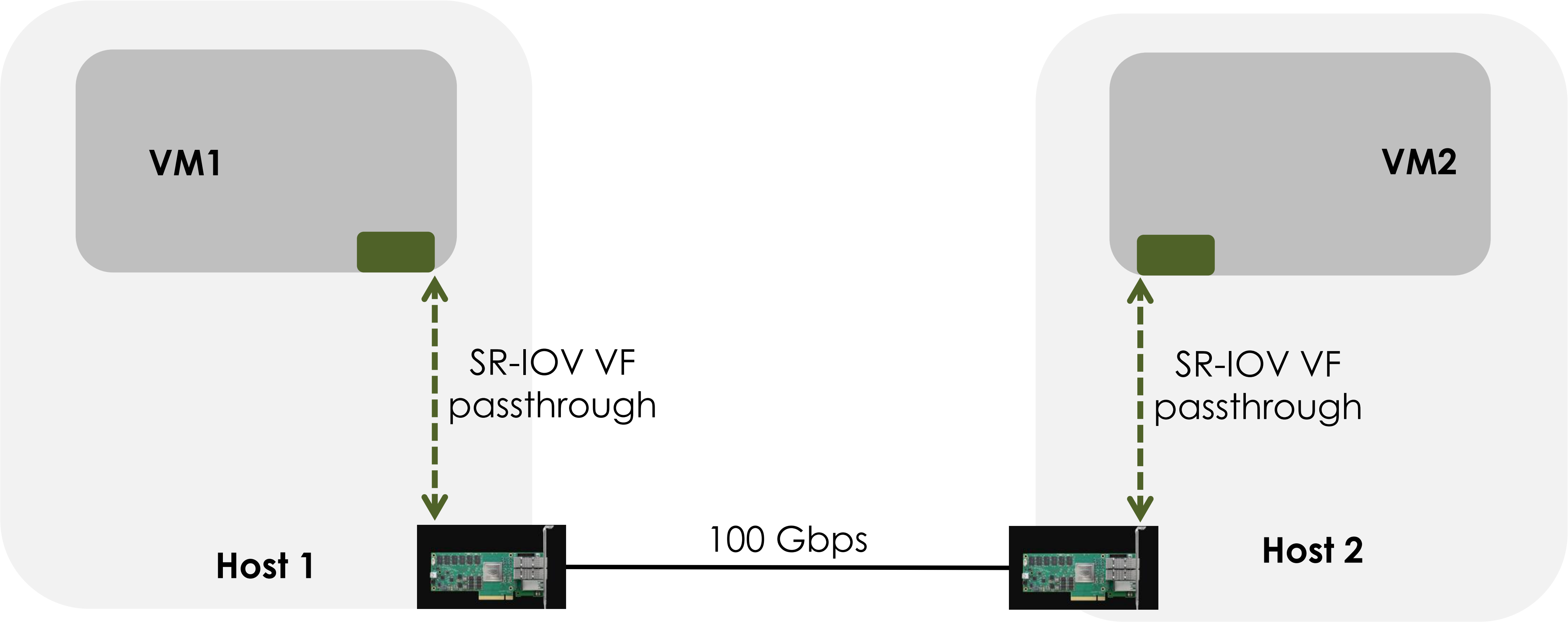
CLIENT

TOPIC: "ping"

TOPIC: "pong"

echo SERVER

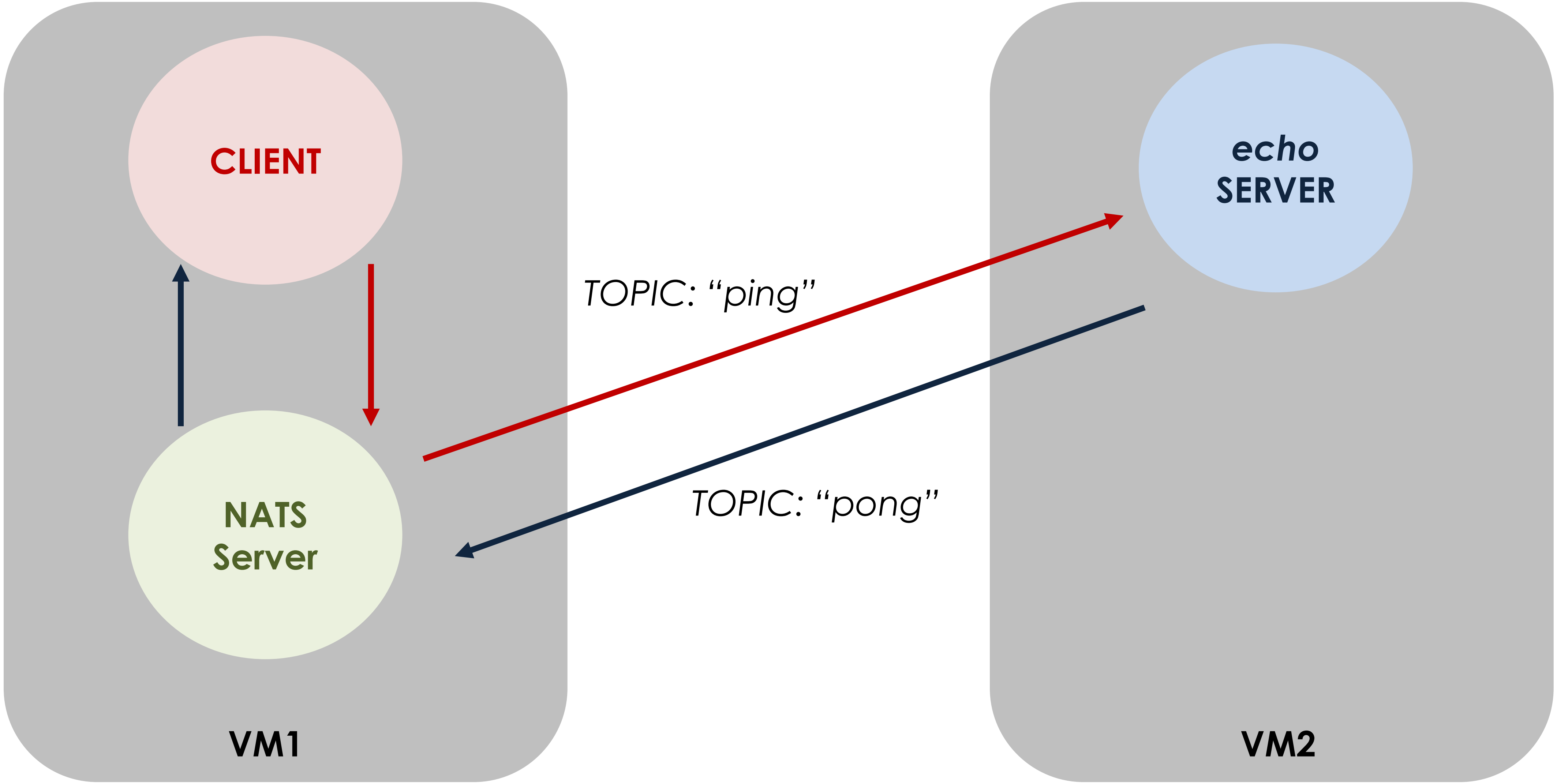
Hands-on testbed



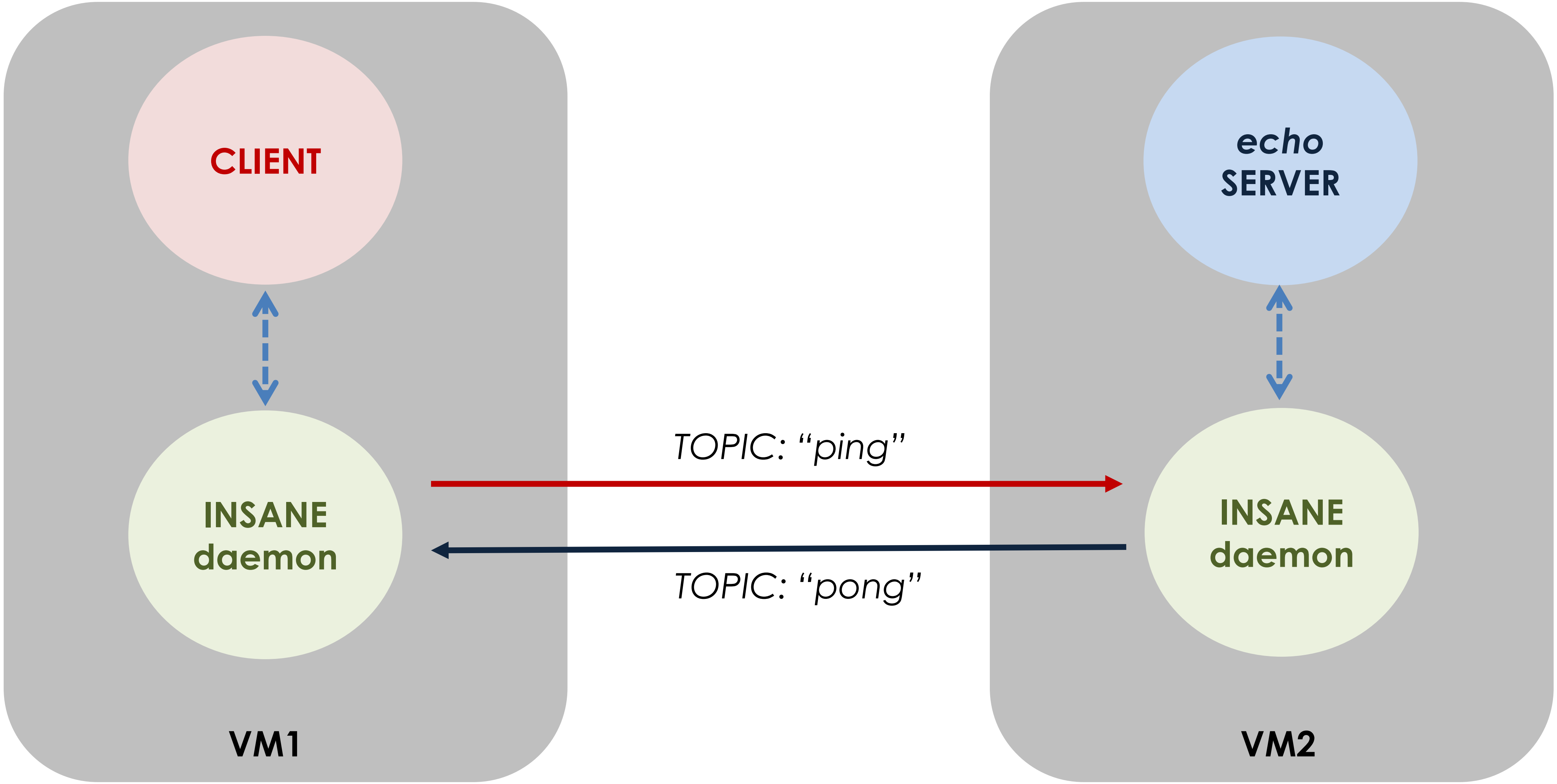
VMs receive a SR-IOV VF in passthrough, i.e., a hardware slice of the NIC.

Performance should match bare-metal settings.

Starting example: NATS-based ping-pong application deployment



Goal: INSANE-based ping-pong application deployment



Accessing the VMs

The instructions **to access the VMs will be provided by the tutorial presenter**

Tutorial repository structure

Available at:
https://github.com/ellerre/insane-tutorial

Once inside the machines, you will find an ***insane-tutorial*** directory, with the starter kit

You can **cd** into the **insane-*tutorial*** repository, you will find:

- a *nats* directory: contains the NATS server binaries and the NATS-based ping-pong application
- an *insane* directory: contains the INSANE binaries, config and lib files needed to create the INSANE-based ping-pong application.

For those not attending the tutorial, you can download the files from:

NATS binaries: https://nats.io/

INSANE binaries: https://github.com/MMw-Unibo/INSANE

Starting the NATS-based application

After taking a look at the Python code, you can activate the virtual env:

```
source venv/bin/activate
```

Then, execute the NATS-based example:

1. On **VM1**, start the NATS server:
   ```
   ./nats-server --addr=<local_ip> --port=4222
   ```
2. On **VM2**, start the *echo server*:
   ```
   python3 pong.py --server-ip <ip> --port 4222
   ```
3. On **VM1**, start the *client*:
   ```
   python3 ping.py --server-ip <ip> --port 4222 --size 64
   ```

The client will start printing RTT values (in microseconds)

Porting the application to INSANE

The tutorial will guide the participants to modify the NATS-based ping-pong application, thus creating an INSANE-based ping-pong application.

People not attending the tutorial will find the solution (`ping.py` / `pong.py`) in the online repository of the tutorial, under the *insane* directory.

Starting the INSANE-based application

You can execute the INSANE-based example:

1. On **VM2**, start the INSANE daemon:

   ```
   sudo ./nsnd
   ```

2. On **VM2**, start the *echo server*:

   ```
   sudo python3 pong.py --qos [fast|default]
   ```

3. On **VM1**, start the INSANE daemon:

   ```
   sudo ./nsnd
   ```

4. On **VM1**, start the *client*:

   ```
   sudo python3 ping.py --qos [fast|default] --size <size>
   ```

The config files of the daemon (nsnd.cfg) and app (nsn-app.cfg) must be in the same directory of the binaries. The client will start printing RTT values (in microseconds)

INSANE-based application: easily switch to kernel-bypass!

You can execute the INSANE-based example in two ways:

1. In *compatibility* mode: **`--qos default`**

   This will map the communication to the kernel-based TCP stack

2. In *accelerated* mode: **`--qos fast`**

   This will map the communication to the DPDK-based TCP stack

The **performance difference** between the two modes is significant.

You can check the CPU to assess the **resource consumption** wrt NATS.

Note: even with more concurrent applications, INSANE will use the same number of polling cores, whose cost can thus be shared.

INSANE-based application: considerations (1/2)

- **QoS affects performance**: the *fast* mode delivers clearly lower RTT than the default one.
- **Acceleration without expertise**: INSANE transparently selects high-performance backends (e.g., DPDK, RDMA). No low-level tuning required.
- **Portability preserved**: existing applications can be migrated with <u>*minimal code changes*</u>
- **Ease of programming**: the ping-pong example showed how INSANE boosts performance while keeping the development workflow simple.

Key Takeaway:

High-performance kernel-bypass networking is attainable for everyone: *INSANE makes acceleration (finally!) easy*.

INSANE-based application: considerations (2/2)

To appreciate the ease of programming, consider:

ping.py + pong.py **84** lines of high-level Python code

That would be:

10x more lines, requiring expertise

dpdk-pingpong.c **819** lines of low-level C code

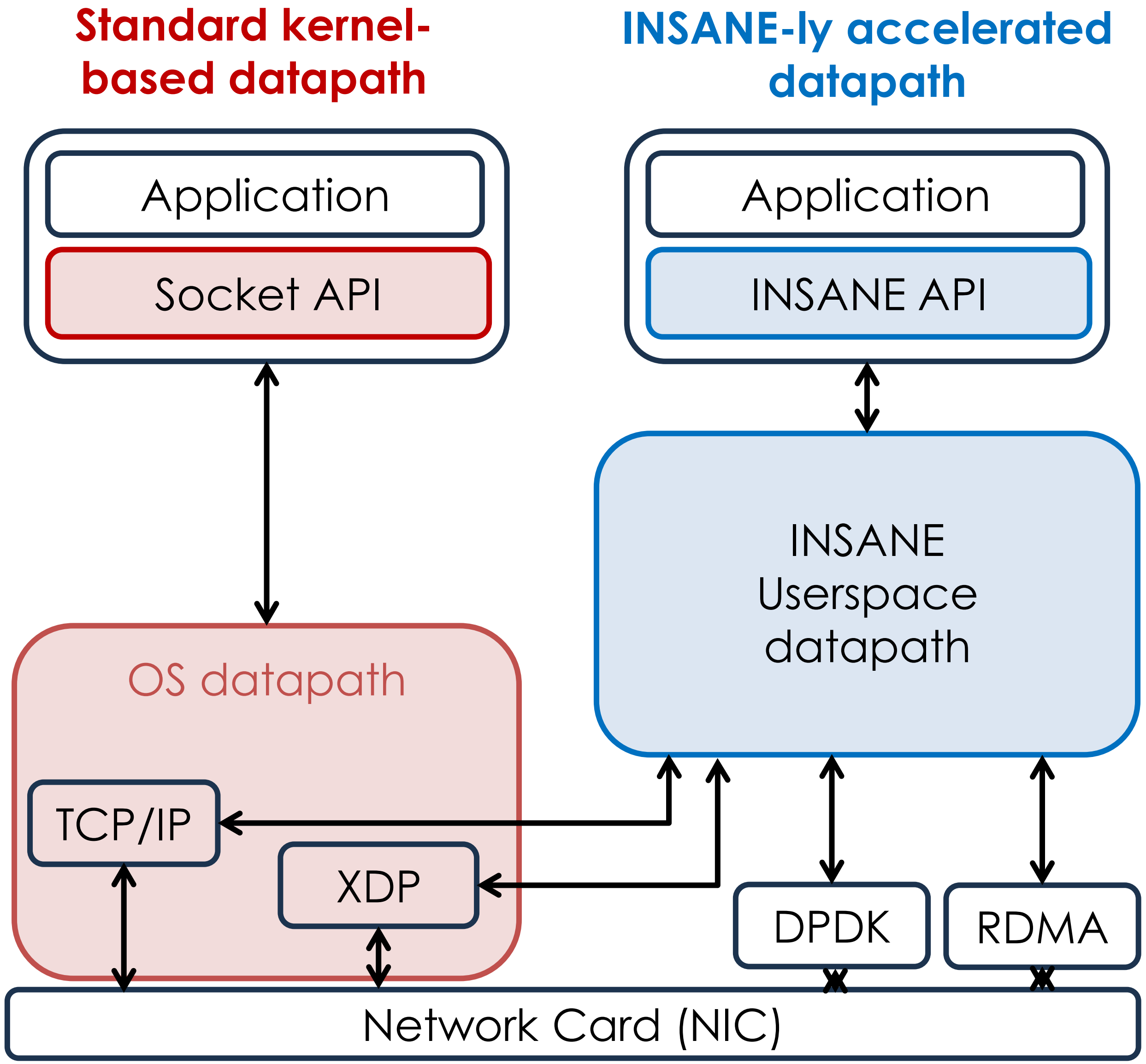
~ **150,000** lines of C code for a userspace TCP stack (TLDK)

rdma-pingpong.c **790** lines of low-level C code

Key Takeaway: High-performance kernel-bypass networking is attainable for everyone. *INSANE makes acceleration (finally!) easy*.

Take-away message

- High-performance kernel-bypass networking is attainable for everyone.
- INSANE makes acceleration (finally!) as easy as standard networking.
- This tutorial leaves with a working template to easily build your own accelerated applications.



Open Source:
https://github.com/MMw-Unibo/INSANE

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lorenzo Rosa

Department of Computer Science and Engineering
University of Bologna – Italy

lorenzo.rosa@unibo.it

site.unibo.it/middleware