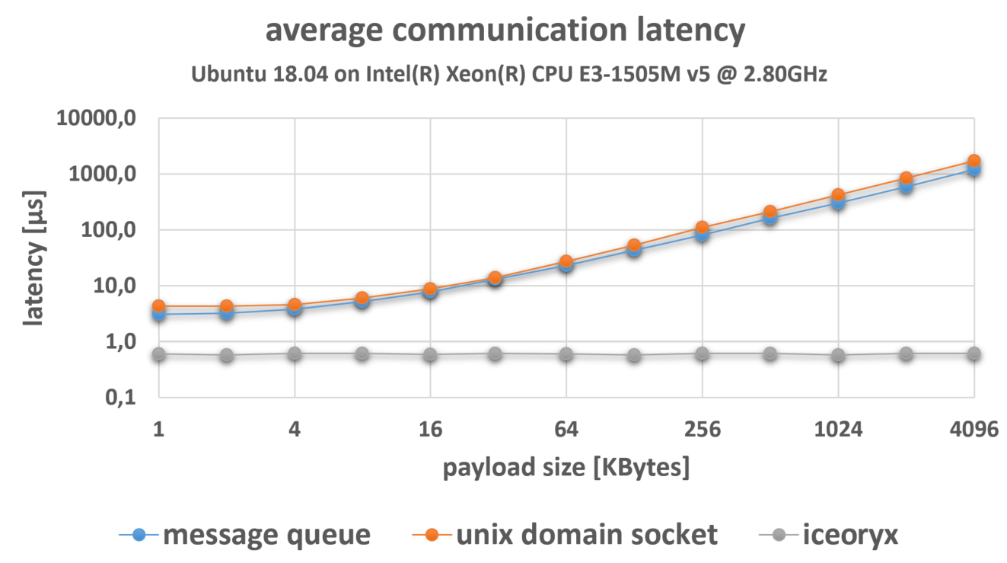
# What Is Iceoryx?

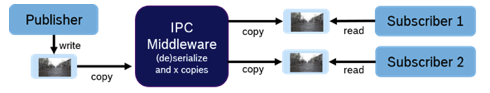
Eclipse Iceoryx is an IPC middleware for various OSs (currently Linux, macOS, QNX, FreeBSD and Windows 10), designed and implemented by Robert Bosch Corp.

It uses **True Zero-Copy** – a **shared-memory** approach that allows to transfer data from publishers to subscribers without a single copy. This ensures data transmissions with constant latency, regardless of the size of the payload.



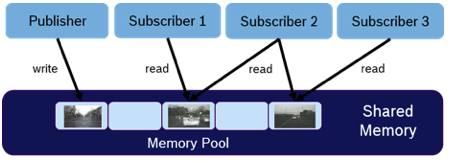
# How It Works?

A copy perspective of a typical IPC middleware solution:



When speeds reach GB/s, every message copy has a significant cost in terms of runtime and latency.

By contrast, true zero-copy communications:



Iceoryx is based on shared memory, which is nothing new. But it takes the approach further, combining with a **publish/subscribe architecture**, **service discovery**, **modern C++**, and **lock-free algorithms**.

A publisher writes the message directly to a chunk of memory that was previously requested from the middleware. When the message is delivered, subscribers receive references to these memory chunks while maintaining their own queue with a configurable capacity.

Every subscriber can have a unique view of which messages are still in process and which can be discarded. **References** are counted and memory chunks are released when they have no readers left .

One important aspect is that publishers can write again while subscribers are still reading because there is no interference from subscribers. The publisher is simply allocated a new memory chunk if the previous one is still in use.

If a subscriber is operating in polling mode and chunks are queued up until the subscriber checks the queue again, we can recycle older memory chunks using the **lock-free queue** in a process we call "safely overflowing."

The lock-free queue allows us to guarantee a memory-efficient contract is made with the subscriber with respect to the maximum number of latest messages that are stored in the queue, no matter how long the time between successive subscriber polls. This is a very helpful approach in common use cases such as those with a high-frequency publisher and a subscriber that is only interested in the latest, greatest message.

Because it is simply passing around **smart pointers**, iceoryx enables data transfers without actually transferring the data.

However, because the **message payload is not serialized**, a message must have the same memory layout for publishers and subscribers. For IPC on a specific processor, this can be ensured by using the same compiler with the same settings.

The message cannot contain any pointers to memory within the process’ internal virtual address space. This restriction also applies to heap-based data structures. If these constraints cannot be fulfilled, iceoryx can still be used with a top-level layer that handles the serialization in and the deserialization from the shared memory. In this case, iceoryx handles the low-layer transport that creates no copy itself.

Iceoryx depends on the **POSIX API**. Currently it supports Linux and QNX as underlying OSs. Because there are sometimes slight API differences, small adaptions might be necessary when porting iceoryx to another POSIX-based operating system.

Full guideline: <https://iceoryx.io/latest/>

## Pros and Cons

**Pros:**

* Message transfers with a latency of less than 1 µs
* Made to handle GBytes/sec data transfers

**Cons:**

* Very high-level, so difficult to customize

## Example

<https://iceoryx.io/latest/examples/icedelivery/>