Breaking the SDV Communication Bottleneck with iceoryx2

Eclipse SDV Community Days

Michael Pöhnl - ekxide IO GmbH

Feb 24, 2026

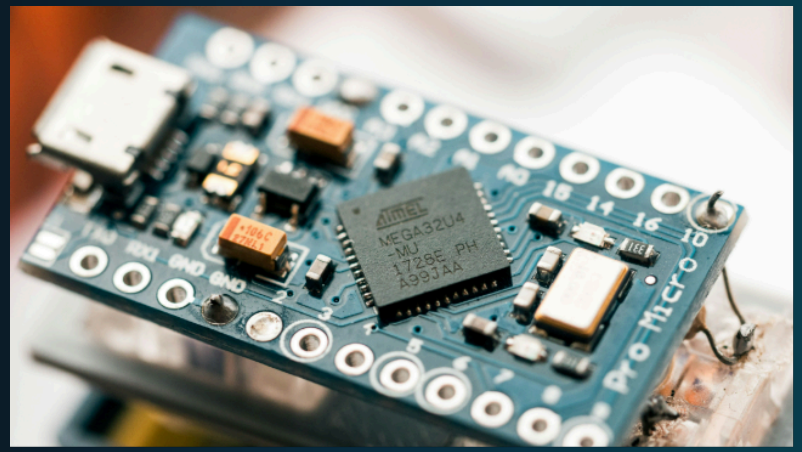
Based on true stories from 10 years in SDV

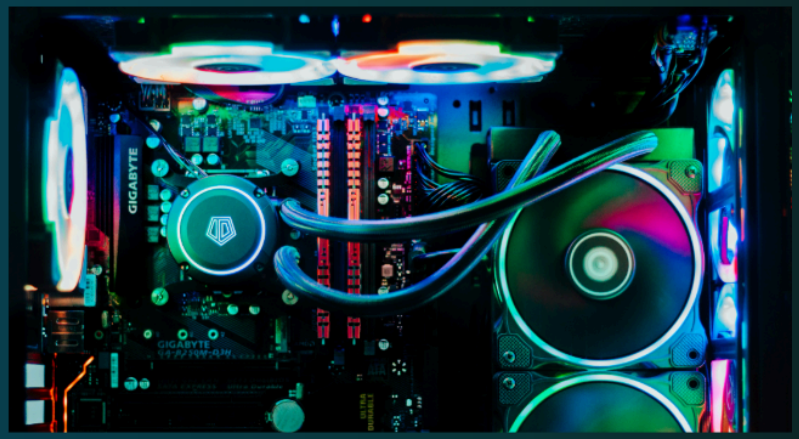
Tales from the glorious SDV Vision

What people thought SDV would be

Photo by Stephan Eickschen on Unsplash

From ECUs to Vehicle Computers





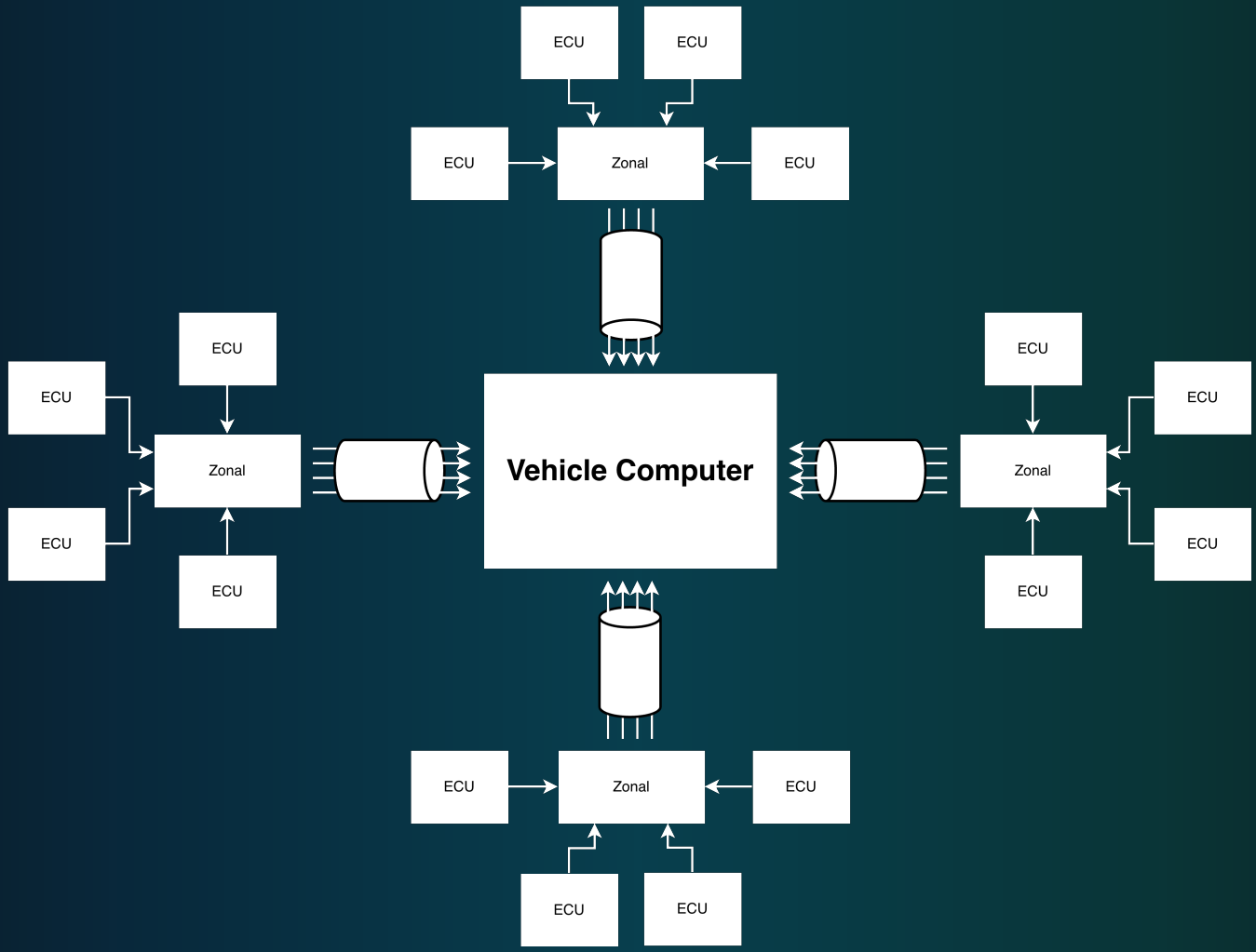
Photos by Harrison Broadbent on Unsplash and Tai Bui on Unsplash

Lots of everything

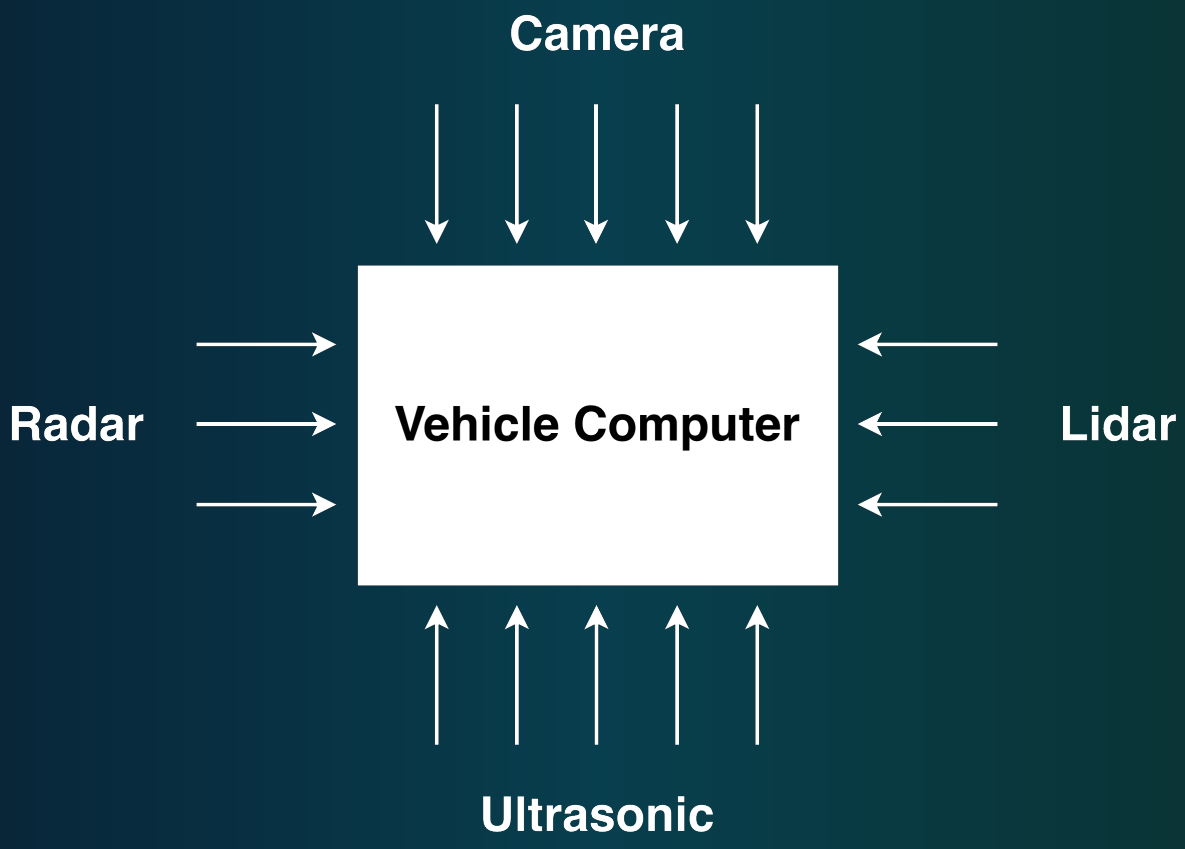
- Many, many cores
- Plenty of memory
- “Real” operating system

Photo by Austin Distel on Unsplash

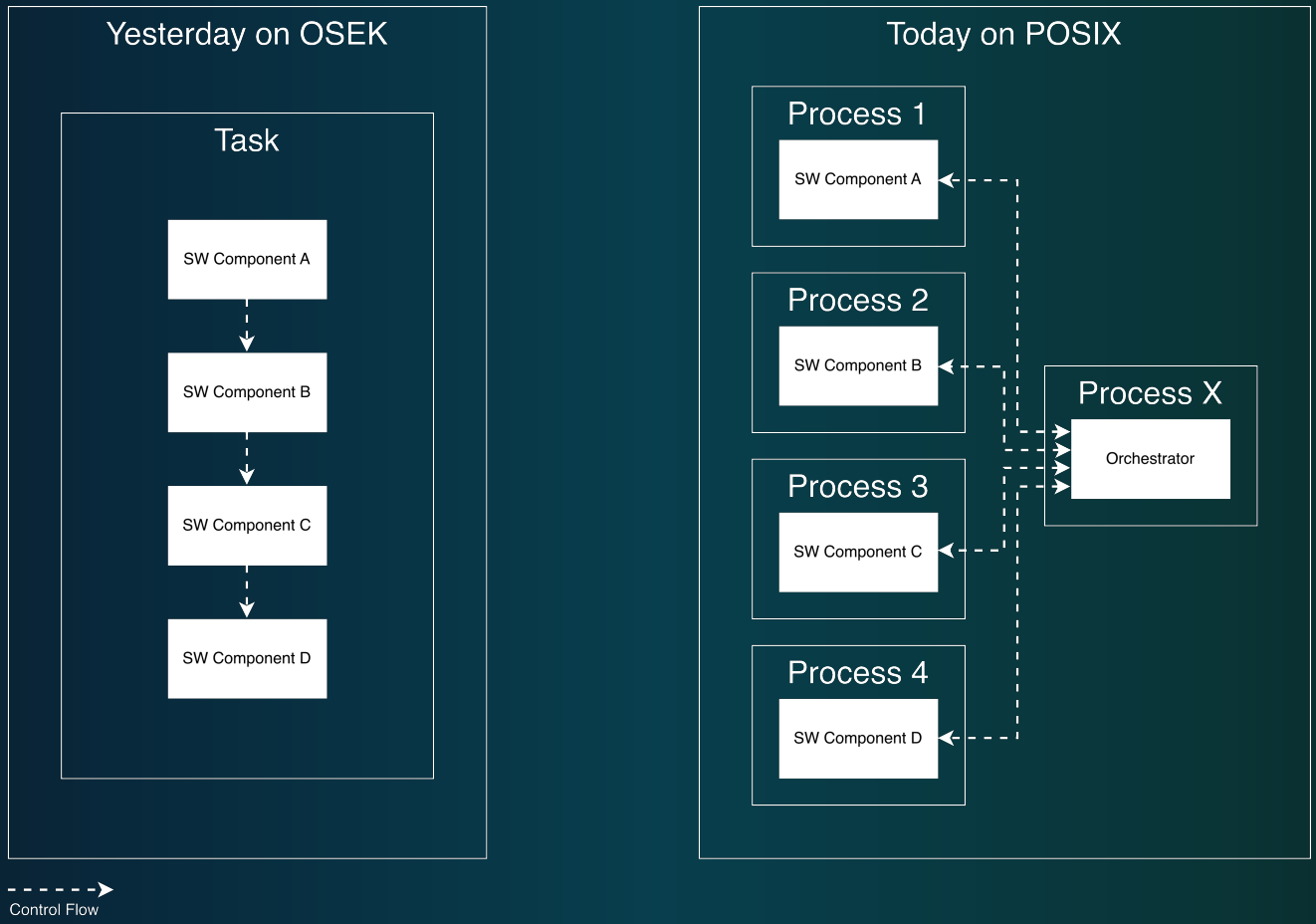
The sink of all data, the source of all power



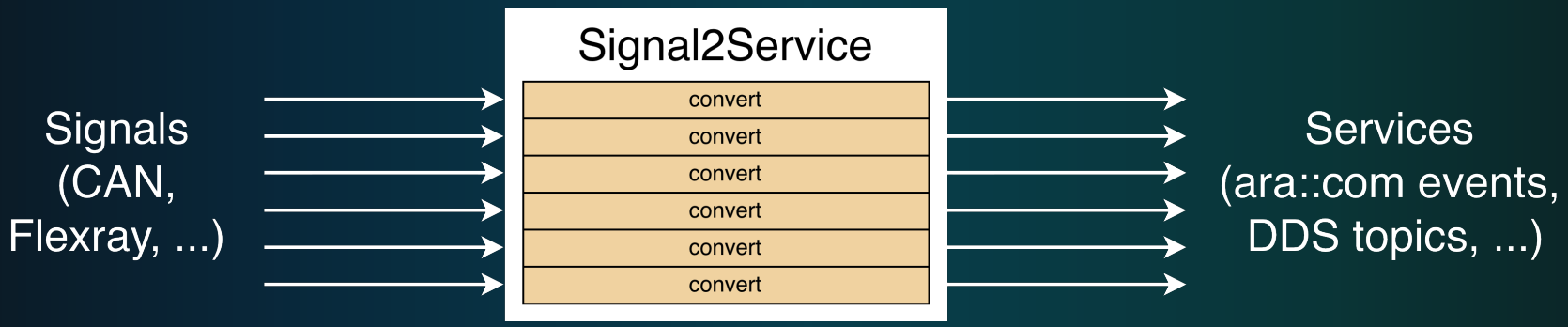
More of sensing enables more of acting



Divide and Conquer



Signal to Service, literally



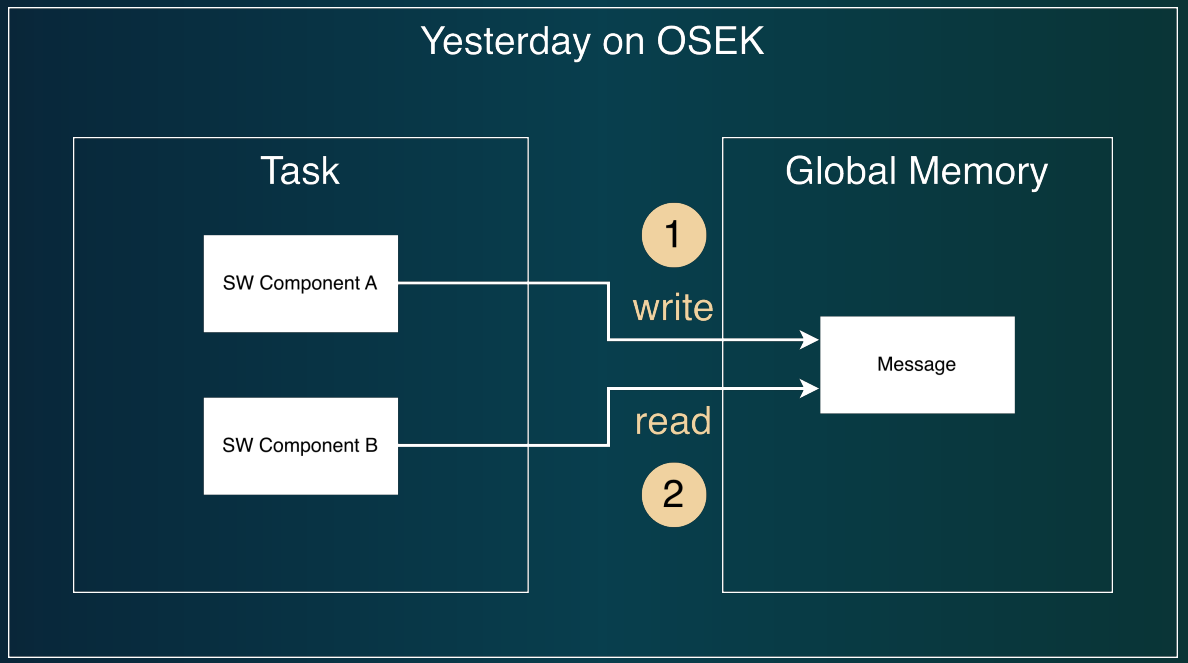
When reality hits hard

Many processes, many inputs, many services, loads of IPC

- Integration of a lot of legacy and a lot of new SW components
- Every SW component is a process ⟶ a lot of processes
- Zonal ECUs and sensors provide input ⟶ a lot of inputs
- Every signal becomes a service ⟶ a lot of services
- Data transfers are inter-process ⟶ a lot of IPC

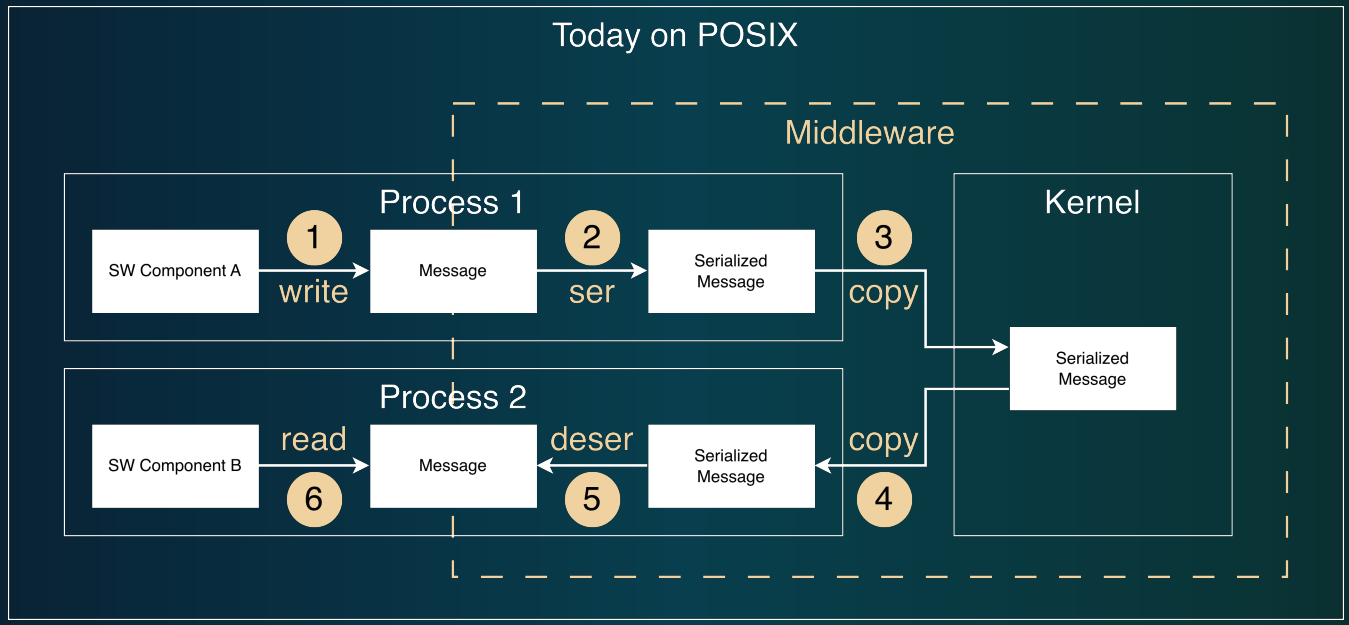
What a data transfer used to be on OSEK

- In-memory passing of messages in a global address space
- A single copy of (typically small) data in case of concurrent access



What a data transfer is on POSIX

- Message transfer across separate virtual address spaces
- Worst case includes all of this:
  - de/serialization, copies, context switches, user–kernel mode switches



Why things do not scale

- “Vehicle Computers“ are often just bigger ECUs with 2 to 8 cores
- Hundreds of POSIX processes
- Hundreds to thousands of threads (good luck with determinism)
- Thousands to tens of thousands of IPC data transfers per second
- IPC data transfers come with syscalls, context switches, copies…

Scaling limits: message rate

Even 1–3 context switches per message add up

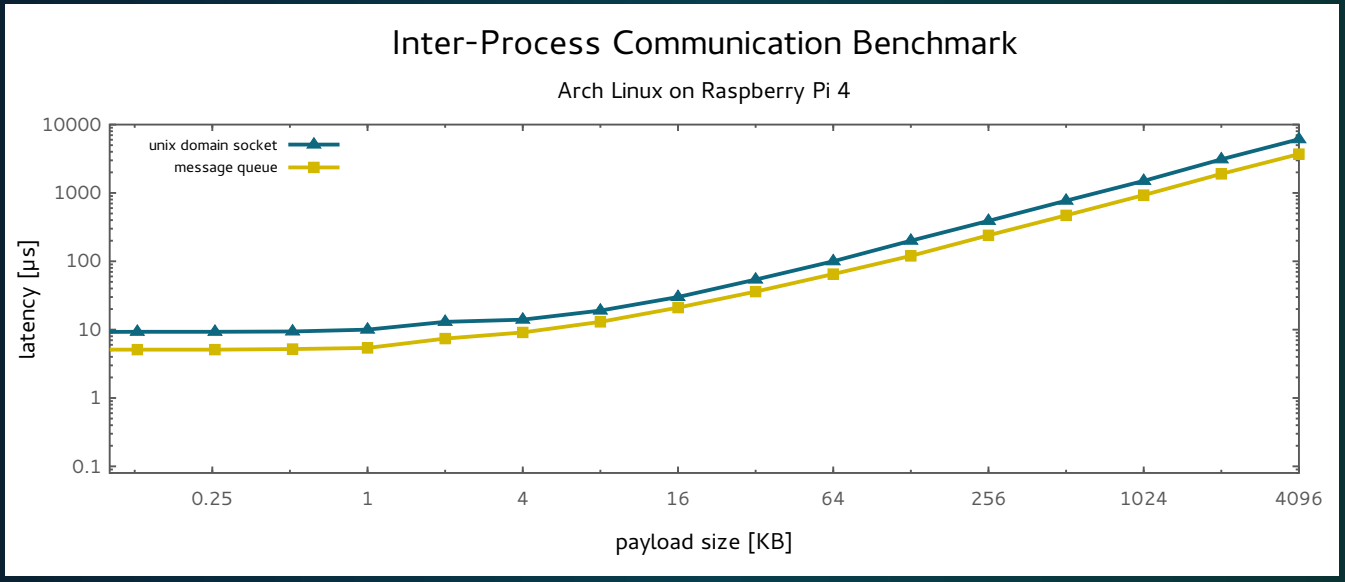
- Many middleware solutions use per-message callbacks
- As a result, message delivery typically triggers a context switch
- Internal middleware threads can introduce additional context switches
- Thousands of messages per second ⟶ thousands of context switches

10,000 messages per second
× 1–3 context switches
= tens of thousands of context switches per second

Scaling limits: message size

Copies and serialization inside the IPC come with a price

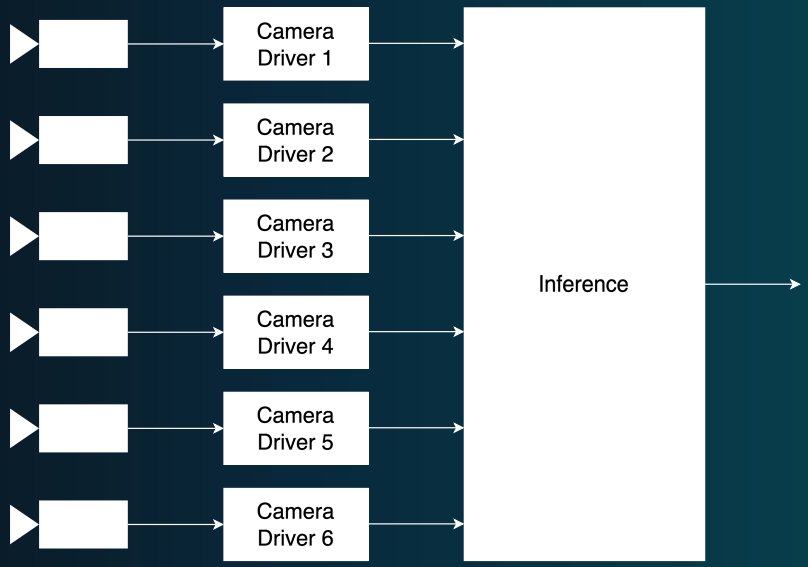
- This takes CPU cycles, the bigger the message, the more of them
- This introduces latency, the bigger the message, the higher it is
- Advanced ADAS and AD systems can have tens of GB/s IPC



Scaling limits: message size

Real-life example

- Autonomous driving based on an end-to-end AI model
- 6 cameras × ~35 MB × 20 fps ⟶ more than 4 GB/s camera data stream
- Now think: copying on sender side, copying on receiver side, recording...



Unless you’re currently performing a context switch, you’re probably copying some data

Isch over SDV?

Is there a way out?

Photo by Alexandre Debiève on Unsplash

The communication overhead equation

$$\text{communication overhead} = \frac{\text{system decomposition}}{\text{middleware efficiency}}$$

You have to take care of your architecture

Decomposition is good, but there is also over-decomposition

Photo by Gayatri Malhotra on Unsplash

iceoryx2 takes care of the rest

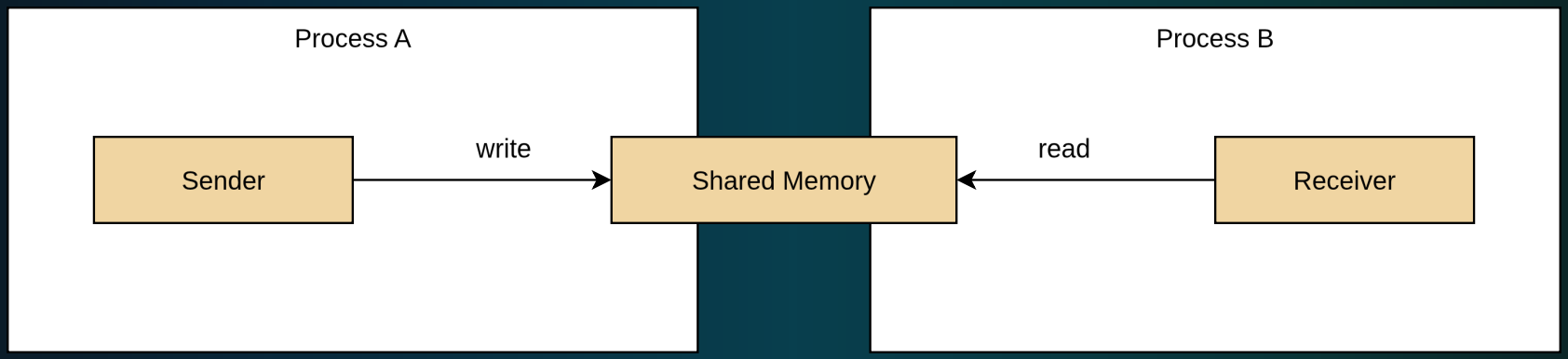
Introduction to Eclipse iceoryx™

What is Eclipse iceoryx?

- OSS project hosted by the Eclipse Foundation
- Inter-process communication (IPC) based on shared memory
- Kicked off in 2015; impacted AUTOSAR Adaptive ara::com and ROS 2 APIs
- Open-source since 2019; second generation (iceoryx2) started in 2023
- Based on 70+ years of combined experience in high-performance IPC

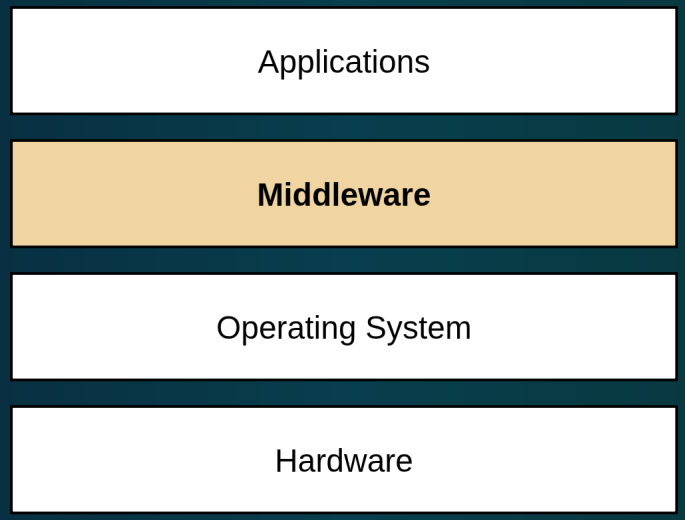
How does it work?

- Sender writes directly into shared memory
- Receiver reads directly from shared memory
- iceoryx provides the complete communication infrastructure
- iceoryx takes care of discovery, message delivery, memory management
- This is what we call true zero-copy communication



Why open-source middleware is great

- Middleware is infrastructure software, needed across many domains
- It's not trivial: low-level stuff, multi-threaded programming, etc.
- Reinventing the wheel makes no sense if your business is not middleware
- With a robust OSS solution, teams can focus on their domain problems

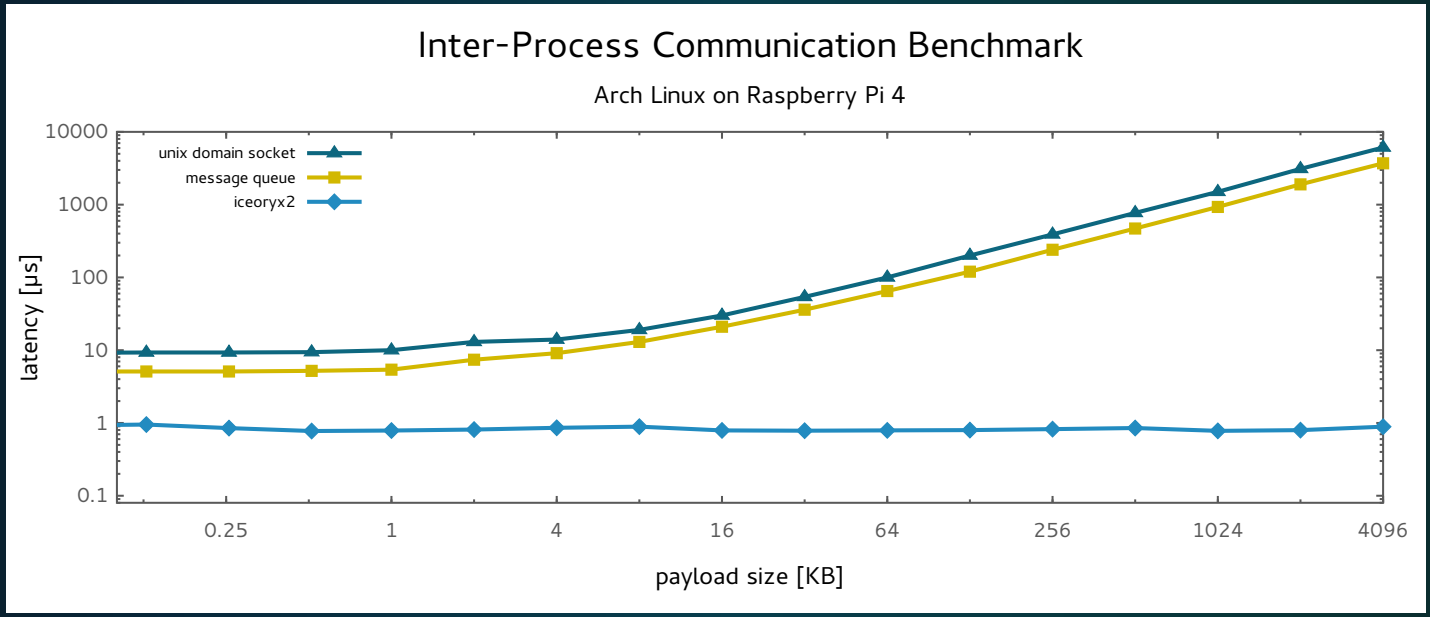


iceoryx2, a solid SDV foundation

Efficiency

True zero-copy IPC means

- Passing messages between processes without a single copy
- Constant sub-microsecond latency, independent of message size
- More than 1000x lower latency for MB messages (think < µs, not > ms)



Efficiency

What it enables

- Dozens of GB/s IPC with ultra-low CPU load
- Fast reaction times from sensing to acting
- Enables use of smaller, cheaper CPUs for the same workload
- Scalability: going embedded without going crazy

Photo by Leo_Visions on Unsplash

Determinism

iceoryx2 comes with

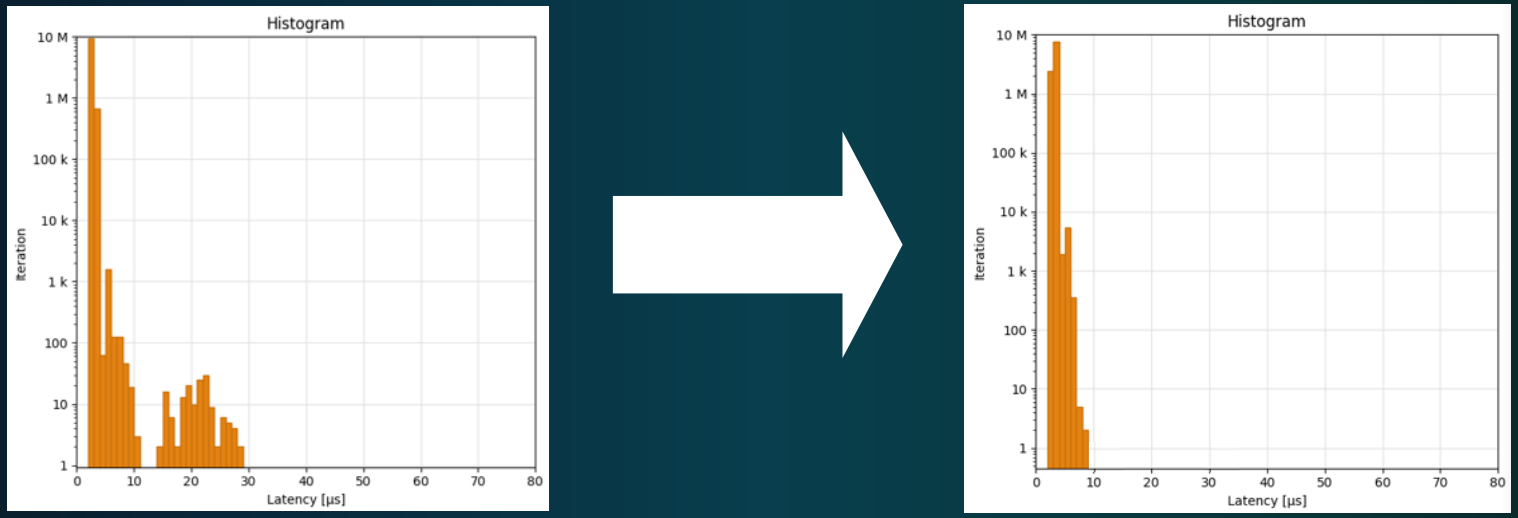
- No internal threads, no syscalls in the data delivery path
- No hidden blocking calls, no heap allocation during runtime
- No central daemon, no surprises behind the scenes

Photo by Keagan Henman on Unsplash

Determinism

What it enables

- Stable low latency with minimal jitter
- Predictable execution without hidden scheduling effects
- Deterministic timing behavior for real-time applications
- Reduced scheduling overhead, leading to lower CPU usage



Left side: interference from middleware thread. Right side: no middleware threads

Flexibility

- Written in Rust, runs everywhere
  - Linux, Windows, macOS, QNX, VxWorks, FreeBSD, eMCOS, bare metal
  - In addition to Rust APIs, language bindings for C, C++, Python, C#

- Huge set of messaging patterns
  - publish/subscribe, request/response, key/value storage (blackboard)
  - Event mechanism for data-driven triggering of execution

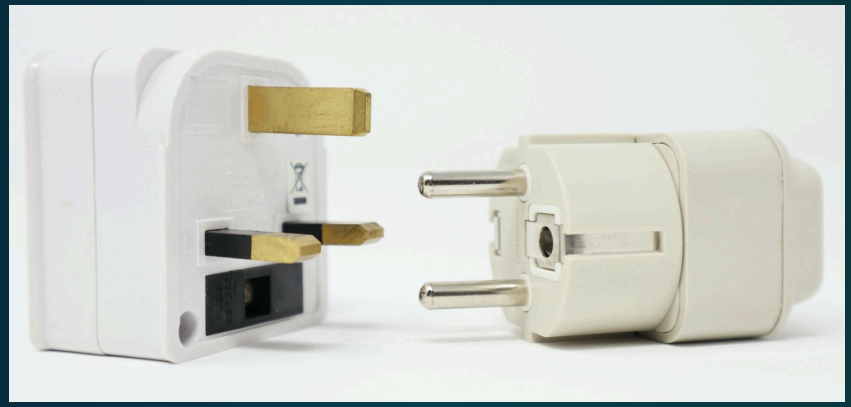
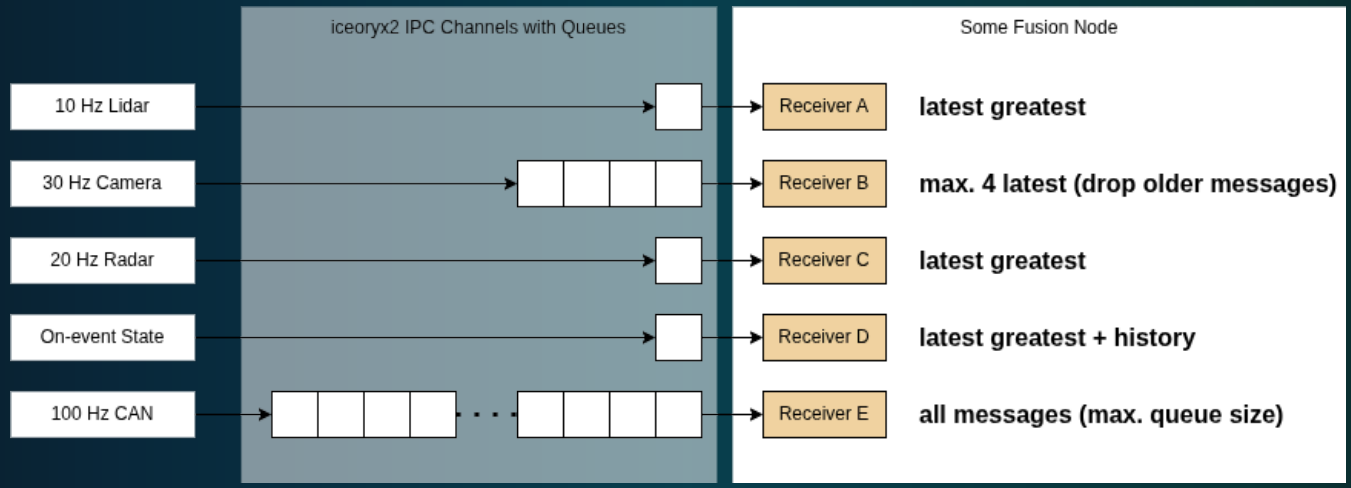


Photo by Call Me Fred on Unsplash

Flexibility

- Per receiver queues with configurable size
- Queue overflow behavior can be configured
  - Return an error and drop the new message
  - Block the sender, wait for the receiver
  - Overflow and drop the oldest message
- Optional history on the sender side for late joining receivers



See my FOSDEM talk for a more technical overview

iceoryx2 adopters

A vast community with a need for low latency and high-volume data transfer

iceoryx within the Eclipse ecosystem

iceoryx or iceoryx2

- can be used as IPC plugin in Cyclone DDS
- can be used as IPC plugin in eCAL
- can be used as IPC transport in uProtocol
- is used within OpenSOVD
- is used within S-CORE
- uses Zenoh to connect networked systems

Questions?

iceoryx