



copper-rs

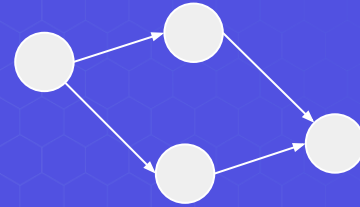
An open source Rust first software engine for robotics.

⇒ Software “engine” for Robots?



⇒ Goal: Focus on the end product, not just a demo.





⇒ **Models the set of tasks as a kind of flexible set of microservices.**

- asynchronous
- non deterministic
- foot-gun bazooka situation of C++
- terrible performance
- in practice: Ubuntu is mandatory to run this.
- very challenging to put in production as a company
- Safety certifying a system using this?





copper-rs

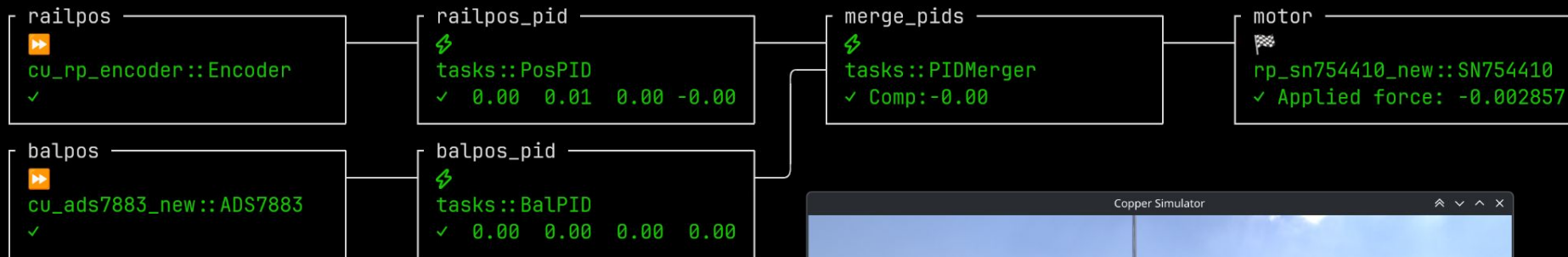


- **deterministic** *by design*
- **Designed for safety** - conceptually Copper is a compiler
- **performant**
 - ◆ High bandwidth
 - ◆ Low latency
 - ◆ Low jitter
- allows developers to **focus on algorithms**, not buffers
- **Can target both CPU & MCU**



A Quick tour?





Defining the DAG

RON (Rust Object Notation)

copperconfig.ron

```
1 (
2   tasks: [
3     (
4       id: "src",
5       type: "tasks::CaterpillarSource",
6     ),
7     (
8       id: "ct-0",
9       type: "tasks::CaterpillarTask",
10    ),
11    (
12      id: "gpio-0",
13      type: "cu_rp_gpio::RPGpio",
14      config: {
15        "pin": 4,
16      },
17    ),
18  ],
19 )
```

```
96   cnx: [
97     // Make a caterpillar by propagating messages from the source to
98     (src: "src", dst: "ct-0", msg: "cu_rp_gpio::RPGpioPayload"),
99     (src: "ct-0", dst: "ct-1", msg: "cu_rp_gpio::RPGpioPayload"),
100    (src: "ct-1", dst: "ct-2", msg: "cu_rp_gpio::RPGpioPayload"),
101    (src: "ct-2", dst: "ct-3", msg: "cu_rp_gpio::RPGpioPayload"),
102    (src: "ct-3", dst: "ct-4", msg: "cu_rp_gpio::RPGpioPayload"),
103  ],
104 )
```



Implementing a task

```
#[derive(Default)]
pub struct CaterpillarSource {
    state: bool,
}

impl Freezable for CaterpillarSource {
    fn freeze<E: Encoder>(&self, encoder: &mut E) → Result<(), EncodeError> {
        Encode::encode(&self.state, encoder)
    }

    fn thaw<D: Decoder>(&mut self, decoder: &mut D) → Result<(), DecodeError> {
        self.state = Decode::decode(decoder)?;
        Ok(())
    }
}

impl CUSrcTask for CaterpillarSource {
    type Output<'m> = output_msg!(RPGpioPayload);

    fn new(_config: Option<&ComponentConfig>) → CuResult<Self>
    where
        Self: Sized,
    {
        Ok(Self { state: true })
    }

    fn process(&mut self, clock: &RobotClock, output: &mut Self::Output<'_>) → CuResult<()> {
        // forward the state to the next task
        self.state = !self.state;
        output.set_payload(RPGpioPayload { on: self.state });
        output.tov = Tov::Time(clock.now());
        output.metadata.set_status(self.state);
        Ok(())
    }
}
```



This is just a normal Rust workflow

```
# start a project
cargo cunew [destination folder]

# build
cargo build

# unit test
cargo test

# cross compile to arm
cross build --target aarch64-unknown-linux-gnu
⇒ 1 static executable to deploy
```



An incredible open source ecosystem already!

faer

nalgebra

Kornia

Bevy + Avian

Iceoryx2

Zenoh

Rerun

Foxglove

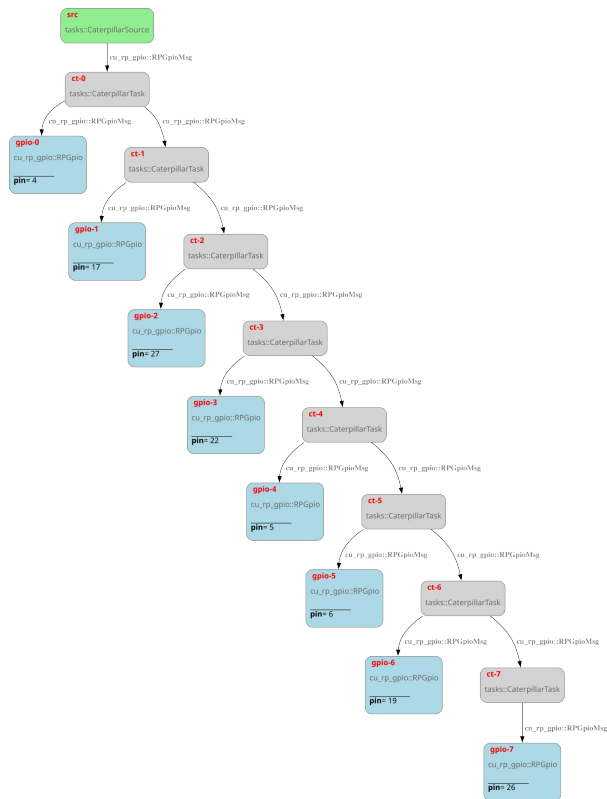
... and much more



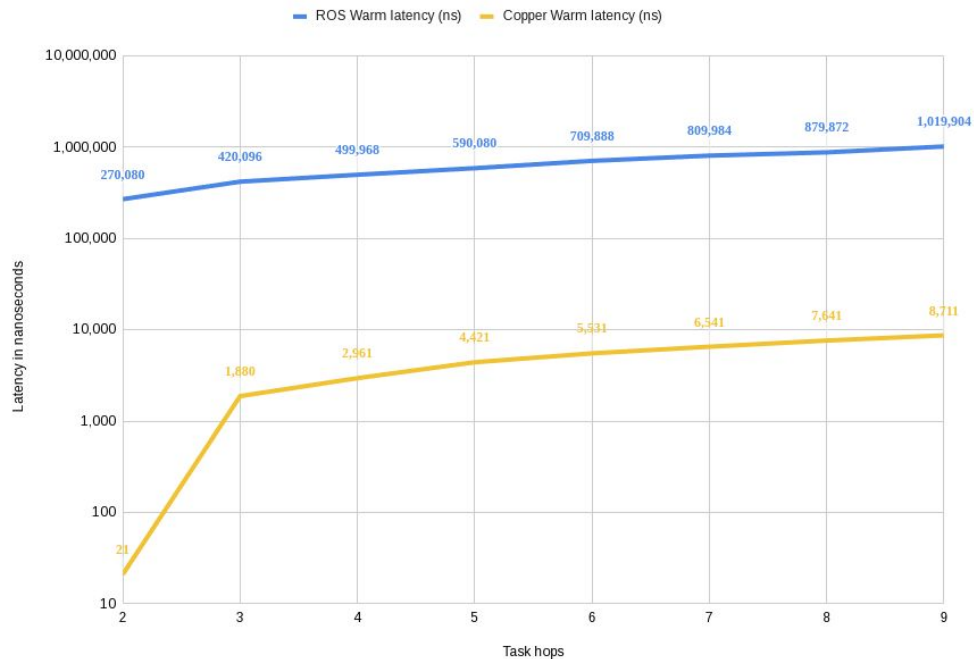


Check us out on Github, and star the repo!

100x less latent at runtime than ROS2



ROS2 [Iron / C++] vs Copper [pre-alpha / Rust] -- Latency



Linear data logging

31000x faster than ROS2 (sqlite backend)

But MCAP? Copper is still 12x faster ROS2 logging with MCAP



Structured logging

10x reduction size vs text logging

makes debug log a typed time series like everything else

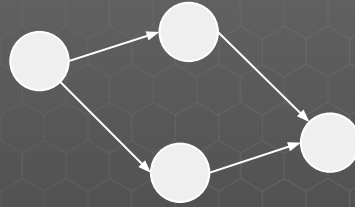
```
debug!("This string won't be stored nor interpreted on the robot", myvaluenam = 42);
```



A Quick demo?



Conceptual view



Implementation with
instant **Copper**
feedback

```
fn process(  
    &mut self,  
    clock: &RobotClock,  
    _empty_msg: &mut  
    CuMsg<Self::Output>,  
    ) -> CuResult<()> {  
    Ok(())  
}
```

Copper infers an
execution plan.



Deploy and test.



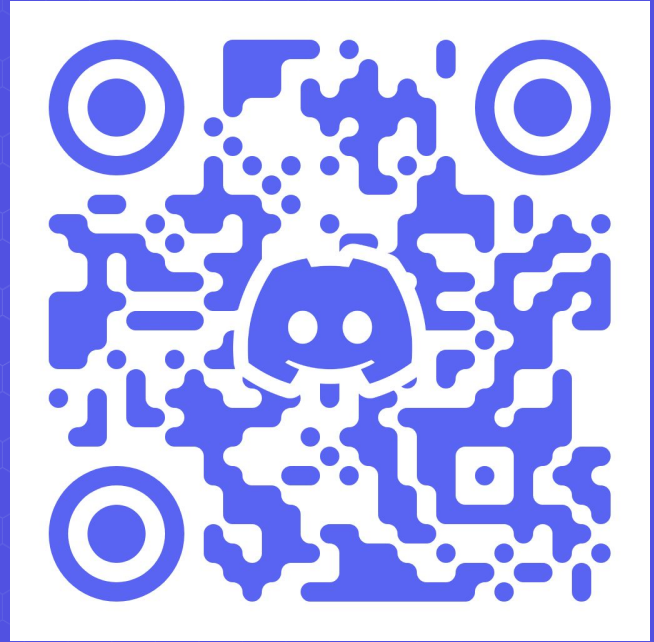
Collect data

```
0101010101011001  
100101010100110010  
110011010110101010  
101010000111000111  
100101110000101010  
100111000010100110
```

Automated
execution strategy
tuning

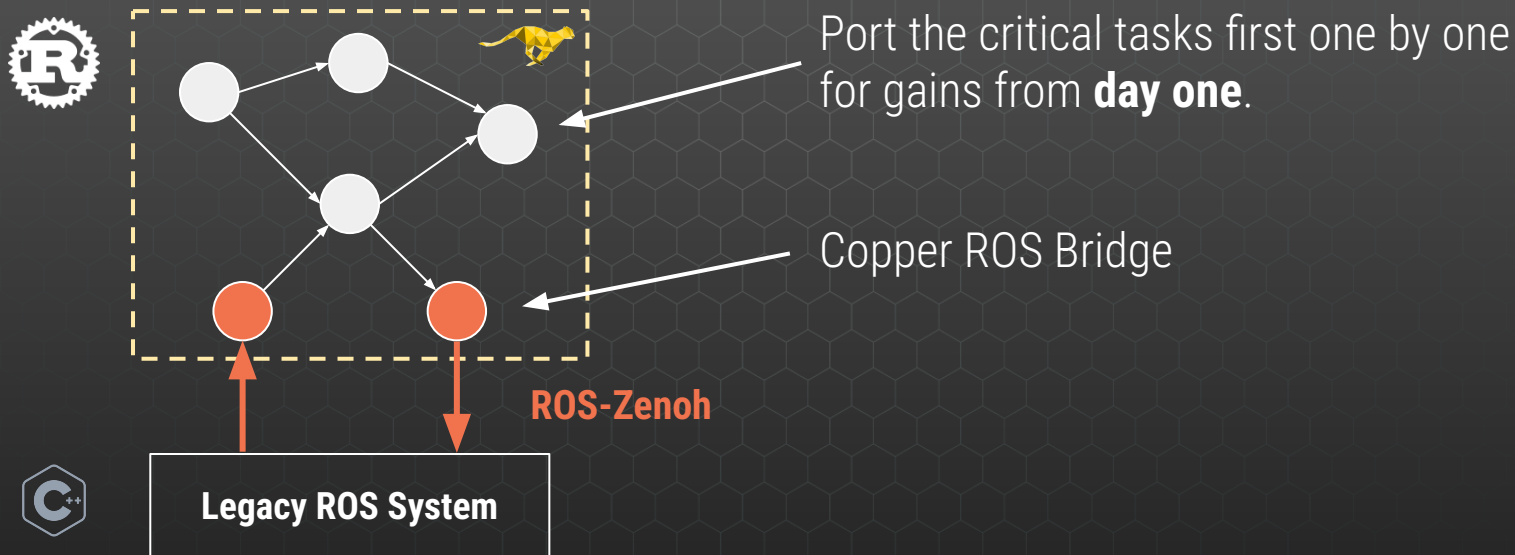


Rust for Robotics



But what about existing ROS users?

→ Provide a **progressive** path to migrate their systems



* Leveraging existing bridges like rosrust

Some robotics companies using Rust:

- Scythe
- Bedrock robotics
- Komboi
- farm-ng / bonsai





Konboi One

2025 Commercial Launch of our Secret Weapon Hypermile, our AI eco-driving Co-Pilot

Hypermile, our **aftermarket kit**, reduces fuel consumption while efficiently harvesting huge amounts of data to train our AI Pilot:

11% fuel savings with superhuman efficiency driving

→ **22t less CO2** per truck per year

→ **+50% net margin** for our clients

★ **huge market demand**

❖ **7k€ ARR** per truck



2024

Prototype Completion

2025

First revenues

2026

Scale

Konboi AI

Superhuman AI Pilot

17

