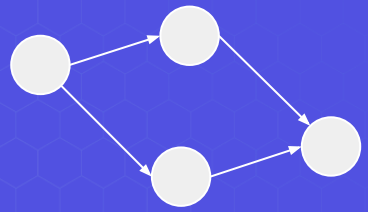
copper-rs

An open source Rust first software engine for robotics.

ROS

C++



⇨ 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?

An incredible community & open source ecosystem already!

@yangrobotics
@bengalus
@fuzzybunny258
@mikemikemike
@Luc F
@kamibo

faer
nalgebra
Kornia
Bevy + Avian
Iceoryx2
Zenoh
Rerun
Foxglove
… and much more

⭐🙏🥹 Check us out on Github, and star the repo!

Spare slides

[1] SysInfo [2] DAG [3] Latencies [4] Memory Pools [5] Debug Output [q] Quit | Scroll: hjkl or ↑↓←→

railpos
⏩
cu_rp_encoder::Encoder
✓

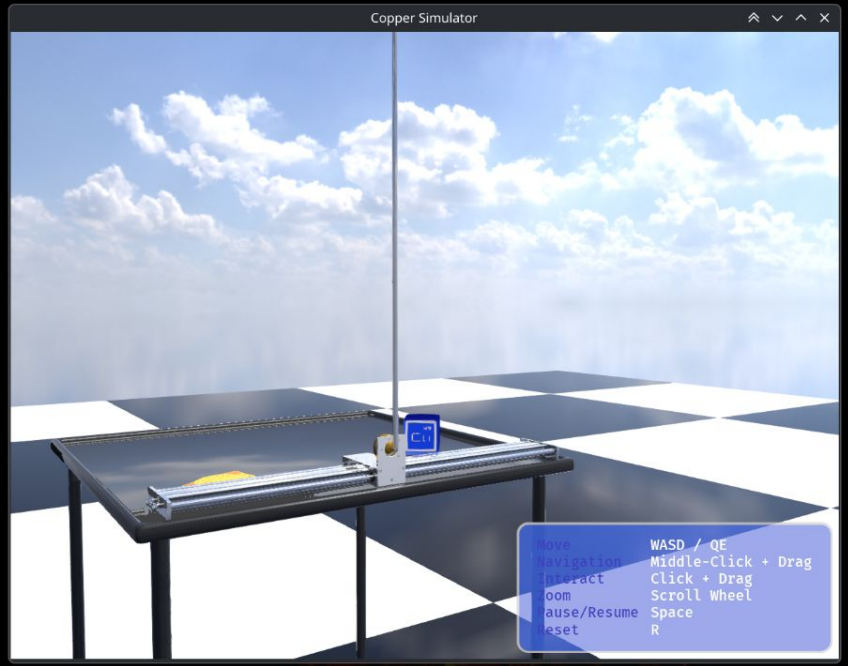
railpos_pid
↯
tasks::PosPID
✓ 0.00 0.01 0.00 -0.00

merge_pids
↯
tasks::PIDMerger
✓ Comp:-0.00

motor
🏁
rp_sn754410_new::SN754410
✓ Applied force: -0.002857

balpos
⏩
cu_ads7883_new::ADS7883
✓

balpos_pid
↯
tasks::BalPID
✓ 0.00 0.00 0.00 0.00



Defining the DAG

RON (Rust Object Notation)

copperconfig.ron

```
(
    tasks: [
        (
            id: "src",
            type: "tasks::CaterpillarSource",
        ),
        (
            id: "ct-0",
            type: "tasks::CaterpillarTask",
        ),
        (
            id: "gpio-0",
            type: "cu_rp_gpio::RPGpio",
            config: {
                "pin": 4,
            },
        ),
```

```
    cnx: [
        // Make a caterpillar by propagating messages from the source to
        (src: "src",  dst: "ct-0",   msg: "cu_rp_gpio::RPGpioPayload"),
        (src: "ct-0", dst: "ct-1",   msg: "cu_rp_gpio::RPGpioPayload"),
        (src: "ct-1", dst: "ct-2",   msg: "cu_rp_gpio::RPGpioPayload"),
        (src: "ct-2", dst: "ct-3",   msg: "cu_rp_gpio::RPGpioPayload"),
        (src: "ct-3", dst: "ct-4",   msg: "cu_rp_gpio::RPGpioPayload"),
```

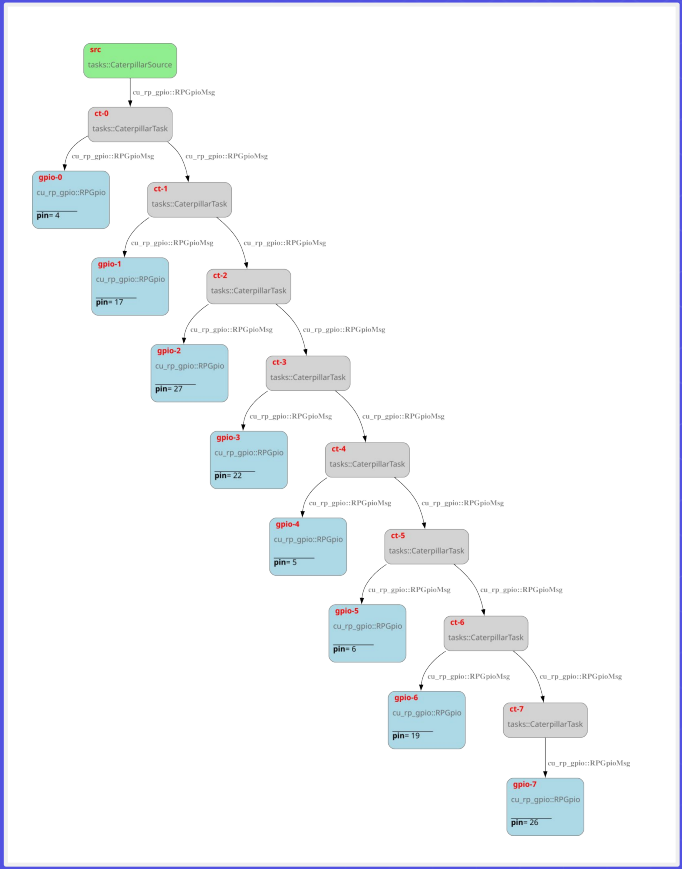
Implementing a task

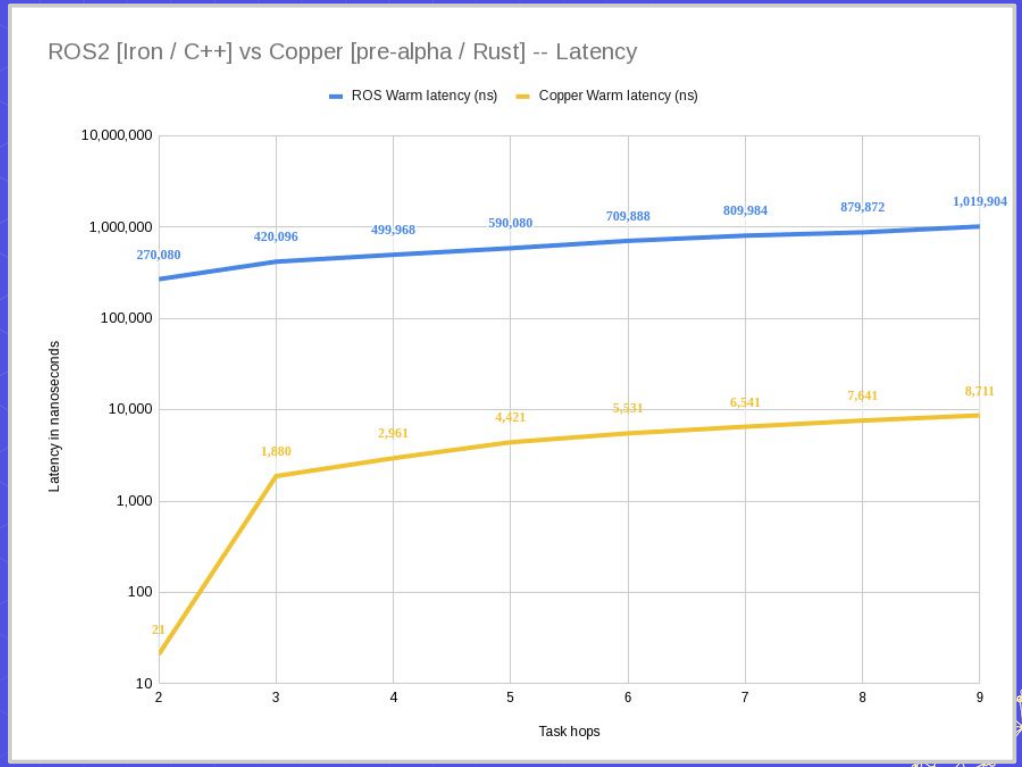
```rust
#[derive(Default)]
pub struct CaterpillarSource {
    state: bool,
}

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
```

100x less latent at runtime than ROS2





Linear data logging

31000x faster than ROS2 (sqlite backend)

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

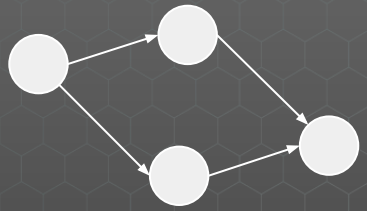
Structured logging

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", myvaluename = 42);
```

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**.

Automated execution strategy **tuning**

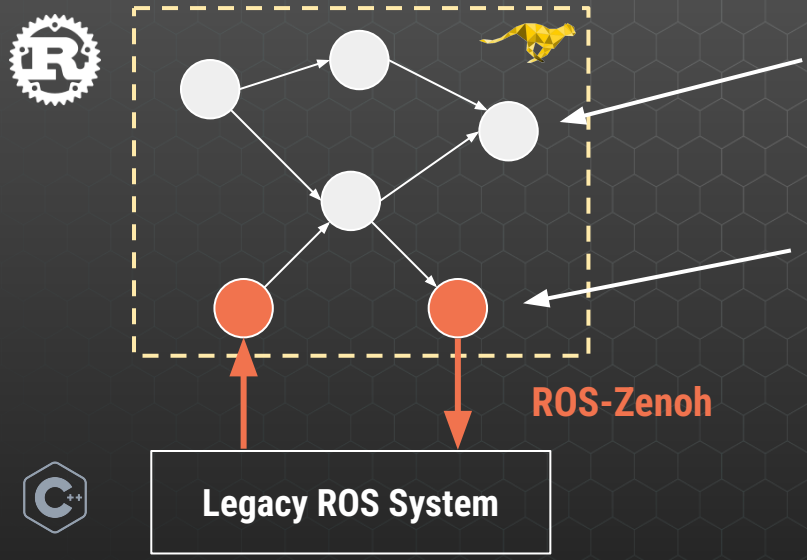
Collect data

```
0101010101011001
1001010100110010
1100110101101010
1010100001110001 11
1001011100001010 10
1001110000101001 10
```

Deploy and test.

But what about existing ROS users?

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



Port the critical tasks first one by one for gains from **day one**.

Copper ROS Bridge

* Leveraging existing bridges like rosrust