Guillaume Binet
AKA “gbin”



Rust for Robotics







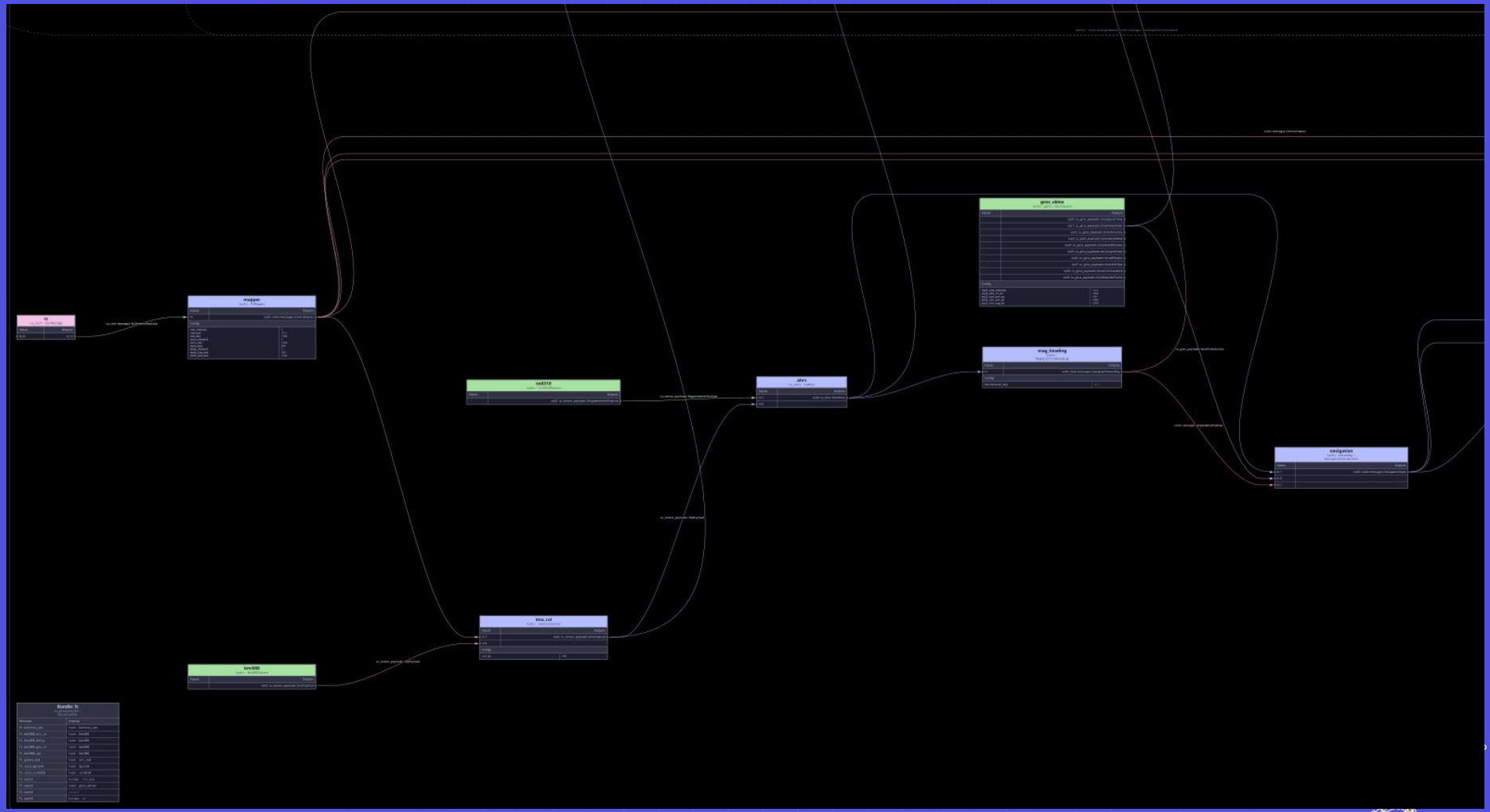
~~Rust for Robotics~~

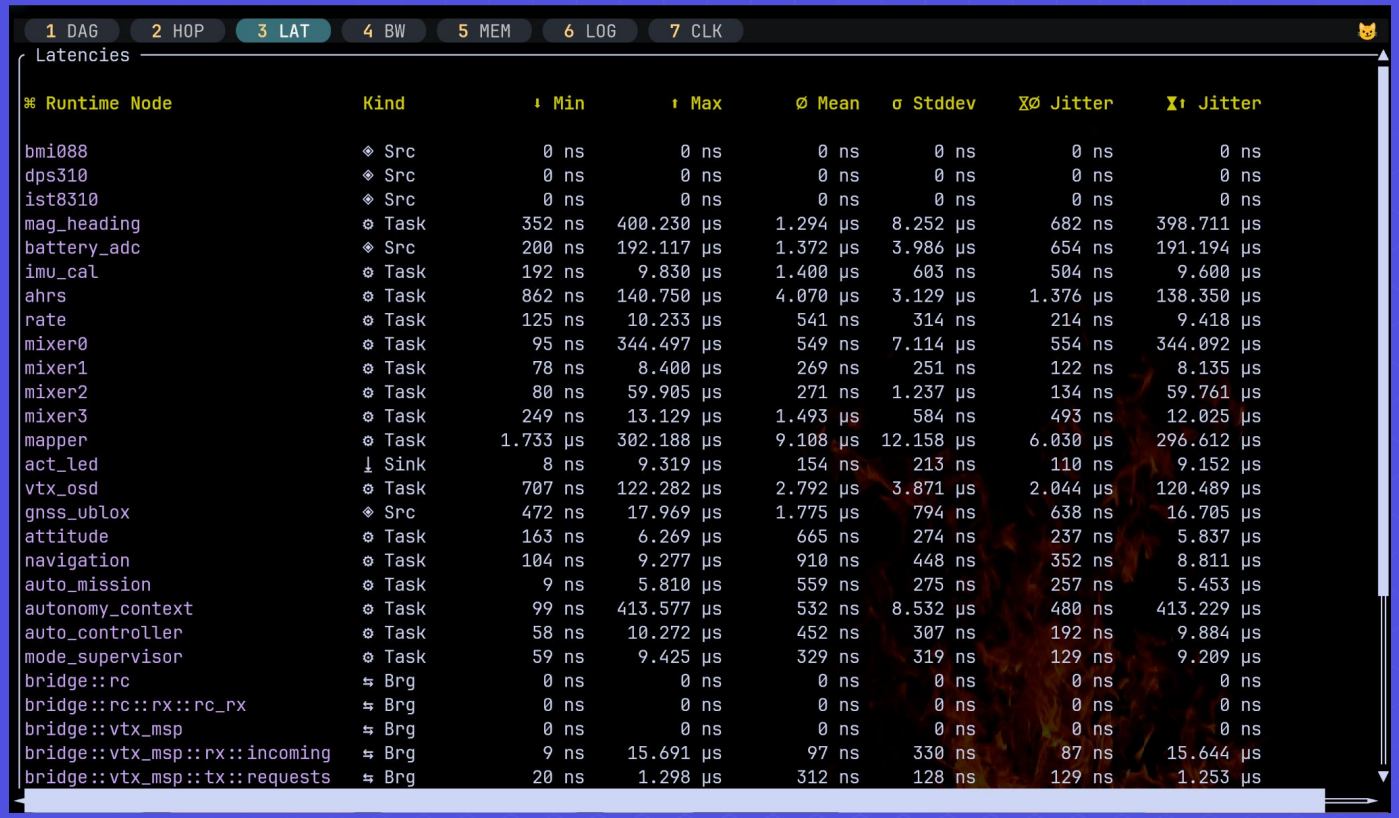
Robotics for Rust



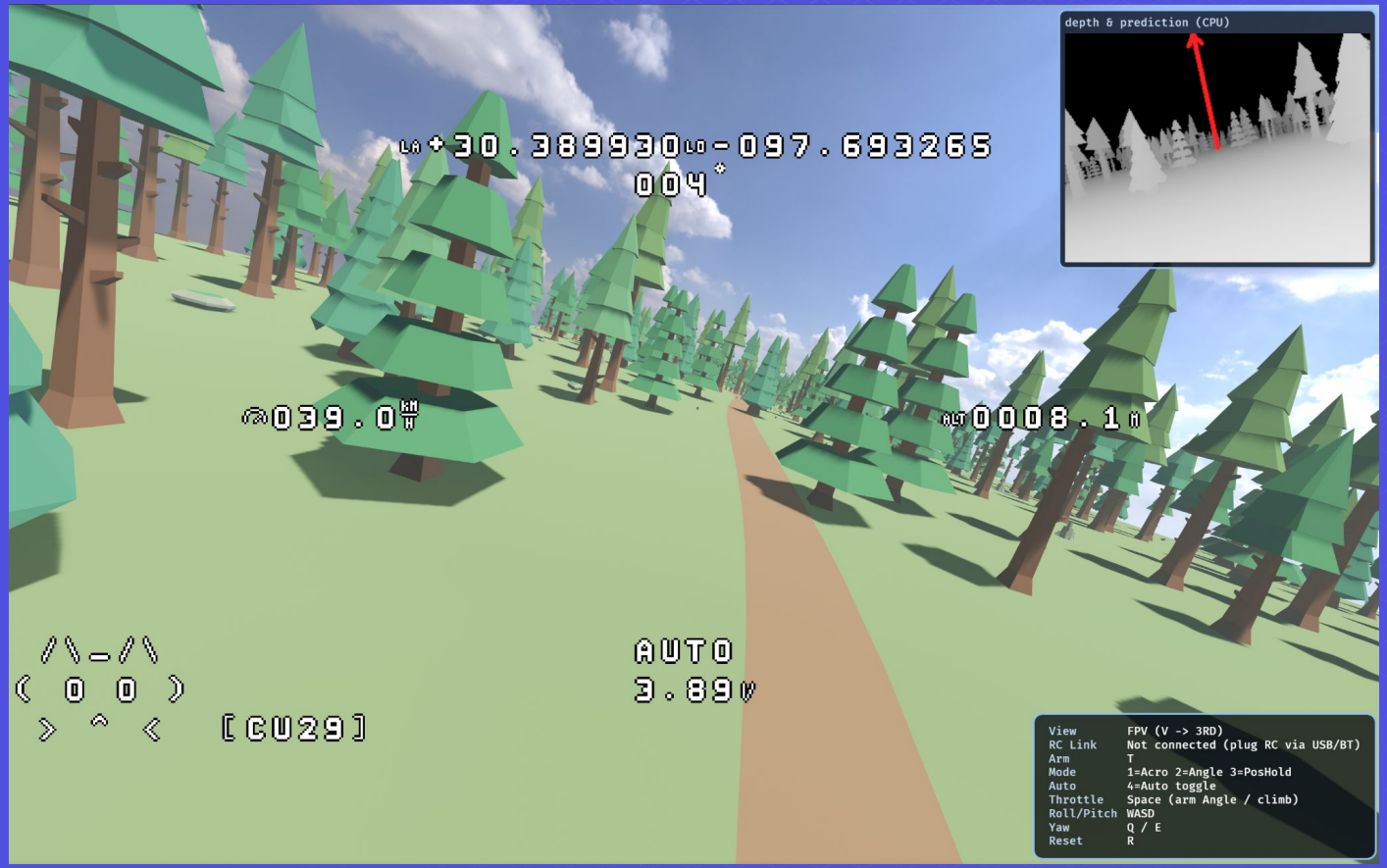
Demo / intro to autonomous robotics



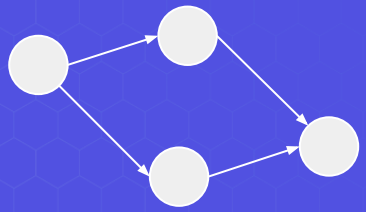




⌘ Runtime Node	Kind	↓ Min	↑ Max	Ø Mean	σ Stddev	⧖Ø Jitter	⧖↑ Jitter
bmi088	◈ Src	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
dps310	◈ Src	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
ist8310	◈ Src	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
mag_heading	⚙ Task	352 ns	400.230 µs	1.294 µs	8.252 µs	682 ns	398.711 µs
battery_adc	◈ Src	200 ns	192.117 µs	1.372 µs	3.986 µs	654 ns	191.194 µs
imu_cal	⚙ Task	192 ns	9.830 µs	1.400 µs	603 ns	504 ns	9.600 µs
ahrs	⚙ Task	862 ns	140.750 µs	4.070 µs	3.129 µs	1.376 µs	138.350 µs
rate	⚙ Task	125 ns	10.233 µs	541 ns	314 ns	214 ns	9.418 µs
mixer0	⚙ Task	95 ns	344.497 µs	549 ns	7.114 µs	554 ns	344.092 µs
mixer1	⚙ Task	78 ns	8.400 µs	269 ns	251 ns	122 ns	8.135 µs
mixer2	⚙ Task	80 ns	59.905 µs	271 ns	1.237 µs	134 ns	59.761 µs
mixer3	⚙ Task	249 ns	13.129 µs	1.493 µs	584 ns	493 ns	12.025 µs
mapper	⚙ Task	1.733 µs	302.188 µs	9.108 µs	12.158 µs	6.030 µs	296.612 µs
act_led	⍊ Sink	8 ns	9.319 µs	154 ns	213 ns	110 ns	9.152 µs
vtx_osd	⚙ Task	707 ns	122.282 µs	2.792 µs	3.871 µs	2.044 µs	120.489 µs
gnss_ublox	◈ Src	472 ns	17.969 µs	1.775 µs	794 ns	638 ns	16.705 µs
attitude	⚙ Task	163 ns	6.269 µs	665 ns	274 ns	237 ns	5.837 µs
navigation	⚙ Task	104 ns	9.277 µs	910 ns	448 ns	352 ns	8.811 µs
auto_mission	⚙ Task	9 ns	5.810 µs	559 ns	275 ns	257 ns	5.453 µs
autonomy_context	⚙ Task	99 ns	413.577 µs	532 ns	8.532 µs	480 ns	413.229 µs
auto_controller	⚙ Task	58 ns	10.272 µs	452 ns	307 ns	192 ns	9.884 µs
mode_supervisor	⚙ Task	59 ns	9.425 µs	329 ns	319 ns	129 ns	9.209 µs
bridge::rc	⇆ Brg	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
bridge::rc::rx::rc_rx	⇆ Brg	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
bridge::vtx_msp	⇆ Brg	0 ns	0 ns	0 ns	0 ns	0 ns	0 ns
bridge::vtx_msp::rx::incoming	⇆ Brg	9 ns	15.691 µs	97 ns	330 ns	87 ns	15.644 µs
bridge::vtx_msp::tx::requests	⇆ Brg	20 ns	1.298 µs	312 ns	128 ns	129 ns	1.253 µs


depth & prediction (CPU)
LA +30.389930 LO -097.693265
004°
039.0 KM/H
ALT 0008.1 M
/_/\
(0 0)
> ^ <
[CU29]
AUTO
3.89V
View FPV (V -> 3RD)
RC Link Not connected (plug RC via USB/BT)
Arm T
Mode 1=Acro 2=Angle 3=PosHold
Auto 4=Auto toggle
Throttle Space (arm Angle / climb)
Roll/Pitch WASD
Yaw Q / E
Reset R

ROS



➪ Models the DAG 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 just mandatory to run this.
- ➔ very challenging to put in production as a company
- ➔ Safety certifying a system using this? 🤷



So how did we fixed this?





Copper-rs





Rust first SDK & Runtime for robotics.

⭐🙏🥹

An Open Source engine for robotics.



- Software “engine” for Robots?



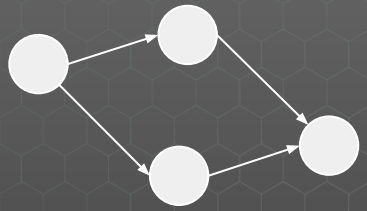




- Goal: Focus on the end product, not just a demo.



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**.



Automated execution strategy **tuning**



Collect data

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

Deploy and test.



ROS runtime vs Copper Runtime





Copper



- **Deterministic** *by design*
- **Design for safety** - conceptually Copper is a compiler
- **Performant**
  - High bandwidth
  - Low latency
  - Low jitter
- Allows developers to **focus on algorithms**, not buffers
- **Tunes the system** as you iterate on it
- **Runs on everything**, Linux, Baremetal, Mac, Windows, WASM…



Rethinking Robotics, One First at a Time.

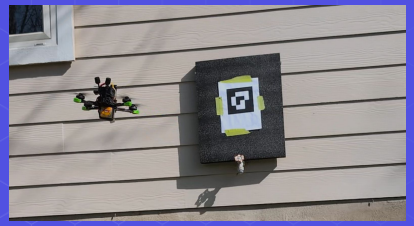
- v1.3 Profile Guided Scheduling (PGS)
- v1.0 Time Travel Debugging API
- v1.0 Distributed Replay & Swarm Support
- v1.2 Live Twinning Telemetry Over Lossy Links
- v1.1 Anytime Computation (w/ Global Sched)
- v1.0 Safety Monitoring
- v1.0 Typed Lossless Log Codecs
- v1.3 First Class Clock Sync With Replay



Running **today**





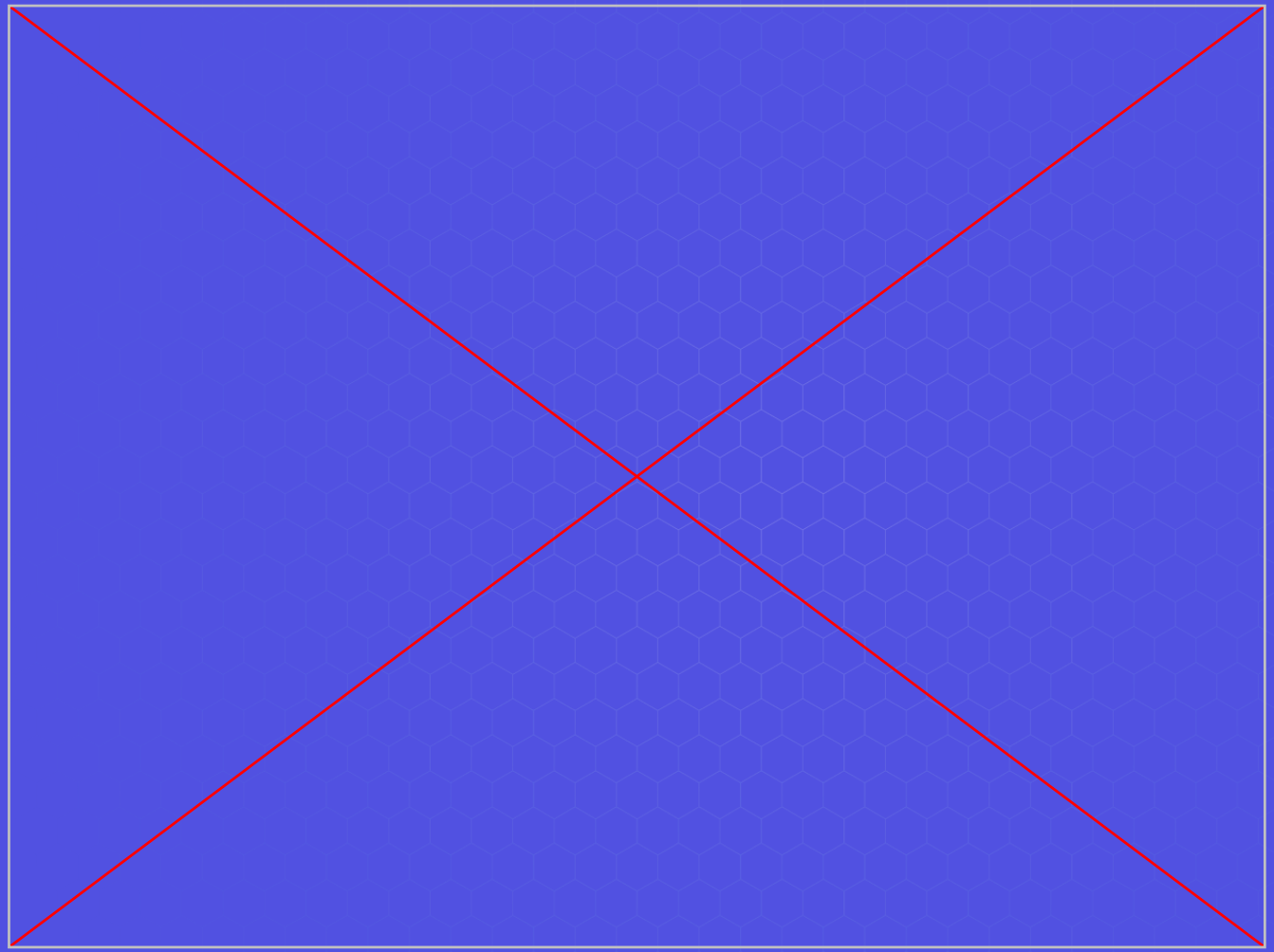






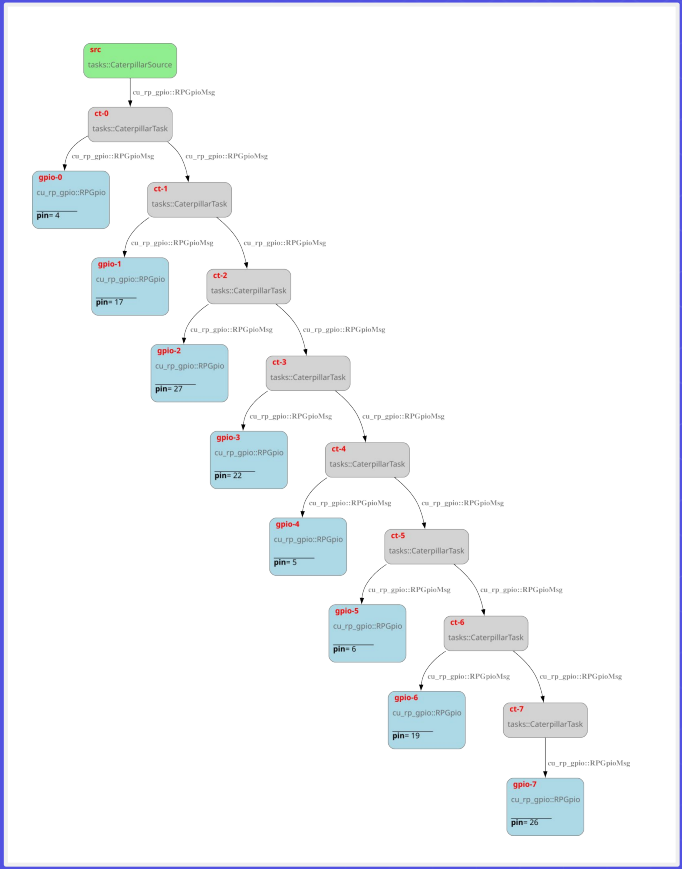


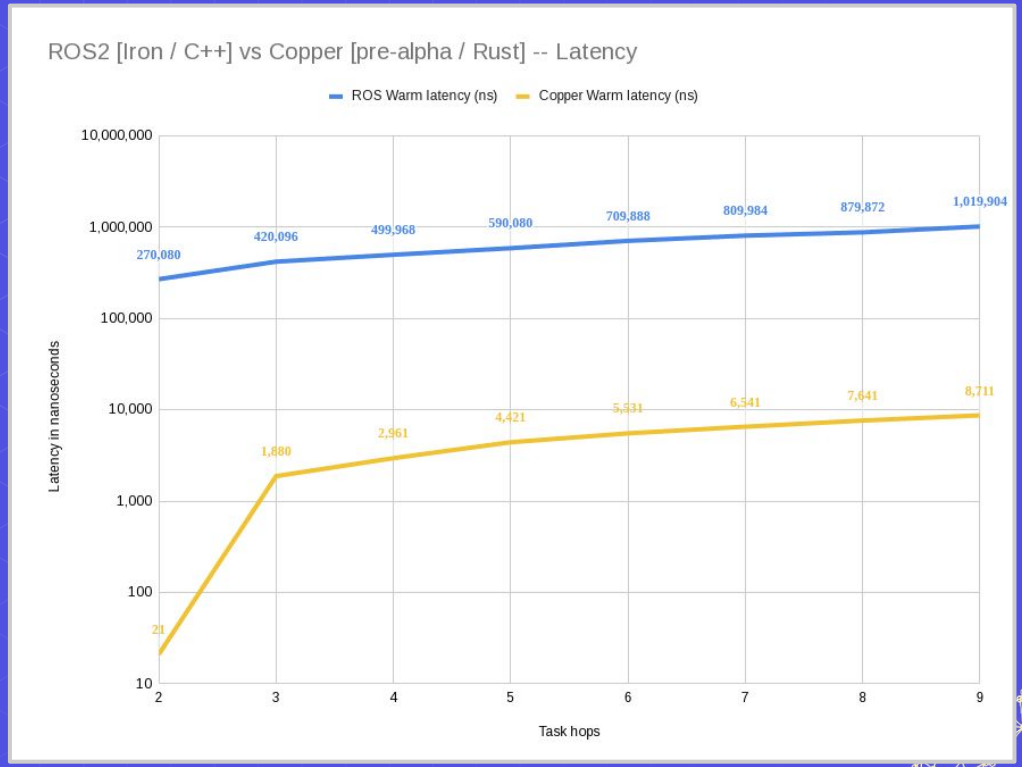






100x less latent at runtime than ROS2





Linear data logging

31000x faster OOTB than ROS2 (sqlite backend)

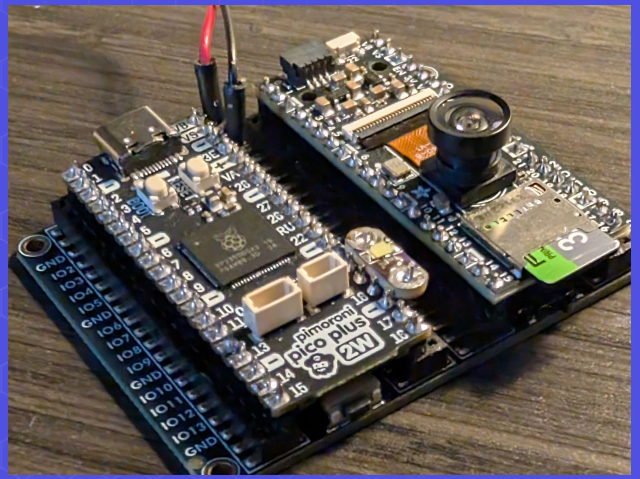
But MCAP?

Copper is still 12x faster ROS2 logging with MCAP



We run on everything . . . including baremetal

	Linux	x86_64 armv7 aarch64 riscv64
	macOS	arm64
	Windows	x86_64
	Android	arm64



RP2350 (32-bit Arm Cortex-M)

But also STM32H, N6 . . .

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);
```



Defining the Graph

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 familiar 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
```



Copper is a bridge for the emergent Rust Robotics Ecosystem

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



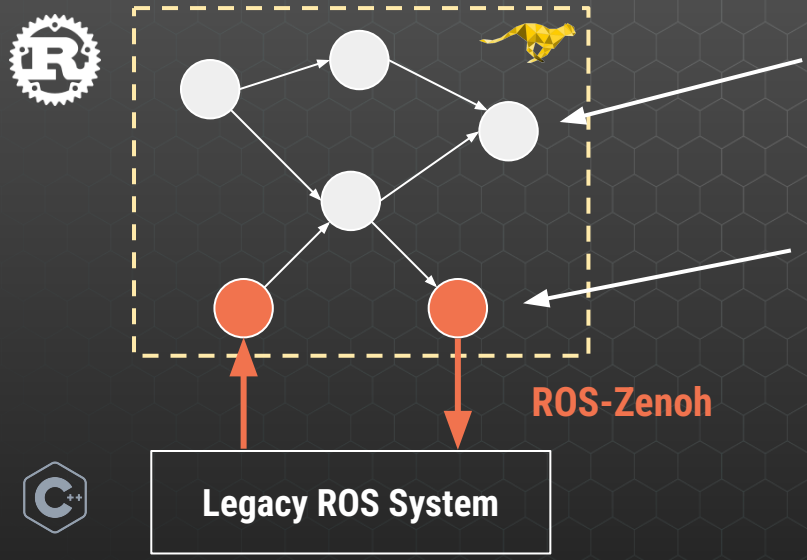


Ask for a sticker of
our mascot
Cunick Sense.

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

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

Some robotics companies using Rust:

- Scythe
- Bedrock robotics
- Komboi
- farm-ng / bonsai
- Yaak
- and many more…

