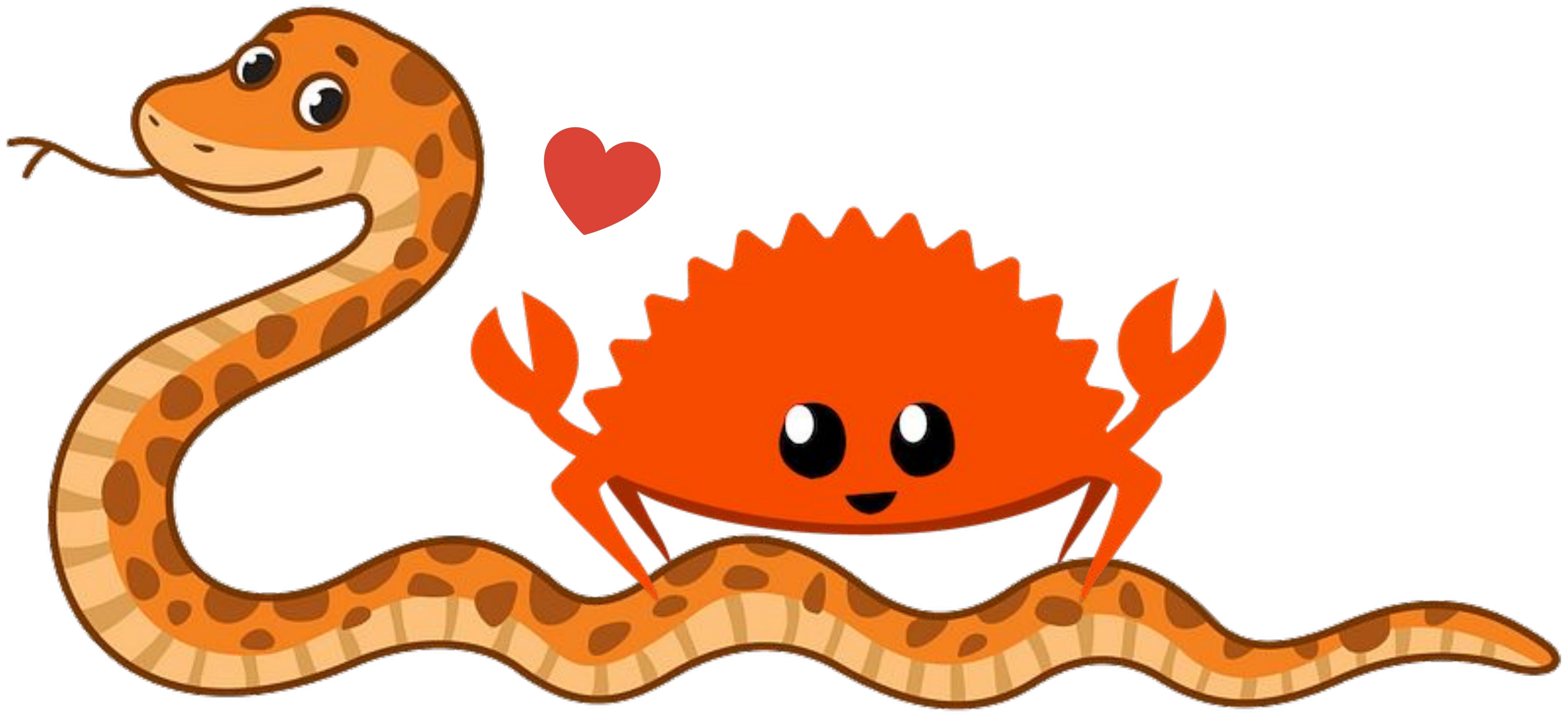


Would Rust Make You A Better Pythonista?

EuroPython 2023

Bonjour !

Alexys Jacob

CTO at
numberly

Open Source community service

@ultrabug

- Open Source author & contributor
- Author of py3status, uhashring, mkdocs-static-i18n
- MkDocs contributor & maintainer
- PSF contributing member
- 10th EuroPython!

Gentoo Linux developer

- Gentoo Docker containers

Tech speaker & writer

- Tech conferences, webinars, blog posts
- https://ultrabug.fr

Foreword

NOT a Rust **vs** Python talk

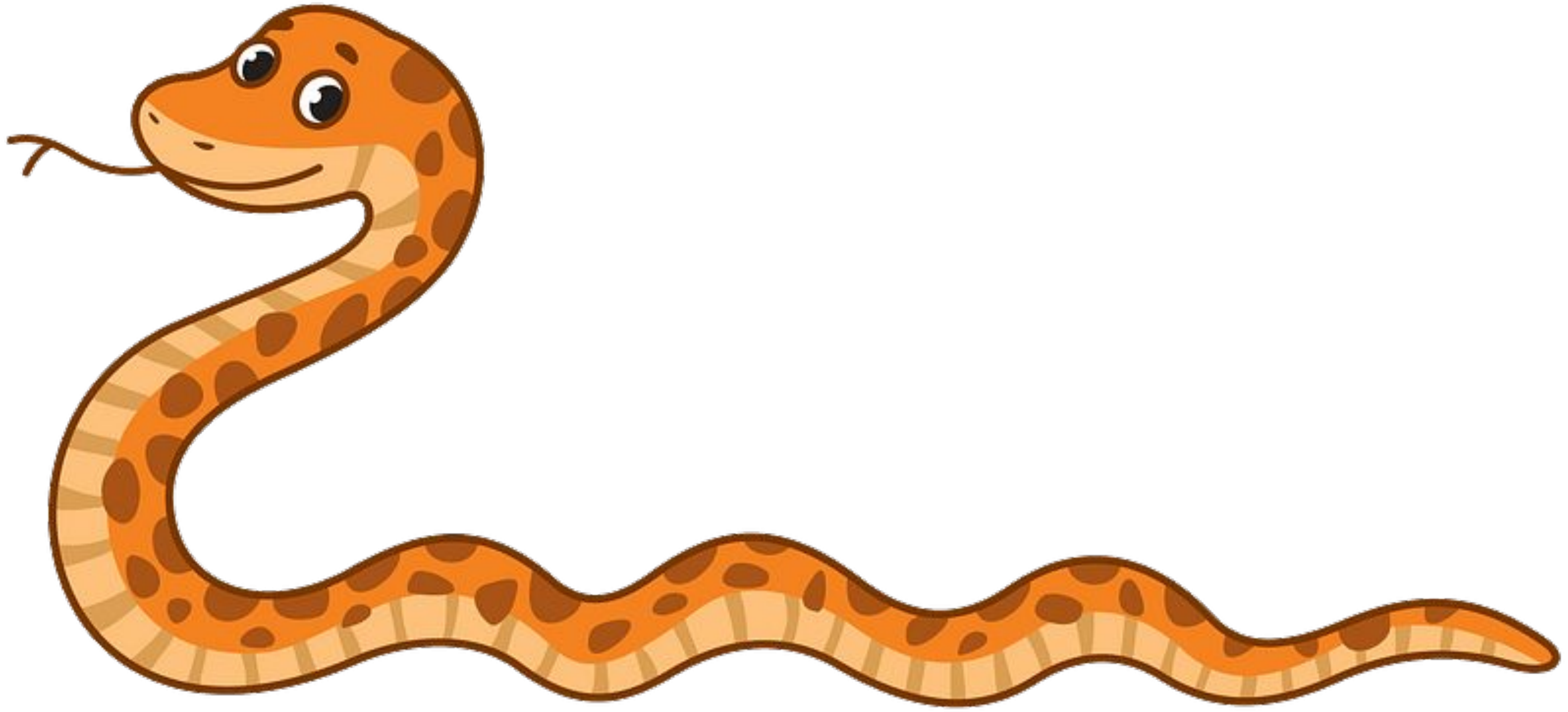
NOT a **comparison** between Python and Rust

NOT trying to convince you to use Rust **instead of** Python

SHARE my **rationale** in adopting Rust

SHARE my **experience** in getting Rust in production

SHARE some **thoughts** and **perspectives**

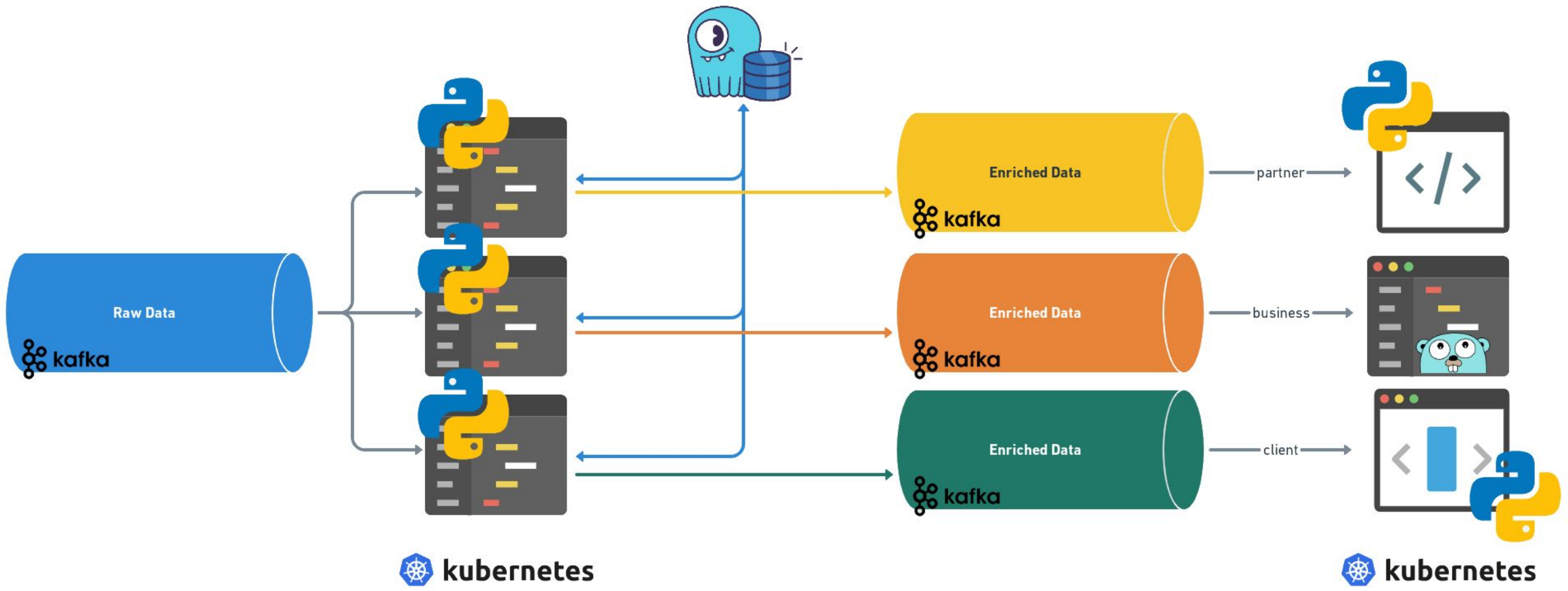


Python powered data pipelines

Business Context

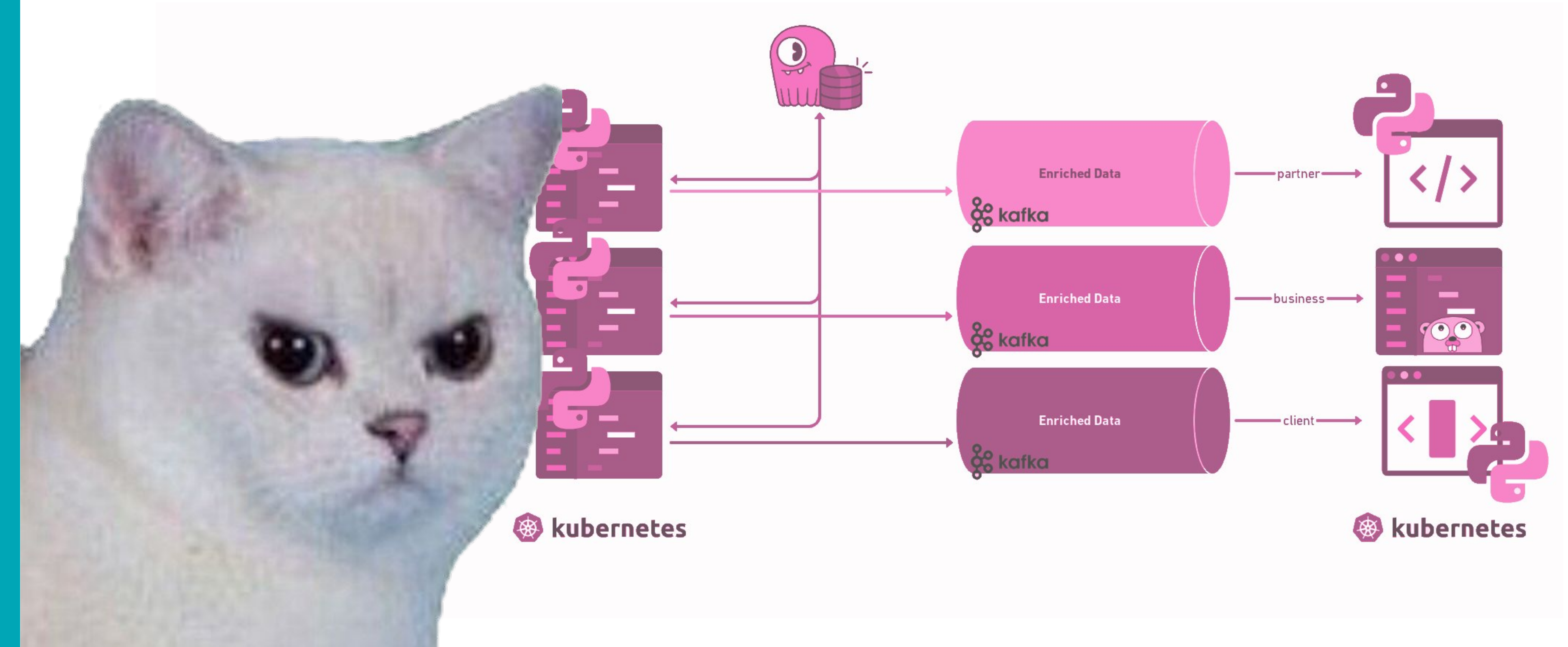
Project context at Numberly

At **Numberly**, we use **Python** **data processor** applications to **wrangle and move** (a lot of) data using **Kafka streams and pipelines** that are enriched using **ScyllaDB**.



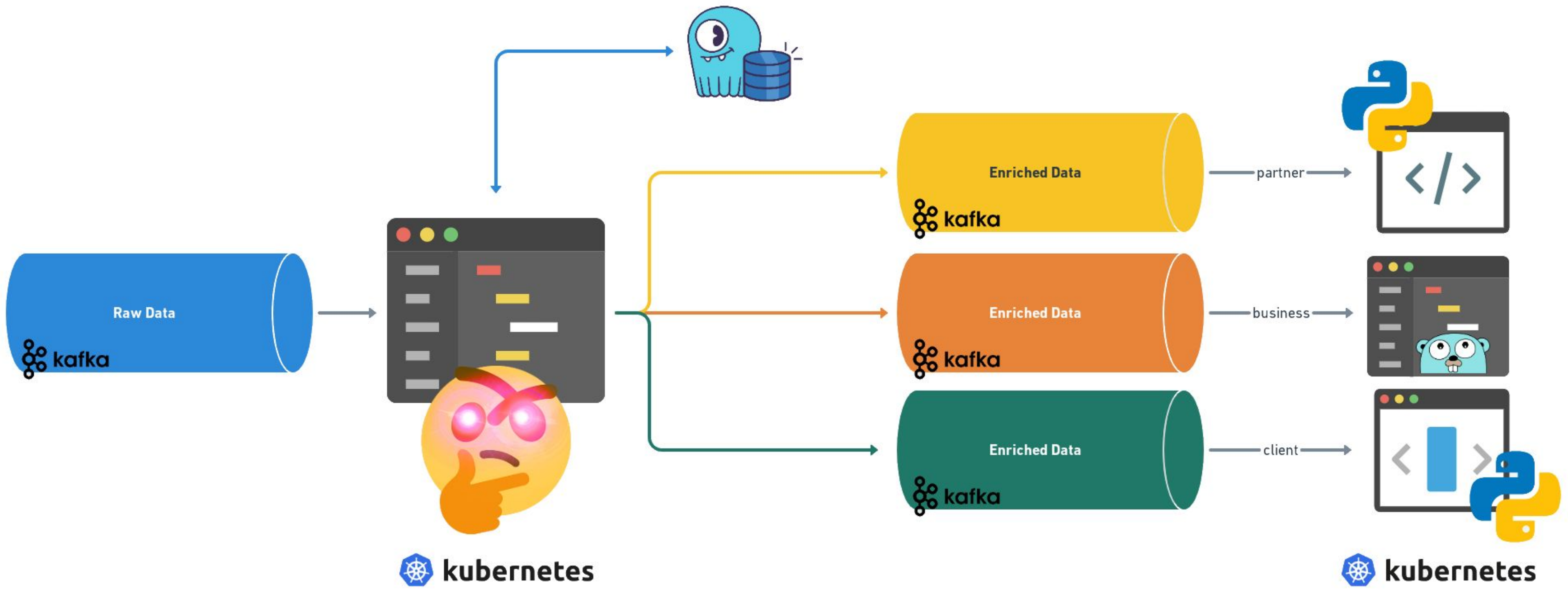
Pipeline reliability = latency + resilience

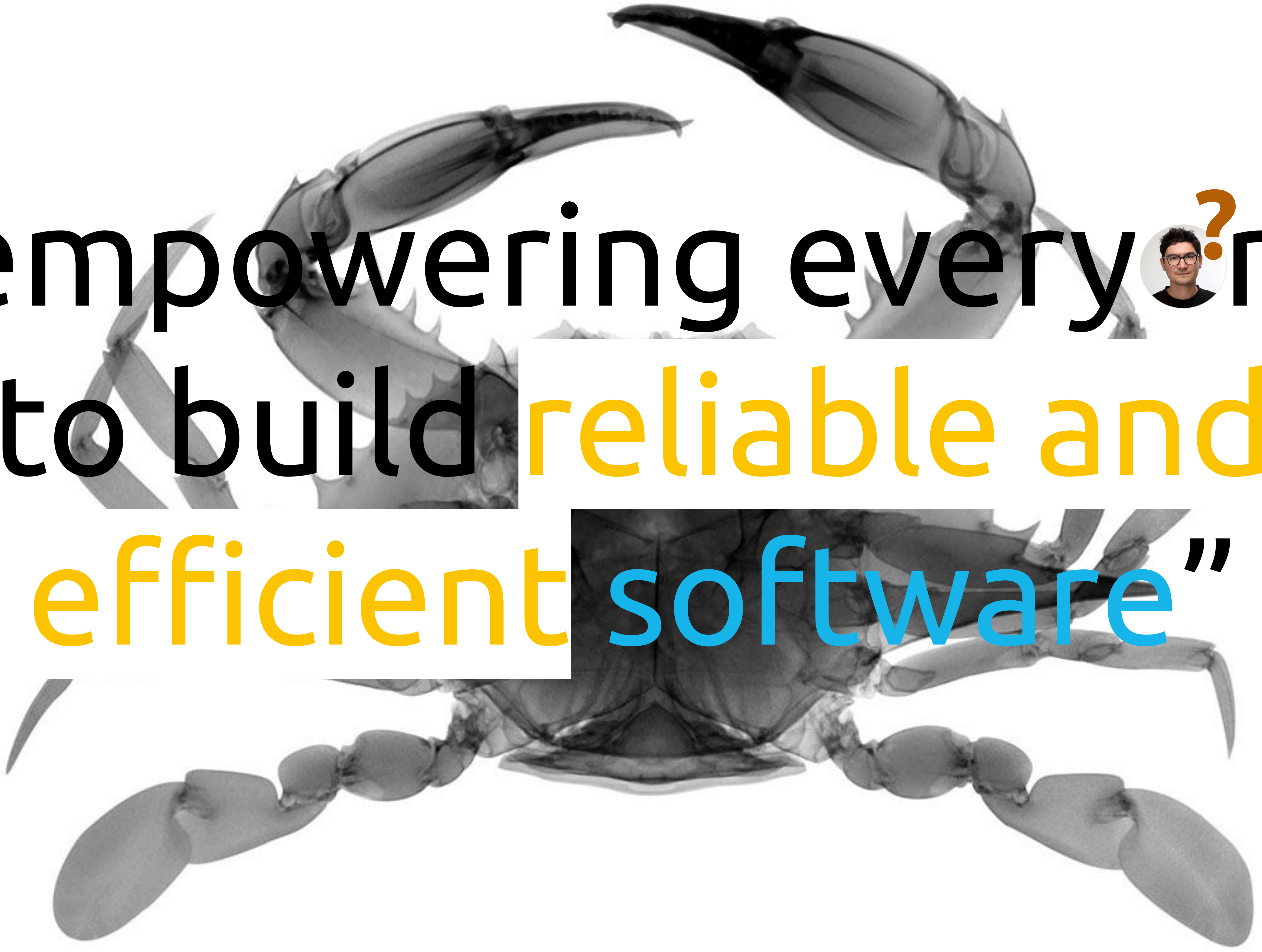
If a Python processor application is **slow or fails**, our business and partners are at risk…
… and **we get angry clients**.



Rewriting 3 Python apps into ONE

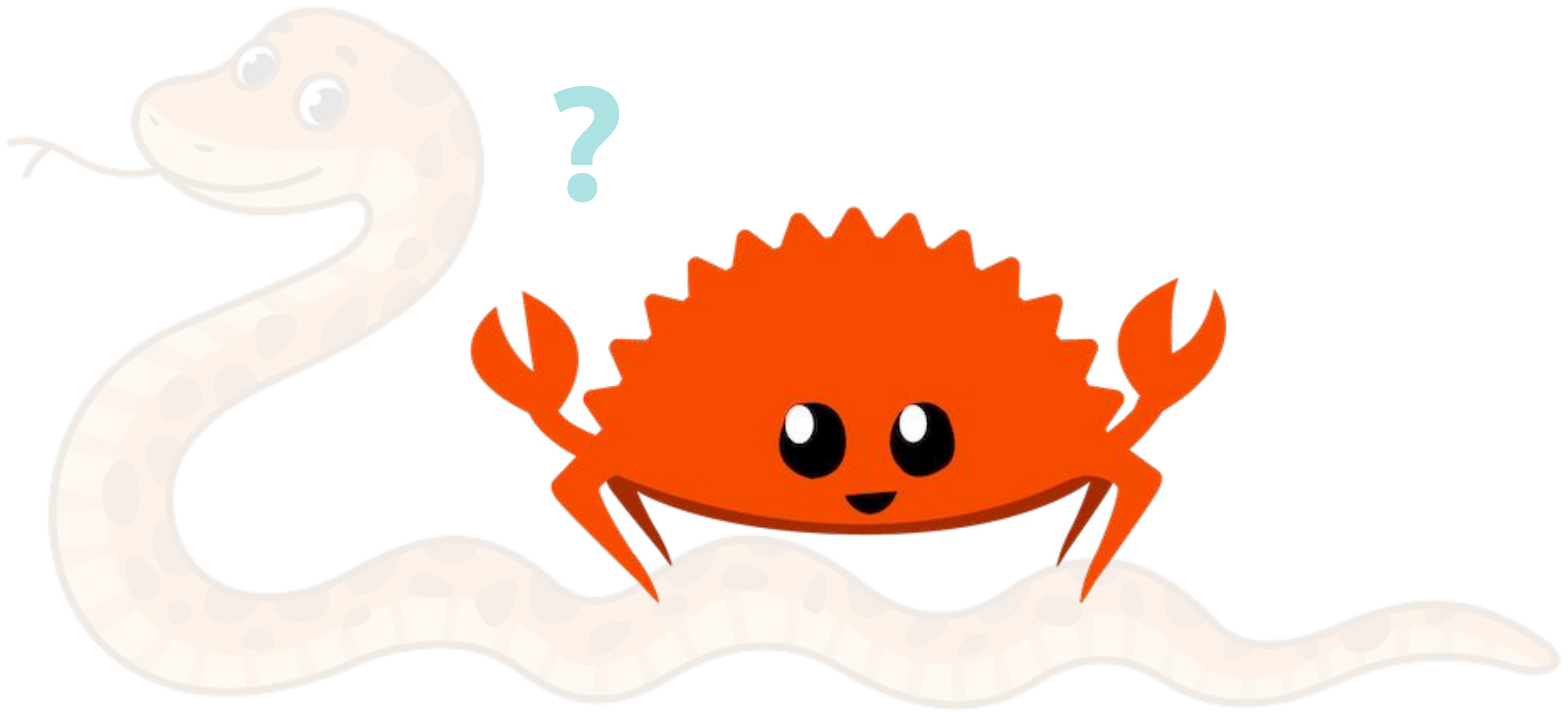
A major **change in our pipeline processing logic** had to be undertaken, giving us the **opportunity to redesign and merge them into one**.





“empowering everyone to build reliable and efficient software”

“Hey, why not rewrite those 3 Python processor apps into 1 Rust app?”



Engaging with Rust instead of Python

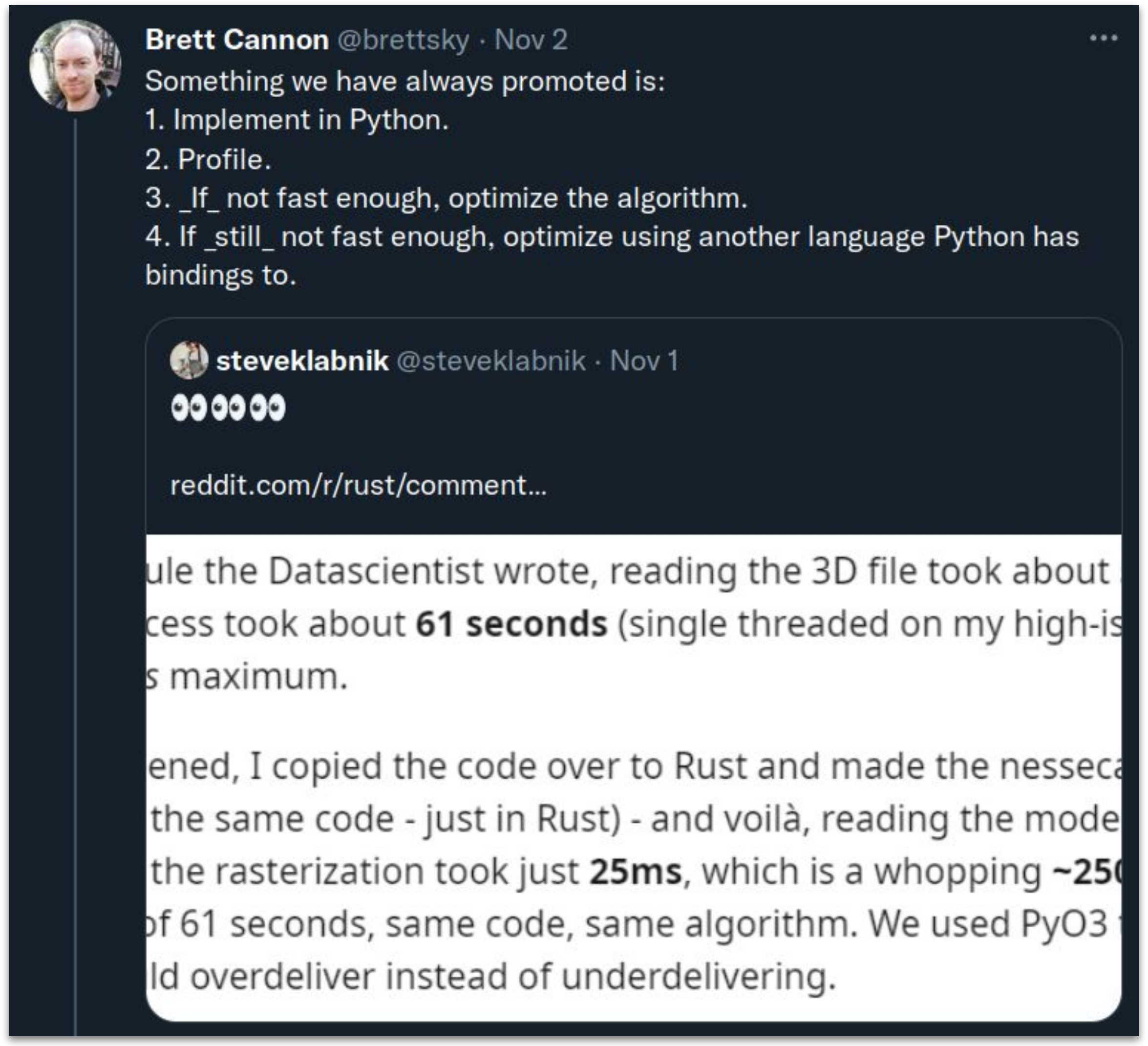
Motivation & Rationale

"a general purpose programming language"

Python was **Fast** enough!

I did NOT choose Rust to be Faster.

“ Selecting a programming language can be a form of premature optimization
Brett Cannon, Python Core Dev


Brett Cannon @brettsky · Nov 2
Something we have always promoted is:
1. Implement in Python.
2. Profile.
3. _If_ not fast enough, optimize the algorithm.
4. If _still_ not fast enough, optimize using another language Python has bindings to.

steveklabnik @steveklabnik · Nov 1
👀👀👀

reddit.com/r/rust/comment...

ule the Datascientist wrote, reading the 3D file took about
cess took about **61 seconds** (single threaded on my high-is
s maximum.

ened, I copied the code over to Rust and made the nesseca
the same code - just in Rust) - and voilà, reading the mode
the rasterization took just **25ms**, which is a whopping **~25**
of 61 seconds, same code, same algorithm. We used PyO3
ld overdeliver instead of underdelivering.

Efficient software != Faster software

"Fast" meanings vary depending on your objectives and experience.

- Fast to **develop**?
  - Python is way faster, been using for 15+ years
- Fast to **prototype**?
  - No, code must be complete to compile and run
- Fast to **process data**?
  - Maybe, but prove it
- Fast to **cover** all **failure** cases?
  - Definitely: mandatory exhaustivity + error handling primitives
- Fast to **maintain**?
  - Nobody at Numberly did Rust yet

Innovation cannot exist
if you don’t accept to lose time.

The Reliable software paradigms

What makes me slow can make me stronger!

- Low level **paradigms** (ownership, borrowing, lifetimes)
  - If it compiles, it's safe
- Strong **type safety**
  - Predictable, readable, maintainable
- **Compiler** is a friend
  - Compiler is very helpful vs a random Python exception
- **Dependency** management
  - Finally something looking sane vs Python mess
- **Exhaustive** pattern matching
  - Confidence that you're not forgetting something
- **Error** management **primitives** (Result)
  - Handle failure right from the language syntax

“

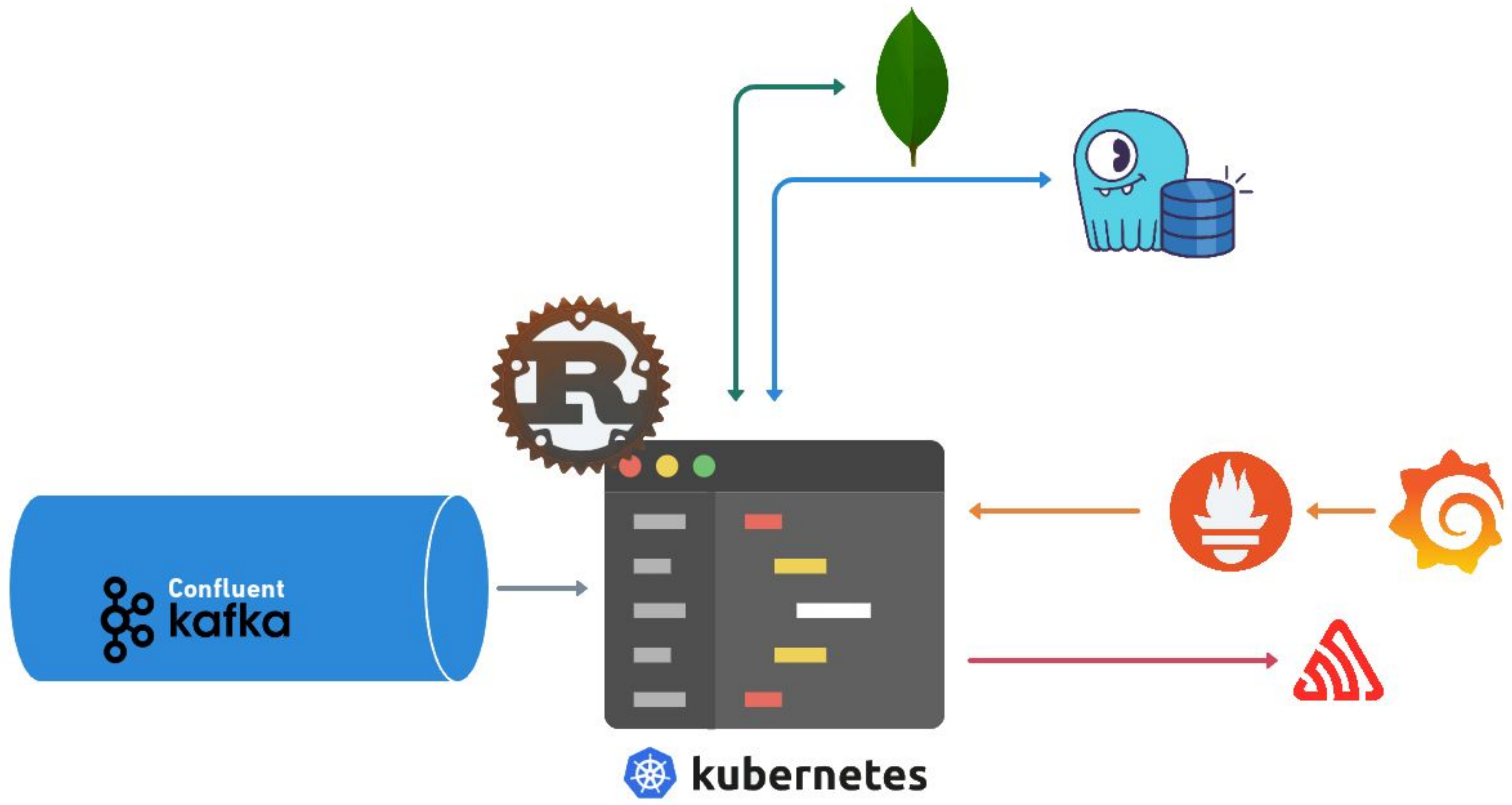
I chose Rust because it provided me with the **paradigms and the abstraction level** that I needed to understand and better **explain the reliability and performance** of my application

Tails of a Pythonista learning Rust

Learning Rust the Hard Way

Production is not a Hello World

Production requires to design for scale, high availability and observability.



Confluent Kafka Schema Registry

Status: BROKEN
Confluent Schema Registry breaks vanilla Apache Avro deserialization.

Fix: use the **schema_registry_converter** crate, by Gerard Klijs.

Manual approach:

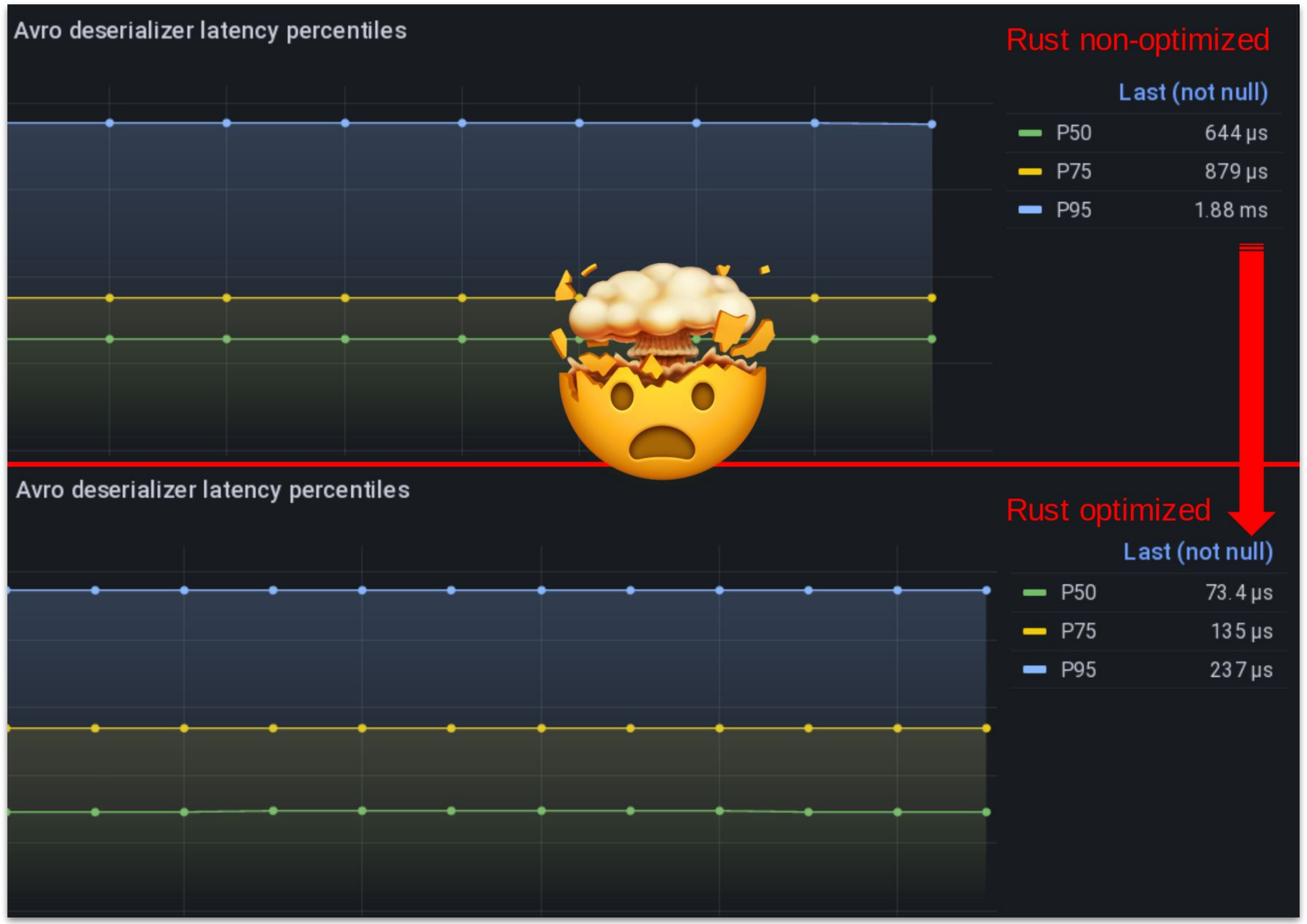
```rust
/// Deserialize the given kafka raw `message` using the provided
/// Avro `schema` and return a Navigation struct message to be
/// used by the processors.
pub fn get_decoded_message(schema: &Schema, message: &BorrowedMessage) -> Result<Navigation> {
    let mut reader = Cursor::new(&message.payload().unwrap()[5..]);
    let val = match from_avro_datum(&schema, &mut reader, None) {
        Ok(inner) => inner,
        Err(err) => {
            return Err(anyhow!(err));
        }
    };
    let navigation: Navigation = match &val {
        Value::Record(_) => from_value::<Navigation>(&val).unwrap(),
        _ => {
            return Err(anyhow!("could not map avro data to struct"));
        }
    };
    Ok(navigation)
}
```

Apache Avro Rust

Status: BROKEN
Deserialization of complex schemas was broken, no appointed Apache Avro maintainer at the time.

Fix: I contributed the fixes on GitHub for AVRO-3232 + AVRO-3240, now merged by Martin Grigorov!

Learnings: impressive Rust compiler **optimizations** + Avro deserialization is faster than JSON!



Rust non-optimized

	Last (not null)
🟩 P50	644 µs
🟨 P75	879 µs
🟦 P95	1.88 ms

Rust optimized

	Last (not null)
🟩 P50	73.4 µs
🟨 P75	135 µs
🟦 P95	237 µs

Exporting metrics properly for Prometheus

Status: GREAT
Fine tune your histogram buckets to match your expected latencies!

```
pub static ref SCYLLA_INSERT_QUERIES_LATENCY_HIST_SEC: Histogram = register_histogram!(
        "scylla_insert_queries_latency_seconds",
        "Scylla INSERT query latency histogram in seconds",
        vec![0.0005, 0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1.0, 5.0, 15.0],
    )
    .expect("failed to create prometheus metric");
```

Effectively measuring latencies down to microseconds:

```
let timer = SCYLLA_INSERT_QUERIES_LATENCY_HIST_SEC.start_timer();
match scylla_session
    .execute(
        statement,
        &(&uid, &message_seen_datetime, &scylla_ttl, &scylla_timestamp),
    )
    .await
{
    Ok(_) => {
        timer.observe_duration();
        Ok(())
    }
    Err(err) => {
        drop(timer);
        PROCESSING_ERRORS_TOTAL.with_label_values(&["scylla_insert"]).inc();
        error!("insert_in_scylla: {:?}", err);
        Err(anyhow!(err))
    }
}
```

Observing your code metrics using Grafana

Status: MUST HAVE

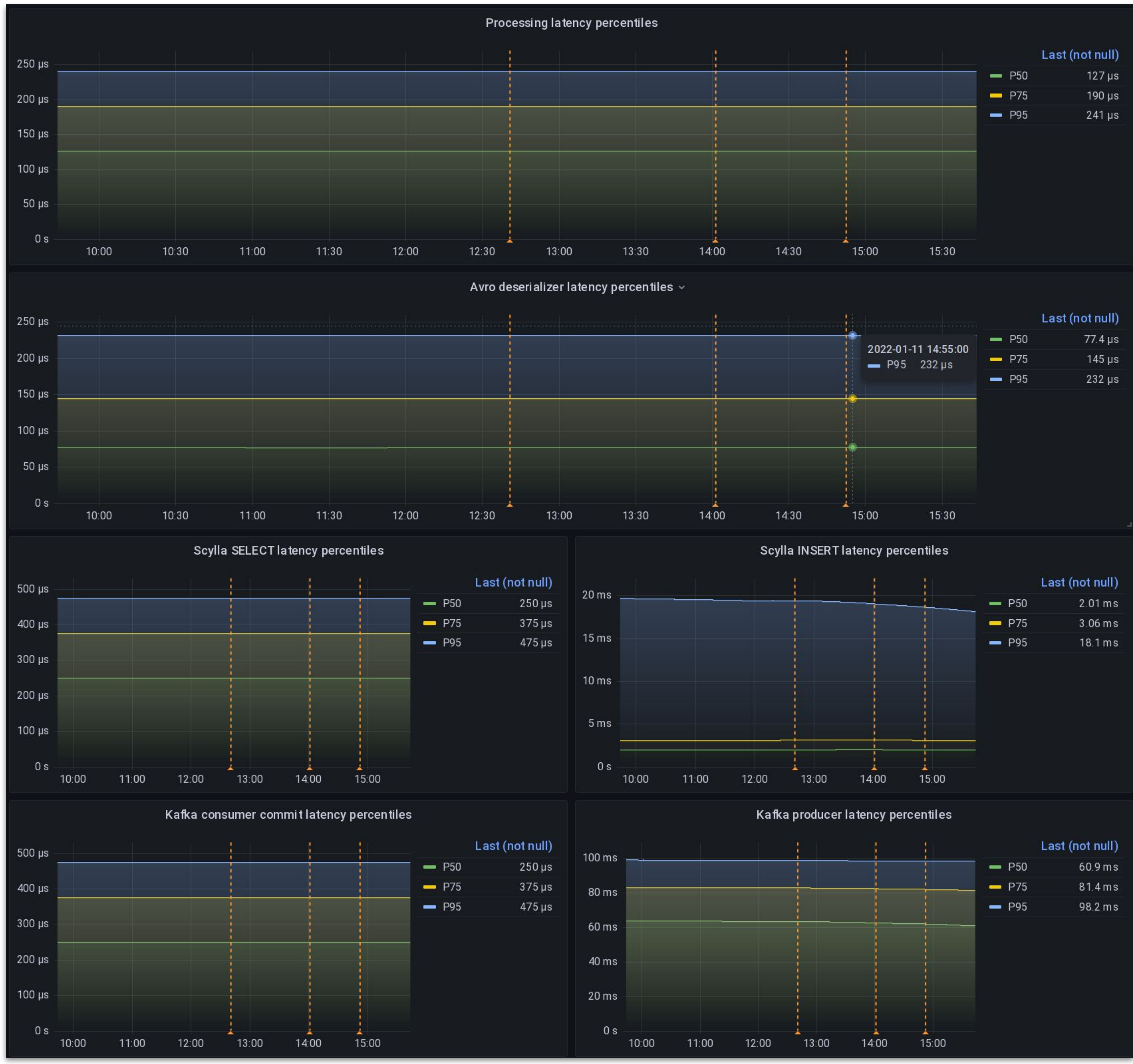
Graph all the things with metrics, but do it right!

- Query and throughput rates
- Kafka commits occurrence
- Errors by type
- Kubernetes pod memory
- ...

Visualizing Prom Histograms

https://grafana.com/blog/2020/06/23/how-to-visualize-prometheus-histograms-in-grafana/

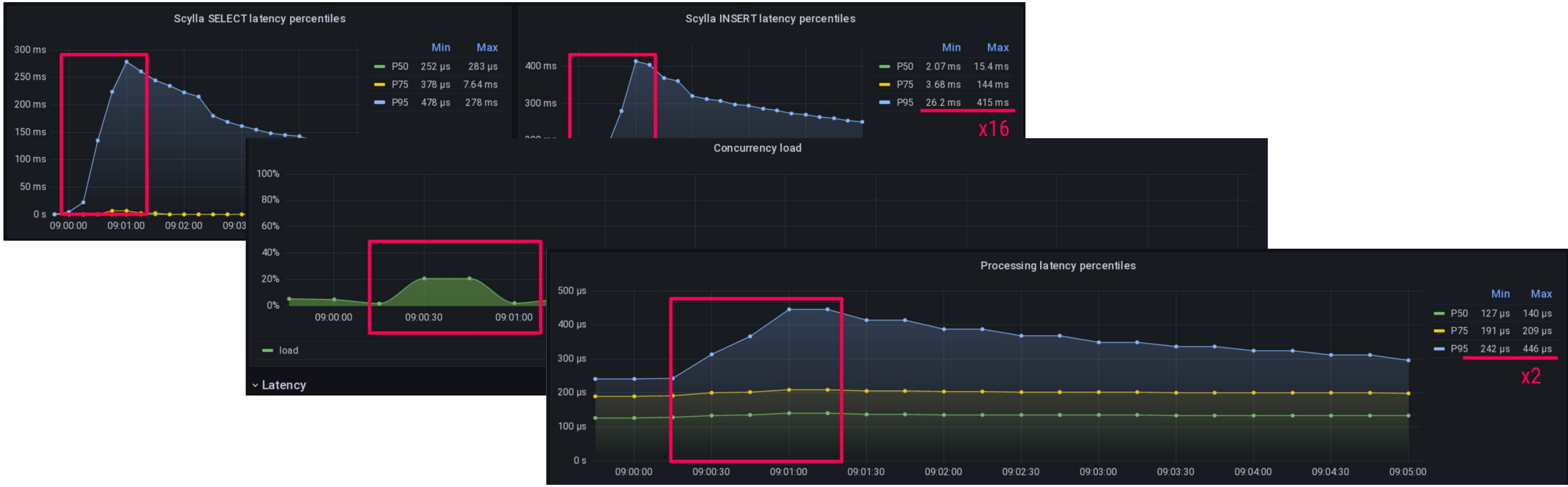
```
max by (environment)(histogram_quantile(0.50, processing_latency_seconds_bucket{...}))
```



Absorbing tail latency spikes with parallelism

Status: GREAT
Find asynchronous processing patterns to optimize latency by controlling your green-threads parallelism!

Learning: keep your CPU-bound operations in the main loop!



Scylla SELECT latency percentiles

	Min	Max
P50	252 µs	283 µs
P75	378 µs	7.64 ms
P95	478 µs	278 ms

Scylla INSERT latency percentiles

	Min	Max
P50	2.07 ms	15.4 ms
P75	3.68 ms	144 ms
P95	26.2 ms	415 ms

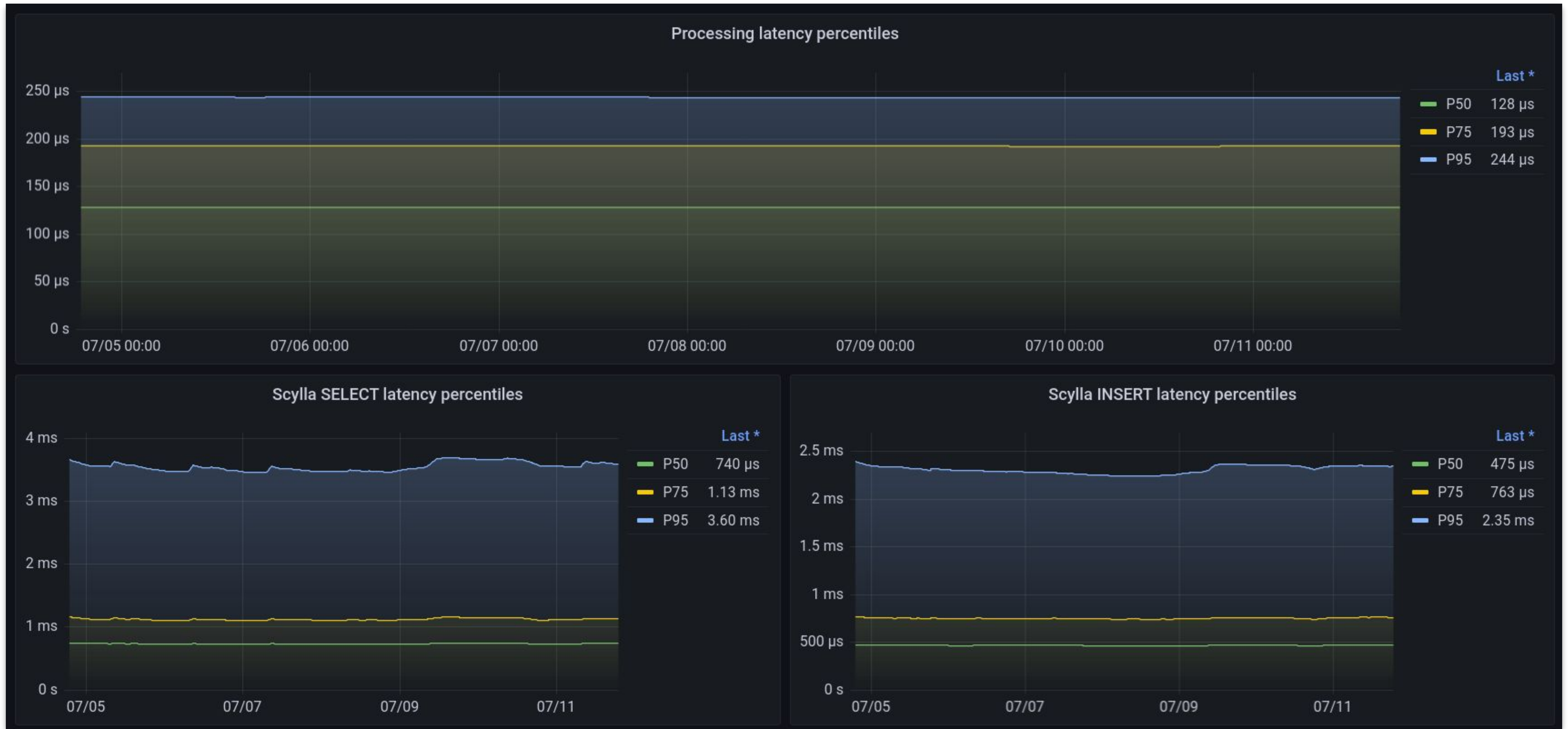
Processing latency percentiles

	Min	Max
P50	127 µs	140 µs
P75	191 µs	209 µs
P95	242 µs	446 µs

Production numbers & graphs

Key figures for this application:

- **Kafka** consumer max **throughput** (including data processing)? 200K msg/s on 20 partitions
- **Avro deserialization** P50 latency? ~100µs
- **ScyllaDB SELECT** P50 latency on 2B+ rows tables? 740µs
- **ScyllaDB INSERT** P50 latency on 2B+ rows tables? 475µs



How Rust influences my daily Python?

Thoughts and perspectives

Rust is appealing to Pythonistas

Not an entrance programming language

Less intimidating than C/C++

A syntax **surprisingly easy to read and learn**

Great **level of abstraction easing adoption**

Python **plays** very well with Rust!

Easy to Adopt **!=** Accessible

Expect Rust to be difficult

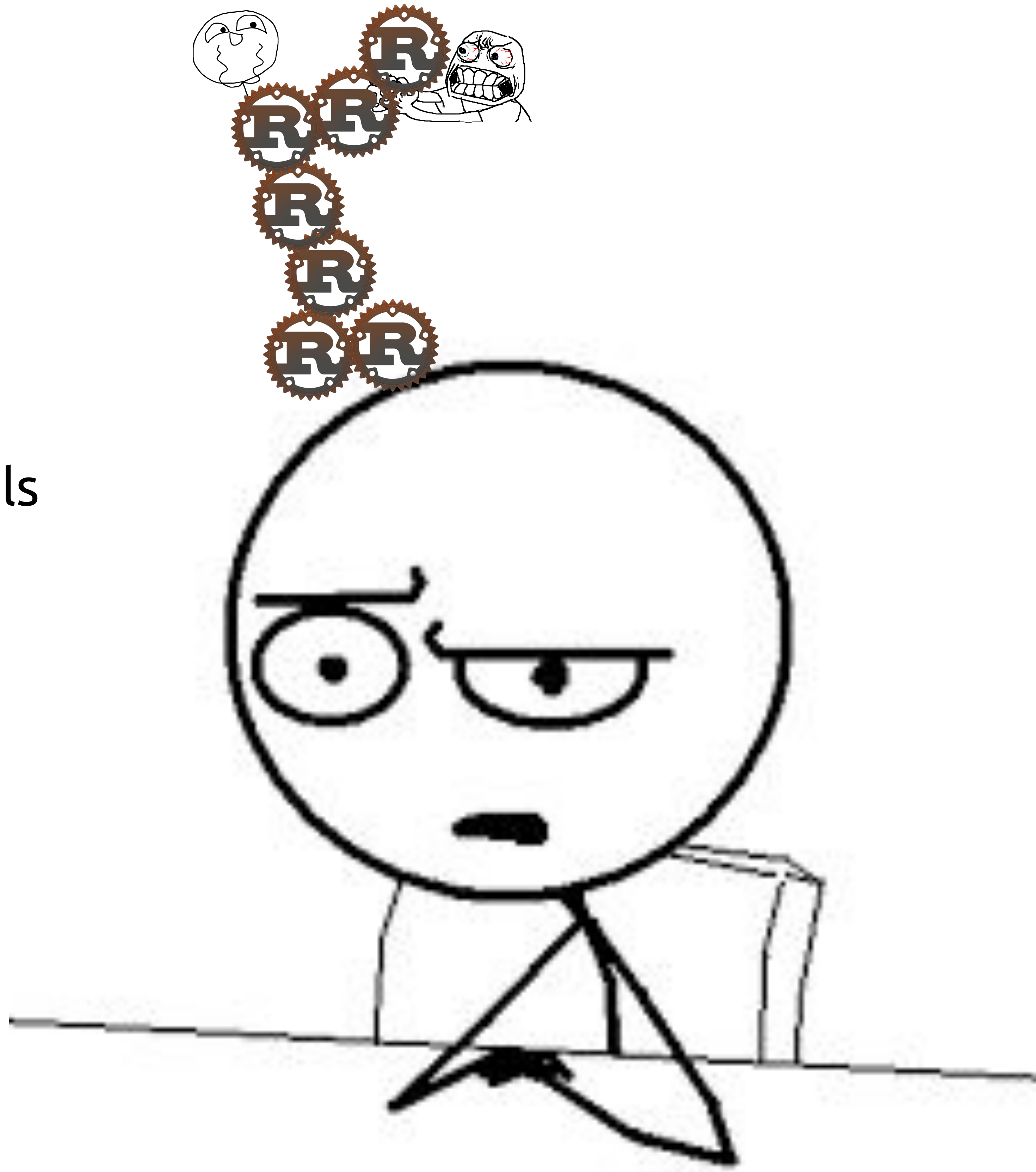
Python is **accessible** and comes with “batteries included”

Rust sets **higher expectations on your design decisions**

Rust **bureaucracy forces you to learn and care** about the details

You will be **slower** coding Rust than Python!

Slower + Harder = **Bargain**



You get more than brackets and semicolons

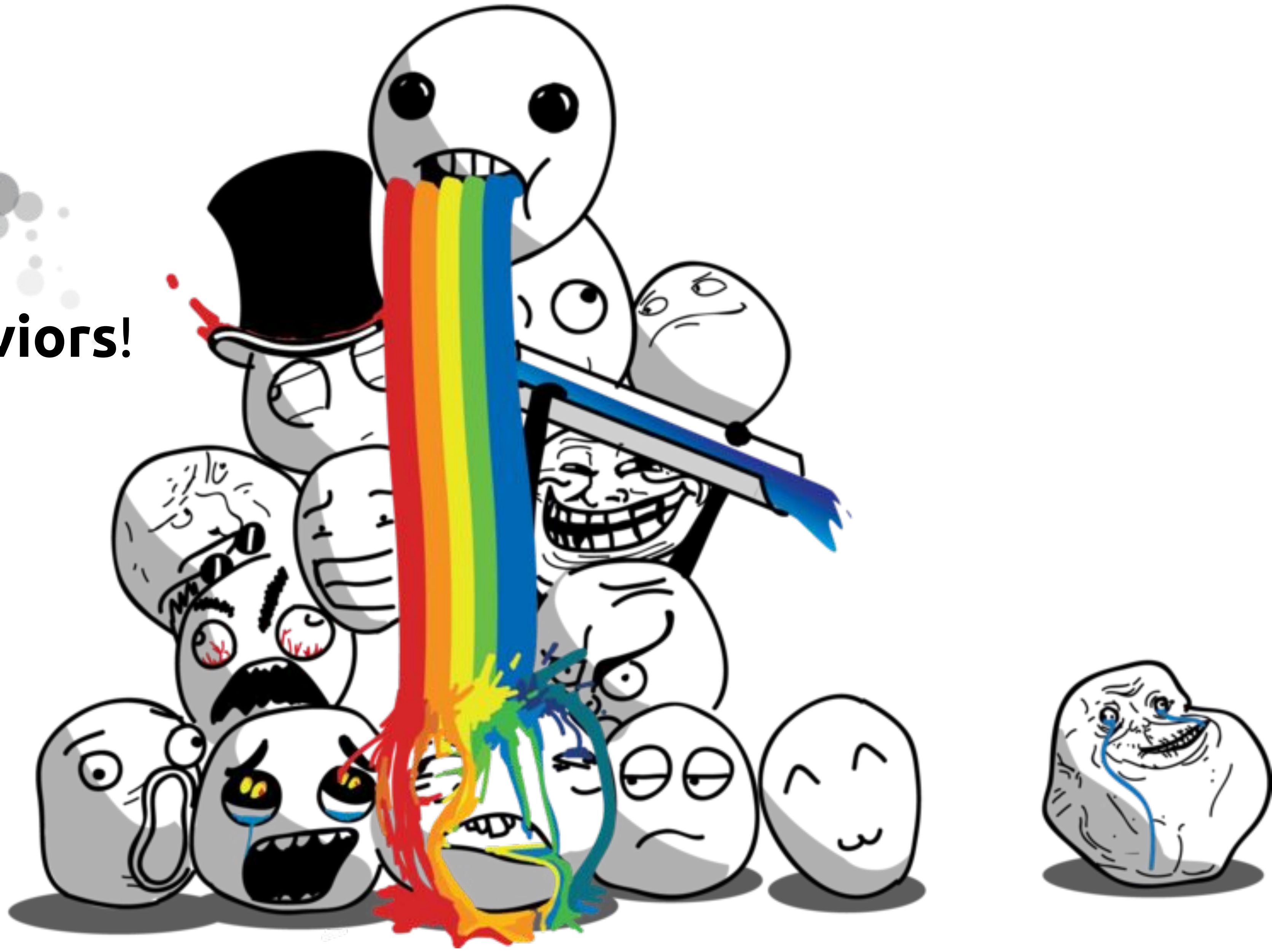
Rust **paradigms** provide a **strong** feeling of code **reliability**

The Result / Option syntax is mind blowing!

Rust bureaucracy is a **security against unexpected behaviors**!

The Rust **compiler helps a lot** and makes your life easier

Safe development = **Confidence**



Reflect on the “Fast” meanings

Rust **is no silver bullet**

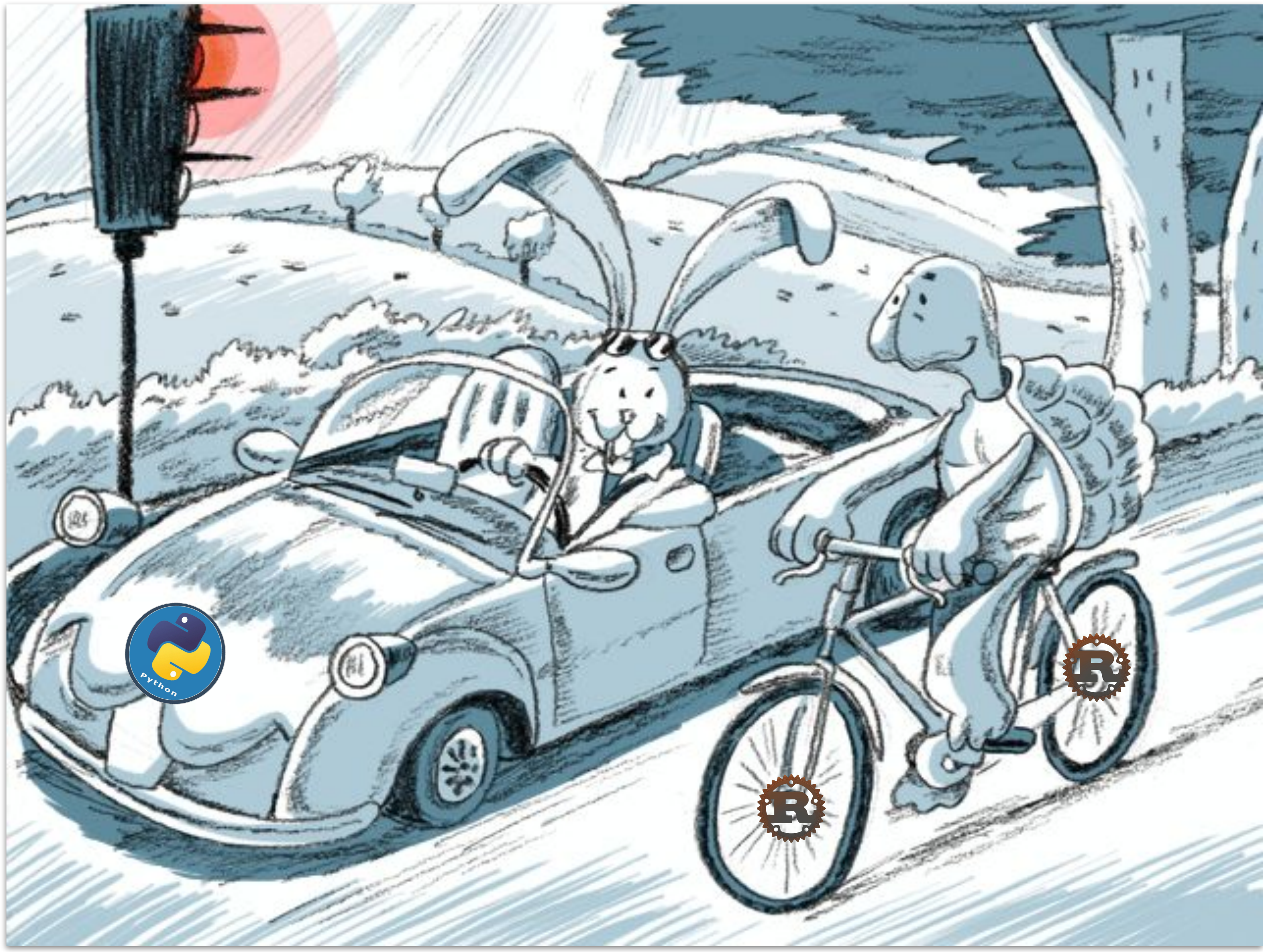
Python will materialize your ideas **faster**

Rust type checking != Python type hinting

Apply the Rust **bargain** to your use cases!

Rust is a tool to tackle challenges **differently**

Efficient **!=** Fast



Community, Diversity & Inclusion

Not so random thoughts

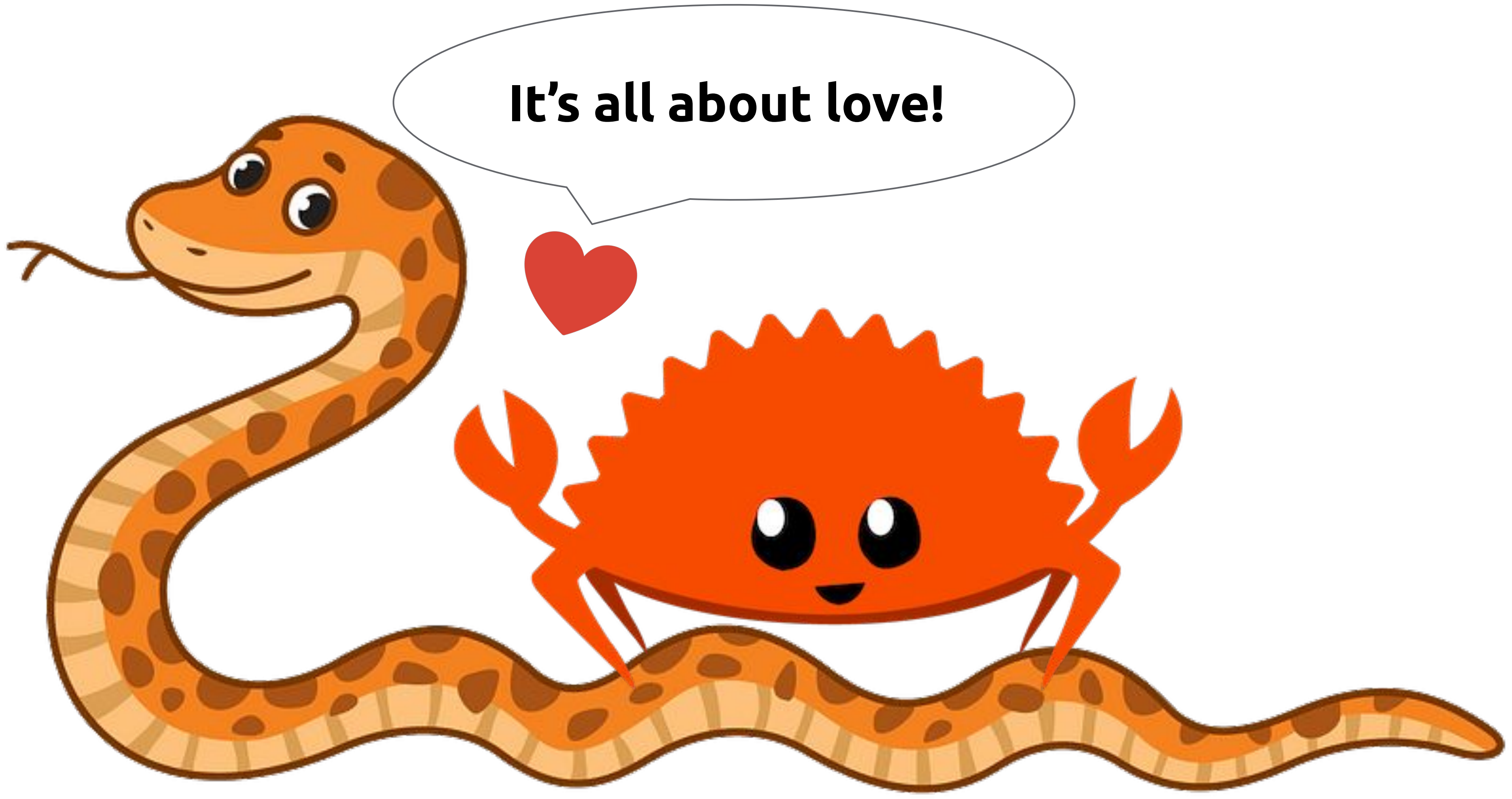
Assimilating Rust **paradigms** makes me feel like a better Pythonista!

You don't have to use every Rust feature!

Rust **does not fit all -** Python does and **is intended to**!

I wish Python's **tooling experience** takes inspiration from Rust

- *cargo new <project_name>*	*pykg new <project_name>*
- *cargo add <package>*	*pykg add <package>*
- *cargo fmt*	*pykg fmt*
- *cargo clippy*	*pykg clippy*



Keep on being inclusive Pythonistas

Thank you!

ultrabug alexys@numberly.com