OSA CON 25

Micromegas
Unified observability for video games

Marc-Antoine Desroches
Technical Architect
madesroches@gmail.com

November 4-5, 2025

Disclaimer

These are my personal opinions and experiences.
I'm not speaking for my employer.

The Objective

Quantify how often and how bad issues are,
with enough context to fix them

no need to reproduce.

The Problem

Traditional tooling force a choice

🔍 High-frequency debugging tools
Great detail, but you need to reproduce the bug yourself

📊 Low-frequency analytics tools
Will report statistics, not detailed traces

We refuse to choose.

The Challenge

- Video games at 60fps generate enormous telemetry volumes:
- ~10s of log entries/sec
- ~1k measures/sec - 10s/frame
- ~60k-200k CPU trace events/sec - 1000s/frame
- Thousands of concurrent processes
- Unified system for logs, metrics, and traces

Operating Costs - real production example

Data volume

- Retention: 90 days
- 9 billion log entries
- 275 billion metrics
- 165 billion sampled trace events

- Total 449 billion events
- Spikes of 266M events/minute

Monthly Cost breakdown

- Compute ~$300
- PostgreSQL ~$200
- S3 Storage: ~$500

Total ~$1,000/month

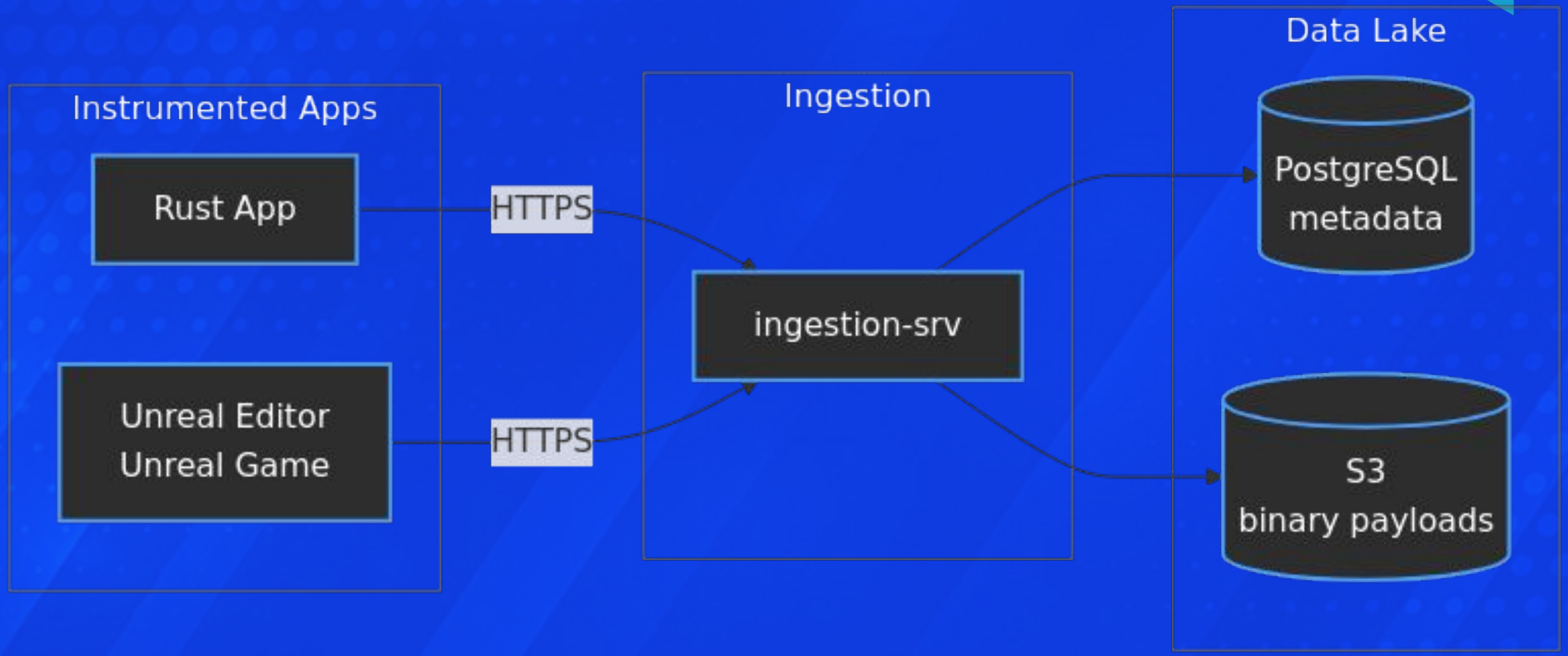
How

Recording & querying high volumes of data can be cheap

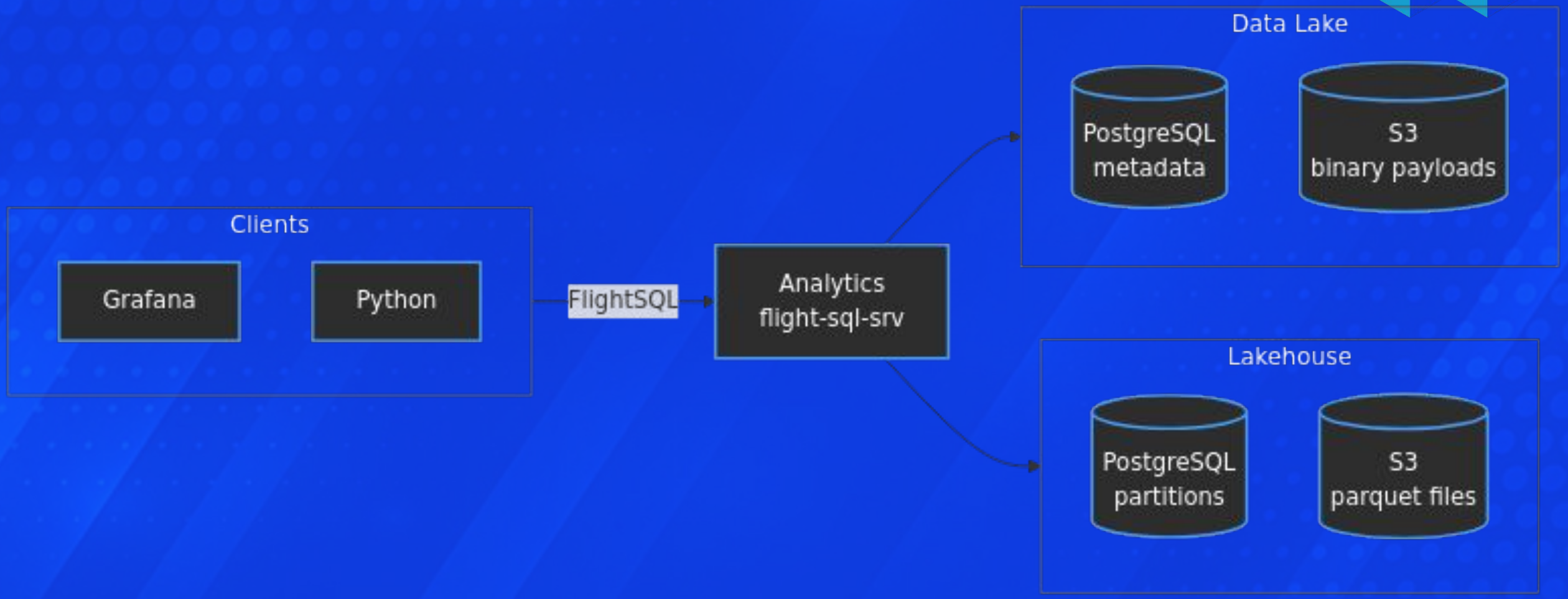
- Stage 1. Low-overhead always-on instrumentation - 20ns/event
- Stage 2. Cheap ingestion & storage
- Stage 3. Daemon processes low-frequency streams (logs & metrics)
- Stage 4. Analytics service executes SQL queries
- Stage 5. Frugal user interface

Nothing in here is specific to games

Architecture Overview - Ingestion



Architecture Overview - Analytics



Stage 1: Low-Overhead Instrumentation

Optimized instrumentation libraries

How?

- Events are tiny (a few bytes)
- Events can contain pointers to avoid repetition (metric names, file paths, …)
- Instrumented threads only queue events - work done in background thread
  - overhead ~20 ns / cpu trace event in calling thread
- Sampling logic applies to event batches
  - ex: does this section of the trace include slow frames?
- Serialization is mostly memcpy
  - Event layout is the native memory format
  - Event layout changes from one process to another
- Fast compression using LZ4

Code Example: Rust Instrumentation


```rust
use micromegas_tracing::prelude::*;

#[span_fn]
async fn process_request(user_id: u32) -> Result<Response> {
    info!("request user_id={user_id}");
    let begin_ticks = now();
    let response = handle_request(user_id).await?;
    let end_ticks = now();
    let duration = end_ticks - begin_ticks;
    imetric!("request_duration", "ticks", duration as u64);
    info!("response status={}", response.status());
    Ok(response)
}
```

Code Example: Unreal Instrumentation


```cpp
#include "MicromegasTracing/Macros.h"

float AMyActor::TakeDamage(float Damage, ...)
{
    MICROMEGAS_SPAN_FUNCTION("Combat");
    float ActualDamage = Super::TakeDamage(...);
    MICROMEGAS_FMETRIC("Combat",
        MicromegasTracing::Verbosity::High,
        TEXT("DamageDealt"), TEXT("points"),
        ActualDamage);
    return ActualDamage;
}
```

Stage 2: Ingestion & Storage

Simple, horizontally scalable design

HTTP service accepts LZ4-compressed payloads

- Metadata → PostgreSQL (for fast lookups)
- Payloads → S3 (for cheap storage)

Data lake: optimized for cheap writes

Stages 3 & 4: daemon & analytics

Bridge between data lake (cheap writes) and lakehouse (fast reads)

- Extract binary payload data in compressed custom format
- Transform into Parquet
- Let DataFusion execute SQL queries on the parquet files

Stage 3: Daemon

Different streams, different strategies

- Logs (low frequency)
  - Process eagerly → Parquet
- Metrics (medium frequency)
  - Process eagerly → Parquet
- Keeps aggregated views up to date
- Ignores CPU traces (very high frequency)

Incremental Data Reduction

Example: SQL-defined log_stats view

```sql
SELECT date_bin('1 minute', time) as time_bin,
       process_id,
       level,
       target,
       count(*) as count
FROM log_entries
WHERE insert_time >= '{begin}'
AND insert_time < '{end}'
GROUP BY process_id, level, target, time_bin
```

Allows to query data over multiple days

Stages 4 : Analytics

Serves SQL queries over Arrow FlightSQL

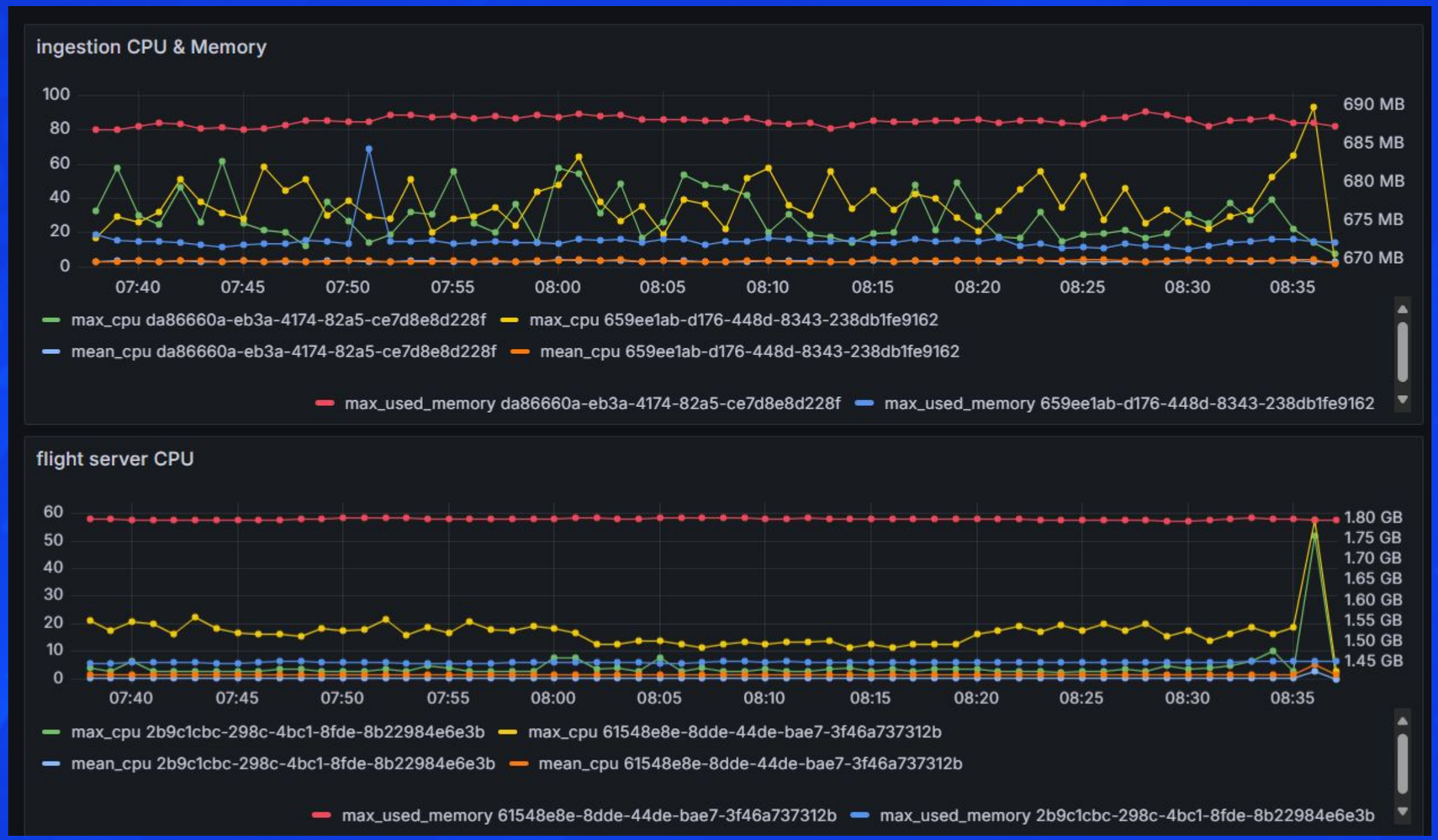
- Transform playloads into Parquet if daemon did not already do it
  - CPU traces
  - Process-specific views
- Let DataFusion SQL engine do its magic
- DataFusion is augmented by extensions
  - JSONB
  - Histograms (median, p99, …)

Stage 5: User Interfaces

Three main interfaces

- Grafana
  - dashboards and alerting
- Jupyter Notebooks
  - Python API for data exploration
- Perfetto
  - Deep trace visualization

Grafana Dashboard



Jupyter Notebooks

Python API for flexible data exploration and custom analysis

```
import datetime
import pandas as pd
import micromegas
pd.set_option('display.max_rows', 30)
client = micromegas.connect()
```

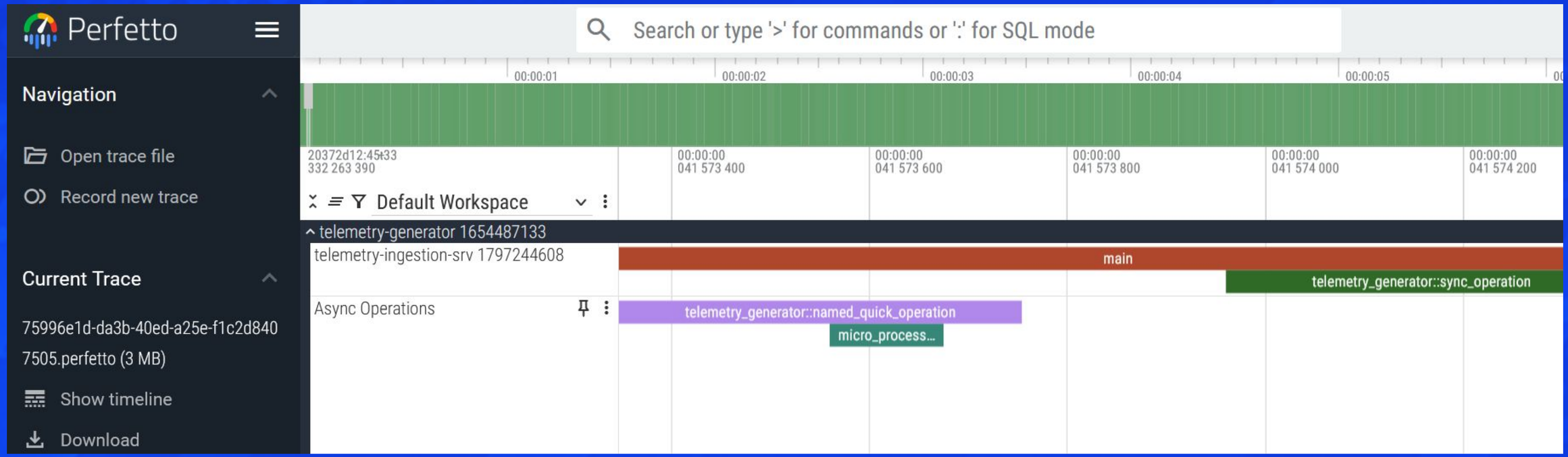
```
%%time
now = datetime.datetime.now(datetime.timezone.utc)
begin = now - datetime.timedelta(days=7)
end = now
sql = """
SELECT time, level, exe, target, msg
FROM log_entries
WHERE level <= 4
ORDER BY time DESC
limit 10
"""
client.query(sql, begin, end)
```

```
CPU times: user 0 ns, sys: 2.35 ms, total: 2.35 ms
Wall time: 62.7 ms
```

	time	level	exe	target	msg
0	2025-10-31 11:24:33.573681118+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::lakehouse::write_partition	[PARTITION_WRITE_COMMIT] view=streams/global t...
1	2025-10-31 11:24:33.571009166+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	[PARTITION_WRITE_START] view=streams/global ti...
2	2025-10-31 11:24:33.570000195+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	creating empty partition record for [streams, ...
3	2025-10-31 11:24:33.563986492+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	writing [streams, global] 2025-10-31T11:24:31+...
4	2025-10-31 11:24:33.563922756+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	[2025-10-31T11:24:31+00:00, 2025-10-31T11:24:3...
5	2025-10-31 11:24:33.557295458+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::lakehouse::write_partition	[PARTITION_WRITE_COMMIT] view=processes/global...
6	2025-10-31 11:24:33.554438214+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	[PARTITION_WRITE_START] view=processes/global ...
7	2025-10-31 11:24:33.553383338+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	creating empty partition record for [processes...
8	2025-10-31 11:24:33.544960836+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	writing [processes, global] 2025-10-31T11:24:3...
9	2025-10-31 11:24:33.544897840+00:00	4	/home/mad/micromegas/rust/target/debug/telemet...	micromegas_analytics::response_writer	[2025-10-31T11:24:31+00:00, 2025-10-31T11:24:3...

Perfetto Trace Viewer

Detailed CPU trace analysis



Thank You

Micromegas would not be possible without open source

- FlightSQL, DataFusion, Apache Arrow, Parquet
- InfluxData
- PostgreSQL
- Rust

And many other amazing projects

Micromegas is Open Source

🌟 https://github.com/madesroches/micromegas

- Drop a star (always makes my day!)
- Try it out, use it as a library, copy the code
- Open an issue, tell me what's missing
- Share your use cases
- madesroches@gmail.com

Q&A