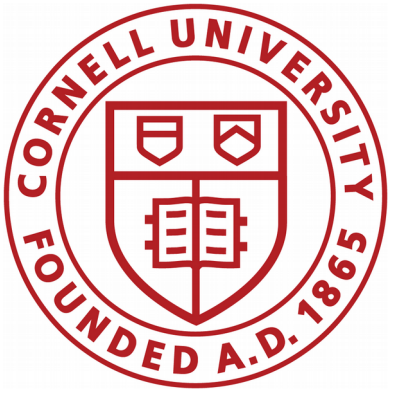


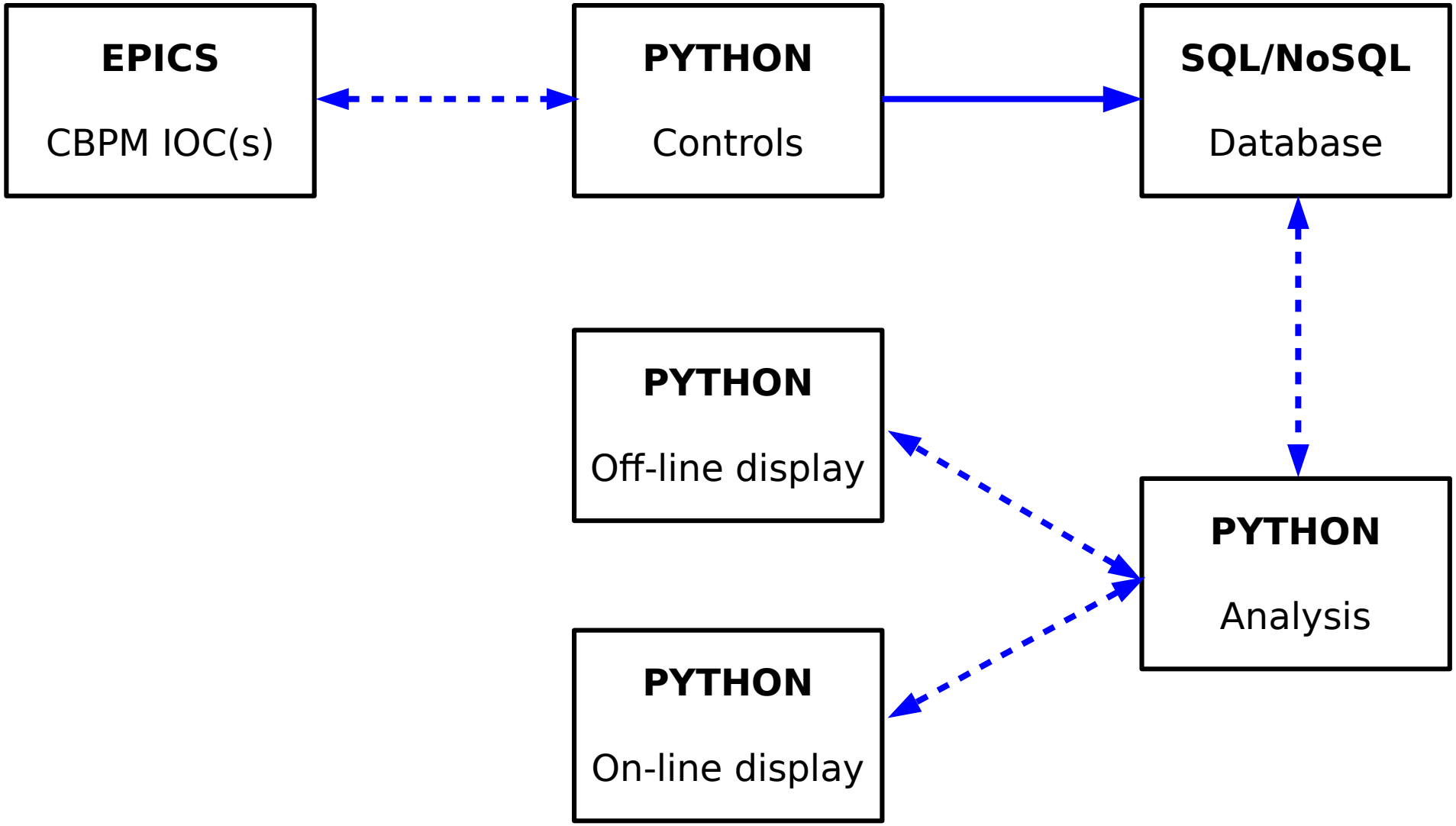
CBPM3 development

Antoine

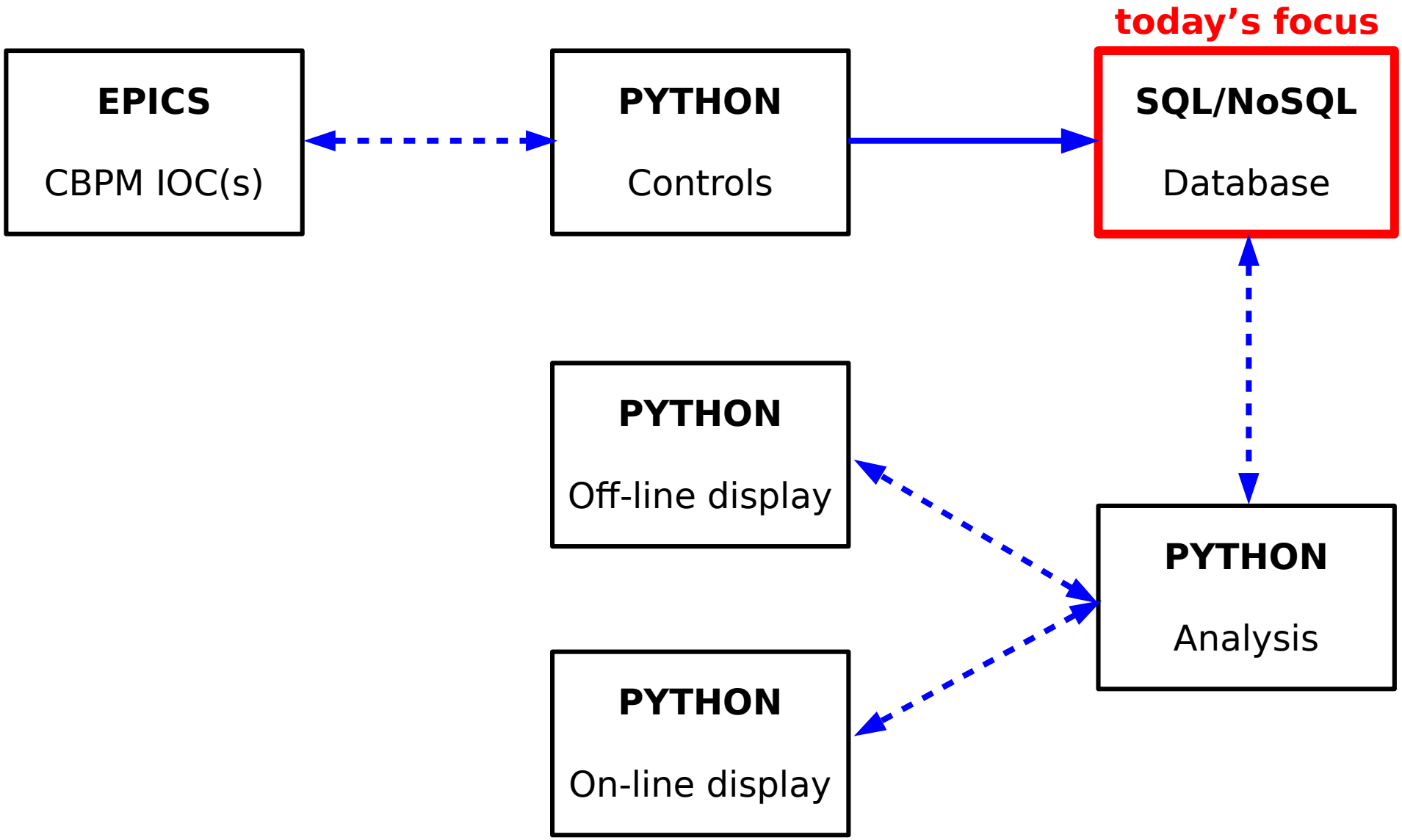
CBPM meeting

May 1, 2020

Goal: mock end-to-end system



Goal: mock end-to-end system



PostgreSQL write

Write test: INSERT implementation

The write rate test is as follow:

- ✗ 120 tables in database (one per instrument)
- ✗ each row (entry) has 6 values: instr id, timestamp and 4 button values
- ✗ code executes sequentially 120 individual queries and one common commit
- ✗ one connection left open

```
def populate(conn, data, n_instr):
    cur = conn.cursor()

    for i in range(n_instr):
        sql = "INSERT INTO instr" + str(i) + "(timestamp, top_in, bot_in, bot_out, top_out) VALUES(%s, %s, %s, %s, %s);"

        cur.execute(
            sql,
            (
                time.time(),
                data[1],
                data[2],
                data[3],
                data[4]
            )
        )

    conn.commit()
```

Multi-threading

Code dispatches individual threads. Each thread write one entry (one trigger) to the 120 tables. All the threads are running on the same CPU and thus share its time and are potentially competing against each other.

```
pool = ThreadPoolExecutor(max_workers=10)

try:
    while True:
        data = [time.time(), counter, counter, counter, counter]
        pool.submit(psql.populate, conn, data, n_instr)
        counter += 1
        time.sleep(0.001)
except KeyboardInterrupt:
    conn.close()
```

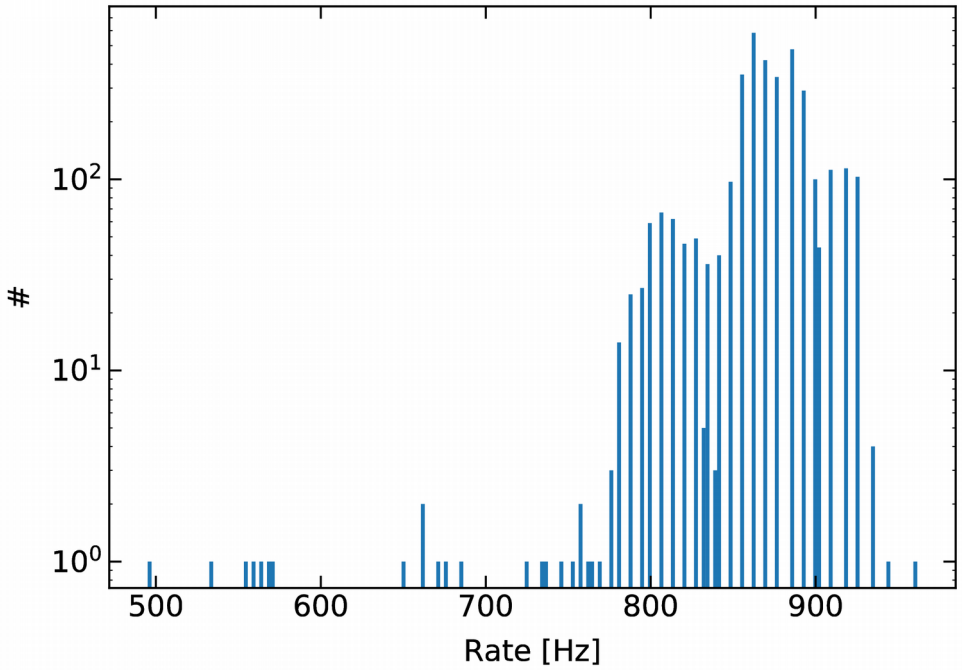
limit to 10 active threads at a time (the rest waits in memory to start)

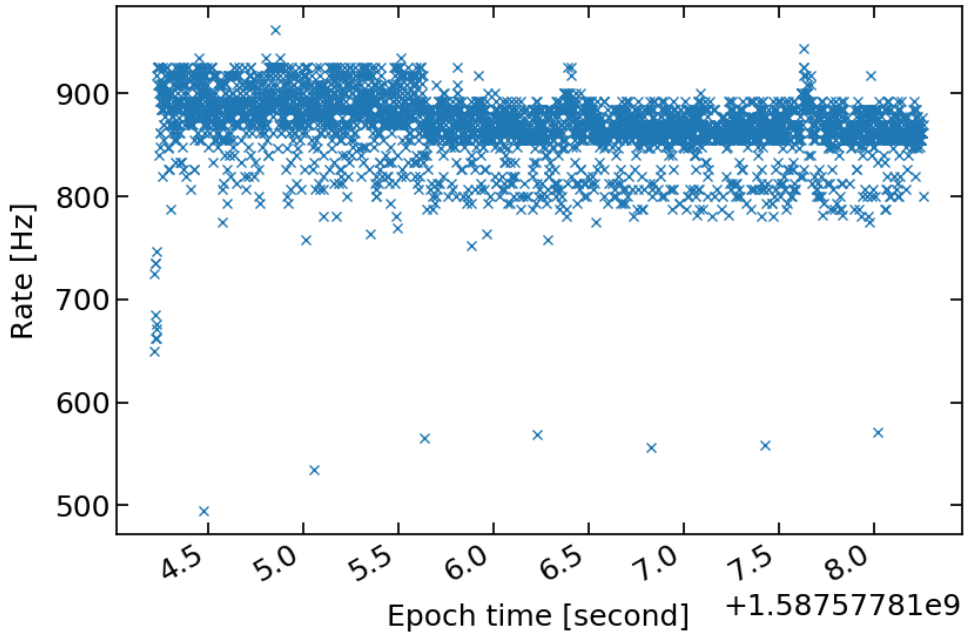
sleep required to avoid the number of thread to blow up and crash the machine (value iteratively tweaked started from 0.1 s down to 1 ms)

There is one database connection share between all the threads.

Multi-threading

Code dispatches individual threads. Each thread write one entry to the 120 tables. All the threads are running on the same CPU and thus share its time and are potentially competing against each other.



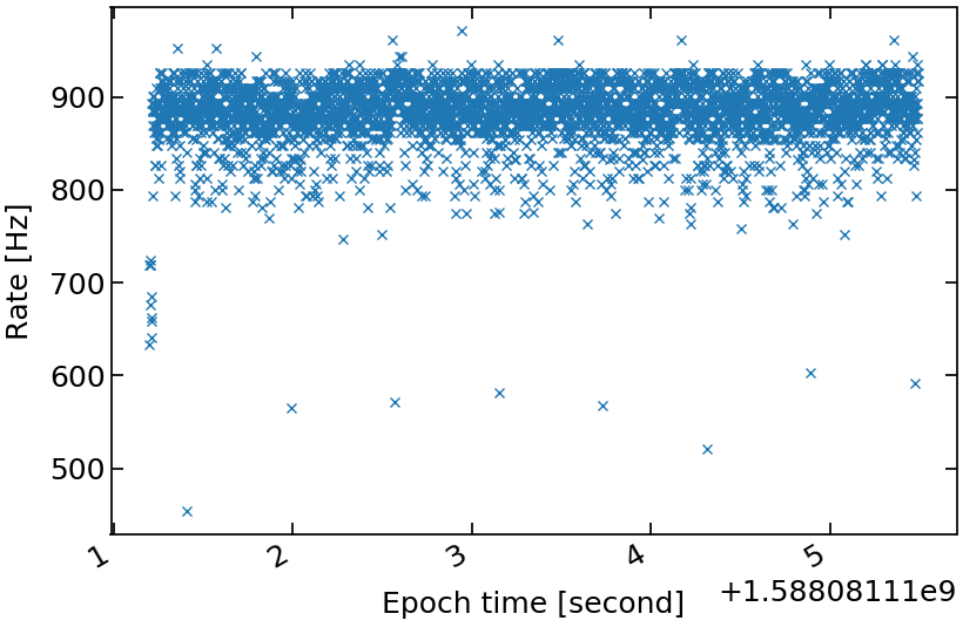


→ multi-threading clearly helps, up to a factor ~20. This is though a solution that adds code complexity to ensure thread safety and book-keeping.

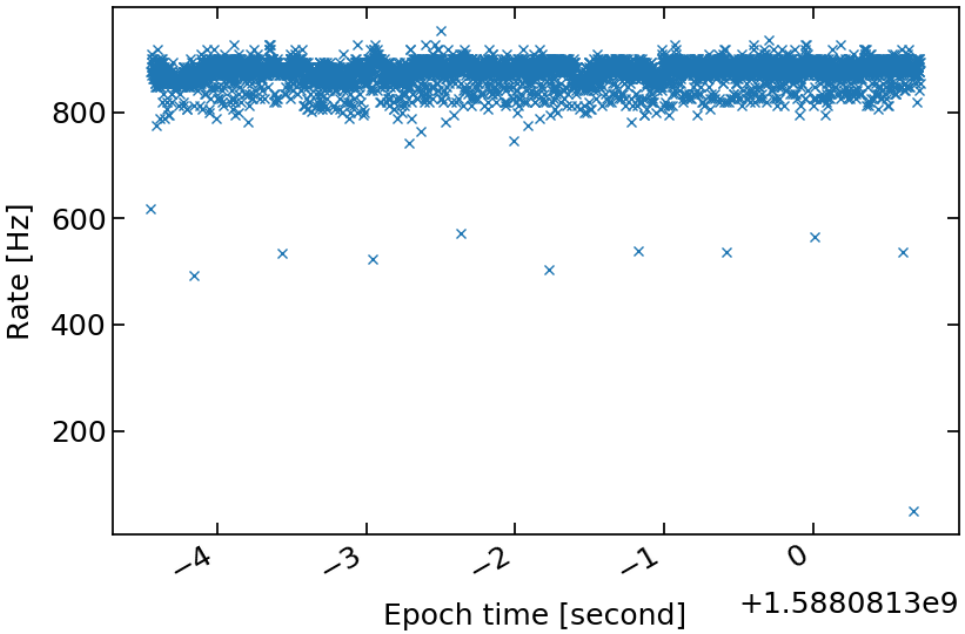
Multi-threading: number of threads

What happens if we change the limit of thread that can run at the same time?

10 threads



1 thread

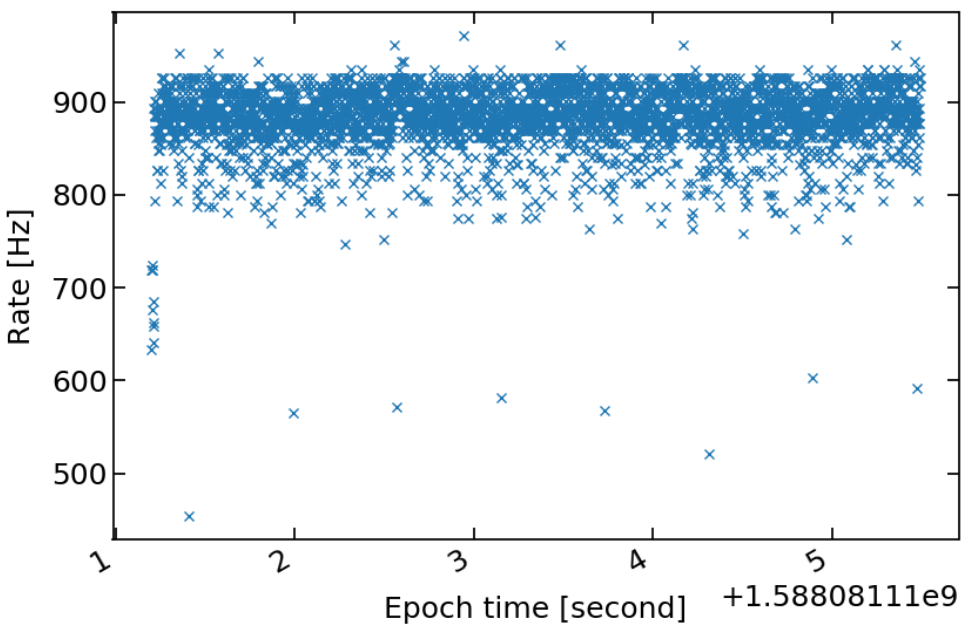


I think it makes sense: the threads utilize a common connection to the database thus only one thread at a time can talk to the database.

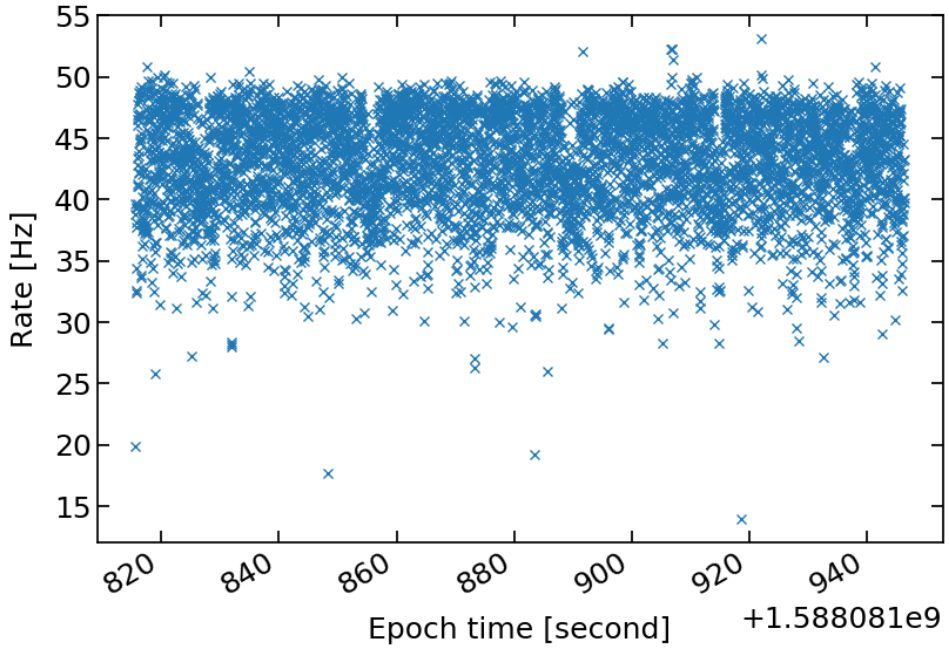
Multi-threading versus no-threading

Then why is multi-threading faster than the no-threading case?

multi-threading



no-threading

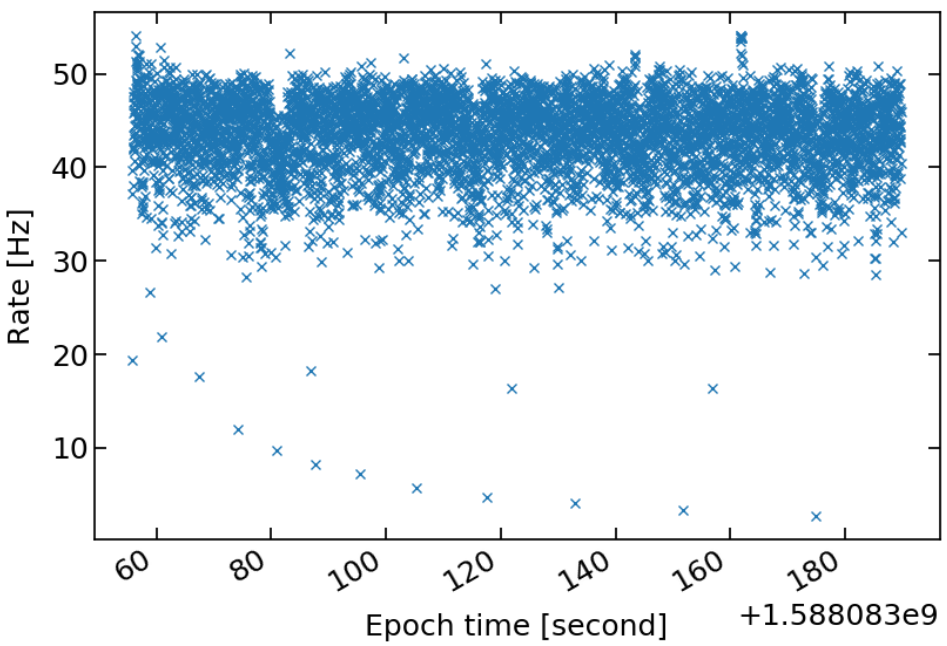


It actually is not... the timestamp written to the database for the multi-threading case was the one corresponding to the thread launching and not to the actual query to the database.

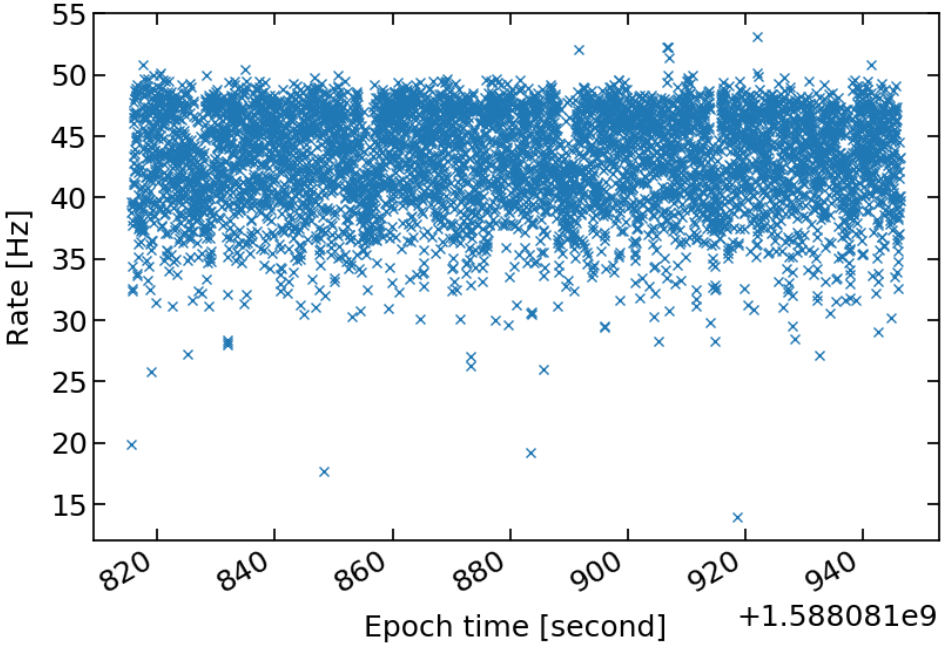
Multi-threading versus no-threading

Multi-threading is not faster...

multi-threading



no-threading

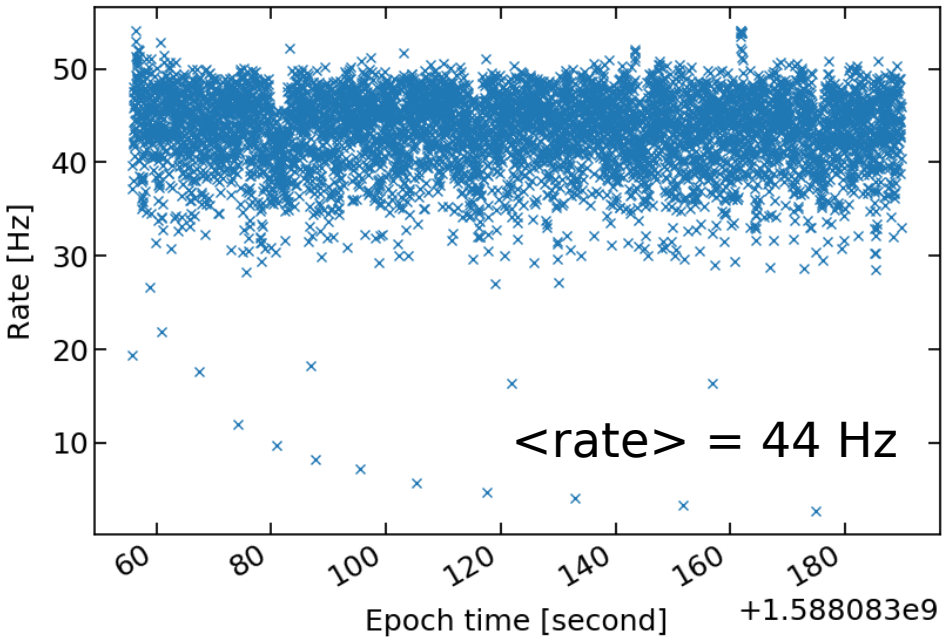


It actually make sense that multi-threading does not help in this case: one connection can only handle one query at a time so there is no actual multi-threading happening.

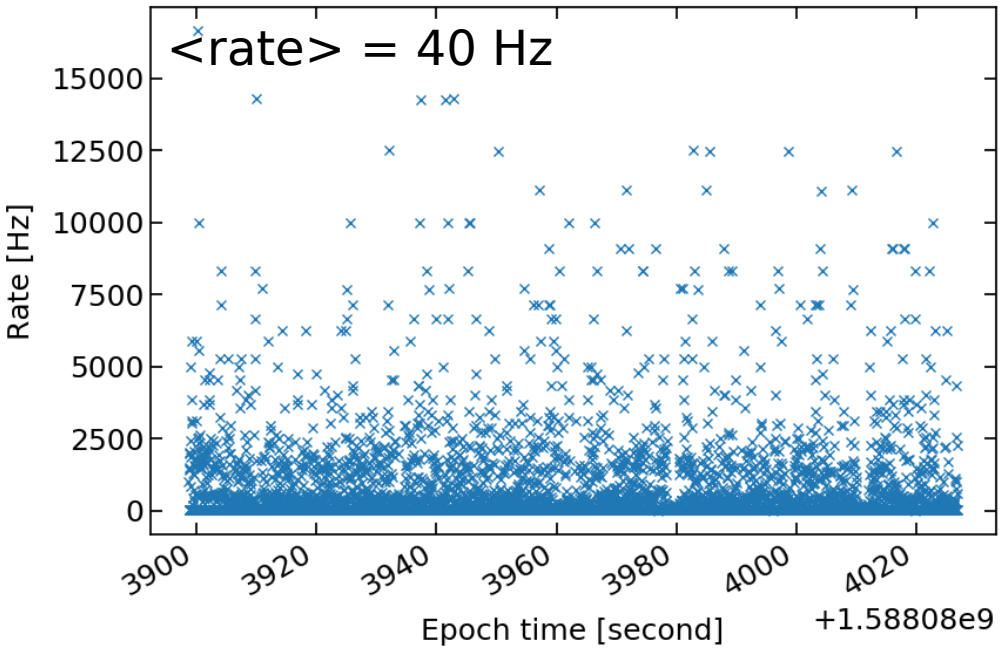
Multi-threading with multiple connection

What happens if each thread manages its own connection to the database?

multi-threading w/ common connection



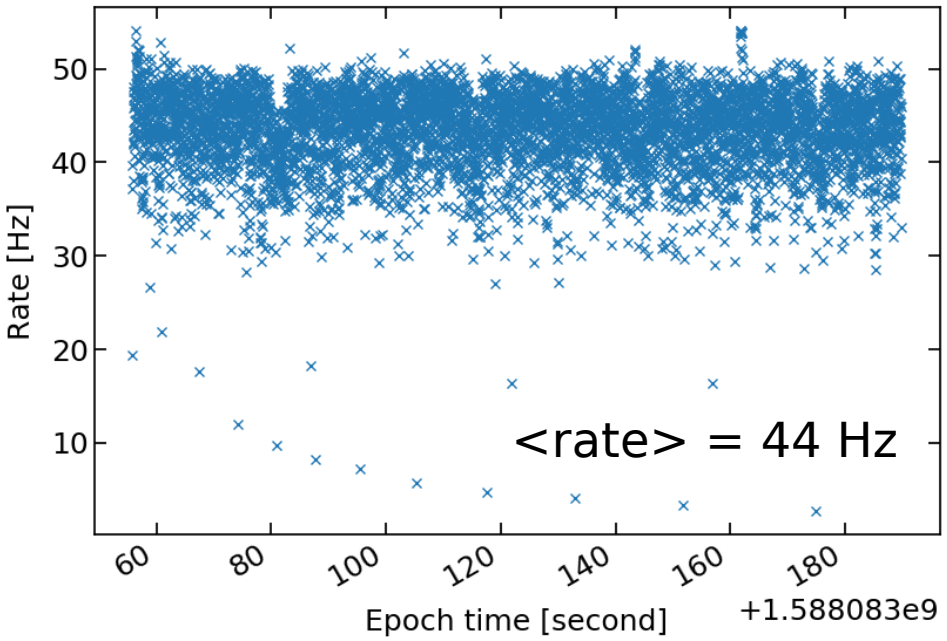
Multi-treading w/ individual connections (2 threads)



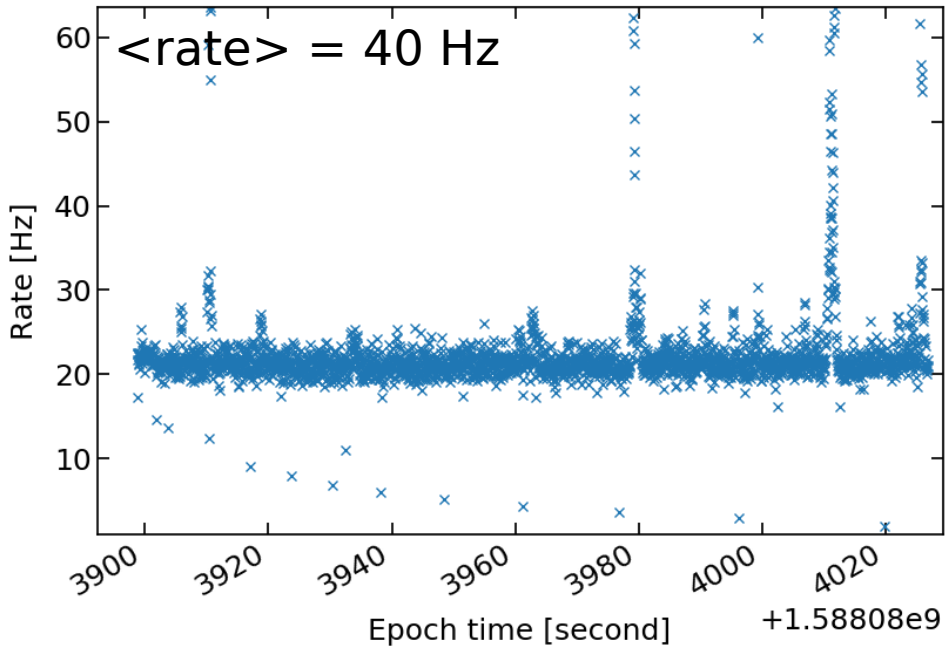
Multi-threading with multiple connection

What happens if each thread manages its own connection to the database?

multi-threading w/ common connection



multi-treading w/ individual connections (2 threads)



The bulk of the 2-thread scenario is half the average rate of the 1-thread scenario, meaning that opening/closing X connections takes down the rate by X/2.

Connection pooling could potentially alleviate that. It consists in creating a set of active connections that are being re-used, no need to open new connections. I will not investigate it for now as we have a 400 Hz non multi-threading solution.

PostgreSQL write + read

Outlook

On a pure “write to database” basis (no concurrent read), we have many options to reach a rate a lot higher than the 10 Hz we aim for. The safest and most straightforward is to aggregate all the data from one trigger (i.e. from 120 instruments) and do a COPY query.

Next:

- ✗ run a concurrent read to see how performance holds
- ✗ benchmark MariaDB and MongoDB

Additional materials