Performance is a FEATURE

Moving compute closer to data

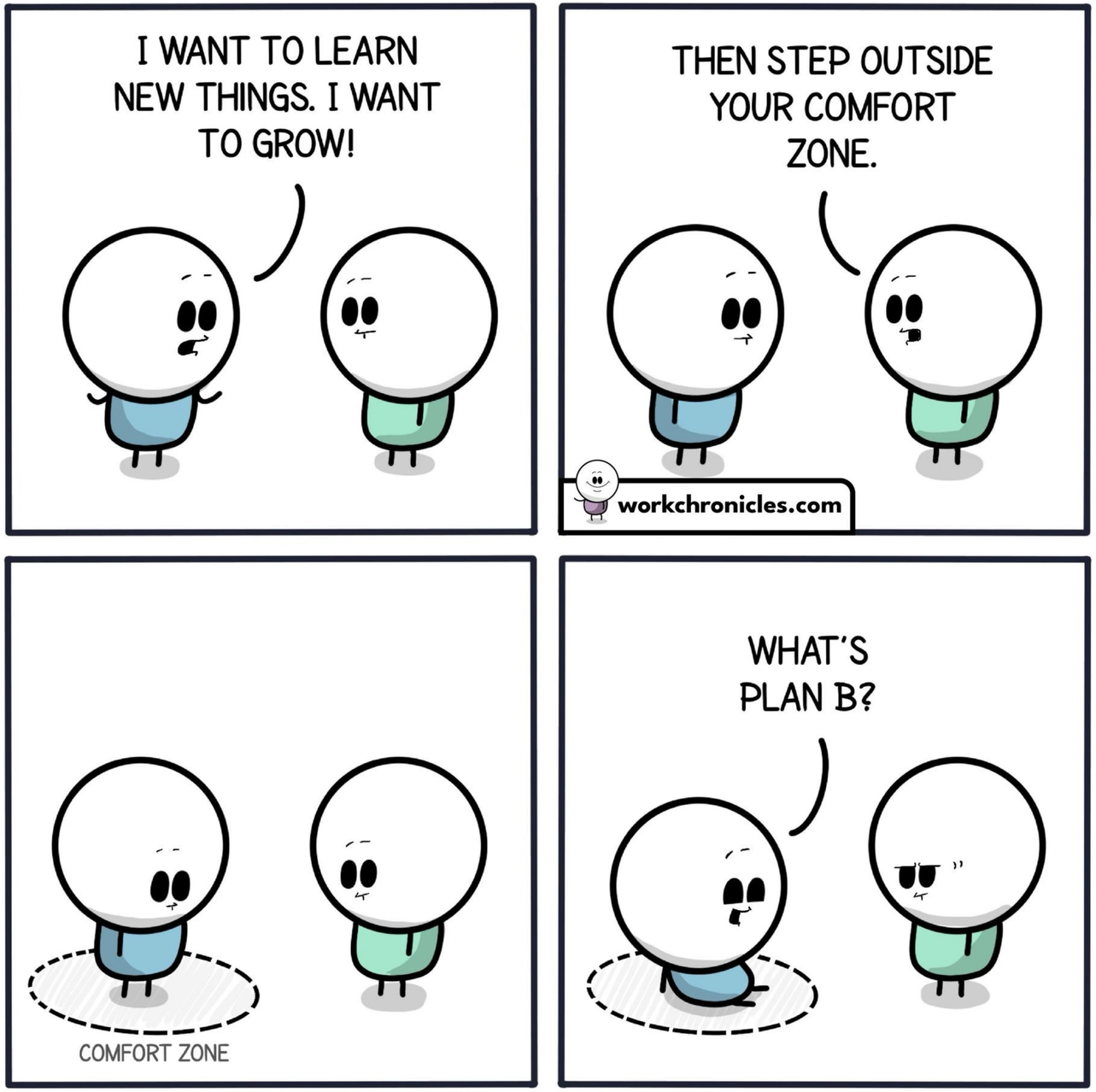
Anup Sharma (Nutanix)

Mehboob Alam (PostgresNX)

Disclaimer

The views, opinions, and conclusions presented in these slides are solely my own and do not reflect the official stance, policies, or perspectives of my employer or any affiliated organization. Any statements made are based on my personal experiences and research and should not be interpreted as official guidance or endorsement.

You?



Comics about work. Made with ♥ & lots of coffee.
Get the comics straight to your Inbox. Join the Newsletter.

Work Chronicles
workchronicles.com

Postgres for Developers

Don’t Fear the Database

ORIGINAL IDEA

50x

PLATFORM

POSTGRES IS NOT A DATABASE

A. BASIC NETWORK THEORY
B. NETWORK LATENCY
C. DISTRIBUTED SYSTEMS
D. MOVING CLOSER TO DATA
E. NO ORMS
F. FUTURE ARCHITECTURES
G. QUESTIONS/DISCUSSIONS

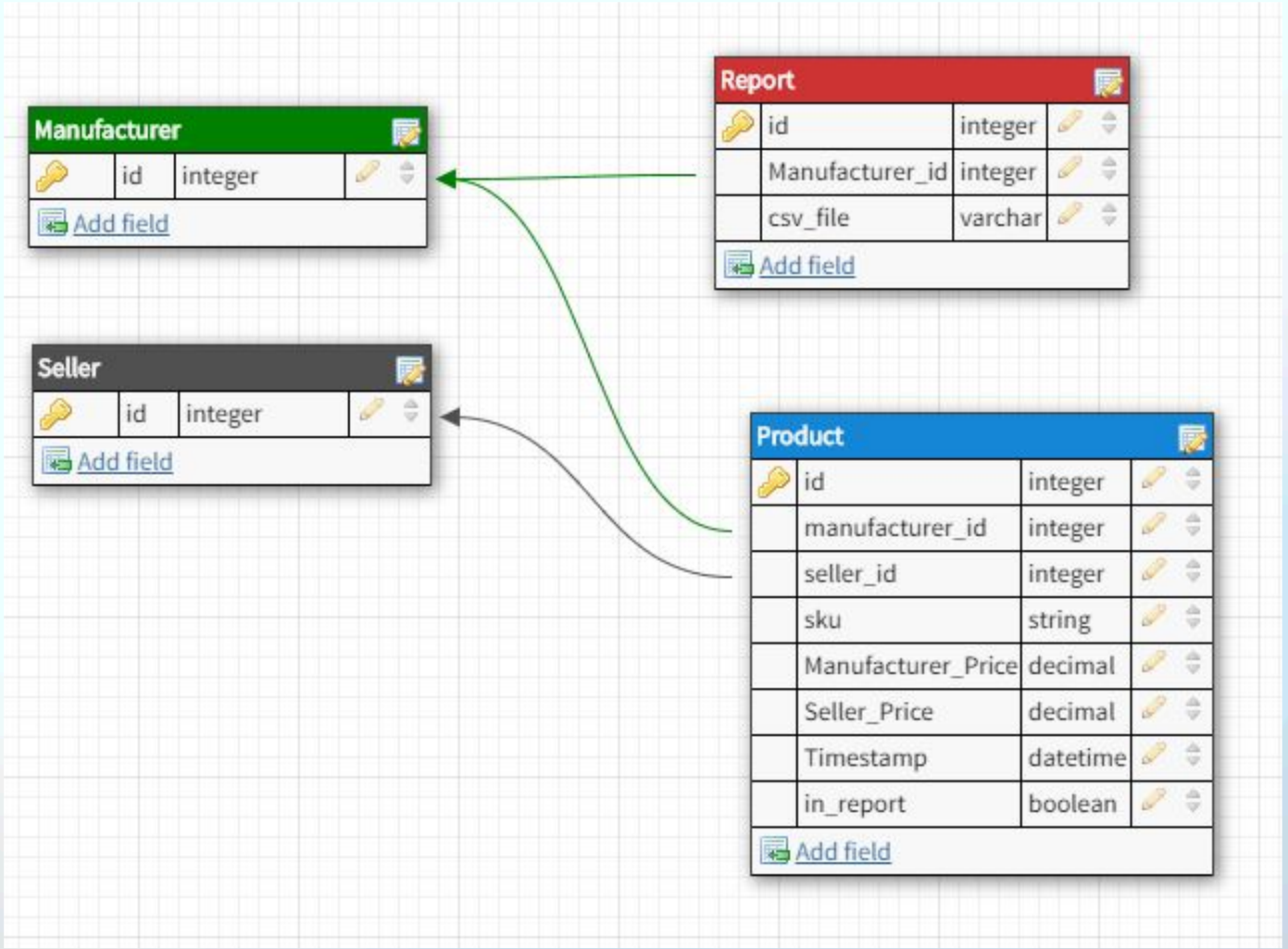
NETWORK LATENCY

Latency numbers every programmer should know

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

TRADITIONAL PYTHON

CONNECT TO POSTGRES



https://neon.tech/postgresql/postgresql-python

TRADITIONAL PYTHON

SEND QUERY

```
import psycopg2

# establishing the connection

conn = psycopg2.connect(

database="test",

    user='postgres',

    password='password',

    host='localhost',

    port= '5432'

)
```

TRADITIONAL PYTHON

SEND QUERY

```
sql = '''CREATE TABLE WORKER(

ID BIGSERIAL NOT NULL PRIMARY KEY,

NAME VARCHAR(100) NOT NULL,

COUNTRY VARCHAR(50) NOT NULL,

AGE INT,

SALARY FLOAT

)'''

cursor.execute(sql)

conn.close()
```

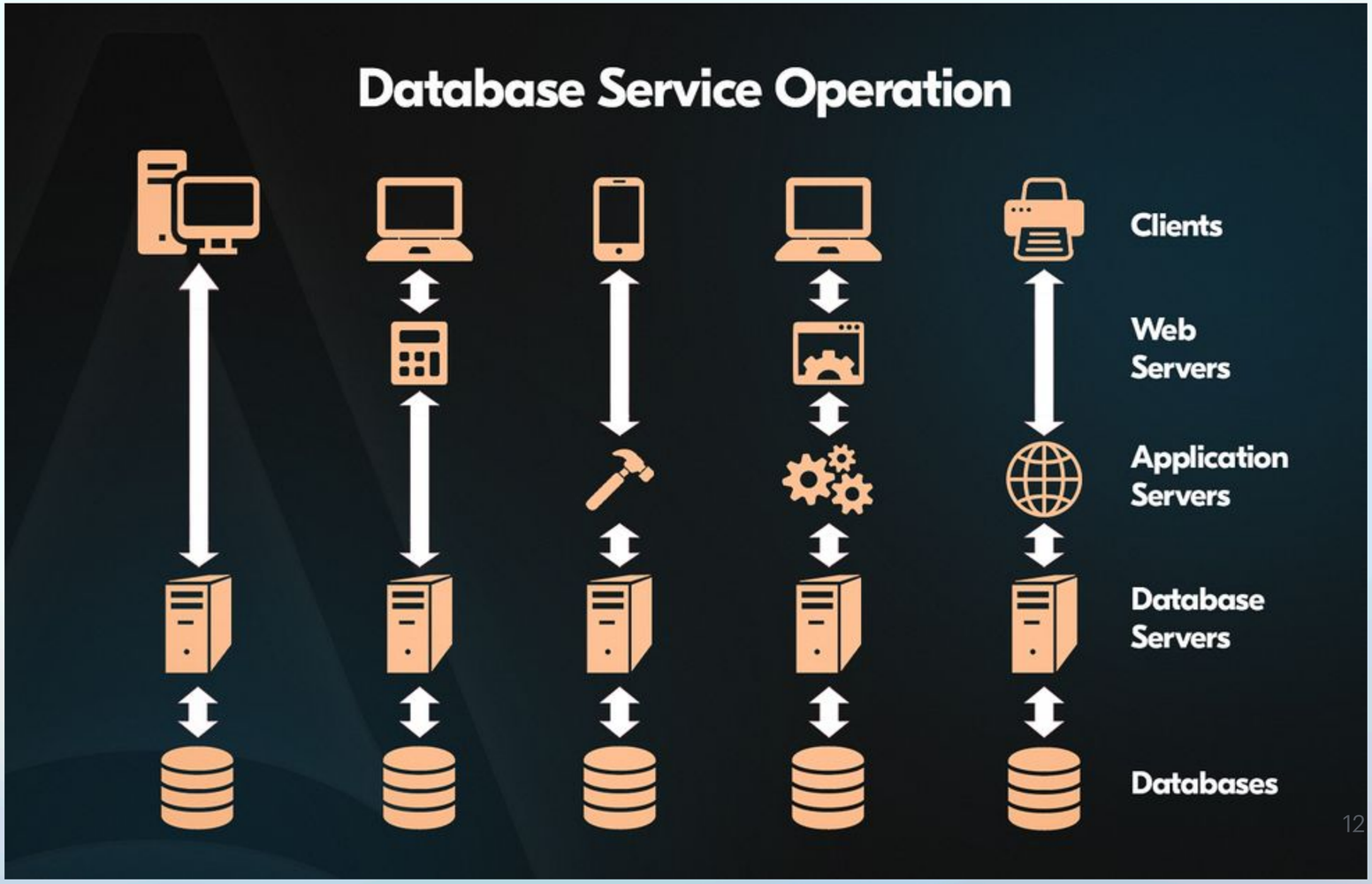
NETWORK LATENCY

Latency numbers every programmer should know

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

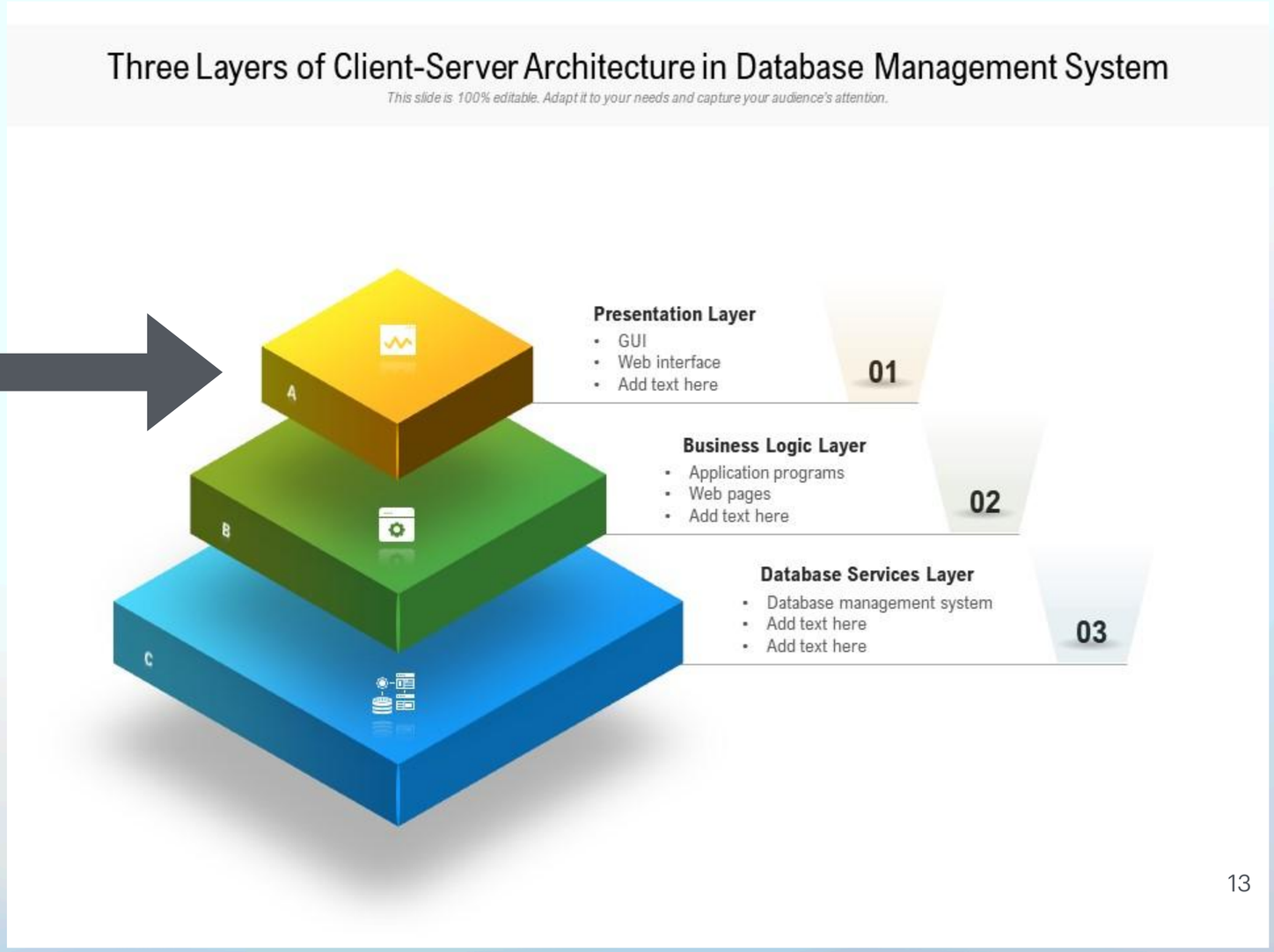
TRADITIONAL PYTHON

CONNECT TO POSTGRES



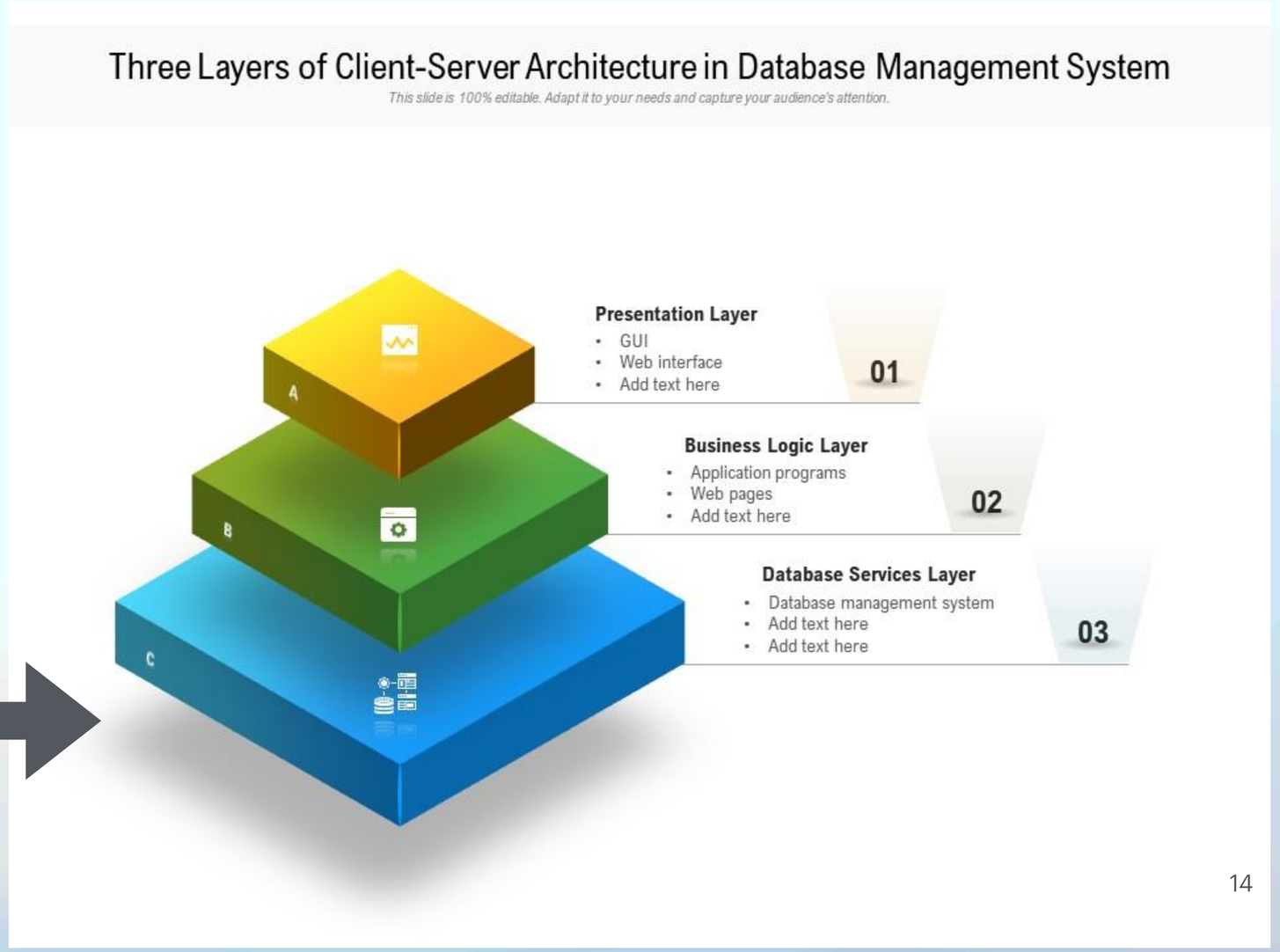
TRADITIONAL PYTHON

CONNECT TO POSTGRES



MODERN PYTHON

EMBED IN POSTGRES



PLATFORM

A. BASIC NETWORK THEORY
B. NETWORK LATENCY
C. DISTRIBUTED SYSTEMS
D. MOVING CLOSER TO DATA
E. NO ORMS
F. FUTURE ARCHITECTURES
G. QUESTIONS/DISCUSSIONS

POSTGRES EXTENSIONS

CREATE EXTENSION

WHAT IS AN EXTENSION

LANGUAGES/PACKAGES

COMPILE/BUILD

SYNTAX REQUIREMENTS

MY JOURNEY

Anup Sharma

PL/PYTHON

Safety and Efficiency Matter

DYNAMIC vs COMPILED LANGUAGES

Why Rust?

A brief overview of Rust as a programming language

Showcasing its feature in comparison with C

Modern Compiled Language

Safety by Construction

Low Level Control

Compatibility with C

Tooling (pkgs, build, testing)

PL/RUST

INTRO TO PGRX

Extensions framework for PostgreSQL

- PGRX exposes the PostgreSQL C API via safe Rust code, removing many opportunities for crash or corruption
- PGRX Rust code is compiled and runs close to the speed of C, and many times faster than code in a PL/ dynamic language
- PGRX helps out with your development process; from auto-creating SQL objects, to testing and packaging your extension
- PGRX makes high performance PostgreSQL Extensions more accessible

HOW TO USE IT

A fully managed development environment ...

- cargo pgrx new: Create new extensions quickly
- cargo pgrx init: Install new (or register existing) PostgreSQL installs
- cargo pgrx run: Run your extension and interactively test it in psql (or pgcli)
- cargo pgrx test: Unit-test your extension across multiple PostgreSQL versions
- cargo pgrx package: Create installation packages for your extension

BENEFITS

Target Multiple Postgres Versions

- Support from Postgres 12 to Postgres 17 from the same codebase
- Use Rust feature gating to use version-specific APIs
- Seamlessly test against all versions

SAFETY FEATURES

PL/Rust

- Translates Rust panic! into Postgres ERROR that abort the transaction, not the process
- Memory Management follows Rust's drop semantics, even in the face of panic! and elog(ERROR)
- #[pg_guard] procedural macro to ensure the above
- Postgres Datums are Option<T> where T: FromDatum
- NULL Datums are safely represented as Option::<T>::None

BENCHMARK RESULTS

FAKE DATA SIMULATION

FAKER-JS ~ 1M RECORDS

index.js

```
const { faker } = require('@faker-js/faker');
const fs = require('fs');

function generateTransaction() {
    return [
        faker.string.uuid(), // Unique ID
        faker.date.past().toISOString(), // Date
        faker.person.fullName(), // Customer Name
        faker.internet.email(), // Email
        faker.finance.amount(10, 1000, 2), // Amount
        faker.finance.transactionType(), // Payment Method
        faker.helpers.arrayElement(['Pending', 'Completed', 'Failed']), // Status
    ];
}

function generateTransactions(num) {
    const transactions = [];
    for (let i = 0; i < num; i++) {
        transactions.push(generateTransaction());
        if (i % 100000 === 0) {
            console.log(`Generated ${i} transactions...`);
        }
    }
    return transactions;
}

// Generate 1 million transactions
const totalTransactions = 1_000_000;
const transactions = generateTransactions(totalTransactions);

// Convert to CSV format
const header = 'ID,Date,CustomerName,Email,Amount,PaymentMethod,Status\n';
const csvContent = transactions.map(t => t.join(',')).join('\n');

// Save to a CSV file
fs.writeFileSync('transactions.csv', header + csvContent);

console.log(`Successfully generated ${totalTransactions} transactions and saved to transactions.csv`);
```

This is how the data looks like

transaction.txt

```
root@ubuntudemo ~/pgexp/faker-example # cat transactions.csv | head -10
ID,Date,CustomerName,Email,Amount,PaymentMethod,Status
c91207a9-60ce-4826-b2be-6b5675021ab6,2024-10-23T23:38:40.579Z,Roman Rosenbaum,Sally2@gmail.com,162.97,withdrawal,Completed
2dca1aaa-55b3-42f2-8452-90f0f746c5ac,2024-03-02T16:47:04.923Z,Joel Ferry,Dillon.Grimes@hotmail.com,150.63,payment,Failed
95ba47c6-3377-44d4-bf6f-926f840177f9,2024-07-31T16:18:10.223Z,Melinda Kuvalis,Brook_Wiegand@yahoo.com,148.96,invoice,Completed
155803bb-a45e-4ade-8cc0-503c96a2ab60,2024-12-13T14:57:27.647Z,Jeremy Ondricka,Myra79@gmail.com,896.99,payment,Pending
ca7783aa-8a26-48d3-92e5-d4703085f9e0,2024-02-04T04:38:39.740Z,Faith Ledner,Vivienne85@gmail.com,405.89,withdrawal,Pending
6e0e80c0-3d96-4d32-916f-a4f7b3695d1f,2024-08-19T06:39:01.221Z,Joseph O'Reilly,Cleora.Olson81@hotmail.com,988.76,invoice,Completed
d74a094c-c485-4bc1-a304-8665ab6dff5f,2024-06-26T08:07:22.193Z,Jennie Corkery,Robin14@gmail.com,431.43,payment,Completed
8a10aa2c-73a3-4a76-8643-de5f657ae21b,2024-02-21T18:43:56.534Z,Dr. Carlos Fisher I,Bernadette_Marvin28@gmail.com,190.52,withdrawal,Failed
583ff215-2d3f-4118-8916-3dbb7fd1644e,2024-05-08T10:09:35.361Z,Mrs. Suzanne Orn,Bernardo27@gmail.com,18.50,payment,Failed
```

setup.sh

```
root@ubuntudemo ~/pgexp/fastrx/db-bench-net-latency # cat setup.sh
#!/bin/bash

# Check if PostgreSQL is installed
check_installed() {
    if ! command -v psql &> /dev/null; then
        echo "PostgreSQL is NOT installed. Please install it first."
        exit 1
    fi
    echo "PostgreSQL is installed."
}

# Check if PostgreSQL service is running
check_running() {
    if ! pg_isready -q; then
        echo "PostgreSQL is NOT running. Please start the service."
        exit 1
    fi
    echo "PostgreSQL is running."
}

# Run checks
check_installed
check_running

# Proceed with database setup
echo "Setting up PostgreSQL database..."

sudo -u postgres psql -t -c 'CREATE DATABASE test_transaction'
sudo -u postgres psql test_transaction -t < create.sql
sudo -u postgres psql test_transaction -t -c '\timing' -c "\\copy store_nx FROM 'transactions.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',');"

echo "Done!"
```

MOVING AVERAGE

local_benchmark.out

```
(myenv) root@ubuntudemo ~/pgexp/fastrx/db-bench-net-latency # python local_benchmark.py
Connected to the database
Locally

Time taken for compute: 0:00:01.809840

Weekly Average Amount:
EventDate
2024-01-21    501.803721
2024-01-28    497.677958
2024-02-04    499.782711
2024-02-11    501.733576
2024-02-18    496.229913
2024-02-25    505.429129
...
...
...
2024-11-24    501.428493
2024-12-01     500.53383
2024-12-08    497.118089
2024-12-15    501.233535
2024-12-22    500.513281
2024-12-29    503.490907
2025-01-05    500.266006
2025-01-12     497.63711
2025-01-19    500.506667
Freq: W-SUN, Name: Amount, dtype: object
Database connection closed.
```

OVER THE NETWORK

remote_benchmark.out

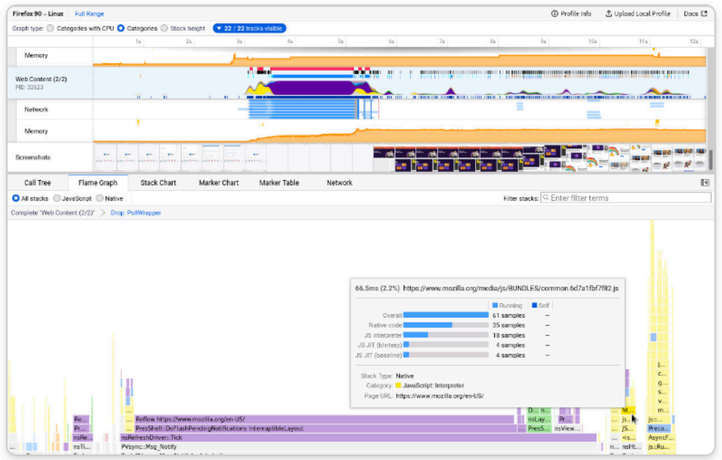
```
❯ python remote_benchmark.py
Connected to the database
Remote

Time taken for compute: 0:00:12.636833

Weekly Average Amount:
EventDate
2024-01-21    501.803721
2024-01-28    497.677958
2024-02-04    499.782711
2024-02-11    501.733576
2024-02-18    496.229913
2024-02-25    505.429129
...
...
...
2024-11-24    501.428493
2024-12-01     500.53383
2024-12-08    497.118089
2024-12-15    501.233535
2024-12-22    500.513281
2024-12-29    503.490907
2025-01-05    500.266006
2025-01-12     497.63711
2025-01-19    500.506667
Freq: W-SUN, Name: Amount, dtype: object
Database connection closed.
```

remote_benchmark.out

```
❯ sudo perf record -g ../../test_transaction/trans/bin/python remote_benchmark.py
Connected to the database
Remote

Time taken for compute: 0:00:12.616733

Weekly Average Amount:
EventDate
2024-01-21    501.803721
2024-01-28    497.677958
2024-02-04    499.782711
2024-02-11    501.733576
2024-02-18    496.229913
...
...
...
2024-12-29    503.490907
2025-01-05    500.266006
2025-01-12     497.63711
2025-01-19    500.506667
Freq: W-SUN, Name: Amount, dtype: object
Database connection closed.
[ perf record: Woken up 6 times to write data ]
[ perf record: Captured and wrote 1.969 MB perf.data (14753 samples) ]
```

Firefox Profiler — Web app for Firefox performance analysis

Capture a performance profile. Analyze it. Share it. Make the web faster.



+ Install the Chrome extension

Documentation

Use the Firefox Profiler extension for Chrome to capture performance profiles in Chrome and analyze them in the Firefox Profiler. Install the extension from the Chrome Web Store.

Once installed, use the extension's toolbar icon or the shortcuts to start and stop profiling. You can also export profiles and load them here for detailed analysis.

`Ctrl` + `Shift` + `1` Stop and start profiling

`Ctrl` + `Shift` + `2` Capture and load profile

Load existing profiles

You can **drag and drop** a profile file here to load it, or:

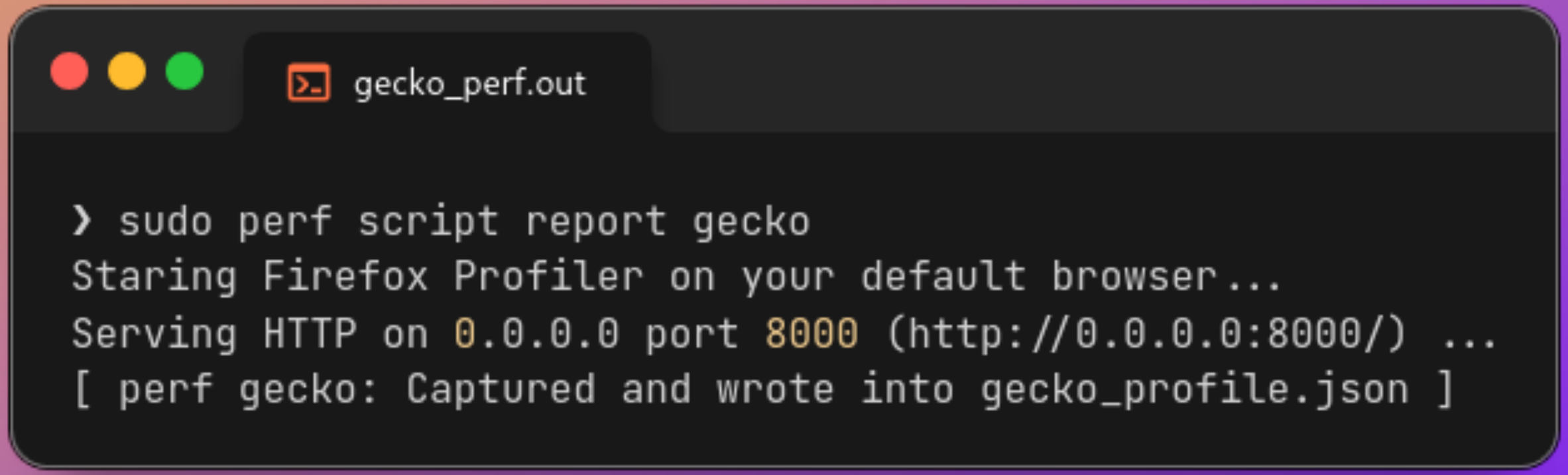
Load a profile from file	Load a profile from a URL

The Firefox Profiler can also import profiles from other profilers, such as Linux perf, Android SimplePerf, the Chrome performance panel, Android Studio, or any file using the dhat format or Google's Trace Event Format. Learn how to write your own importer.

Your recent uploaded recordings

No profile has been uploaded yet!

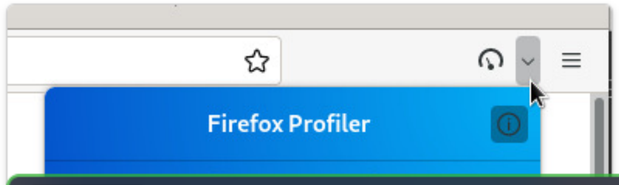
GSOC PROJECT



GSOC PROJECT

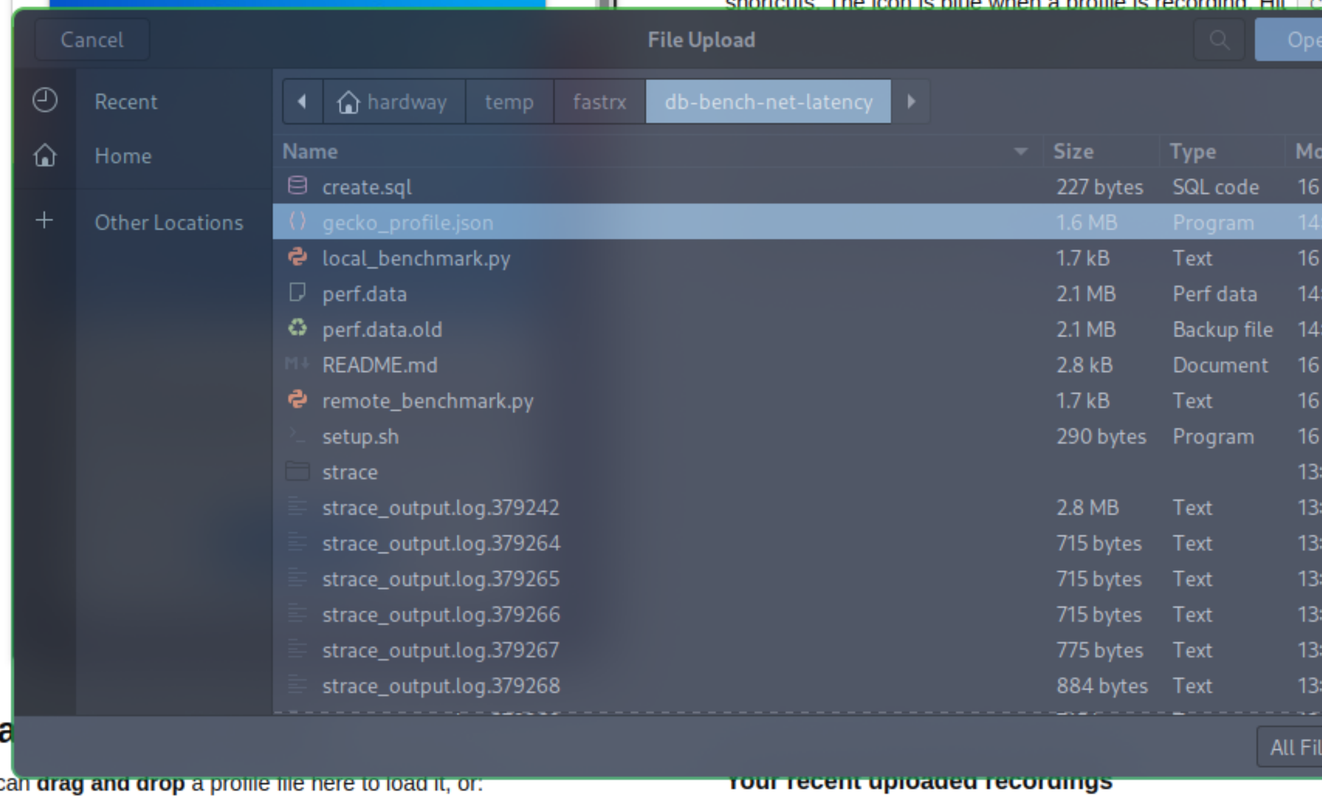
Firefox Profiler — Web app for Firefox performance analysis

Capture a performance profile. Analyze it. Share it. Make the web faster.



Documentation

To start profiling, click on the profiling button, or use the keyb
shortcuts. The icon is blue when a profile is recording. Hit



Loa

You can **drag and drop** a profile file here to load it, or:

Load a profile from file | Load a profile from a URL

The Firefox Profiler can also import profiles from other profilers, such as Linux perf, Android SimplePerf, the Chrome performance panel, Android Studio, or any file using the dhat format or Google's Trace Event Format. Learn how to write your own importer.

You can also compare recordings. Open the comparing interface.

Your recent uploaded recordings

No profile has been uploaded yet!

https://profiler.firefox.com/from-file/calltree/?globalTrackOrder=0&thread=0&timelineType=category&v=10

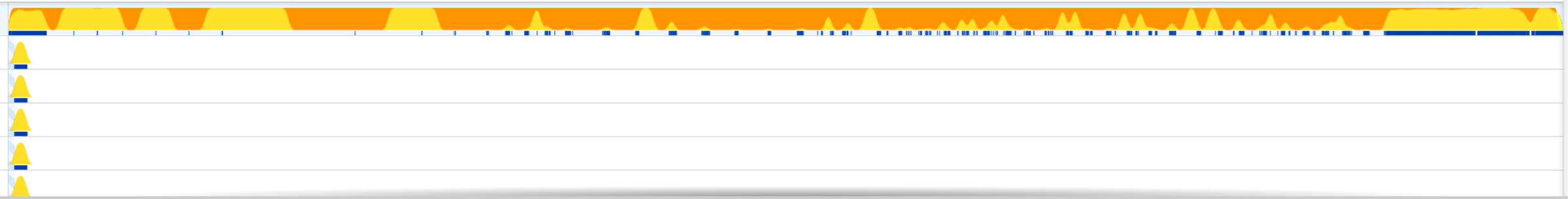
unknown GNU/Linux | Full Range (15s) | Profile Info | Upload Local Profile | Docs

9 / 9 tracks

500ms 1.0s 1.5s 2.0s 2.5s 3.0s 3.5s 4.0s 4.5s 5.0s 5.5s 6.0s 6.5s 7.0s 7.5s 8.0s 8.5s 9.0s 9.5s 10.0s 10.5s 11.0s 11.5s 12.0s 12.5s 13.0s 13.5s 14.0s 14.5s 15.0s

Process 397048

python



Call Tree | Flame Graph | Stack Chart | Marker Chart | Marker Table

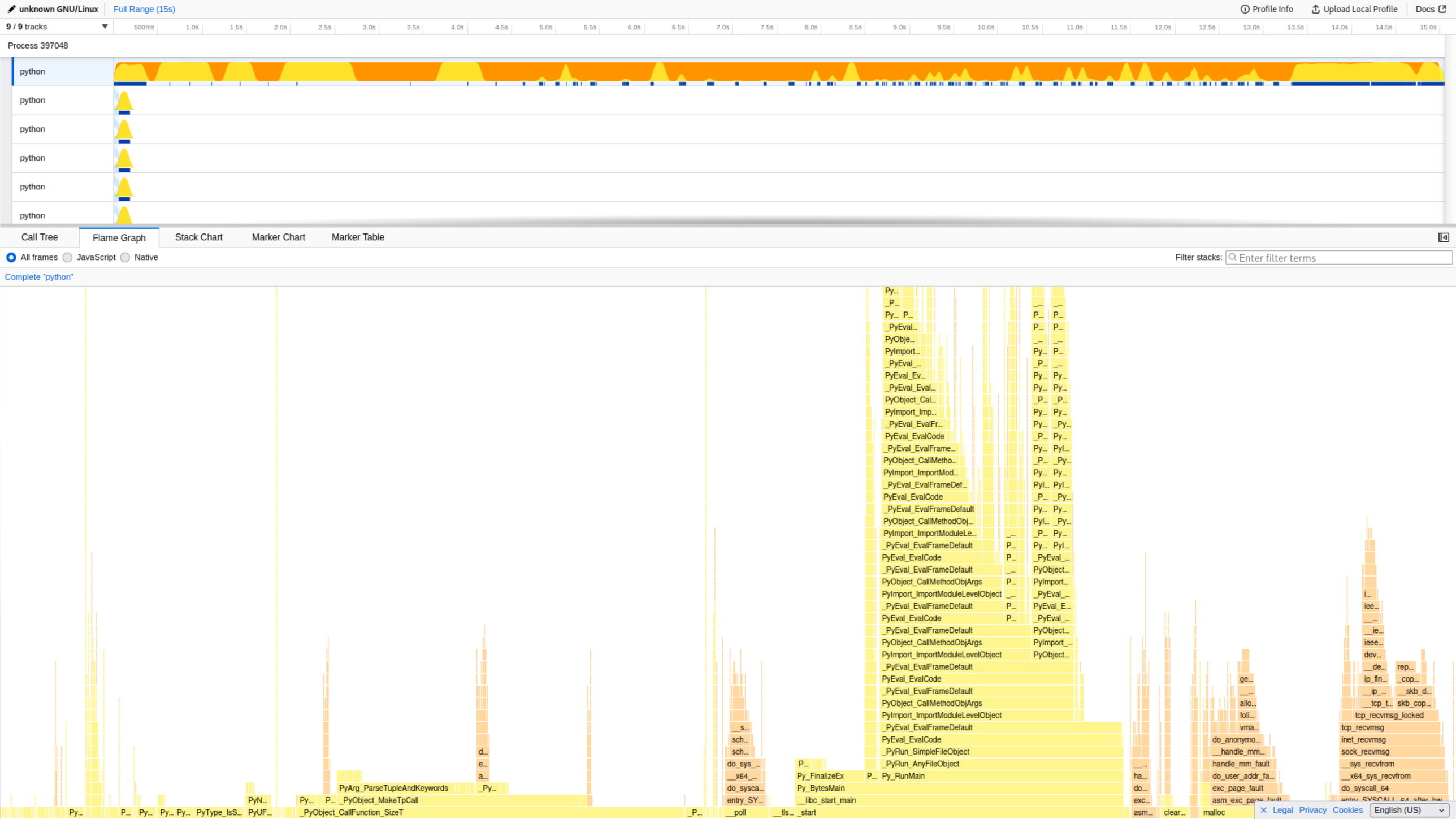
All frames | JavaScript | Native | Invert call stack | Filter stacks: Enter filter terms

Complete "python"

Total (samples)		Self	
27%	2,102	493	▼ _PyObject_CallFunction_SizeT /usr/lib/libpython3.12.so.1.0
17%	1,316	378	▼ _PyObject_MakeTpCall /usr/lib/libpython3.12.so.1.0
8.1%	640	503	▼ PyArg_ParseTupleAndKeywords /usr/lib/libpython3.12.so.1.0
0.6%	50	50	PyLong_AsLong /usr/lib/libpython3.12.so.1.0
0.5%	43	43	PyMem_Free /usr/lib/libpython3.12.so.1.0
0.4%	35	35	PyMem_Malloc /usr/lib/libpython3.12.so.1.0
0.1%	9	9	__tls_get_addr /usr/lib/ld-linux-x86-64.so.2
1.7%	132	58	▶ _PyObject_New /usr/lib/libpython3.12.so.1.0
0.5%	42	42	PyMem_Free /usr/lib/libpython3.12.so.1.0
0.4%	33	33	__tls_get_addr /usr/lib/ld-linux-x86-64.so.2
0.3%	26	26	PyContextVar_Get /usr/lib/libpython3.12.so.1.0
0.3%	24	24	PyObject_Malloc /usr/lib/libpython3.12.so.1.0
0.2%	17	14	▶ PyMem_Malloc /usr/lib/libpython3.12.so.1.0
0.1%	10	10	PyType_IsSubtype /usr/lib/libpython3.12.so.1.0
0.1%	6	6	mpd_qfinalize /usr/lib/libmpdec.so.4.0.0
0.0%	3	3	mpd_qset_string /usr/lib/libmpdec.so.4.0.0
0.0%	2	2	_Py_NewReference /usr/lib/libpython3.12.so.1.0
0.0%	2	2	mpd_maxcontext /usr/lib/libmpdec.so.4.0.0
0.0%	1	—	▶ asm_sysvec_apic_timer_interrupt [kernel.kallsyms]
1.3%	106	105	▶ PyLong_FromLong /usr/lib/libpython3.12.so.1.0
1.0%	76	37	▶ PyMem_Malloc /usr/lib/libpython3.12.so.1.0
0.6%	44	44	__tls_get_addr /usr/lib/ld-linux-x86-64.so.2
0.4%	32	32	PyMem_Free /usr/lib/libpython3.12.so.1.0
0.4%	31	5	▶ asm_exc_page_fault [kernel.kallsyms]
0.0%	3	3	_Py_VaBuildStack_SizeT /usr/lib/libpython3.12.so.1.0
0.0%	1	—	▶ asm_sysvec_apic_timer_interrupt [kernel.kallsyms]
23%	1,792	—	▶ _start /usr/bin/python3.12
8.3%	661	13	▶ recv /usr/lib/libc.so.6
7.0%	554	70	▶ malloc /usr/lib/libc.so.6
3.4%	269	269	PyType_IsSubtype /usr/lib/libpython3.12.so.1.0
3.1%	248	9	▶ __poll /usr/lib/libc.so.6
1.9%	150	85	▶ clear_object_strided_loop /home/hardway/temp/test_transaction/trans/lib/python3.12/site-packages/numpy/_core/_multiarray_umath.cpython-312-x86_64-linux-gnu.so
1.9%	147	5	▶ PyUFunc_OO_O /home/hardway/temp/test_transaction/trans/lib/python3.12/site-packages/numpy/_core/_multiarray_umath.cpython-312-x86_64-linux-gnu.so
1.8%	139	29	▶ asm_exc_page_fault [kernel.kallsyms]
1.6%	126	126	__tls_get_addr /usr/lib/ld-linux-x86-64.so.2
1.4%	110	85	▶ PyObject_RichCompare /usr/lib/libpython3.12.so.1.0

Select a node to display information about it.

Legal Privacy Cookies English (US)


unknown GNU/Linux
Full Range (15s)
Profile Info
Upload Local Profile
Docs
9 / 9 tracks
Process 397048
python
Call Tree
Flame Graph
Stack Chart
Marker Chart
Marker Table
All frames
JavaScript
Native
Filter stacks:
Enter filter terms
Complete "python"
PyArg_ParseTupleAndKeywords
_PyObject_MakeTpCall
_PyObject_CallFunction_SizeT
do_sys_...
__x64_...
do_sysca...
entry_SY...
__poll
__libc_start_main
_start
Py_FinalizeEx
Py_BytesMain
Py_RunMain
_PyRun_SimpleFileObject
_PyRun_AnyFileObject
malloc
tcp_recvmsg
inet_recvmsg
sock_recvmsg
__sys_recvfrom
__x64_sys_recvfrom
do_syscall_64
Legal
Privacy
Cookies
English (US)

PL/PYTHON FUNCTION

CALCULATING MOVING AVERAGE

pl/python_moving_avg.sql

```
CREATE OR REPLACE FUNCTION moving_average_py(value_list double precision[], window_size int)
RETURNS TABLE(avg double precision)
AS $$
    result = []
    sum = 0.0
    count = 0

    for i, value in enumerate(value_list):
        sum += value
        count += 1

        if count ⩾ window_size:
            avg = sum / window_size
            result.append(avg)

            sum -= value_list[i + 1 - window_size]

    return result
$$ LANGUAGE plpython3u;
```

PL/RUST FUNCTION

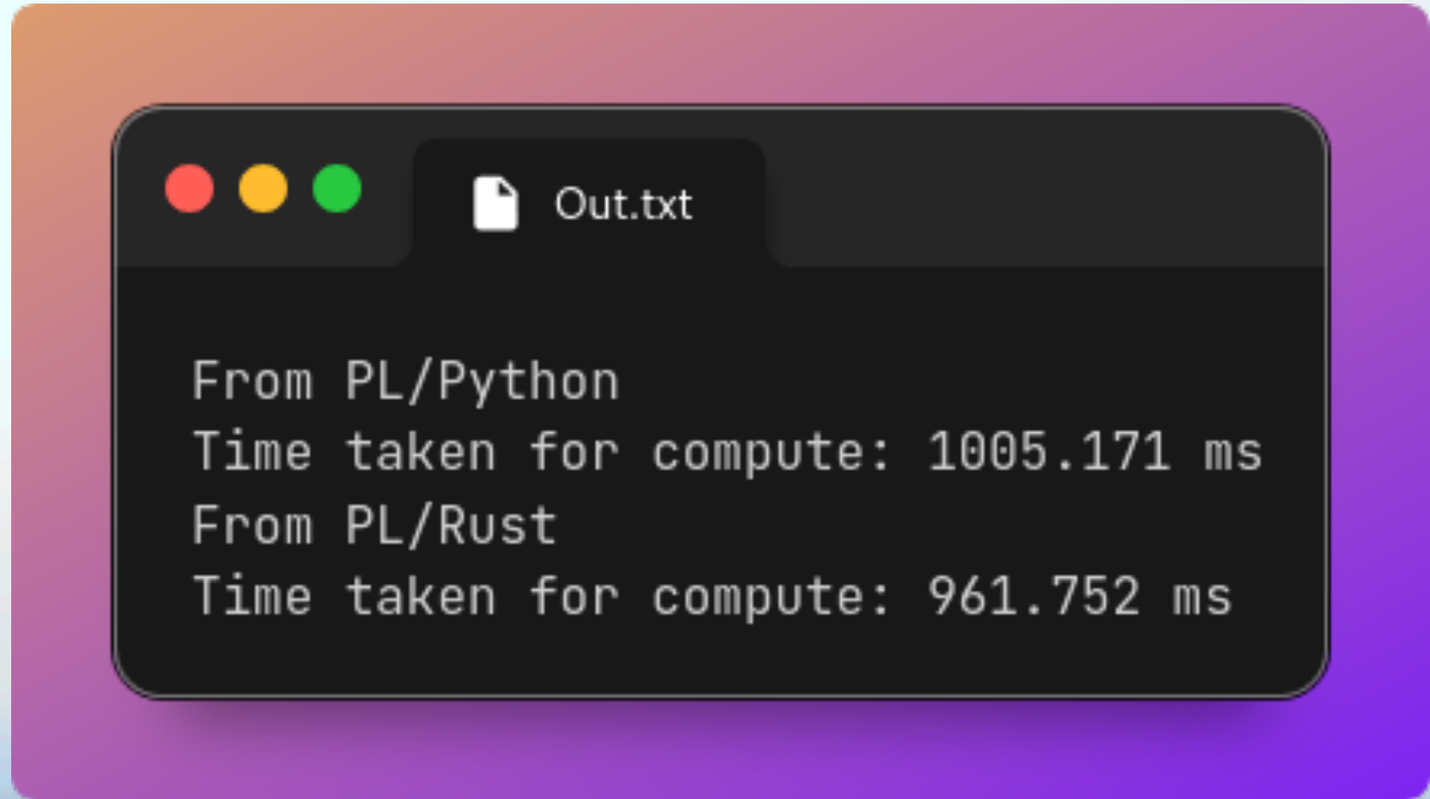
CALCULATING MOVING AVERAGE

pl/rust_moving_avg.rs

```
(myenv) root@ubuntudemo ~/pgexp/fastrx/db-bench-pl-latency/pl_rust_bemchmark # cat src/lib.rs
use pgrx::prelude::*;

#[cfg(feature = "pgrx_embed")]
::pgrx::pgrx_embed!();

::pgrx::pg_module_magic!();

#[pg_extern]
pub fn moving_average(values: Vec<f64>, window_size: i32) -> TableIterator<'static, (name!(avg, f64),)> {
    let mut result = Vec::new();
    let mut sum = 0.0;
    let mut count = 0;

    for (i, &value) in values.iter().enumerate() {
        sum += value;
        count += 1;

        if count >= window_size {
            let avg = sum / window_size as f64;
            result.push((avg,));
            sum -= values[i + 1 - window_size as usize]; // Remove the oldest value
        }
    }

    TableIterator::new(result.into_iter())
}
```

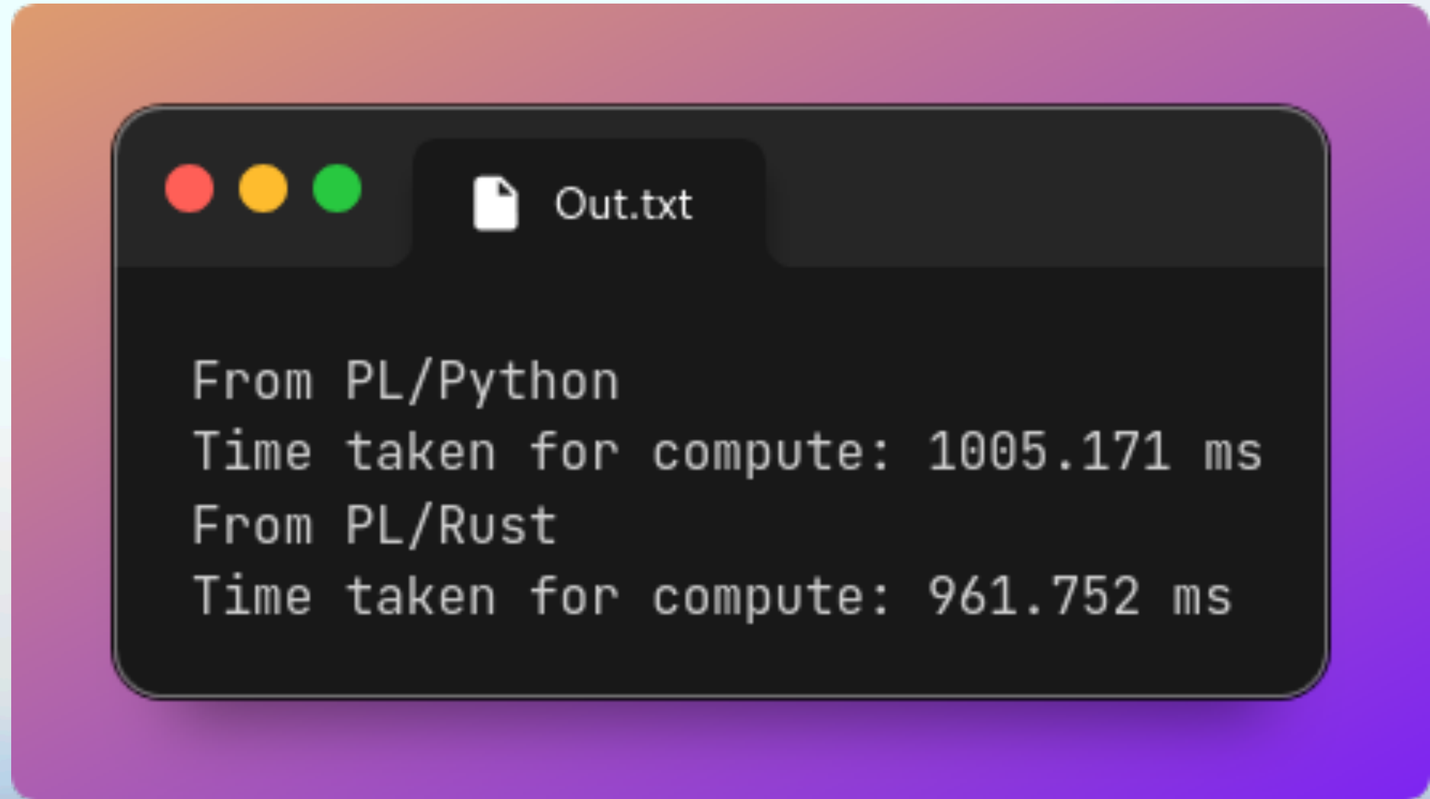
COMPARISION

For a simple moving average computation on just 1 million records, PL/Rust already outperforms PL/Python, taking **~961 ms vs. ~1005 ms**. Now, imagine this pattern repeating across millions or billions of records, with even more complex algorithms and larger datasets. The performance gap doesn't just add up—it **multiplies**.



COMPARISION

Choosing the right language at the database level can make your queries **X times faster**, leading to huge efficiency gains at scale.



CONCLUSION

Performance is a FEATURE

Treat Performance as any Black-Box Feature.

Develop metrics for continuously measuring Performance, and report these frequently across the org

Utilize profilers from the beginning of development.

Develop performance targets early, and if performance ever drops below key levels, then performance work becomes all-hands-on-deck.

Continuously allocate time alongside all feature development to work on performance improvement/tuning.

Never accept “we’ll fix performance at the end of the project”, as it never works out.

“Measure What Really Matters”

John Doerr

REFERENCES

Fearless Extension Development with Rust and PGRX - PGCONFEU 2024

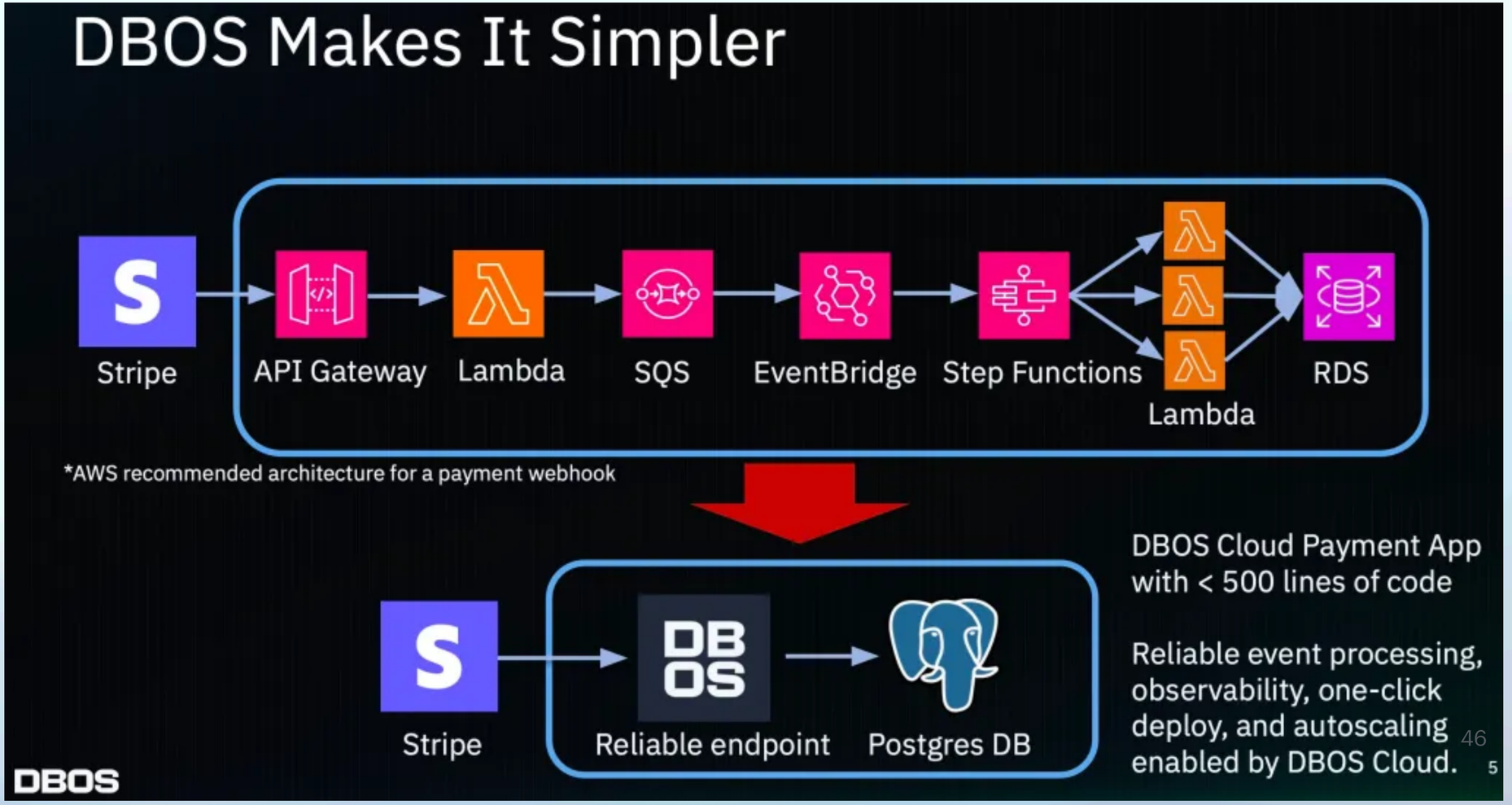
https://www.postgresql.eu/events/pgconfeu2024/sessions/session/5540/slides/584/PGRX%20PGConf.eu%20Athens%202024%20.pdf

GITHUB

https://github.com/postgresnx/fastrx/

MODERN PLATFORM

DATABASE AS OS




IBIZA.SYSTEMS
PGIBIZA.COM

MODERN PLATFORM

QUESTIONS