Postgres for everything, until it isn't



GJ **Gülçin Yıldırım Jelinek**
Senior Postgres PM | ClickHouse

PJ **Petr Jelinek**
Distinguished Engineer | EnterpriseDB (EDB)

PGConf.EU 2026 · Valencia · 20 October 2026

About us



Gülçin

- From Türkiye, lives in Czech Republic
- Organizes Prague PostgreSQL Meetup, 2017
- Previously Xata, EDB, 2ndQuadrant
- Working at ClickHouse Managed Postgres team
- Working with PostgreSQL since 2012
- First PGConf.EU talk in 2015 in Vienna
- PostgreSQL Europe board, 2017
- PostgreSQL contributor
- Co-founder, Kadın Yazılımcı (Women Devs), 2013
- Diva: Dive into AI, 5th Edition 2027
- **What's New with Monitoring in PostgreSQL 19** (Wed, 09:25 room SM1-A)
- Co-organizing **PostgreSQL Observability Summit**
- Co-organizing **PostgreSQL on Kubernetes Summit**

Petr

- From Czech Republic
- Working as Architect/Distinguished Engineer at EDB
- Architect of EDB Postgres Distributed
- Previously 2ndQuadrant, Microsoft, Skype, …
- Major PostgreSQL contributor - logical replication, tablesample, grant on all, default privileges, DO statement
- Original author of pglogical, ex-maintainer of pgbouncer
- First PGConf.EU talk in 2014 in Madrid

OPENING

About us



Who we are

We met at FOSDEM 2015, and we're married

We work for competitors: **ClickHouse** and **EDB**

Both employers sell a version of what this talk concludes

Why we're doing it together

Built-in peer review:

Neither of us can get away with a sales pitch while the other one is holding the second microphone.

House rules for the next 45 minutes

We're here as community members to share insights. We'll highlight various ecosystem solutions, not just our own companies' tech.

This talk isn't a vendor advertisement. All architectural examples come strictly from open-source Postgres and public sources. And there are no benchmarks.

Commit;





OPENING

The “Postgres for everything” era

Queue
SKIP LOCKED

Cache
UNLOGGED tables, materialized views

Documents
JSONB

Search
tsvector, pg_trgm

Vectors

Time series
Partitioning

Analytics

Geo

“Just use Postgres for everything” “Postgres is eating the database world” Every cloud now ships a Postgres

The “Postgres for everything” extended era



Queue

SKIP LOCKED, **pgmq**

Cache

UNLOGGED tables, materialized views

Documents

JSONB

Search

tsvector, pg_trgm, **pg_search**, **pg_textsearch**

Vectors

pgvector

Time series

Partitioning, **timescale**

Analytics

pg_clickhouse, **EDB PGAA**, **pg_lake, pg_duckdb**

Geo

PostGIS

Extensibility was a design goal from day one, so new capabilities arrive without forking Postgres or running a second system. Extensions use the same SQL, transactions and backups, ship on their own schedule, and the good ideas graduate into core.

Three questions

01 You run a Postgres database larger than 1 TB

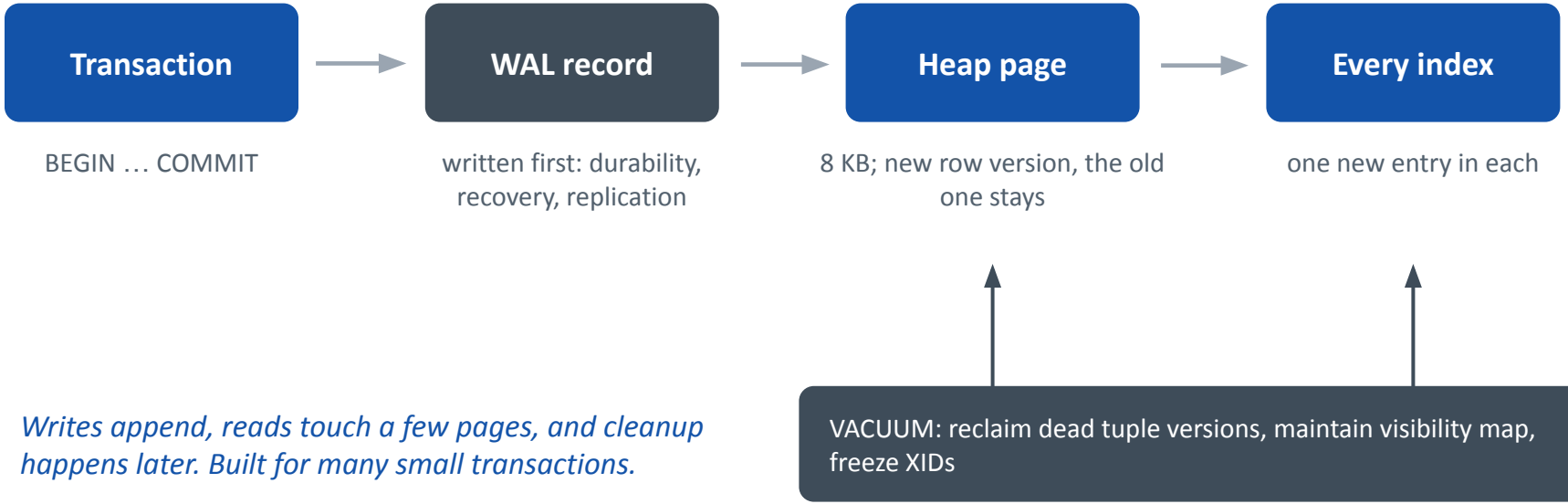
02 You already run a second system next to Postgres

03 You've been paged for database-related incidents



01 Why Postgres is so good at OLTP

The write path



Writes append, reads touch a few pages, and cleanup happens later. Built for many small transactions.

MVCC design



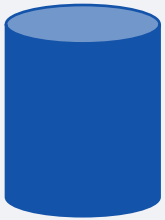
You get

- readers never block writers
- instant rollback
- no undo log to run out of

You defer

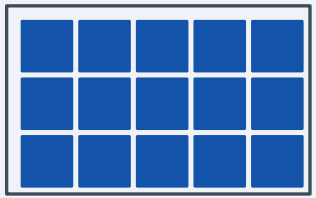
- dead versions, until vacuum
- an index entry per non-HOT write
- mandatory freezing

What the design assumes



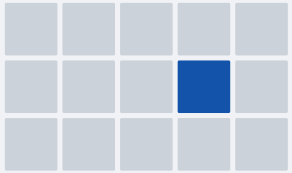
Local block storage

cheap 8 KB random I/O · fsync · same box



Working set fits in memory

shared_buffers and page cache hold what matters



Queries touch few rows

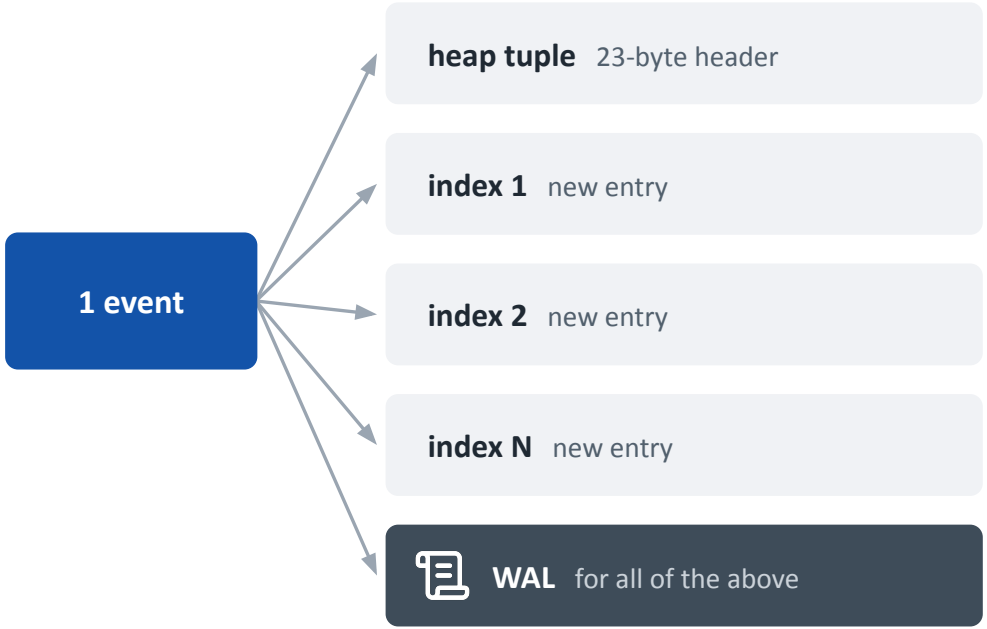
a small fraction of the table per statement

When all three hold, Postgres is close to unbeatable.

02

Three workloads it wasn't tuned for

Workload 1: Append-heavy telemetry



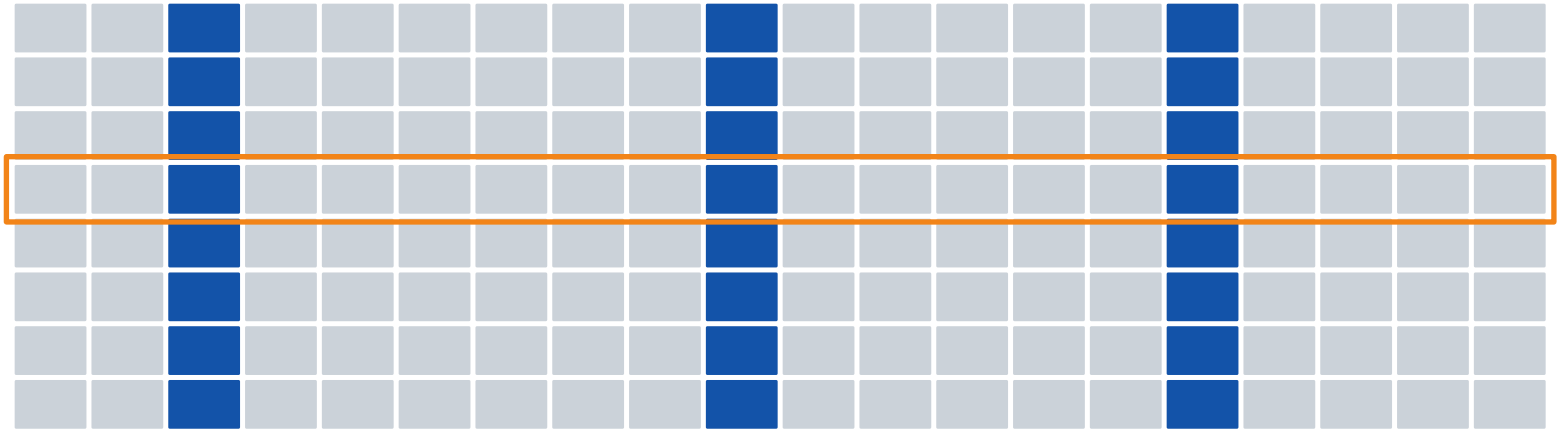
23 B tuple header before any data

1 + N writes per row: heap plus N indexes

2^{31} transaction IDs before wraparound

Workload 2: Large analytical scans

```
SELECT country, count(request_id), avg(latency_ms) FROM events GROUP BY country;
```



3 columns the query needs

37 more it reads anyway, page by page

Workload 3: Data that lives in object storage



What Postgres assumes

- 8 KB random reads and writes
- fsync for durability
- storage and compute on one box

≠



What object storage offers

- high latency, huge throughput
- immutable objects, no page writes
- open formats: Parquet, Iceberg

03 The optimization ladder

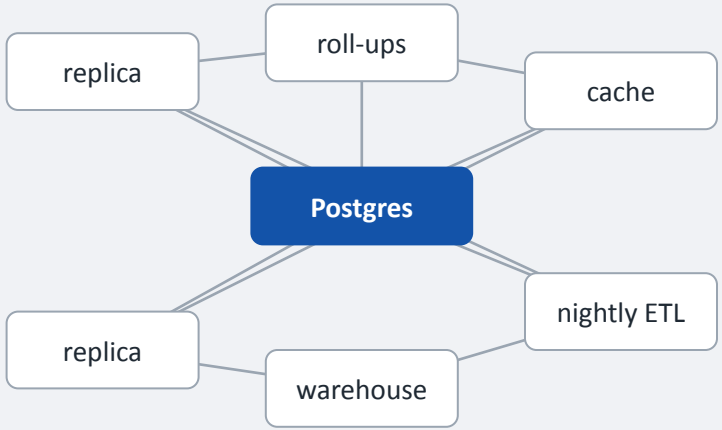
Seven reasonable steps

1	**More indexes**	write amplification
2	**Partitioning**	planning time, partition counts, maintenance jobs
3	**Read replicas**	lag, more hosts, still a row store
4	**Materialised views, roll-ups**	refresh windows, stale data, double writes
5	**Workload specific table access method**	one process model, one buffer cache, one vacuum
6	**Caching layer (Redis, app-side)**	invalidation, and a second system anyway
7	**Nightly ETL out**	hours of latency, brittle pipelines

a second system, already

Complexity cost

Inherited from the ladder



Chosen for the workload



The question was never whether to run more than one engine. It's whether you chose it.

Three public stories



Shard to stay

Figma 2024 · Notion 2021–23

kept the system of record on Postgres, sharded in app



Leave, loudly

Uber 2016

write amplification and replication, still debated



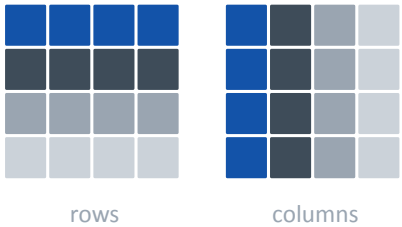
Pair by design

a common pattern

system of record plus a purpose-built engine, fed by CDC

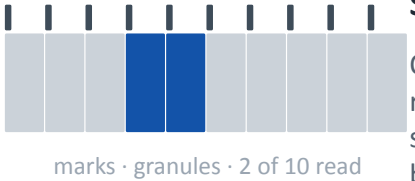
04 What a different execution model buys

Four mechanisms for optimizing analytics scans



Columnar storage

Read the 3 columns the query names, not 40. Similar values sit together, so 10× compression is routine.



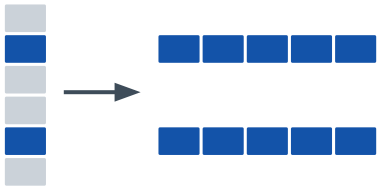
Sparse primary index

One mark per few thousand rows. A range filter on the sort key skips whole granules before reading a byte.



Vectorised execution

Operators work on blocks of values with SIMD instructions instead of interpreting one tuple at a time.



Late materialisation

Evaluate the filter on the compressed filter columns; fetch the remaining columns only for rows that survive.

Why you want to combine

Row store

owns

transactions · constraints · point reads and writes · schema of record

bad at

- wide scans
- compression
- object storage
- long retention

+

Columnar engine

owns

aggregations · time ranges · wide tables · long retention

bad at

- transactions
- point lookups
- updates, deletes
- small inserts

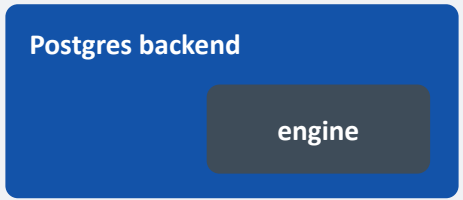
The same lesson, three more times

Event streams	append-only log · sequential I/O · consumer offsets	
Full-text search	inverted index · BM25 · segment merging	*Lucene, Tantivy*
Graph	adjacency traversal instead of join explosion	*graph engines*
Analytics	column files · sparse index · vectorised execution	*columnar engines*

05 How the Postgres ecosystem responded

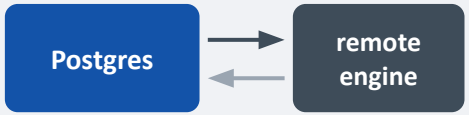
Four patterns to bring another engine to Postgres

Embed the engine in the backend



pg_duckdb, pg_search

Push down to a remote engine



pg_clickhouse, EDB Analytics Accelerator

query out, results back

Replicate out (CDC)

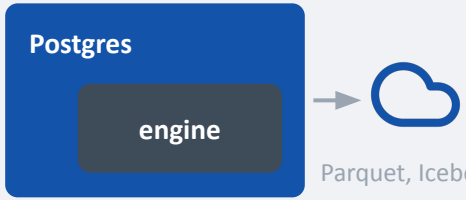


Debezium, logical replication, pgstream

ClickPipes, EDB Analytics Accelerator

logical decoding to any target

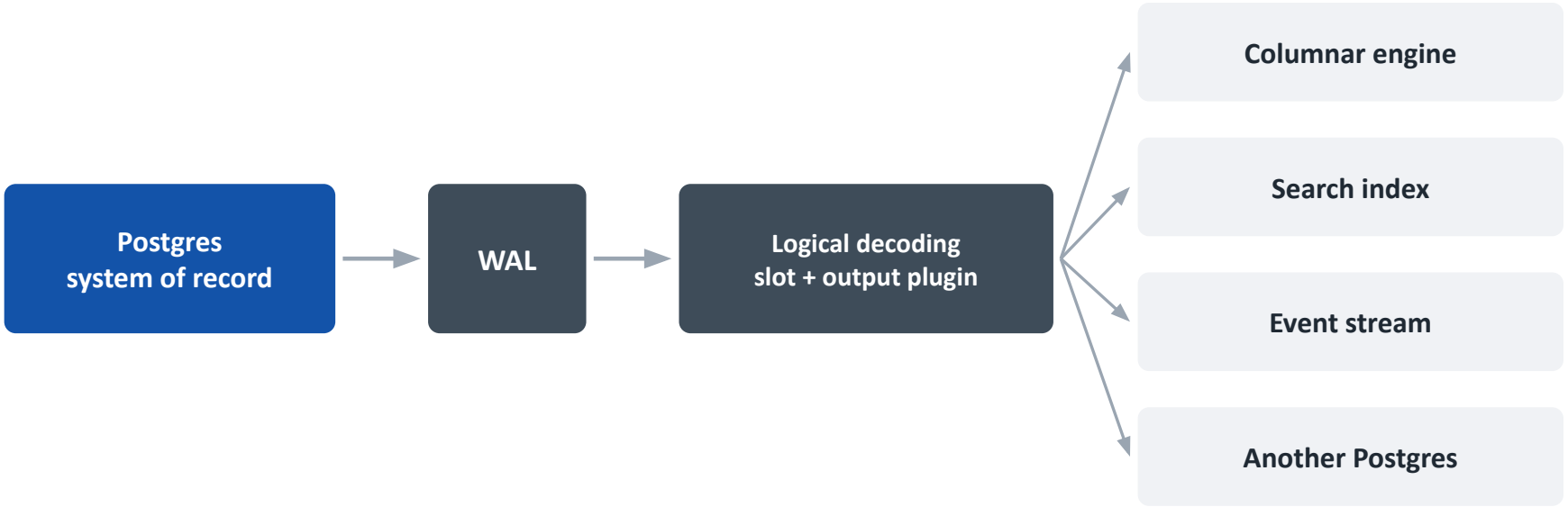
Query lake formats from Postgres



EDB Analytics Accelerator, chdb, pg_duckdb, pg_lake

Parquet, Iceberg

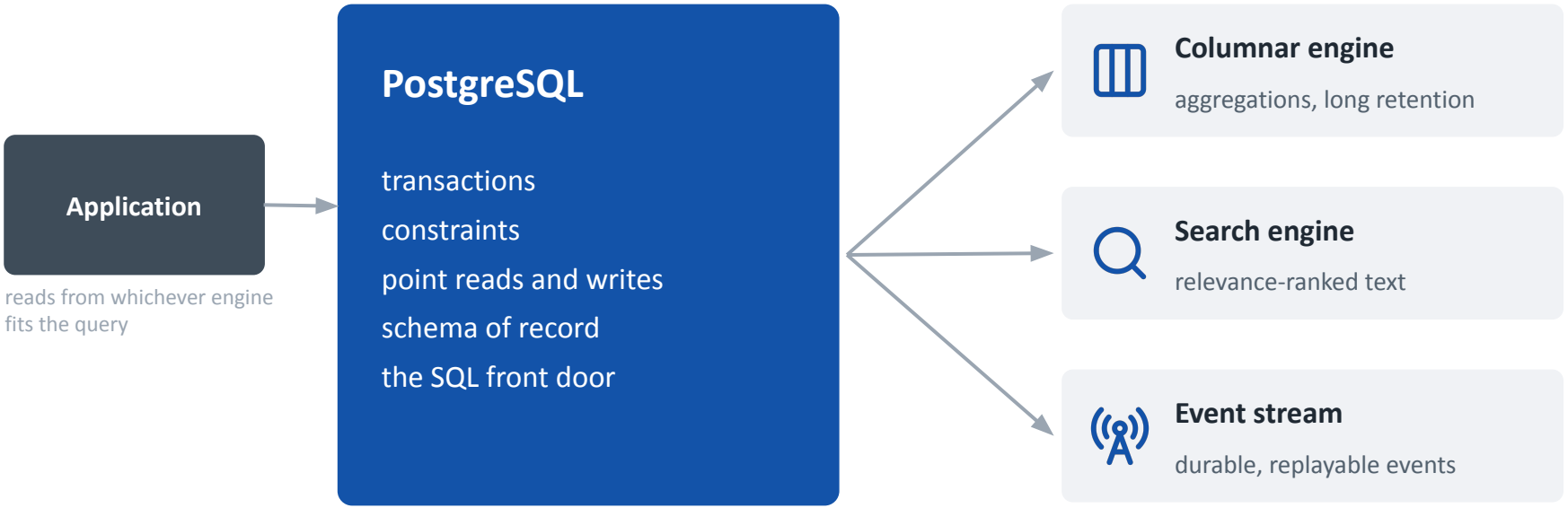
Logical decoding was designed for exactly this



A slot guarantees the subscriber loses nothing. The price is WAL retention on the primary.

06 Intentional composition

Postgres as the transactional control plane



Offloading the scans is the most effective thing you can do for the primary.

What two systems actually cost

Two things to operate

monitoring, backups, upgrades, on-call

Consistency lag

seconds to minutes behind

Schema in two places

renames break pipelines

Someone owns the pipeline

slots, snapshots, backfills

Pushdown coverage

shipping rows is slower than staying

Data stored twice

by design; measure, don't assume

When it's time, and when it isn't

Time to compose

- analytics on the primary hurts OLTP p99
- a table that only grows, only aggregated
- vacuum or wraparound is a recurring ticket
- replicas multiplying for reporting
- event storage is a line item
- dashboards time out despite every optimization

Not yet

- under a few hundred GB, growing slowly
- analytical queries are selective
- one team, one on-call rotation
- nobody is paged for the database
- you can't name the second workload

07

How Postgres core responded

Core keeps moving the boundary

steady logical replication improvements, incremental backup, every release

10	12	18	19	20+
Logical replication	**Table access method API**	**Asynchronous I/O**	**REPACK / pg_plan_advice**	**SQL/PGQ / 64-bit XIDs**
CDC	pluggable storage	io_uring; faster sequential and bitmap scans	online VACUUM FULL plan stability	Graph databases No XID wraparound

Each release moves the boundary outward. None changes the shape of the row store, and none needs to: it is why Postgres is the system of record.

08 Postgres in the brave new world

SQL is an agent interface



MCP

Tool protocol

Discover and invoke tools
Structured inputs and outputs



OpenAPI

HTTP APIs

Describe HTTP APIs
Operations, parameters, responses



CLI

Shell commands

Run commands
Tool-specific syntax and output



SQL

The database

Express data operations
Queryable schema, plans and errors

Nov 19: [code::dive 2026](#) Wrocław, Poland · Nov 24: [PGConf.PL](#) Gdańsk, Poland

Architectural choices

“Just use Postgres” is still the right default. Know where its boundaries are, why they're there, and compose around it on purpose when you reach them.



GJ **Gülçin Yıldırım Jelinek**
Senior Postgres PM | ClickHouse

PJ **Petr Jelinek**
Distinguished Engineer | EnterpriseDB (EDB)

Questions?