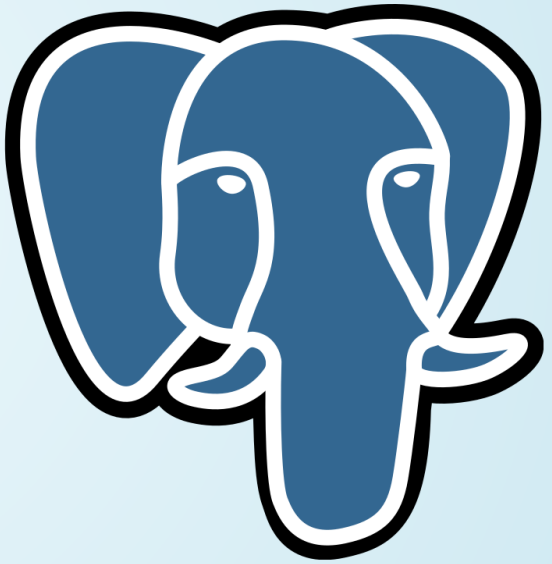
When Postgres Is Enough

Eugen Geist



AGENDA

- **Why Postgres?**
- **Publish/Subscribe**
- **Distributed Queues**
- **Document Storage**
- **Bonus: Vectors**

ABOUT ME

- Software und Data Engineering Freelancer
- Contact
  - eugengeist.com
  - mail@eugengeist.com
  - https://linkedin.com/in/eugen-geist/

WHY THIS TALK?

WE LOVE NEW SHINY THINGS

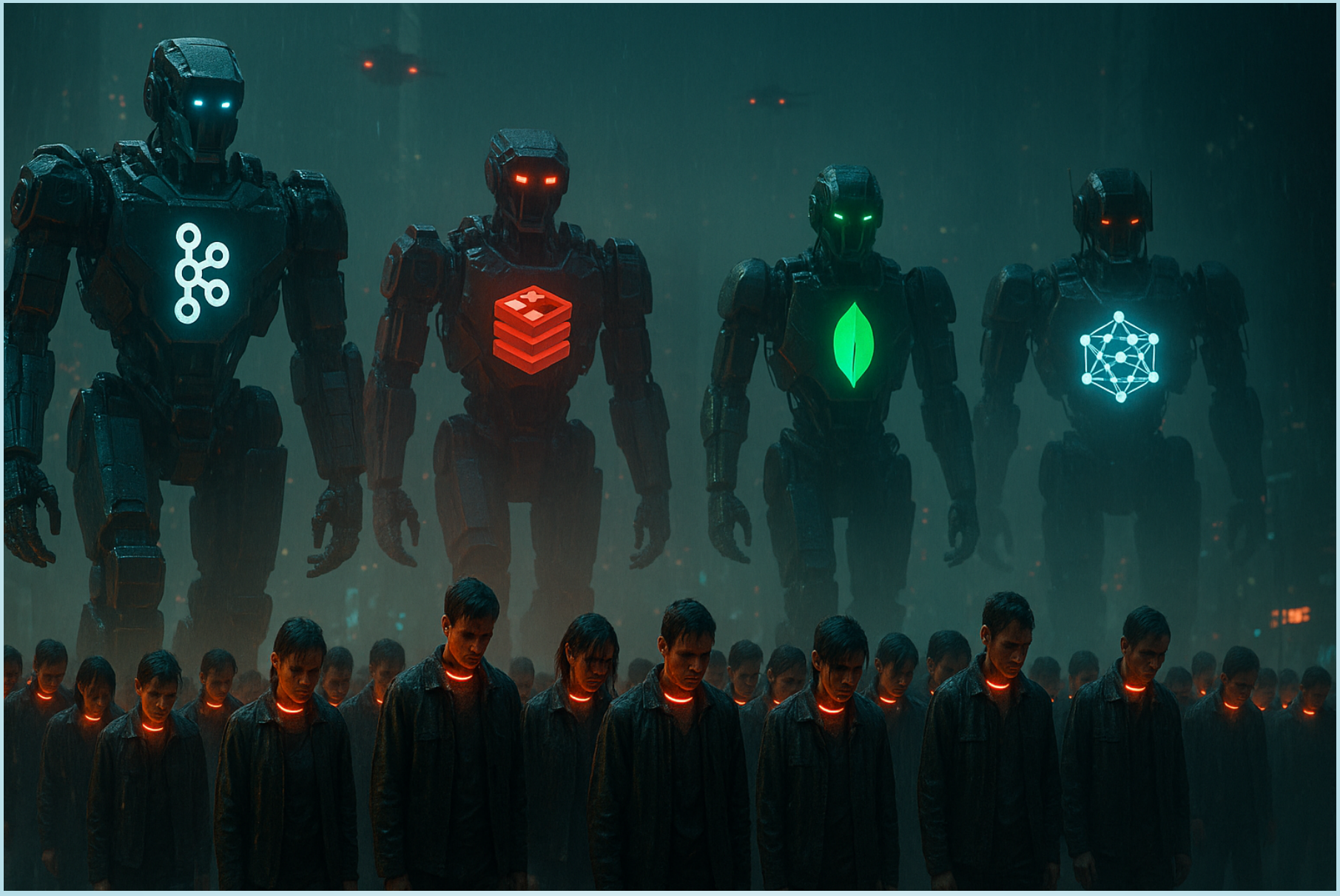
As soon as a new problem occurs, we tend to search the perfect solution for it

WE LOVE COMPLEXITY

Complexity is interesting, you can lose yourself in it


redis


MAINTAIN
UPTIME



INSTEAD OF SOLVING REAL PROBLEMS, WE GET LOST IN TECHNOLOGY

Instead of introducing the *right tool* for the new job, we can get pretty far with the *existing tool*

Every tool has it's purpose, but it also has to be introduced and maintained ➡️ costs money and manpower

Most teams don't need the featureset of Kafka or MongoDB.

Especially not small teams.

Keep it simple and make pragmatic choices, instead of interesting ones.

Boring Technology Club

You are not Google

... and if it turns out, that you need the *right tool* you can still introduce it, as soon as needed!

WHY POSTGRES?

POSTGRES IS FREE

free as in

free beer: you don't pay anything to use it!

AND

freedom: it's open source!

POSTGRES IS CONTINOUSLY DEVELOPED

started in 1996

still gets new features in 2025 and is continously improved

over 62k commits

POSTGRES HAS A LOT OF FEATURES + IS EXTENSIBLE

HOW MANY OF YOU USE(D) POSTGRES?

POSTGRES IS EVERYWHERE

one of top 5 used database engines

you probably have used postgres directly or via proxy (a product) in your life

client libraries for most programming languages

PUBLISH/SUBSCRIBE

PROBLEM

YOU NEED TO NOTIFY ONE APPLICATION FROM ANOTHER

e.g. about added/changed database rows


Stream
at scale
Kafka
Kafka

STOP

You should consider

POSTGRES NOTIFY/LISTEN

HOW DOES IT WORK?

database session *1* runs `LISTEN channel`

database session *2* executes `NOTIFY channel, id123`

➡️ session *1* receives the payload *id123* from session *2*

LISTEN/NOTIFY DEMO

- **every currently-listening session** is informed on **every** `NOTIFY`
- notifications cannot be re-read or re-delivered, they are **ephemeral**
- notifications are sent when the notifier commits
- listeners only receive notifications between transactions
- instead of `NOTIFY ...`, `pg_notify(channel, payload)` can also be used
- payload must be string
- payload in default configuration must be shorter than 8000 bytes
- notifications with same payload to one channel in one transaction are only delivered **once**
- sessions can unsubscribe from a channel by
  - running `UNLISTEN`
  - ending the session
- internal queue holds all notifications that were not processed by listeners yet
  - default-size: 8GB
  - if queue becomes full, transactions calling `NOTIFY` will **fail at commit**

YOU SHOULD USE *LISTEN/NOTIFY*, IF YOU

- have events with *small* payload e.g. IDs
- have small to mid data volume
- don't need to re-process missed events
- don't need redundancy ➡️ LISTEN/NOTIFY only possible on primary postgres servers
- don't need control over who is allowed to LISTEN and NOTIFY ➡️ no access rights available

DISTRIBUTED QUEUES

PROBLEM

YOU NEED TO DECOUPLE WORKFLOWS WITH A DISTRIBUTED QUEUE


Queue
reliab
Redis
Distributed
queues

STOP

You should consider

POSTGRES

SELECT ... FOR UPDATE SKIP LOCKED

HOW DOES IT WORK?

`SELECT … FOR UPDATE SKIP LOCKED` grabs row-level locks on the rows your query returns

but silently skips any rows that are already locked by another transaction

SELECT … FOR UPDATE SKIP LOCKED DEMO

Different waiting mechanisms in Postgres when selecting with lock

- `SELECT FOR UPDATE` ➡️ waits for locked rows
- `SELECT FOR UPDATE ... NOWAIT` ➡️ errors if it hits a locked row
- `SELECT FOR UPDATE ... SKIP LOCKED` ➡️ ignores locked rows and keeps going

➡️

`SELECT FOR UPDATE ... SKIP LOCKED` is perfect for *concurrent job pickers* where many workers select the next row without stepping on each other

- use SKIP LOCKED when multiple workers are contending for the same *next items*
- always ORDER BY something deterministic, e.g. priority, then run_at, then id
- always `LIMIT` to avoid scanning huge sets
- don't do long work while holding the row lock
  - long transactions bloat and block vacuum
  - use a lease column
- idempotency matters ➡️ crashes can cause reprocessing
- make sure queries on queue tables are fast
  - use (partial) indices for queries
  - archive old entries, that are finished
- batching can reduce database round trips, but also lowers parallelism as one worker holds many rows
- reduce polling of table by using NOTIFY/LISTEN to be notified about new entries

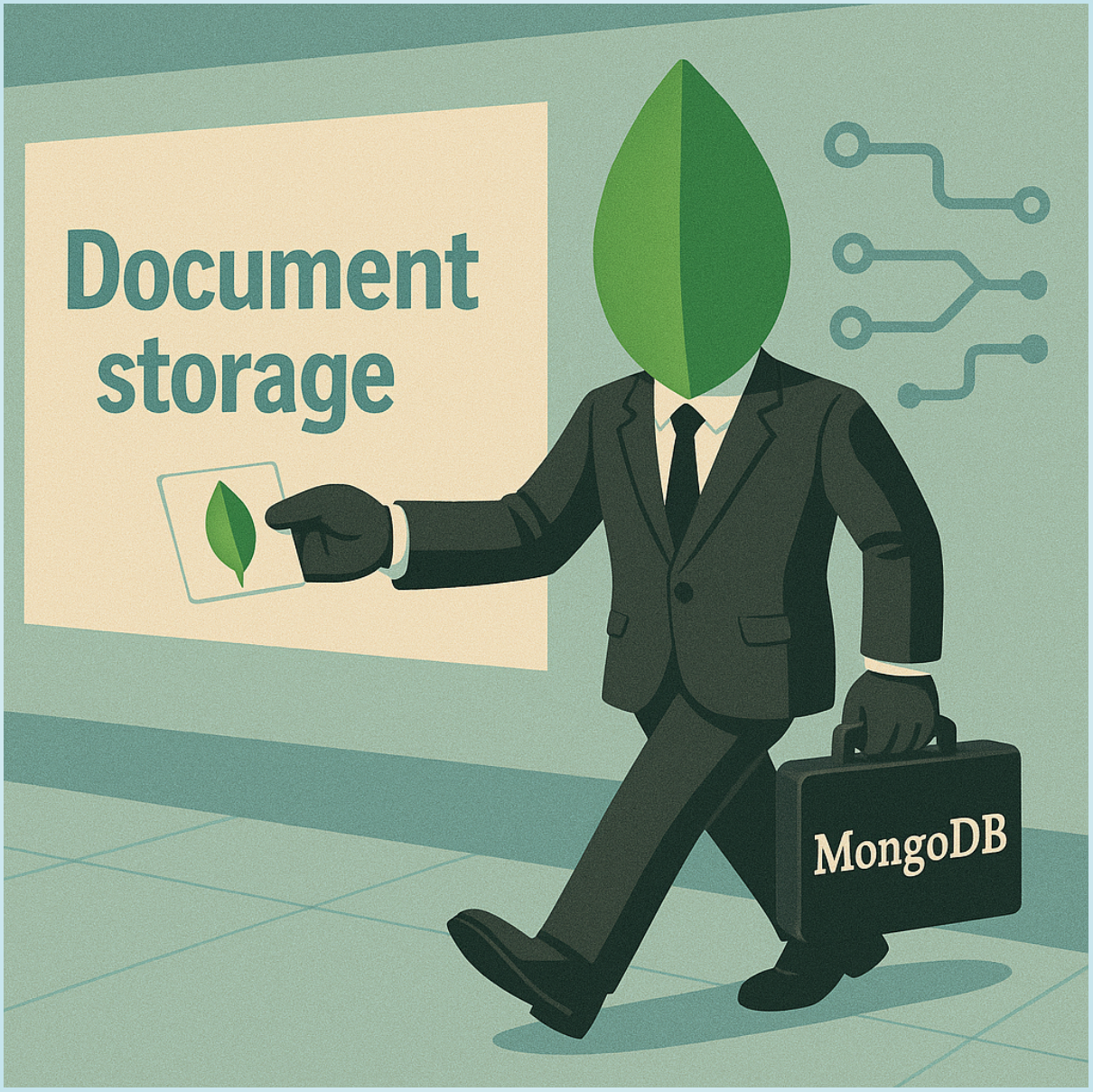
YOU SHOULD USE *FOR UPDATE SKIP LOCKED*, IF YOU

- want to concurrently process entries from a table/queue without taking care of locking yourself
- need persistency of items to process
- don't need
  - FIFO
  - per-key fairness
  - sophisticated queue patterns
- run at moderate scale ➡️ thousands to tens of thousands jobs per minute

DOCUMENT STORAGE

PROBLEM

YOU NEED TO STORE SCHEMALESS OR DATA WITH A VARYING SCHEMA


Document storage
MongoDB

STOP

You should consider

POSTGRES

JSONB

HOW DOES IT WORK?

`JSONB` columns allow storing objects with varying fields

`JSONB` columns' keys and values can be indexed and used for selection and projection

JSONB DEMO

- Postgres provides two types of columns for JSON
  - **`JSON`**: stored as raw text ➡️ Postgres validates it is valid JSON on insert but does not parse or index the structure
  - **`JSONB`**: stored in a binary, decomposed form ➡️ ignores key order and removes duplicate keys
  - ➡️ for most use-cases **you want `JSONB`**
- `JSONB` operators allow retrieval and checking of nested keys/values
- `JSONB` columns' keys/values can be indexed
  - GIN indices allow fast containment queries ➡️ does a rows' `JSONB` contain a specific key?
  - Btree indices allow fast value retrieval ➡️ does a rows' `JSONB` key contain a specific value?
  - Partial indices are also possible, if many documents don't have a key

- `CHECK` constraints can be defined on `JSONB` columns to enforce certain rules and/or format
- large `JSONB` values will be TOASTed and compressed, but max size for one field is 1GB
- any update creates a new row version ➡️ even changing one nested key rewrites the whole JSONB value and row ➡️ impact on bloat and VACUUM
- regular column values can be generated on insert from `JSONB` key/values

YOU SHOULD USE *JSONB*, IF YOU

- want to combine the advantages of a relational database with document storage
  - use regular columns for always present data
  - use `JSONB` columns for varying data
- have data with a lot of varying fields
- don't need to update the JSONB fields very frequently

BONUS

VECTOR STORAGE

PROBLEM

YOU NEED TO STORE VECTORS AND EXECUTE OPERATIONS ON THEM, E.G. NEAREST SEARCH AND CALCULATIONS


vector
operations
Weaviate
Weaviate

STOP

You should consider

POSTGRES

WITH A VECTOR EXTENSION

HOW DOES IT WORK?

install a postgres vector extension, e.g. vectorchord

create and query tables with vectors stored in columns

VECTOR DATA DEMO WITH VECTORCHORD

- vectorchord introduces vector column types, indexes and operators
  - column types `vector`, `halfvec`, `sparsevec`, `bit`
  - operators for ordering
    - `<->`: squared Euclidean distance
    - `<#>`: negative dot product
    - `<=>`: cosine distance
  - operators for selection are different to push the filter down to the vector index
  - index `vchordrq` divides vectors into lists and searches only a subset of lists closest to the query vecto
- indexes configurable depending on vector size and used operators
- usually vector search is done with `ORDER BY` and `LIMIT`, for `WHERE` different operators must be used

YOU SHOULD USE *POSTGRES WITH A VECTOR EXTENSION*, IF YOU

- are just starting out with vectors
- need to keep vectors beside your relational data
- don't need extensive vector functionality
- have vectors in the millions rather than in the billions

HIDDEN BONUS

- Postgres full-text search
- PostGIS extension for geospatial data
- Citus extension for distributed Postgres sharding and replication

and many more...

We can get away most of the times with a lot less, than we think...

...but of course, if you need them, use the proper tools to solve your problems

SOURCES

- LISTEN/NOTIFY
  - https://www.postgresql.org/docs/current/sql-notify.html
  - https://www.postgresql.org/docs/current/sql-listen.html
- Queues
  - https://www.inferable.ai/blog/posts/postgres-skip-locked
  - https://www.postgresql.org/docs/current/sql-select.html
- Document storage JSONB: https://www.postgresql.org/docs/current/datatype-json.html
- Vectors: https://docs.vectorchord.ai
- Full-text search: https://www.postgresql.org/docs/current/textsearch.html
- PostGIS: https://postgis.net/
- Citus: https://www.citusdata.com/
- DB ranking: https://db-engines.com/en/ranking
- Images: AI generated

Thank you!

Questions?

Slides + Demos:

- eugengeist.com
- mail@eugengeist.com
- https://linkedin.com/in/eugen-geist/