EDB
Postgres® for the AI Generation

# Incremental Backup in PostgreSQL 17

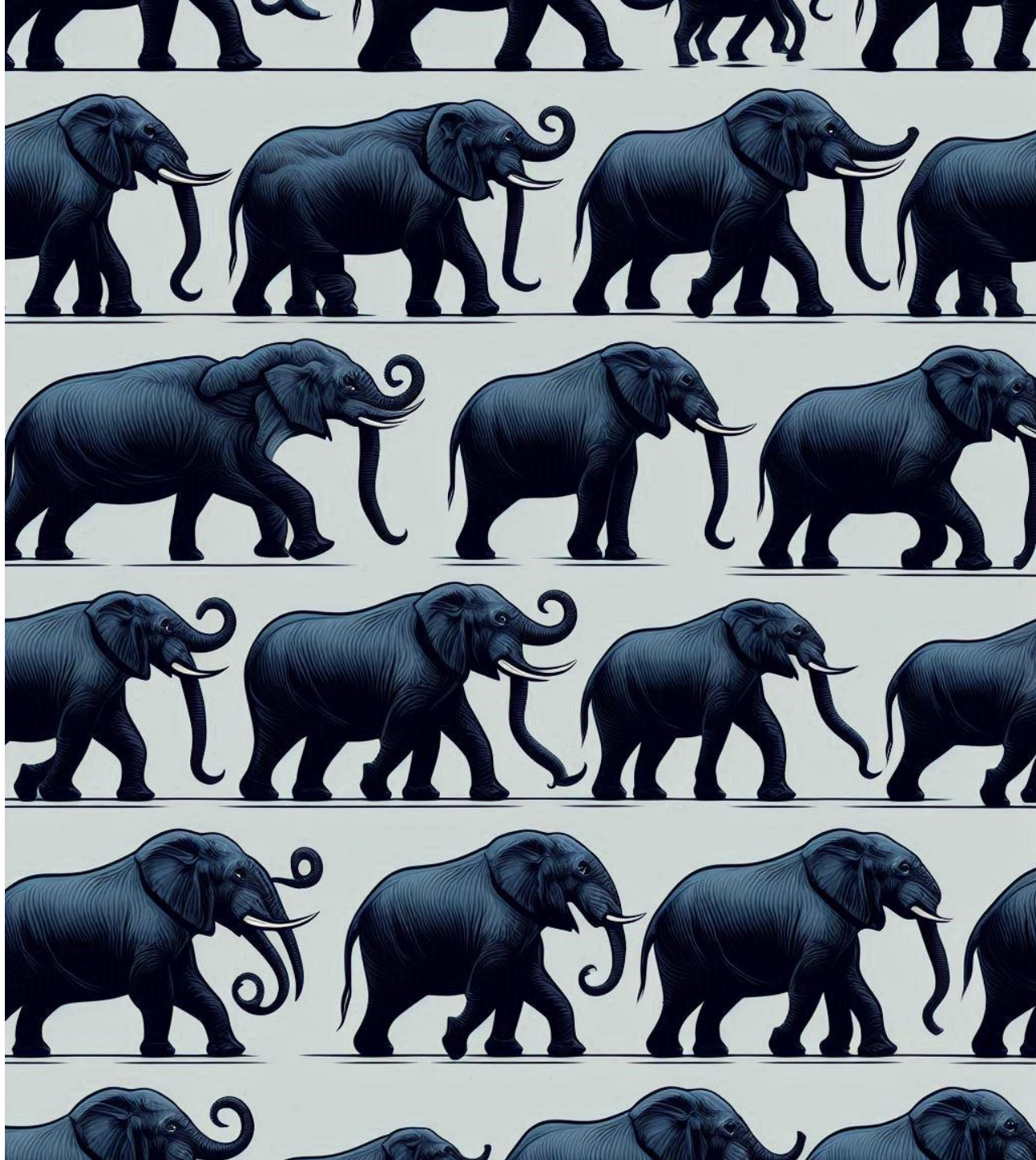
Robert Haas
VP, Chief Architect, Database Servers
POSETTE 2025 | June 10, 2025

# Overview & Motivation

- Backing up large databases can be very challenging, with backup times sometimes exceeding one day.
- Instead of copying the entire database when we make a backup, let's try to copy only the portions of the database that have changed. By doing this, we can make backups *smaller* and therefore *faster*.
- Various out-of-core tools and proprietary forks of PostgreSQL have provided features like this in the past, but we have not had anything in PostgreSQL itself.
- *Disclaimer*: This is still a new feature and, like any code, it might have bugs. Please try it, but please be careful!

EDB
Postgres® for the AI Generation

# Taking an Incremental Backup



# Basic Usage

- In `postgresql.conf` (or using `ALTER SYSTEM`), set `summarize_wal = on`.
  - Reload the configuration (or restart the server).

- `pg_basebackup -c fast -D sunday`
  - Full backup.
  - Use `-c fast` for testing to speed it up, but maybe not on a production system.

- `pg_basebackup -c fast -D monday --incremental sunday/backup_manifest`
  - Incremental backup based on Sunday's full backup.

- `pg_basebackup -c fast -D tuesday --incremental sunday/backup_manifest`
  `pg_basebackup -c fast -D tuesday --incremental monday/backup_manifest`
  - Incremental backup based on either on Sunday's full backup or Monday's incremental backup.

# Tooling

- Writing your own backup scripts is a bad idea!
- Use a quality backup tool that supports whatever you want to do.


EDB
Postgres® for the AI Generation

# Architecture

# Knowing What Has Changed: Requirements

- *Accurate*. If we think something has not changed when actually it did, then we will not include it in the backup and our data will be lost. If we think something has changed when it didn't really, that will not break anything but our backups will be larger.
- *Efficient*. It should be possible to determine what has changed without much effort.
- *Easy to Implement*. Reuse as much existing code as possible so that we don't have to write and debug too much new code.
- *Not Reliant on OS Features*. Especially, I like to avoid relying on things that work differently on different operating systems. Also, if something is entirely internal to PostgreSQL, it's easier to debug problems than if some of it is controlled at the OS level.

# The Write-Ahead Log To The Rescue!

- PostgreSQL's write-ahead log contains all the information about which blocks have been modified.
- It's already used for many other purposes and has existing debugging tools like `pg_waldump` and `pg_walinspect`.
- However, the write-ahead log is very big, so we can't use it directly.
- Instead, we add a new WAL summarizer process which will read the WAL as it's generated and produce small WAL summary files containing only the information that is required for incremental backup.
- Together with the backup manifest, this gives us everything we need in order to identify changed blocks and files.

EDB
Postgres® for the AI Generation

# Restoring an Incremental Backup

# Using pg_combinebackup

- Consider this example again:

```
pg_basebackup -c fast -D sunday
pg_basebackup -c fast -D monday --incremental sunday/backup_manifest
pg_basebackup -c fast -D tuesday --incremental monday/backup_manifest
```

- Everything that has changed between Monday and Tuesday is in the `tuesday` backup.
- Everything that has changed between Sunday and Monday is in the `monday` backup.
- Everything else is in the `sunday` backup.
- So we will need all three backups in order to restore:

```
pg_combinebackup sunday monday tuesday -o tuesday_full
```

# Recovery Is Still Required!

- The output of `pg_combinebackup` is a full backup.
- When you start `postgres` on any full backup whatsoever, database recovery is required.
  - If the required WAL is present in the backup's `pg_wal` directory, then you can just start the server and it will perform recovery as normal.
  - Otherwise, you need to create `recovery.signal` or `standby.signal` and set `primary_conninfo` and/or `restore_command` just as you normally would.
- Incremental backup does not let you skip any step that would otherwise be required.
- Again, it's a good idea to leave this orchestration up to a well-written backup tool!

# Summary

- `summarize_wal = on` runs the new WAL summarizer process, which will read the write-ahead log and write small WAL summary files
- `pg_basebackup --incremental $BACKUP_MANIFEST` uses the data in the backup manifest and the WAL summary files to create an incremental backup.
- `pg_combinebackup` reconstructs a synthetic full backup. Recovery is still required!
- Backup catalog, backup retention, etc. are out of scope for this feature; use external tools.

EDB
Postgres® for the AI Generation

# Thank you!

Any questions?