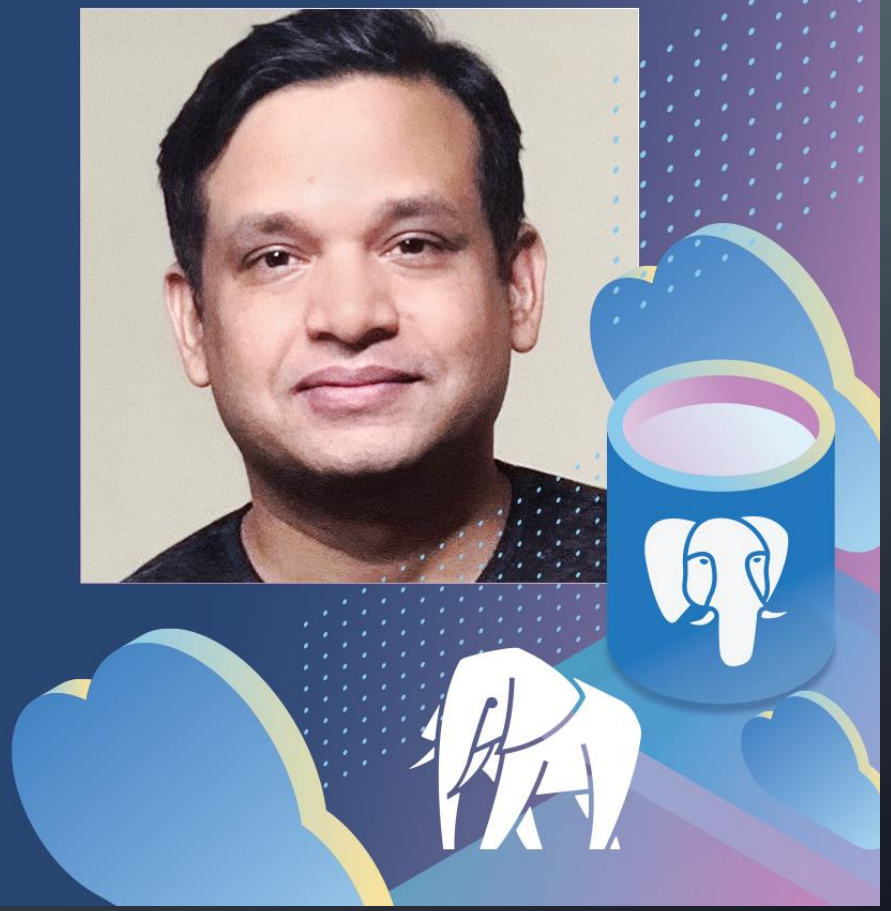
Microsoft | PGConf India

PGConf India, 2025

Postgres: ServerLESS is more?

Nikhil Sontakke

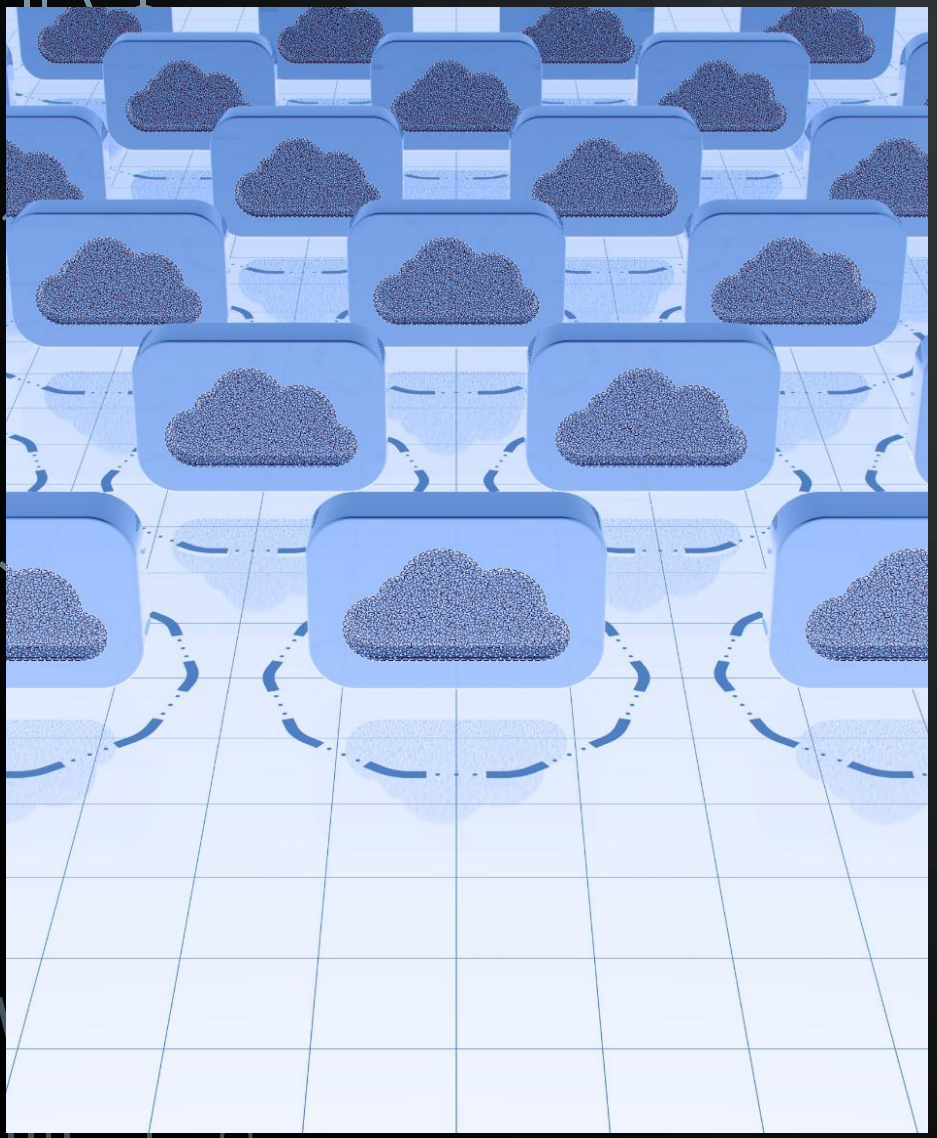
Fri 7 Mar | 11:30 IST



AGENDA OVERVIEW

- Introduction to Serverless Computing
- Why We Need Serverless in Cloud Environments
- Requirements of a Serverless Database in Cloud Environments
- Evolving PostgreSQL Into a Serverless Database
- Technical Components for Serverless PostgreSQL


INTRODUCTION TO
SERVERLESS
COMPUTING



WHAT IS SERVERLESS COMPUTING?

Definition

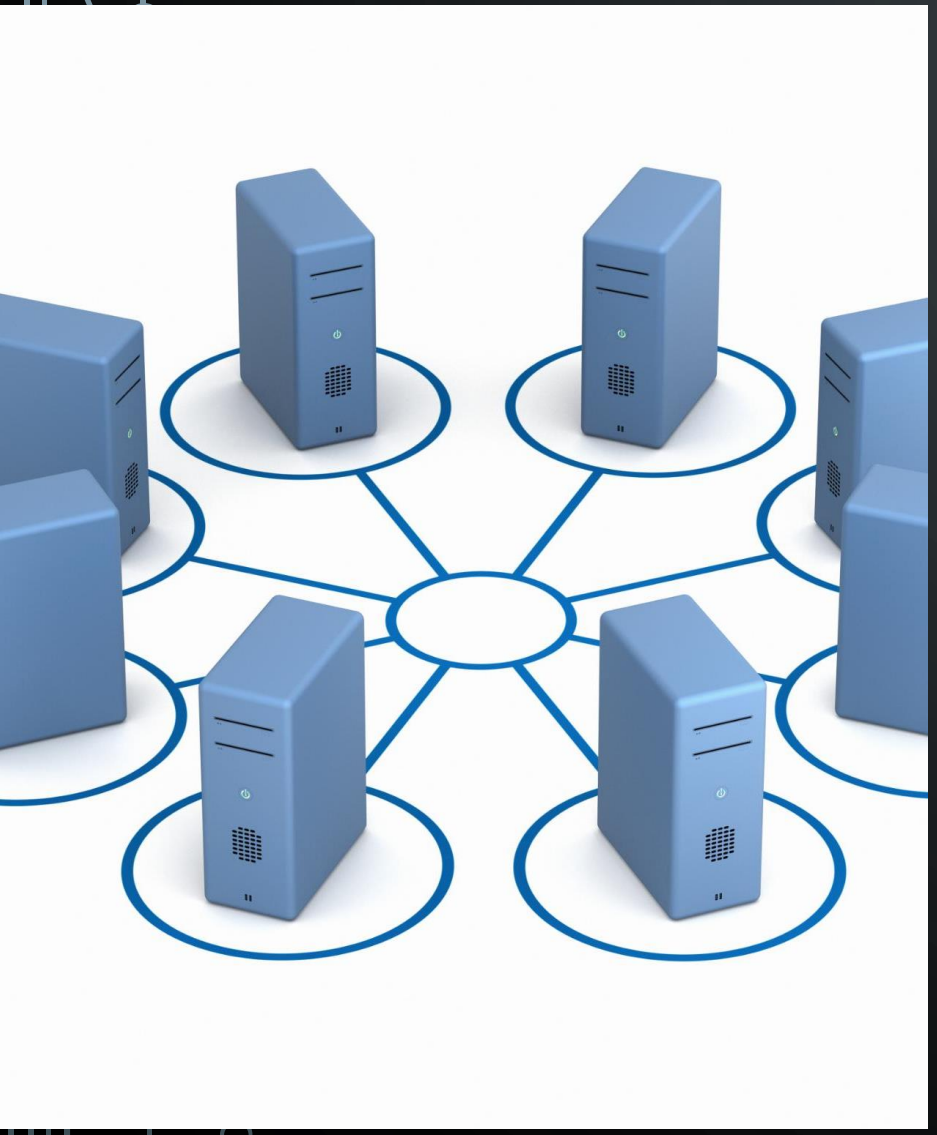
Serverless computing is a cloud computing model that allows developers to run applications without managing servers.

Focus on Development

In serverless computing, developers can concentrate on writing code and developing business logic instead of managing infrastructure.

Cloud Provider Responsibilities

The cloud provider manages the underlying infrastructure, such as servers and storage, allowing for scalability and flexibility.



HOW SERVERLESS WORKS

Function-based Architecture

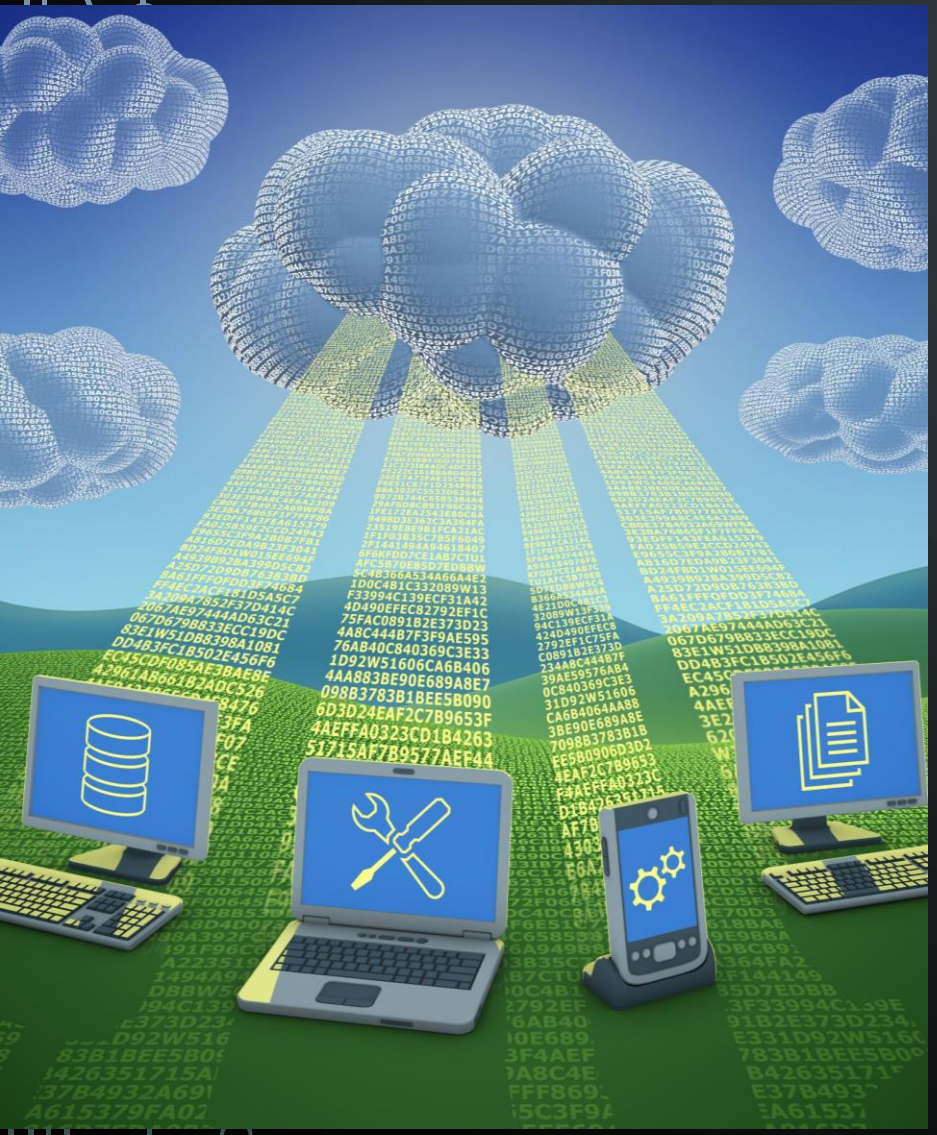
In a serverless model, applications are broken down into individual functions that respond to specific events.

On-demand Execution

Functions in a serverless environment are executed only when required, optimizing resource usage and costs.

Automatic Scaling

Resources are automatically provisioned and scaled based on the demand of the functions, ensuring efficiency.



BENEFITS OF GOING SERVERLESS

Increased Agility

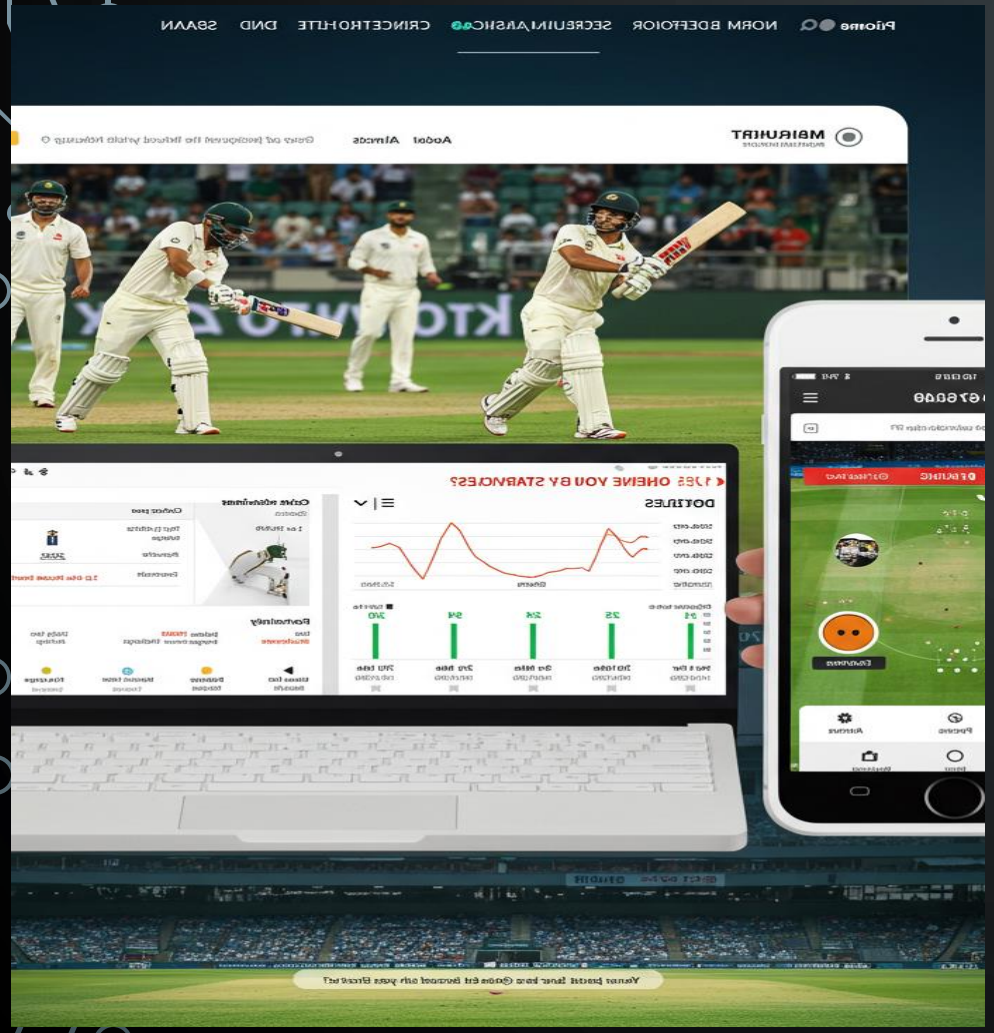
Serverless architecture enables faster development cycles and quicker deployment of applications, enhancing overall agility in software development.

Reduced Operational Burden

By eliminating the need for managing infrastructure, serverless solutions reduce the operational workload on development teams.

Cost Savings

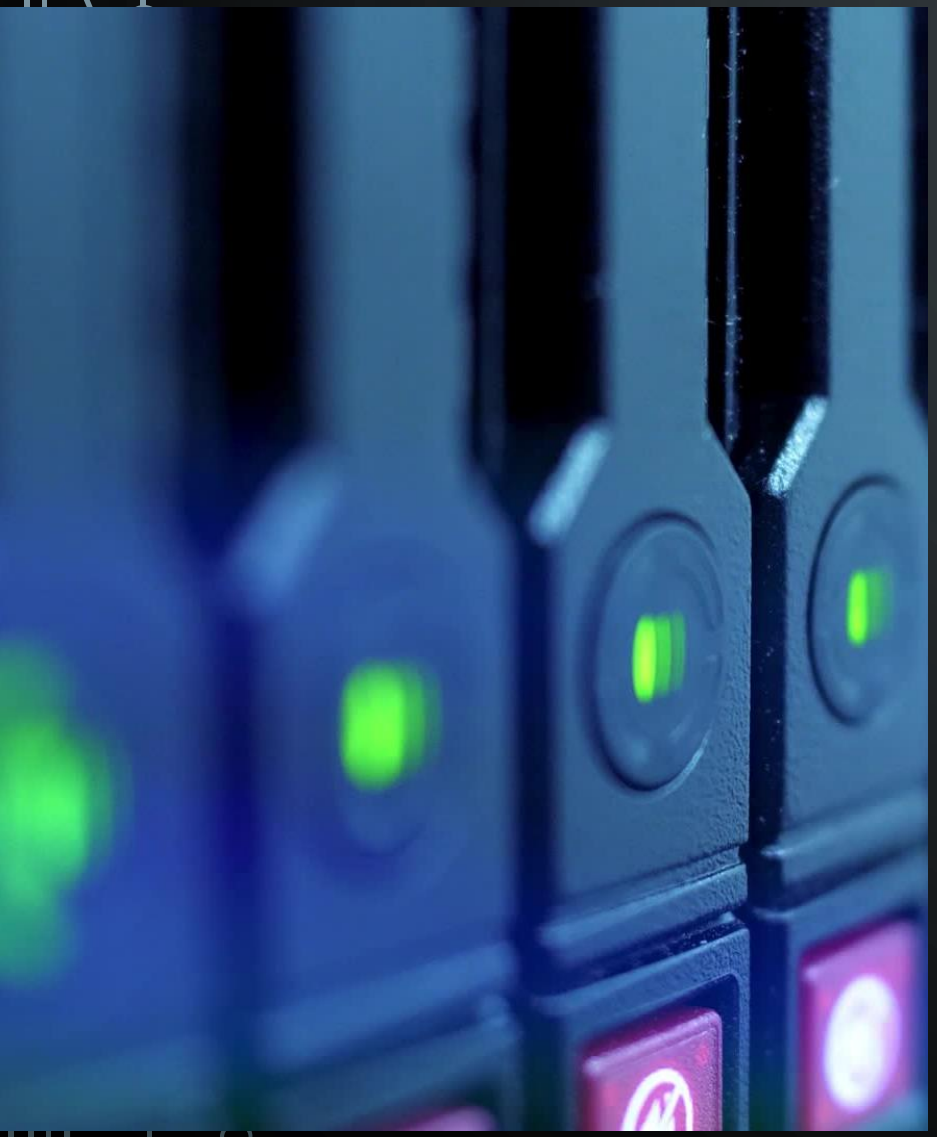
The pay-as-you-go pricing model in serverless architecture helps organizations save costs by only paying for actual resource usage.



EXAMPLE: MLB/IPL CRICKET MOBILE APP/WEBSITE

- Mobile/Website Login based UI
- Security auth for 1000s of concurrent requests
- Users trigger events which invoke FaaS offerings (Microsoft Azure Functions, Google Cloud Functions, AWS Lambda)
- Backend Database where info/data shared with users is stored. BaaS solutions offering SQL Databases, static content, media storage etc.

REQUIREMENTS FROM A SERVERLESS POSTGRESQL DATABASE IN CLOUD ENVIRONMENTS



AUTOMATIC SCALABILITY

Seamless Scaling

A serverless Postgres database needs to adjust resources dynamically based on real-time workload demands without human intervention. Right now typical provisioning is for peaks.

Performance Consistency

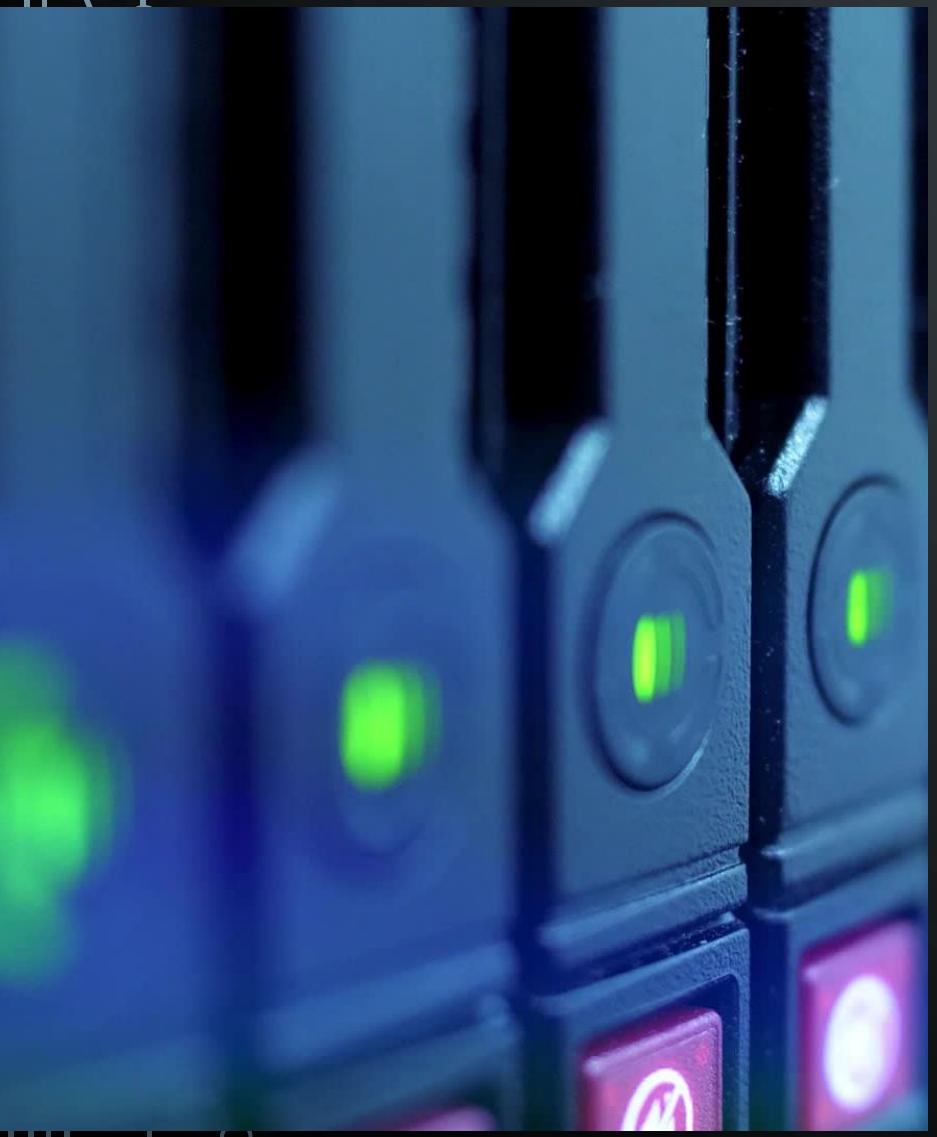
Automatic scalability needs to ensure that performance remains consistent even during varying workload conditions, enhancing user experience.

Workload Adaptation

The system needs to intelligently adapt to changes in workload, optimizing resource allocation for efficiency.

HIGH AVAILABILITY AND RELIABILITY

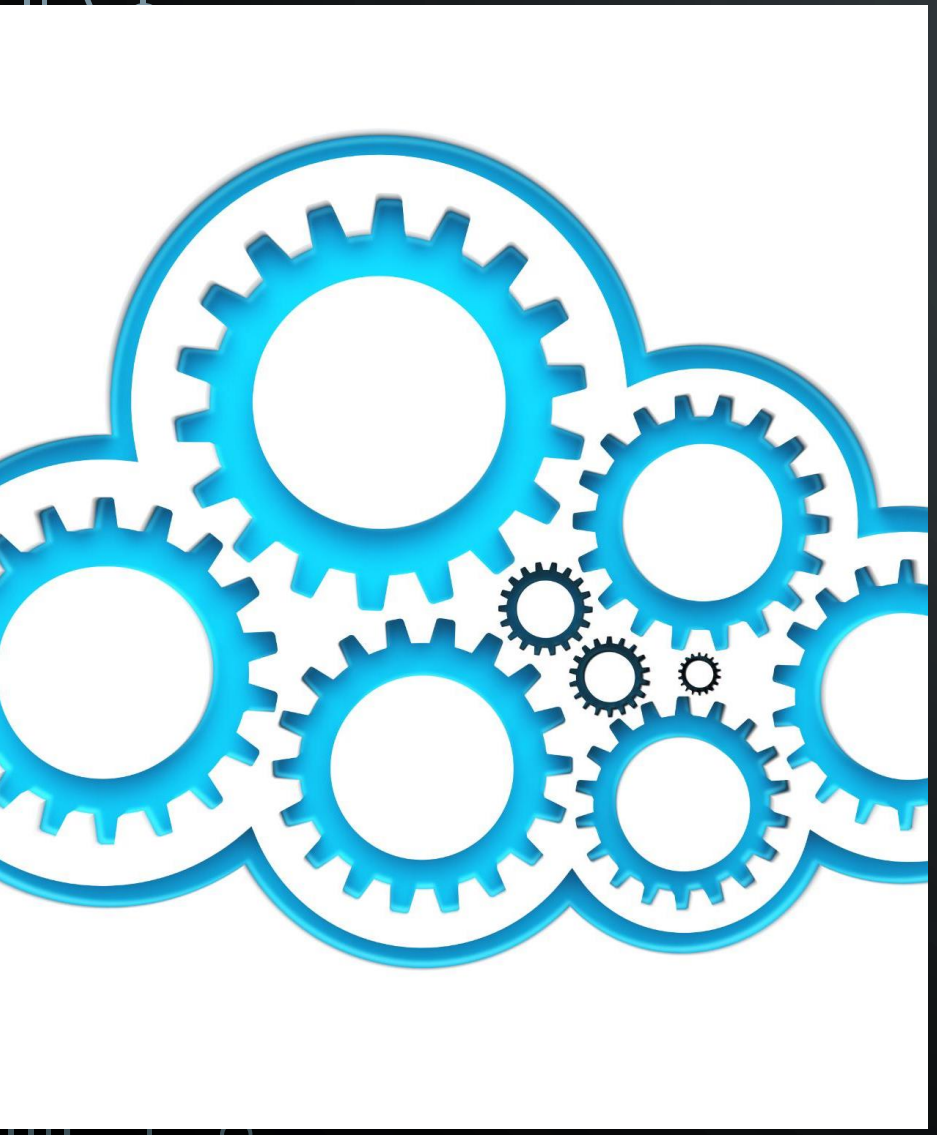
- High availability and fault tolerance are crucial. Serverless databases need to provide features like read replicas, availability zones, and self-healing capabilities to ensure data durability and minimize downtime
- High availability is essential for maintaining service continuity and minimizing downtime in database systems.
- Reliability ensures that systems perform consistently under expected workloads.
- Robust failover mechanisms are necessary to quickly switch to backup systems without service interruption.



EASY TO MANAGE AND COST EFFICIENCY

- Ease of Management: One of the main benefits of serverless databases is the reduction in operational overhead. Users expect automated database management, allowing them to focus on business development rather than infrastructure maintenance

- Cost Efficiency: Pay-as-you-go pricing models are highly valued, as they allow users to pay only for the database resources they actually use. This helps in aligning costs with usage and finding an effective balance

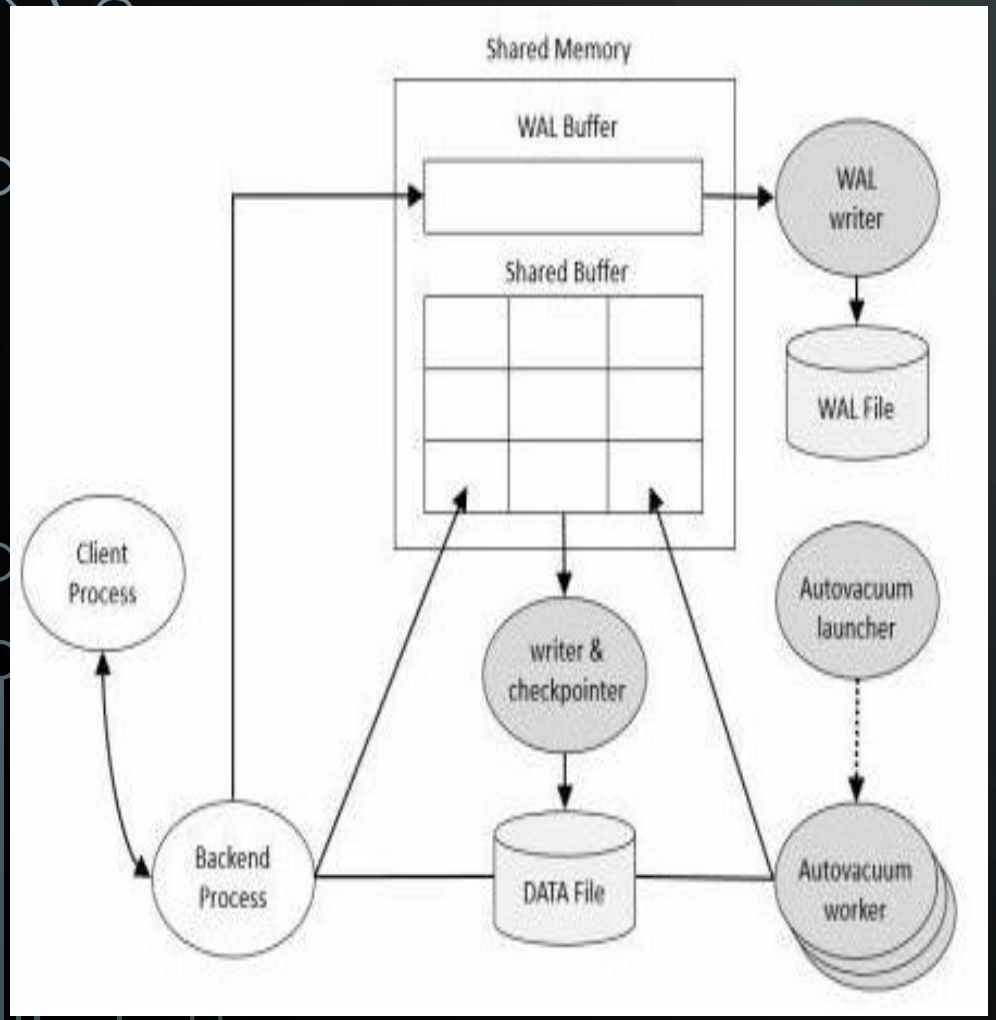


EFFICIENT DATA STORAGE

- **Serverless Database needs to scale to "infinite" storage sizes. Object storage in the cloud can be leveraged to store the data.**
- Efficient data processing and storage management are crucial for optimizing system performance and ensuring quick access to data.
- Serverless databases need to offer seamless data handling, allowing applications to scale automatically according to demand without traditional infrastructure.

TECHNICAL COMPONENTS FOR SERVERLESS POSTGRESQL

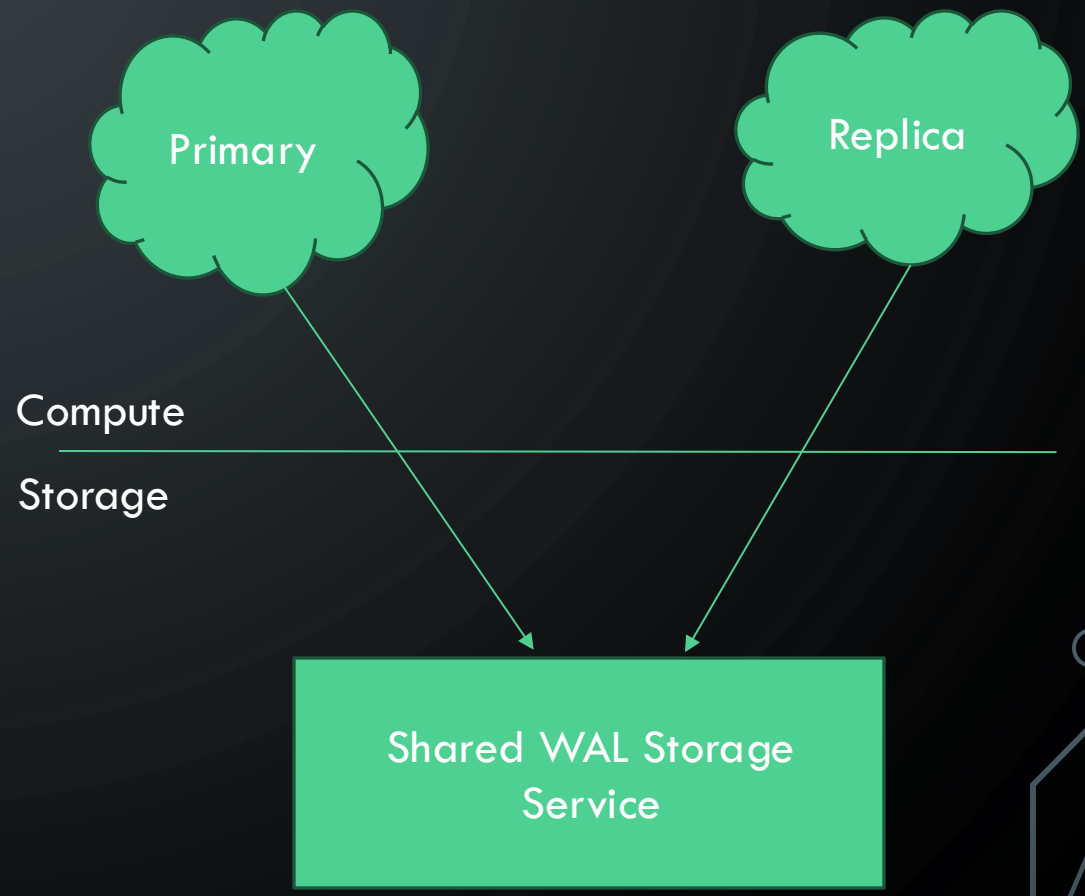
ALL'S WELL THAT ENDS WAL!



- Standard method for ensuring data durability and data integrity
- All data changes have to be logged in WAL records and they need to flushed to permanent storage to indicate transaction commit success, data pages flushed asynchronously
- Database can be fully re-created by "replaying" the WAL log (how physical replicas work)
- Checkpoints save data files into permanent storage and bring the data pages in sync with the WAL upto that point

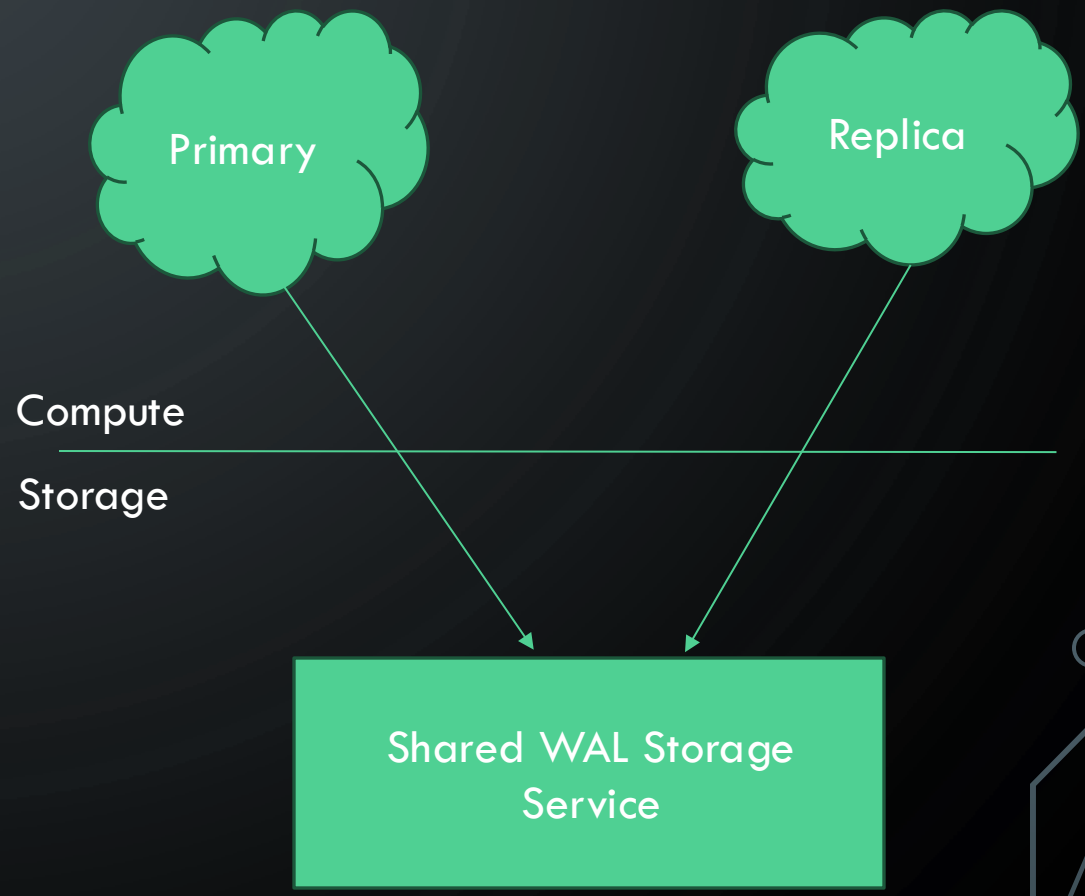
EVOLVE: SHARE THE WAL

- Single source of WAL now
- Replicas can sync directly from single Shared WAL Storage Service
- Failovers are simple now! Any replica can be promoted as the primary (no complicated orchestration for failover)



EVOLVE: SHARE THE WAL

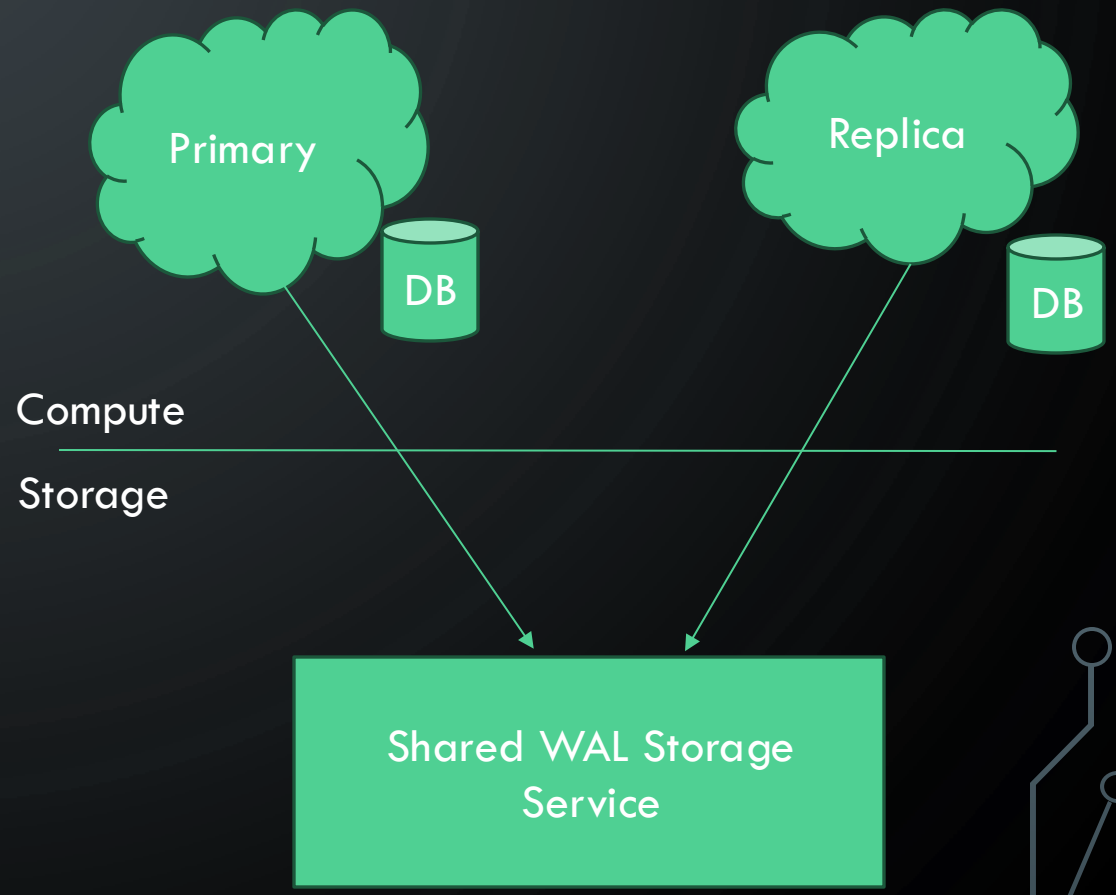
- Single source of WAL now
- Replicas can sync directly from single Shared WAL Storage Service
- Failovers are simple now! Any replica can be promoted as the primary (no complicated orchestration for failover)



EVOLVE: SHARE THE WAL

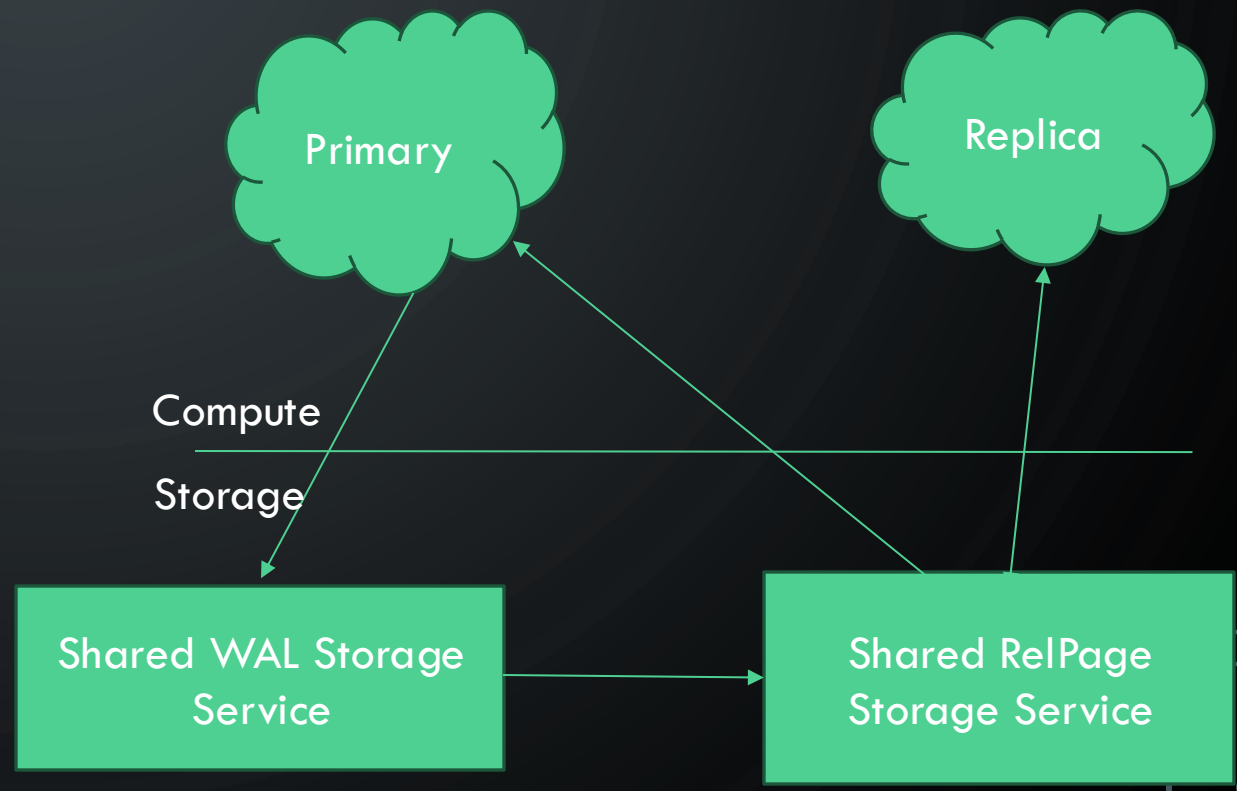
- **Single source of WAL now**
- **Replicas can sync directly from single Shared WAL Storage Service**
- **Failovers are simple now! Any replica can be promoted as the primary (no complicated orchestration for failover)**

SERVERLESS? ….No! Each node still has a copy of the whole data in it.



EVOLVE: SHARE THE WAL AND DATA

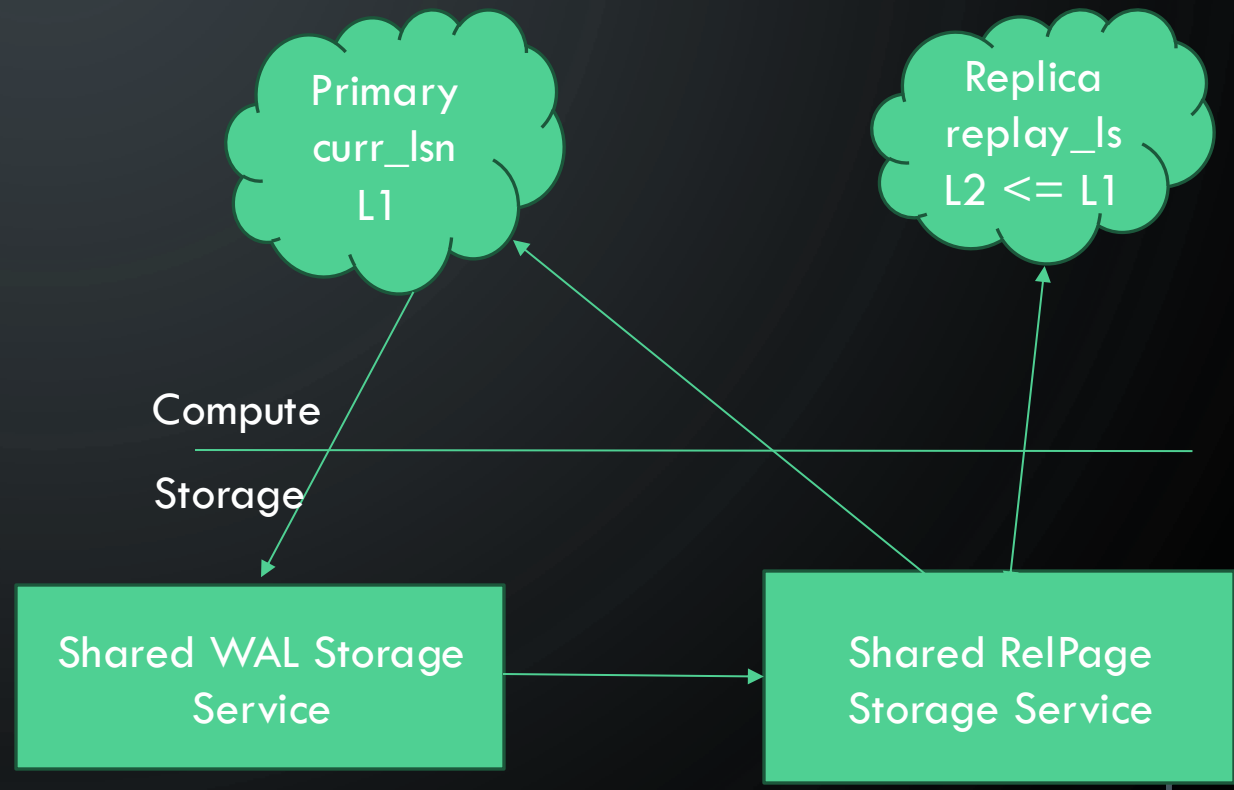
- Previous iteration shared just the WAL, now let's share data.
- LSN: Log Sequence Number is a pointer (offset) in the WAL to record Insert, Update, Delete changes
- The "pd_lsn" field in page header points to the last WAL LSN that modified this page



EVOLVE: SHARE THE WAL AND DATA

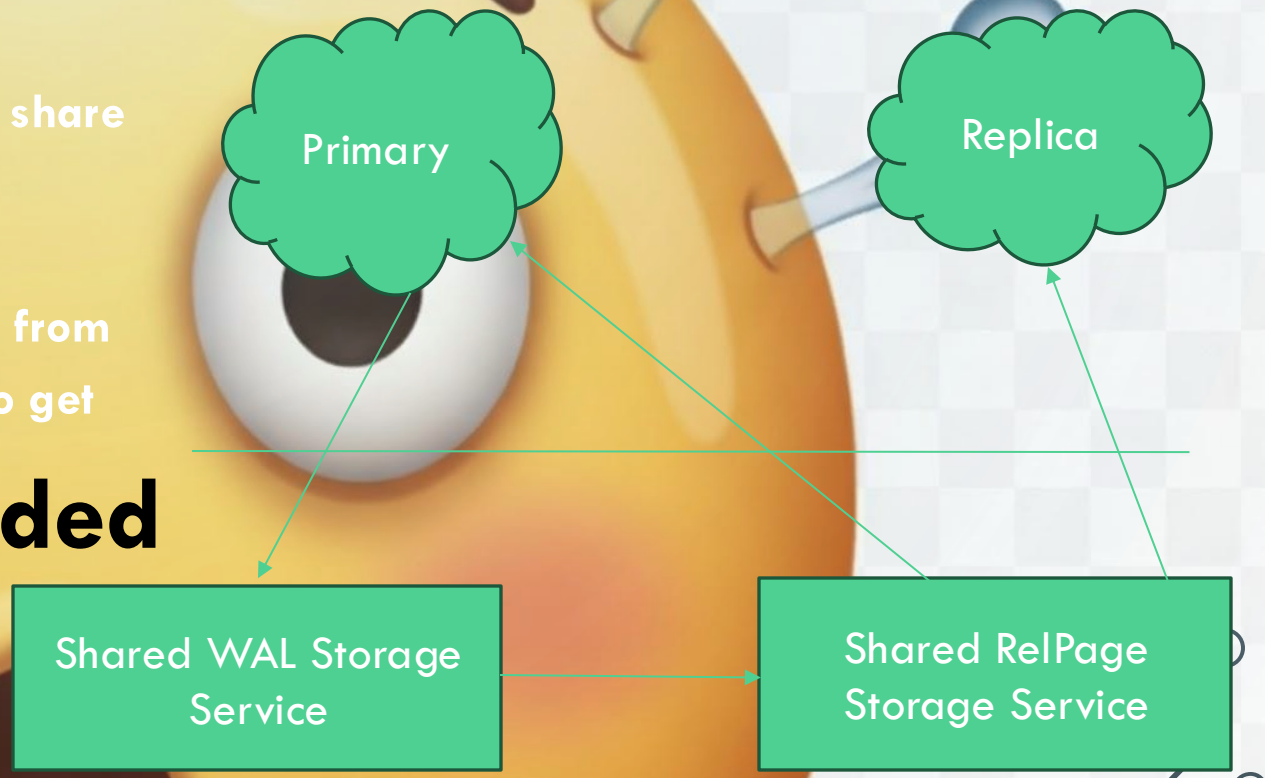
- Compute node's local storage (local disk, shared buffers) acts like a cache. If data is not found locally, compute makes a request to the RS asking for a page at a specific LSN since replica can lag a primary and needs a view at that earlier LSN.

- Relpage Storage Service (RS) will get WAL pages from WAL Storage Service (WS) and do "checkpoint" to get relpages at respective LSNs. No checkpoint needed on Primary!!!!



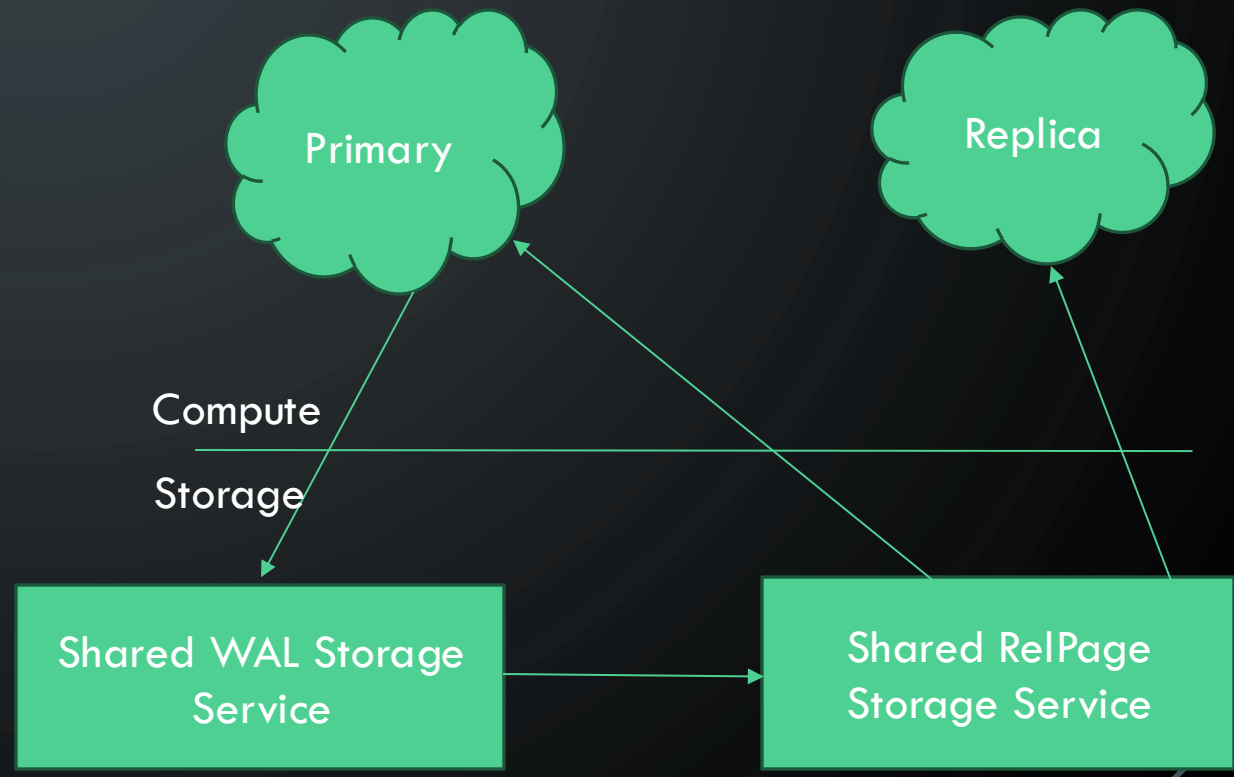
EVOLVE: SHARE THE WAL AND DATA

- Previous iteration shared just the WAL, now let's share data
- Relpage Storage Service (RS) will get WAL pages from WAL Storage Service (WS) and do "checkpoint" to get relpages. **No checkpoint needed on Primary!!!!**



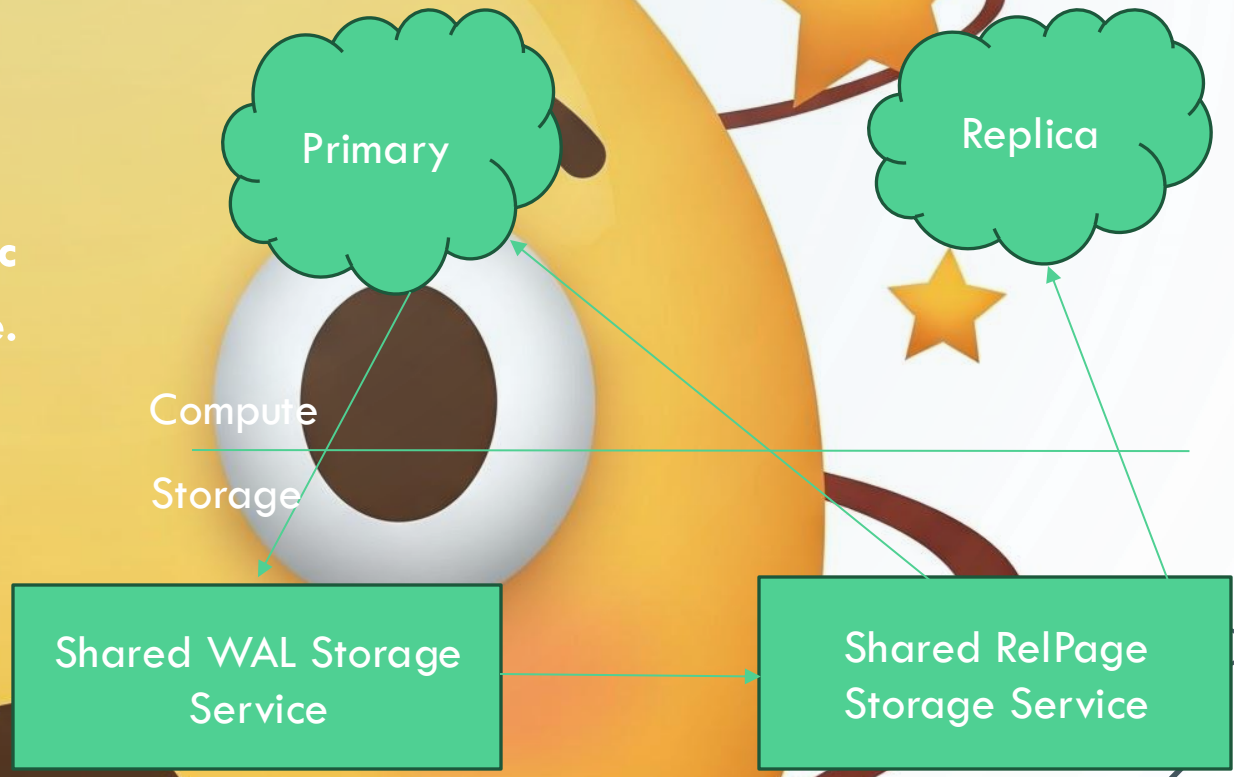
EVOLVE: SHARE THE WAL AND DATA.. CONTINUED.

- **Primary/Replica compute nodes will get the basic required subset of data files at initialization time. (including conf file, SLRU files, relmap file, auxiliary files, etc.)**
- **Primary compute node doesn't need** `full_page_writes` now! Much lesser WAL generation after a "checkpoint"



EVOLVE: SHARE THE WAL AND DATA... CONTINUED.

- Primary/Replica compute nodes will get the basic required subset of data files at initialization time. (including conf file, SLRU files, relmap file, auxiliary files, etc.)

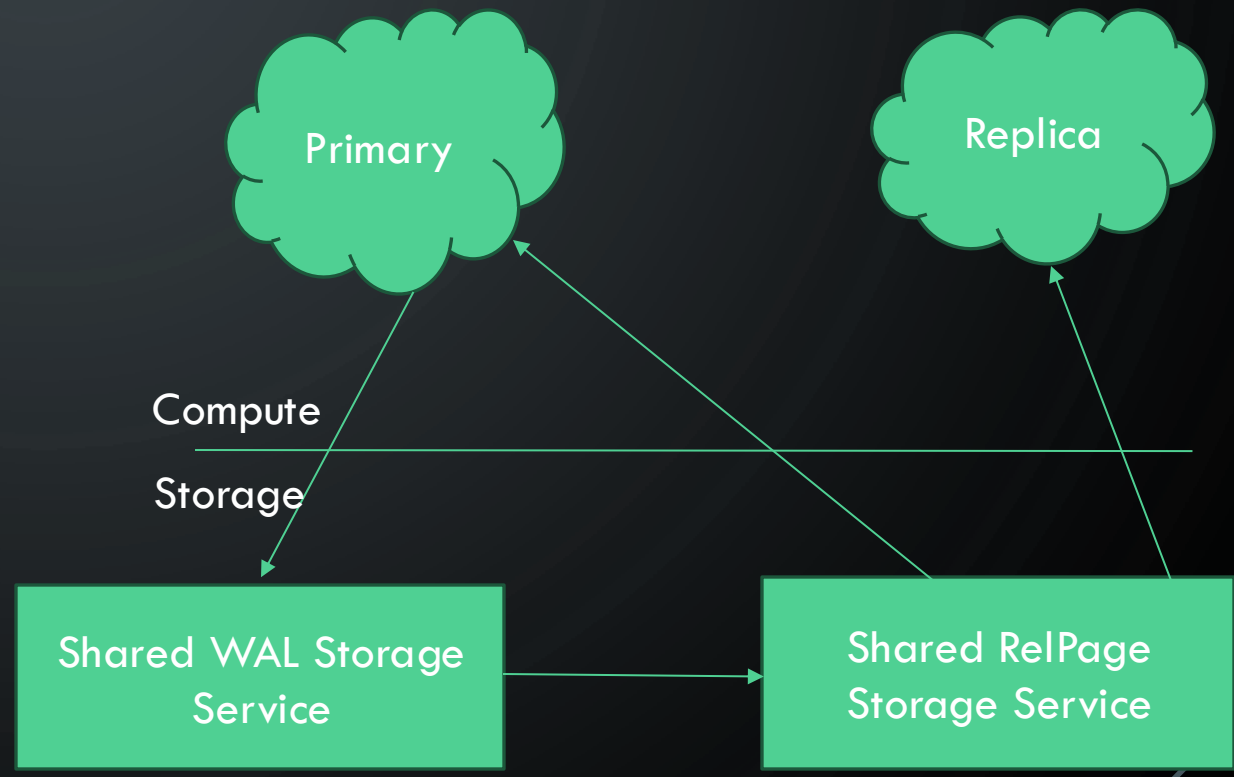


- **Primary compute node doesn't need full_page_writes now!** Much

WAL generation after a "checkpoint"

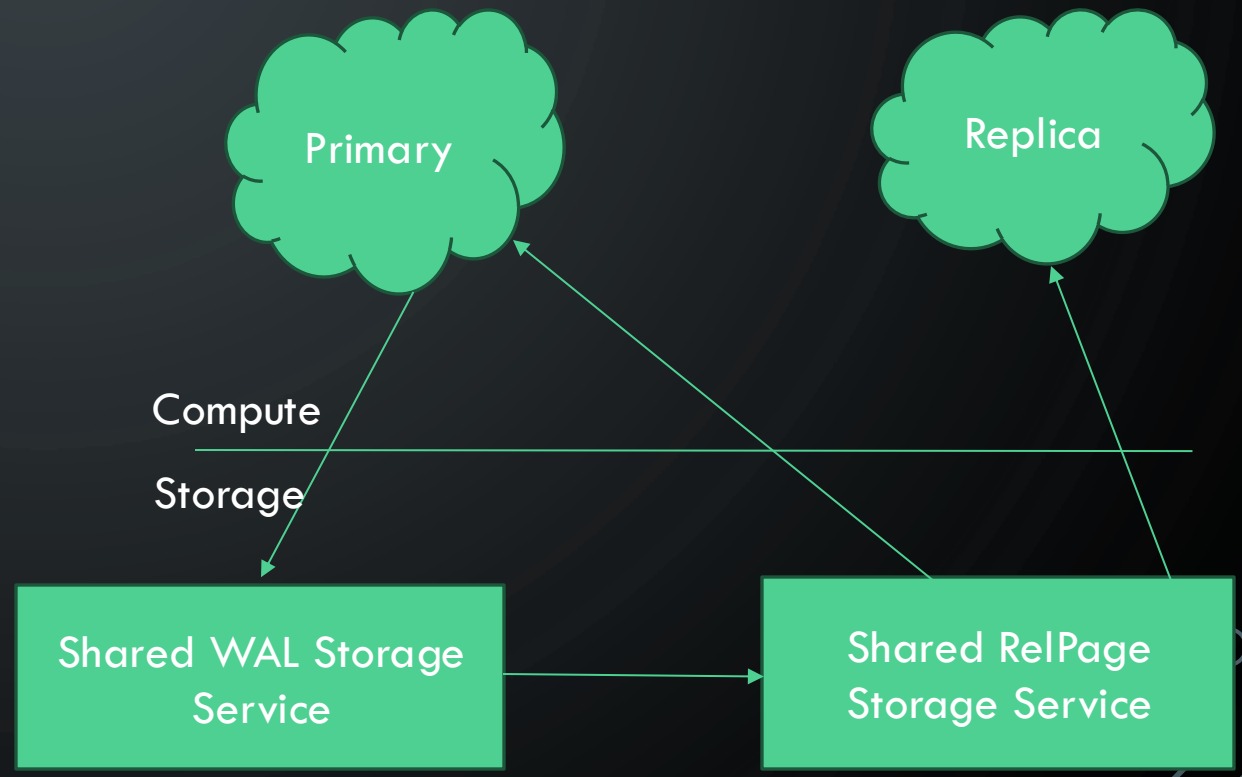
EVOLVE: SHARE THE WAL AND DATA.. CONTINUED.

- Replica compute nodes can be created much faster. They will get the same data as the primary from the RS.
- Replica compute nodes will have lesser lag vis a vis traditional PG replicas. They will only replay WAL for relpages in shared buffers and they don't persist anything locally!



EVOLVE: SHARE THE WAL AND DATA.. CONTINUED.

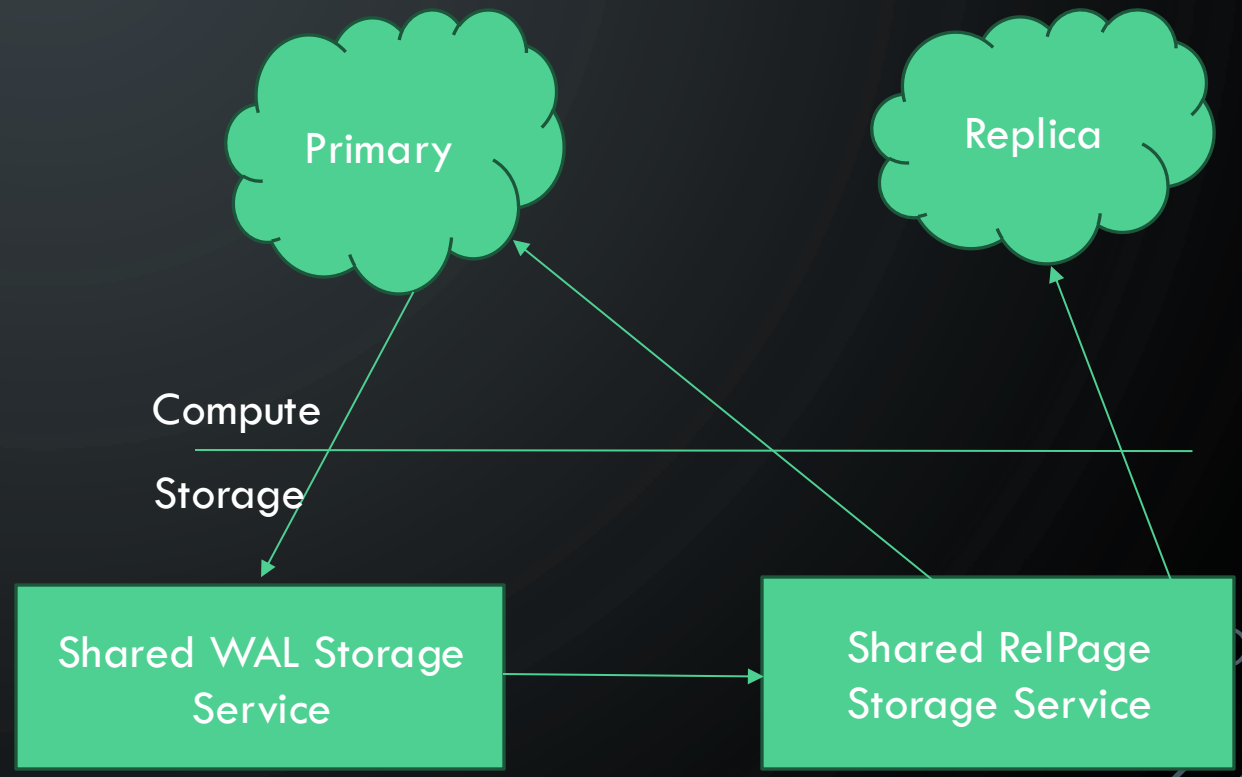
- Faster recovery across compute node restarts! They don't need to perform any recovery and can get latest relpages from RS.
- Fast branching/cloning of the database possible now. (Copy on Write in RS, storage snapshots, etc are metadata activities)



EVOLVE: SHARE THE WAL AND DATA.. CONTINUED.

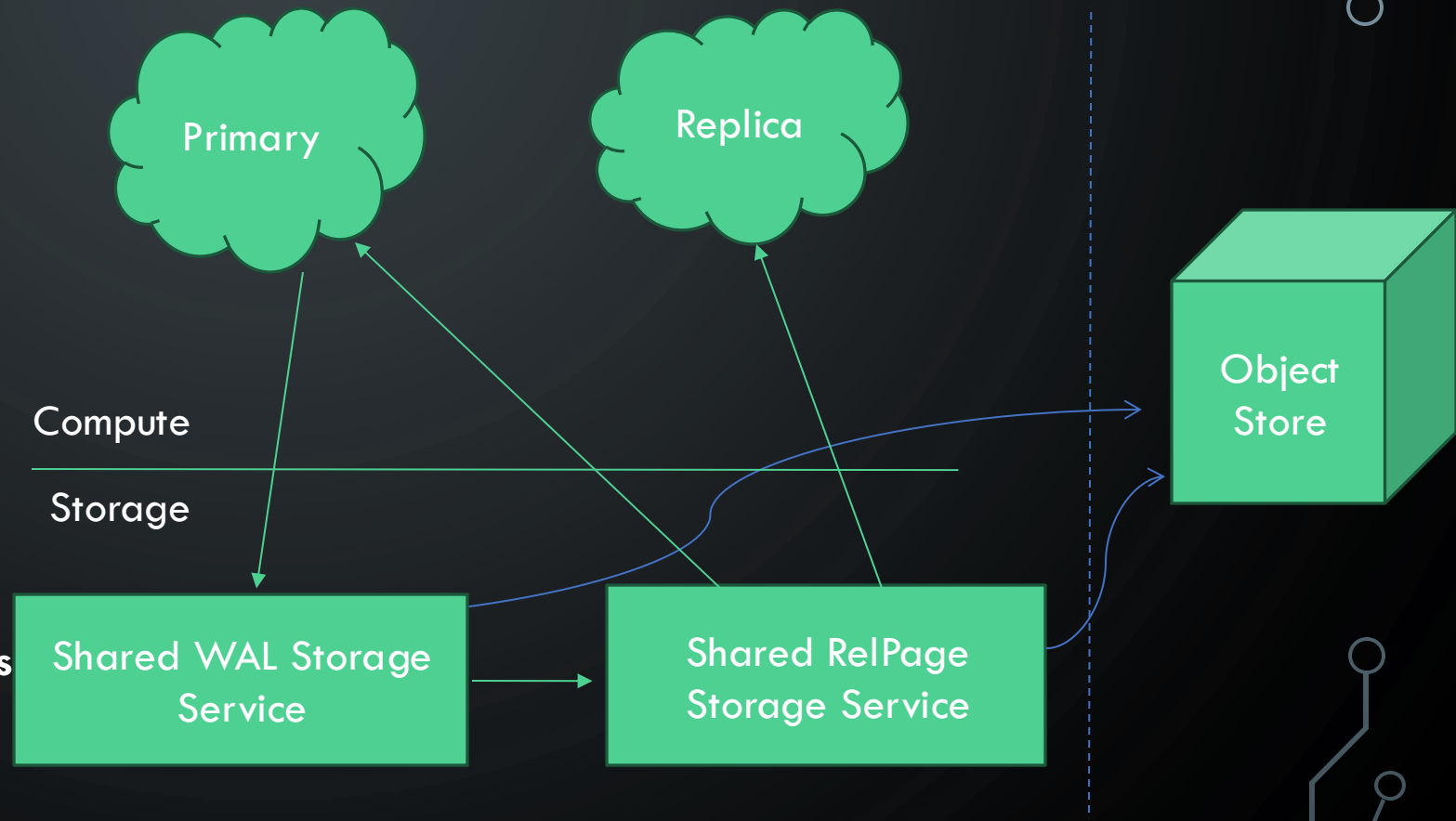
Serverless? Hmmm..

- Faster recovery across compute node restarts! They don't need to perform any recovery and can get latest relpages from RS.
- Fast branching/cloning of the database possible now. (Copy on Write in RS, storage snapshots, etc are metadata activities)



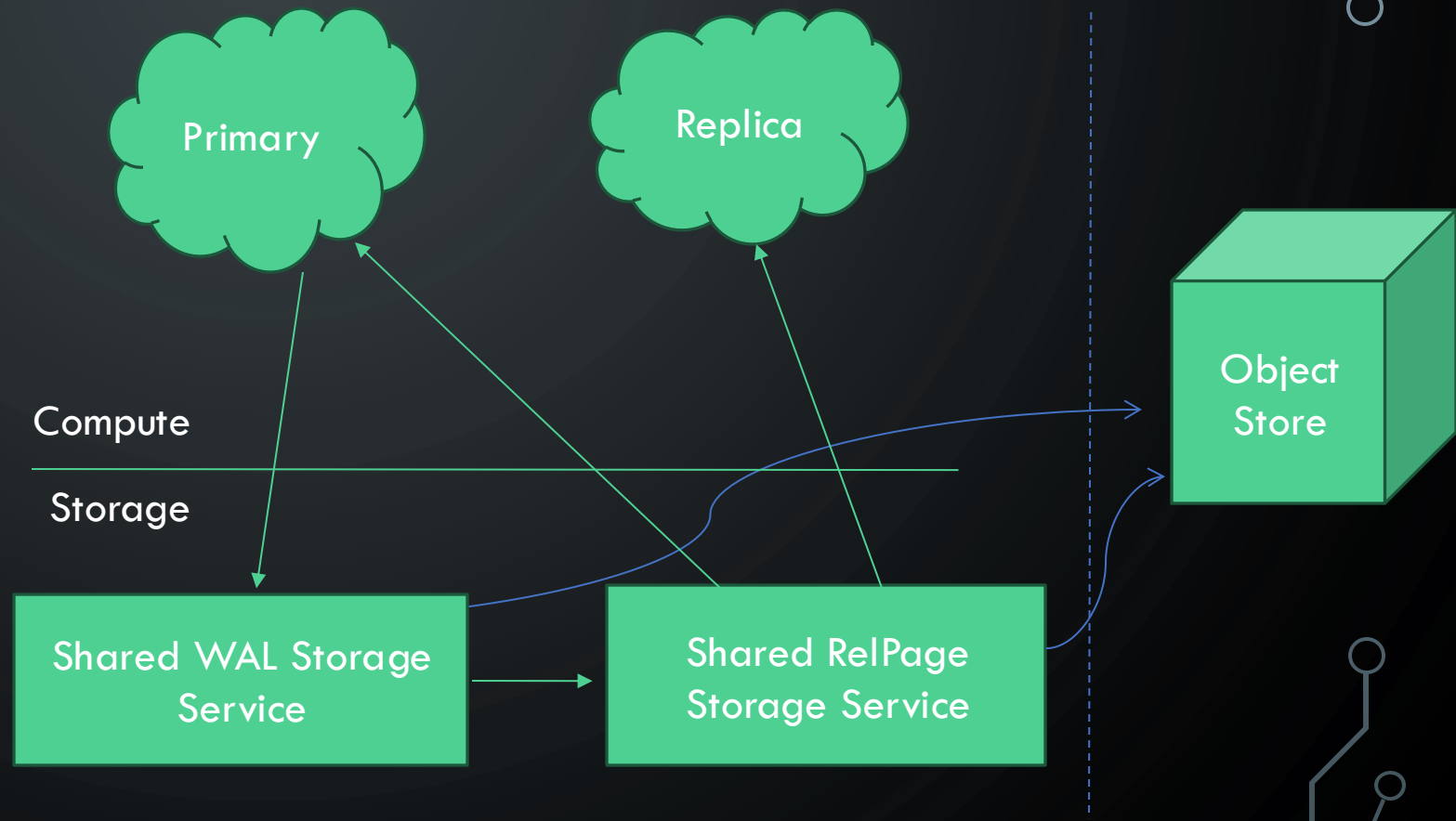
EVOLVE: BACKUP

- Backups now not tied to the PG data layout.
- WS and RS can have "infinite" and "cheap" backup stores in the form of cloud OBJECT STORES (S3, XStore, Google Cloud..)
- WS and RS restarts can re-seed from Object Stores. WS and RS components can have secondary components for HA and quick failover.



EVOLVE: BACKUP..
CONTINUED

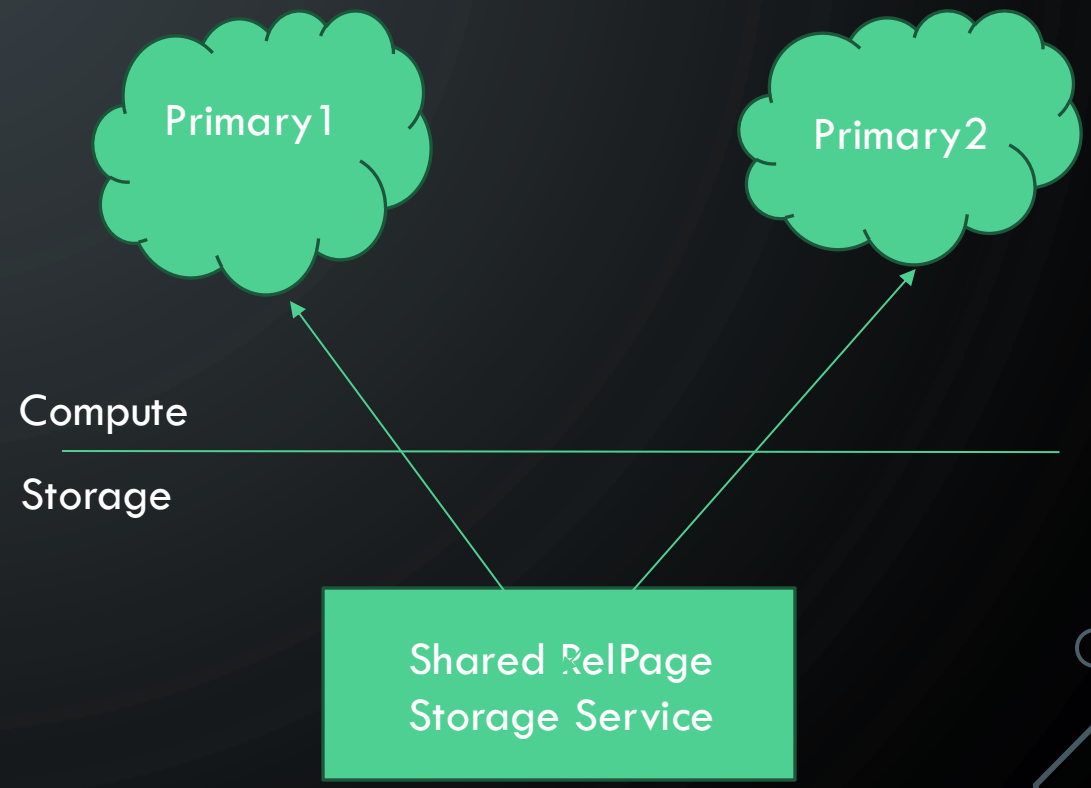
- Fast PITR (Point in Time Recovery) possible if the RS saves full page history.
- Replication of data across continents now possible at storage layer. No need to run any compute node at the other geo-distributed location.



EVOLVE: MULTI-TENANT RS.. CONTINUED.

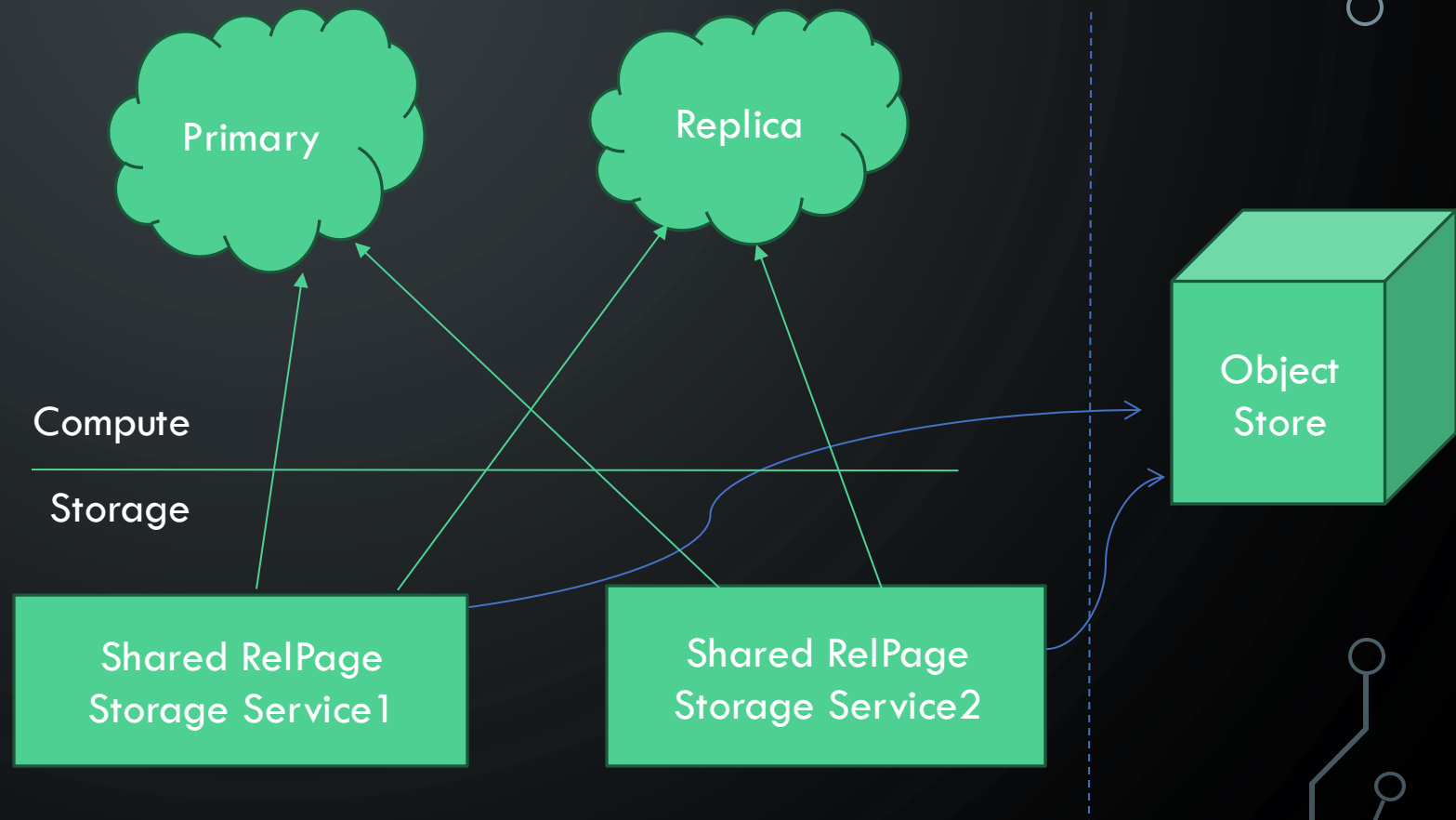
- Single RS can serve multiple tenants (separate namespace, threads for pages, etc. via "tenant_ids")
- Idle compute nodes can now scale down to 0! Live compute/storage nodes can be dynamically "resized" in terms of CPU/RAM/Storage on the fly leading to "PAY AS YOU GO".
- Cold data can be stored in Object Store

Serverless? … Getting there :-)



EVOLVE: SHARDING

- Decide upon a sharding strategy and shard data across multiple RelPage Services.
- With data being backed to object stores, a decent strategy could allow re-sharding with minimal data movement.
- Higher IOPs. Separate WS and sharded RelPage Services



SERVERLESS? ARE WE THERE YET?

- Separation of Postgres into Compute/Storage components.
- Further separation of storage into WS and RS components.
- Compute nodes can scale to 0. Other components can be dynamically resized based on usage (pay as you go)
- Backups in Object Store. Cheap and Limitless.

SERVERLESS? ARE WE THERE YET?

- Effective Sharding, Multi-tenancy
- High Availability in place for various components
- Compute nodes are quick to restart
- Compute nodes are quick to spin-up
- Effective monitoring can allow for performance, availability based on dynamic resizing

FUTURE PERSPECTIVES AND CLOSING THOUGHTS

- **PostgreSQL is geared up for Serverless deployments.**
- **Future enhancements could consider multi-writer compute nodes (by leveraging global lock manager infrastructure)**
- **Cloud infrastructure continues to evolve and dynamic resource management changes can be incorporated on an ongoing basis.**
- **PostgreSQL on track for truly cloud-native architectures!**

Thank You!