Babelfish for PostgreSQL

Rob Verschoor
Principal Database Engineer
AWS

Chandra Sekhar Pathivada
Sr. Database Specialist Solutions Architect
AWS

Agenda

- Introducing Babelfish for PostgreSQL
- Open-source project
- Deployment model
- Migration steps
- Supported T-SQL features
- Architecture
- T-SQL vs. PostgreSQL semantics

Introducing Babelfish for PostgreSQL

Run SQL Server applications on PostgreSQL with little to no code changes

Keep existing queries

Translation layer enables PostgreSQL to understand Microsoft SQL Server's proprietary T-SQL

Accelerate migrations

Lower risk and complete migrations faster, saving you months to years of work

Freedom to innovate

Run T-SQL code side-by-side with new open source functionality and continue developing with familiar tools

What is Babelfish for PostgreSQL?

Babelfish for PostgreSQL is:

- *“Babelfish is a migration accelerator providing semantically correct execution of T-SQL over the TDS protocol, natively implemented in PostgreSQL.”*
- A native implementation of TDS and T-SQL, using PG building blocks
- A PostgreSQL extension (in fact, 3 extensions)
- A second endpoint in an Aurora cluster (TDS + PG ports)
- Open-source

It is not:

- A SQL ‘mapping’ proxy between the client app and PG
- A separate server
- A temporary solution for customers
- Replacing PG

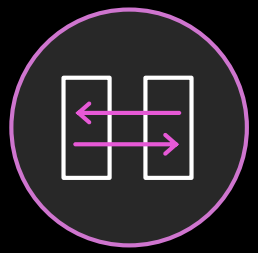
Open-source project: Babelfish for PostgreSQL

Project website
https://babelfishpg.org

Freedom from proprietary databases

No vendor lock-in

Apache 2.0 and PostgreSQL licenses



Use it for any purpose, innovate, and distribute your modifications

Available on GitHub
https://github.com/babelfish-for-postgresql

Is community driven

Babelfish for PostgreSQL design tenets

GUIDING PRINCIPLES

No compromises on correctness

Database calls either work the same as in SQL Server or return an error

Wire protocol compatibility

Applications work without changing database drivers

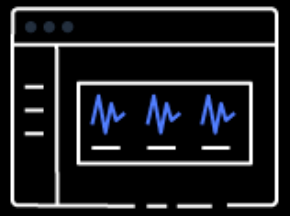
Interoperability

Use PostgreSQL functionality from T-SQL and T-SQL functionality from PostgreSQL code

Deployment model for Babelfish for PostgreSQL

HOW DO I ADD NEW FUNCTIONALITY IN MY MIGRATED APPLICATIONS?

Develop new functionality in T-SQL using SQL Server database drivers



Develop new functionality in PostgreSQL using PostgreSQL database drivers

Develop new functionality in PostgreSQL and call from T-SQL using SQL Server database drivers

Potential migration opportunities

- Home-grown applications
- Database-agnostic applications
- ISV applications
- RDS for SQL Server databases
- On-premises SQL Server databases
- Self-managed SQL Server on Amazon EC2 or Azure VMs
- Azure SQL Distributed Transaction Units

Migration Steps

1. Export DDL (reverse-engineer with SSMS)
   - Make sure to include triggers, logins, owners, and permissions (not included by default)

2. Run Babelfish Compass assessment tool on the DDL to find incompatibilities
   - Rewrite SQL you find to be Babelfish-incompatible. Ex: SELECT..[UN]PIVOT
   - Compass can rewrite selected features with supported T-SQL (MERGE, numeric datetime)

3. Import adjusted DDL script into Babelfish with **sqlcmd**
   - No AWS SCT conversion needed! Babelfish supports T-SQL SQL/DDL syntax
   - First set Babelfish escape hatches to 'ignore' with sp_babelfish_configure

4. Migrate data using AWS Database Migration Service (DMS)
   - (Or, test with a smaller data set to test getting the app going)

5. Reconfigure the client app to connect to Babelfish instead of SQL Server

Current Releases of Babelfish for PostgreSQL

- October 2021: 1.0.0 -- GA ! → PG 13.4
- February 2022: 1.1.0 → PG 13.5
- March 2022: 1.2.0 → PG 13.6

Documentation:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/babelfish.html

Supported T-SQL features

SQL SERVER-SPECIFIC FEATURES

- Triggers, Stored Procedures, scalar SQL Functions, Views
- T-SQL transactional semantics, incl. nested transactions & savepoints
- Data types (**money**, **sql_variant**, 3-millisecond **datetime**)
- Control-of-Flow statements (e.g. GOTO, TRY/CATCH)
- Table Data Types, Table Parameters and Table-Valued functions
- Static cursors
- Computed columns
- Dynamic SQL (*EXEC with string param*) and *sp_executesql*
- Application Locks
- SELECT…FOR XML { RAW | PATH }
- Multiple results sets per procedure/batch

Supported T-SQL features

SQL SERVER-SPECIFIC FEATURES

- #Temporary Tables
- Built-in functions
- IDENTITY columns
- Case-insensitive identifiers
- Collation support
- DML OUTPUT clause support
- @@ERROR code mapping
- CREATE DATABASE; USE <db>
- SQL Server catalogs (selection)
- SSL/TLS; Kerberos

v.1.2.0 : Supported T-SQL features

SQL SERVER-SPECIFIC FEATURES

- TIMESTAMP/ROWVERSION columns
- CREATE USER
- CREATE SCHEMA…AUTHORIZATION…
- GRANT/REVOKE object permissions to a DB user
  - SELECT,INSERT,UPDATE,DELETE,REFERENCES,EXECUTE
- Trigger functions: COLUMNS_UPDATED(), UPDATE()
- ISJSON(), JSON_QUERY(), JSON_VALUE()
- Additional SQL Server catalogs; system stored procs; INFORMATION_SCHEMA

- Coming: support for DMS (initial load)

Currently not supported

- Use of in-database encryption
- MARS, Database Mail, Geospatial, SQLCLR, Filetable, Columnstore
- Dependency on MSDTC OR SQL Server Agent
- Limited use case support with SSRS, SSIS, or SSAS
- Dependency on remote servers (linked servers)
- Dependency on OPENJSON()
- Dependency on OPENXML(), XQuery, Xpath
- Dependency on Bulk Copy API (bcp-in, BULK-INSERT)
- SQL Server Parallel Data Warehouse
- (see Aurora/PG documentation for more)

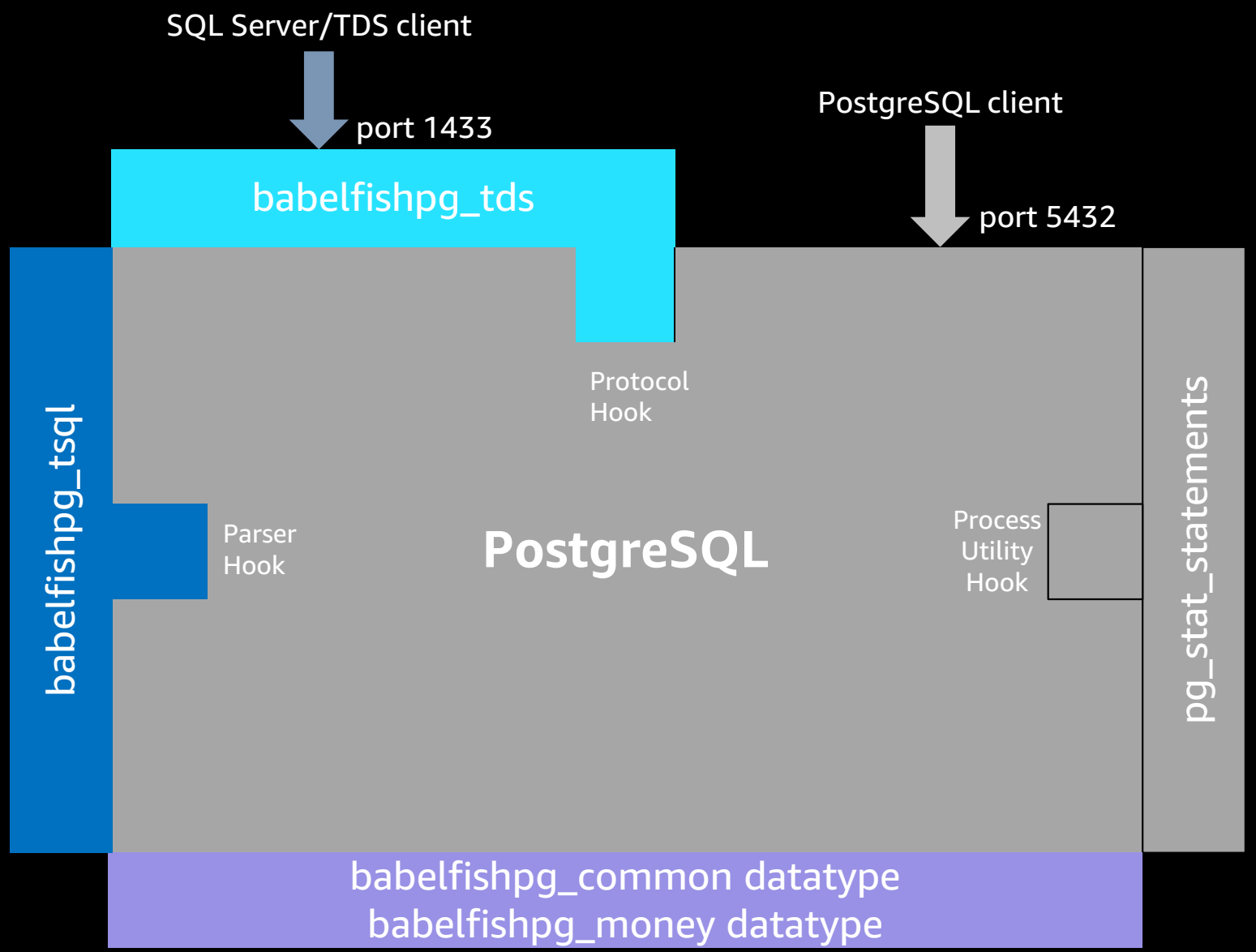
Support for SQL Development Tools

- Limited support for SSMS (Query Editor works)

- DBeaver (recommended GUI tool)
  - Free, open source and works on all major OSes (Win/Mac/Linux)
- **sqlcmd** (recommended for script execution)

- With other tools, your mileage will vary
- High priority to support other tools post GA (such as VS Code)

Babelfish for PostgreSQL architecture



Babelfish Architecture

How does a T-SQL procedure work in Babelfish?

1. Execute CREATE PROCEDURE with full T-SQL syntax, when connected through the TDS port
2. Babelfish creates a PostgreSQL procedure
3. The procedure is executed from T-SQL (with EXECUTE), with T-SQL semantics
4. Alternatively, the procedure can also be executed for PG (with CALL). In this case, transactional semantics will be those of PG

Detecting you’re running on Babelfish

Users can detect programmatically whether their T-SQL application is running on Babelfish:

```
SELECT CAST(SERVERPROPERTY('Babelfish') AS VARCHAR)
```

- Returns 1 when running on Babelfish, NULL when running on SQL Server

```
SELECT CAST(SERVERPROPERTY('BabelfishVersion') AS VARCHAR)
```

- Babelfish’s version number, e.g. 1.0.0 for the GA release

Transactional semantics: SQL Server vs. PostgreSQL (PG)

SQL Server:

```
1> create table t1 (a int not null unique)
2> go
1> begin tran
2> insert into t1 values(1)
3> insert into t1 values(2)
4> update t1 set a = 3 -- causes duplicate key error
5> commit
6> go


Msg 2627, Level 14, State 1, Server EC2AMAZ-5Q6FMIK, Line 4Violation of UNIQUE KEY constraint
'UQ__t1__3BD0198F21AFC10E'. Cannot insert duplicate key in object 'dbo.t1'. The duplicate key
value is (3).The statement has been terminated.


1> select * from t1
2> go
a
-----------
          1
          2
(2 rows affected)
```

By default, SQL Server keeps the transaction open and rolls back only the statement causing the error

Result in SQL Server: 2 rows

Transactional semantics: SQL Server vs. PG

PostgreSQL:

```
postgres=> create table t1 (a int not null unique);
CREATE TABLE
postgres=> DO $$
postgres$> begin
postgres$> insert into t1 values (1);
postgres$> insert into t1 values (2);
postgres$> update t1 set a = 3;   -- causes duplicate key error
postgres$> commit;
postgres$> end$$;


ERROR:  duplicate key value violates unique constraint "t1_a_key"
DETAIL:  Key (a)=(3) already exists.
CONTEXT:  SQL statement "update t1 set a = 3"
PL/pgSQL function inline_code_block line 5 at SQL statement


postgres=> select * from t1;
a
---
(0 rows)
```

PG rolls back the entire transaction

Result in PG: 0 rows

Transactional semantics: SQL Server vs. PG

Babelfish solution:

- put an (internal) savepoint before each DML statement
- in case of an error, roll back to that savepoint
- the savepoint consumes a transaction ID

Error/Exception handling: SQL Server vs. PG

SQL Server:

```
1> create table t2 (a int not null unique)
2> go
1> begin tran
2> insert into t2 values (123)
3> insert into t2 values (123) -- will cause duplicate key error
4> insert into t2 values (456)
5> commit
6> print 'more processing here!'
7> go


Msg 2627, Level 14, State 1, Server EC2AMAZ-EULFEOJ, Line 4
Violation of UNIQUE KEY constraint 'UQ__t2__3BD0198F04E3D22F'. Cannot insert duplicate key in object
'dbo.t2'. The duplicate key value is (123).
The statement has been terminated.


more processing here!

1> select * from t2
2> go
a
-----------
        123
        456
(2 rows affected)
```

By default, SQL Server continues with the next statement after an error;
To take action, must check @@ERROR explicitly

Result in SQL Server: after error, subsequent statements are still executed (by default)

Error/Exception handling: SQL Server vs. PG

PostgreSQL:

```
postgres=> create table t2 (a int not null unique);
CREATE TABLE
postgres=> DO $$
postgres$> begin
postgres$> insert into t2 values (123);
postgres$> insert into t2 values (123); -- will cause duplicate key error
postgres$> insert into t2 values (456);
postgres$> commit;
postgres$> raise notice 'more processing here!';
postgres$> end;
postgres$> $$;

ERROR: duplicate key value violates unique constraint "t2_a_key"
DETAIL: Key (a)=(123) already exists.
CONTEXT: SQL statement "insert into t2 values (123)"
PL/pgSQL function inline_code_block line 4 at SQL statement


postgres=> select * from t2;
a
---
(0 rows)
```

When an error occurs, PG jumps to the exception handler and leaves the block; subsequent statements are not executed

Not executed

Error/Exception handling: SQL Server vs. PG

Babelfish solution:

- Wrap each DML statement in an (internal) block with a dummy exception handler
- In case of an error, control passes to the next block ➔ the next statement
- Each such block consumes a transaction ID

```
DO $$
begin

begin
insert into t2 values (123);
exception when others then null;
end;

begin
insert into t2 values (456);
exception when others then null;
end;

$$
```

Error code mapping from PG to SQL Server

Babelfish (unchanged from SQL Server):

```
1> create table t3 (a int not null unique)
2> go
1> begin tran
2> insert into t3 values (1)
3> insert into t3 values (1) -- will cause duplicate key error
4> if @@error = 2627
5> begin
6>   print 'constraint violation, rolling back xact!'
7>   rollback
8> end
9> else
10>   commit
11> go


Msg 2627, Level 14, State 1, Server BABELFISH, Line 4
duplicate key value violates unique constraint "t3_a_key"


constraint violation, rolling back xact!


1> select * from t3
2> go
(0 rows affected)
```

PG SQLSTATE value is '23505'; mapped to 2627 by Babelfish, so existing T-SQL code can be kept

Not only mapping @ERROR value, but also severity

Xact rolled back

Error code mapping from PG to SQL Server

- Babelfish currently maps 100+ error codes from PG to SQL Server
  - including @@ERROR = 0

- Also mapping error severity: affects whether aborting statement or xact

- When not mapped, @@ERROR contains a large number
- E.g. for syntax error:

```
1> select select
2> go
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
syntax error near 'select' at line 1 and character position 7
1> select @@error
2> go
error
-----------
   33557097
(1 rows affected)
```

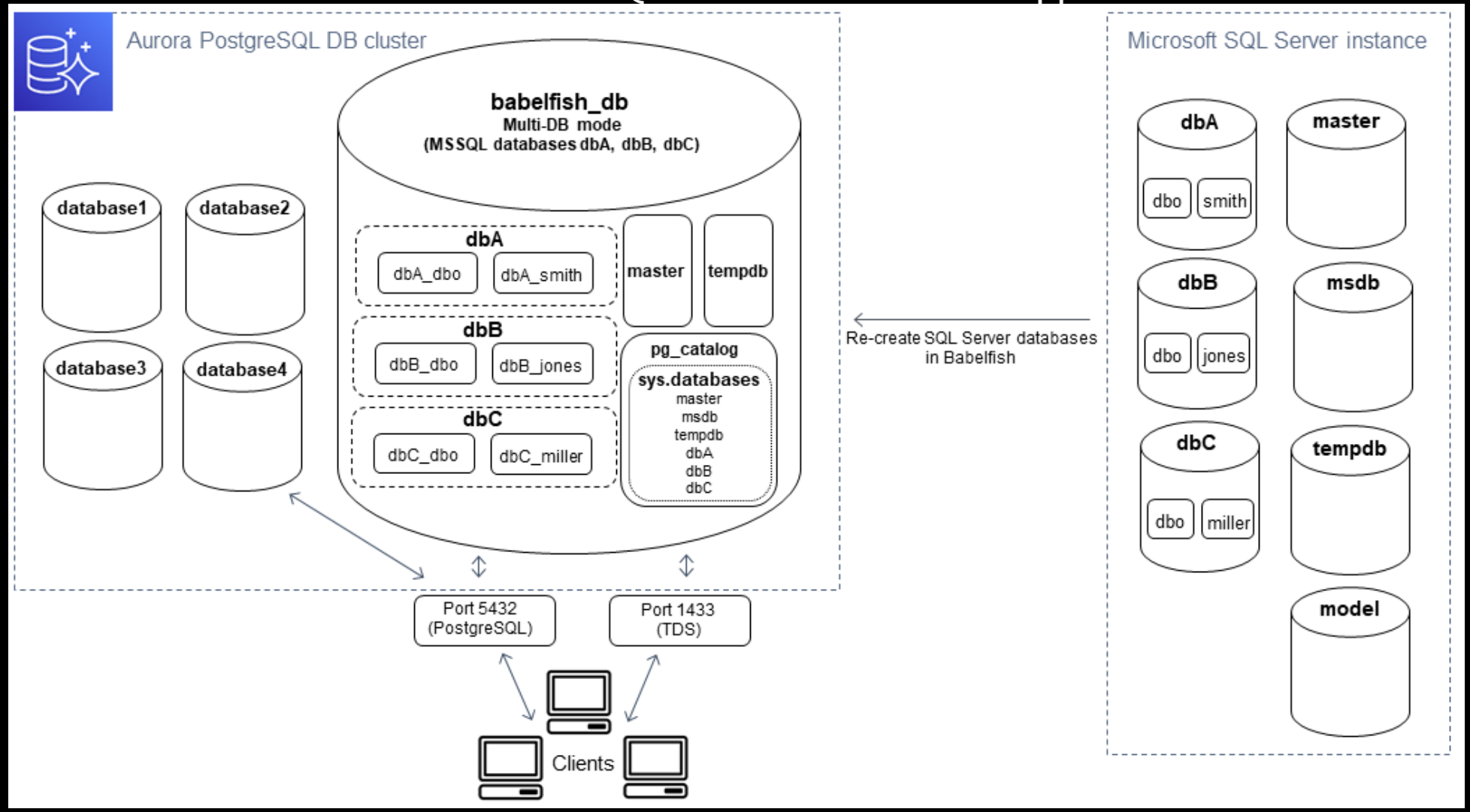
Migrating SQL Server database structure

- Babelfish provides a ‘server experience’ similar to SQL Server
  - Existing T-SQL code and existing client apps do not need to be changed
  - There’s **master**, **tempdb, msdb,** user databases, **sysdatabases** & **sp_helpdb**
  - Can run **CREATE DATABASE; USE** <database>; **SELECT DB_NAME()** from T-SQL

- How this is done:
  - The Aurora/PG cluster contains a PG database named **babelfish_db**
  - When connecting to the TDS port, the connection is placed in this PG database
  - SQL Server databases and schemas are mapped to schemas in this PG database
  - The **babelfish_db** database and the schema name mapping are transparent to the T-SQL user
  - Users can choose between single-db and multi-db mode, which affects the schema name mapping
    - This is relevant only when connecting to the PG port; in T-SQL (through TDS port), the original schema names can be used

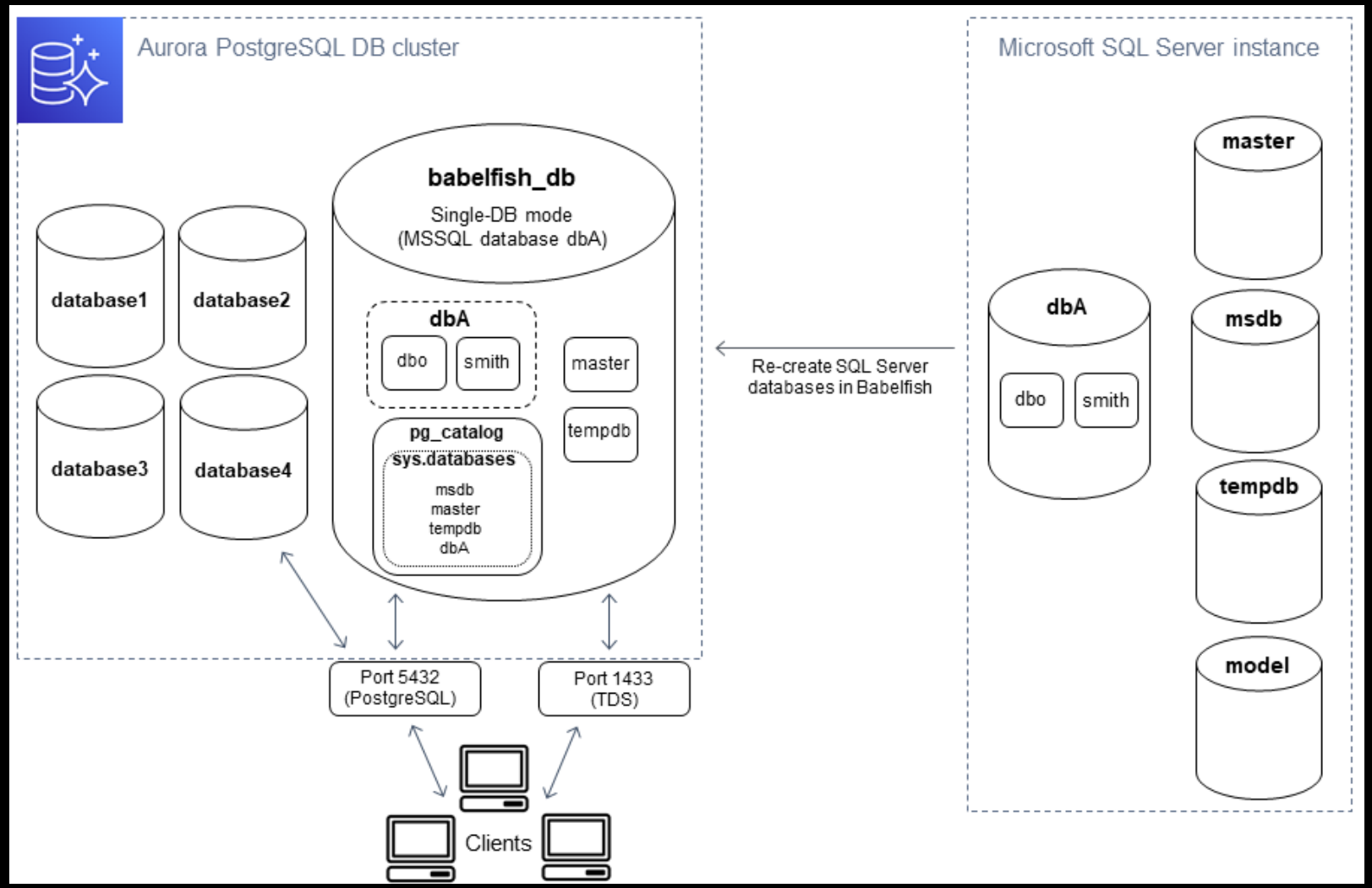
Migrating SQL Server database structure

- Multi-db mode: schema names from SQL Server user DB are mapped to different PG schema names



Migrating SQL Server database structure

- Single-db mode: schema names from SQL Server user DB remain unchanged in PG



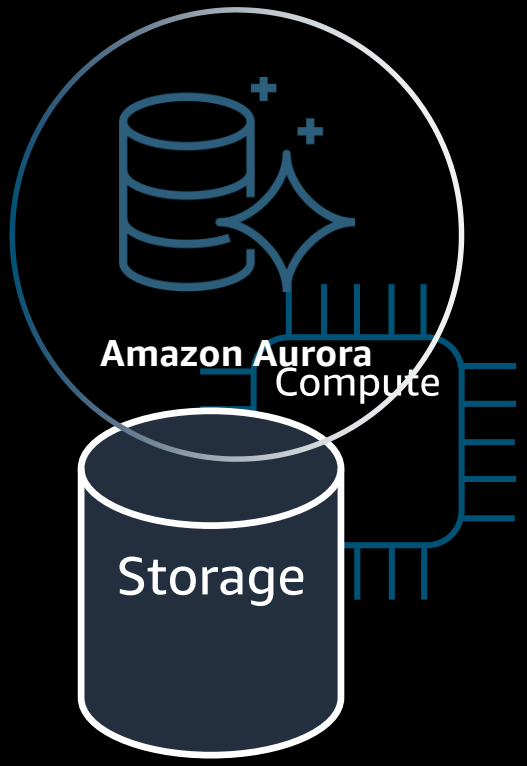
Aurora for SQL Server DBAs

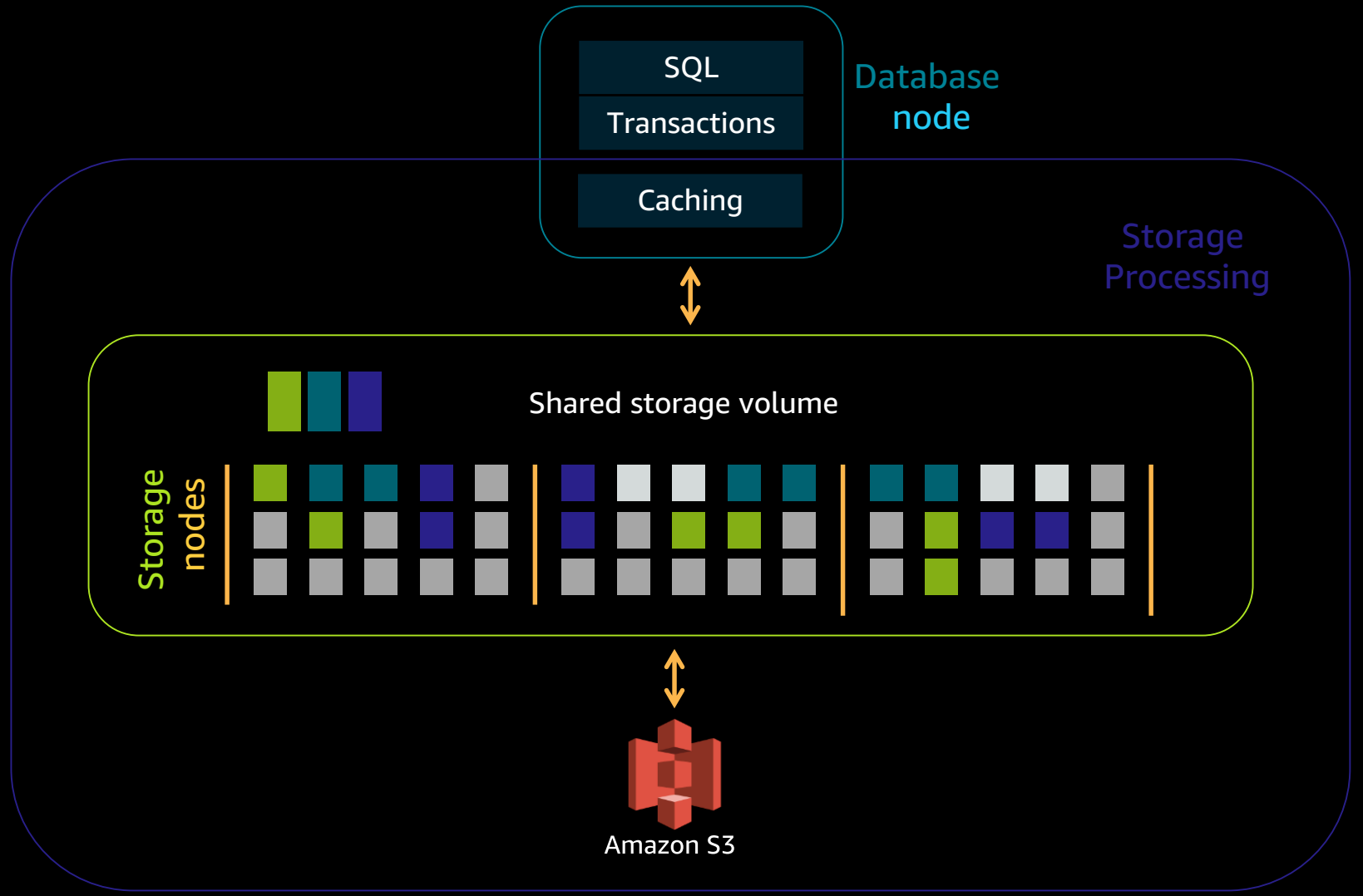
Agenda

- Architecture
- WAL Comparison with SQL Server
- Storage Architecture
- Read Replicas
- Aurora Failover
- Backups
- Global Database
- Fast Clones
- Monitoring

Architecture

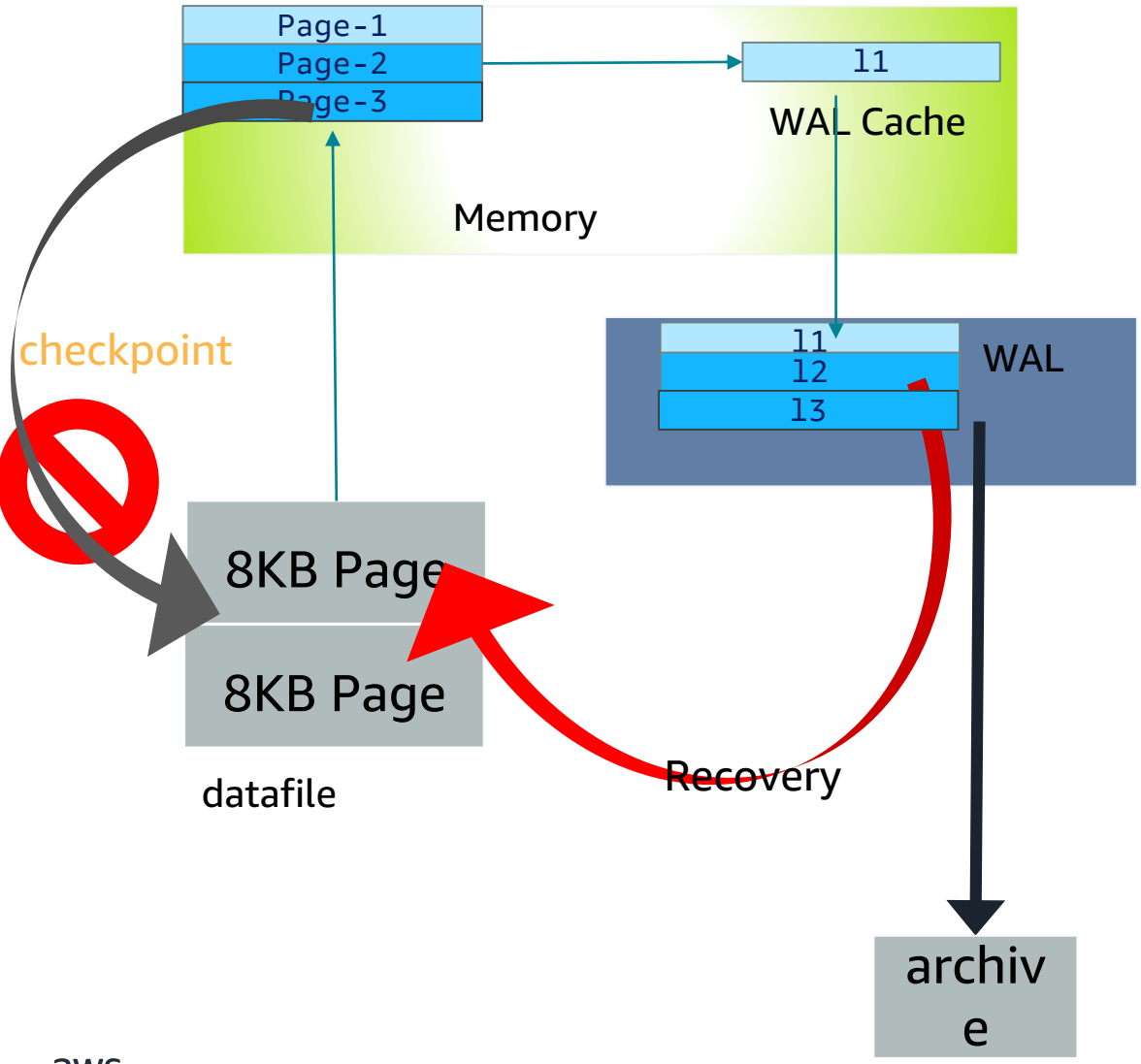
Aurora Architecture





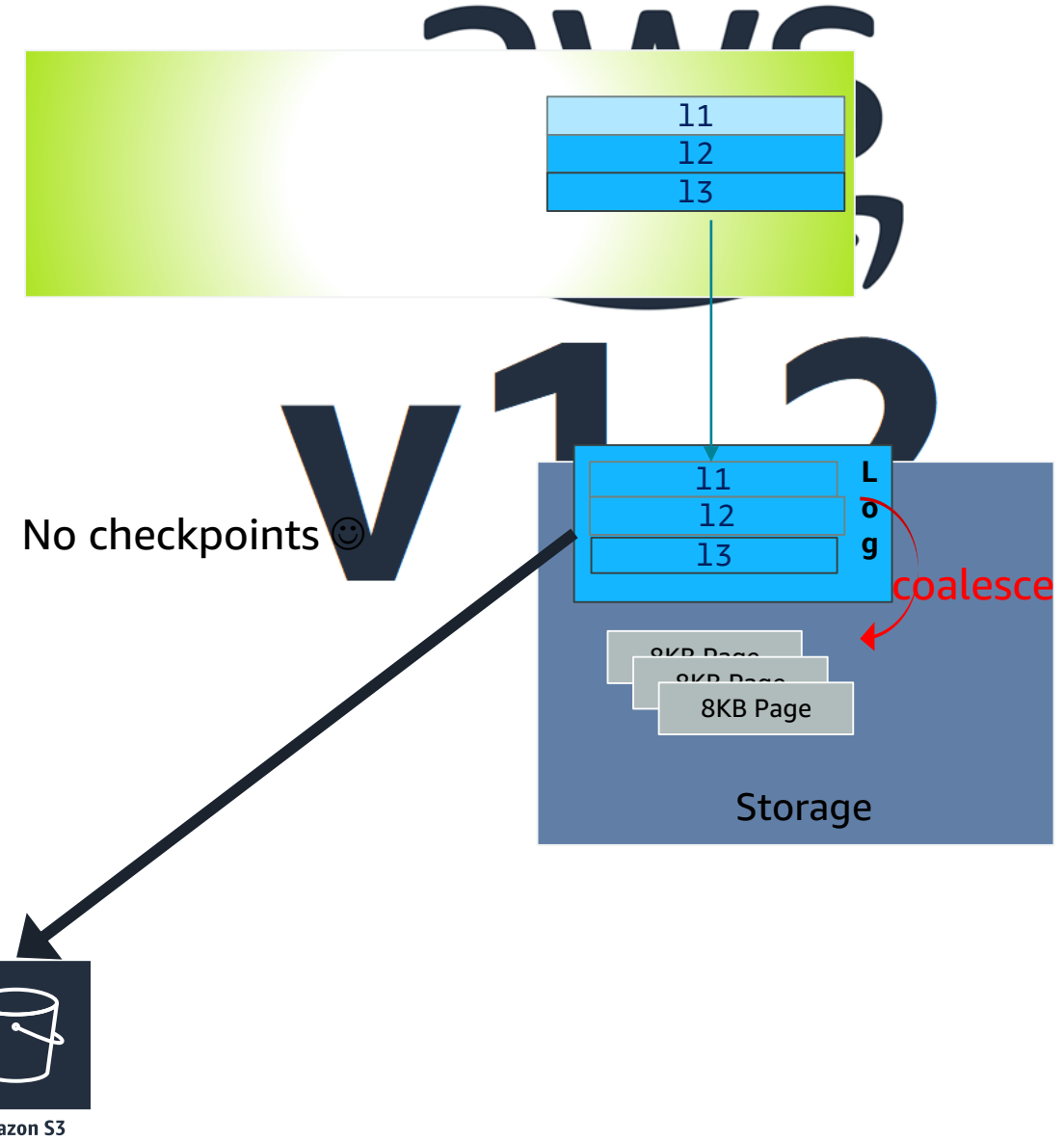
SQL Server

update my_table set my_col = 6;



Aurora

update my_table set my_col = 6;



Read Replicas

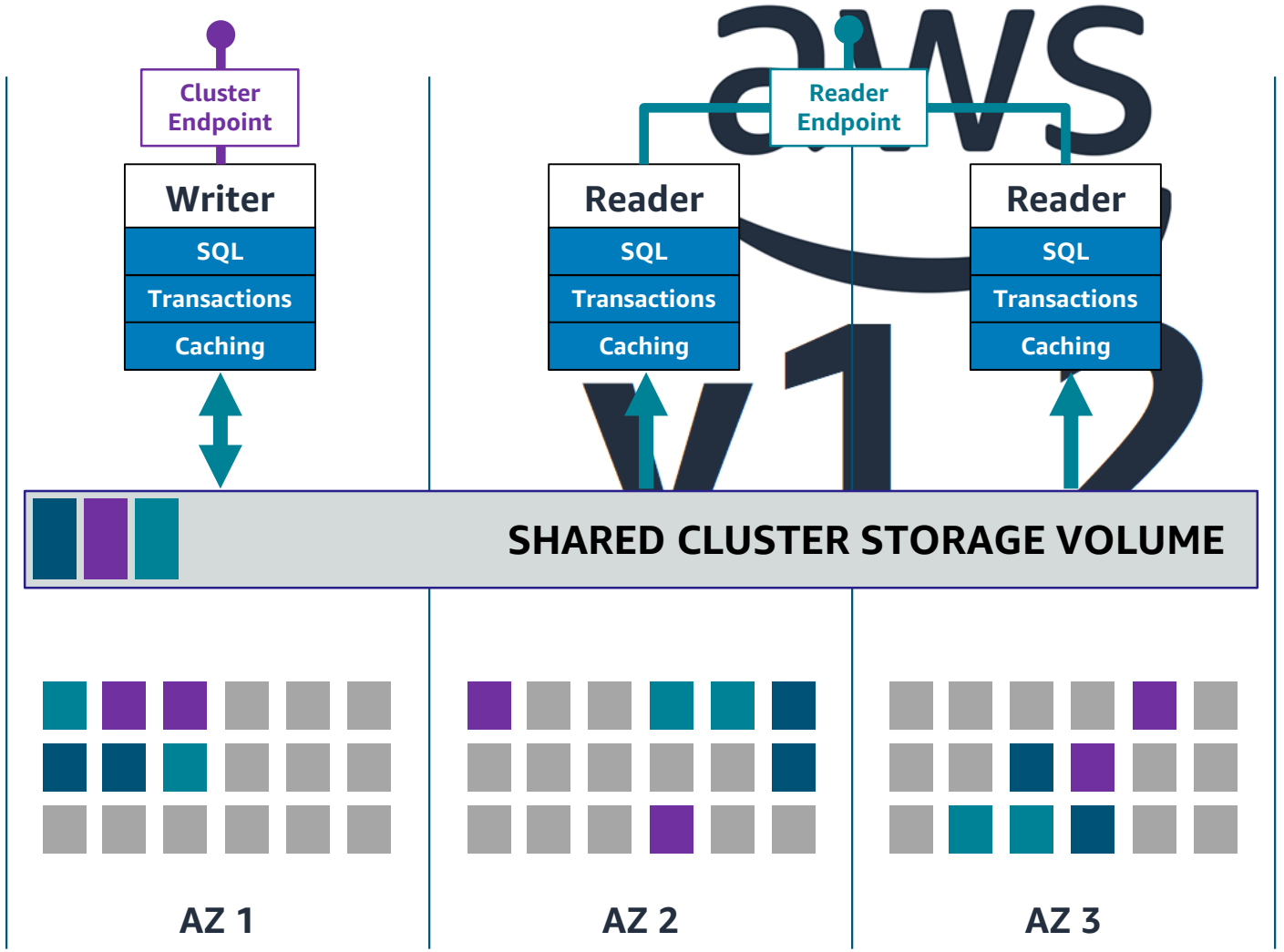
Read Replicas

Managed DB service, no OS or filesystem level access

Connect to writer using **Cluster (DNS) Endpoint** – always points to writer!

Round robin load balancing for reads using **Reader (DNS) Endpoint** (excludes writer)

Custom (DNS) Endpoints, read replica auto scaling (CPU & connections) supported as well



Aurora Failover

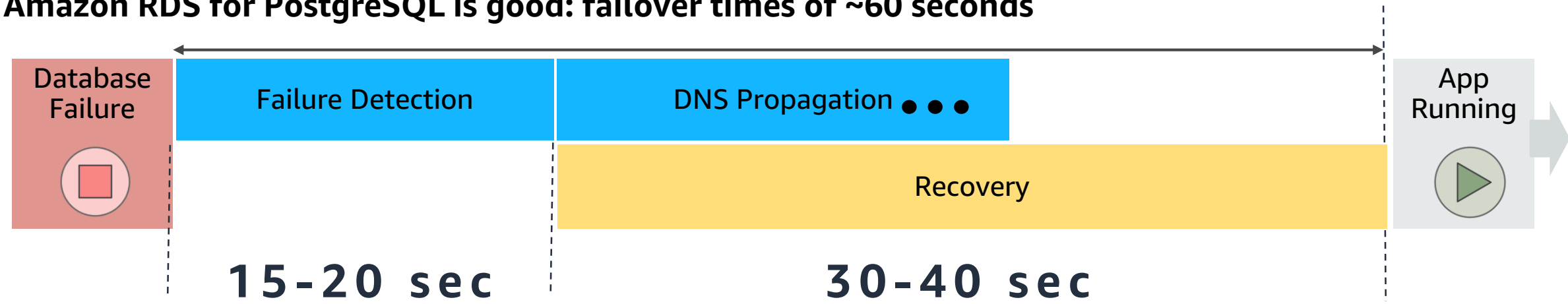
Aurora Failover

- When Writer becomes unavailable…
  - First attempt will be to restart the PostgreSQL process
  - If that fails, the failed Writer will be demoted to Reader and then a Reader will be promoted

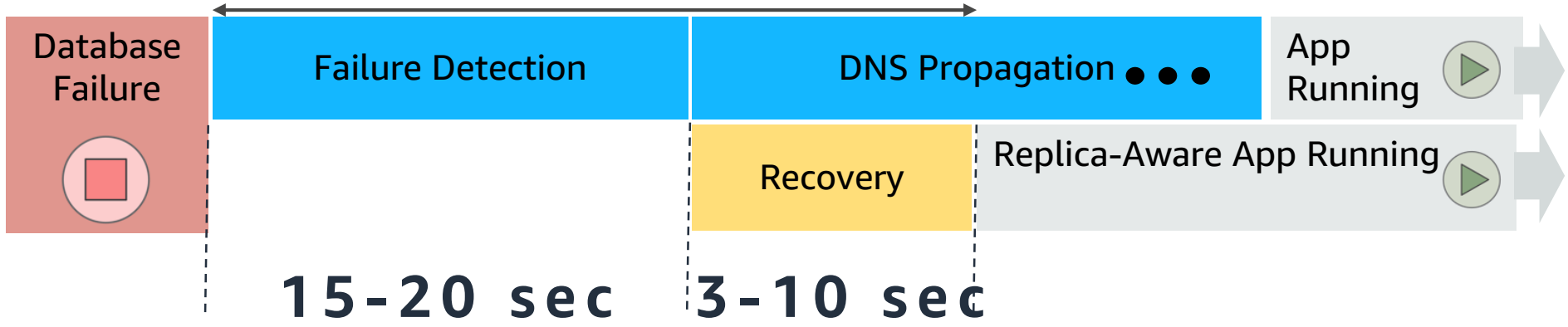
- What if there are no read replicas?
  - The restart will be attempted
  - If the writer node is not recoverable?
    - Then a rebuild will take place – downtime of up to 10 minutes
    - This is called “Host Replacement”

Faster, more predictable failover with Amazon Aurora

Amazon RDS for PostgreSQL is good: failover times of ~60 seconds

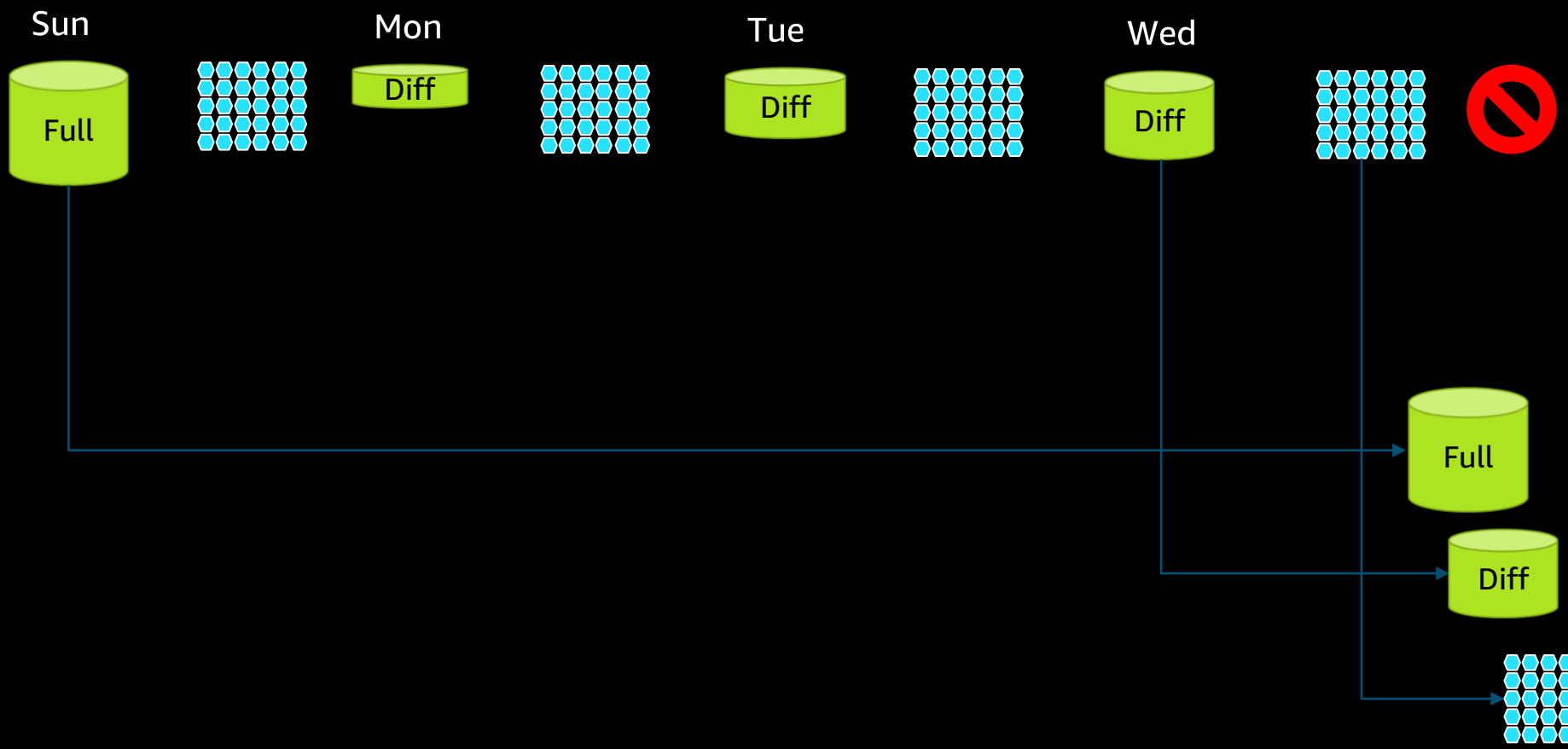


Amazon Aurora is better: failover times < 30 seconds

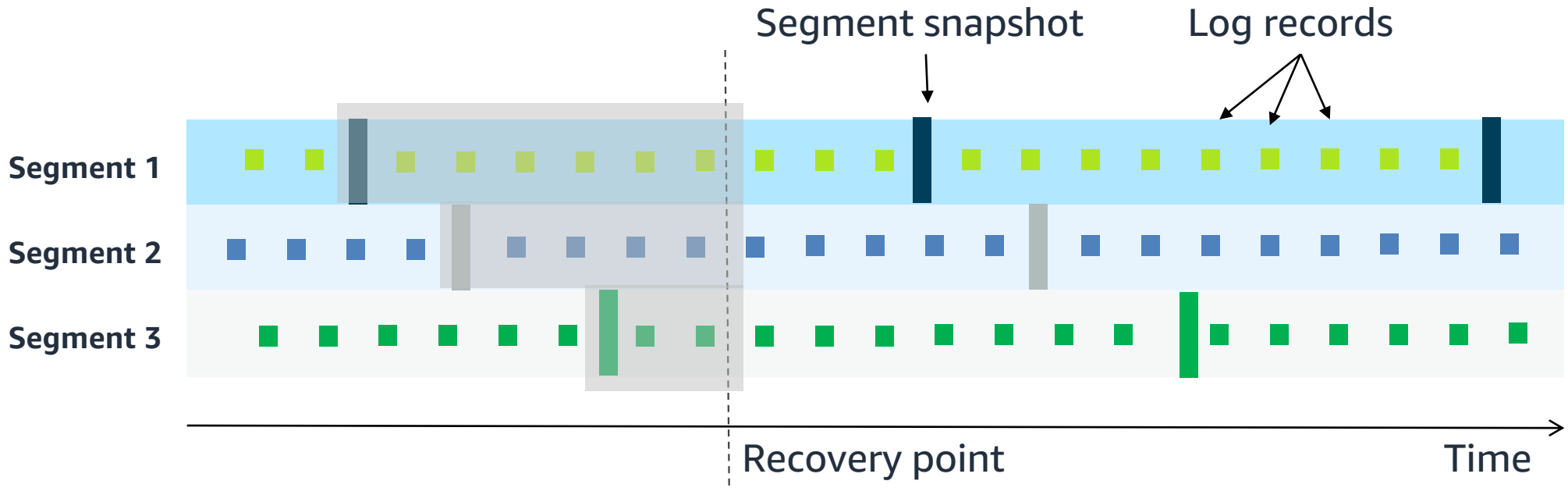


Backups

SQL Server Backups



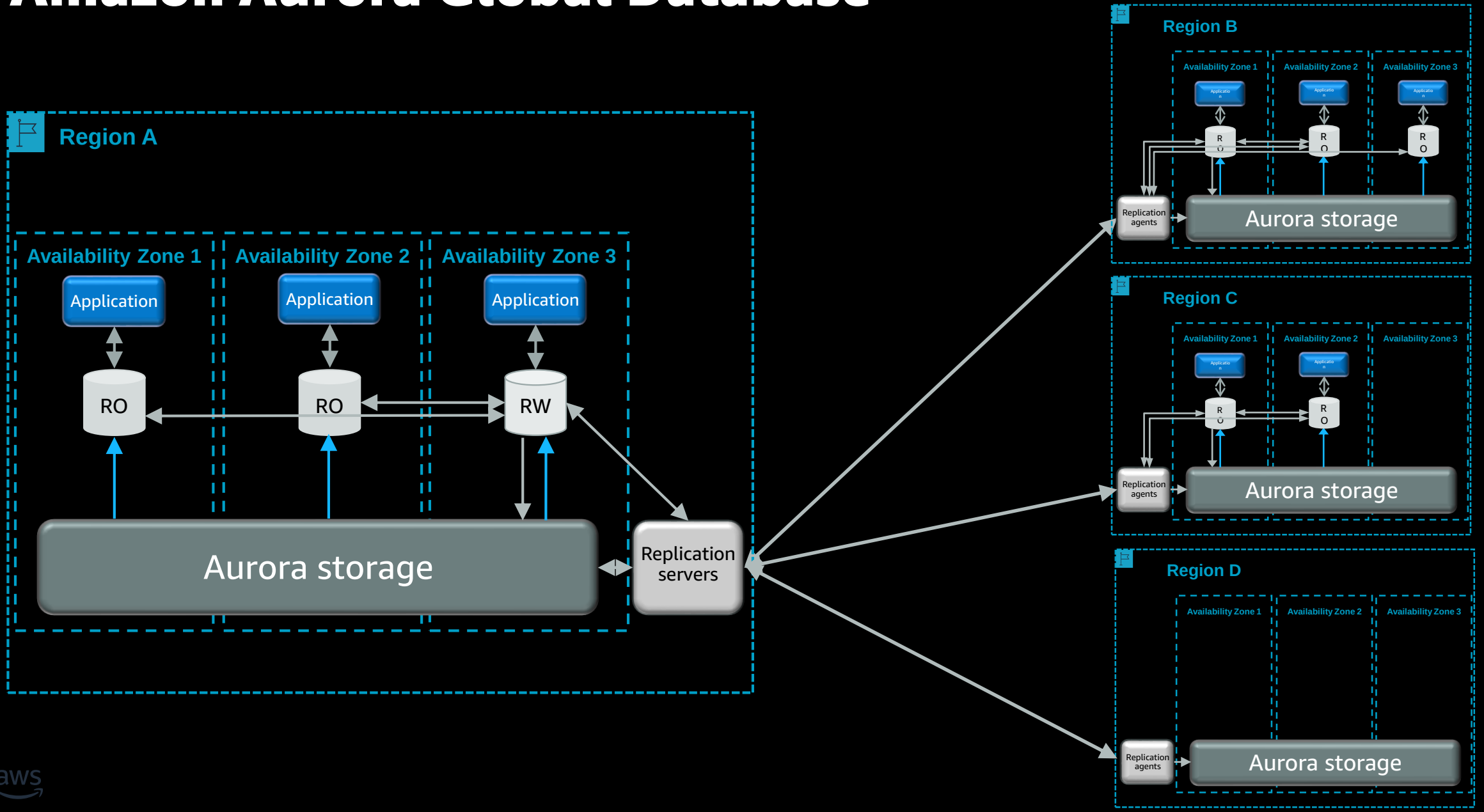
Amazon Aurora Continuous Backup



- Take periodic snapshot of each segment in parallel; stream the logs to Amazon S3
- Backup happens continuously without performance or availability impact
- At restore, retrieve the appropriate segment snapshots and log streams from S3 to storage nodes
- Apply log streams to segment snapshots in parallel and asynchronously

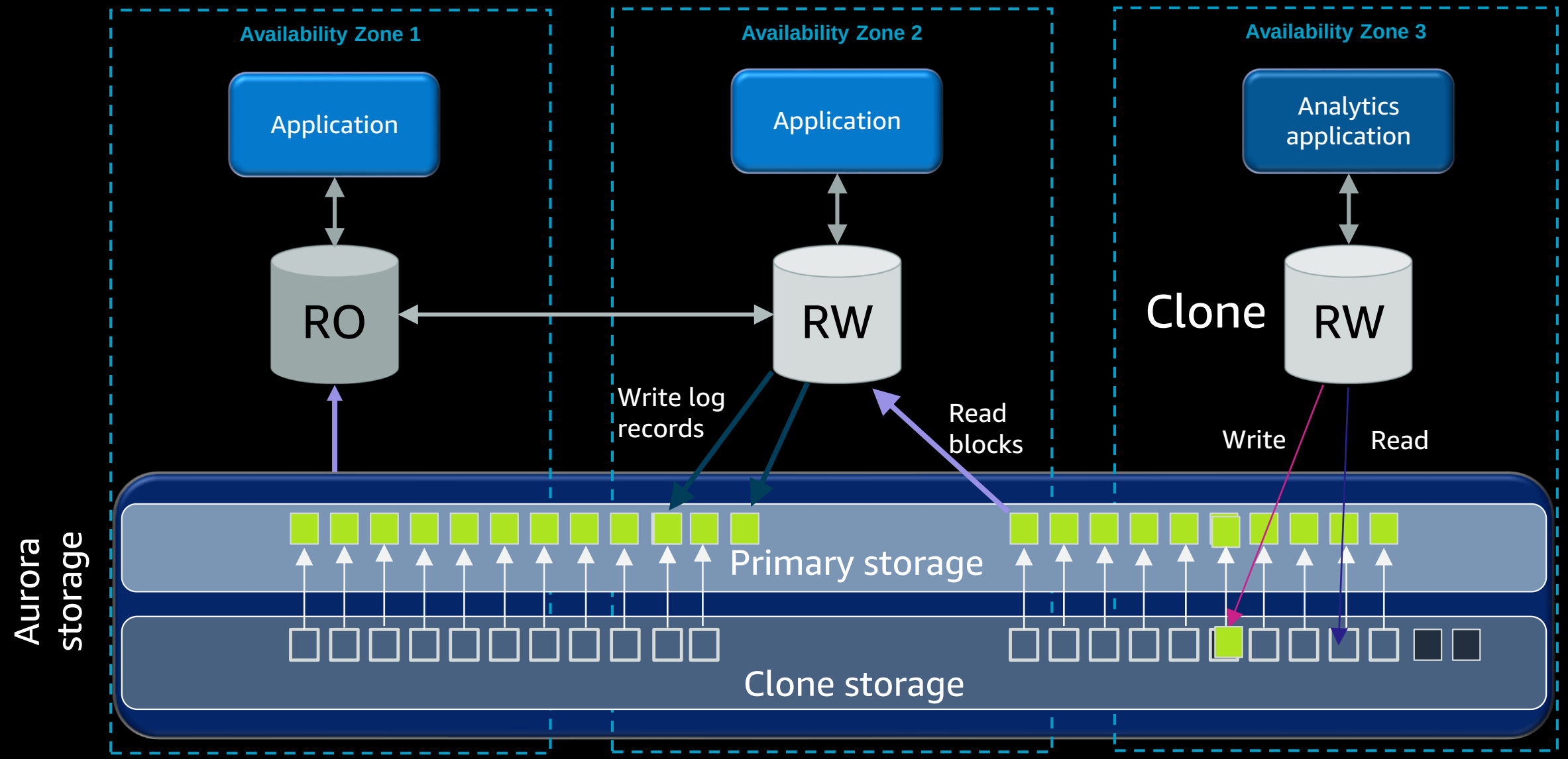
Global Database

Amazon Aurora Global Database



Fast Clones

Fast clones


Availability Zone 1
Availability Zone 2
Availability Zone 3
Application
Application
Analytics application
RO
RW
Clone
RW
Write log records
Read blocks
Write
Read
Aurora storage
Primary storage
Clone storage

Monitoring

Thank You