POSTGRESQL DISASTER RECOVERY AT SCALE: LESSONS FROM AMAZON RELATIONAL DATABASE SERVICE OPERATIONS

Andrei Dukhounik

Amazon Relational Database Service

Alisdair Owens

Amazon Relational Database Service

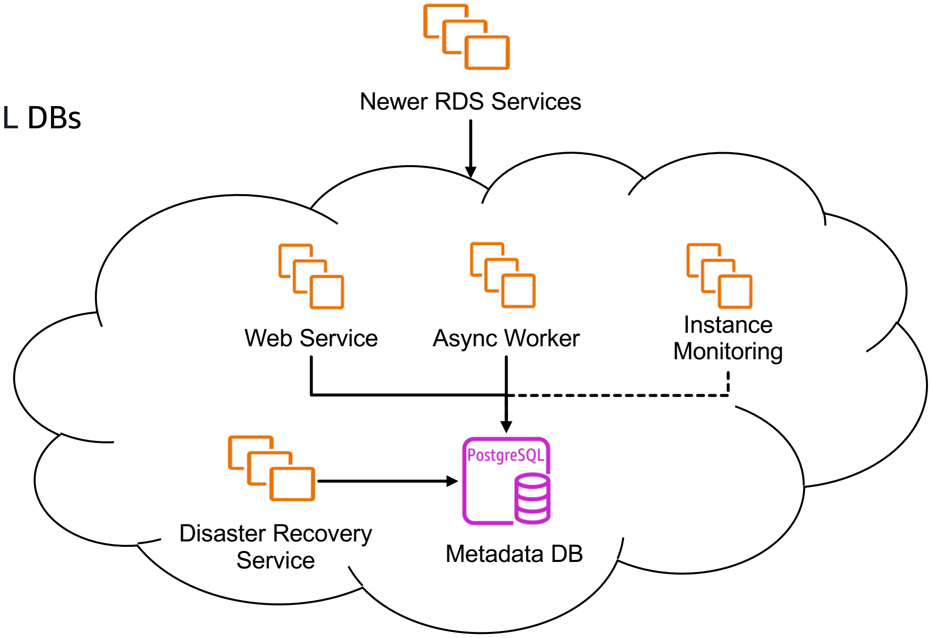
Agenda

- Introduction and background
- Database overload
- Hardware failures
- Logical and Physical corruption
- Operational readiness

Introduction and background

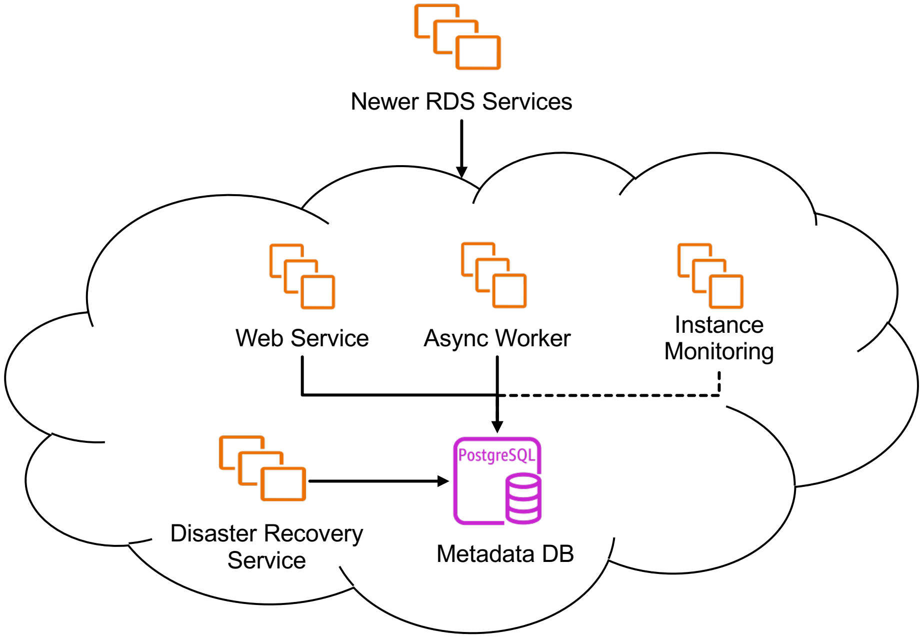
Introduction

- Amazon Relational Database Service (Amazon RDS) manages relational databases on behalf of an enormous number of customers.
- Including ourselves!
  - Core system backed by regional RDS for PostgreSQL DBs



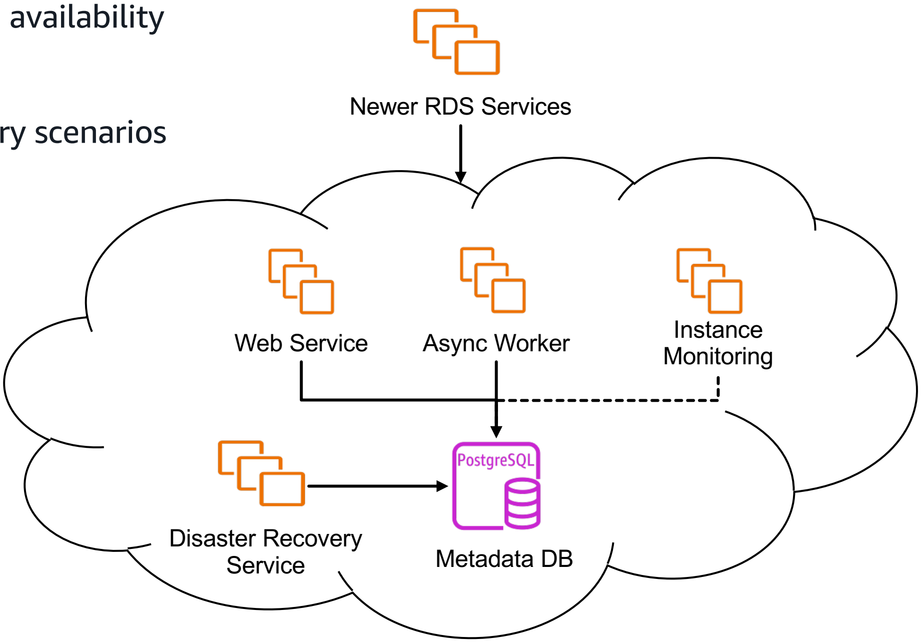
Self-Hosting

- We get a lot from RDS!
  - Health monitoring
  - Multi-AZ/failover
  - Reliable backups
  - Replica maintenance
  - etc



Self-Hosting

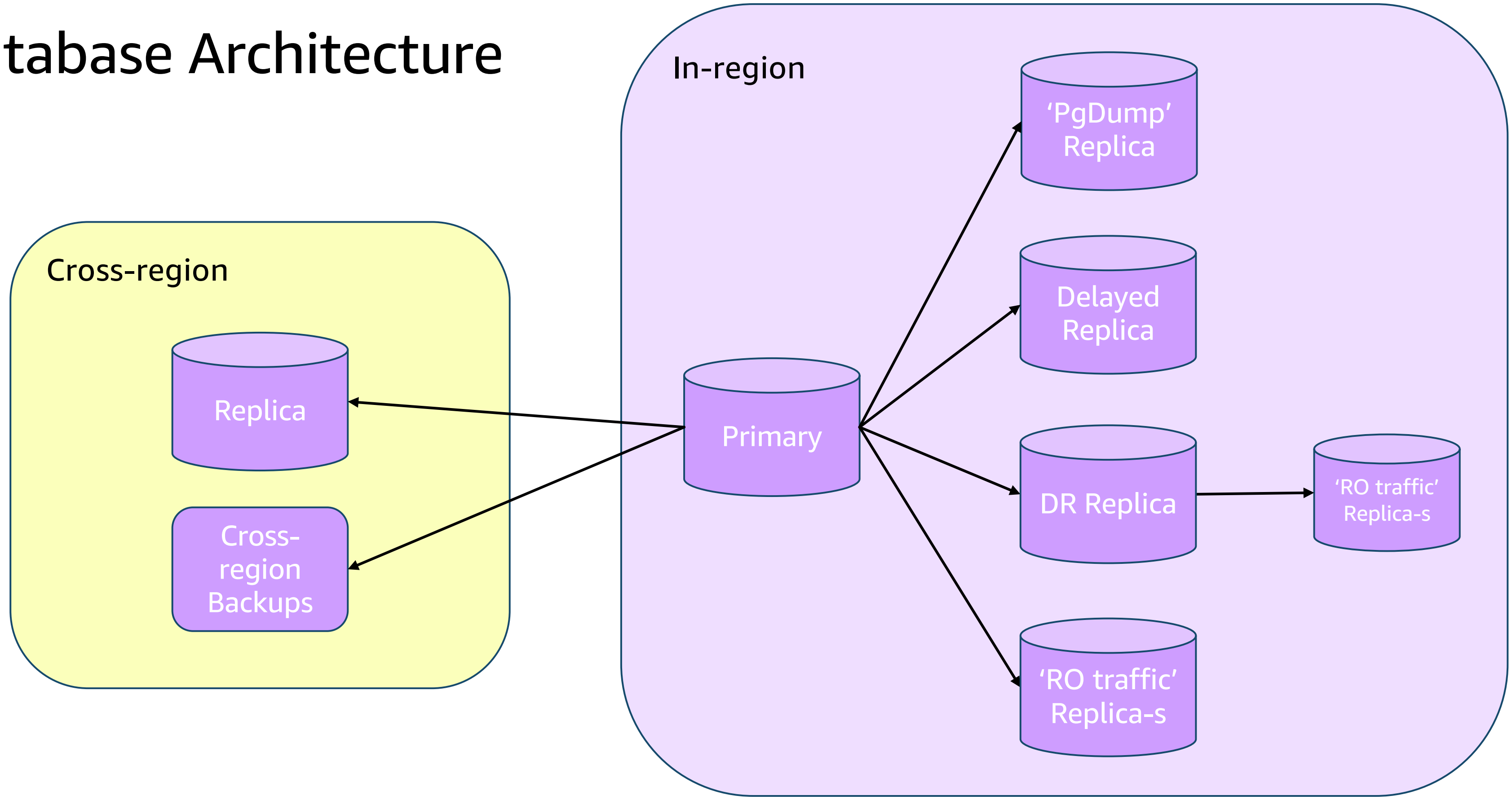
- We need supplements to safely self-host
  - Complex recovery scenarios rely on core metadata DB availability
  - Instance failover architected to be independent
  - Independent internal service to cover complex recovery scenarios



Scale

TPS	Size	Workload
Several 100K	Multi-TB	Mostly point operations Some complex queries

Database Architecture



Disaster scenarios

Disaster scenarios

Overload	Physical Failure	Data corruption
Traffic changes	Hardware failure	Physical corruption
DB performance cliffs	AZ/Region outages	Logical corruption

Database overload

Growth Management

Metrics	Optimizations	Scaling
CPU, IOPS, Storage, etc p99 or p100	Connect monitoring -> code	Vertical + Horizontal

Traffic Surges

Prevention

Throttling

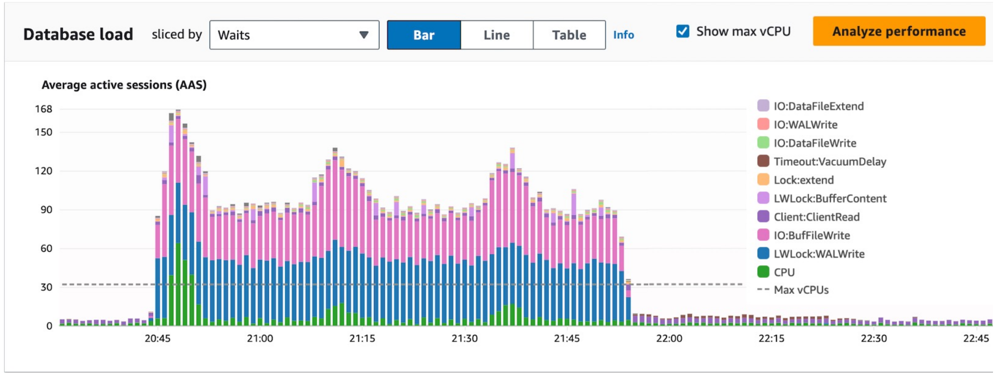
Load shedding

Mitigation

Rapid auto-scaling

Performance Cliffs

- Relational databases are not easily predictable at scale
- Query plan flip
- Savepoints
- LWLock contention
- Locks and Deadlocks



Database load sliced by Waits

☑ Show max vCPU

Hardware Failures and Corruption

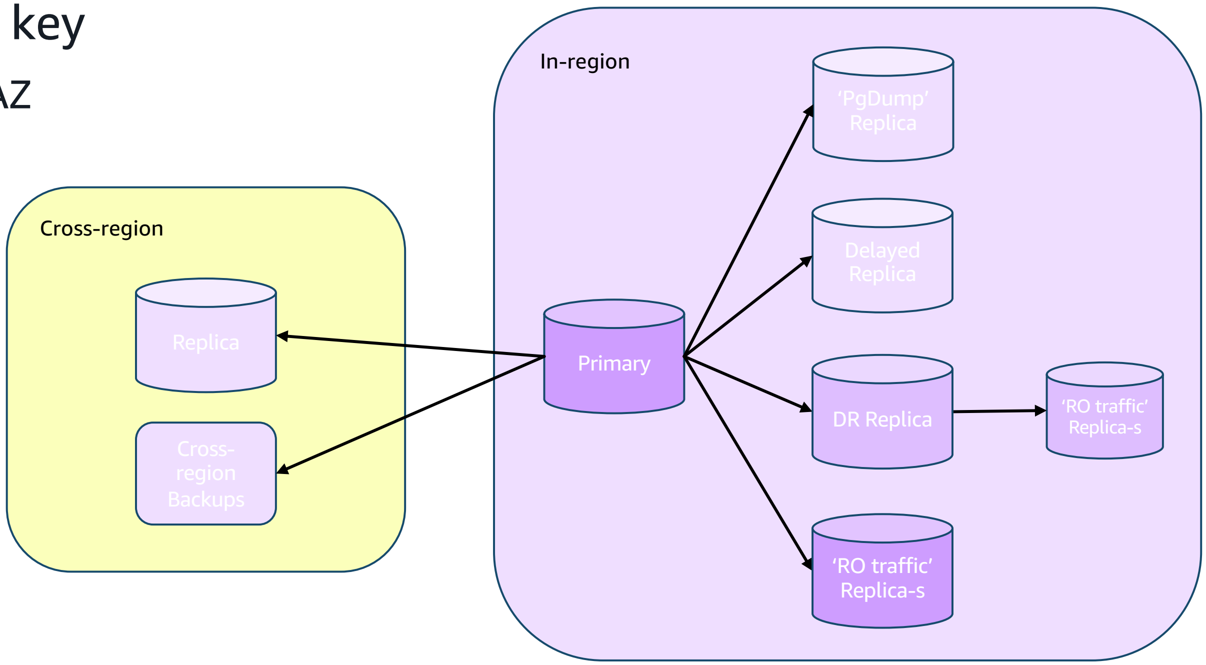
DR Capabilities

Recovery option	Hardware Failure	AZ Outage	Region Outage	Physical Corruption	Logical Corruption
Multi-AZ Failover	?	?	?	?	?
Replica Promotion	?	?	?	?	?
XR Replica	?	?	?	?	?
DML Change Log Recovery	?	?	?	?	?
Delayed Replica	?	?	?	?	?
PITR	?	?	?	?	?
XR PITR	?	?	?	?	?
Logical Dump	?	?	?	?	?

Hardware failures

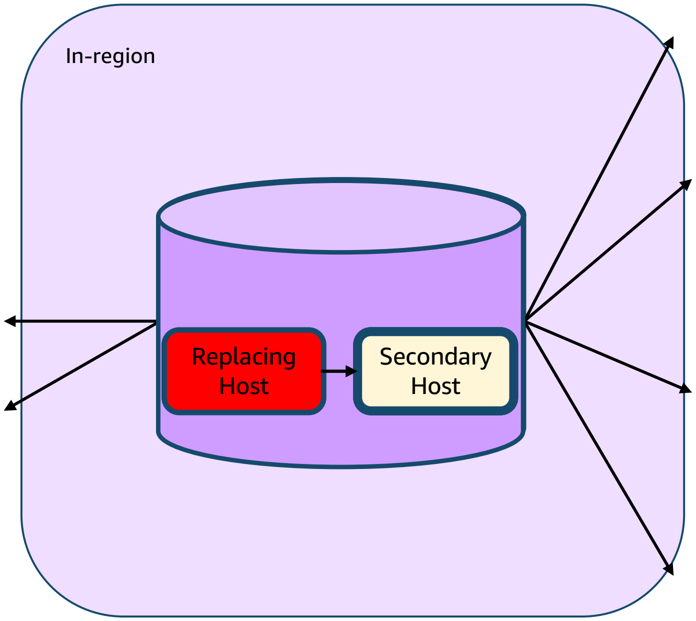
Hardware failures

- Storage → Hosts → Datacenter/AZ → Region
- Redundancy is the key
  - EBS replication in AZ
  - Multi-AZ



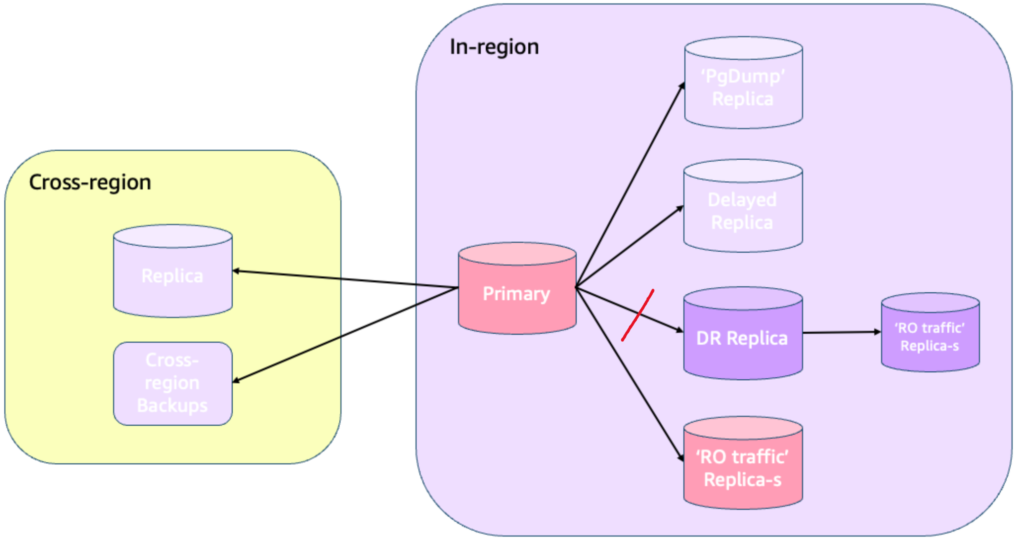
Multi-AZ deployment

- Multi-AZ/Failover
  - Failover within 60-120 sec
  - 2-3 failover per year across our fleet
- Circular dependency
- Other options?



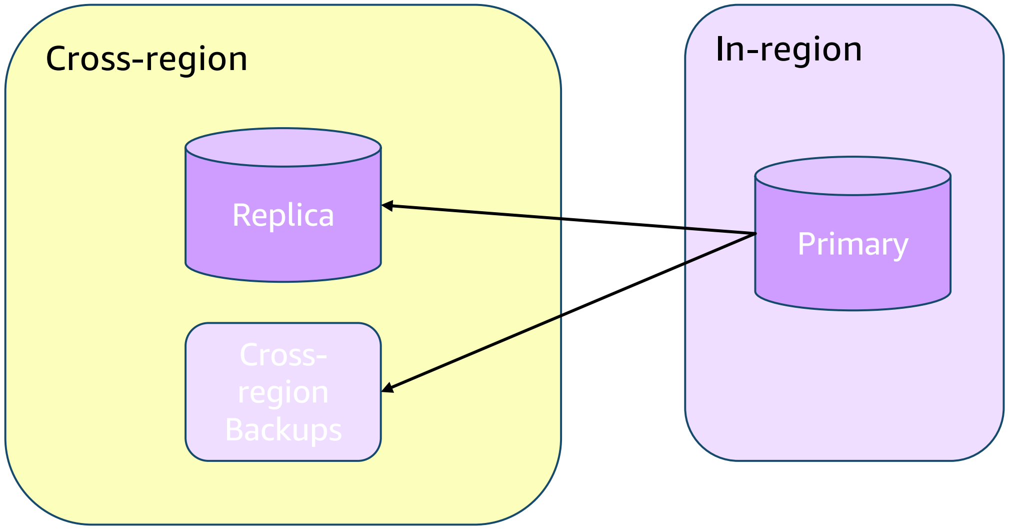
Read Replica Promotion

- In-sync, idle, same config
  - chained RRs
- When and how to promote?
  - Manual operation
  - Risk: async replication → data loss



Cross-Region Replica Recovery

- Managed by a different region
- Not a (likely) promotion target
  - High latency



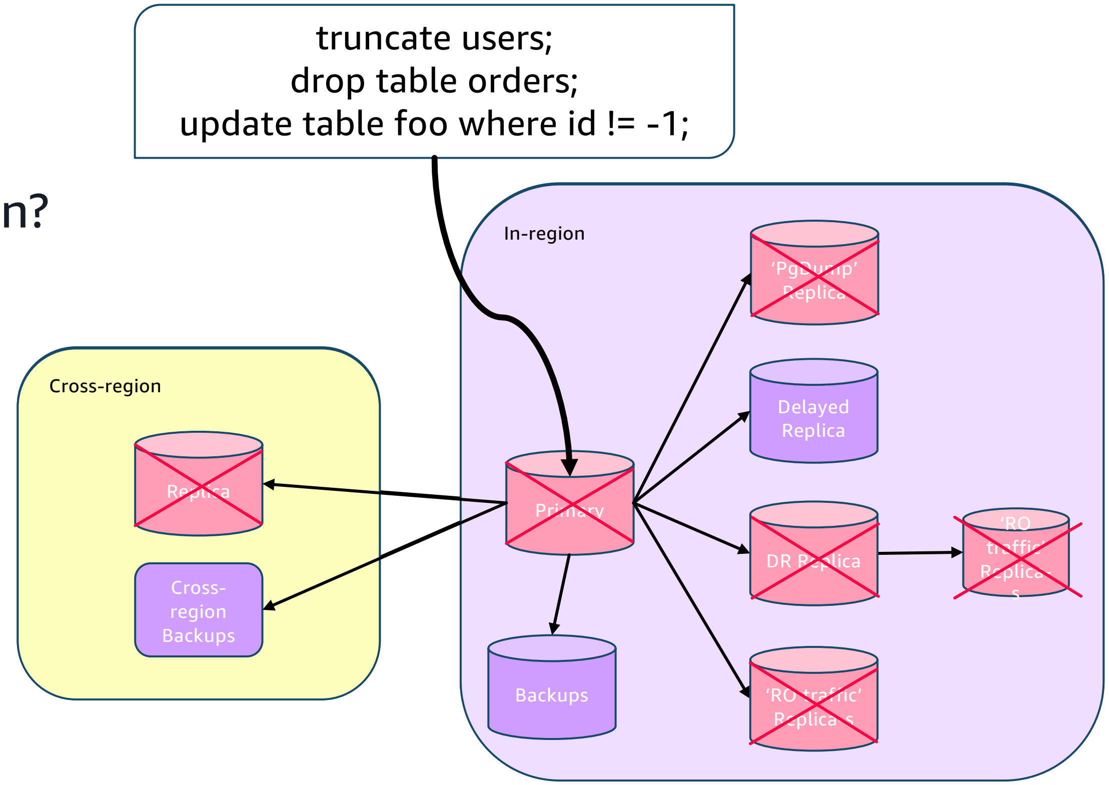
DR Option Summary

Recovery option	Hardware Failure	AZ Outage	Region Outage	Physical Corruption	Logical Corruption
Multi-AZ Failover	✅	✅	❌	❌	❌
Replica Promotion	✅	✅	❌	?	❌
XR Replica	✅	✅	✅	?	❌
DML Change Log Recovery	?	?	?	?	?
Delayed Replica	?	?	?	?	?
PITR	?	?	?	?	?
XR PITR	?	?	?	?	?
Logical Dump	?	?	?	?	?

Logical and Physical corruption

Logical corruption

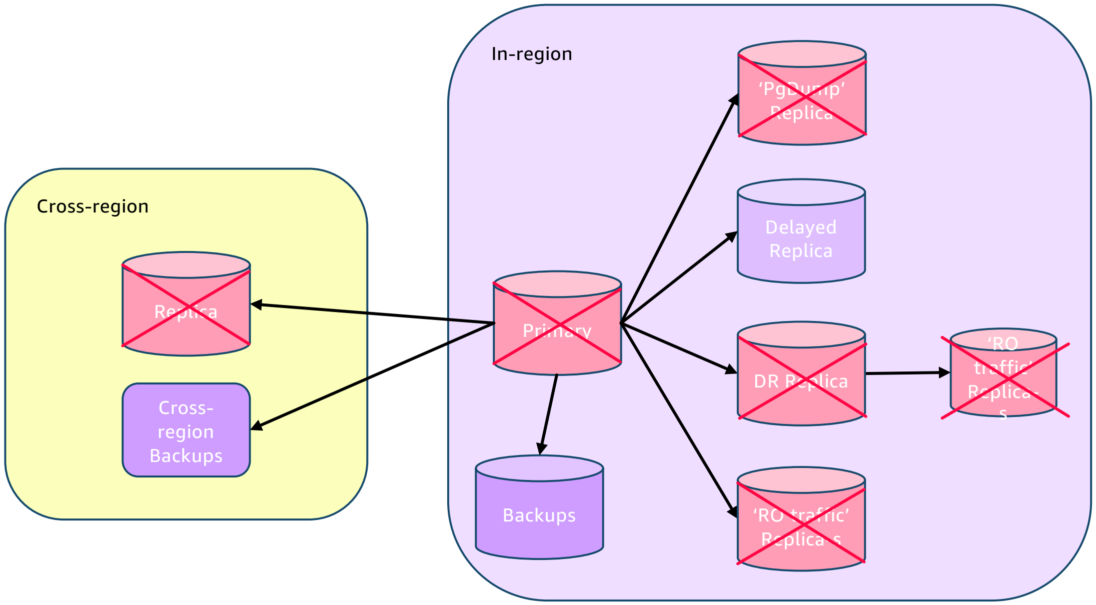
- What is logical corruption?
  - Not a problem for DB engine
- Source
- Impact
- How to recover?



truncate users;
drop table orders;
update table foo where id != -1;

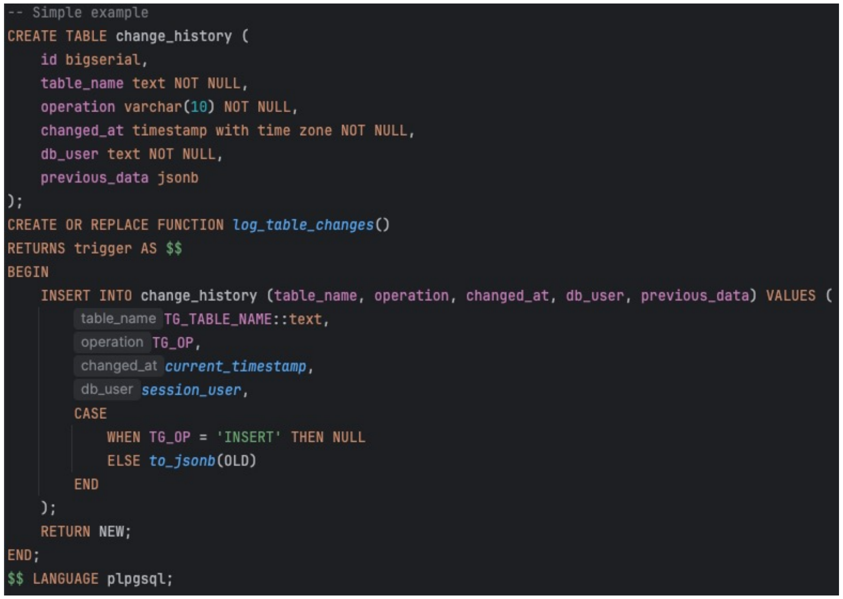
Point In Time Recovery

- How to get data for recovery?
- Backups
- Point In Time Restore
  - Multiple-attempts, but can be slow
- Cross-Region



Logical corruption: DML Change Log

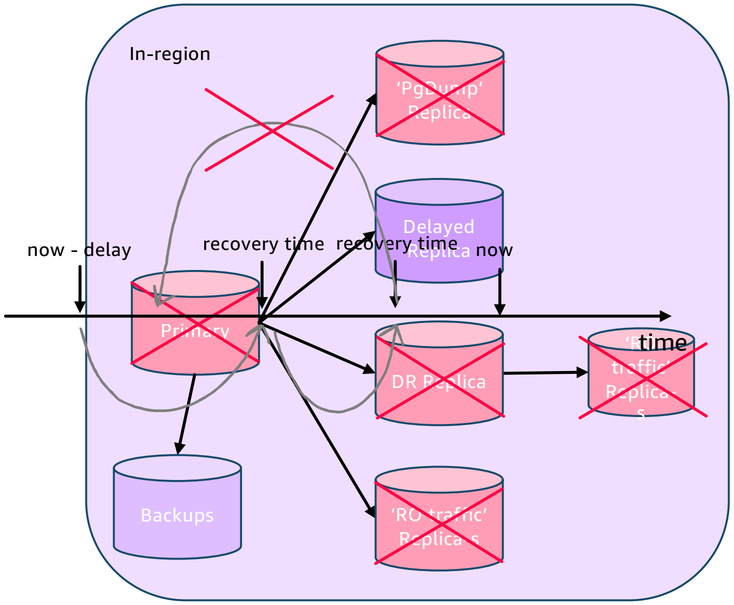
- Questions: When? What? Previous values?
- Implementation
  - Triggers → Audit table
- Forensic queries
- Small logical corruption recovery



```
-- Simple example
CREATE TABLE change_history (
    id bigserial,
    table_name text NOT NULL,
    operation varchar(10) NOT NULL,
    changed_at timestamp with time zone NOT NULL,
    db_user text NOT NULL,
    previous_data jsonb
);
CREATE OR REPLACE FUNCTION log_table_changes()
RETURNS trigger AS $$
BEGIN
    INSERT INTO change_history (table_name, operation, changed_at, db_user, previous_data) VALUES (
        table_name TG_TABLE_NAME::text,
        operation TG_OP,
        changed_at current_timestamp,
        db_user session_user,
        CASE
            WHEN TG_OP = 'INSERT' THEN NULL
            ELSE to_jsonb(OLD)
        END
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Delayed Replica Recovery

- Replica is always behind primary by X seconds
  - How to use for recovery?
- New RDS Postgres feature
- Move to anywhere between (*now* - delay, *now*]
  - Irreversible, can only move back to the future!
- Quicker than PiTR
- Watch out for storage



Logical corruption: Preventive measures

Database

FKs, Constraints and Triggers

Application

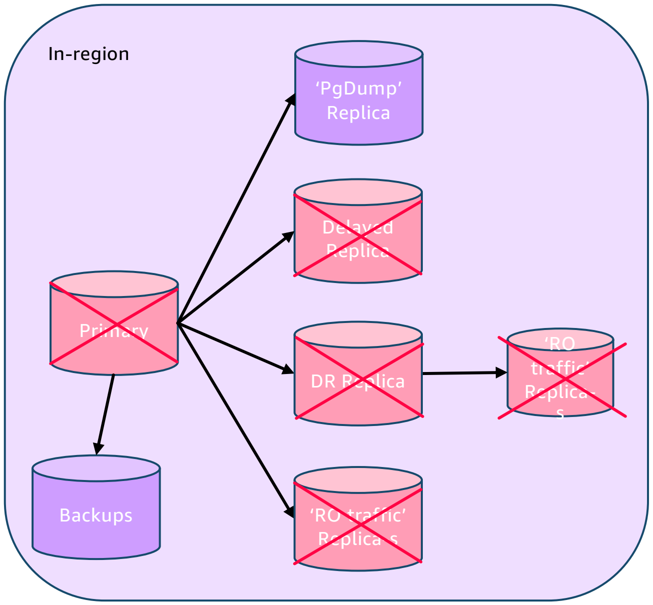
Secure queries, AuthN/AuthZ

Operations

Monitoring, rate limiting and circuit breakers

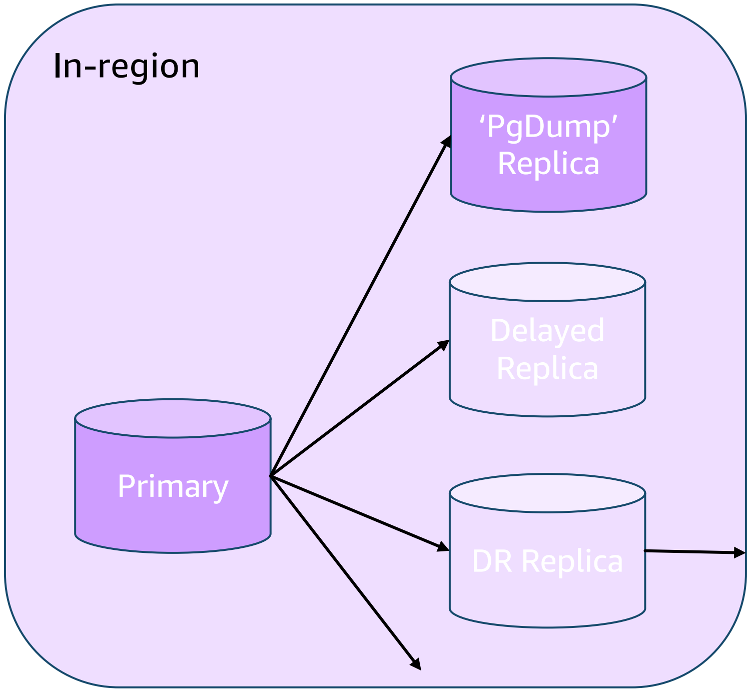
Physical corruption

- What is physical corruption?
  - A problem for DB engine
- How to detect?
  - Checksums, errors, query crashes
- How to recover?



Logical Dump Recovery

- Pg_dump every few hours
  - from dedicated replica
  - pause WAL → dump → resume → restore test
- No dependency on RDS
- But hours of data loss



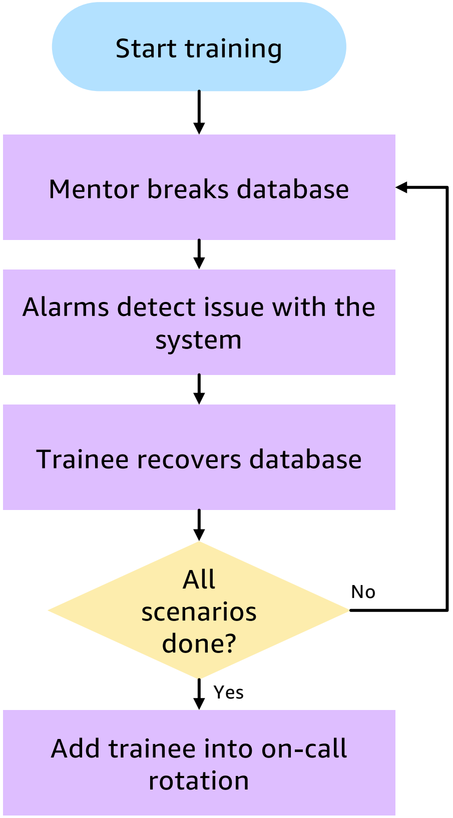
DR Options Summary

Recovery option	Hardware Failure	AZ Outage	Region Outage	Physical Corruption	Logical Corruption
Multi-AZ Failover	✅	✅	❌	❌	❌
Replica Promotion	✅	✅	❌	?	❌
XR Replica	✅	✅	✅	?	❌
DML Change Log Recovery	❌	❌	❌	❌	✅
Delayed Replica	❌	❌	❌	✅	✅
PITR	✅	✅	❌	✅	✅
XR PITR	✅	✅	✅	✅	✅
Logical Dump	❌	❌	✅	✅	✅

Operational readiness

Operational readiness

- Standard: backups, security and monitoring
- Disaster recovery automated tests
- Operator training


Start training
Mentor breaks database
Alarms detect issue with the system
Trainee recovers database
All scenarios done?
No
Yes
Add trainee into on-call rotation

Operational readiness: Trainings scenarios

Scenario	Simulation	Diagnose	Trainee’s task
Failover	Force Multi-AZ failover	App logs and metrics	Confirm failover & recovery
Overload	Drop index + API spam	App vs db metrics	Throttle API traffic
Locking	Lock critical table	App metrics and pg_stat_activity	Terminate blocking query
Logical corruption	Delete or update critical data	App logs and metrics and db data	Choose recovery method and restore data

Thank you!

Andrei Dukhounik
dukhouni@amazon.com

Alisdair Owens
alow@amazon.com