AWS re:Invent

AMZ302

Amazon Wallet: Increasing performance with DynamoDB

Jagan Cheboluraghava
Senior Software Dev Manager
Amazon, Payment Services

Artem Pliasunov
Senior Software Dev Engineer
Amazon, Payment Services

Agenda

Overview of Amazon Wallet

Why did we move to Amazon DynamoDB?

How did we implement migration with zero downtime?

How did Wallet benefit by migrating to Amazon DynamoDB?

Q&A

Amazon Wallet

Your Amazon Wallet

An overview of your payment methods, settings and subscriptions with Amazon.

Your default purchase preference

Change Preference

ADDRESS

Your default preferences are used for Alexa, Kindle, and other digital purchases.

NICKNAME	PAYMENT METHOD	SHIPPING METHOD
Purchase Preference	VISA Visa ending in 1111	Standard Shipping

1-Click settings

Manage backup payment method settings

Edit payment method for a current order

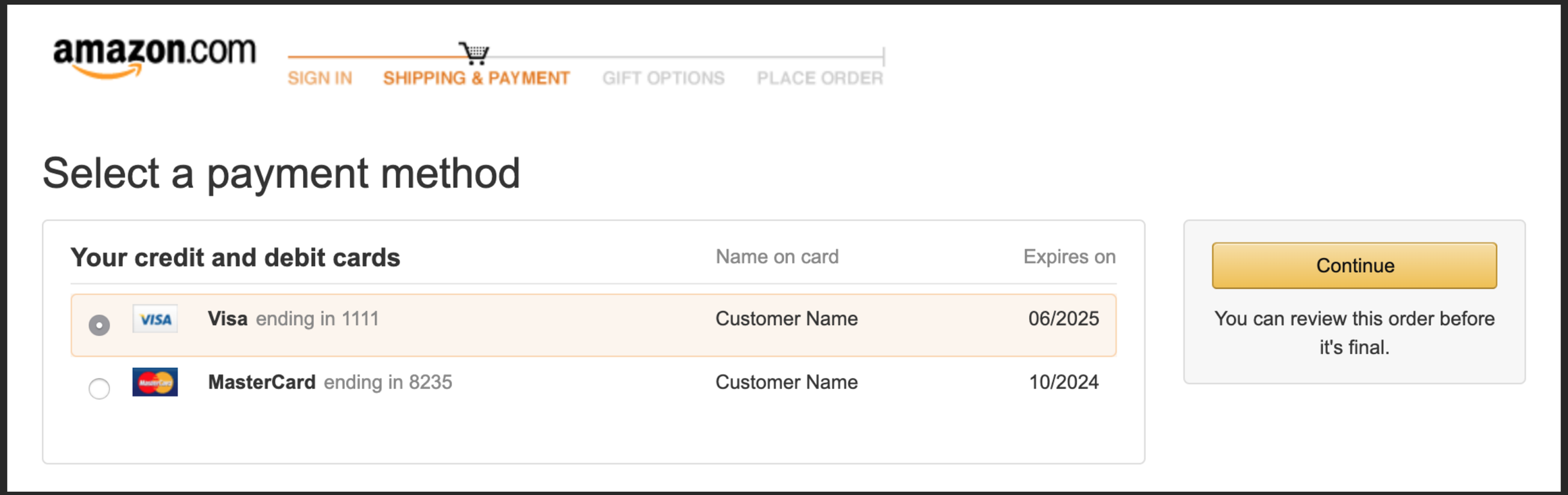
Your credit and debit cards

	Expires
VISA Visa ending in 1111	06/2025
MasterCard ending in 8235	10/2024

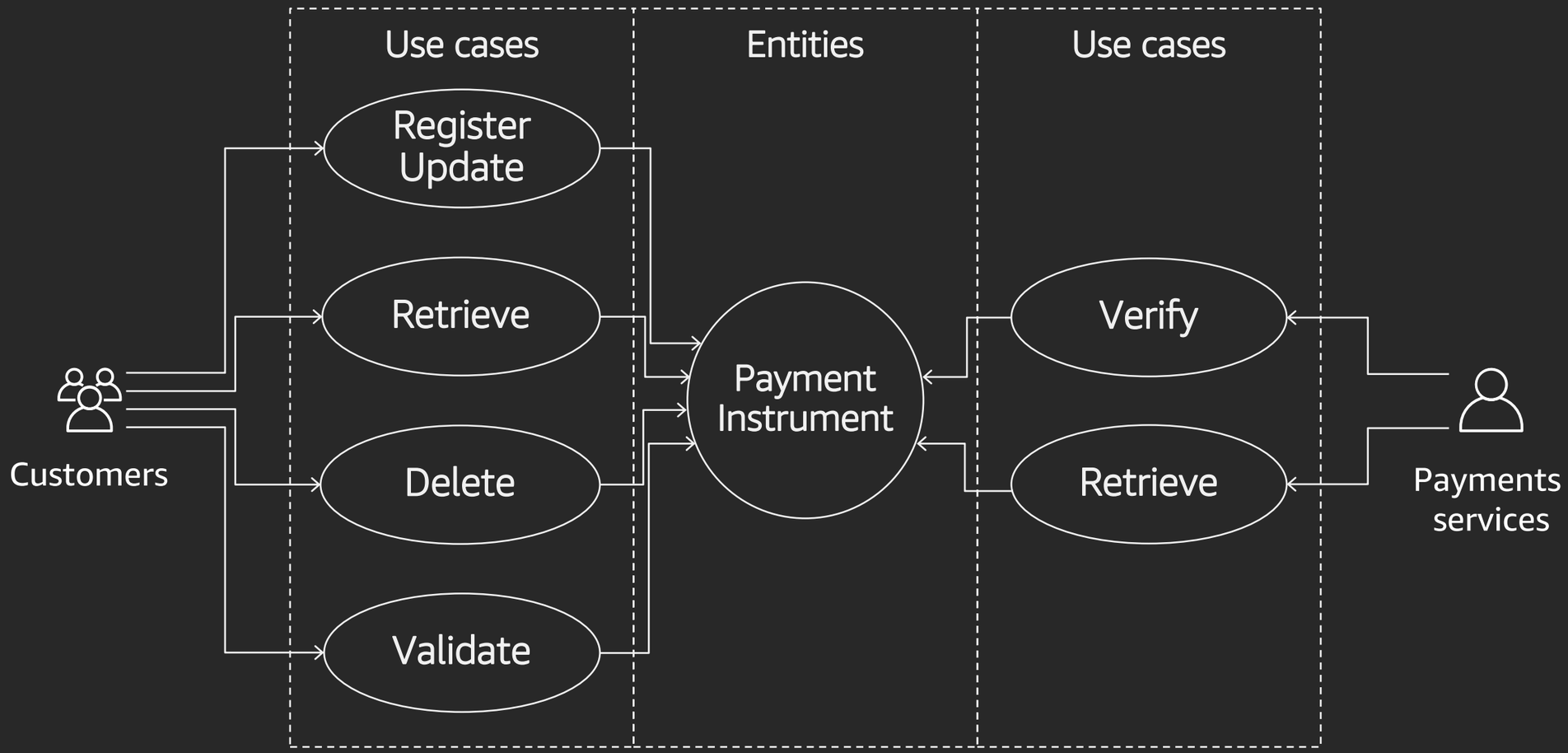
Your balances

Amazon Gift Card Reload your balance | Redeem a gift card	$0.00

Wallet enables customers to pay for their orders



Wallet’s functionality

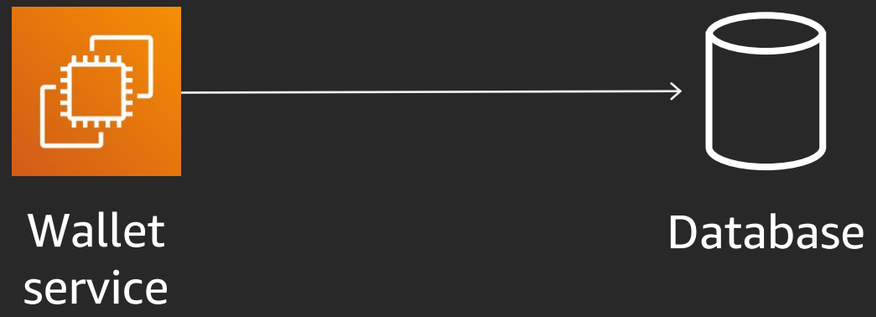


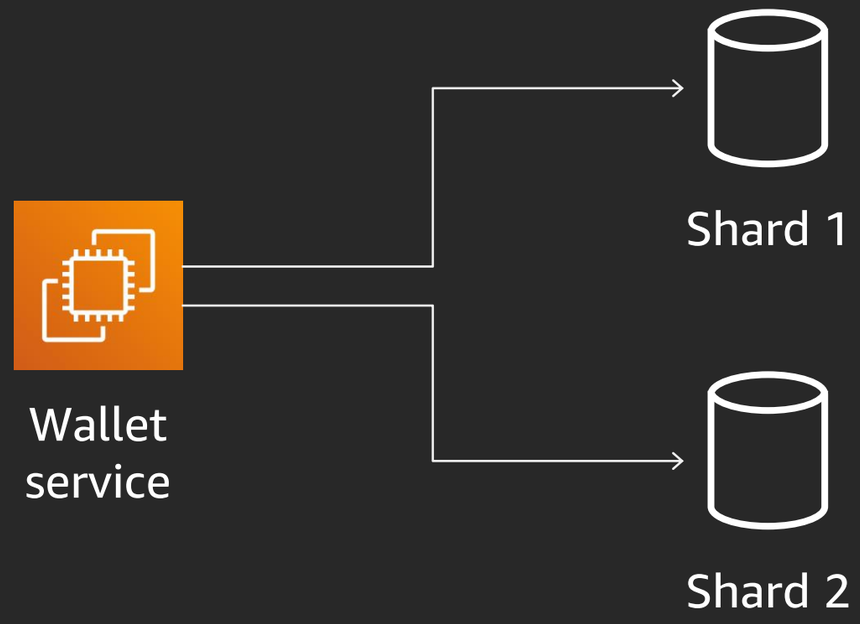
Keeping up with exponential growth

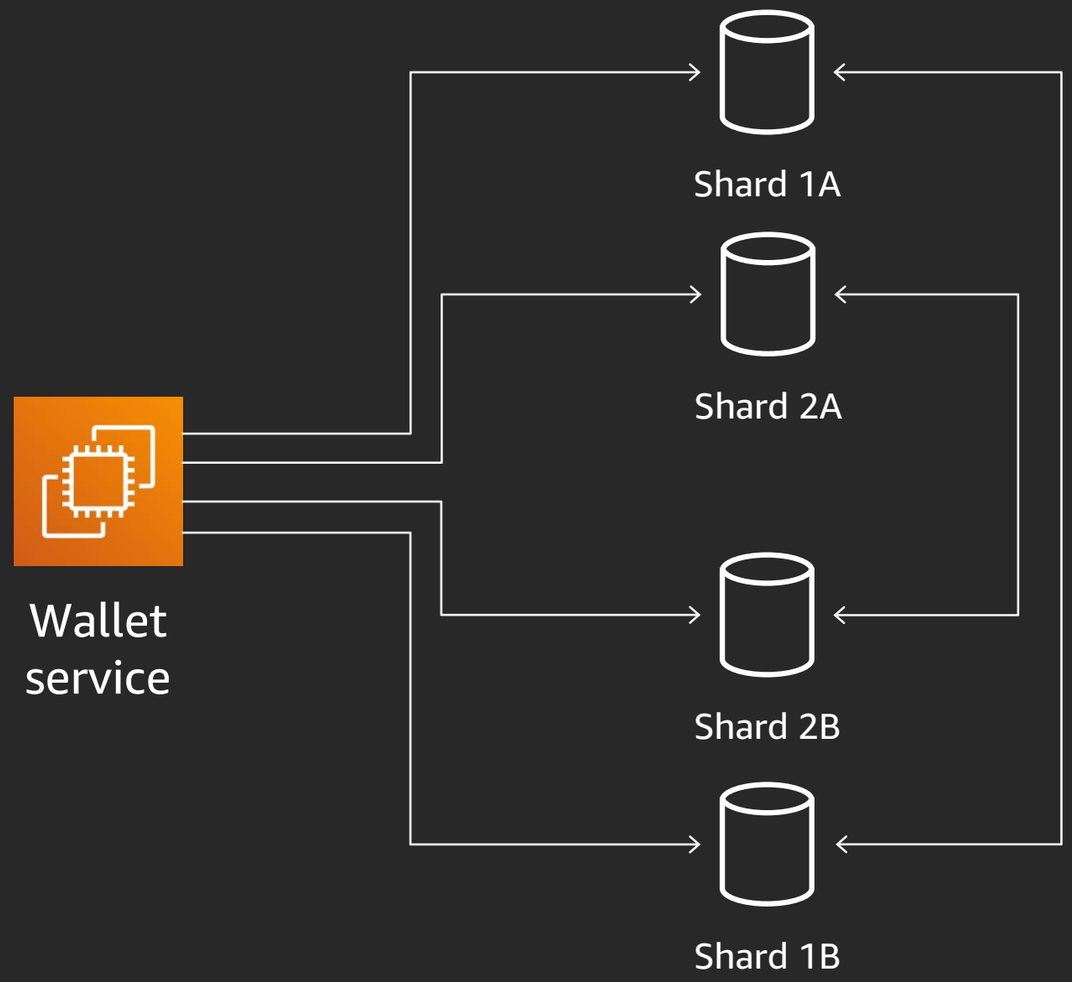
- 5 billion requests daily
- 10 billion records
- Exponential growth year over year

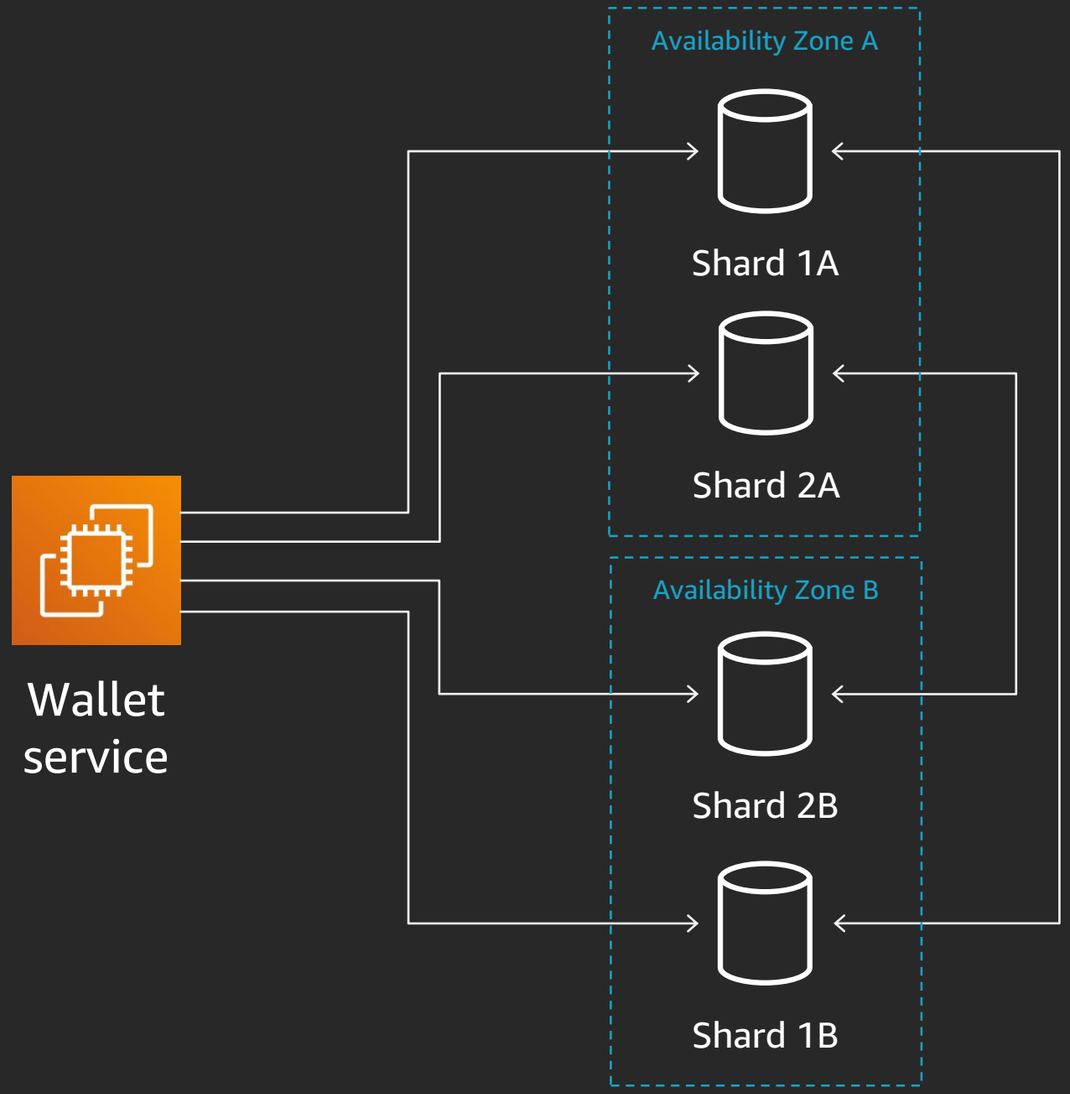
Legacy architecture

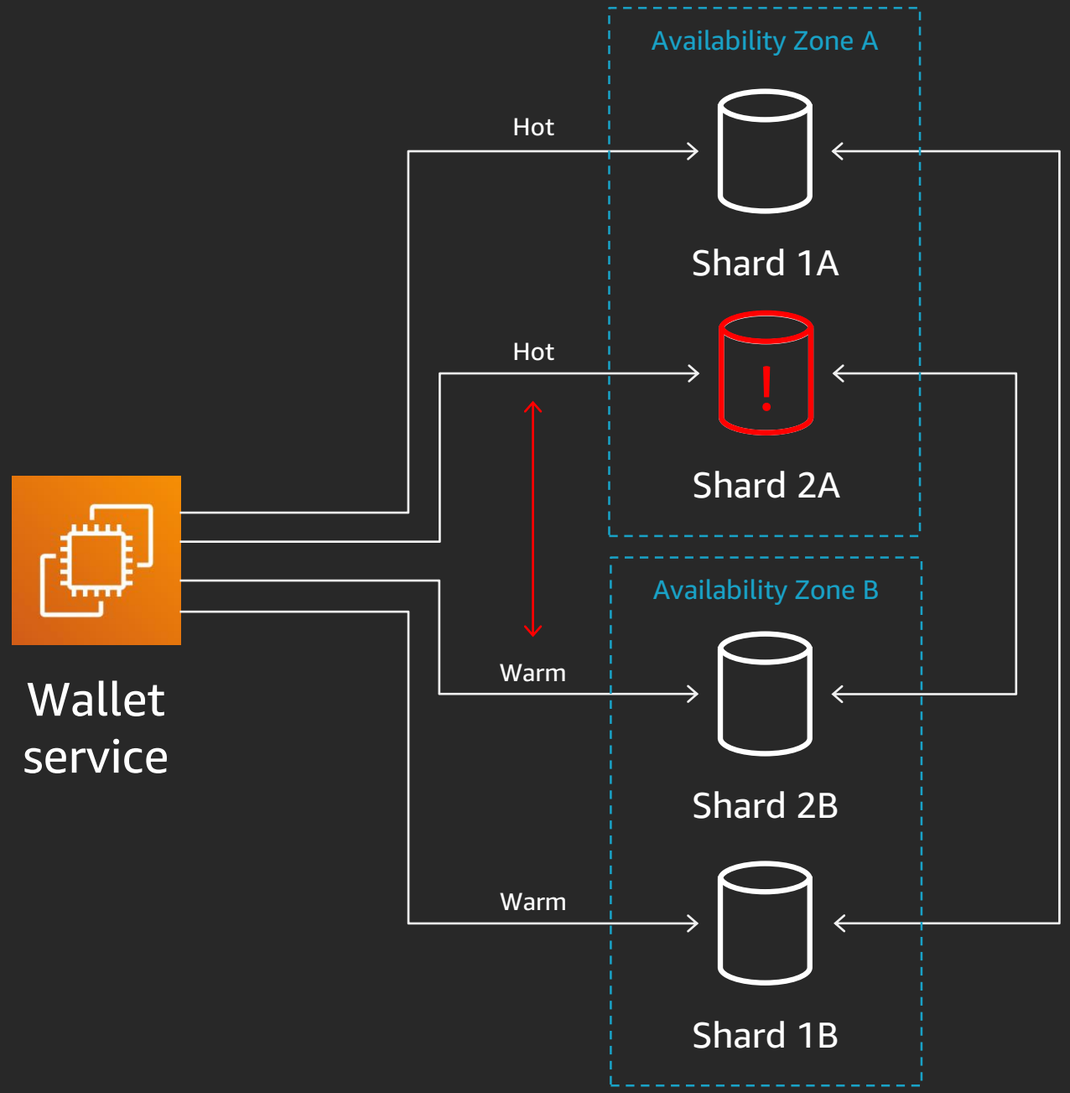

Wallet
service
Database

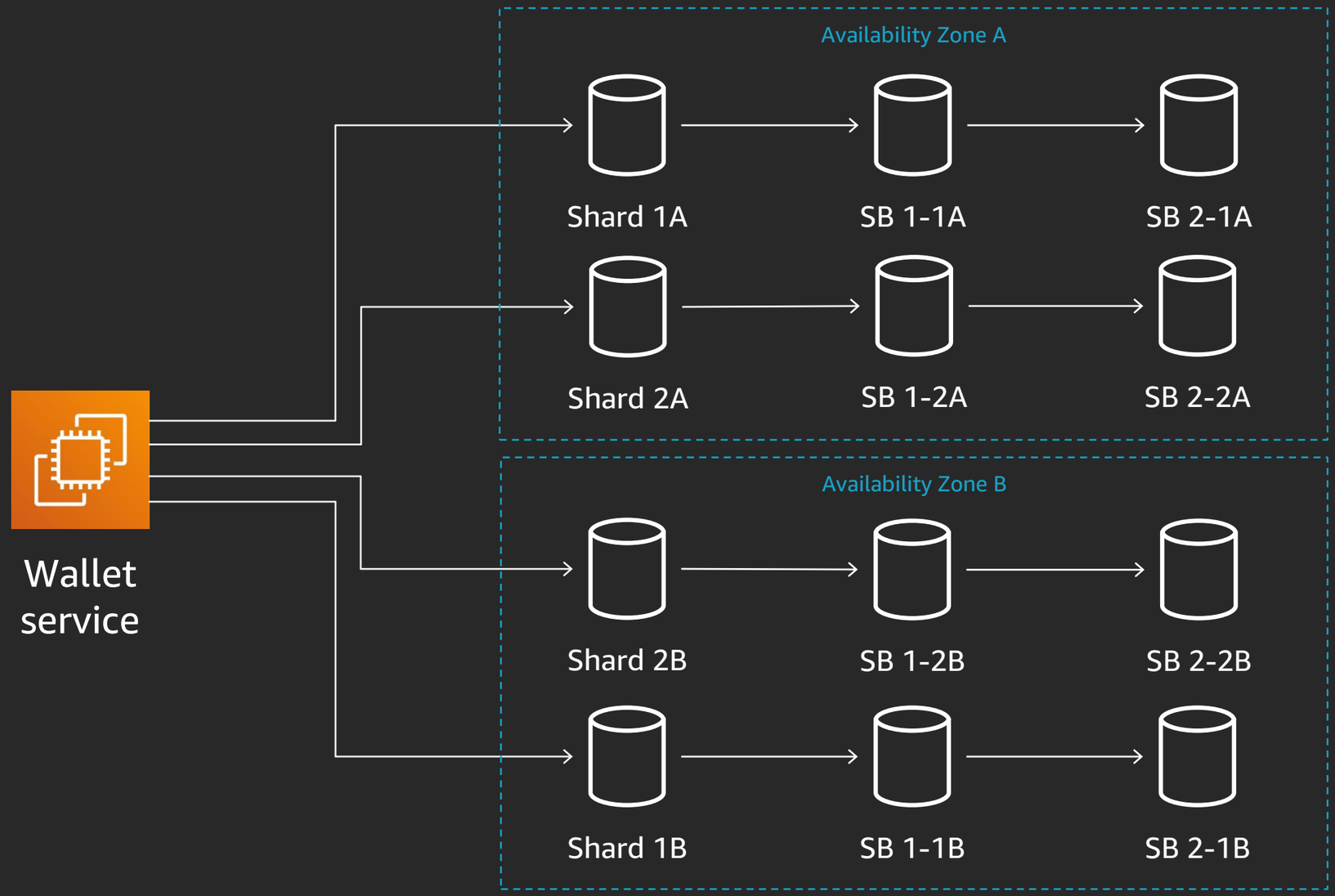

Wallet service
Database


Shard 1
Wallet service
Shard 2


Shard 1A
Shard 2A
Wallet service
Shard 2B
Shard 1B


Availability Zone A
Shard 1A
Shard 2A
Availability Zone B
Shard 2B
Shard 1B
Wallet service

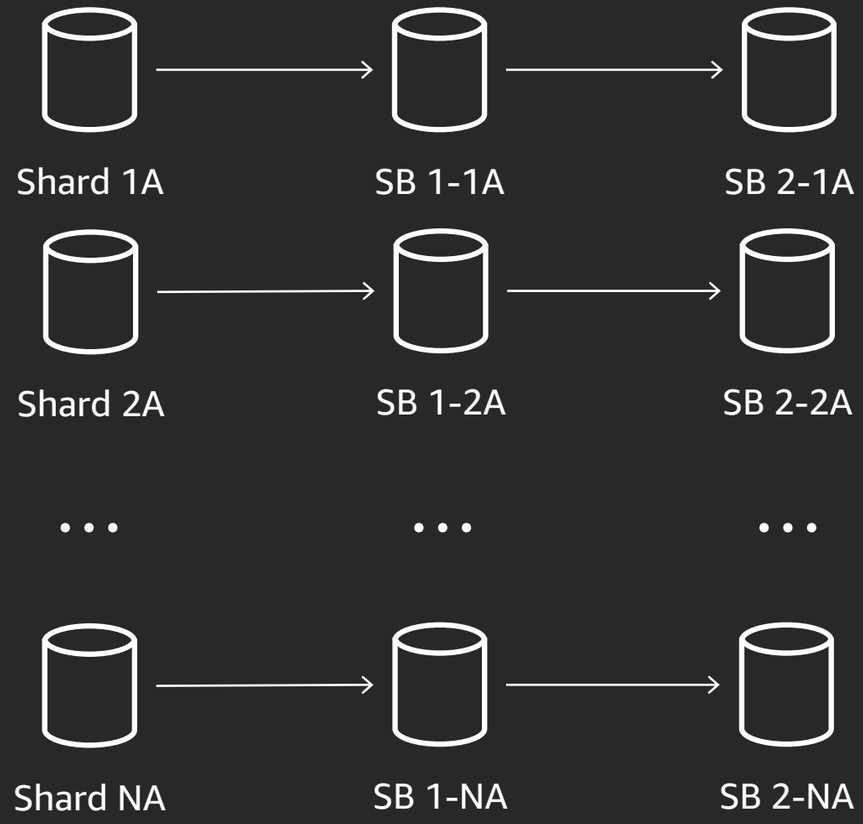

Availability Zone A
Hot
Shard 1A
Hot
!
Shard 2A
Availability Zone B
Warm
Shard 2B
Warm
Shard 1B
Wallet service

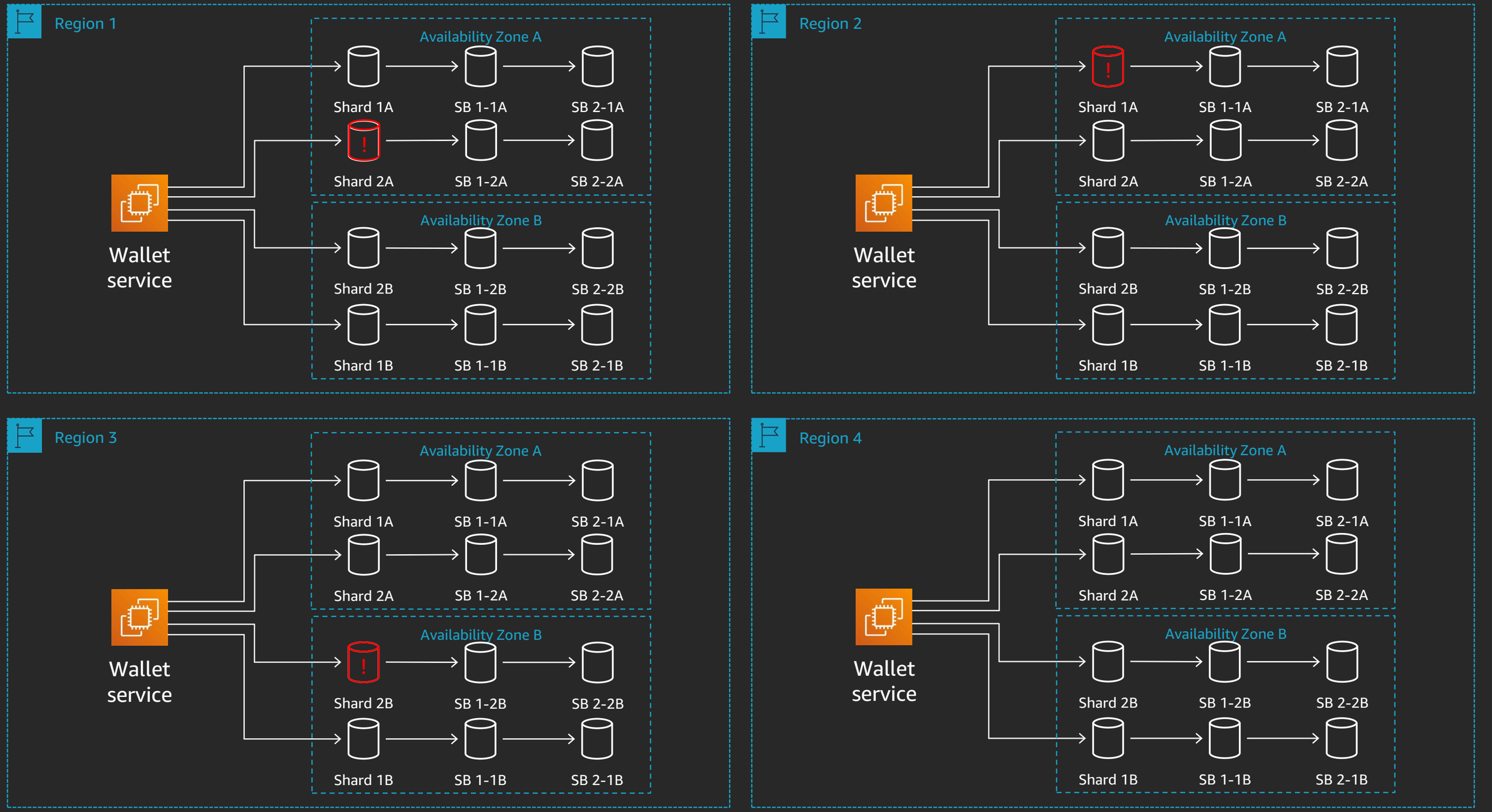

Availability Zone A
Shard 1A
SB 1-1A
SB 2-1A
Shard 2A
SB 1-2A
SB 2-2A
Wallet service
Availability Zone B
Shard 2B
SB 1-2B
SB 2-2B
Shard 1B
SB 1-1B
SB 2-1B

DB scaling

Vertically

Horizontally




Region 1
Availability Zone A
Shard 1A
SB 1-1A
SB 2-1A
Shard 2A
SB 1-2A
SB 2-2A
Availability Zone B
Shard 2B
SB 1-2B
SB 2-2B
Shard 1B
SB 1-1B
SB 2-1B
Wallet service
Region 2
Availability Zone A
Shard 1A
SB 1-1A
SB 2-1A
Shard 2A
SB 1-2A
SB 2-2A
Availability Zone B
Shard 2B
SB 1-2B
SB 2-2B
Shard 1B
SB 1-1B
SB 2-1B
Wallet service
Region 3
Availability Zone A
Shard 1A
SB 1-1A
SB 2-1A
Shard 2A
SB 1-2A
SB 2-2A
Availability Zone B
Shard 2B
SB 1-2B
SB 2-2B
Shard 1B
SB 1-1B
SB 2-1B
Wallet service
Region 4
Availability Zone A
Shard 1A
SB 1-1A
SB 2-1A
Shard 2A
SB 1-2A
SB 2-2A
Availability Zone B
Shard 2B
SB 1-2B
SB 2-2B
Shard 1B
SB 1-1B
SB 2-1B
Wallet service

Moving to Amazon DynamoDB

AWS re:Invent

aws

Goals

- Reduce the cost of operations
- Simplify and reduce the cost of scaling

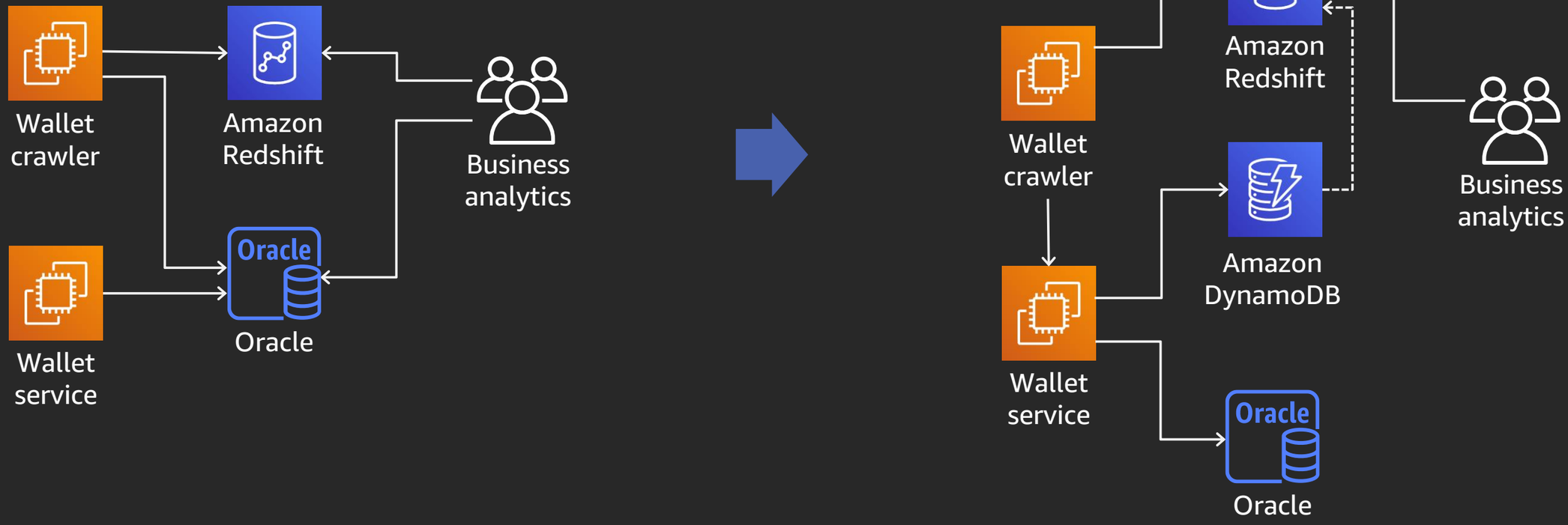
Requirements for migration

- Zero downtime
- Keep data integrity
- Improve performance
- Improve availability
- Allow launch of new features

Database decoupling

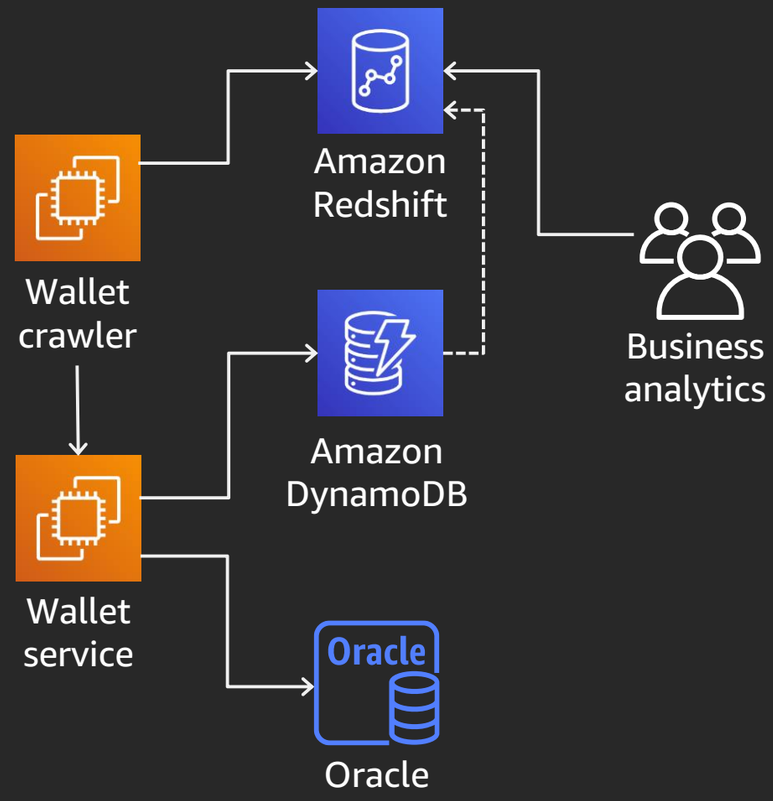
- ❑ Database consumers
- ❑ Join queries
- ❑ Triggers
- ❑ Stored procedures
- ❑ Indexes
- ❑ Unique constraints
- ❑ Transactions
- ❑ Views

Decoupling database consumers

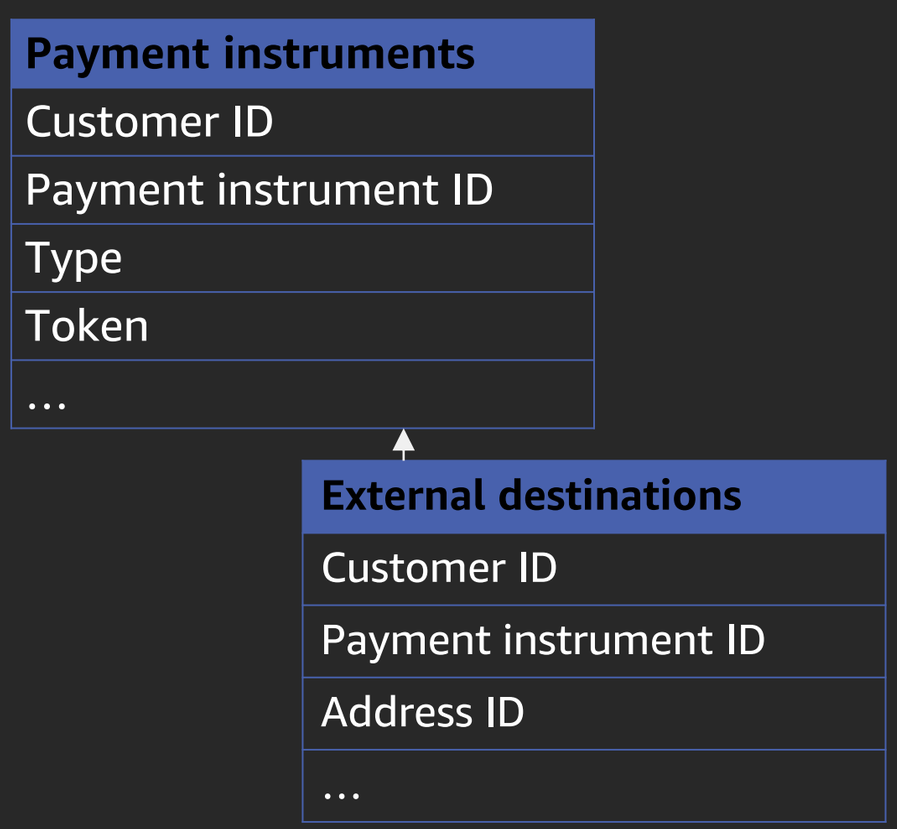


Database decoupling

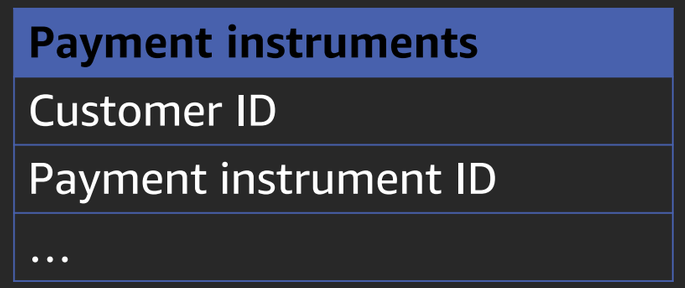
- ✓ Database consumers
- ❑ Join queries
- ❑ Triggers
- ❑ Stored procedures
- ❑ Indexes
- ❑ Unique constraints
- ❑ Transactions
- ❑ Views



Breaking join query



QUERY external_destinations



QUERY payment_instruments

```
SELECT FROM external_destinations
JOIN payment_instruments
```

Database decoupling

- ✓ Database consumers
- ✓ Join queries
- ❑ Triggers
- ❑ Stored procedures
- ❑ Indexes
- ❑ Unique constraints
- ❑ Transactions
- ❑ Views

External destinations
Customer ID
Payment instrument ID
Address ID
…

QUERY external_destinations

Payment instruments
Customer ID
Payment instrument ID
…

QUERY payment_instruments

Database decoupling

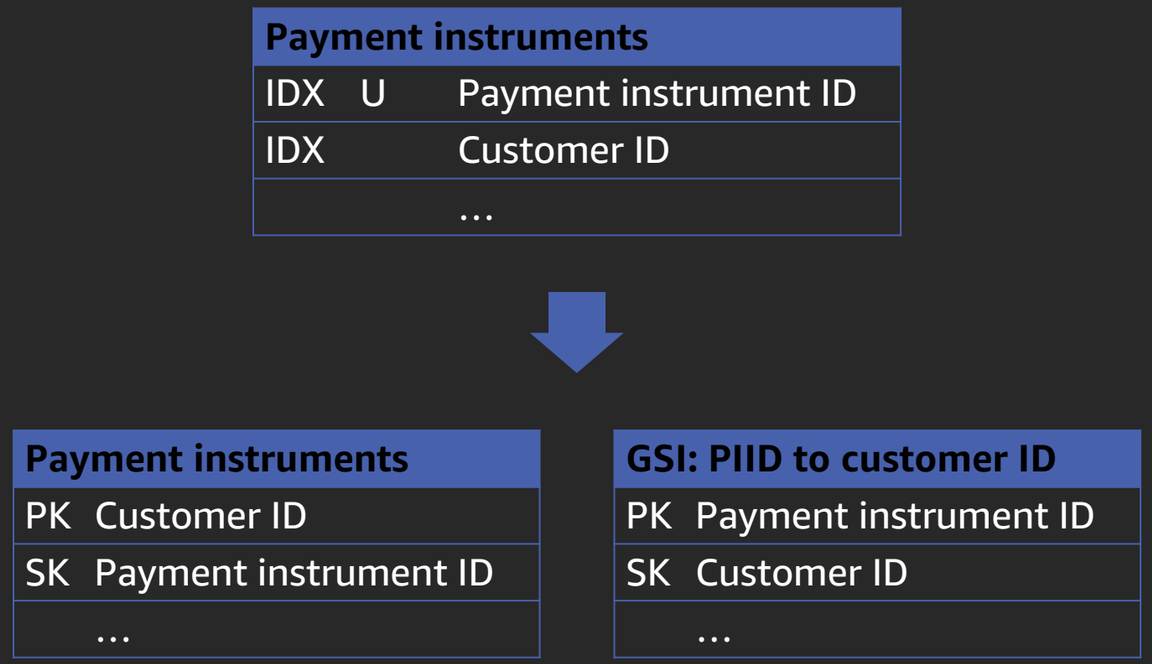
- ✓ Database consumers
- ✓ Join queries
- ✓ Triggers
- ✓ Stored procedures
- ❑ Indexes
- ❑ Unique constraints
- ❑ Transactions
- ❑ Views

Stored procedures

- Moved to application side
- Set before create/update

Database decoupling

- ✓ Database consumers
- ✓ Join queries
- ✓ Triggers
- ✓ Stored procedures
- ✓ Indexes
- ✓ Unique constraints
- ❑ Transactions
- ❑ Views



Database decoupling

- ✓ Database consumers
- ✓ Join queries
- ✓ Triggers
- ✓ Stored procedures
- ✓ Indexes
- ✓ Unique constraints
- ✓ Transactions
- ✓ Views

Best Practices for DynamoDB

PDF | Kindle | RSS

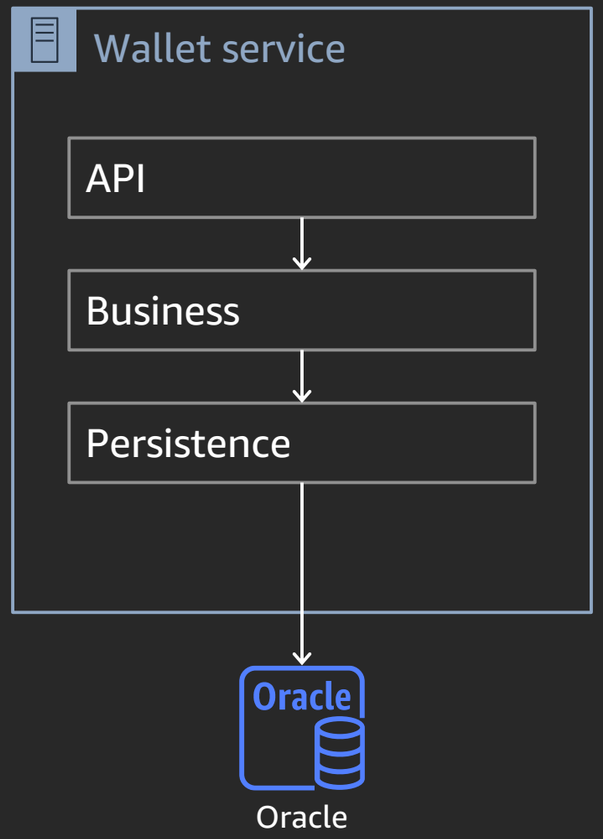
Use this section to quickly find recommendations for maximizing performance and minimizing throughput costs when working with Amazon DynamoDB.

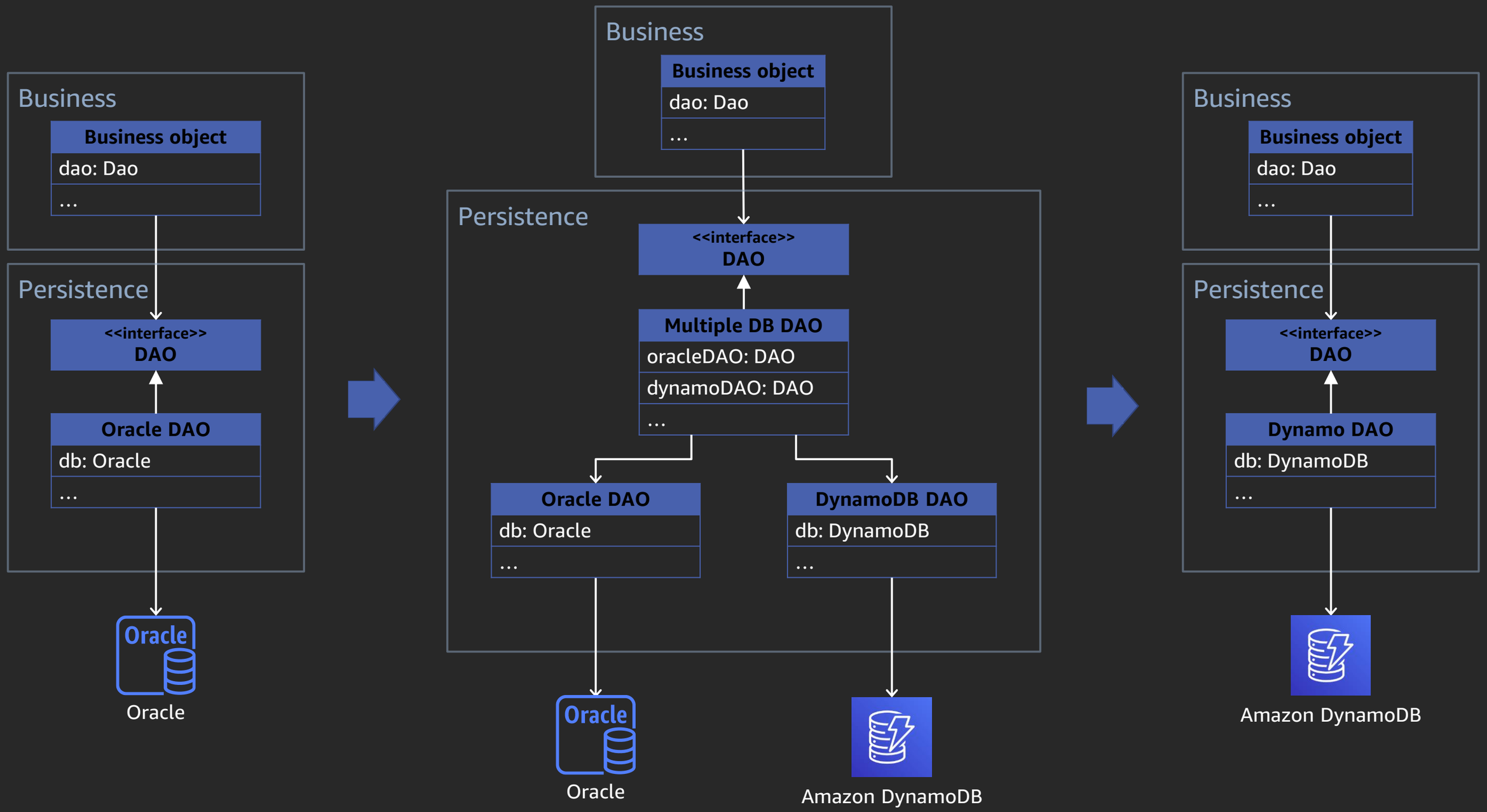
Contents

- NoSQL Design for DynamoDB
  - Differences Between Relational Data Design and NoSQL
  - Two Key Concepts for NoSQL Design
  - Approaching NoSQL Design
- Best Practices for Designing and Using Partition Keys Effectively
  - Using Burst Capacity Effectively
  - Understanding DynamoDB Adaptive Capacity
  - Designing Partition Keys to Distribute Your Workload Evenly
  - Using Write Sharding to Distribute Workloads Evenly
    - Sharding Using Random Suffixes
    - Sharding Using Calculated Suffixes
  - Distributing Write Activity Efficiently During Data Upload
- Best Practices for Using Sort Keys to Organize Data
  - Using Sort Keys for Version Control
- Best Practices for Using Secondary Indexes in DynamoDB
  - General Guidelines for Secondary Indexes in DynamoDB
    - Use Indexes Efficiently
    - Choose Projections Carefully
    - Optimize Frequent Queries to Avoid Fetches
    - Be Aware of Item-Collection Size Limits When Creating Local Secondary Indexes
  - Take Advantage of Sparse Indexes
    - Examples of Sparse Indexes in DynamoDB
  - Using Global Secondary Indexes for Materialized Aggregation Queries
  - Overloading Global Secondary Indexes
  - Using Global Secondary Index Write Sharding for Selective Table Queries
  - Using Global Secondary Indexes to Create an Eventually Consistent Replica

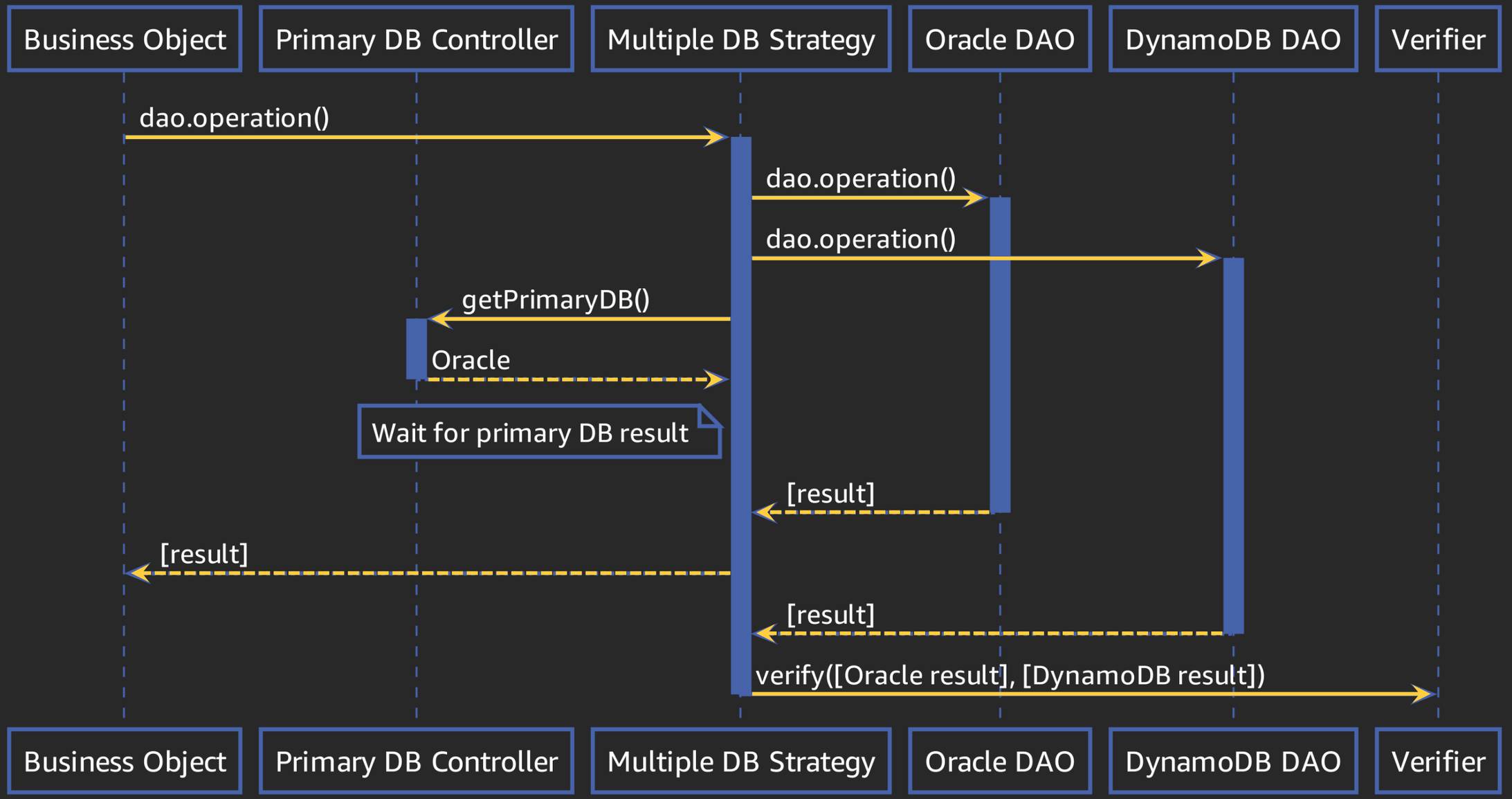
Migration to Amazon DynamoDB

3-step migration


Wallet service
API
Business
Persistence
Oracle
Oracle

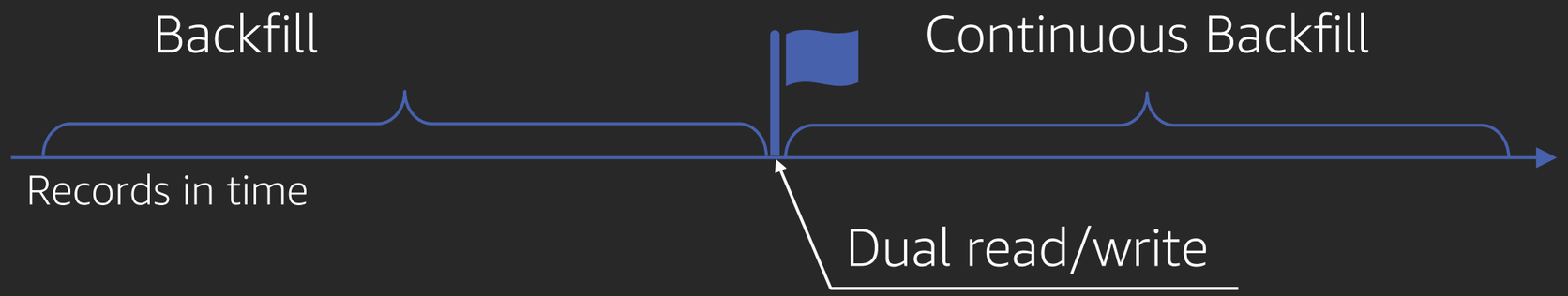

Business
Business object
dao: Dao
…
Persistence
<<interface>>
DAO
Oracle DAO
db: Oracle
…
Oracle
Oracle
Business
Business object
dao: Dao
…
Persistence
<<interface>>
DAO
Multiple DB DAO
oracleDAO: DAO
dynamoDAO: DAO
…
Oracle DAO
db: Oracle
…
DynamoDB DAO
db: DynamoDB
…
Oracle
Amazon DynamoDB
Business
Business object
dao: Dao
…
Persistence
<<interface>>
DAO
Dynamo DAO
db: DynamoDB
…
Amazon DynamoDB

Dual read/write

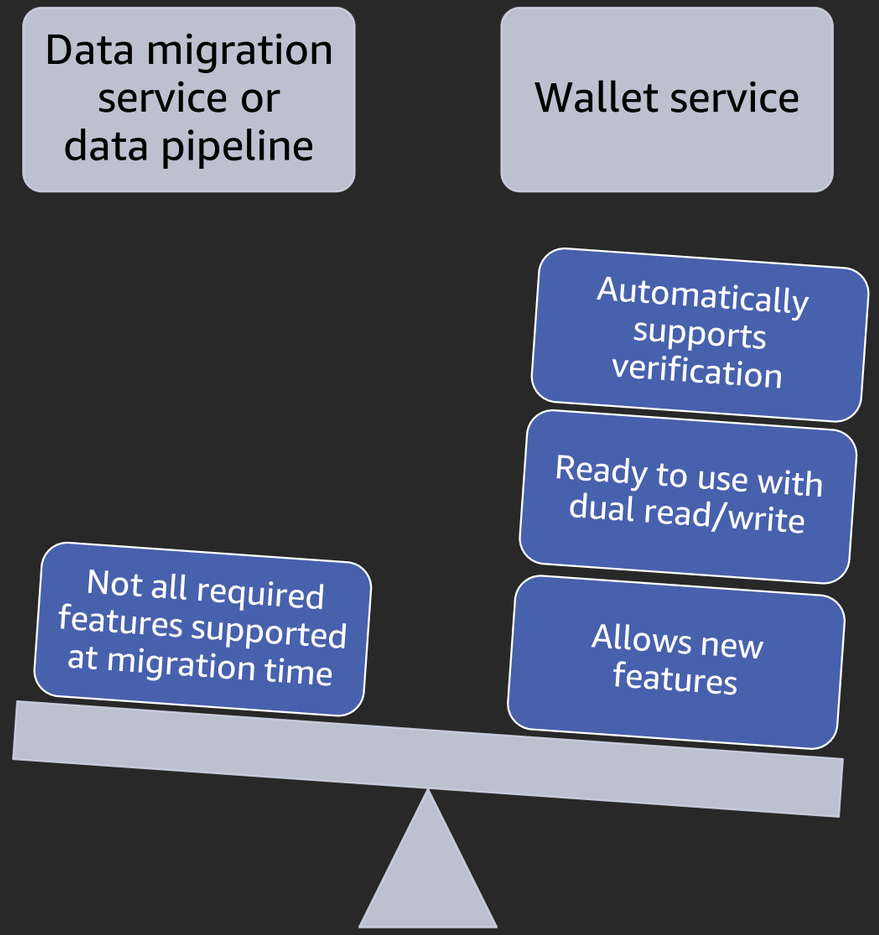


Verification

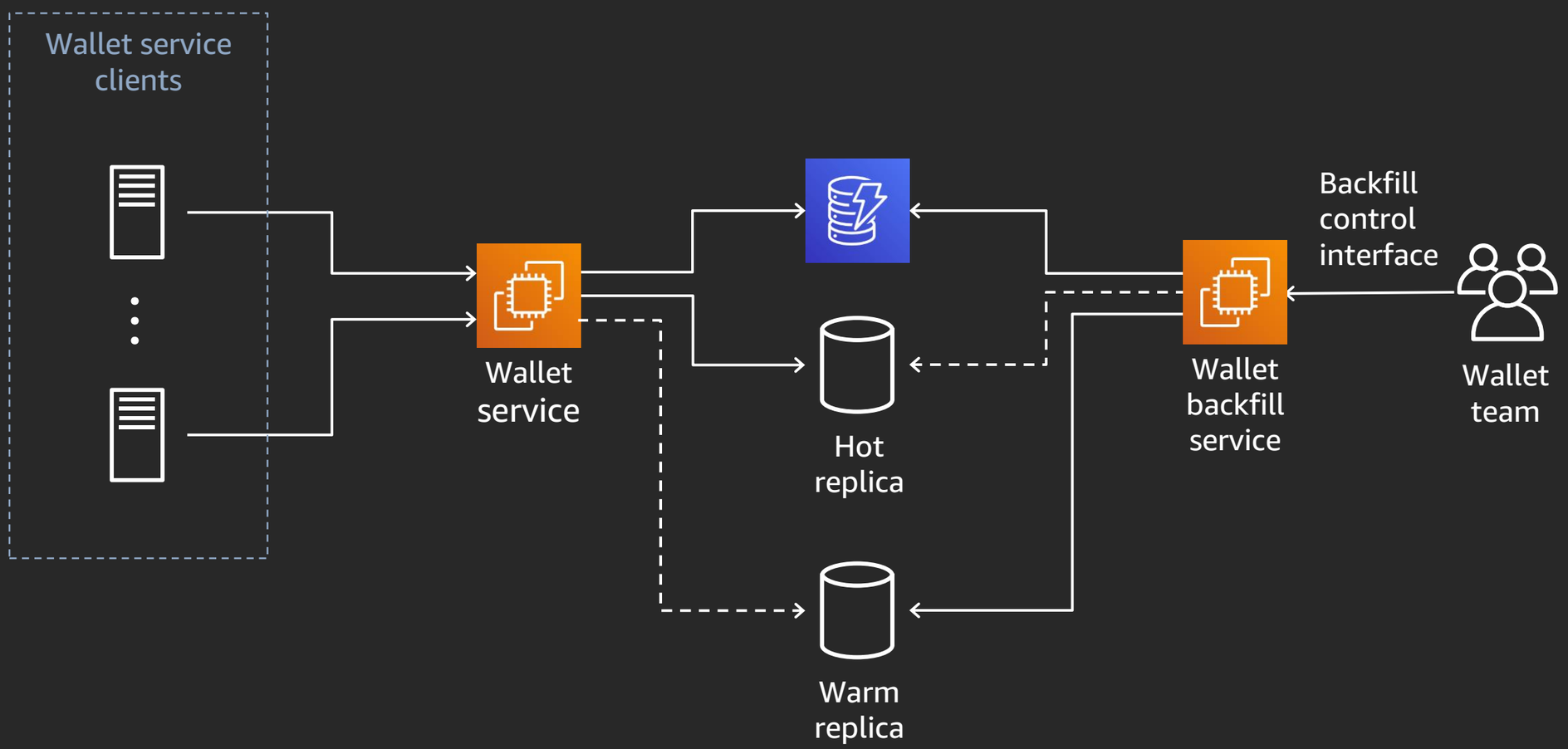
Backfill


Backfill
Continuous Backfill
Records in time
Dual read/write

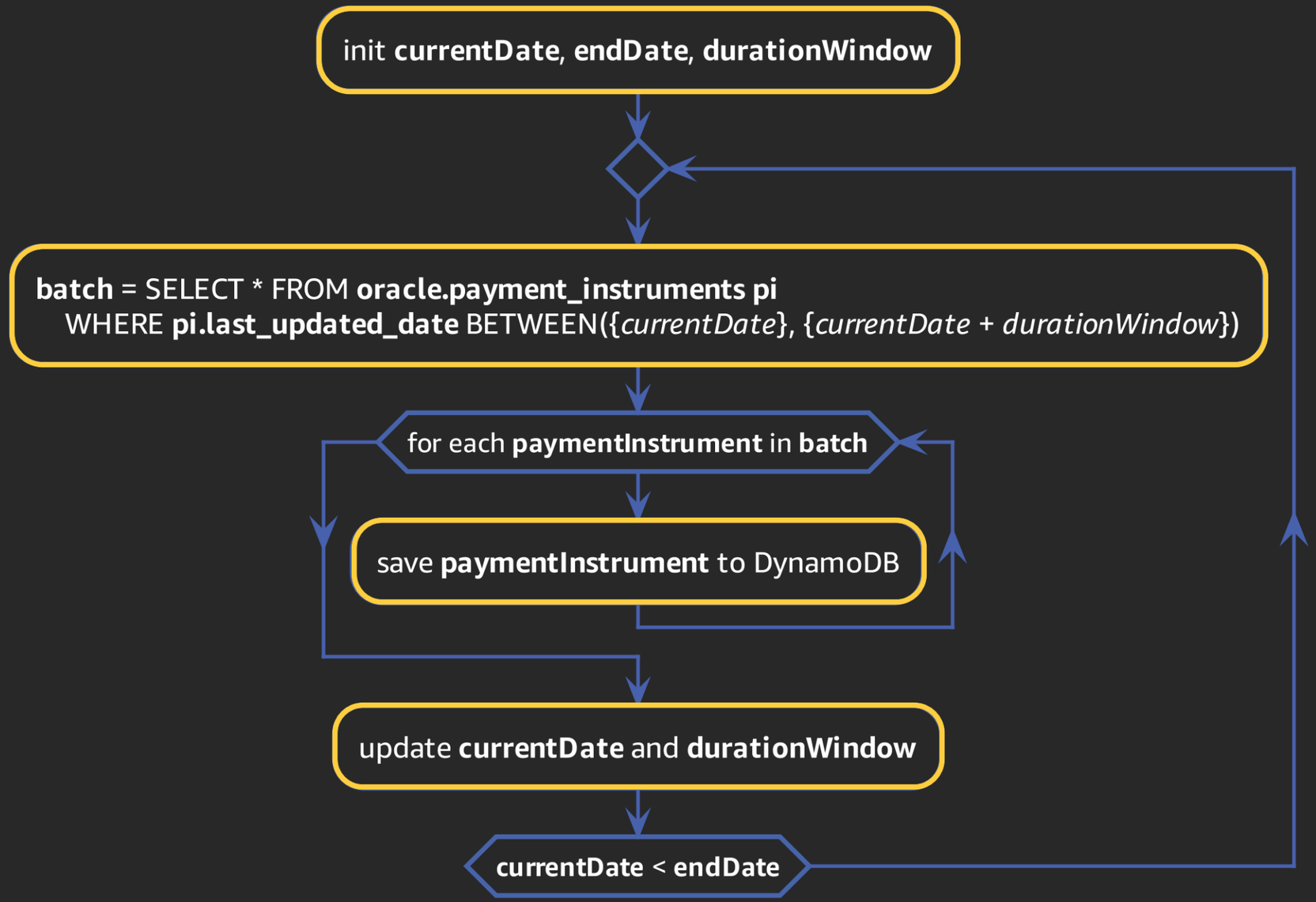
Using Wallet service to migrate itself



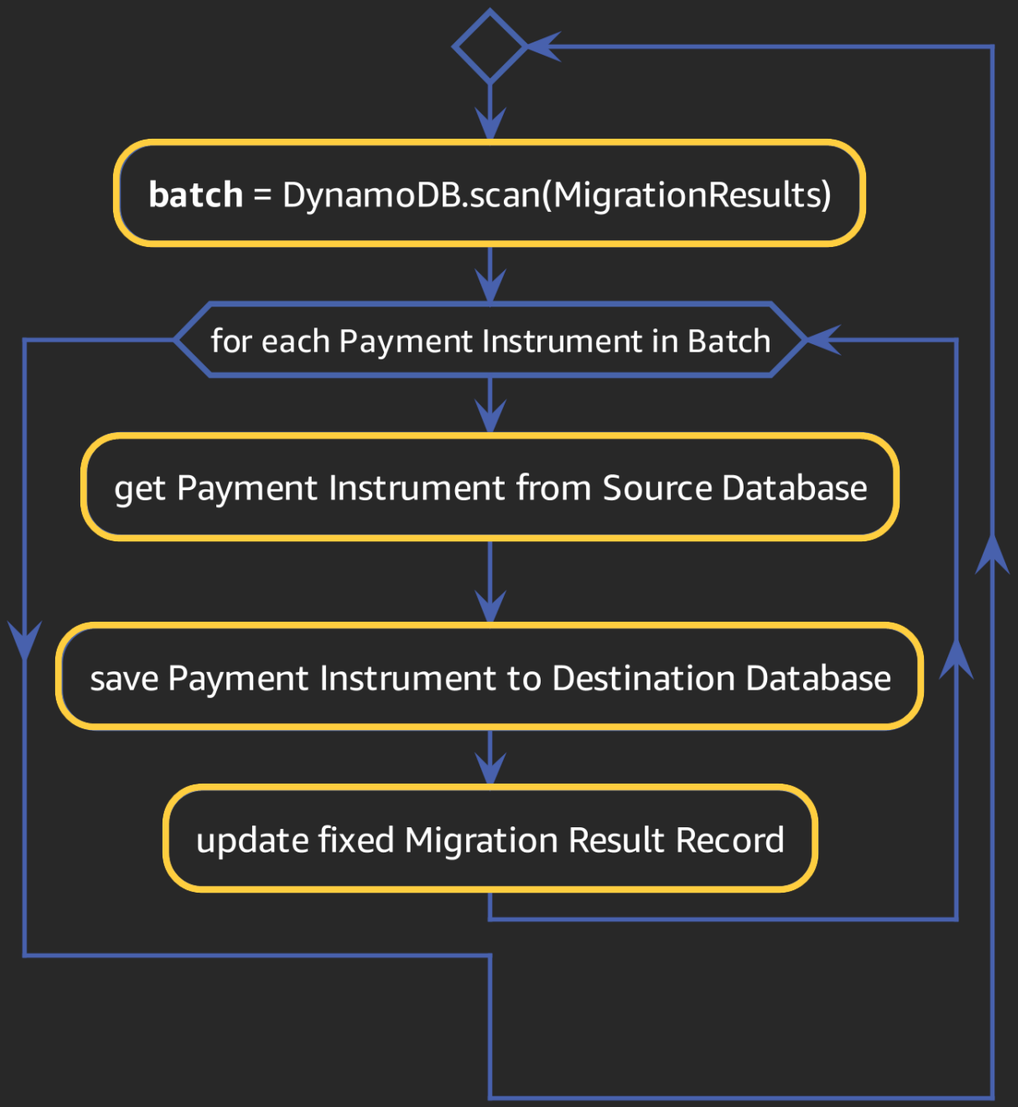
Backfill setup



Full Backfill



Continuous Backfill

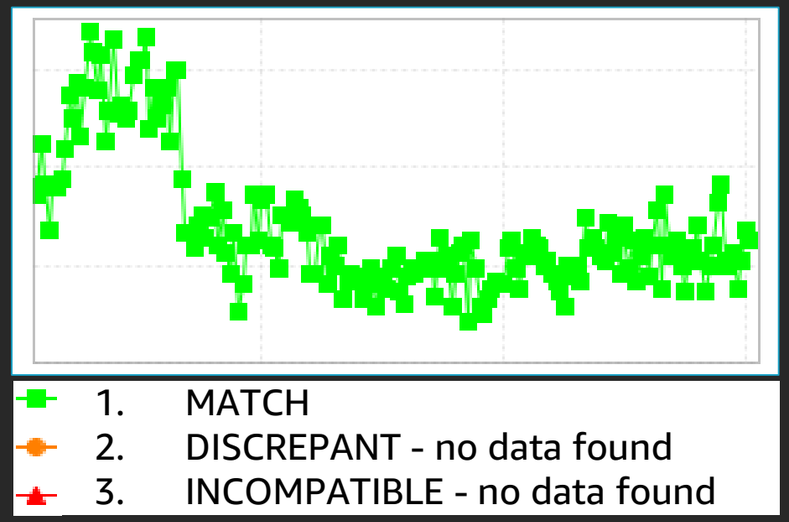


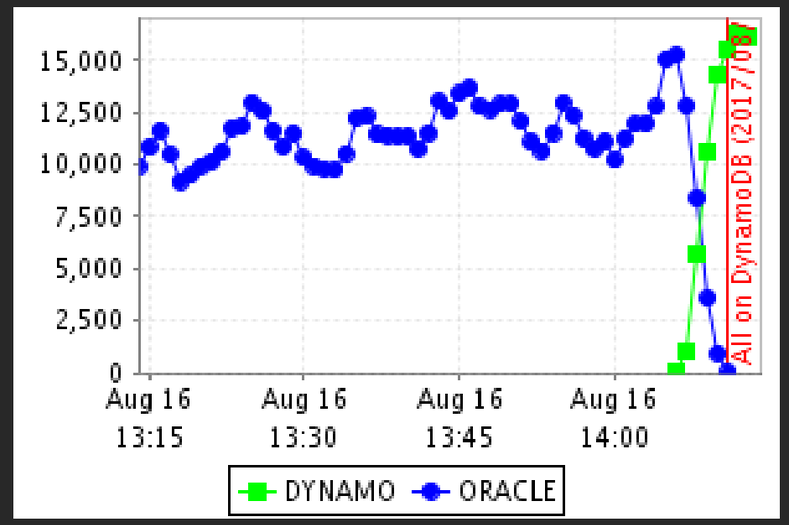
Migration takes time

- Expected: 1 billion records / 1 day requires ~ 11,600 TPS
- Expected: 5,000 TPS → ~2.3 days?
- Actual: TPS ~600 = ~19 days

Getting ready to switch

- ✓ Dual read/write
- ✓ Backfill
- ✓ Switch





Migration results

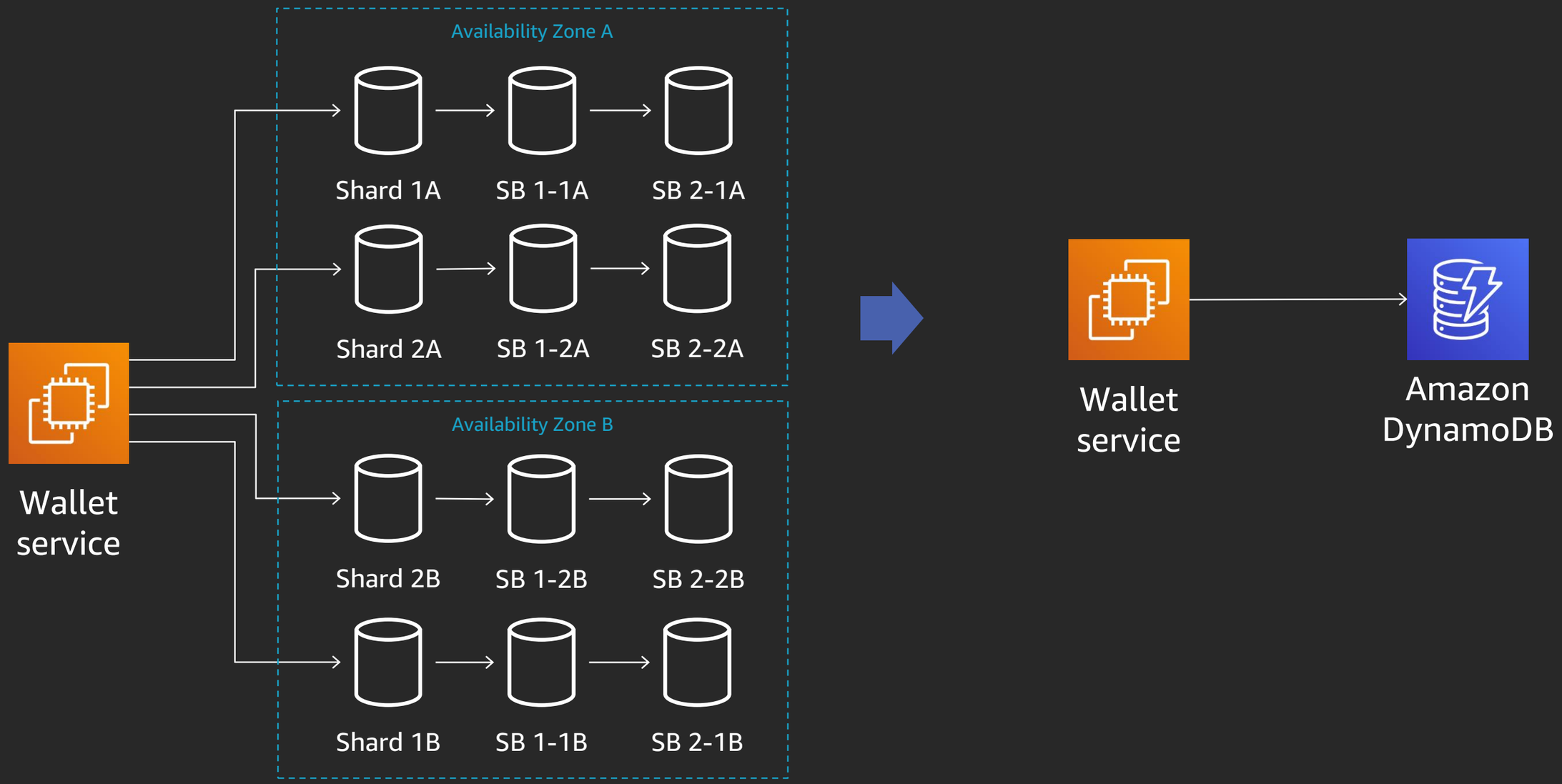
Benefits

- Improved latency by 50%
- Increased request per second per host by 40%
- Reduced operations costs by 90%

Benefits

- Autoscaling (including on-demand)
- Time to live (TTL)
- Point-in-time recovery
- On-demand backup and restore
- Global tables
- Increased number of supported global secondary indexes

From Managing DB to Managed DB



Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.