In-Memory Computing SUMMIT | NORTH AMERICA 2018

Improve Application Performance with SQL Auto-Caching, No Code changes

Roland Lee

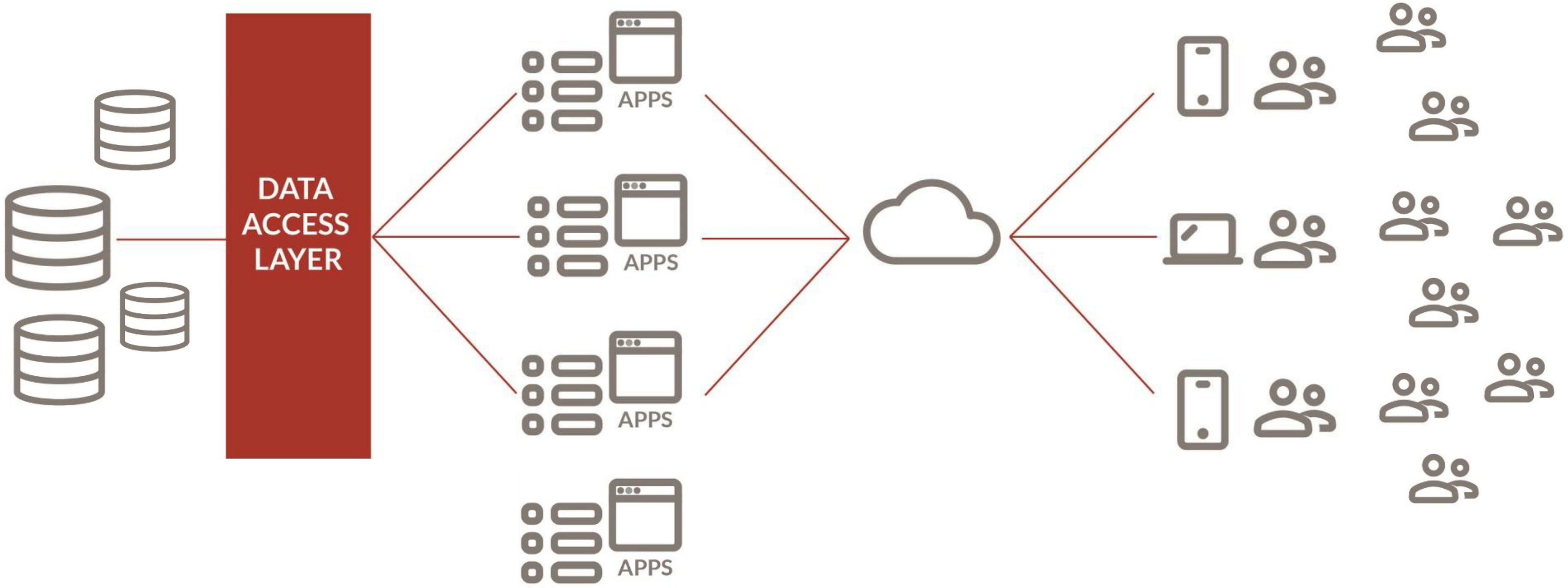
Heimdall Data

Agenda

1) Database Proxy Introduction

2) Demo

Web scale Challenges



More users and applications implies

- Higher data volumes
- More tables, changes to schema
- **Higher latency responses**

Database Proxy Vendors

Feature	HEIMDALL DATA	ScaleArc	MariaDB	ProxySQL
Automated Failover	✓	✓	✓	✓
SQL Read/Write Splitting	✓	✓	✓	✓
Automated Cache invalidation	✓			
Reduces network latency	✓			

IMDG or Database Proxy

Amazon ElastiCache

- Best scale & performance
- Greenfield applications
- Requires code changes

- Good scalability & performance
- Existing applications, small dev
- **No code changes**

Heimdall Data Software Options

Application Server

Application

Vendor Database Driver

Heimdall Data Proxy

Runs as an agent

PostgreSQL

MySQL

Microsoft SQL Server

Application Server

Application

Heimdall Data JDBC

Vendor JDBC Driver

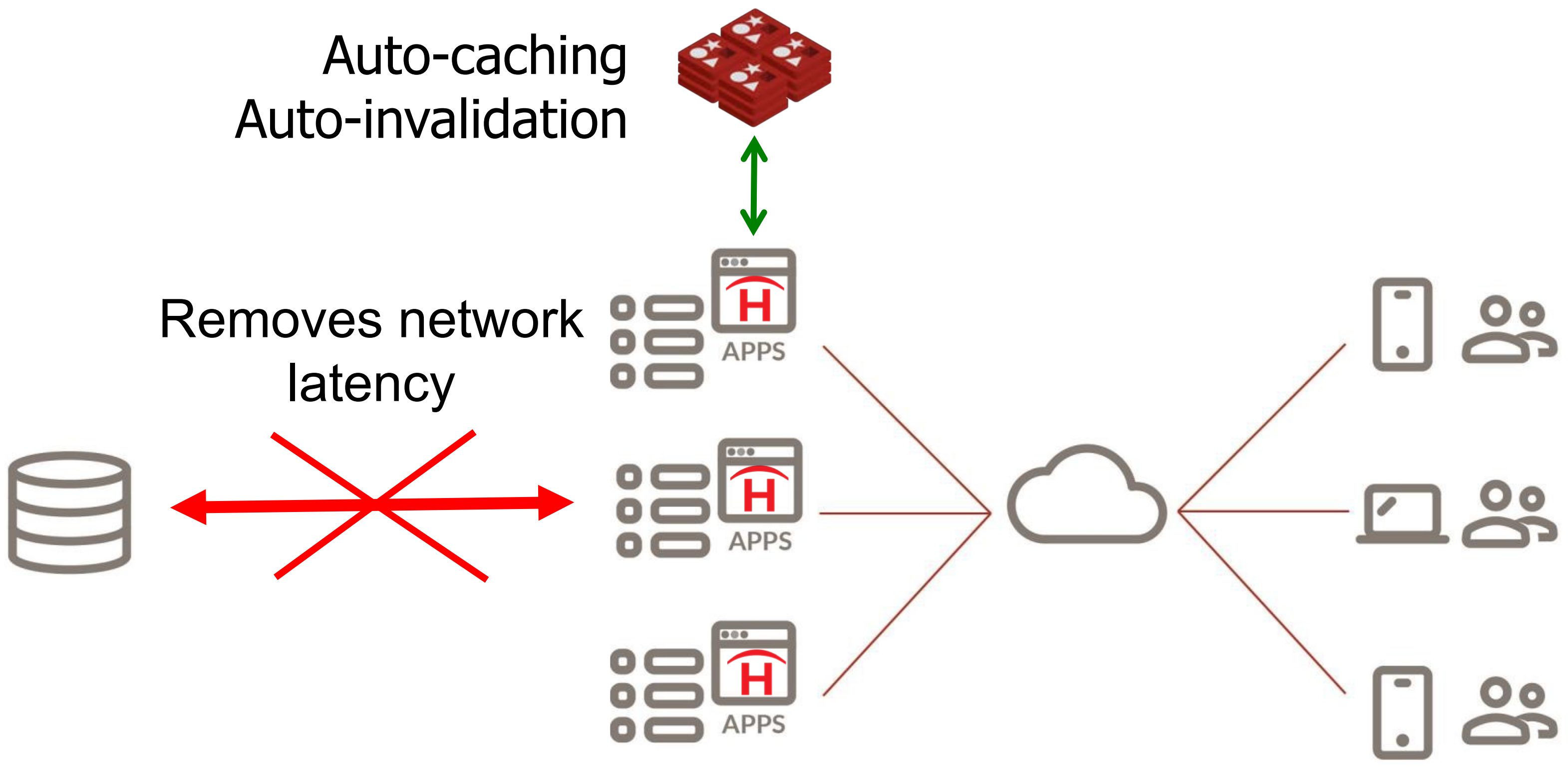
JDBC driver, .jar file

PostgreSQL

ORACLE

Any JDBC data source

Heimdall Data Distributed Proxy

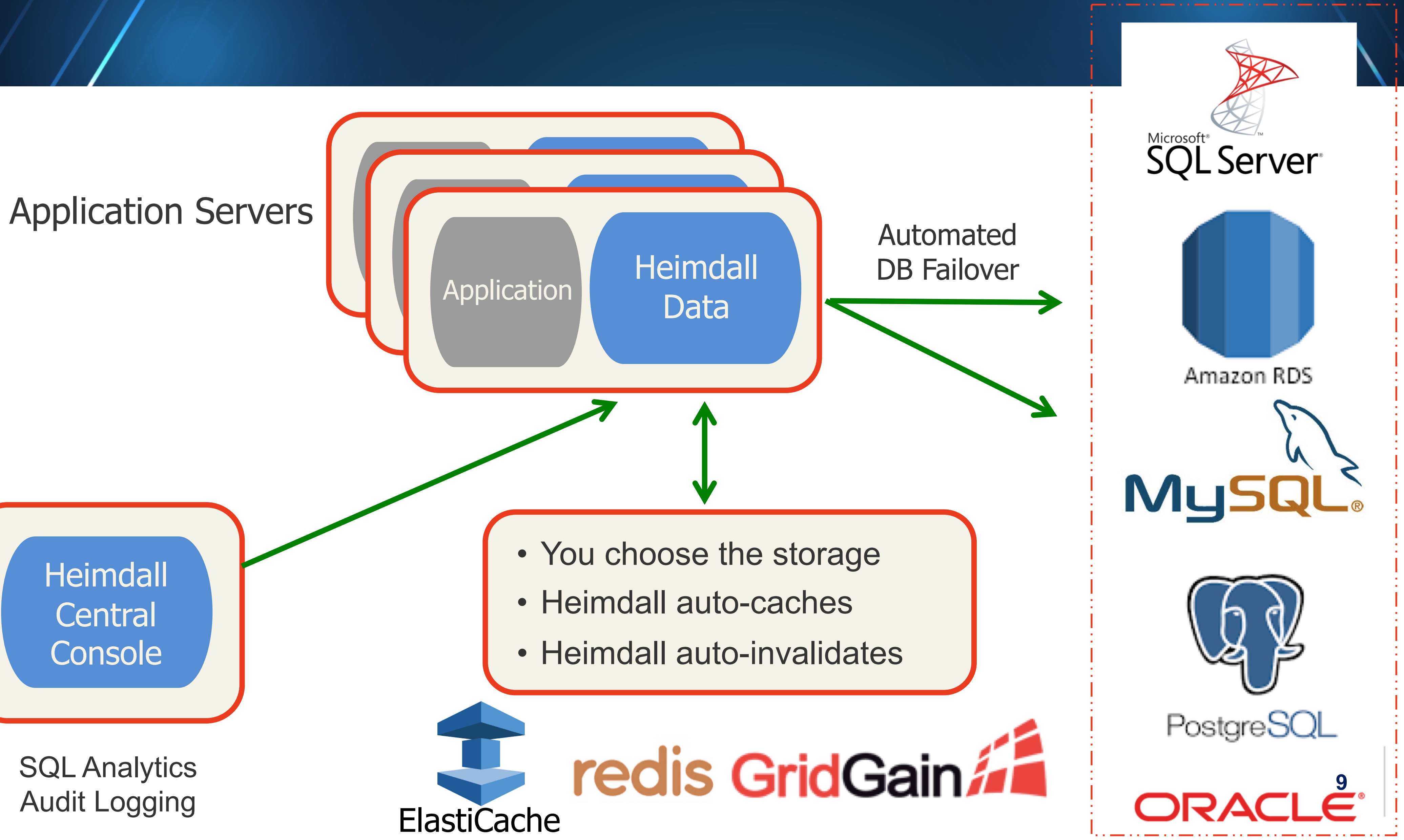


1. Heimdall is installed on each application instance
2. Direct Heimdall to the grid-cache of your choice (e.g. Redis, Hazelcast)
3. Heimdall will auto-cache and auto-invalidate SQL results to the look-aside cache

Heimdall Data Uses Cases

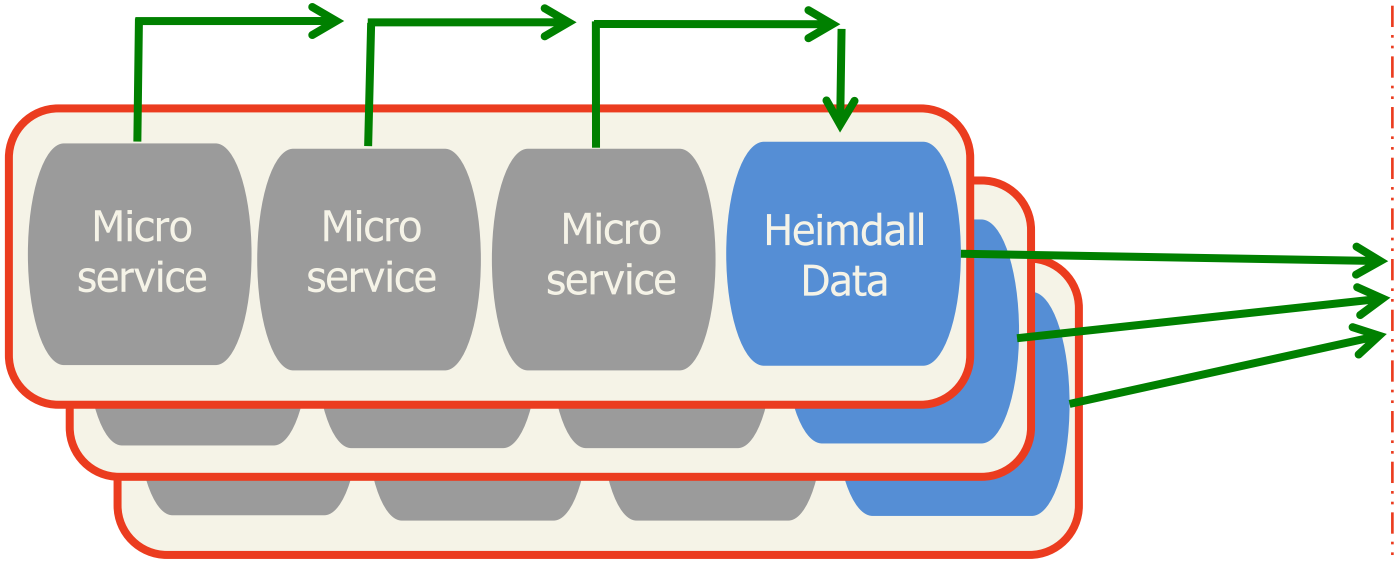
Use case	Customer Benefit
SQL Results Caching	• Auto-caching / Auto-invalidate • NO code changes
Automated failover	• Faster failover for MySQL & SQL Server AlwaysOn • PGPool-II Replacement
Batch Processing	• Improves write performance • Batches singleton DML operations • Removes Commit overhead
Auditing for Privacy Compliance	• Logs data access: Who, what, when • GDPR, SOX, PCI, HIPPA

Heimdall Data Architecture



Heimdall Data for Microservices

Microservice Application Hosts



Microsoft SQL Server

Amazon RDS

MySQL

PostgreSQL

ORACLE

Heimdall Data Solution

DB Failover

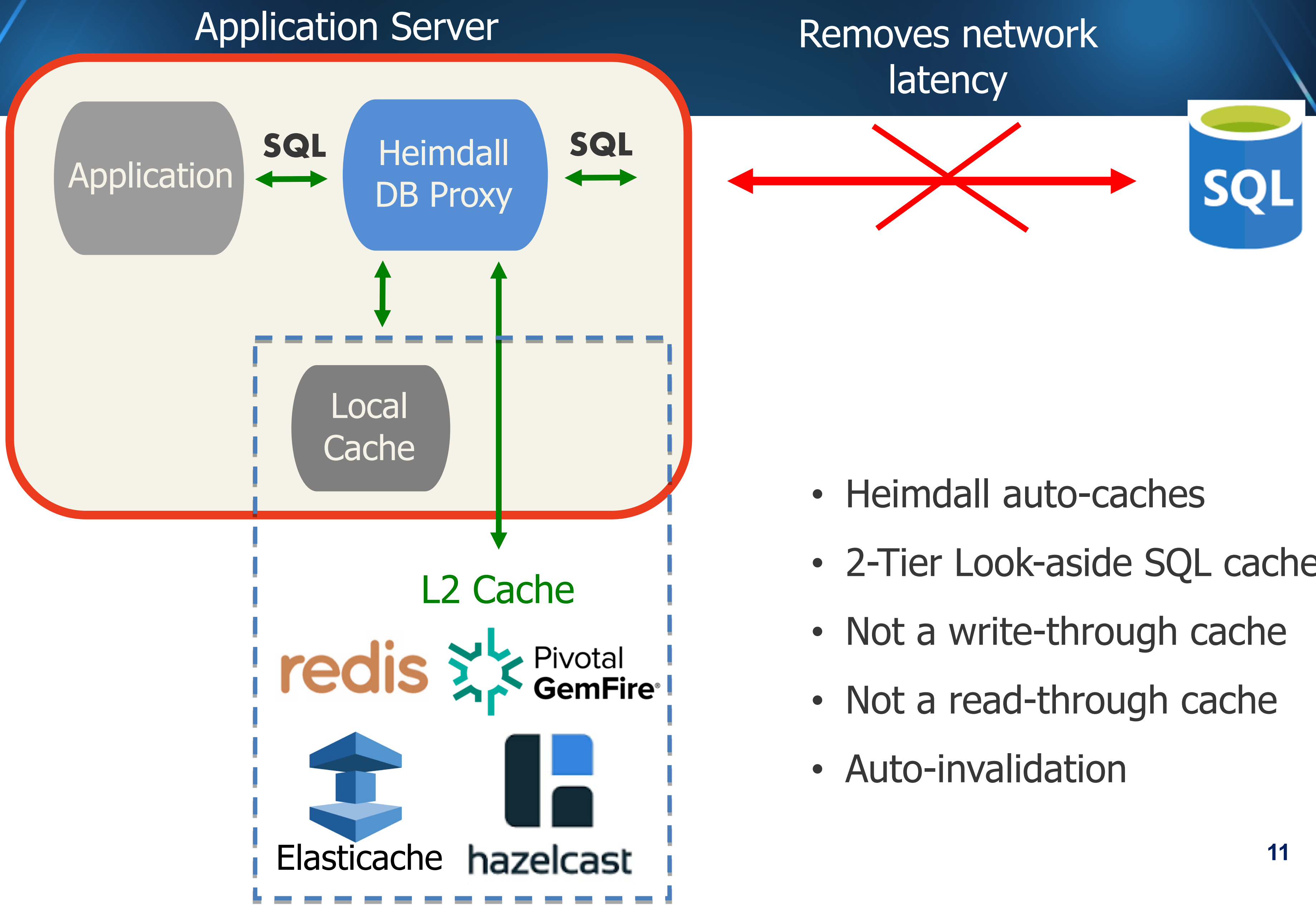
Connection Pooling

Caching

SQL Analytics

Audit logging

Application Side – SQL Caching



- Heimdall auto-caches
- 2-Tier Look-aside SQL cache
- Not a write-through cache
- Not a read-through cache
- Auto-invalidation

What to Cache?

Uses real-time analysis and statistics on:

- Query frequency and variability
- Relative performance of Cache vs. Database

Provides:

- Auto-cache only if there is a performance benefit
- Cache recommendations and benefits

Heimdall Query Analytics

Heimdall Data Console

demoa.heimdalldata.com:8087/#/analytics

Very cacheable. 700 µs per query

Rank	One-Click Optimize	Query	Server Time (%)	Duplicate Query & Response (%)	Cache Index	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Average Result Size
1	Cache	SELECT user_id, meta_key, meta_value FROM wp_usermeta WHERE user_id IN (?) ORDER BY umeta_id ASC	13	100	100	86400	0.0	2.1k	697.2	730.7	1k
2	Cache	SELECT * FROM wp_users WHERE ID = ?	13	100	100	86400	0.0	2.1k	693.7	733.2	310.0
3	Cache	SELECT * FROM wp_posts WHERE ID = ? LIMIT ?	8.9	100	100	86400	0.0	1.4k	756.1	836.8	2.9k
4	Cache	SELECT option_name, option_value FROM wp_options WHERE autoload = ?	7.1	100	100	86400	0.0	805.0	1026.5	1123.9	21.9k
5	Cache	SELECT option_value FROM wp_options WHERE option_name = ? LIMIT ?	6.8	100	100	86400	0.0	1.1k	691.7	707.6	6.9
6	Cache	SELECT wp_posts.* FROM wp_posts WHERE ?=? AND ((YEAR(wp_posts.post_date) = ? AND MONTH(wp_posts.post_date) = ? AND DAYOFMONTH(wp_posts.post_date) = ?)) AND wp_posts.post_name = ? AND wp_posts.post_type = ? ORDER BY wp_posts.post_date DESC	5.4	100	100	86400	0.0	773.0	811.6	886.1	3k
7	Cache	SELECT t.*, tt.*, tr.object_id FROM wp_terms AS t INNER JOIN wp_term_taxonomy AS tt ON t.term_id = tt.term_id INNER JOIN wp_term_relationships AS tr ON tr.term_taxonomy_id = tt.term_taxonomy_id WHERE tt.taxonomy IN (?, ?, ?) AND tr.object_id IN (?) ORDER BY t.name ASC	5.1	100	100	86400	0.0	804.0	733.1	784.2	909.4
8	Cache	SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN (?) ORDER BY meta_id ASC	4.7	100	100	86400	0.0	804.0	680.7	700.2	1.0
9	Cache	SELECT wp_posts.* FROM wp_posts WHERE ID IN (?, ?, ?, ?, ?)	2.4	100	100	86400	0.0	340.0	813.1	915.2	13k
10	Cache	SELECT wp_comments.* FROM wp_comments WHERE comment_ID IN (?, ?, ?, ?, ?)	2.3	100	100	86400	0.0	351.0	771.2	840.0	5.9k
11	Cache	SELECT wp_posts.ID FROM wp_posts WHERE ?=? AND wp_posts.post_type = ? AND ((wp_posts.post_status = ?)) ORDER BY wp_posts.post_date DESC LIMIT ?, ?	2.3	100	100	86400	0.0	339.0	791.5	811.0	120.0
		SELECT SQL_CALC_FOUND_ROWS wp_comments.comment_ID FROM wp_comments WHERE (									

Heimdall Cache Rules Analytics

Before: Query response time 700 μs per query. Significant repetitive database load.

Rank	One-Click Optimize	Query	Server Time (%)	Duplicate Query & Response (%)	Cache Index	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (μs)	Result Retrieval Time (μs)	Average Result Size
1	Cache	SELECT user_id, meta_key, meta_value FROM wp_usermeta WHERE user_id IN (?) ORDER BY umeta_id ASC	13	100	100	86400	0.0	2.1k	697.2	730.7	1k
2	Cache	SELECT * FROM wp_users WHERE ID = ?	13	100	100	86400	0.0	2.1k	693.7	733.2	310.0
3	Cache	SELECT * FROM wp_posts WHERE ID = ? LIMIT ?	8.9	100	100	86400	0.0	1.4k	756.1	836.8	2.9k

After: Cached response in 70 μs per query. Reduced database load.

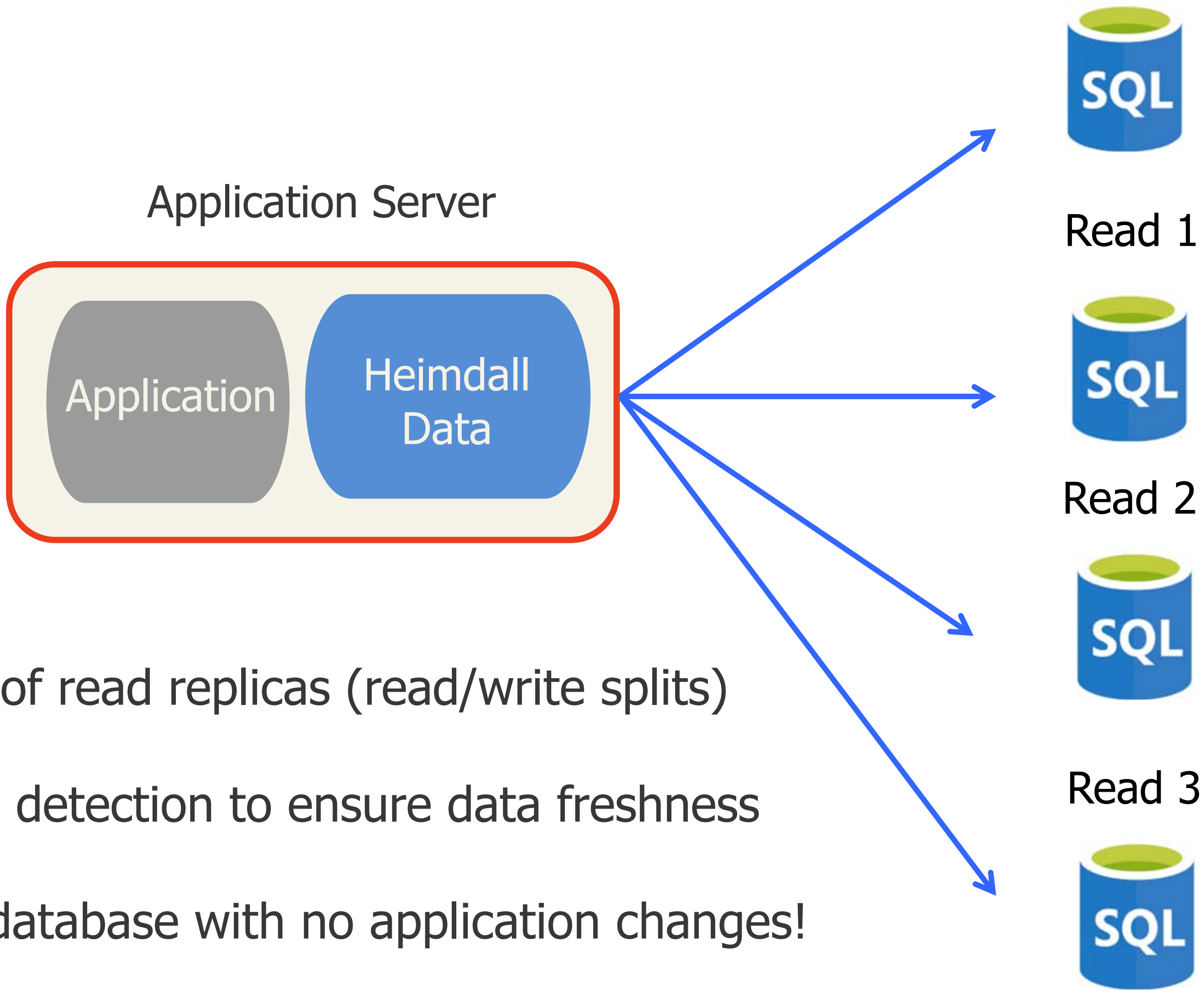
20	Caching	SELECT user_id, meta_key, meta_value FROM wp_usermeta WHERE user_id IN (?) ORDER BY umeta_id ASC	1.8	100	100	86400	99.9	55k	68.4	68.4	1.9
21	Caching	SELECT * FROM wp_users WHERE ID = ?	1.6	100	100	86400	99.9	54.9k	59.5	59.5	1.3
22	Caching	SELECT * FROM wp_posts WHERE ID = ? LIMIT ?	1.4	100	100	86400	97.8	35.4k	77.8	79.3	63.5

In-Memory Computing SUMMIT | NORTH AMERICA 2018

Other use cases

Other use cases

Use Case #1: Read / Write Splitting

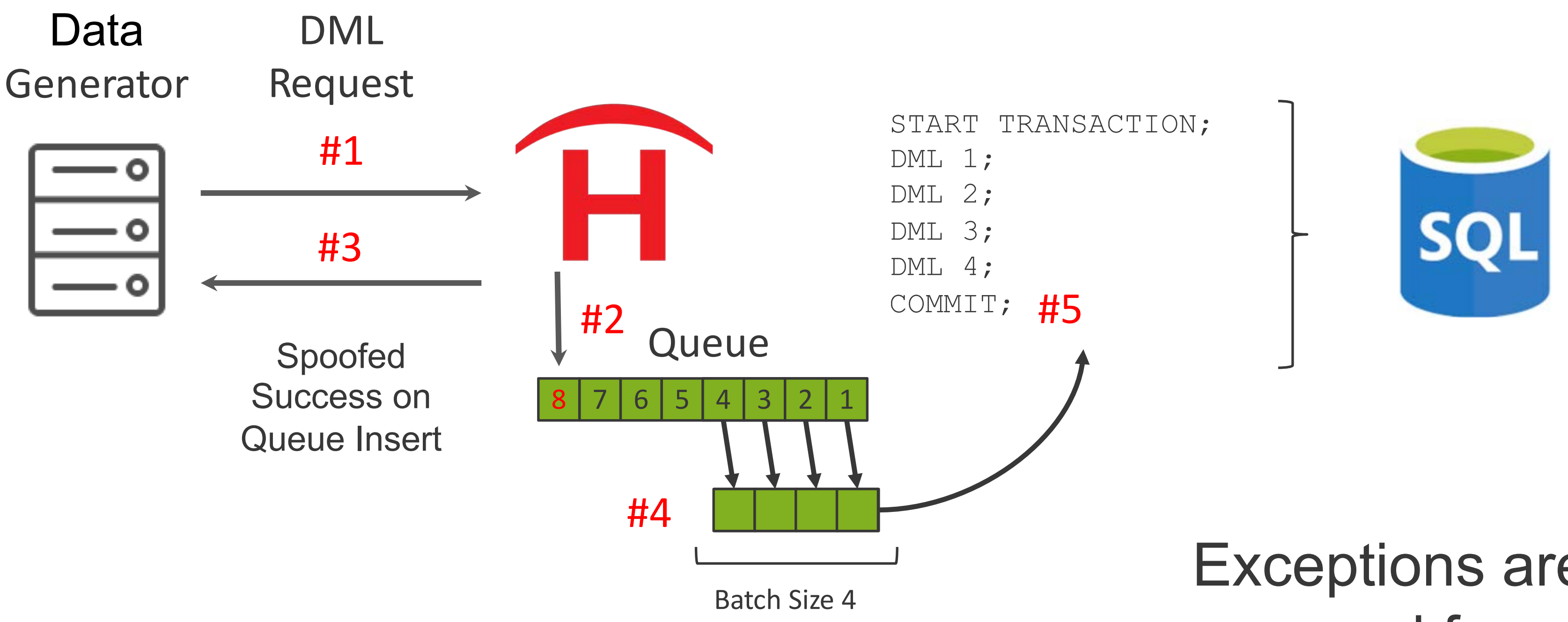


1. Automate use of read replicas (read/write splits)

1. Replication lag detection to ensure data freshness

2. Scale out the database with no application changes!

Use Case #2: Async DML Batching

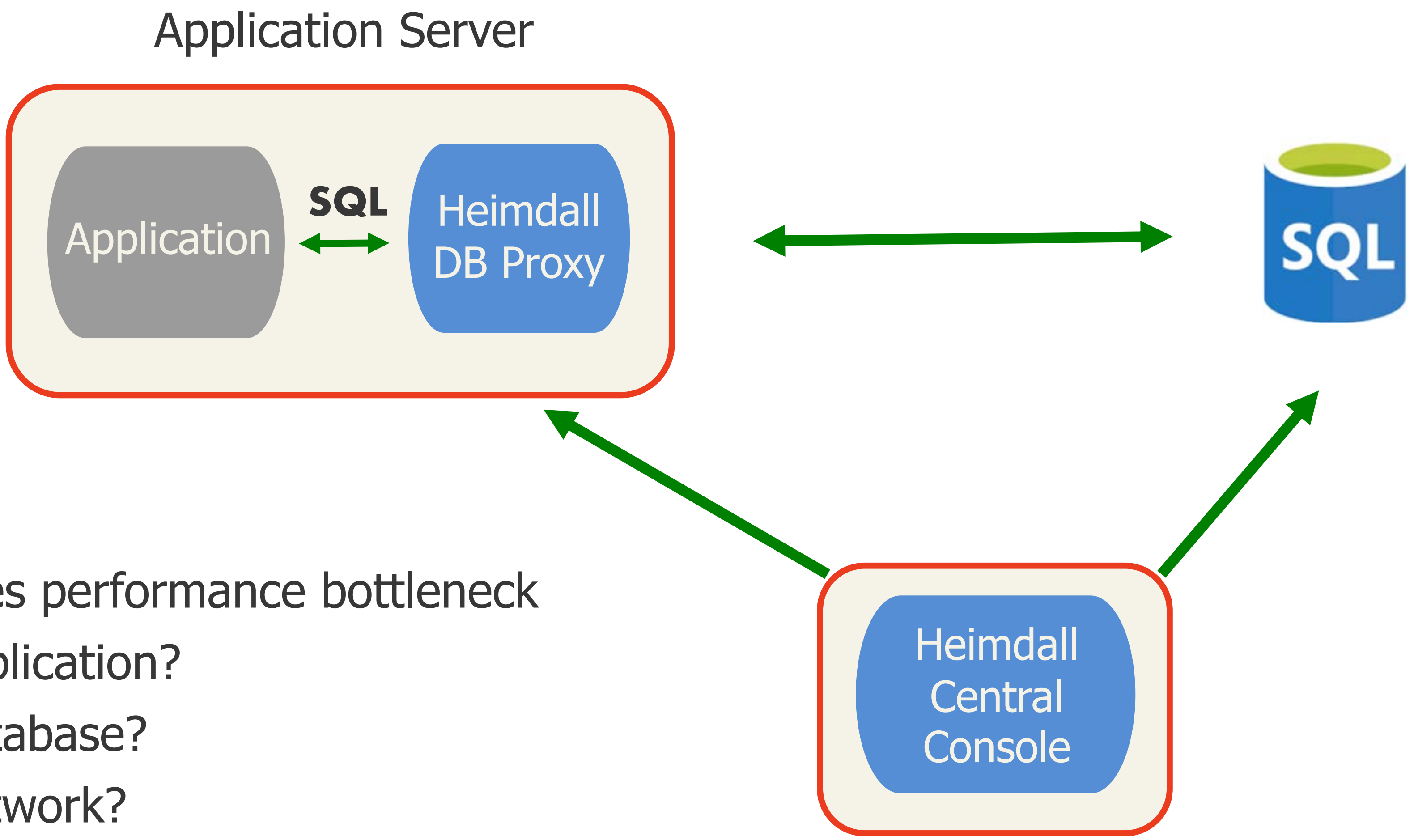


Exceptions are logged, removed from batch, and transaction restarted

Benefits:

- Lower CPU overhead due to fewer commits
- Improved application response time
- Improved DML scale

Use Case #3: SQL Analytics



Identifies performance bottleneck

- Application?
- Database?
- Network?

Demo

In-Memory Computing SUMMIT NORTH AMERICA 2018

Thank you

Roland Lee

Heimdall Data