



Turbocharge Amazon RDS with Amazon ElastiCache for Redis

Damon LaCaille
Sr. Solutions Architect
AWS

Table of contents

Need for Speed

Caching Fundamentals

Lazy Loading Pattern

Caching Demo

Amazon ElastiCache

Q&A

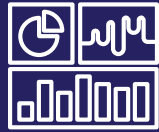
Need for Speed



Common Use Cases



Caching



Real-time
analytics



Leaderboard
apps



Geospatial



Media
streaming



Session
store



Chat
apps



Message
queues



Machine
learning

Common Use Cases



Caching



Real-time
analytics



Leaderboard
apps



Geospatial



Media
streaming



Session
store



Chat
apps



Message
queues



Machine
learning

Why Performance Matters

"A 100-millisecond delay in website load time can hurt conversion rates by 7 percent."

"A two-second delay in web page load time increases bounce rate by 103 percent."

– 2017 Akamai Study

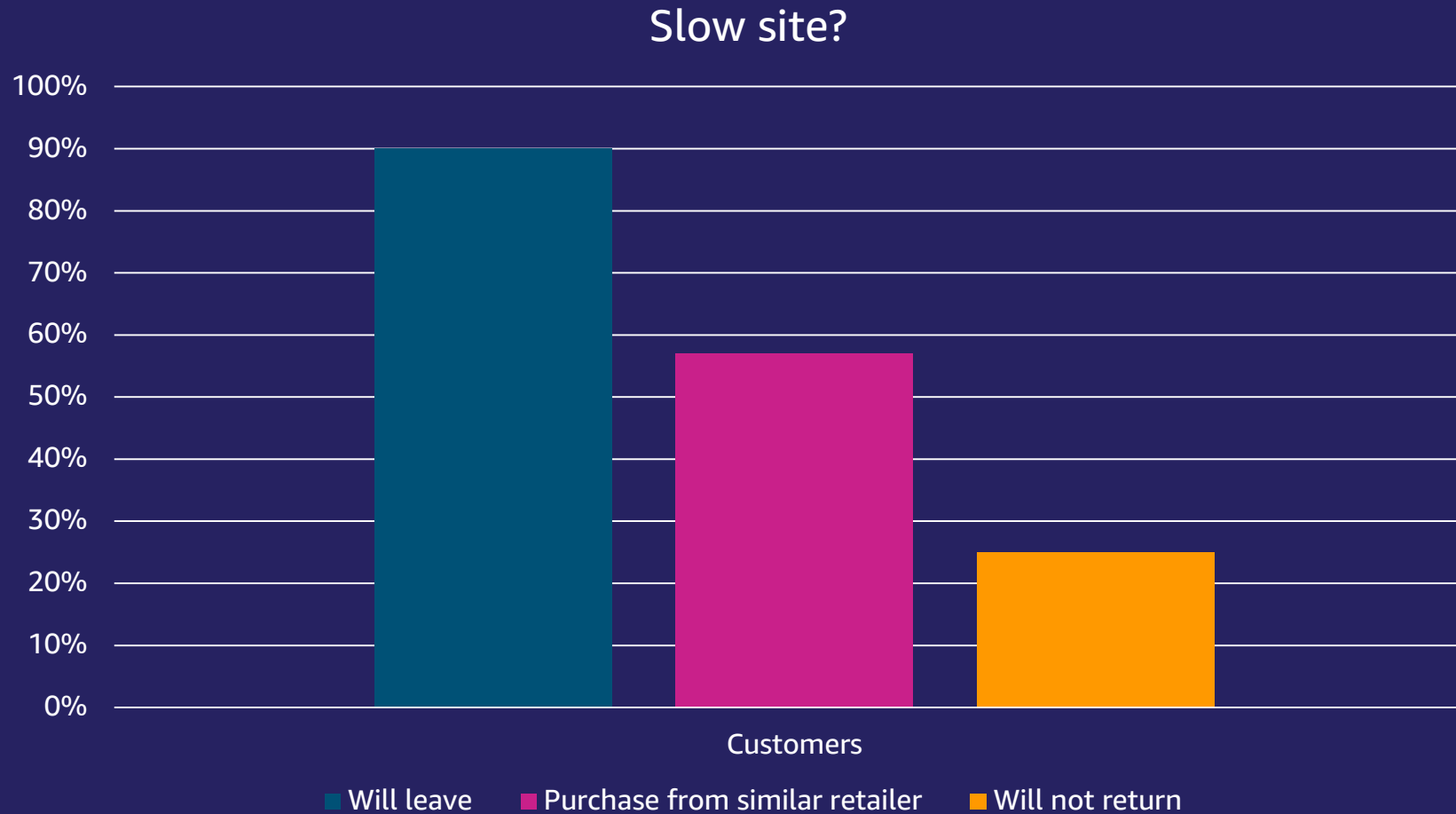
<https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp>

“The brain can identify images seen for as little as 13 milliseconds.”

– 2014 MIT Study

<https://news.mit.edu/2014/in-the-blink-of-an-eye-0116>

Why Performance Matters



<https://www.businessnewsdaily.com/15160-slow-retail-websites-lose-customers.html>

The Need for Speed

FAST: Memory is at least 50x faster than SSDs

PREDICTABLE: Key-based index, no disk seek time

μ s is the new *ms*

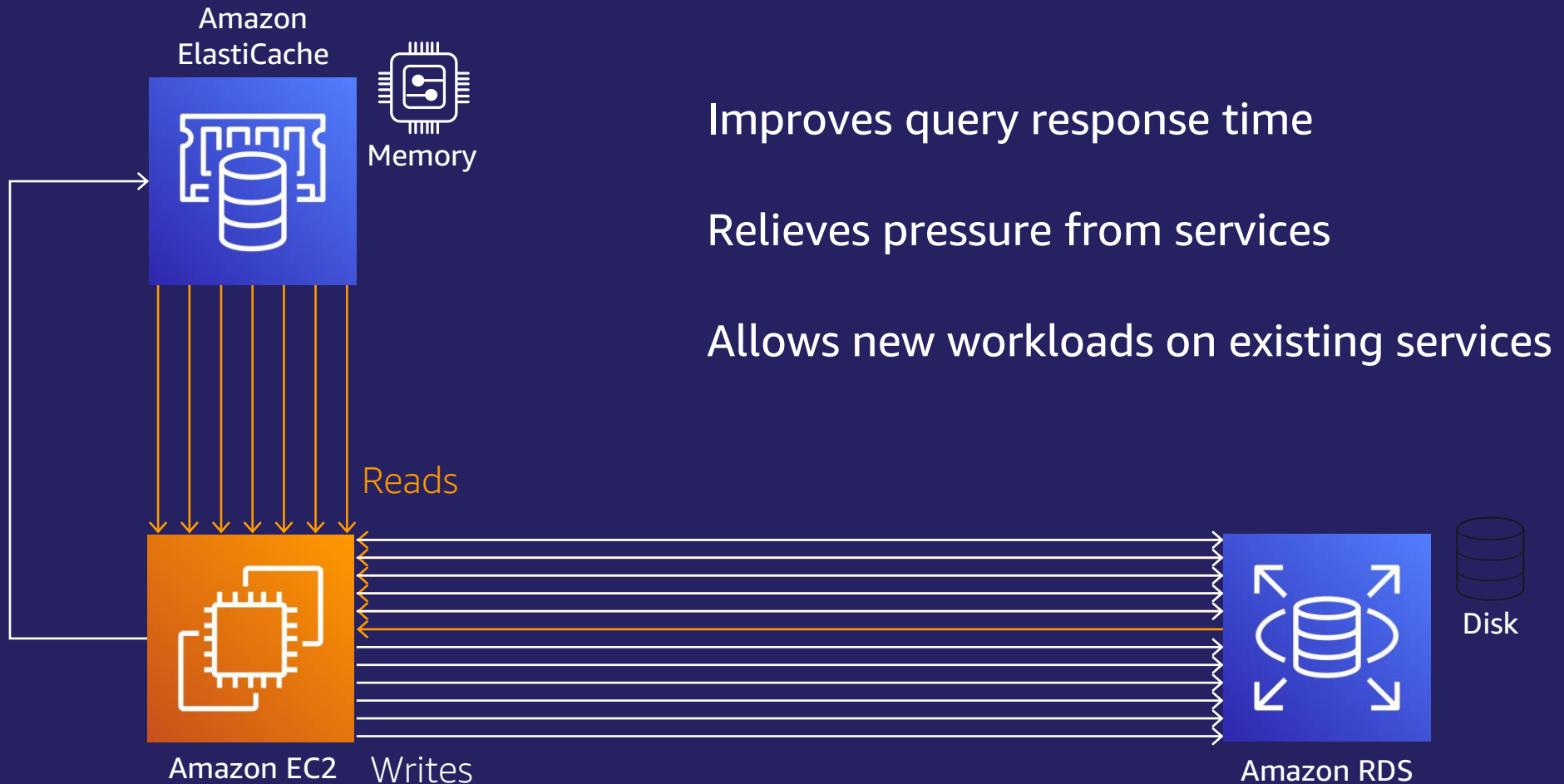


Amazon
ElastiCache

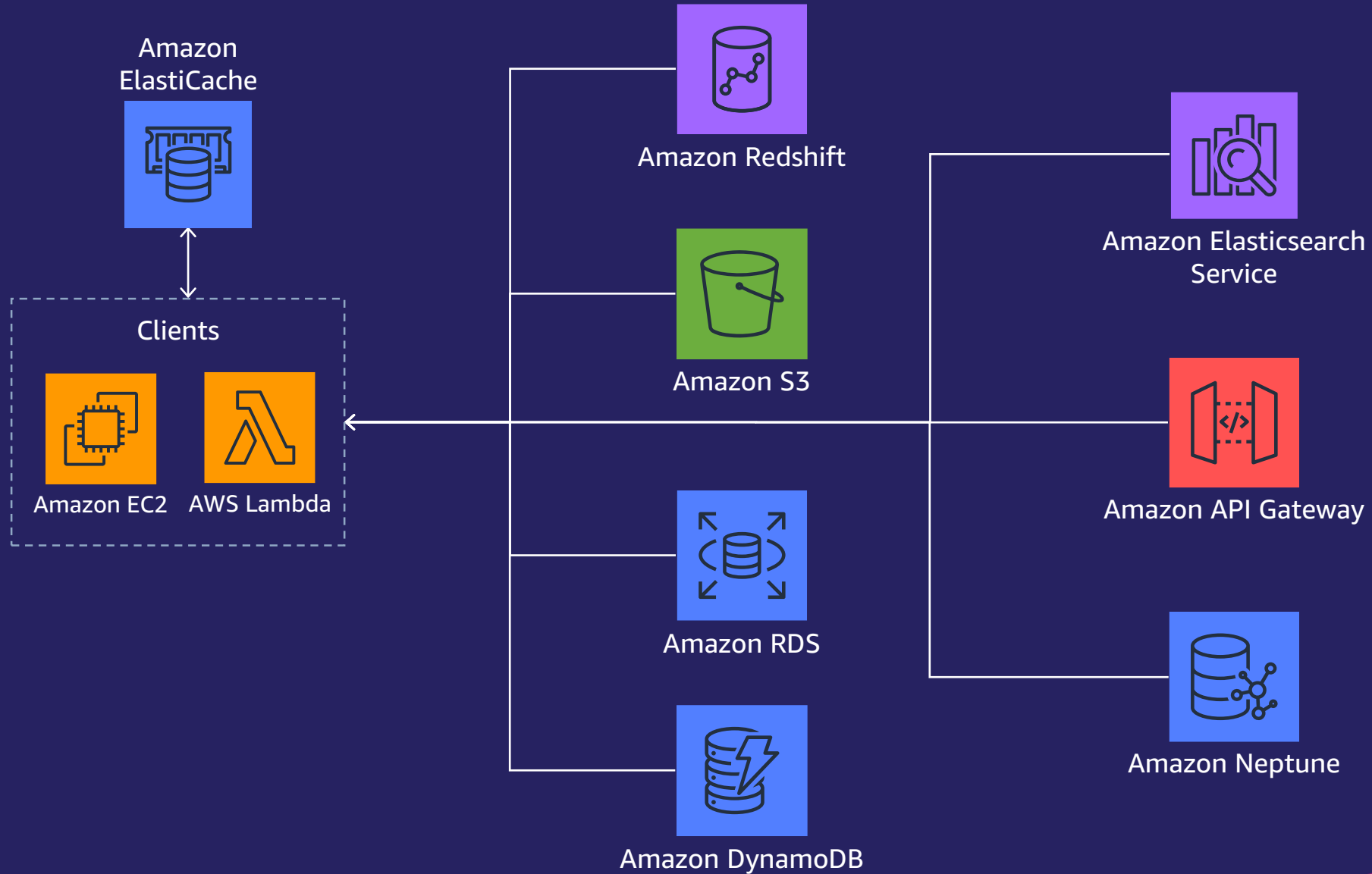
Caching Fundamentals



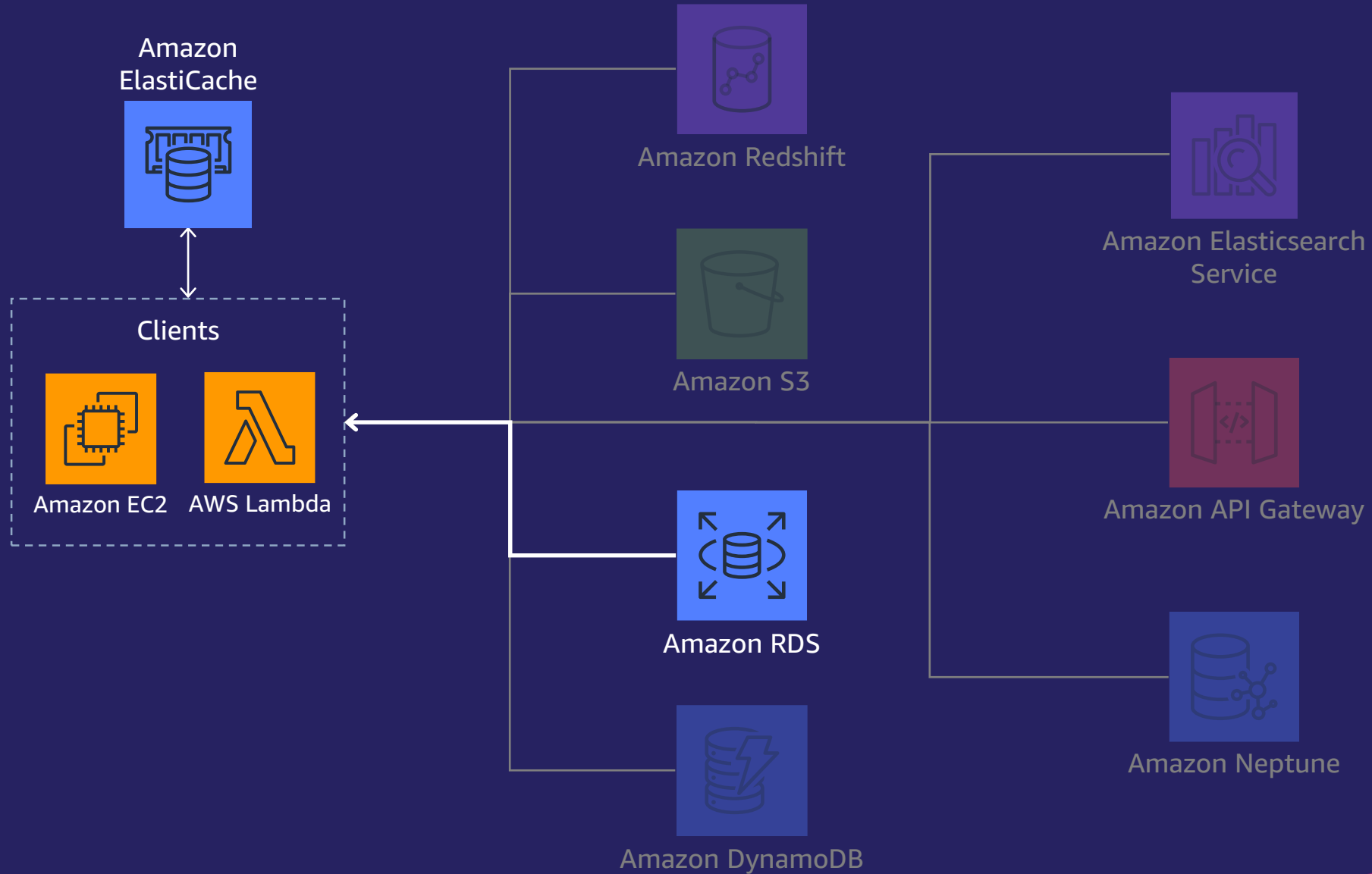
Caching Fundamentals



Caching Fundamentals



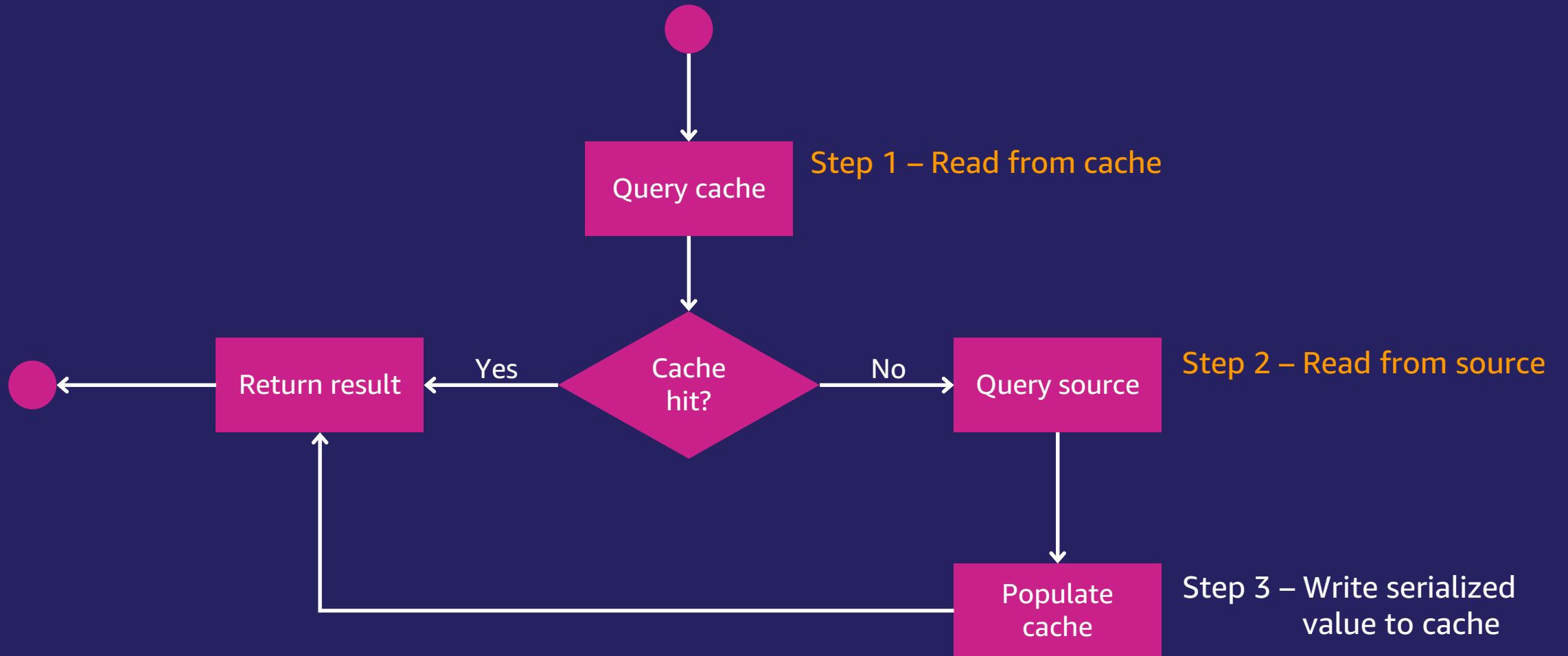
Caching Fundamentals



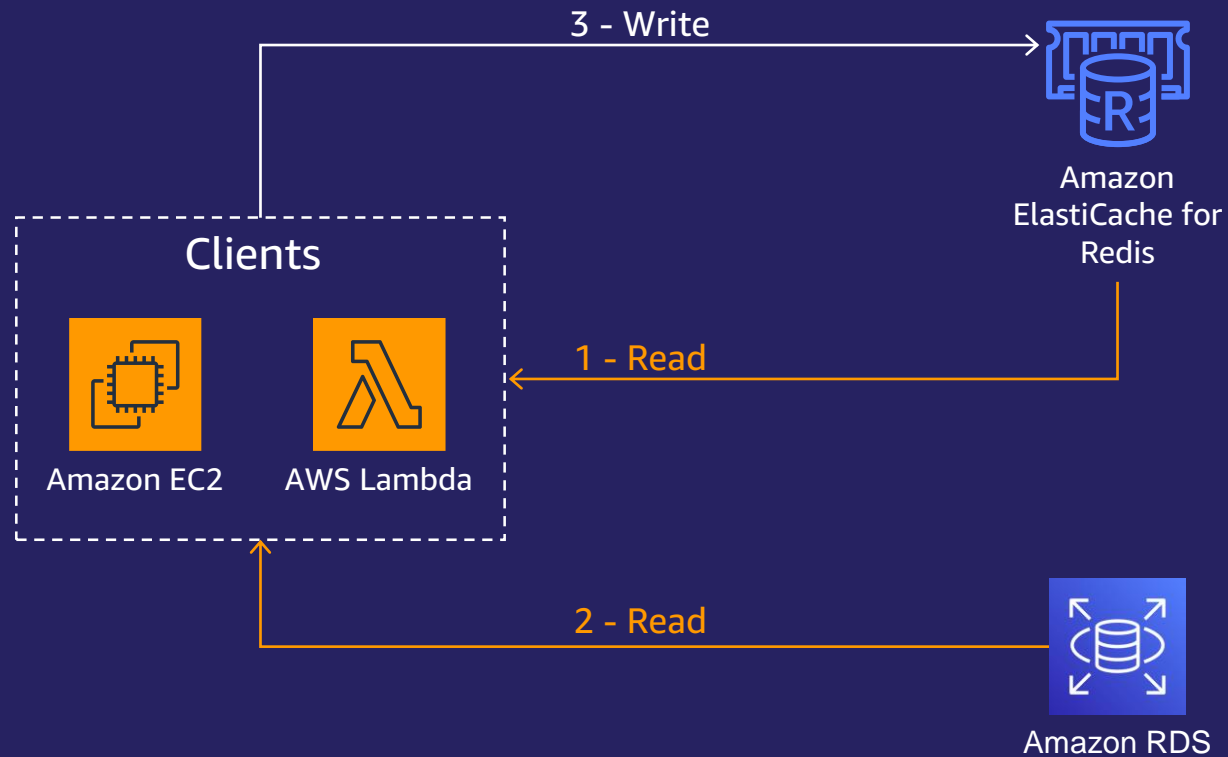
Lazy Loading Pattern



Lazy Loading



Lazy Loading



1. Read from cache

2. Read from source (if miss)

3. Write serialized value to cache

Lazy Loading

```
def fetch(sql):  
    key=get_md5_hash(sql)  
  
    if r.get(key) is not None:  
        return pickle.loads(value)  
  
    else:  
        cursor=m.cursor()  
        cursor=execute(sql)  
        value=cursor.fetchall()  
        r.setex(key, TTL, pickle.dumps(value))  
        return value
```

1. Read from cache

2. Read from source (if miss)

3. Write serialized value to cache

Lazy Loading

```
def fetch(sql):  
  
    key=get_md5_hash(sql)  
  
    if r.get(key) is not None:  
        return pickle.loads(value)  
  
    else:  
        cursor=m.cursor()  
        cursor=execute(sql)  
        value=cursor.fetchall()  
        r.setex(key, TTL, pickle.dumps(value))  
        return value
```

```
SELECT  
    COUNT(*) FROM users  
WHERE . . .
```

```
SELECT  
    customers.customer_id,  
    reviews.review_id  
FROM customers, reviews  
WHERE . . .
```

Amazon RDS Caching Example

1. Read from cache

2. Read from source

3. Write serialized
value to cache

```
#!/usr/bin/env python3
import pymysql, redis, pickle, hashlib

m = pymysql.connect('rds_host')
r = redis.Redis(host='elasticache_endpoint')

def fetch(sql, ttl):
    key = hashlib.md5(sql.encode()).hexdigest()
    # converts lengthy SQL to 32-byte string

    if r.get(key) is not None:
        return pickle.loads(value)
    else:
        cursor=m.cursor()
        cursor=execute(sql)
        value=cursor.fetchall()
        r.setex(key, ttl, pickle.dumps(value))
        return value

def main():
    user_count = fetch("SELECT COUNT(*) FROM users", 5000)
    user_one = fetch("SELECT first, last FROM users WHERE id=1", 60000)

if __name__ == "__main__":
    main()
```

Caching Demo



Benefits of Caching

Helps scale read-heavy workloads

Helps reduce IOPS and DB IO costs

Reduces database over provisioning

Reduces need of multiple database read replicas

Works with all RDS and Aurora database engines

Amazon ElastiCache



Amazon ElastiCache



Fully managed

AWS manages all hardware and software setup, configuration, monitoring.



Scalable

Write and memory scaling with sharding.
Non-disruptive scaling.
Read scaling with replicas (Redis).



Extreme performance

In-memory data store and cache for sub-millisecond response times.



OSS compatible

Fully compatible with open source Redis and Memcached engines.

Community

“Most popular key-value store” since 2013

Rank			DBMS	Database Model	Score		
Jun 2021	May 2021	Jun 2020			Jun 2021	May 2021	Jun 2020
1.	1.	1.	Redis +	Key-value, Multi-model ⓘ	165.25	+3.08	+19.61
2.	2.	2.	Amazon DynamoDB +	Multi-model ⓘ	73.76	+3.69	+8.90
3.	3.	3.	Microsoft Azure Cosmos DB +	Multi-model ⓘ	36.47	+1.76	+5.67
4.	4.	4.	Memcached	Key-value	25.18	+0.68	+0.37

– DB-Engines.com

<https://db-engines.com/en/ranking/key-value+store>

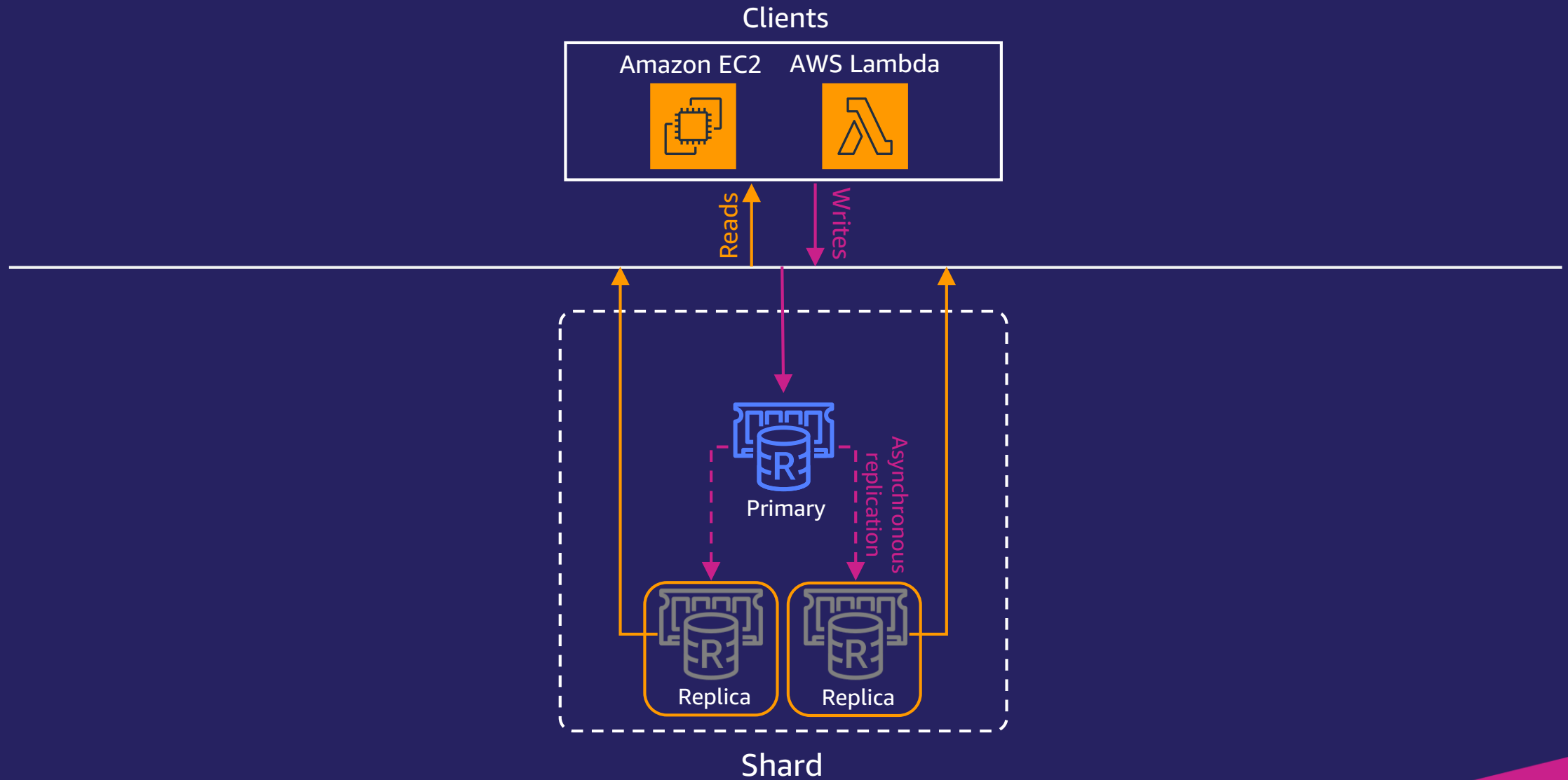
“Most loved database”



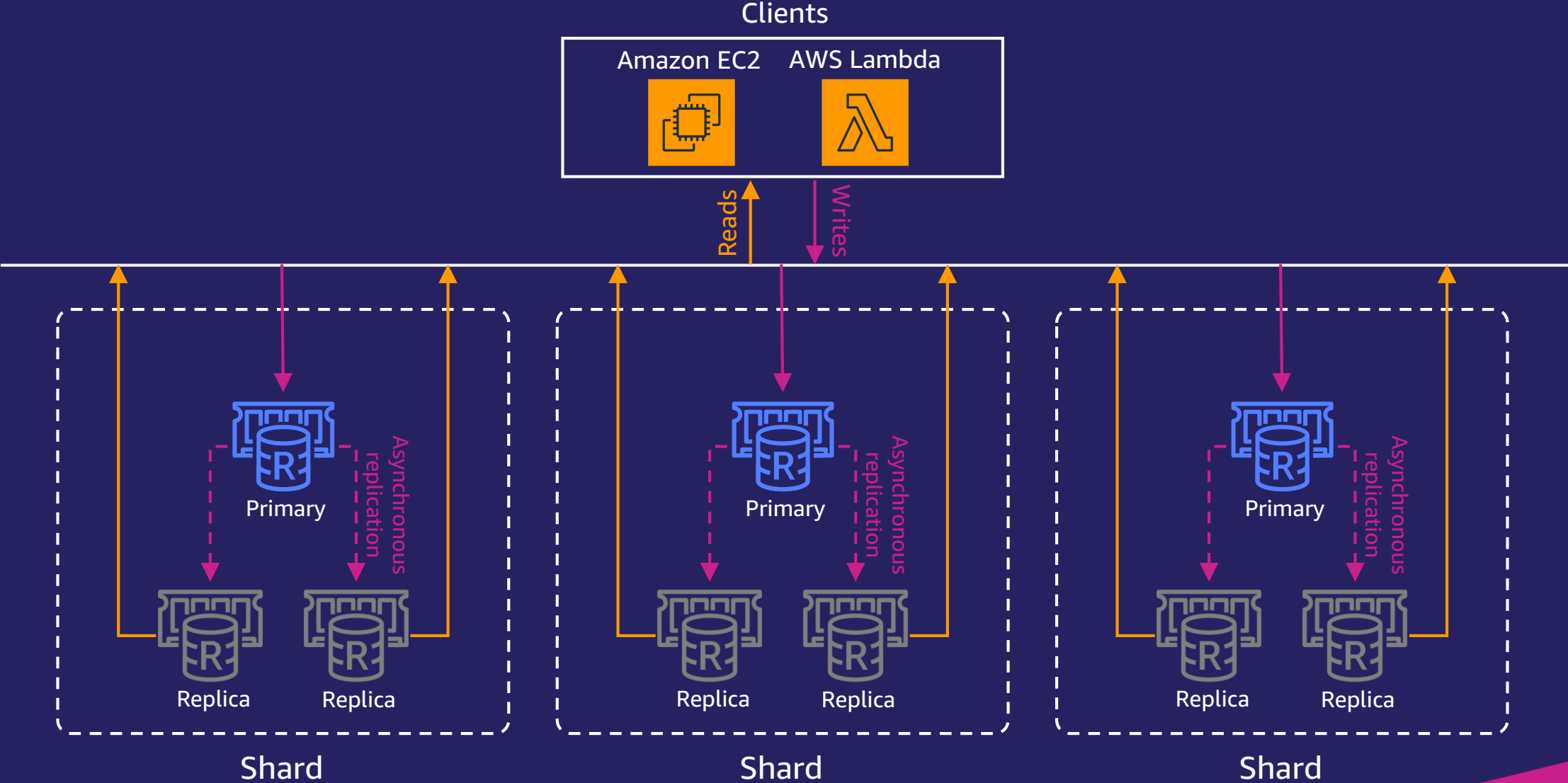
- Stack Overflow

<https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-databases-loved4>

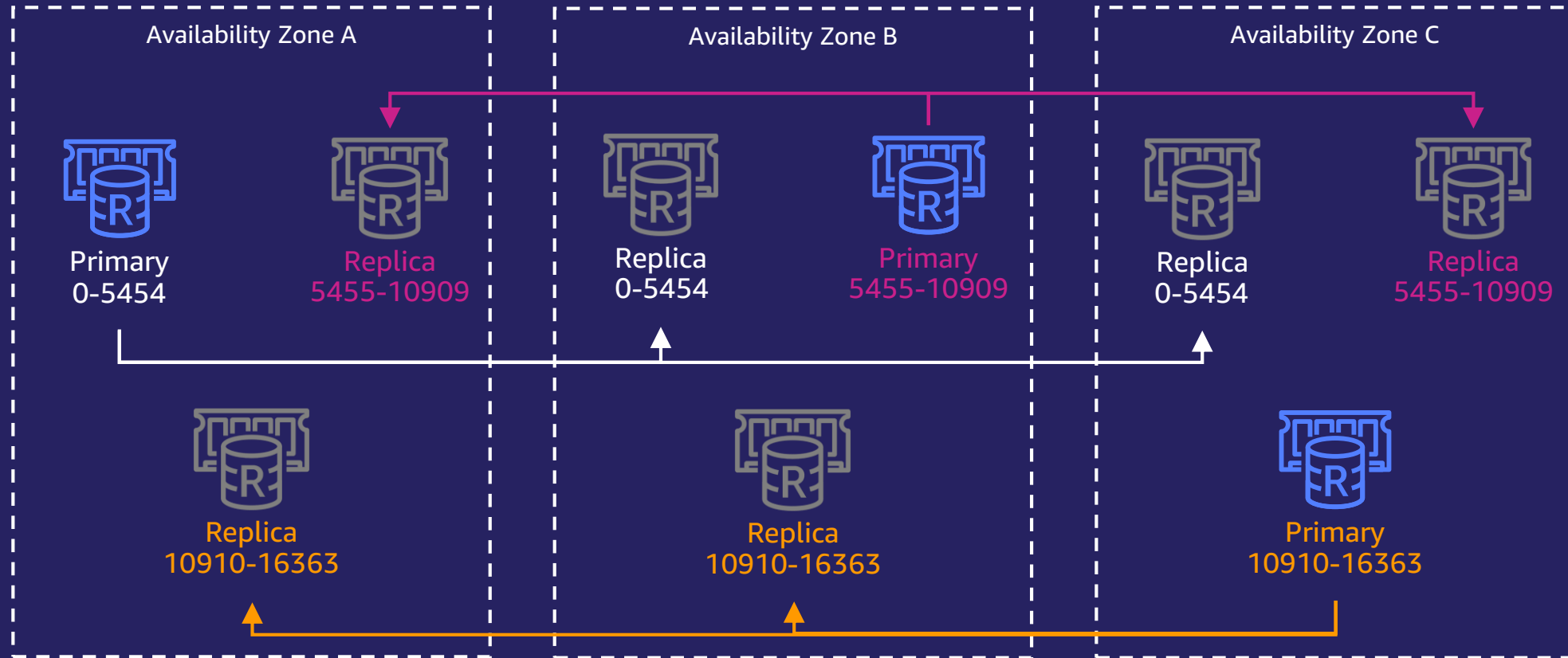
Online Scaling



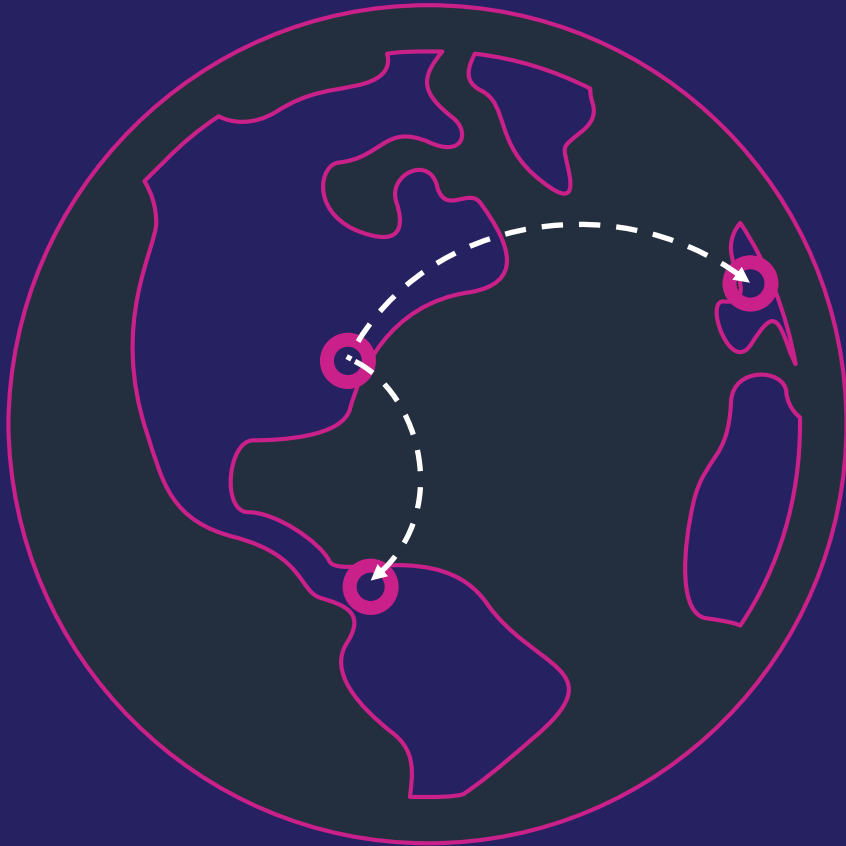
Online Scaling



High Availability



Amazon ElastiCache for Redis - Global Datastore



Fully managed, fast, reliable and secure cross-region replication

Disaster Recovery

Low latency reads

Replication typically < 1s

Q&A





Thank you!



Poll Questions

* Poll question before demo section:

1) What is your typical database query response time? (just before demo)

- Over 1 second
- Under 1 second
- Under 100 milliseconds
- Under 10 milliseconds

* Poll Question before Q&A section:

2) What benefit is associated with adding a distributed cache to RDS?

- Decreases latency for queries.
- Reduces database overprovisioning.
- Controls costs by minimizing RDS IOPS and DB I/O.
- Scalable to hundreds of terabytes of in-memory storage.
- Works with all RDS and Aurora DB engines.
- All of the above