INNOVATE

DATA EDITION

19 August 2021

Extreme performance at cloud scale: Supercharge your real-time applications with Amazon ElastiCache

Damon LaCaille, Sr. Solutions Architect

Agenda

- In-memory Datastore Fundamentals
- Caching Concepts
- Lazy Loading Pattern
- Amazon ElastiCache for Redis
- Best Practices

In-memory Datastore Fundamentals

Datastore Trait Comparison

Feature	Disk	In-memory
Writes / Reads	Disk	Memory
Engine latency	Milliseconds (ms)	Microseconds (µs)
Performance bottleneck	Disk	Network
Throughput	Moderate	High
Data	Data models	Rich data structures

Why Performance Matters

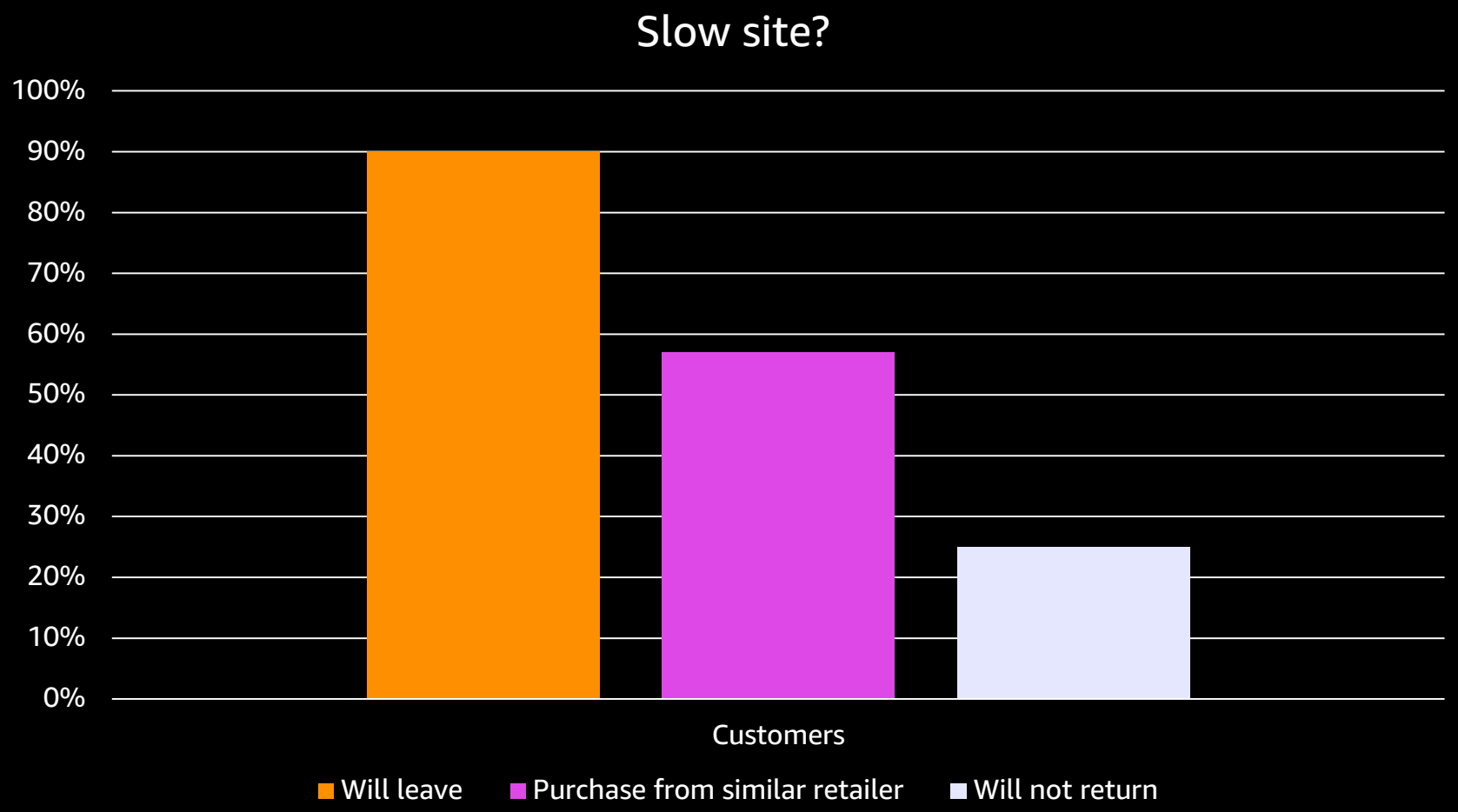
“A 100-millisecond delay in website load time can hurt conversion rates by 7 percent.”

"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

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

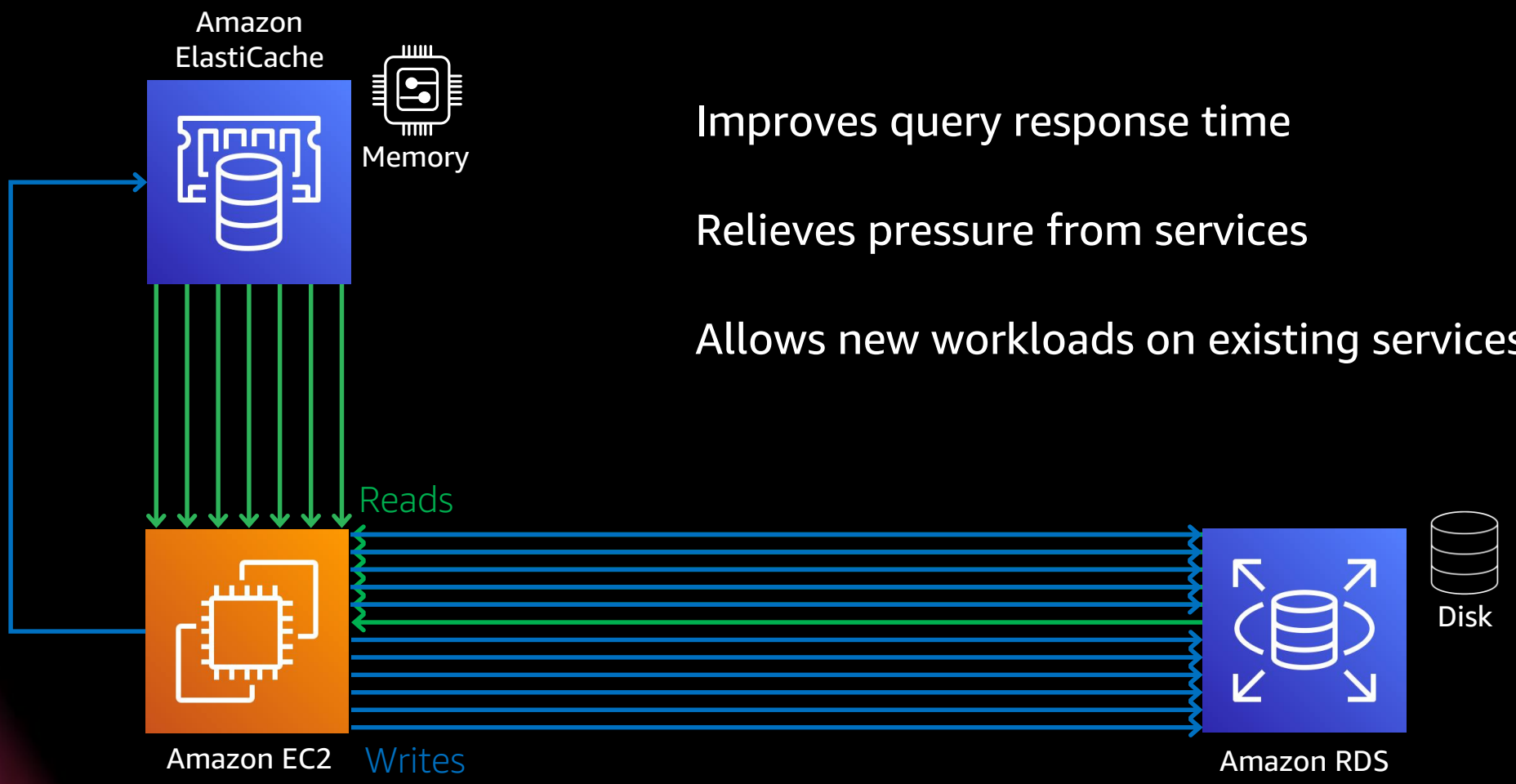
PREDICTABLE: Key-based index, no disk seek time

µs is the new ***ms***

Amazon
ElastiCache

Caching Concepts

Caching Concepts

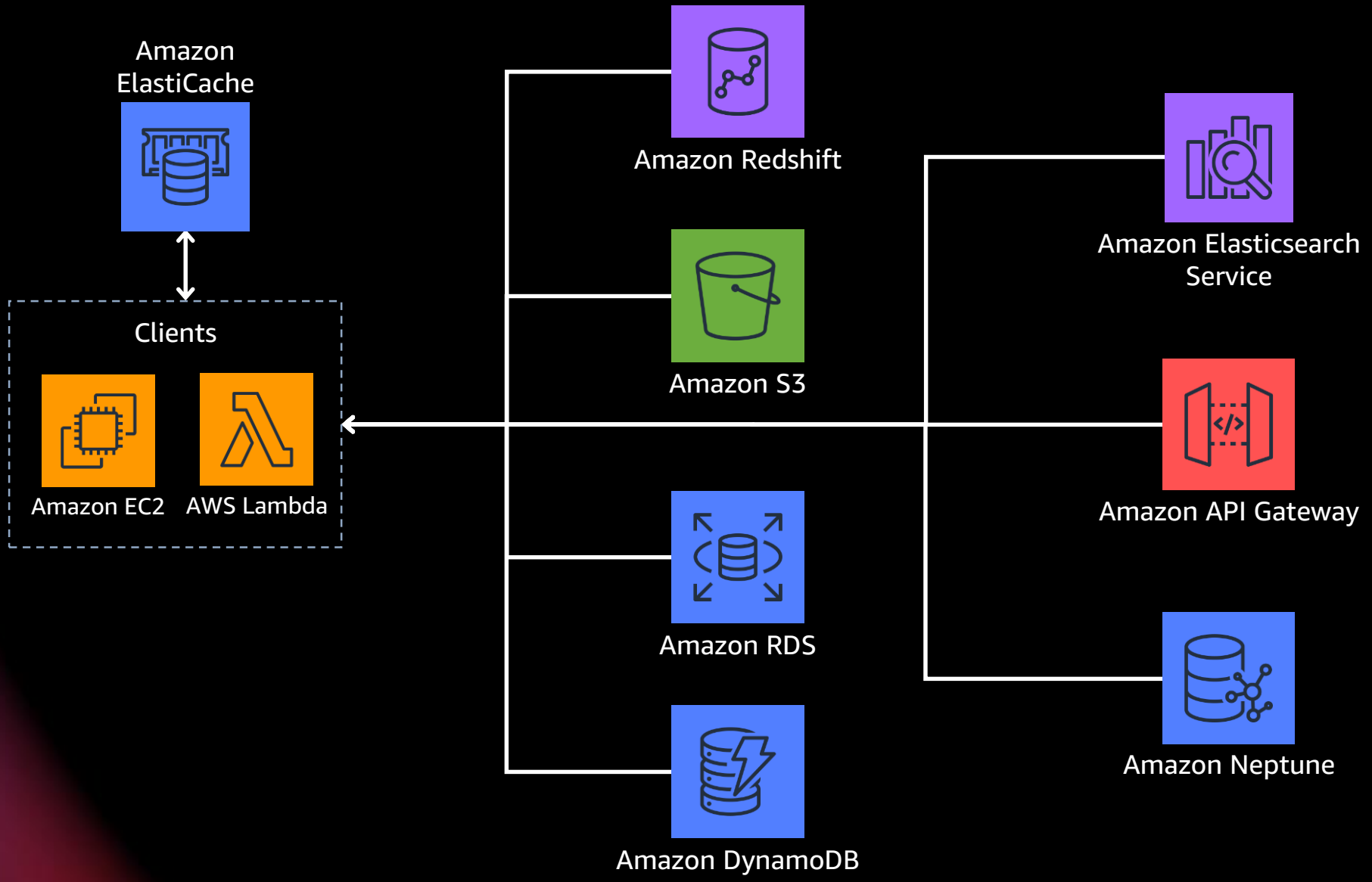


Improves query response time

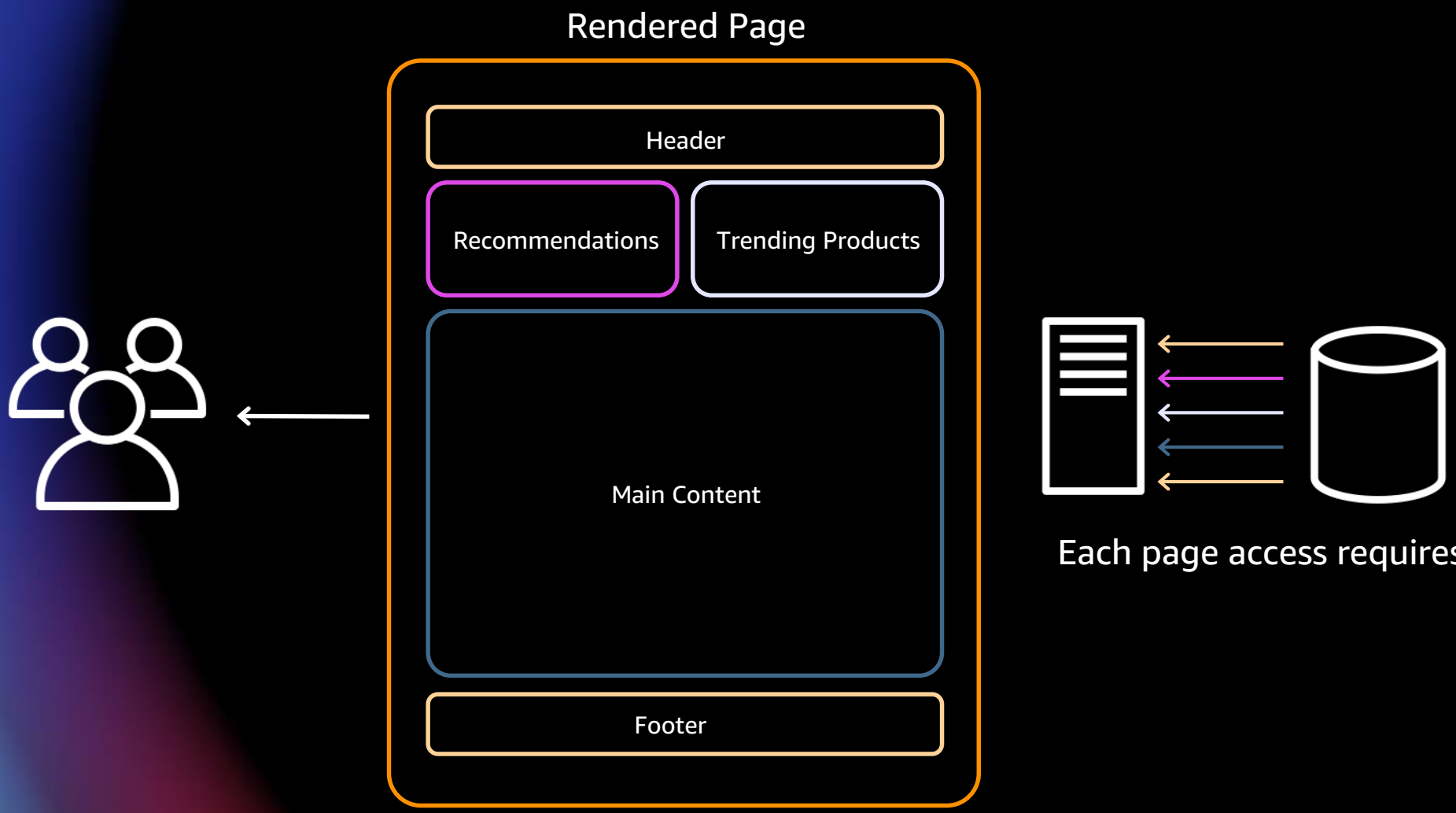
Relieves pressure from services

Allows new workloads on existing services

Caching Concepts

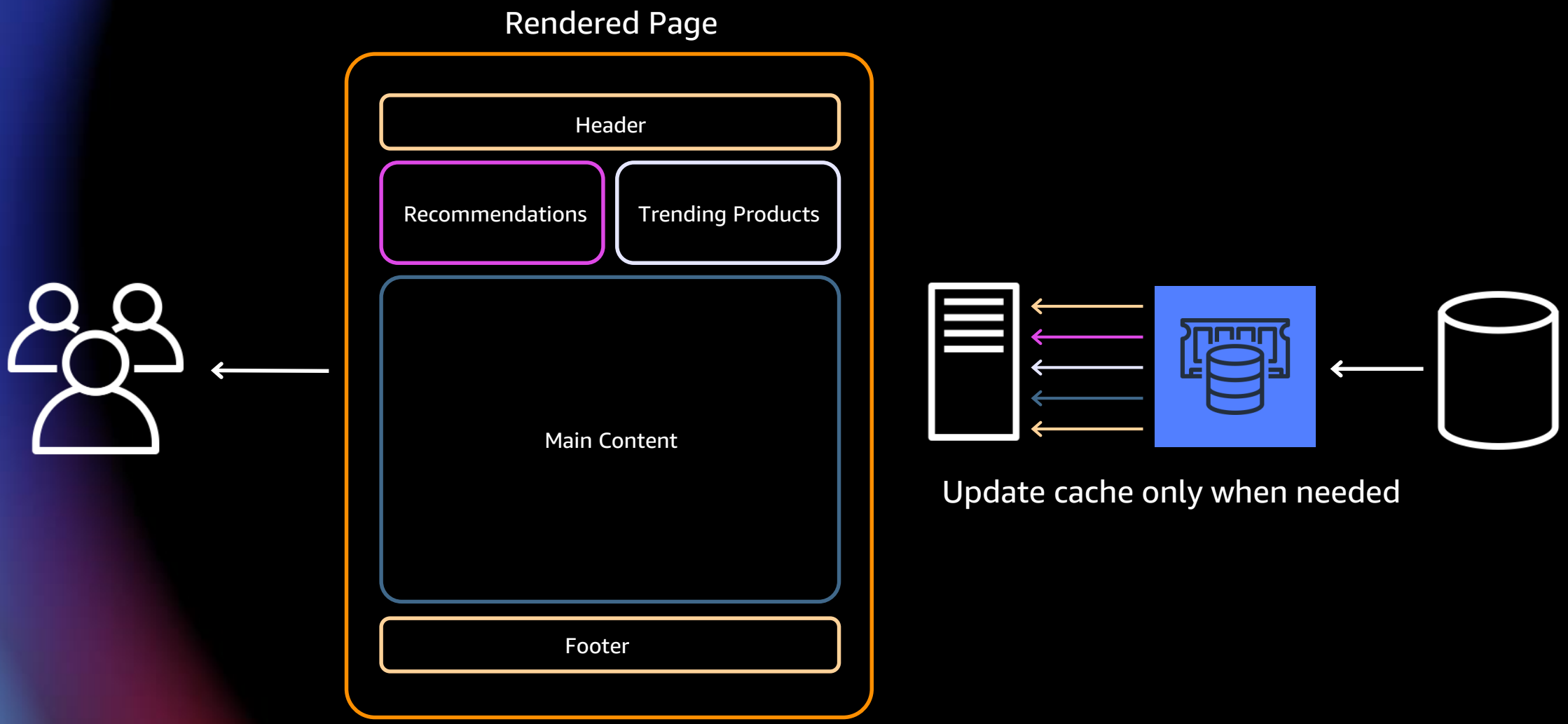


Caching Concepts

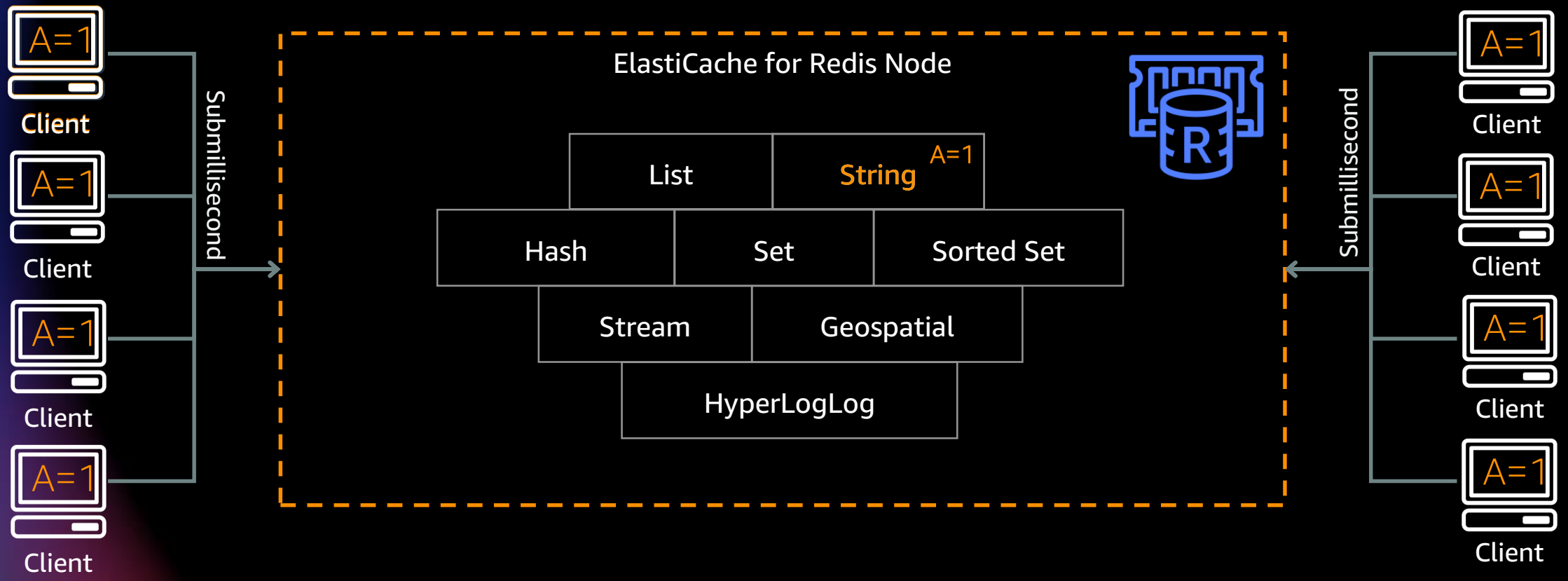


Each page access requires multiple database queries

Caching Concepts

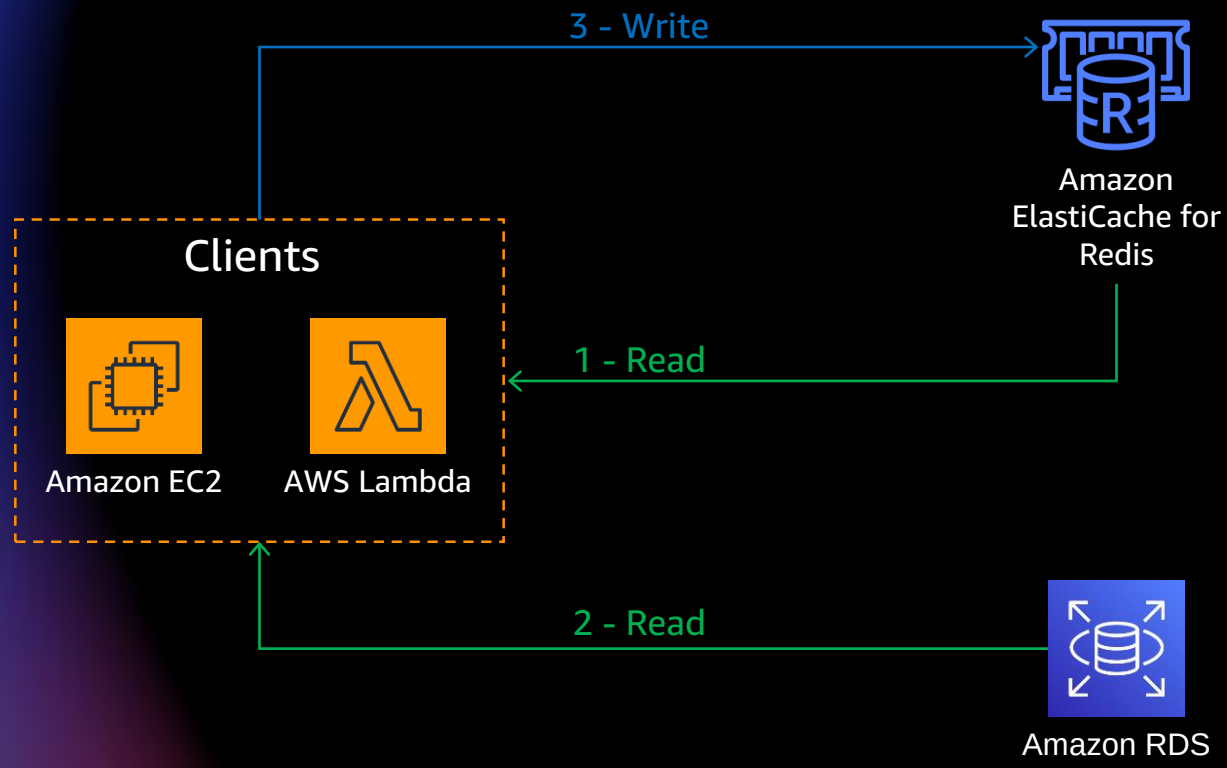


Caching Concepts - Data Structures



Lazy Loading Pattern

Lazy Loading



1. Read from cache
2. Read from source (if miss)
3. Write to cache

Advantages

- Avoids unnecessary data in cache
- Cache can be repopulated at anytime
- Immediate benefit

Disadvantages

- Cache miss may be expensive
- Data "freshness" factor

Lazy Loading

```
> SELECT
    customers.name, SUM(orders.num_items)
  FROM
    customers, orders
  WHERE
    customers.id = X AND
    orders.customer_id = customers.id
```

MD5 Hash

"ecf361704f15aa0e26e3b24e1ce6d1d6"

Key

ecf361704f15aa0e26e3b24e1ce6d1d6

~32 bytes

Value

"\x80\x03M\xd2\x06cdecimal\nDecimal\n…"

~140 bytes

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 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

Step 1: Query cache

Step 2: Read from source

Step 3: Write to cache

```python
import pymysql, redis, pickle, hashlib

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

### Pass in SQL string
sql = "SELECT ... FROM ..."

### Convert SQL string into a shorter, unique hash for use as Redis key
key = hashlib.sha224(sql.encode('utf-8')).hexdigest()

### Check for value in Redis
value = r.get(key)

if value is None:

    print ("Cache Miss")

    ### Fetch result set from RDS
    cursor=m.cursor()
    cursor.execute(sql)
    value = cursor.fetchall()

    ### Store full result set in Redis
    r.psetex(key, pickle.dumps(value))
    return value

else:

    ### Return cached result set from Redis
    print ("Cache Hit")
    return pickle.loads(value)
```

Amazon S3 Caching Example

Step 1: Query cache

Step 2: Read from source

Step 3: Write to cache

```
import boto3
import redis

r = redis.StrictRedis(host="elasticache_endpoint", port=6379)
s3 = boto3.resource('s3')

### Pass in s3_bucket_name and s3_object_key and
### check to see if value is in Redis
value = r.get(s3_bucket_name + ':' + s3_object_key)

if value is None:

    print("Cache Miss")

    ### Get data from S3
    obj = s3.Object(s3_bucket_name, s3_object_key)
    data = obj.get()['Body'].read().decode('utf-8')

    ### Store the data into Redis
    r.set(s3_bucket_name + ':' + s3_object_key, data)

else:

    print("Cache Hit")
    print("Data retrieved from redis = " + value)
```

Amazon ElastiCache for Redis

Amazon ElastiCache

Fully managed

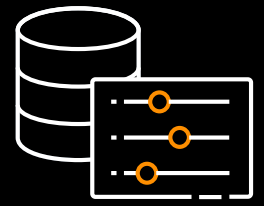
AWS manages all hardware and software setup, configuration, monitoring.

Extreme performance

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

Scalable

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



OSS compatible

Fully compatible with open source.

Redis Community

"Most popular key-value store"

Rank			DBMS	Database Model	Score		
Dec 2020	Nov 2020	Dec 2019			Dec 2020	Nov 2020	Dec 2019
1.	1.	1.	Redis	Key-value, Multi-model	153.63	-1.79	+7.39
2.	2.	2.	Amazon DynamoDB	Multi-model	69.12	+0.23	+7.49
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	33.54	+1.04	+2.11
4.	4.	4.	Memcached	Key-value	25.89	+0.15	+1.43

– DB-Engines.com

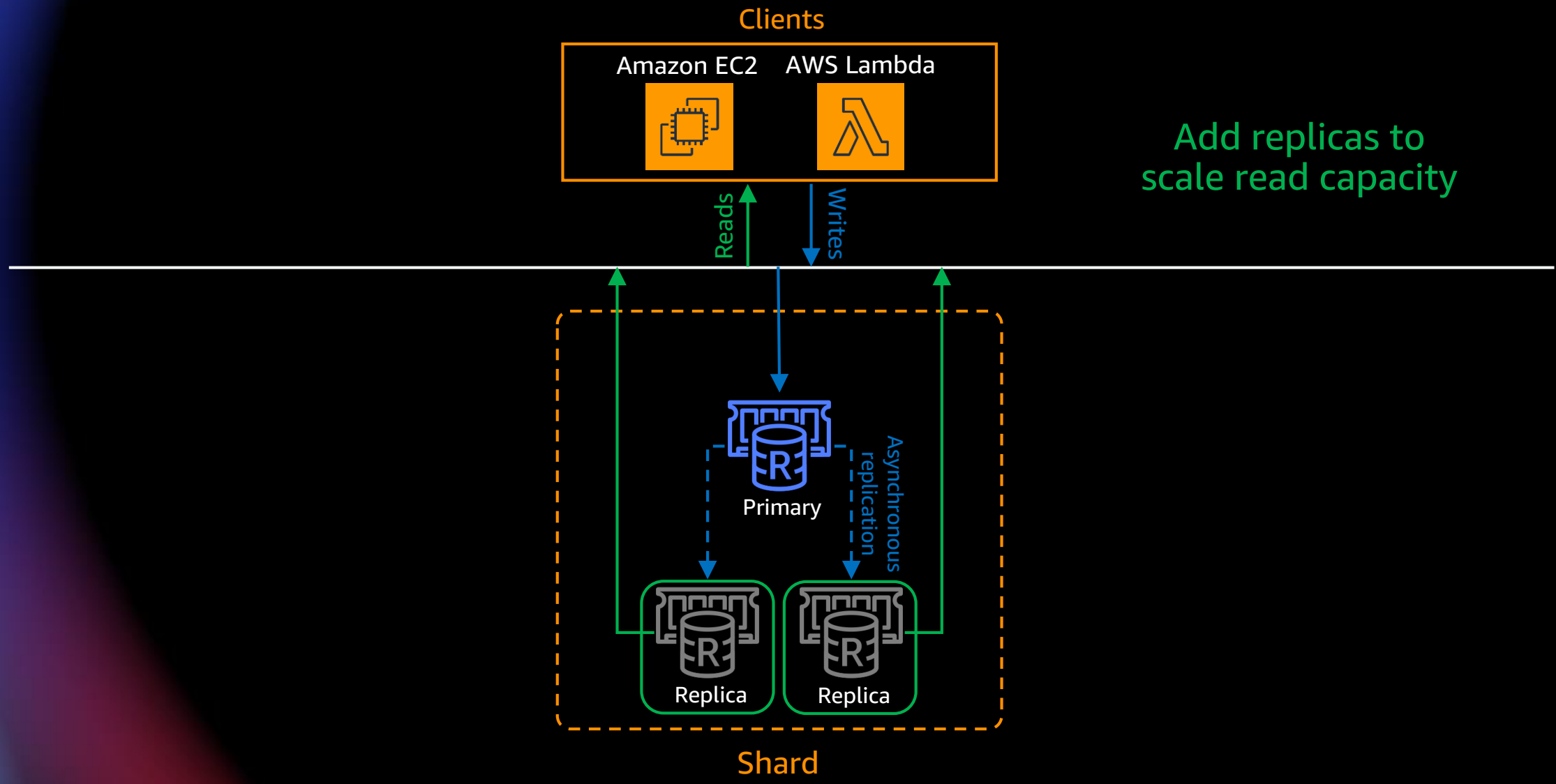
https://db-engines.com/en/ranking/key-value+store

"Most loved database"

- Stack Overflow

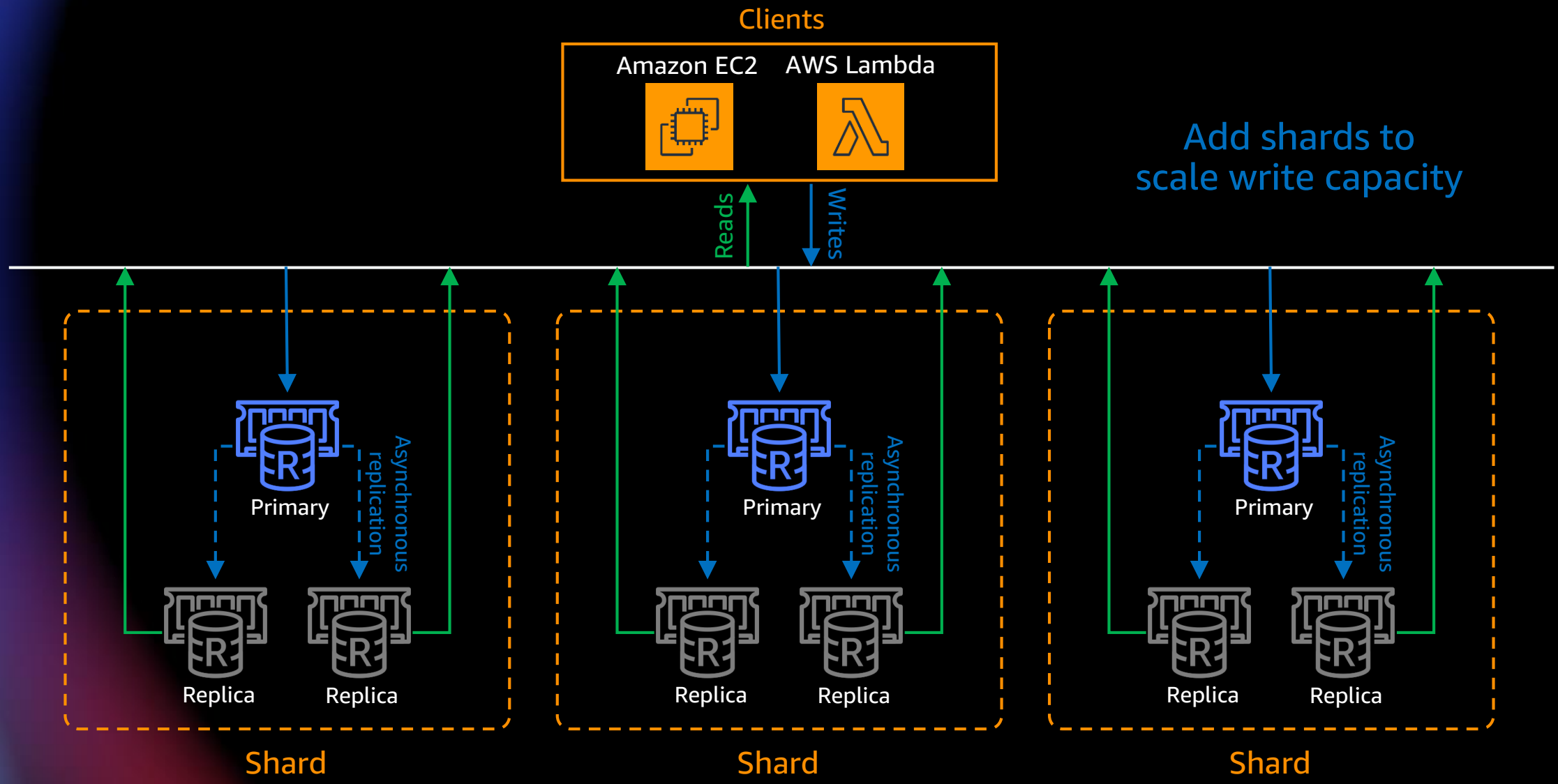
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-databases-loved4

Scaling Reads

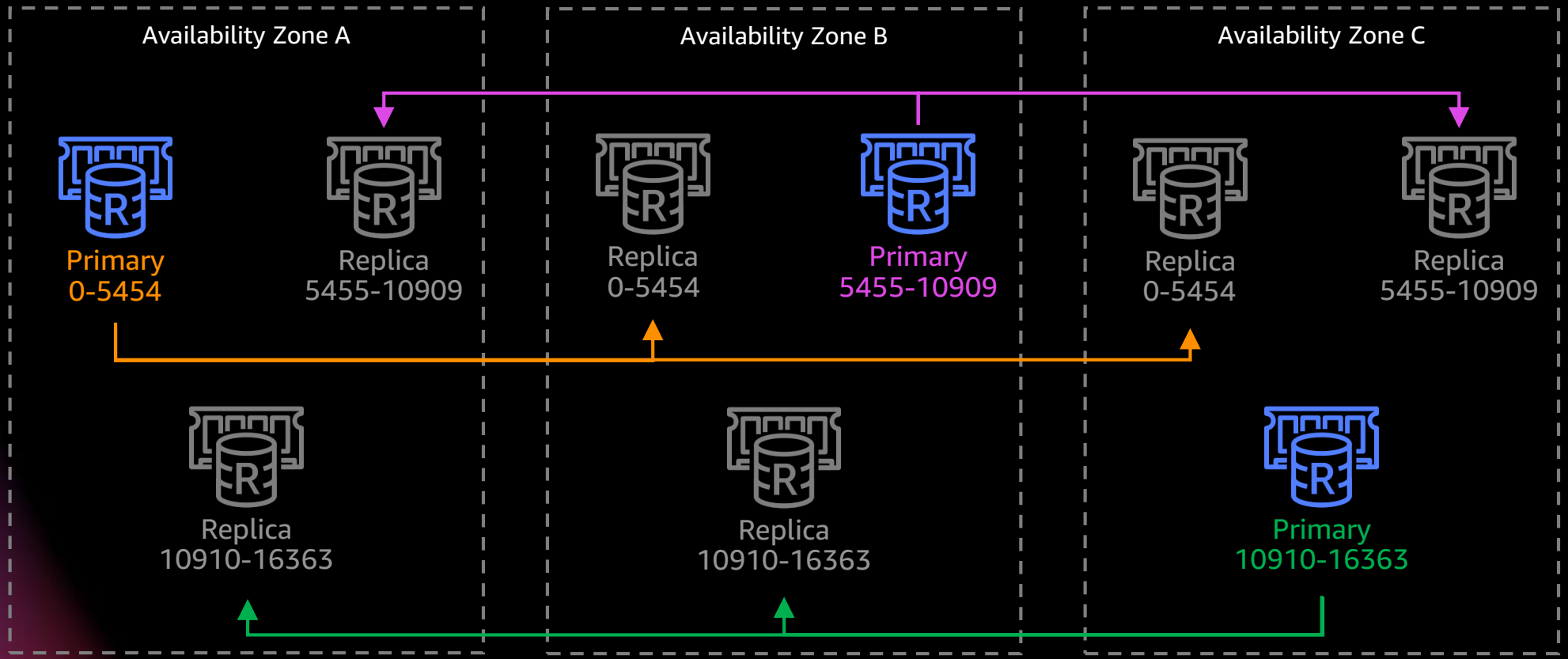


Add replicas to scale read capacity

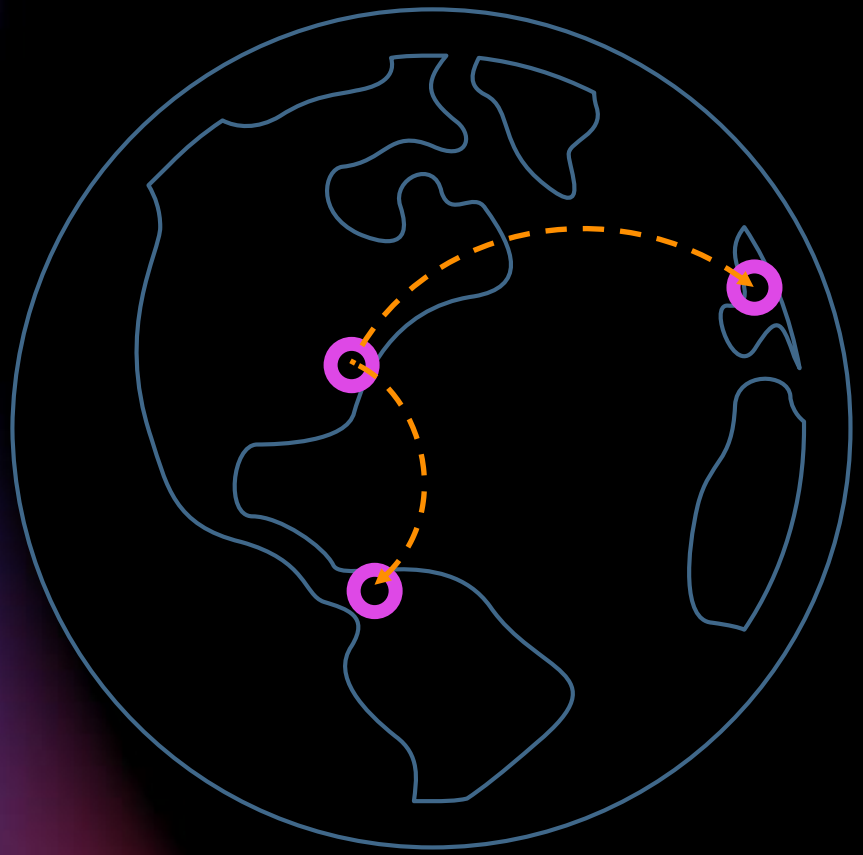
Scaling Writes



High Availability



Global Datastore



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

- Disaster Recovery
- Low latency reads
- Replication typically < 1s

Best Practices

Best Practices - Evictions

Keys can be automatically removed when memory is full

Policy Name	Description
allkeys-lru	Evicts the least recently used (LRU) regardless of TTL set
volatile-lru*	Evicts the least recently used (LRU) from those that have a TTL set
allkeys-lfu	Evict any key using approximated least frequently used (LFU)
volatile-lfu*	Evict using approximated LFU among the keys with a TTL set
volatile-ttl*	Evicts the keys with shortest TTL set
volatile-random*	Randomly evicts keys with a TTL set
allkeys-random	Randomly evicts keys regardless of TTL set
no-eviction	Doesn't evict keys at all. This blocks future writes until memory frees up.

** Volatile policies only evicts keys with TTLs*

Best Practices - Time To Live (TTL)

Assign freshness factor / expiry

Explicitly assign to individual keys (they are not global)

Assign during creation / modification of key

Some keys should not have TTLs

Set TTL as relative (`EXPIRE`/`PEXPIRE`) or fixed time (`EXPIREAT`)

Add a random jitter (+/-)

Best Practices - Clients

Connections

- Connection pooling reduces client and server CPU overhead
- Use exponential back-off for reconnects

Reads or Writes

- Connect to read replicas ('`readonly`' parameter) for reads
- Use the reader endpoint for replicas (cluster-mode-disabled only)

Performance

- Pipelines greatly increase throughput for bulk inserts
- Use '`CLIENT REPLY {OFF|ON|SKIP}`' to control server responses
- Ensure client library supports Redis cluster mode

Visit the AWS Data Resource Hub

Dive deeper with these resources, get inspired and learn how you can use data to make better decisions and innovate faster.

Building a winning data strategy

The new leadership mindset for data & analytics

Harness data to reinvent your organization

Put your data to work with a modern analytics approach

Breaking free from on-premises database constraints

Cloud storage adoption: From cost optimization to agility & innovation

A strategic playbook for data, analytics, and machine learning

… and more!

https://tinyurl.com/aws-data-resource

Visit resource hub

AWS Training and Certification

Empower your teams with comprehensive training

By building skills with AWS Training and Certification, businesses and individuals can see the bigger picture understanding the reasoning behind every data point. As training progresses and teams become data-fluent, previously hidden insights come into view.

Leverage free digital training

Learn how to harness the world's most valuable resource: data. Access digital and virtual instructor-led courses on data analytics and databases built by the experts at AWS and start your learning journey to become data-driven.

Take a digital course »

Get certified

Earn industry-recognized credibility and set tangible goals for success with industry-recognized certifications, like *AWS Certified Data Analytics – Specialty.*

Learn more »

Ramp-up your skills

Deep dive into new topics and focus on knowledge gaps at your own pace with the *AWS Ramp-Up Guide: Database* and *AWS Ramp-Up Guide: Data Analytics*. With a wide range of whitepapers, blog posts, videos, webinars and peer resources available for data professionals to leverage for independent learning.

Download ramp-up guides »

Thank you for attending AWS Innovate – Data Edition

We hope you found it interesting! A kind reminder to **complete the survey.**
Let us know what you thought of today's event and how we can improve the event experience for you in the future.


aws-apj-marketing@amazon.com


twitter.com/AWSCloud


facebook.com/AmazonWebServices


youtube.com/user/AmazonWebServices


slideshare.net/AmazonWebServices


twitch.tv/aws

Thank you!