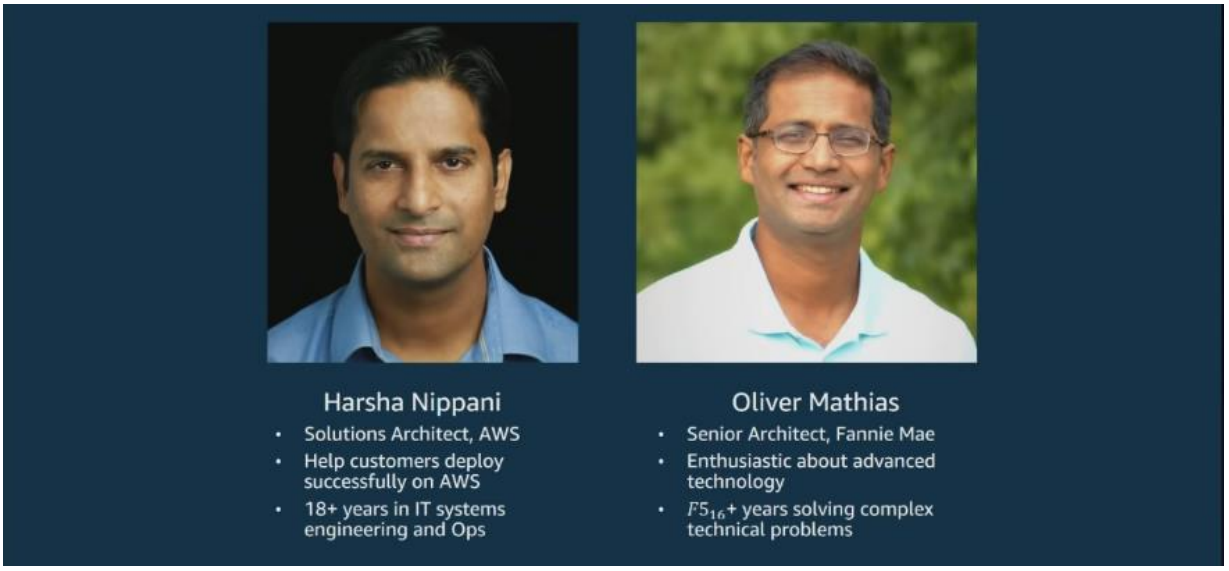

In this session, Fannie Mae discusses how they completely **re-architected a mission-critical application using AWS native services** that **process hundreds of thousands of mortgage loans every day in a highly scalable and reliable manner**. **The transaction-heavy workload uses over 20+ million Amazon S3 transactions a day**, **each within 150-millisecond response times**, thus providing increased uptime and faster response.




# Amazon S3 – Object Store at Scale

## The AWS Storage Portfolio




## Amazon S3 by the Numbers

# Benefits of Amazon S3 & Glacier

Durable, Available, & Scalable

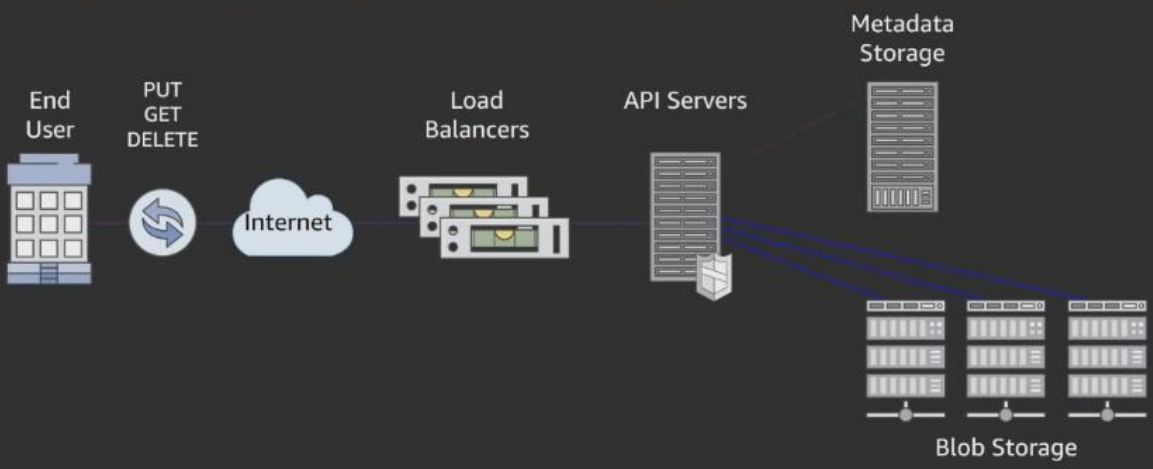
Security & Compliance

Low-Cost

Flexible Management

Ecosystem

# Amazon S3 Architecture




# Amazon S3 Security, Encryption, & Compliance

## One of the broadest set of tools in the industry

### Security

- AWS Identity Access and Management (IAM) and bucket policies
- Access control lists
- Audit logging with AWS CloudTrail & alerts with Amazon CloudWatch
- Secure AWS CloudFormation templates
- Amazon Macie
- Amazon S3 Console permission checks New

### Encryption

- Encryption in transit with TLS
- SSE-S3 – Amazon S3 manages data & keys
- SSE-C – Customer managed keys
- SSE-KMS – Master keys in AWS Key Management Service (AWS KMS)
- CSE – 100% Customer managed New
- Default bucket encryption
- Encryption status in inventory

### Compliance

- PCI-DSS
- HIPAA/HITECH
- FedRAMP
- FISMA
- EU Data Protection Directive

## AWS Storage Customers


## Enterprise Partner Integration (Storage)


Backup and Restore

COMMVAULT

Primary Storage

## High Request Rates

- S3 automatically scales to high request/sec (RPS) rates
  - 3,500 RPS for PUT/POST/DELETE (Per Bucket Prefix)
  - 5,500 RPS for GETS (Per Bucket Prefix)
  - Add prefixes to increase read and write performance exponentially
- SSE-KMS encryption
  - 5,500 – 10,000 RPS for KMS Encrypt/Decrypt actions

## Using a Three or Four Character Hash (Entropy)

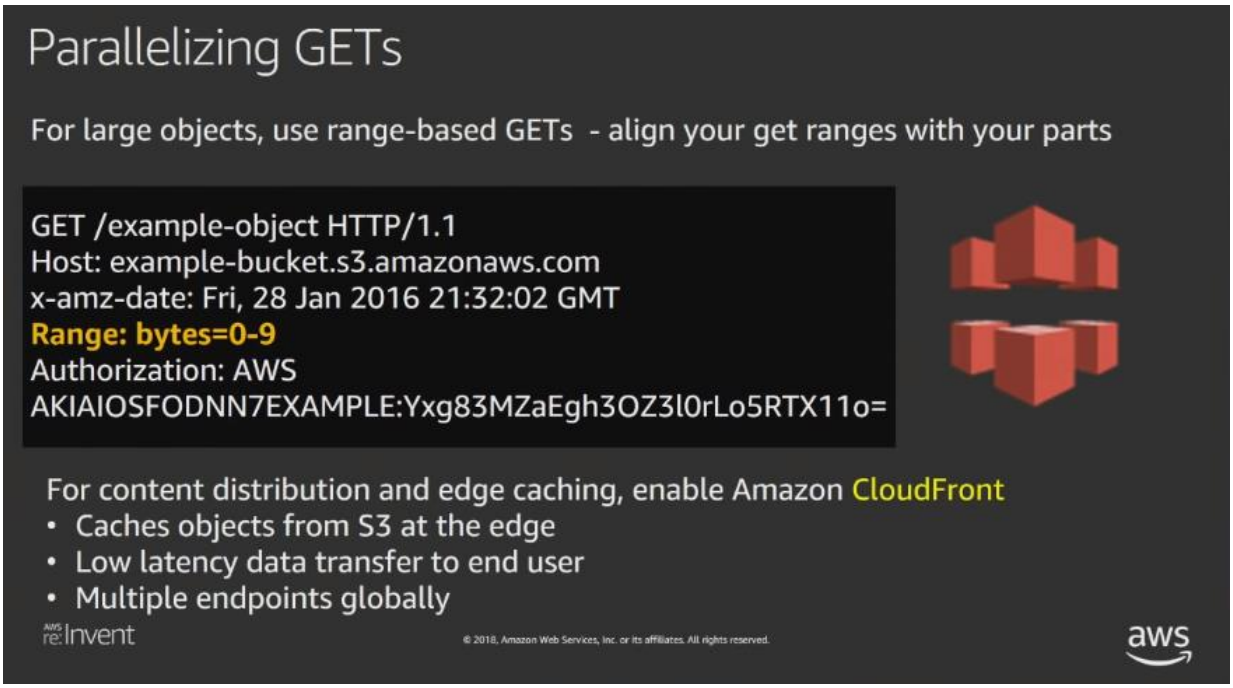
Due to recent Amazon S3 performance enhancements, most customers no longer need to worry about introducing entropy in key names

examplebucket/*232a*-2017-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/*7b54*-2017-26-05-15-00-00/cust3857422/photo2.jpg
*examplebucket/921c-2017-26-05-15-00-00/cust1248473/photo2.jpg*

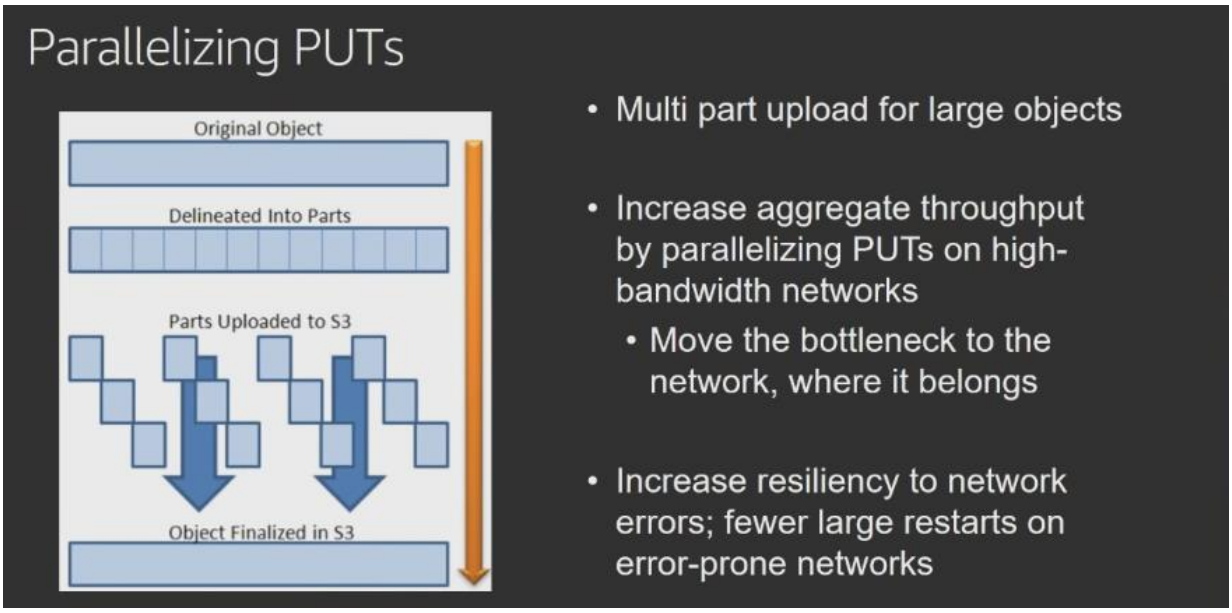
**A bit more LIST friendly:**

examplebucket/*animations/232a*-2017-26-05-15-00-00/cust1234234/animation1.obj
examplebucket/*videos/ba65*-2017-26-05-15-00-00/cust8474937/video2.mpg
examplebucket/*photos/8761*-2017-26-05-15-00-00/cust1248473/photo3.jpg

Random hash should come before patterns such as dates and sequential IDs
Always first ensure that your application can accommodate




If you are repeatedly asking for the same objects, you might want to consider using edge caching using CDNs like CloudFront for low latency.



## Key Amazon S3 best practices – Performance

- ✓Faster upload over long distances with S3 Transfer Acceleration
- ✓Faster upload for large objects with S3 multipart upload
- ✓Optimize GET performance with Range GET and Amazon CloudFront
- ✓SQL Query on S3 with Amazon Athena
- ✓Distribute key name for high TPS workload
- ✓TCP Window Scaling for long, fat networks
- ✓TCP SACK for fast, lossy connections like mobile

## How do you successfully modernize your important customer's highly visible, high throughput customer-facing application?

And live to talk about it?

# Loan Processing at Scale

## Amazon S3

Store any file, any size, any volume

For any amount of time

With stellar durability

Fast, reliable access to data

And predictable scalability

All for a cost that does not break the bank?

## "Usual" List

Static web site hosting

File sync and storage solutions

Image, video, & multimedia sharing

Data store for big data workloads

Data lake & AI storage

Social web and SaaS use cases

## "Unconventional" Thinking

High frequency requests

Blocking transactions

Quick response times

Atomicity and consistency

Guaranteed delivery

Amazon S3 as blazing fast storage system for high throughput transactional apps?

### Age-old solutioning

Traditional solution

- Build out DC
- Upfront high CapEx
- High Complexity
- Peg to **peak** capacity

Costly, unmaintainable, and inflexible. Upgrade treadmill

### Cloud 1.0 strategy

Cloud 1.0 solution

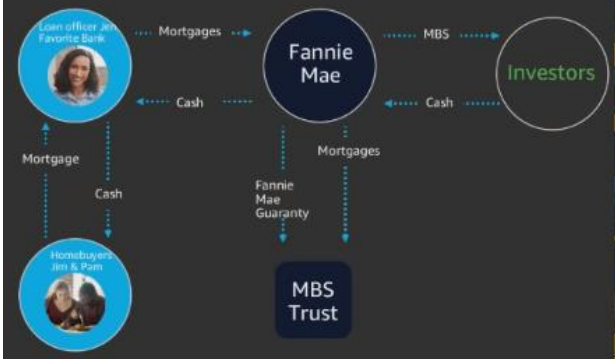
- Lift and shift
- Custom cloud storage
- Poor resiliency
- Peg to **peak** storage capacity

Slightly better, cloud-enabled solution. Checks off "Migrate to the cloud"

Cloud 2.0 Thinking : Get a little innovative, let's see how Fannie Mae did it!

# Fannie Mae's Use Case

## Fannie Mae is at the Heart of the Housing Industry




AWS re:Invent

## Fannie Mae, America's Most Valued Housing Partner

**1 in 3**
Homes in the country are financed by Fannie Mae

**$570 B**
Mortgage Financing in 2017

| 1.2M | 1M | 770K |
|---|---|---|
| New Homes | Refinancing | Multifamily Units |

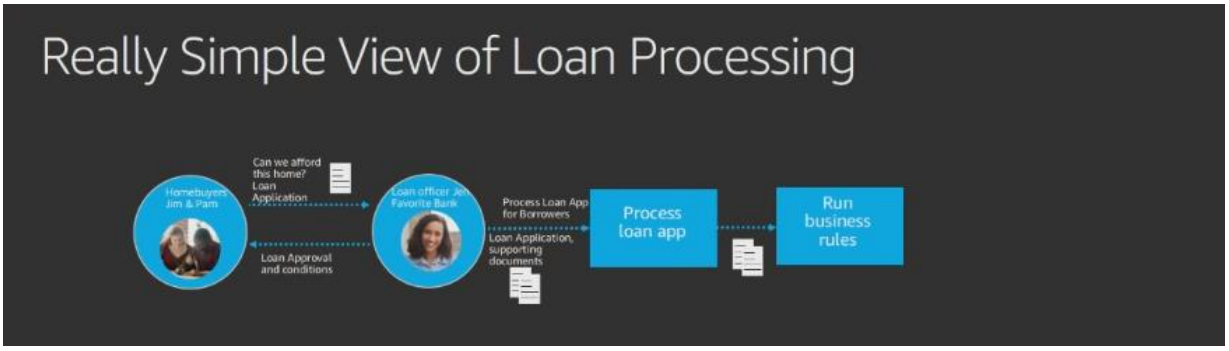
**76%**
Of Americans seeking a first lien mortgage chose 30-Year fixed rate mortgage in 2017

AWS re:Invent

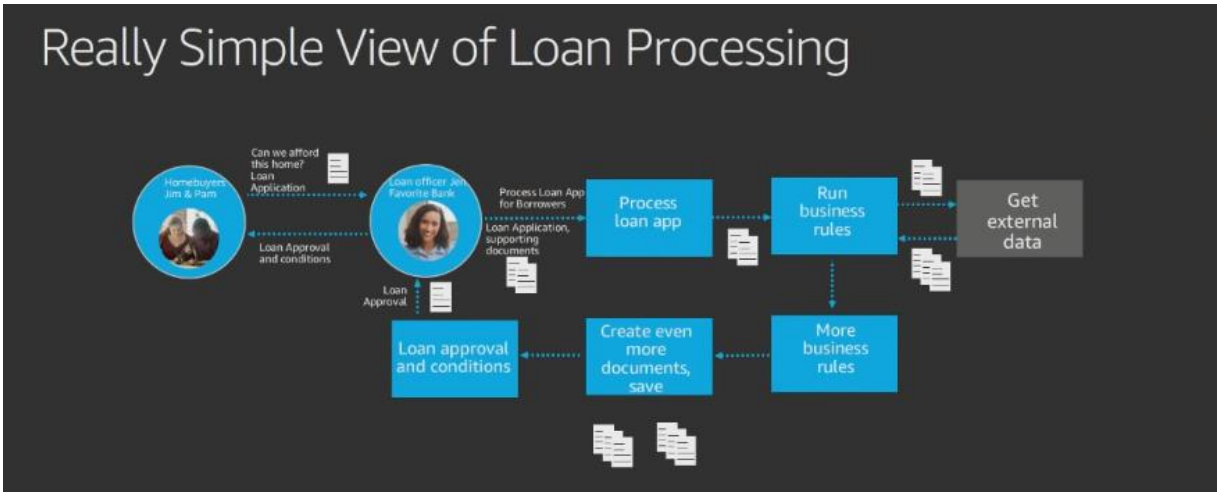


aws

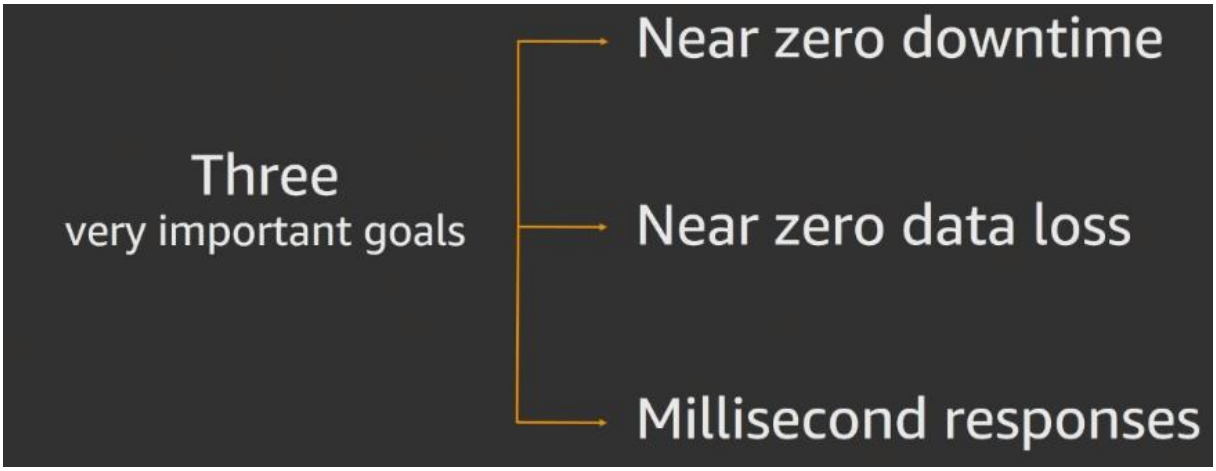

A prospective loan applicant works with a loan officer, submits a loan application along with a set of loan documents.




The loan application and the documents are then processed by Fannie Mae systems using a set of business rules to understand the applicant




More business rules are run in addition to collected external data, then a result is given about the loan application.



# Key Architecture Challenges

Maximizing performance, availability and durability all at the same time

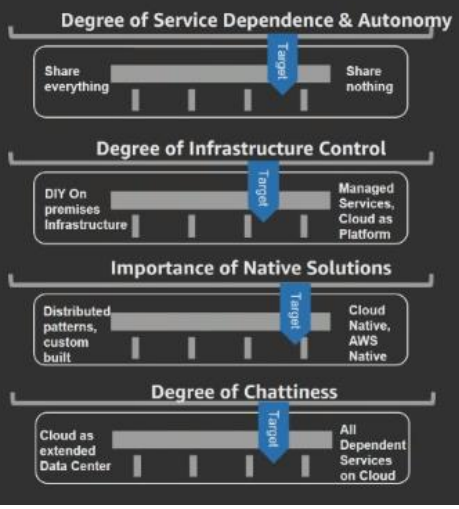
Incredibly low latency

Billions of objects/year

Work with current platform

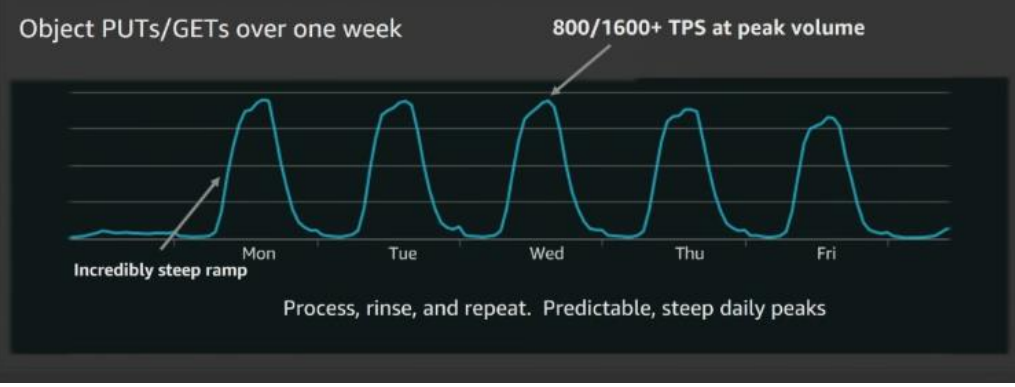
Reduce Operating Expenses(OpEx)

# Create Guardrails Early

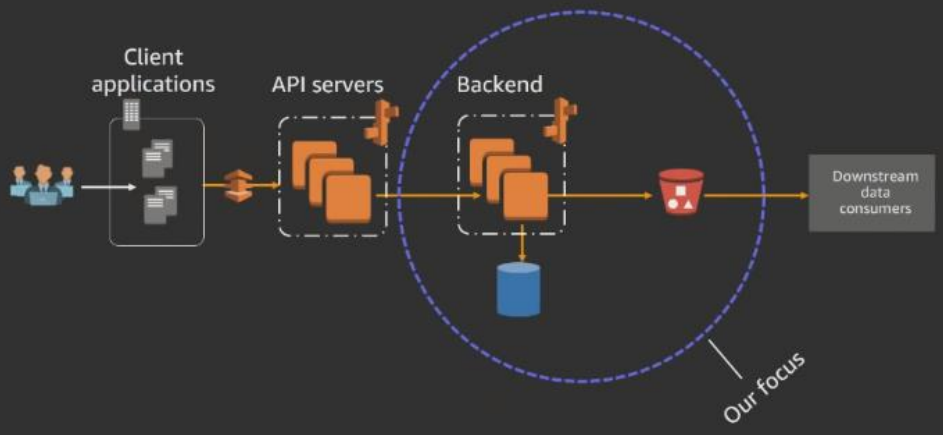



# The Solution

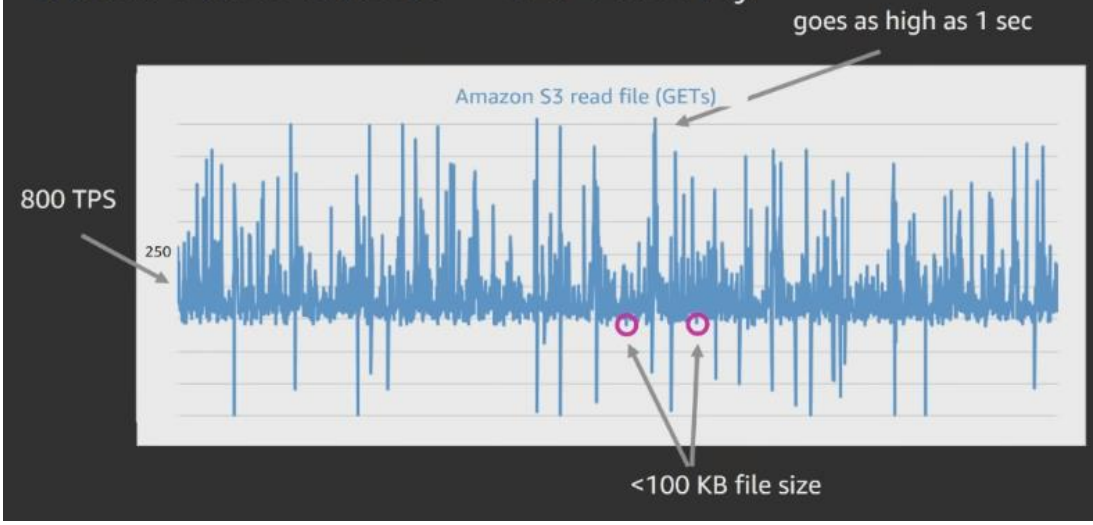
# Just Another Week



# Simplified Architecture View




# Initial Observations – GET Latency




Key architecture challenges

Best practices, key first steps

Second phase, shaving off milliseconds

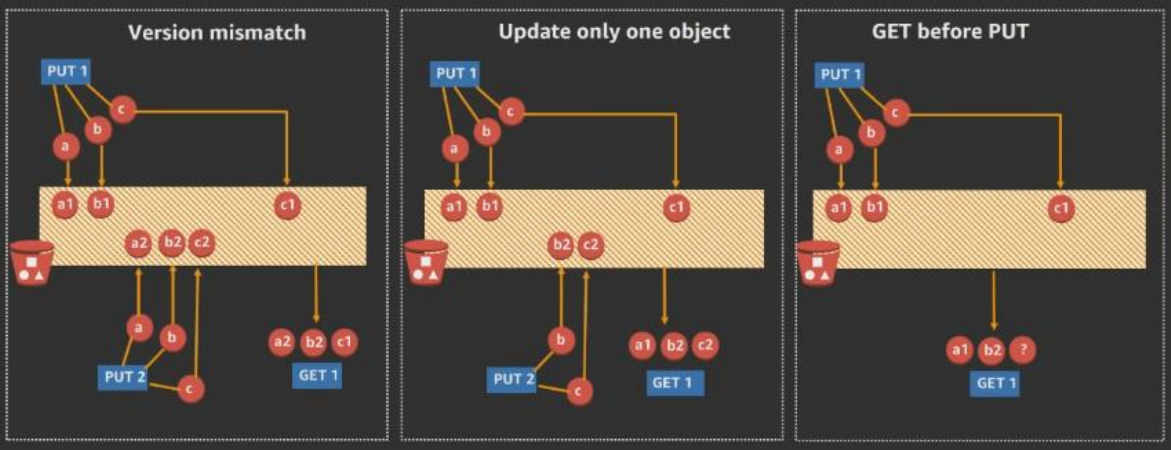
## Standard Best Practices

- Entropy (randomness)
- Parallelizing PUTs & GETs
  - For troubleshooting use Request ID, S3 Extended Request ID & Host ID
- Externalizing markers
- RDS as an indexing mechanism for markers

Pay attention to client execution timeout

```
try {
    ClientConfiguration configuration = new ClientConfiguration();
    configuration.setMaxConnections(….);
    configuration.setRetryPolicy(…);
    configuration.setClientExecutionTimeout(….);
    AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();
    S3Object object = s3Client.getObject(new GetObjectRequest(….);
    } catch (ClientExecutionTimeoutException ce)
            log.error(….);
            throw ce;
    }
// the caller of this method, may retry to get the object successfully
```

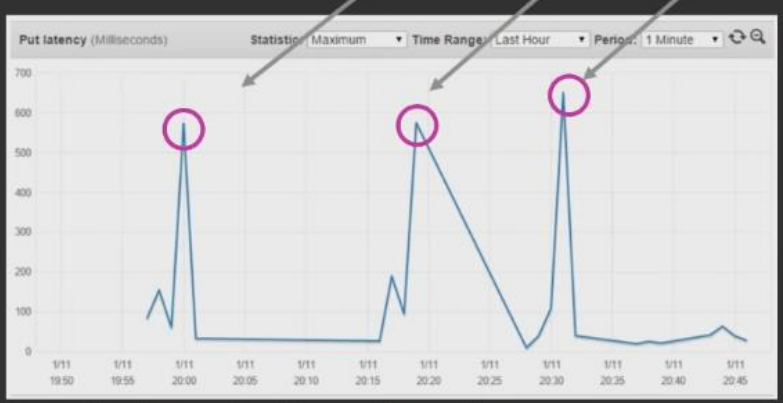
## Working Around Amazon S3 Eventual Consistency




Our application was strongly consistent but S3 is eventually consistent, so we had to augment with an RDS database with Postgres

## PUT Latency

Consistently reproducible response time spikes every 15 min



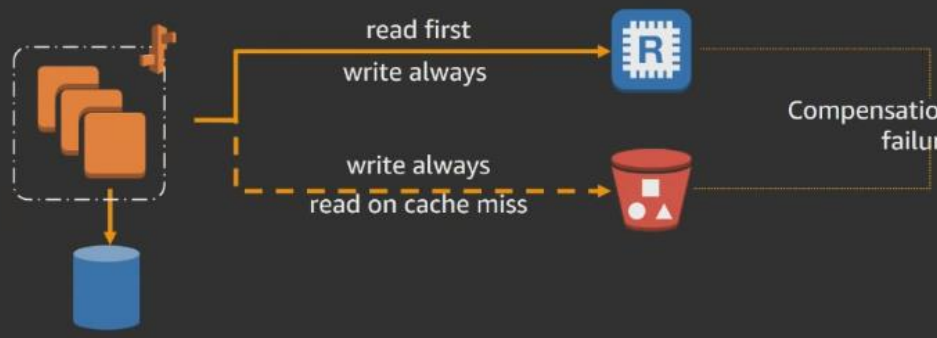

Unpredictable, could add several seconds to customer response times

Best practices, key first steps

Second phase, shaving off milliseconds

Third phase, pushing the limits, literally. Improving resiliency

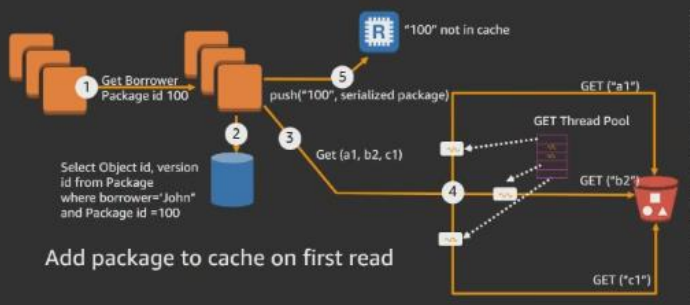
## Caching In




Uncoupling Amazon S3 stores to improve PUT latency

Needs compensation to ensure continued high durability

## Cached Reads



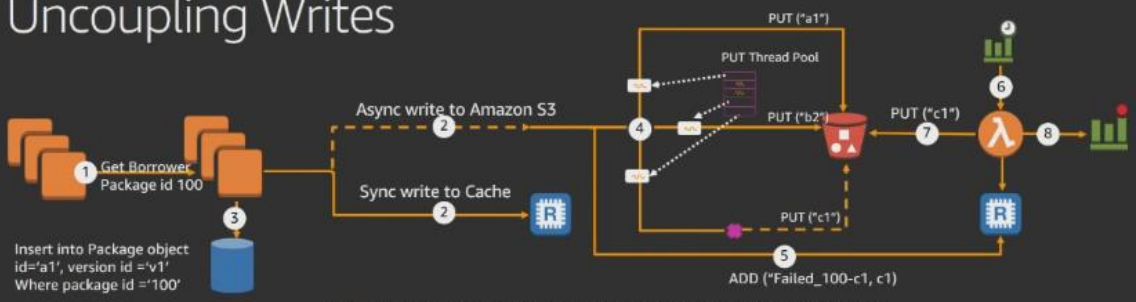

Add package to cache on first read


Read from cache for subsequent requests

Saves 100-150ms per transaction

Over 200,000 seconds of processing time/latency saved every day

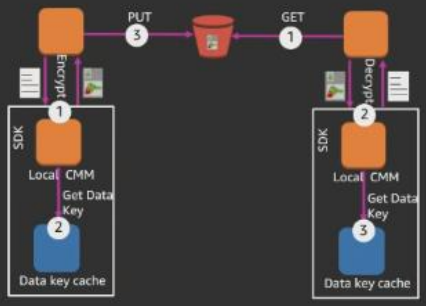
# Uncoupling Writes




Uncoupling transactions with compensation for failures

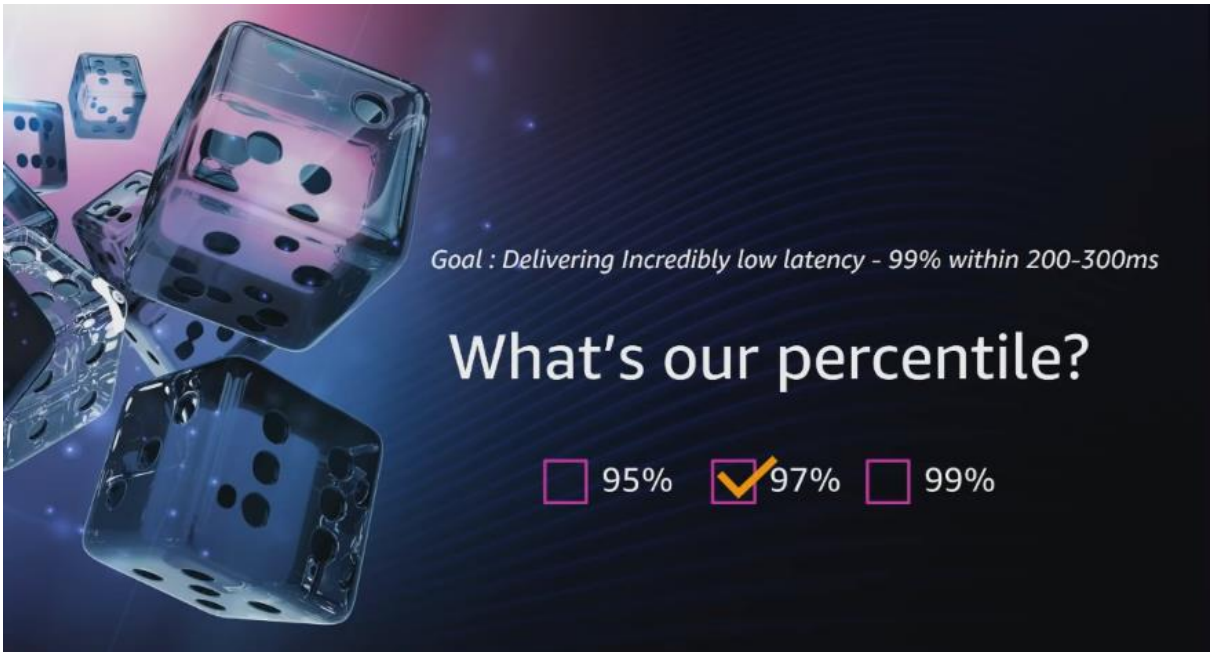
Cache writes are blazing fast, saves 250-300ms per transaction
Over 400,000 seconds of processing time/latency saved every day

# Encrypting Data at Rest

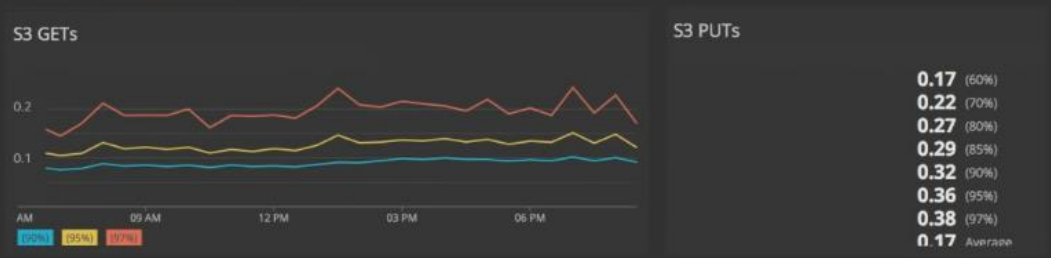



- ✓ SSE-KMS provides many operational advantages, so you might want to stick to it
- ✓ AWS KMS observed availability is pretty high, 99.999%
- ✓ For high variable workloads from 1000s to millions, SSE-S3 might be a better fit

Switching to CSE can save 15-25ms per transaction
Over 100,000 seconds of processing time/latency saved every day



# Optimized Response Times



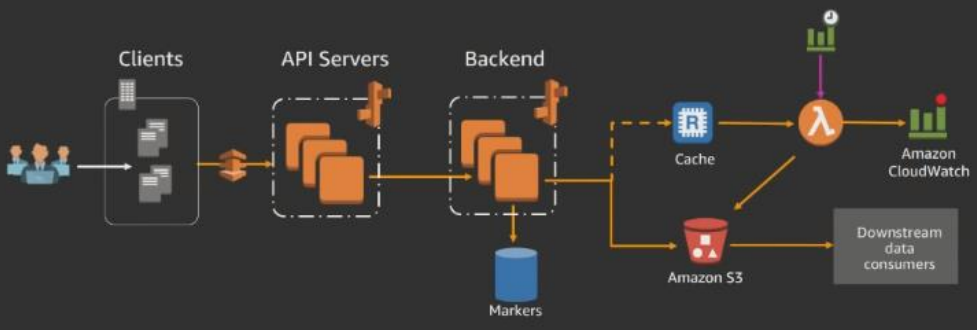

- Improve Store response time by over 100% with uncoupled cache
- < 300ms GET latency for 97% of transactions
- < 300ms PUT latency for 97% of transactions *(not quite, but not relevant due to uncoupling)*

Second phase, shaving off milliseconds

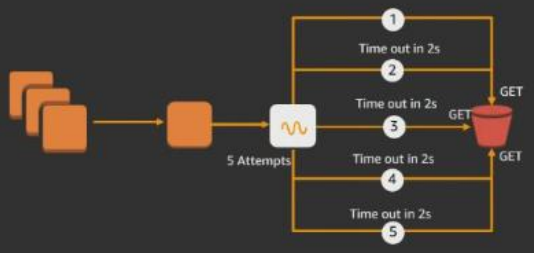
Third phase, pushing the limits, literally. Improving resiliency

More key challenges

# Slightly Less Simple View of the Architecture



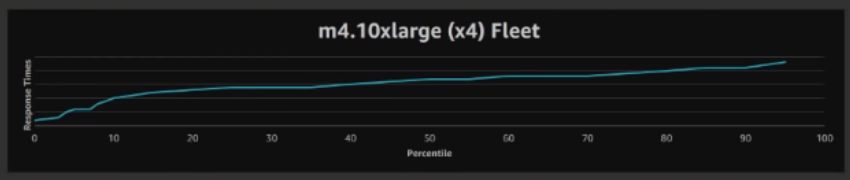
## Retries for Occasional GET Latency Spikes

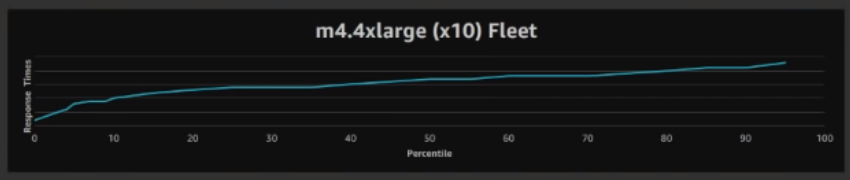



Distributed nature of Amazon S3 = some GETs take over several seconds

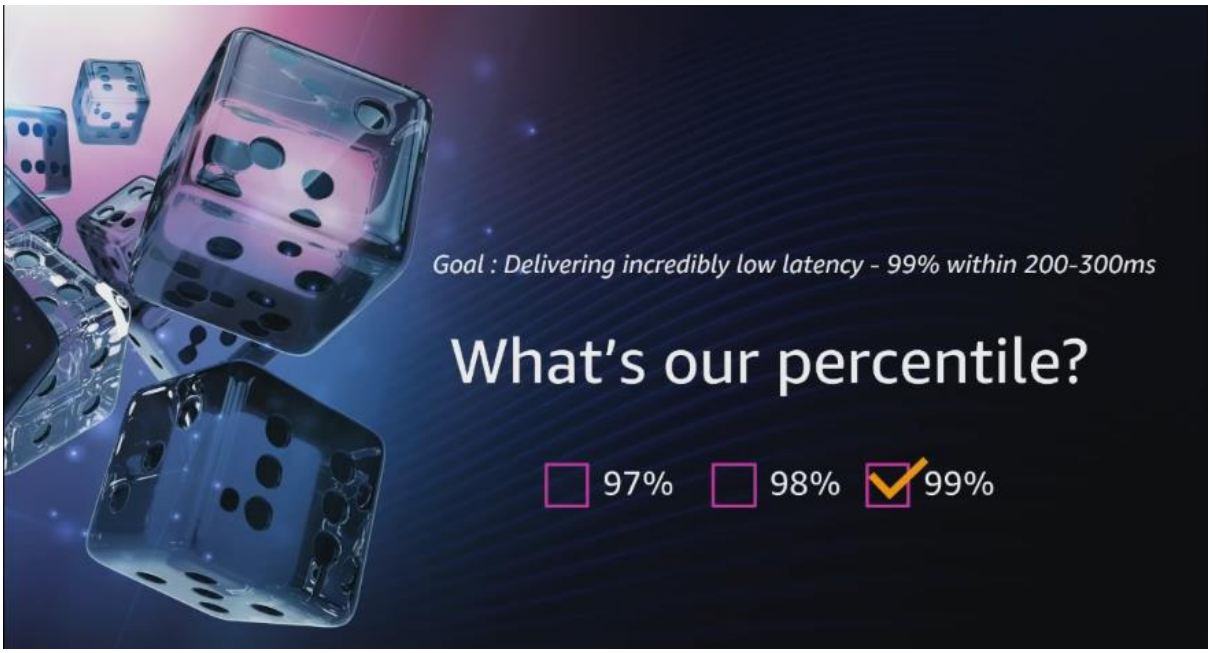
Retry resolves time outs, may take 2-3 attempts

## Rightsizing Instances




Smaller instances have similar, may be even better performance



# Results


99% of stores and reads within 275 milliseconds

# More Key Challenges

High(er) availability

Region failure immunity

Minute scale recovery point objective (RPO) & recovery time objective (RTO)

# Achieving Resiliency

Assume everything in the app could fail

Dependencies could fail

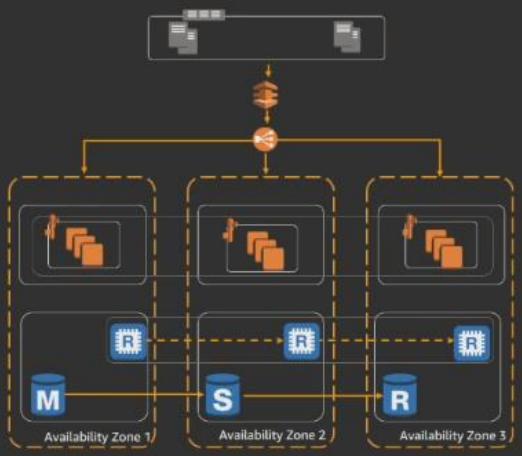
Network & connectivity could fail

Availability Zones and Regions could fail

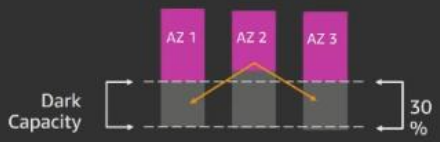
Assume everything will fail all the time

# App & Dependent Service Resiliency, Briefly

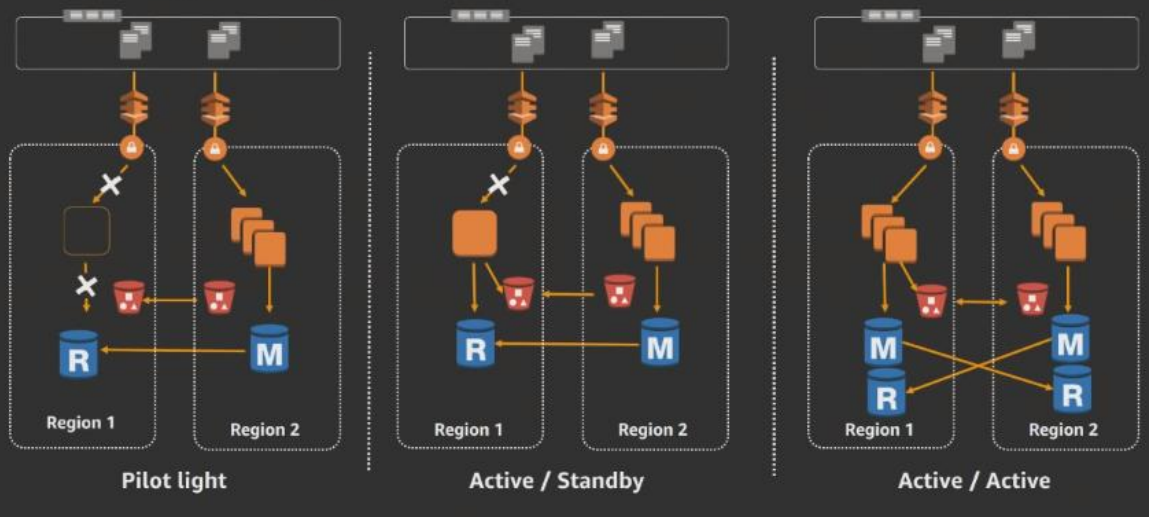

# Resiliency in a Region








# Multi-region Options

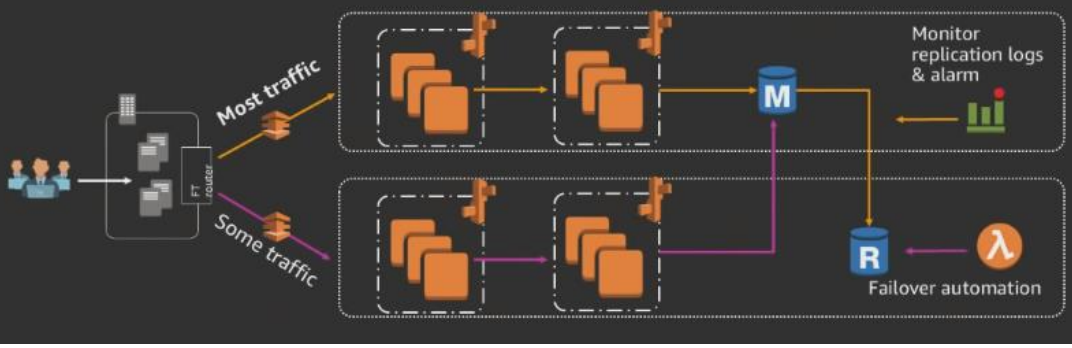


# RPO vs. Cost & Complexity

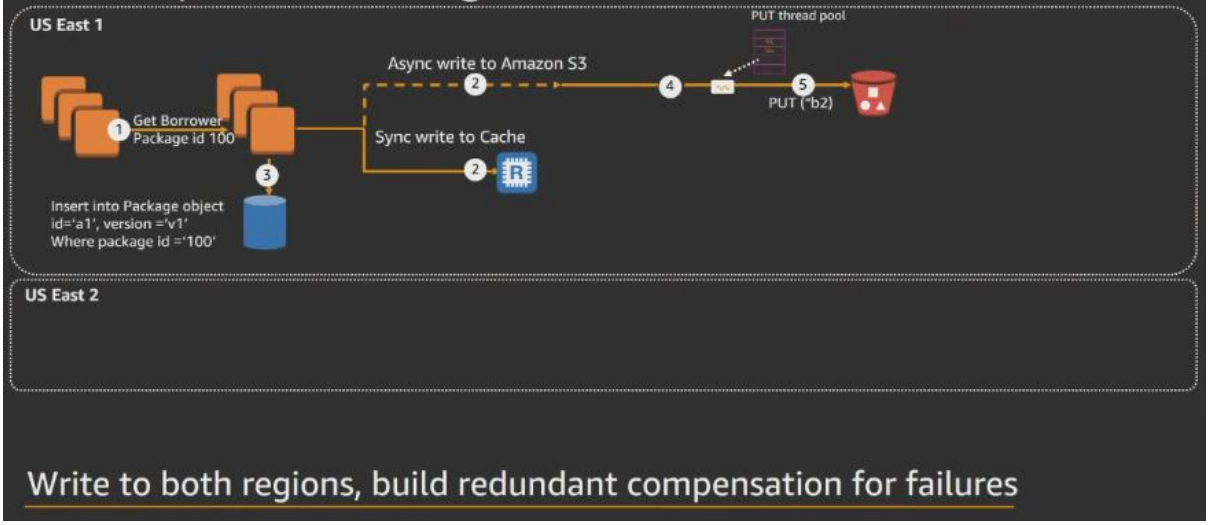

# Typical Multi-region Options

| | All native | Custom DB | Custom Amazon S3 | Custom DB & Custom Amazon S3 |
|---|---|---|---|---|
| Compute | Native Active / Active | Native Active / Active | Native Active / Active | Native Active / Active |
| Data - RDS | Async Native Active / Standby | Async Custom Active / Standby | Async Native Active / Standby | Sync Custom Active / Active |
| Objects – Amazon S3 | Native Active / Standby | Native Active / Standby | Custom Active / Active | Sync Custom Active / Active |
| RPO | High | Medium | Low | Very Low |
| Cost | VL | H | M | VH |
| Complexity | L | M | L | H |
| Response Time Impact | None | Network & lag dependent | None | Slowest due to sync n/w & lag |

# Our Simple High Availability Plan

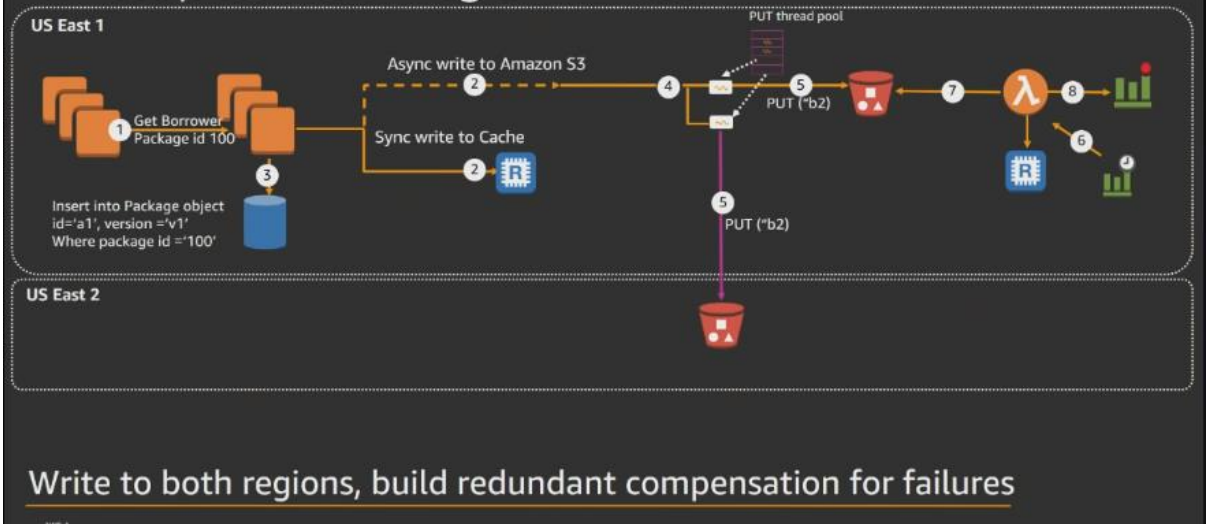


# Uncoupled Multi-Region Writes



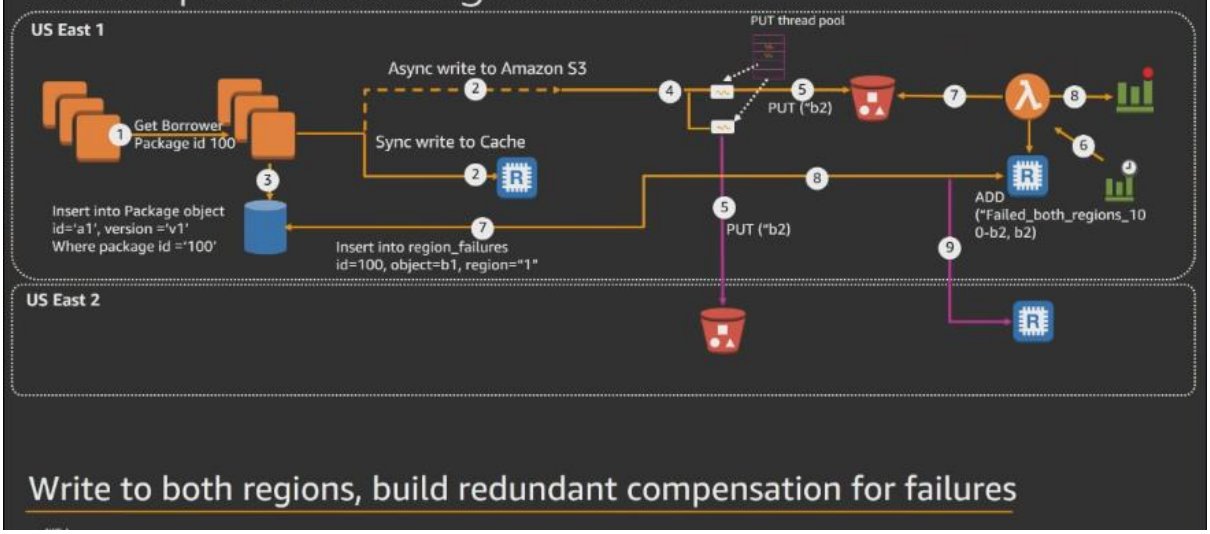

Write to both regions, build redundant compensation for failures

# Uncoupled Multi-Region Writes



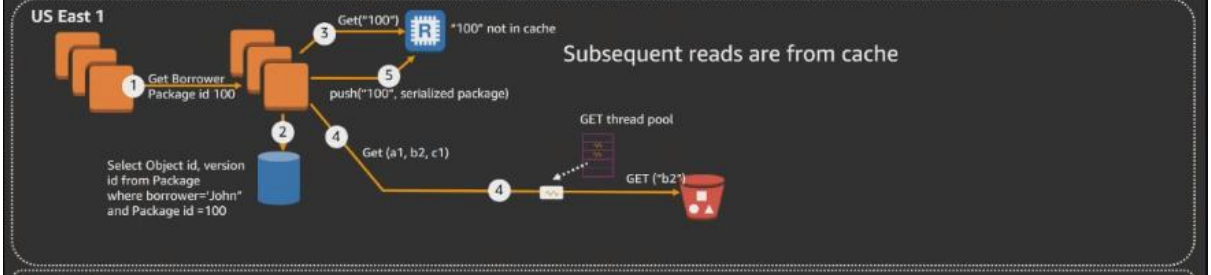

Write to both regions, build redundant compensation for failures

# Uncoupled Multi-Region Writes



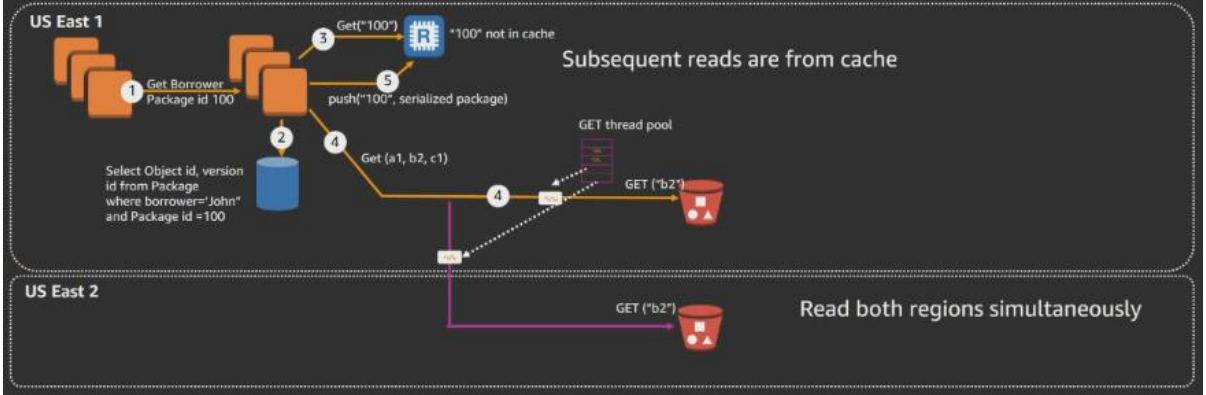

Write to both regions, build redundant compensation for failures

# Multi-region Reads




Eliminates sporadic S3 latency spikes, region failure immunity

# Multi-region Reads




Eliminates sporadic S3 latency spikes, region failure immunity

# Key Takeaways

- 1000's of TPS is possible
- Parallelize GETs & PUTS, add retries
- Last 1% will take significant effort
- SSE KMS good for almost every use case, < 20ms overhead
- Remember: Bigger instances are not always better !
- Objects and metadata can't stand each other
- Cache in for predictable low latency
- 4-9s+, write and read from both regions

Be a little unconventional with Amazon S3, serious benefits are in store

99.9% of transactions < 300 milliseconds

Better availability

Better TCO

# Thank you!

Oliver Mathias
Oliver_mathias@fanniemae.com

Harsha Nippani
nippanih@amazon.com

aws re:Invent



aws