AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

SVS310

Architecting your serverless applications for hyperscale

Justin Pirtle (he/him)
Principal Solutions Architect
Amazon Web Services

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

What we will cover

- Strategies for achieving hyperscale with serverless services, including Amazon API Gateway, AWS Lambda, Amazon CloudFront, and Amazon DynamoDB
- Applied techniques with sample application
- Best practices and tools for use with your next serverless workloads

What we won’t cover

- What is serverless?
- What are Amazon API Gateway, Amazon DynamoDB, AWS Lambda?

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

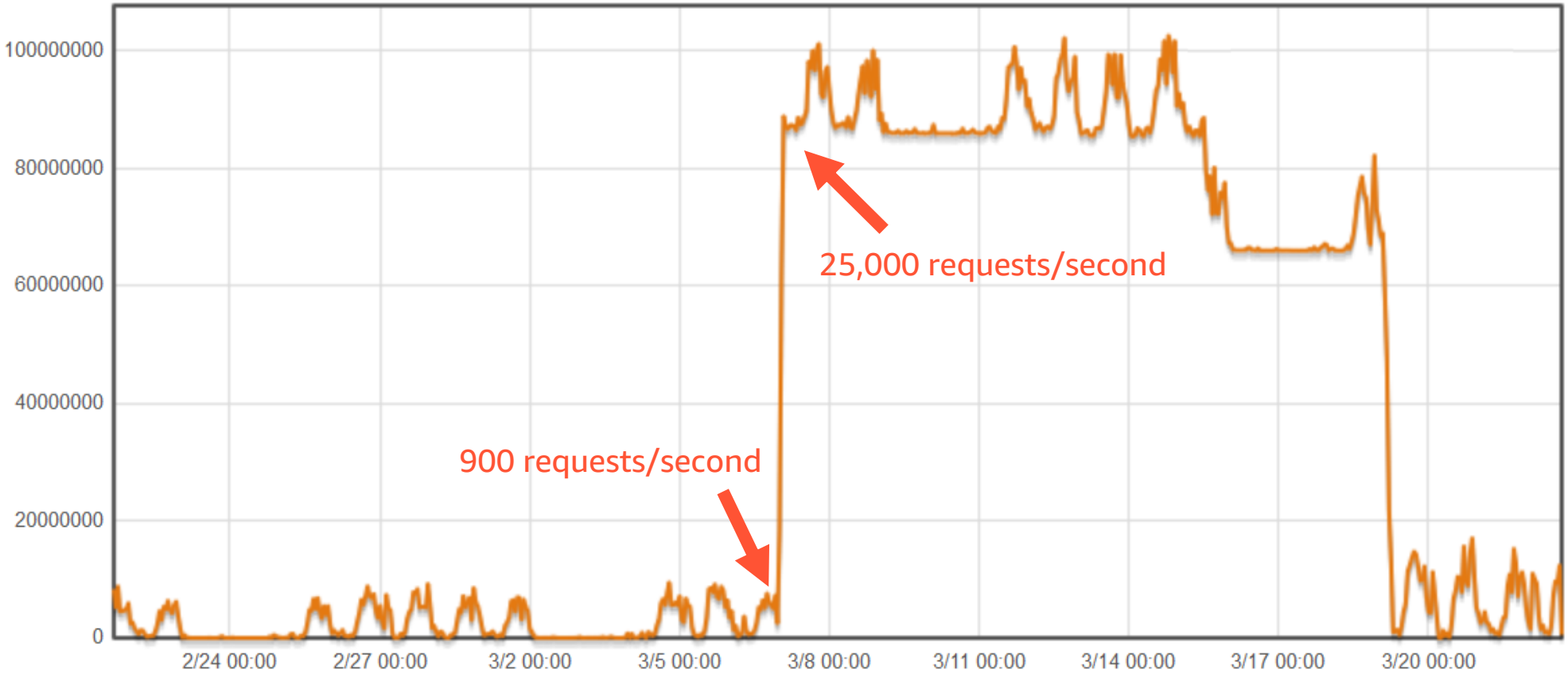
What is hyperscale?

"The ability of an architecture to scale seamlessly as increased demand is added to the system"

- 10s/100s of thousands of requests per second
- Potentially millions of users
- Steep rate of change in traffic bursts

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Burst workload

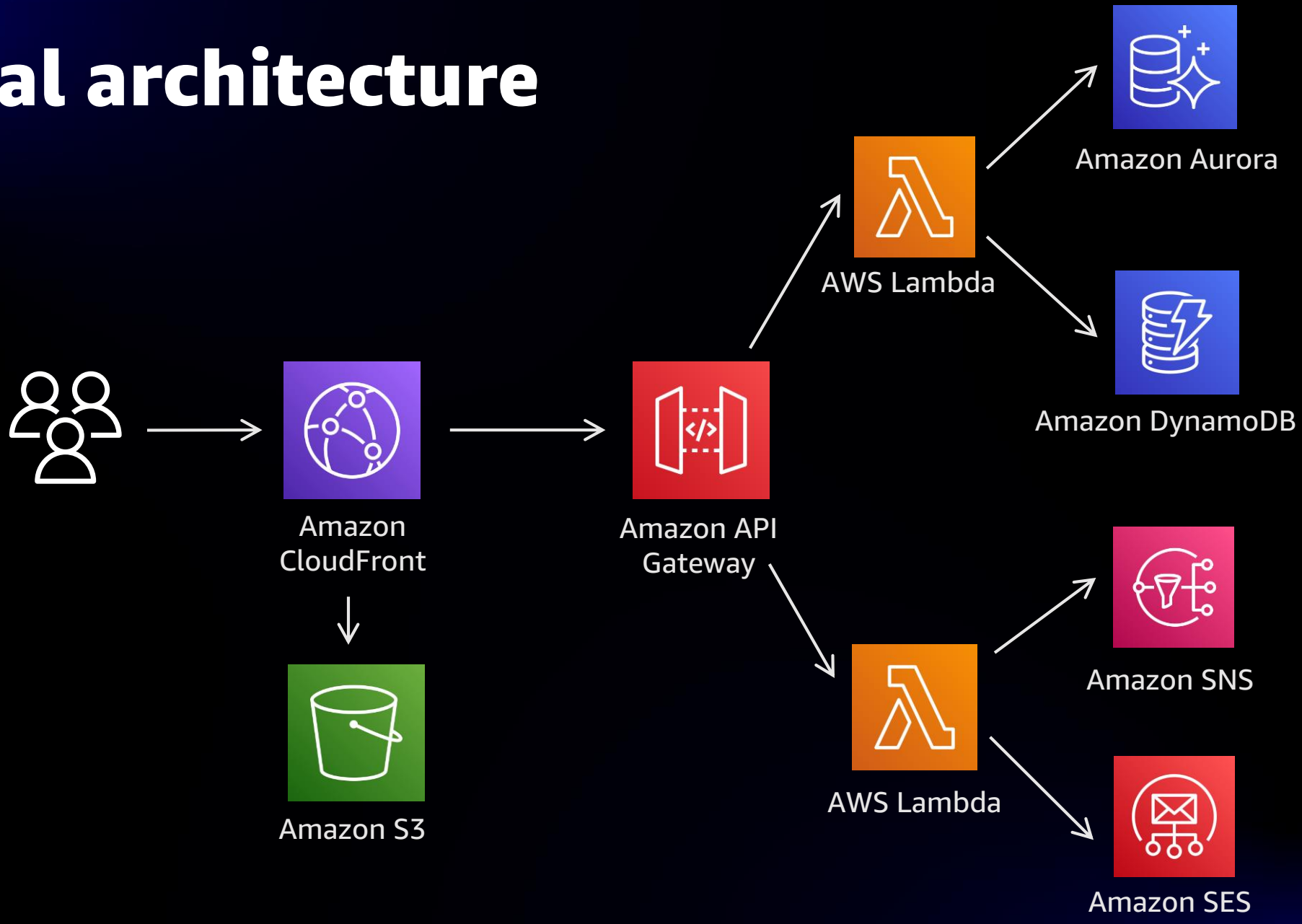


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Today’s application

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture

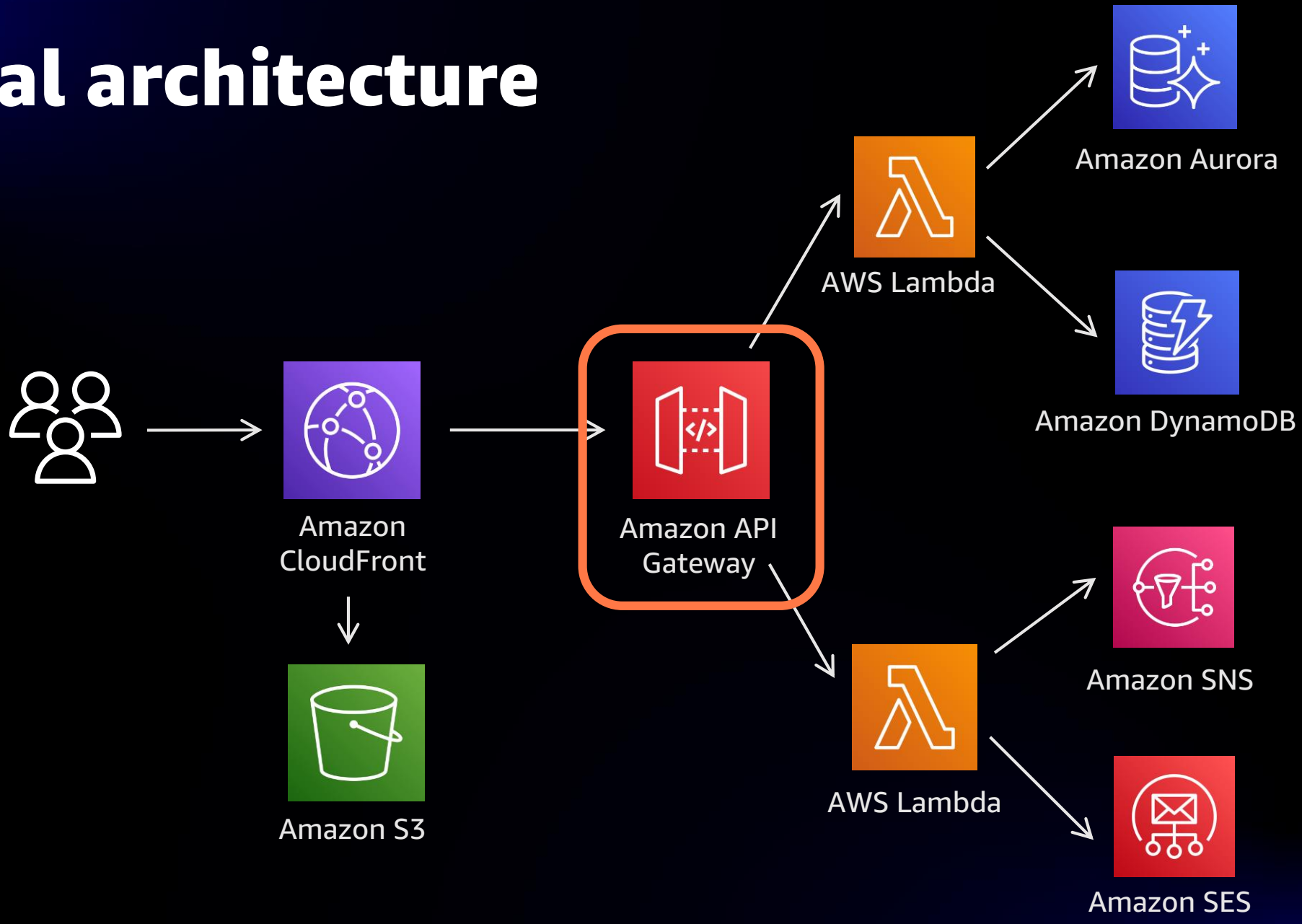


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #1: Understand AWS serverless compute scalability

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



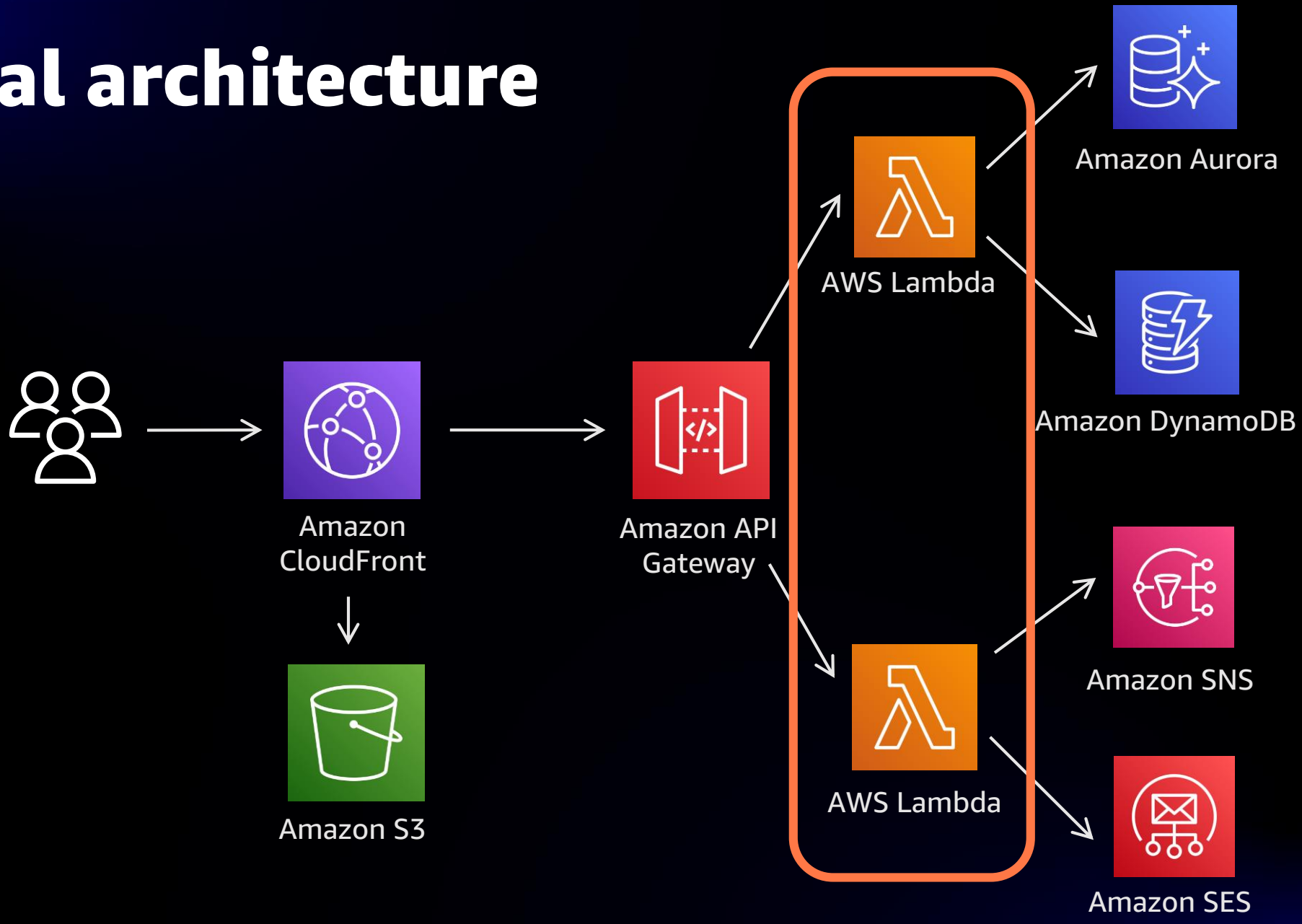
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

API Gateway service quotas

- **Always-available, multi-tenant architecture:** Burst from 0 to 100s of thousands of requests per second (RPS) with no advanced notice and no burst quota
  - Load testing required in advance to grant large quota increases
- **Default throughput:** 10,000 RPS
  - Can be raised to handle the most demanding workloads
- **Max integration timeout:** 29 (REST API) or 30 (HTTP API) seconds
- **Payload size:** 10 MB
- And much more: https://go.aws/3qi1F7Y

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency

A single Lambda execution environment can only process a single event at a time

- Regardless of event source or invoke type
- Batches pulled from Amazon Kinesis Data Streams, Amazon SQS, or Amazon DynamoDB Streams count as a single event

Concurrent requests require new execution environments to be created

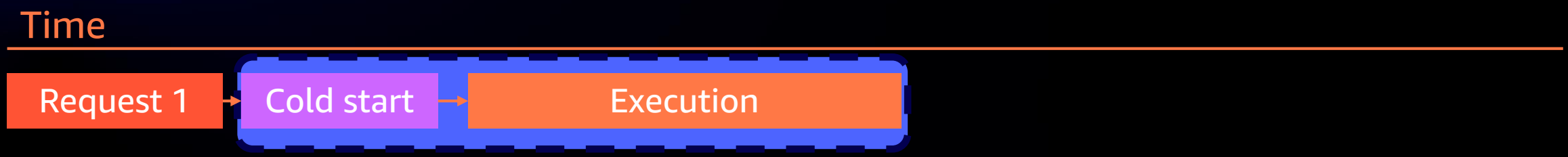
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



This execution environment is blocked for this entire time

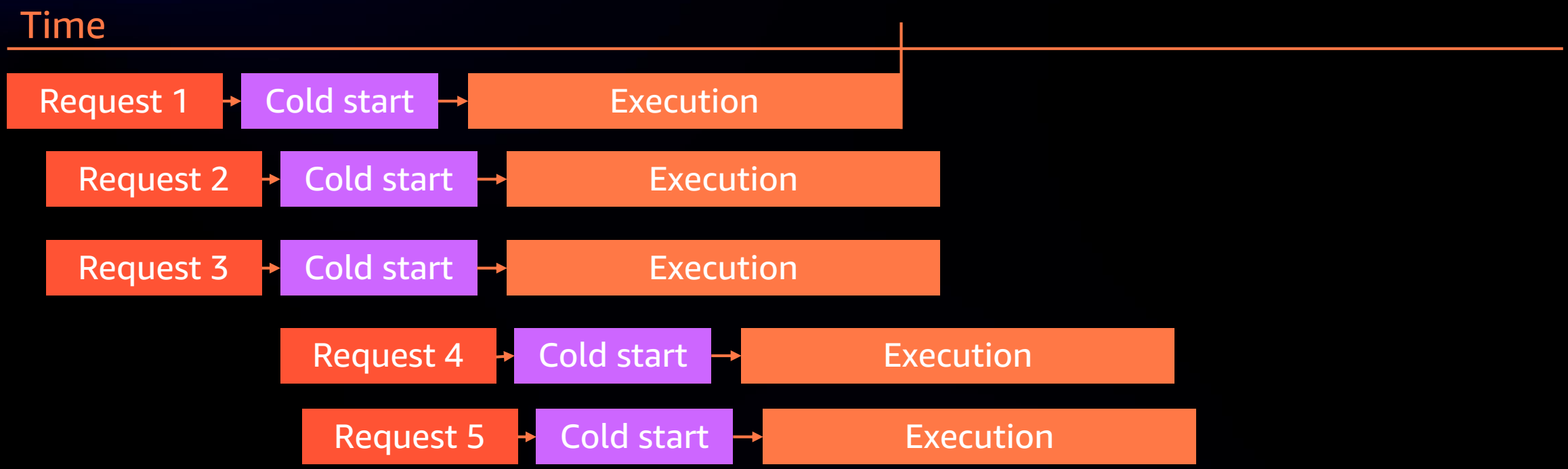
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

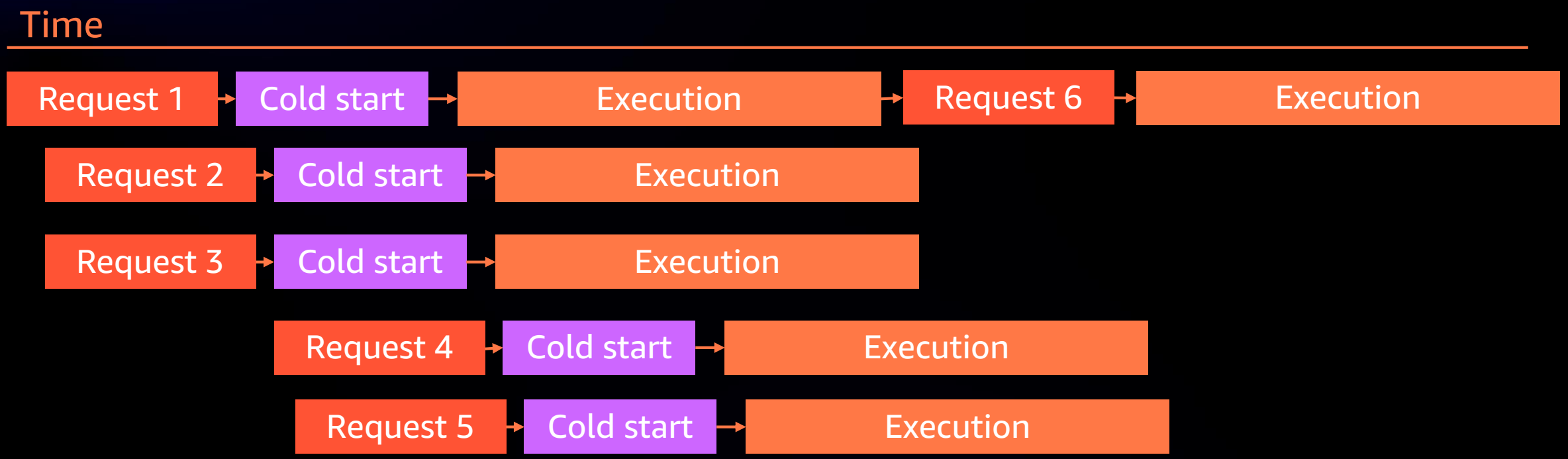
Lambda concurrency



The first environment is now free and warm

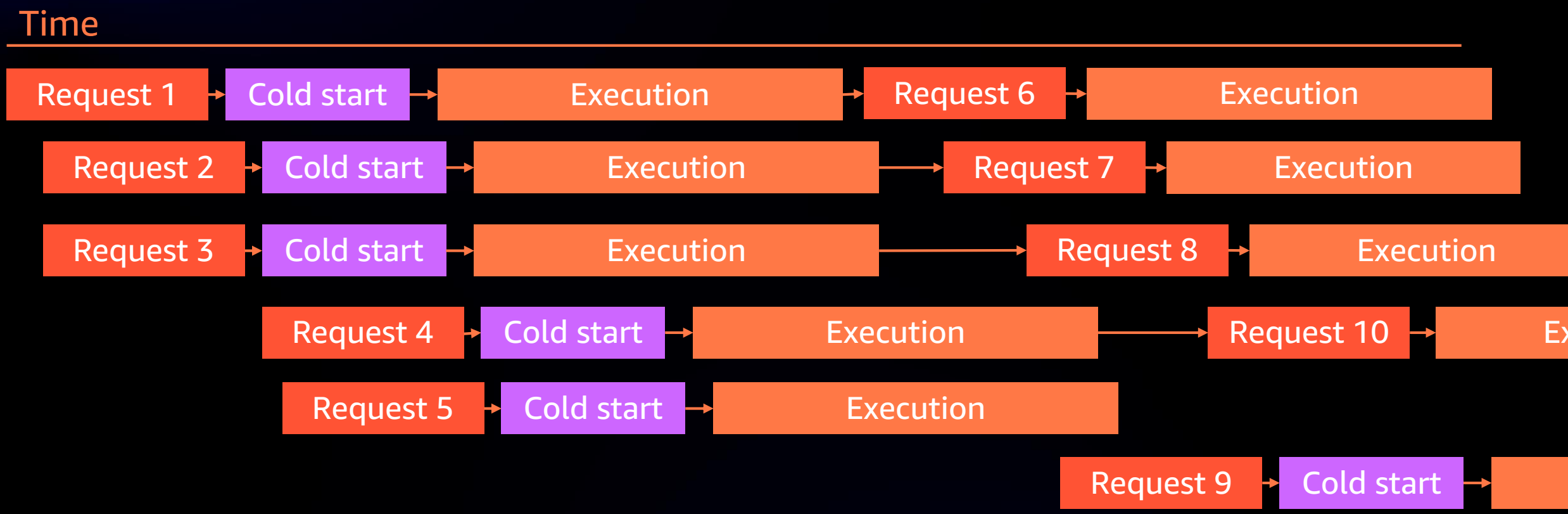
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



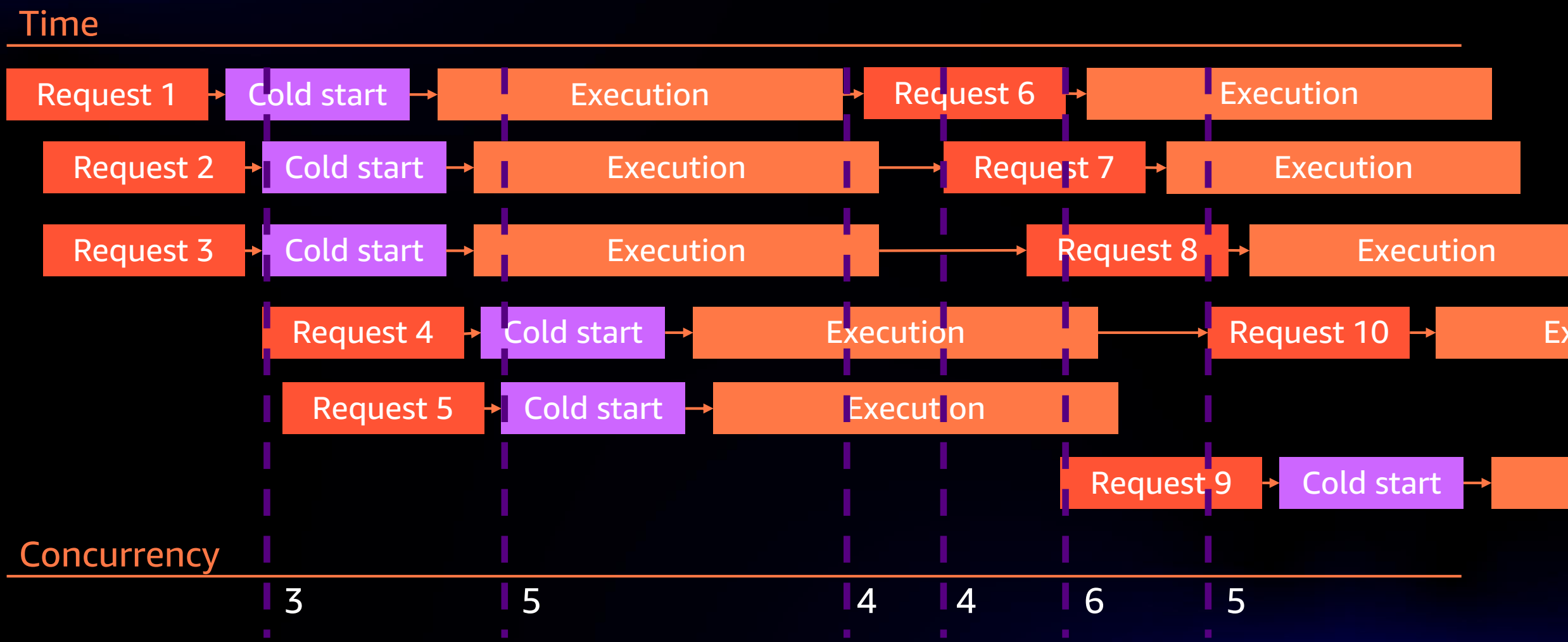
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda scaling quotas

Account concurrency

Maximum concurrency in a given AWS Region across all functions

1,000 is default for each Region

This can be increased

Burst concurrency

Maximum increase in concurrency for an initial burst of traffic

3,000 in US East (N. Virginia), US West (Oregon), and Europe (Ireland)

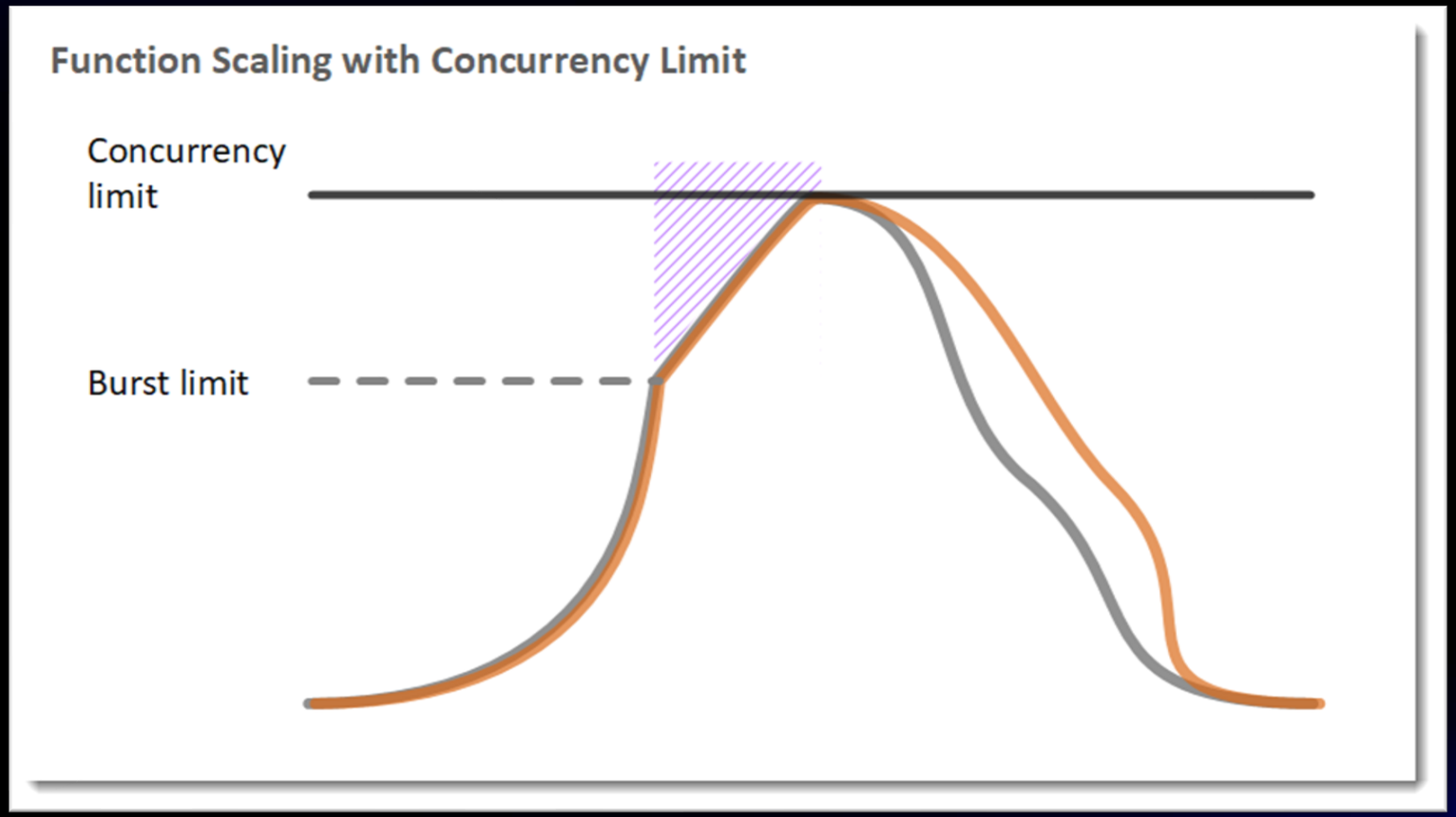
1,000 in US East (Ohio), Asia Pacific (Tokyo), and Europe (Frankfurt)

500 in all other Regions

After that, functions can scale by 500 concurrency per minute

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda scaling quotas

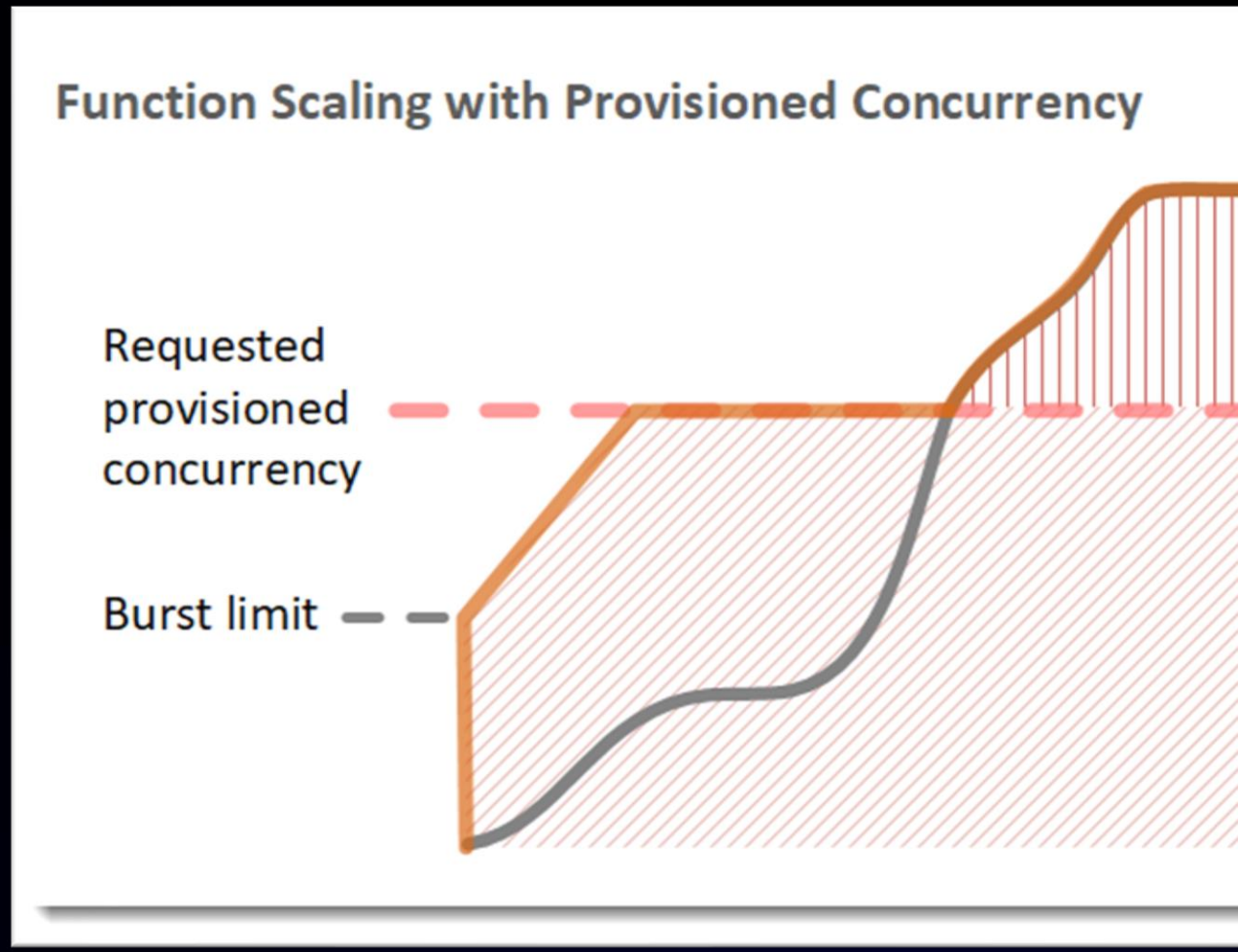


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency controls

Provisioned concurrency:

- Sets floor on minimum number of execution environments
- Pre-warm execution environments to reduce cold-start impact
- Burst to use standard concurrency if desired
- Can save costs in certain situations

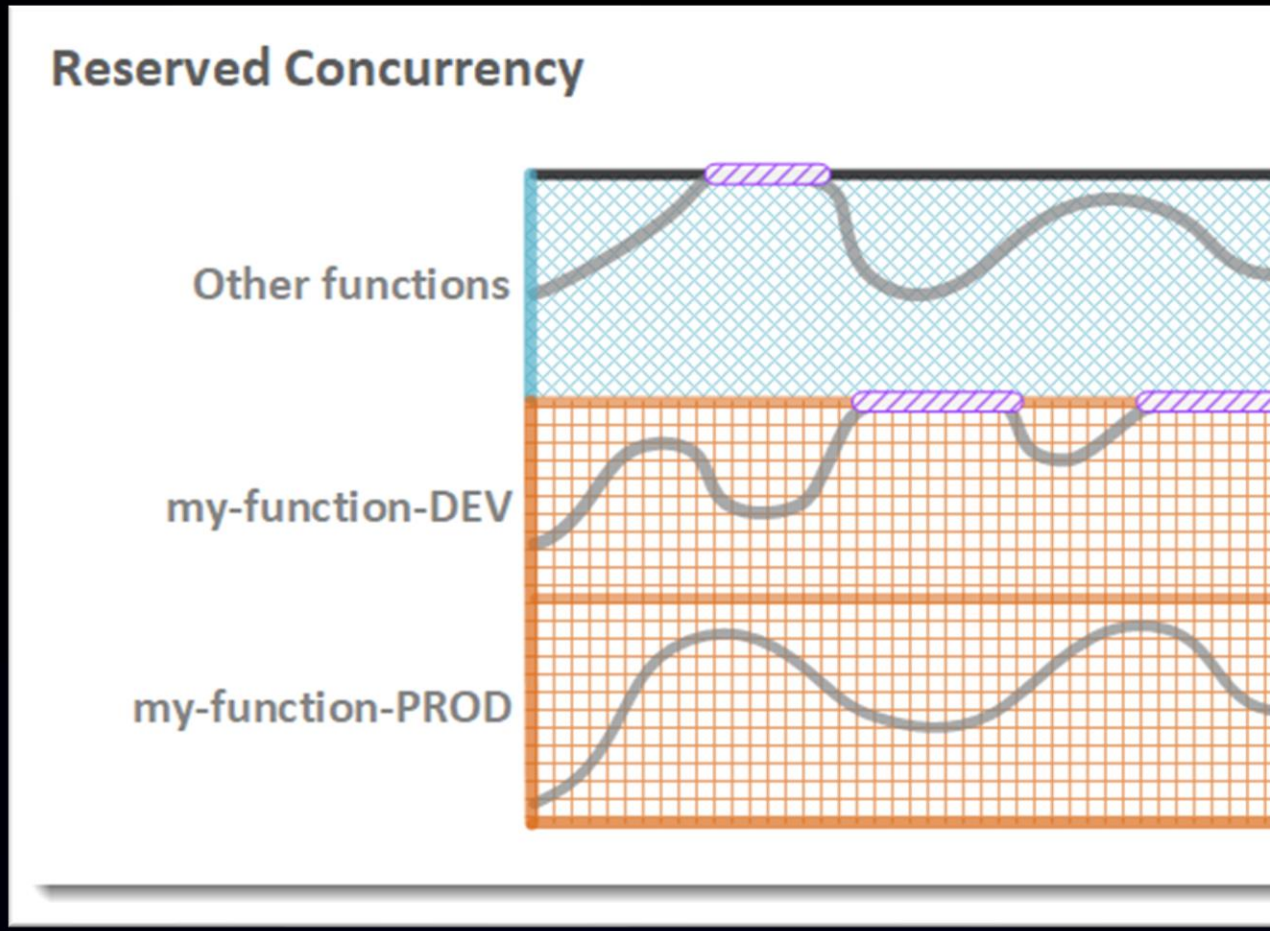


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency controls

Reserved concurrency

- Sets ceiling on maximum number of execution environments – maximum concurrency for a given function
- Also reserves that concurrency from the account's quota



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #2: Optimize Lambda function performance

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda concurrency

Lambda concurrency = requests per second (RPS) x duration in seconds

Examples

- 10,000 RPS x 1,000 ms = 10,000 concurrency
- 10,000 RPS x 500 ms = 5,000 concurrency
- 10,000 RPS x 100 ms = 1,000 concurrency

Long-running functions require more concurrency!

Longer durations greatly increase concurrency needs. . .

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

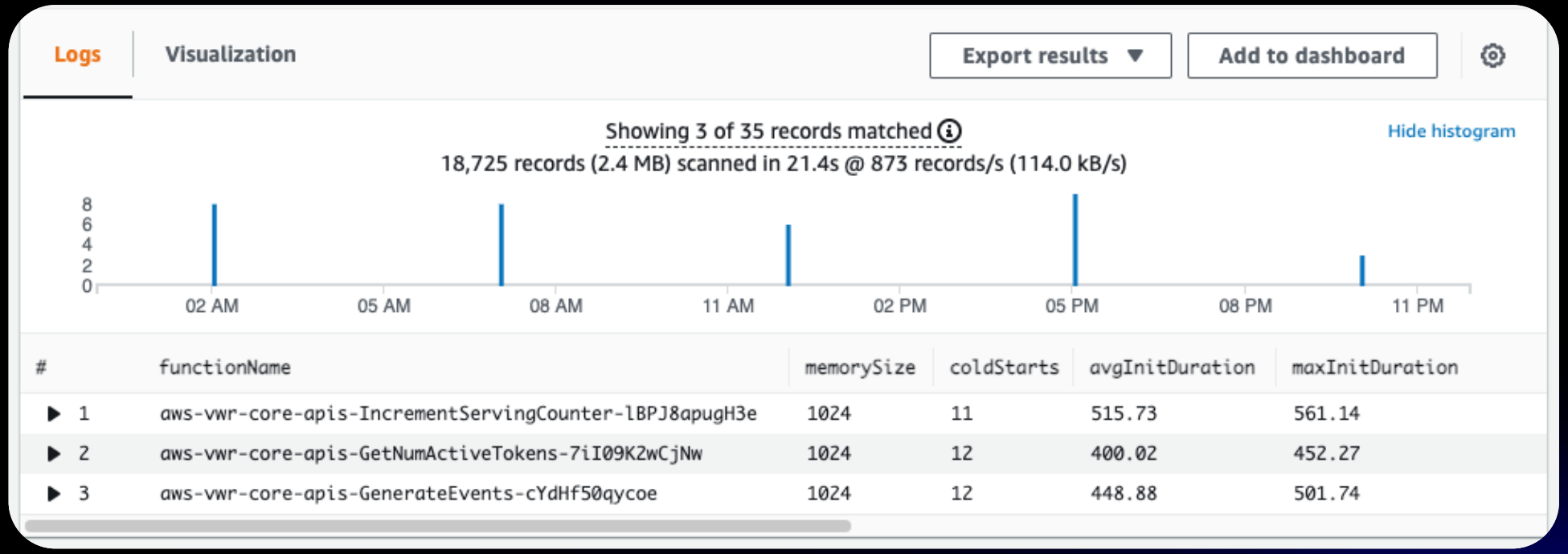
The function lifecycle



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CloudWatch Logs Insights: Profiling cold starts

```
filter @type="REPORT"
| fields @memorySize / 1000000 as memorySize
| filter @message like /(?i)(Init Duration)/
| parse @message /^REPORT.*Init Duration: (?<initDuration>.*) ms.*/
| parse @log /^.*\/aws\/lambda\/(?<functionName>.*)/
| stats count() as coldStarts, median(initDuration) as avgInitDuration, max(initDuration) as
maxInitDuration by functionName, memorySize
```



#	functionName	memorySize	coldStarts	avgInitDuration	maxInitDuration
▶ 1	aws-vwr-core-apis-IncrementServingCounter-lBPJ8apugH3e	1024	11	515.73	561.14
▶ 2	aws-vwr-core-apis-GetNumActiveTokens-7iI09K2wCjNw	1024	12	400.02	452.27
▶ 3	aws-vwr-core-apis-GenerateEvents-cYdHf50qycoe	1024	12	448.88	501.74

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Optimization best practices (cold start)

Avoid monolithic functions

Reduce deployment package size

Minify/uglify production code

Browserify/webpack

Optimize dependencies

Up to 120 ms faster with Node.js SDK

Lazy initialization of shared libs

Helps if there are multiple functions per file

Reuse connections

Keep alive in AWS SDKs

Use provisioned concurrency

On individual functions, no code changes

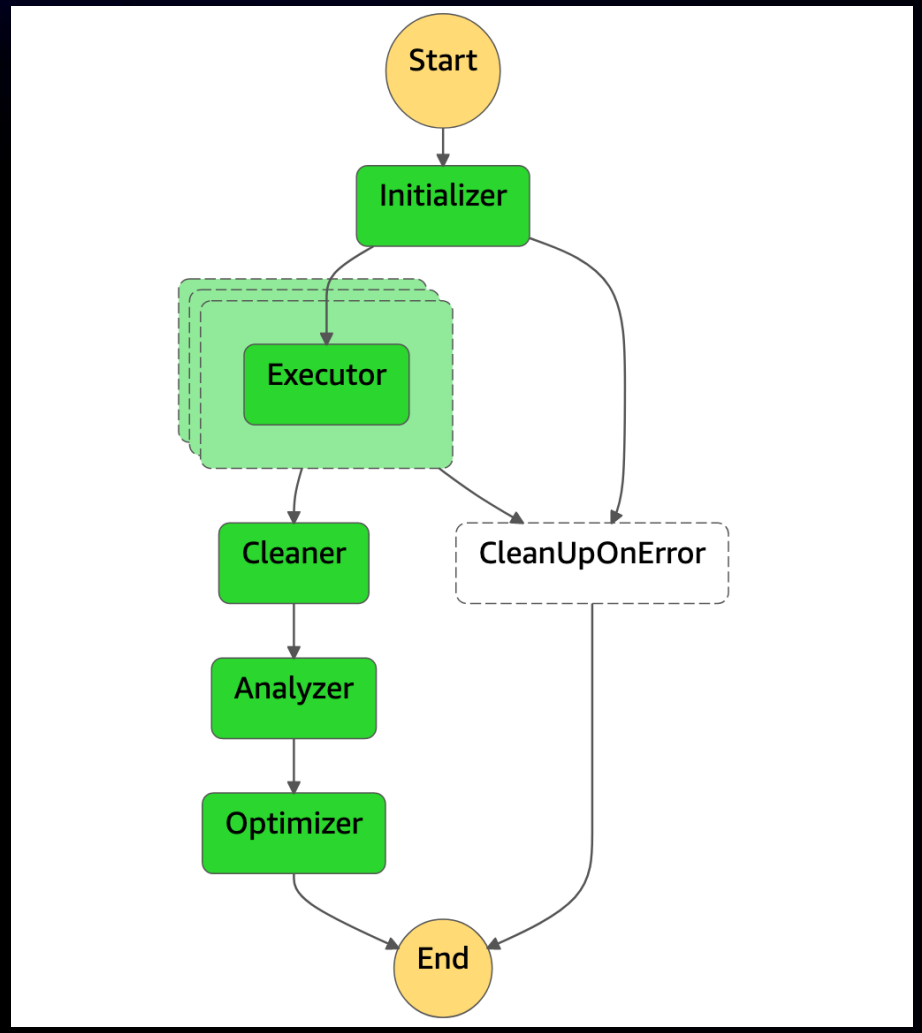
Dive deeper at https://s12d.com/Lambda-Optimizations-2021



aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Meet AWS Lambda Power Tuning



Data-driven cost and performance optimization for AWS Lambda

Available as an SAR app

Easy to integrate with CI/CD

github.com/alexcasalboni/aws-lambda-power-tuning

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Lambda Power Tuning: Input/output

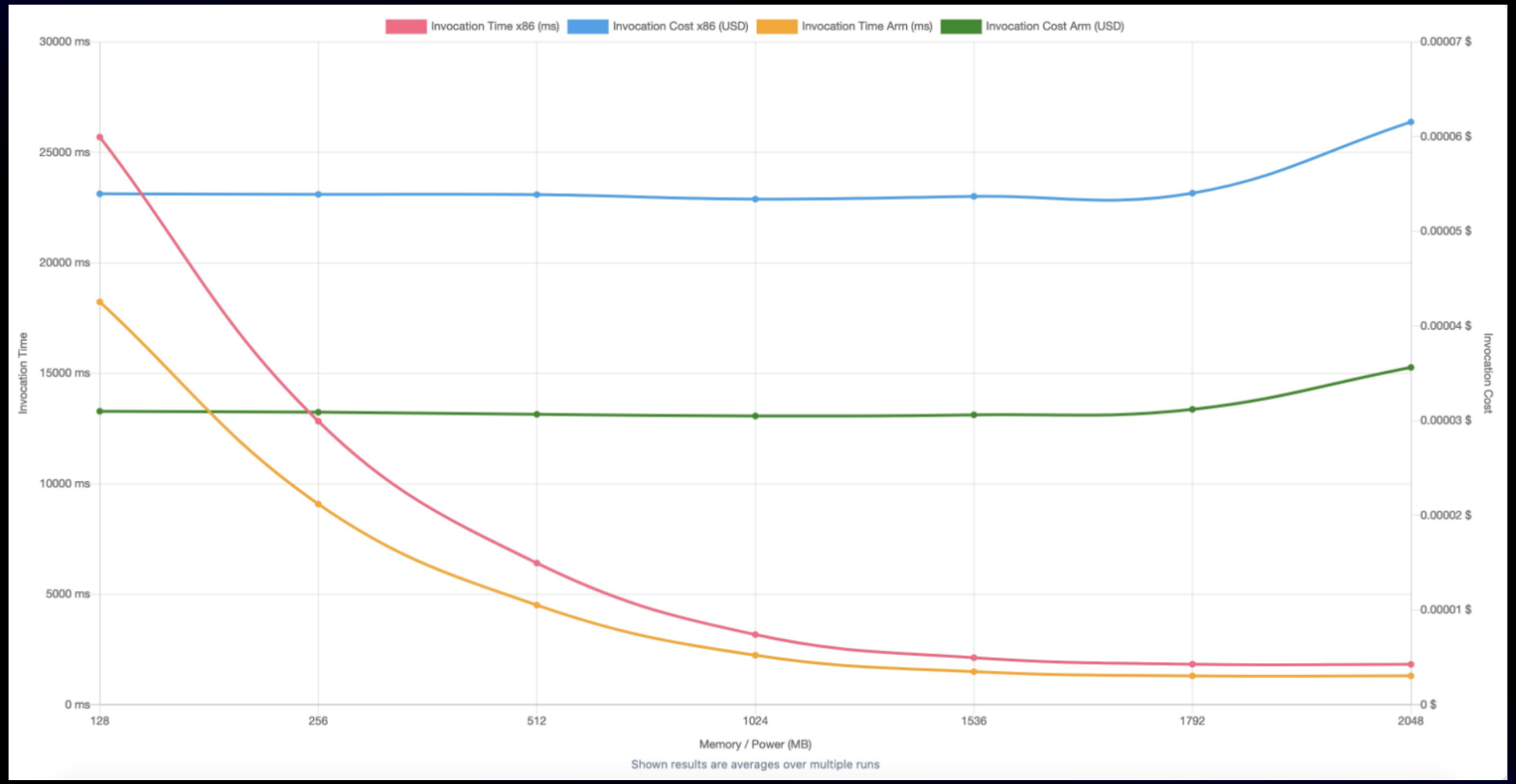
```
{
    "lambdaARN": "your-lambda-function-arn",
    "powerValues": [128, 256, 512, 1024, 2048, 3008],
    "num": 100,
    "payload": {"data": "abc"},
    "strategy": "speed|cost"
}
```

```
{
  "results": {
    "power": "128",
    "cost": 2.08e-7,
    "duration": 2.906,
    "stateMachine": {
      "executionCost": 0.00045,
      "lambdaCost": 0.0005252,
      "visualization": "https://lambda-power-tuning.show/ ... "
    }
  }
}
```

github.com/alexcasalboni/aws-lambda-power-tuning

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Lambda Power Tuning: Visualization



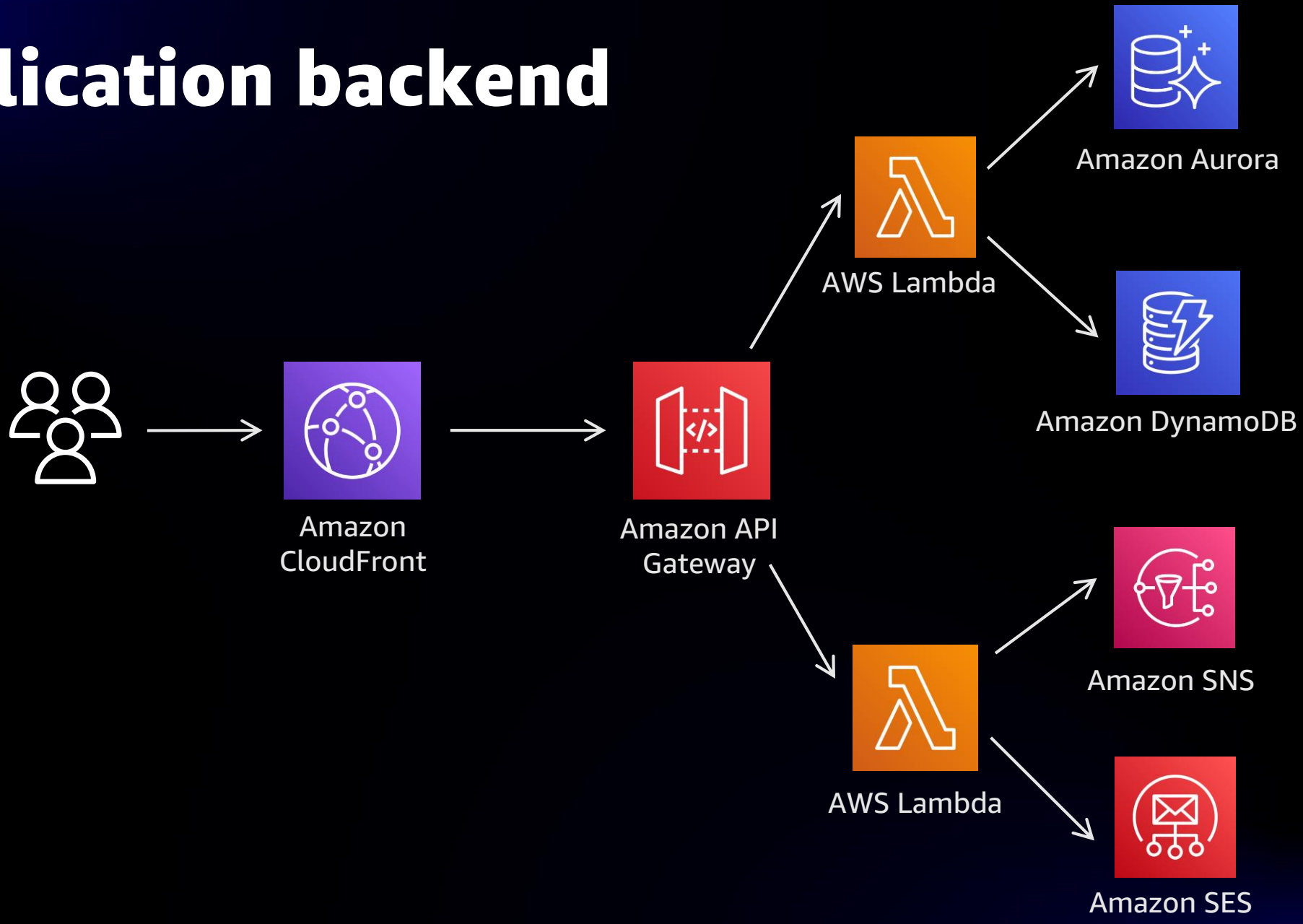
github.com/alexcasalboni/aws-lambda-power-tuning

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #3: Prioritize asynchronous processing where possible

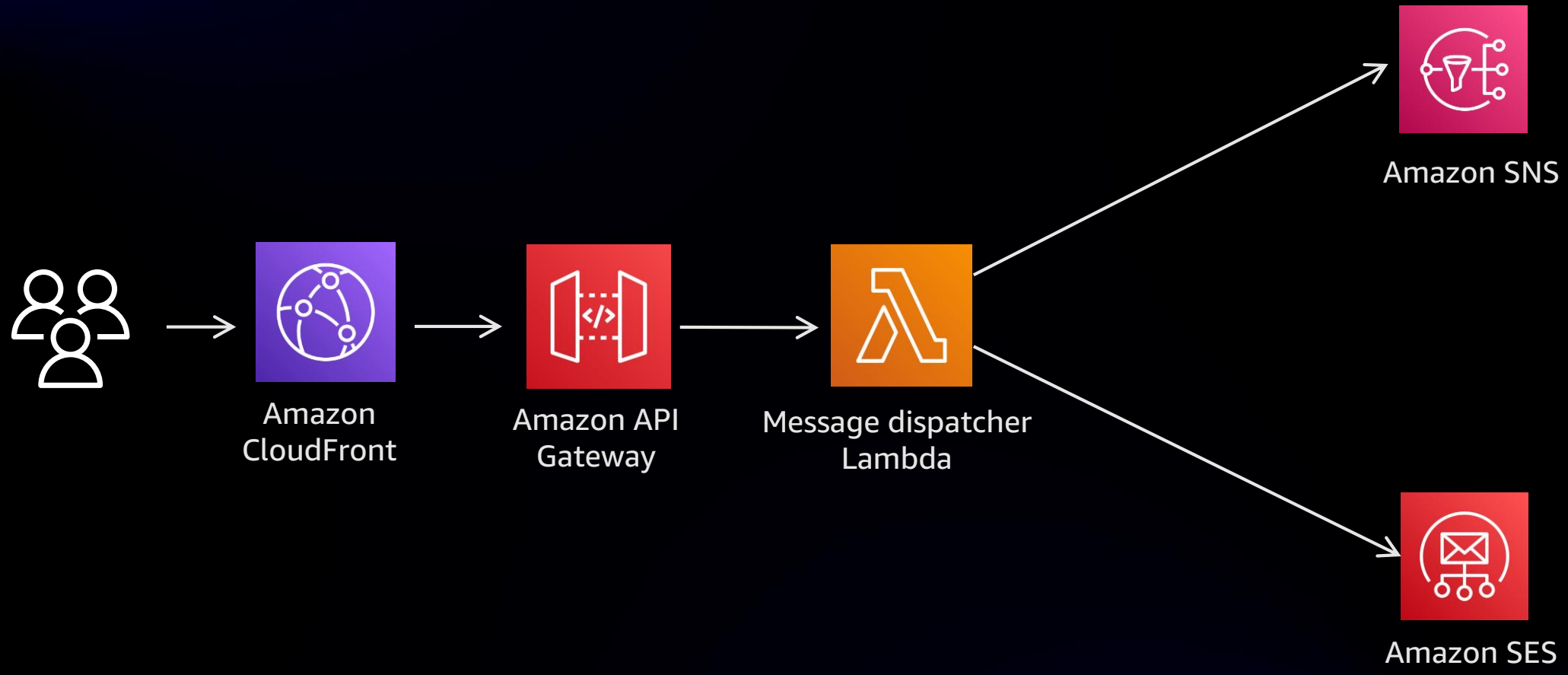
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend



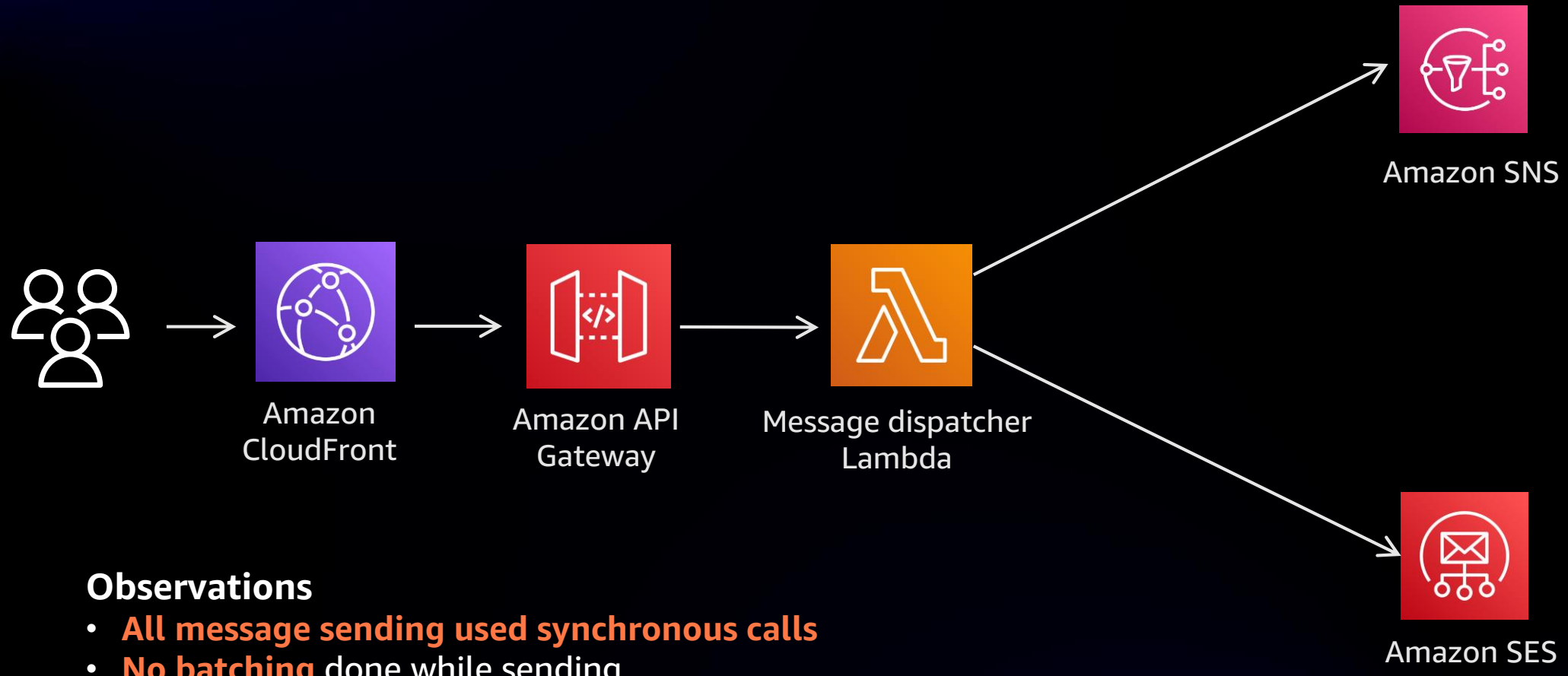
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Messaging components



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Messaging components

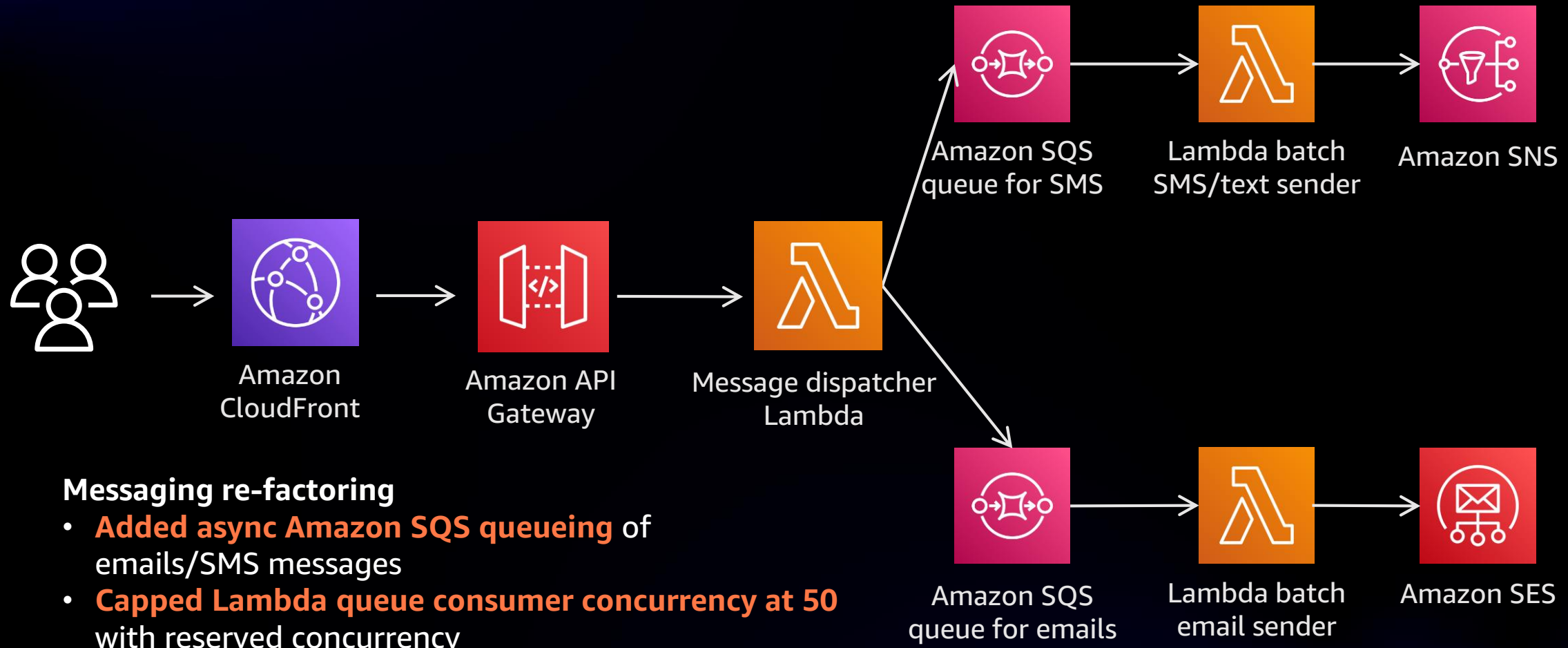


Observations

- **All message sending used synchronous calls**
- **No batching** done while sending
- Email/text sending occurred in bursts and **could deplete Lambda concurrency quotas**
- Throttling possible when sending larger message volumes

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Messaging components



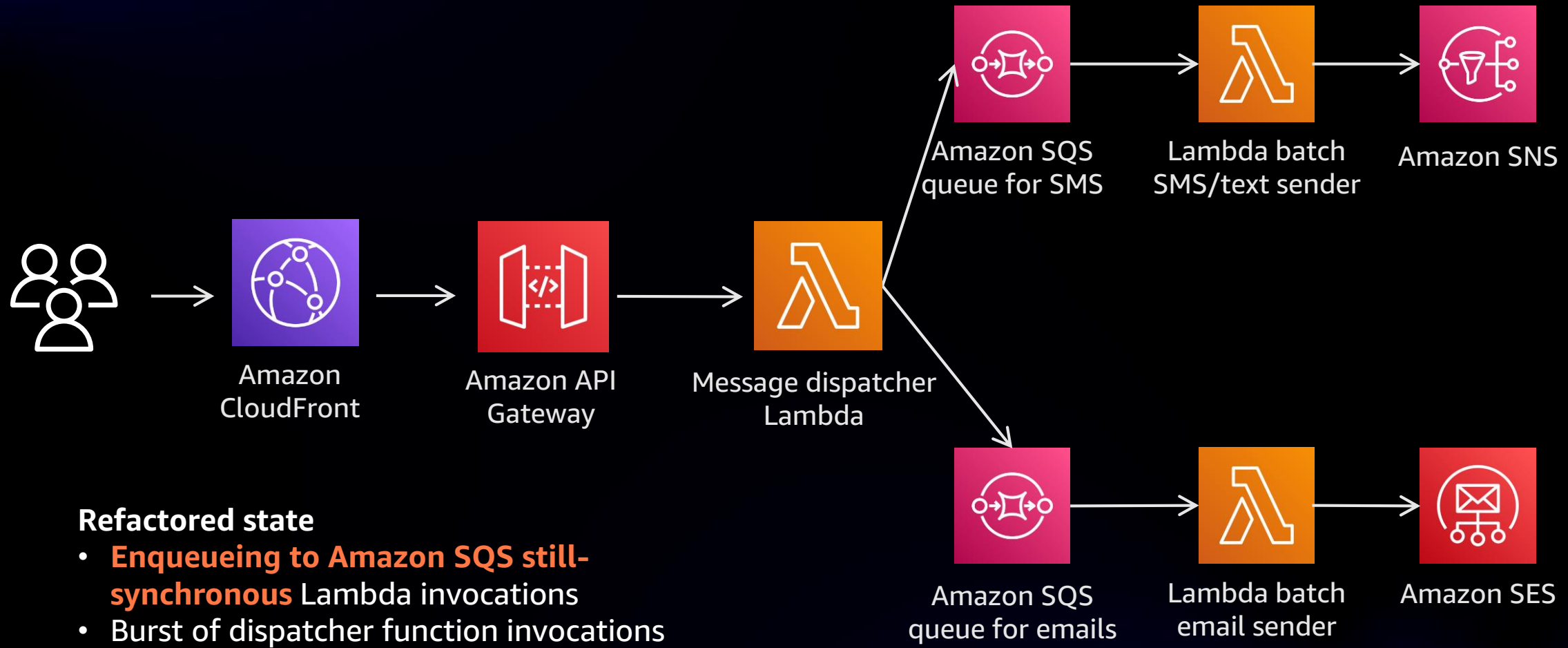
Messaging re-factoring

- **Added async Amazon SQS queueing** of emails/SMS messages
- **Capped Lambda queue consumer concurrency at 50** with reserved concurrency
- **Batch Amazon SQS reads reduced Lambda and SQS costs** and avoided overrunning downstream quotas

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Messaging components

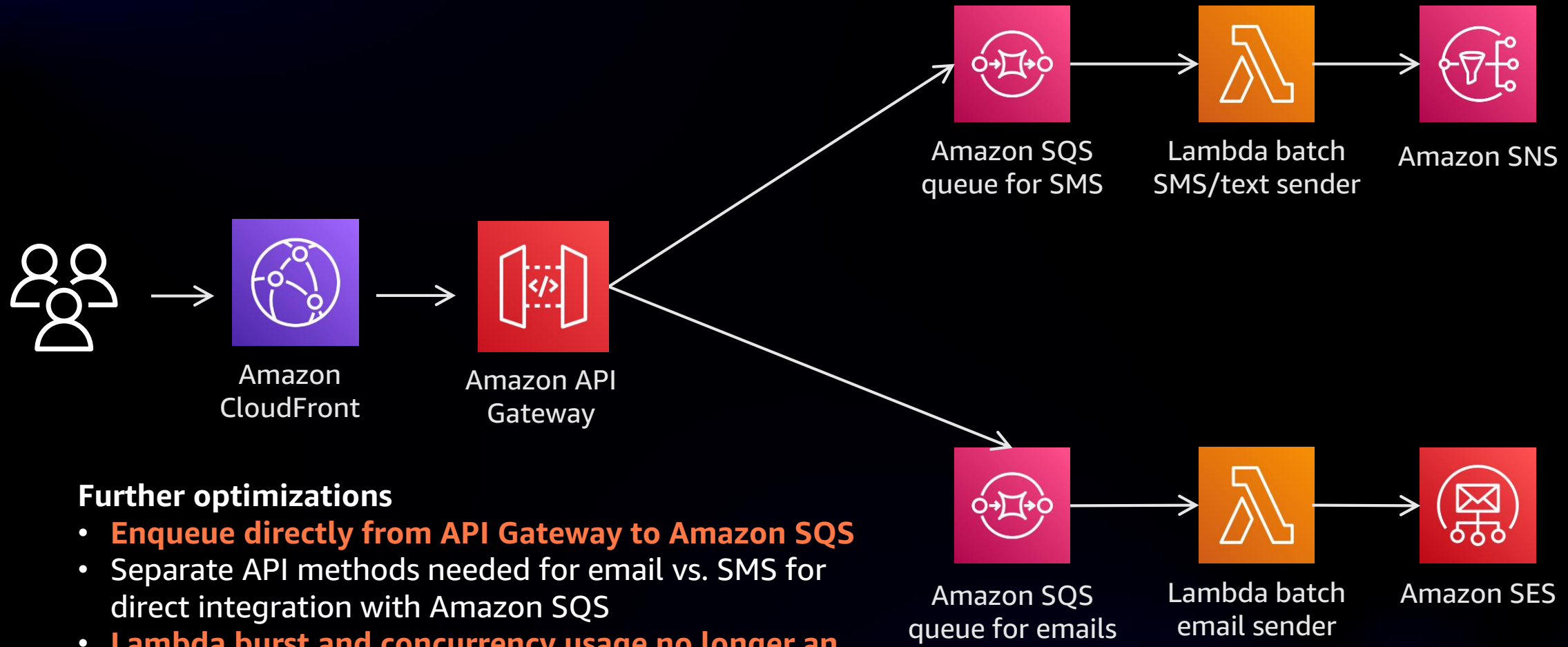


Refactored state

- **Enqueueing to Amazon SQS still-synchronous** Lambda invocations
- Burst of dispatcher function invocations **could still deplete concurrency quota** for core scheduling function capabilities

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Messaging components



Further optimizations

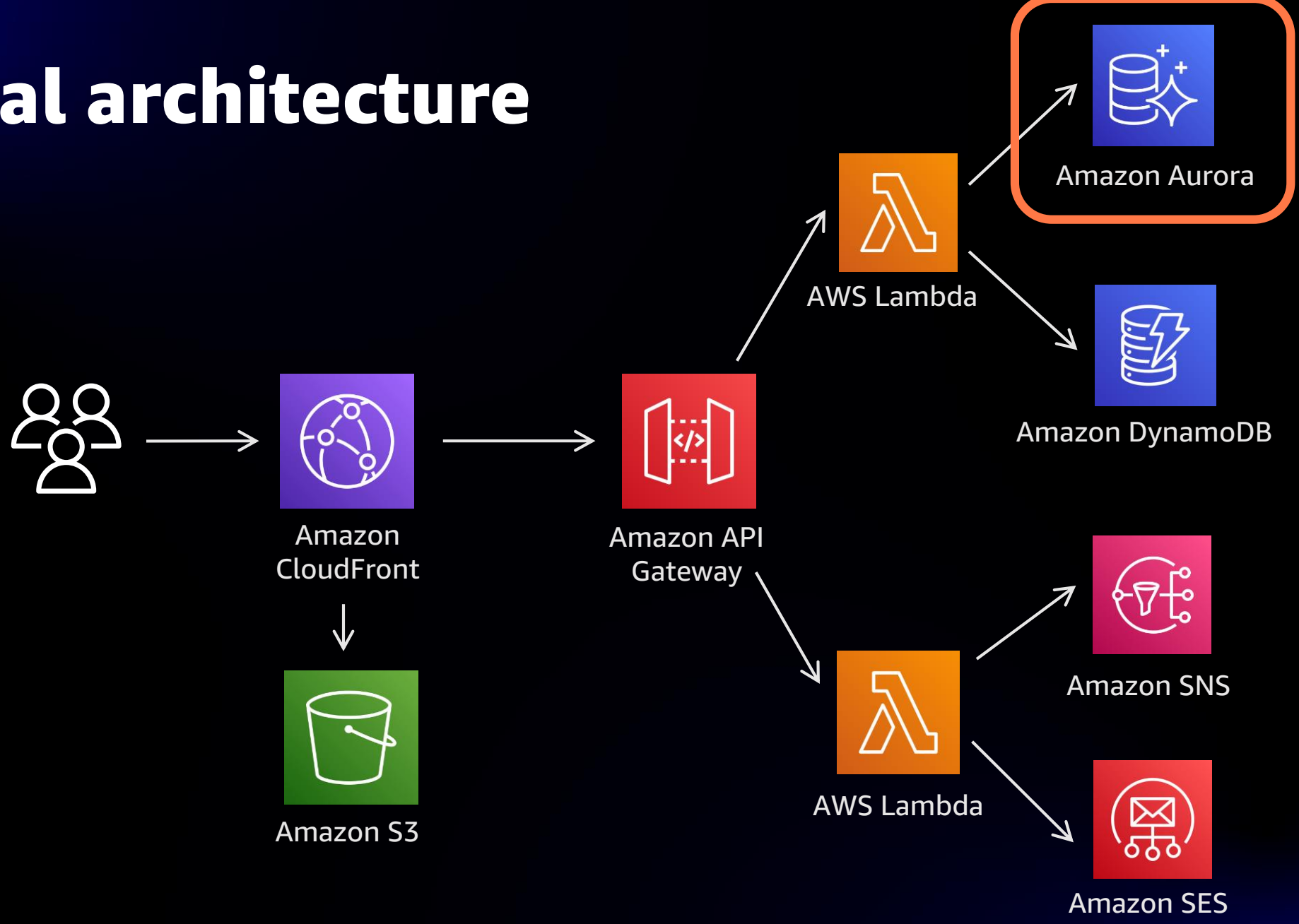
- **Enqueue directly from API Gateway to Amazon SQS**
- Separate API methods needed for email vs. SMS for direct integration with Amazon SQS
- **Lambda burst and concurrency usage no longer an issue** due to reserved concurrency reading Amazon SQS

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #4: Prepare your databases for hyperscale

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Aurora read scaling options

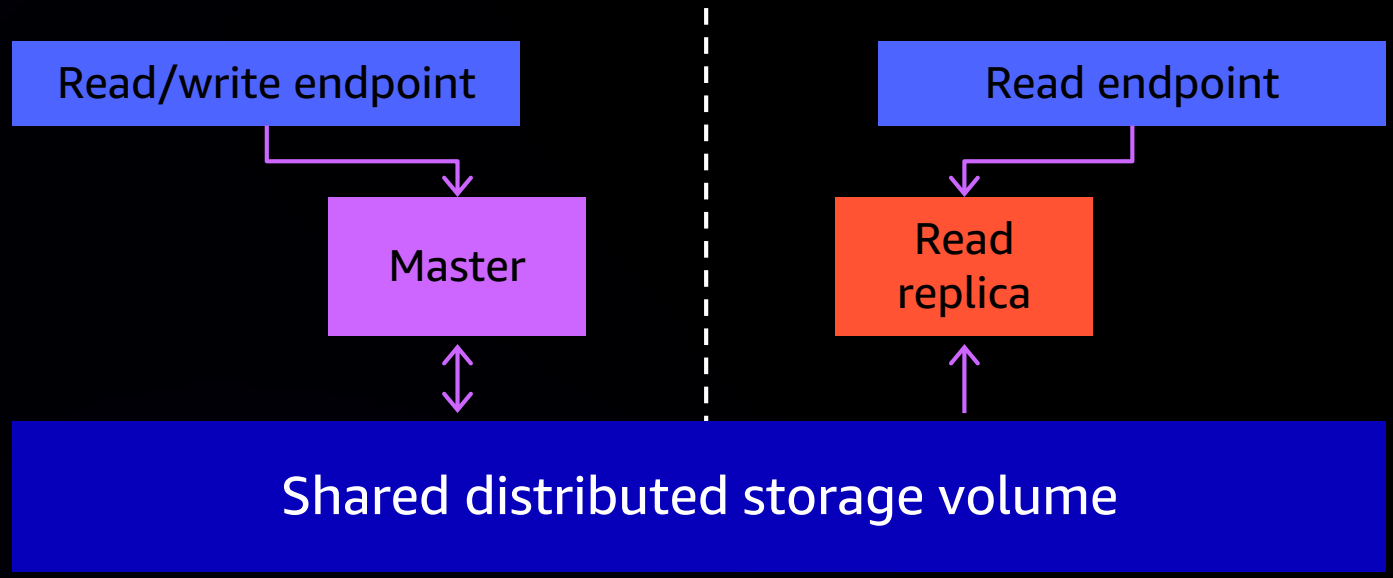
Up to 15 promotable read replicas

Separate reader/writer endpoints

Auto Scaling to automatically add and remove replicas

Physical replication across Regions (Aurora global database)

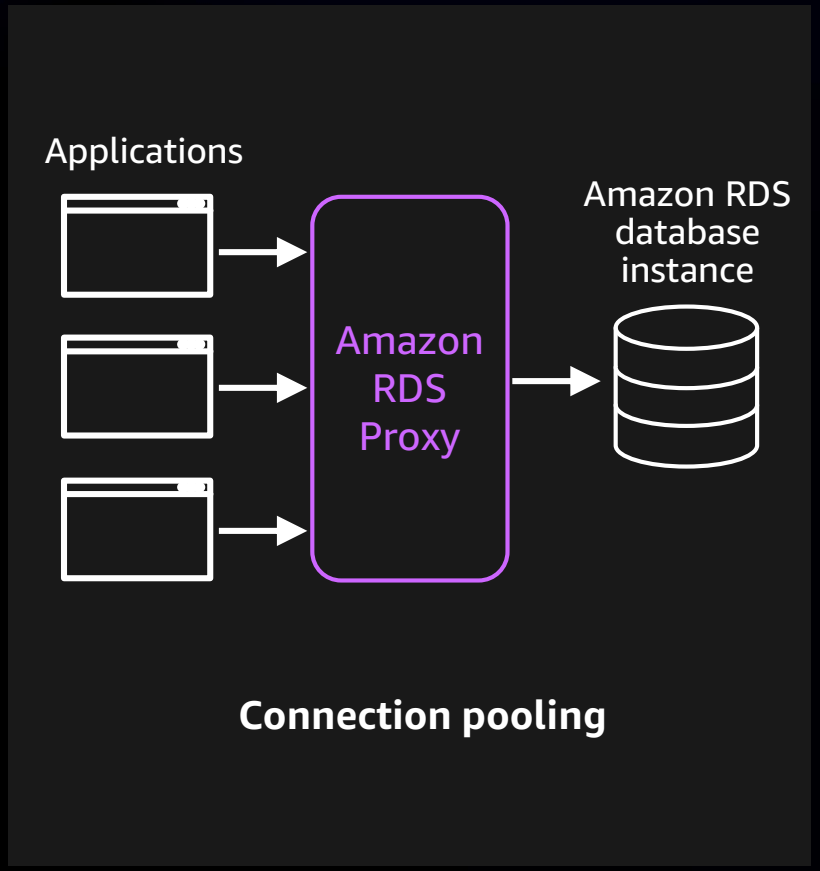
Logical (binlog) replication to any MySQL database



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon RDS Proxy

FULLY MANAGED, HIGHLY AVAILABLE DATABASE PROXY FOR AMAZON RDS



Supports a large number of application connections

Deployed across multiple AZs and fails over without losing a connection

Integrates with AWS Secrets Manager and IAM

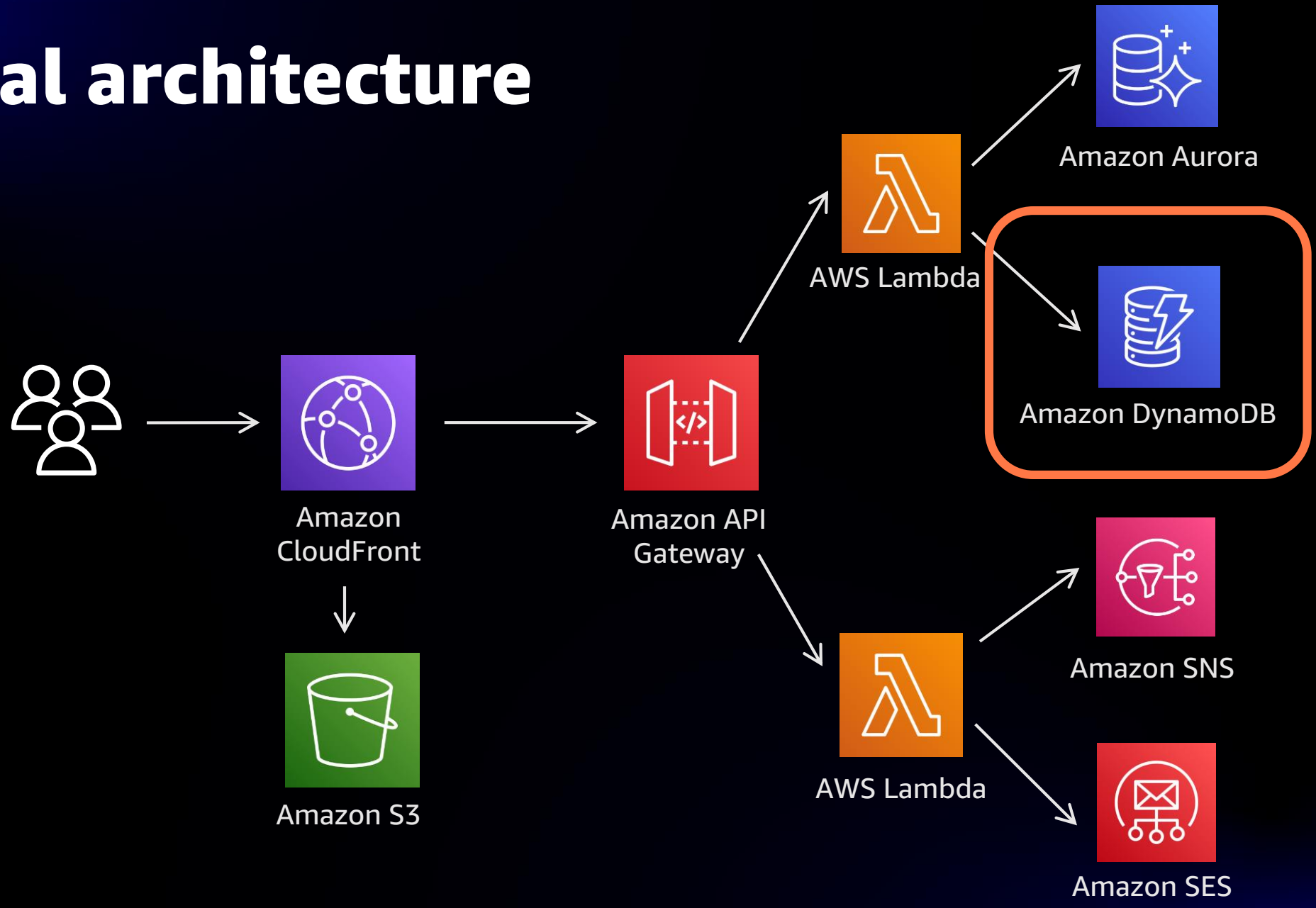
Get started with a few clicks in the console

Available: Aurora MySQL-Compatible Edition and Aurora PostgreSQL-Compatible Edition

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DynamoDB Scaling: Two modes of operation

On-demand mode

Unpredictable workloads

Frequently idle workloads

Events with unknown traffic

“Set it and forget it”

Priced based on count of read/write requests

Provisioned mode

Steady workloads

Gradual ramps

Events with known traffic

Ongoing monitoring

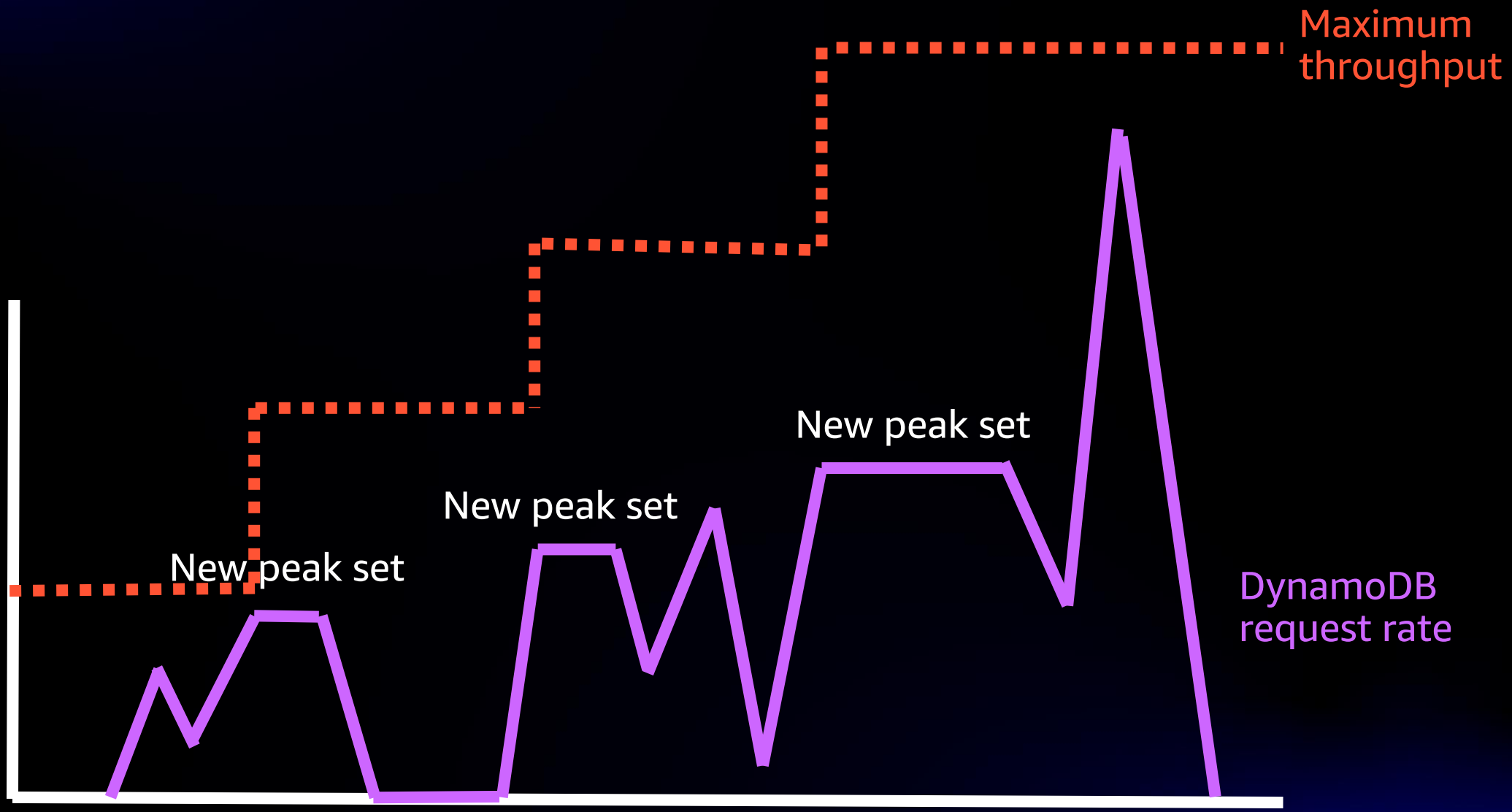
Priced based on read/write capacity units provisioned

Consider your tolerance for operational overhead and overprovisioning

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Up to twice your previous peak



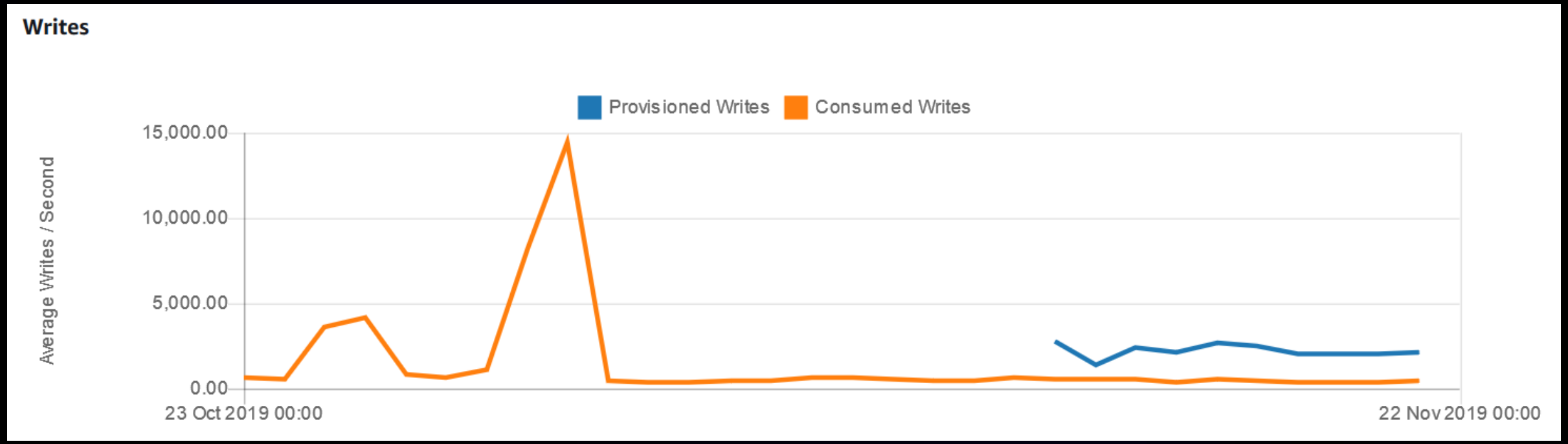
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Up to twice your previous peak



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Post-launch switchover: Conversion after load stabilization

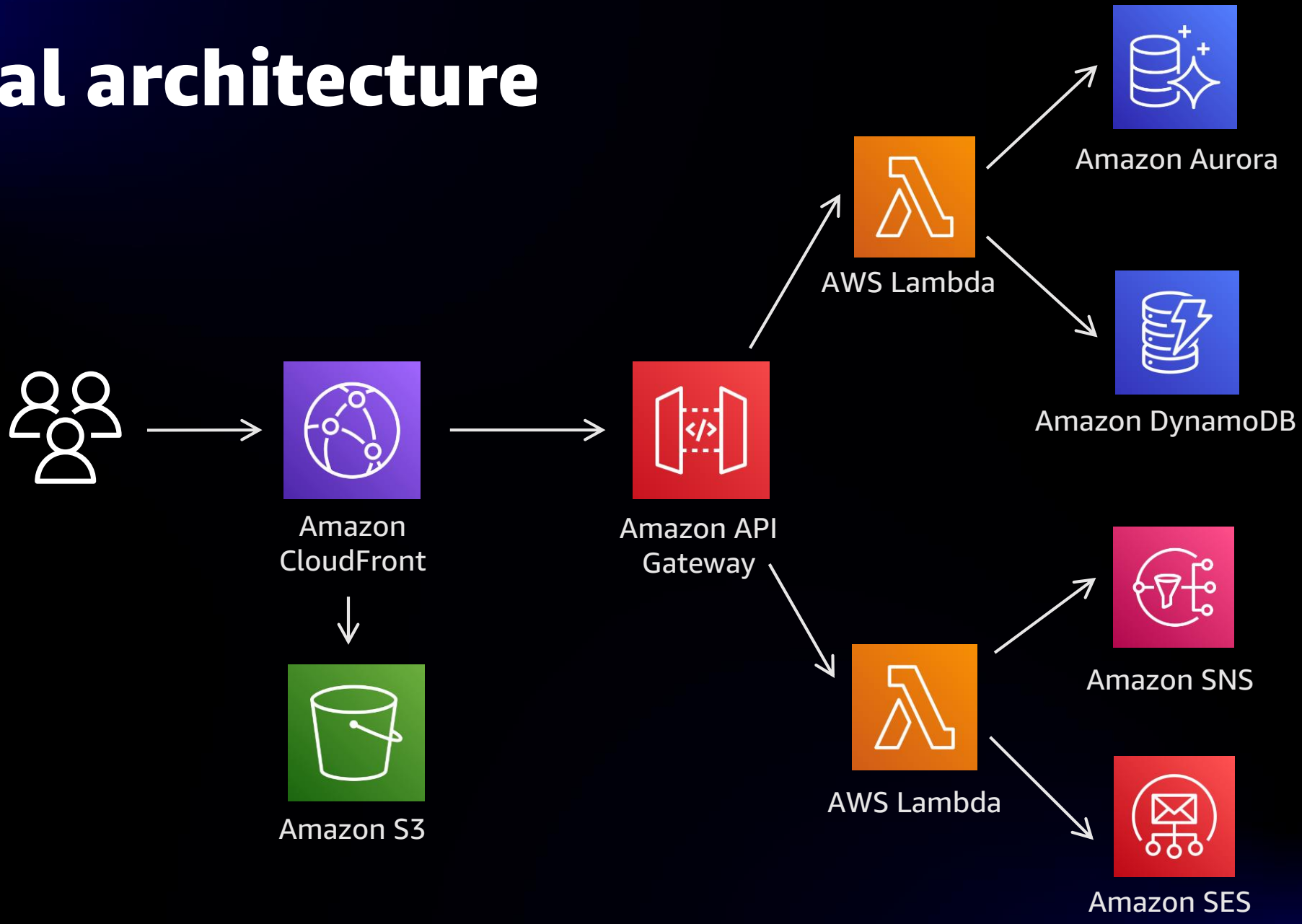


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #5: Leverage caching at multiple layers

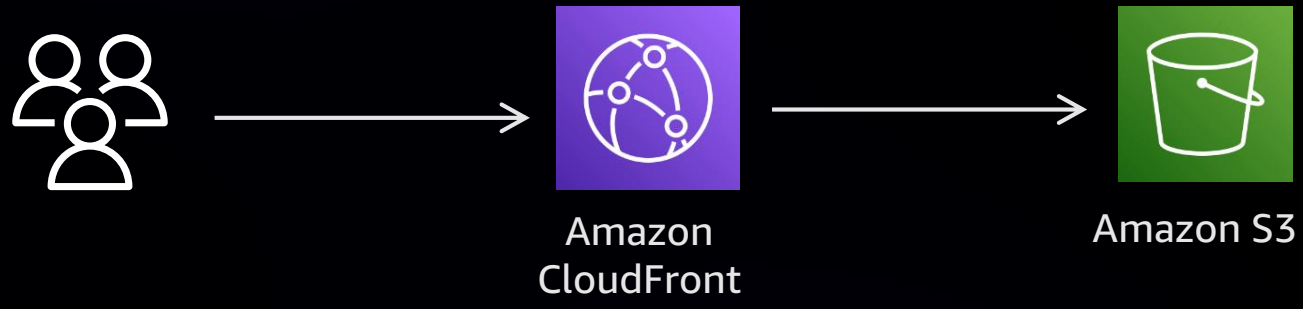
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application front end



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

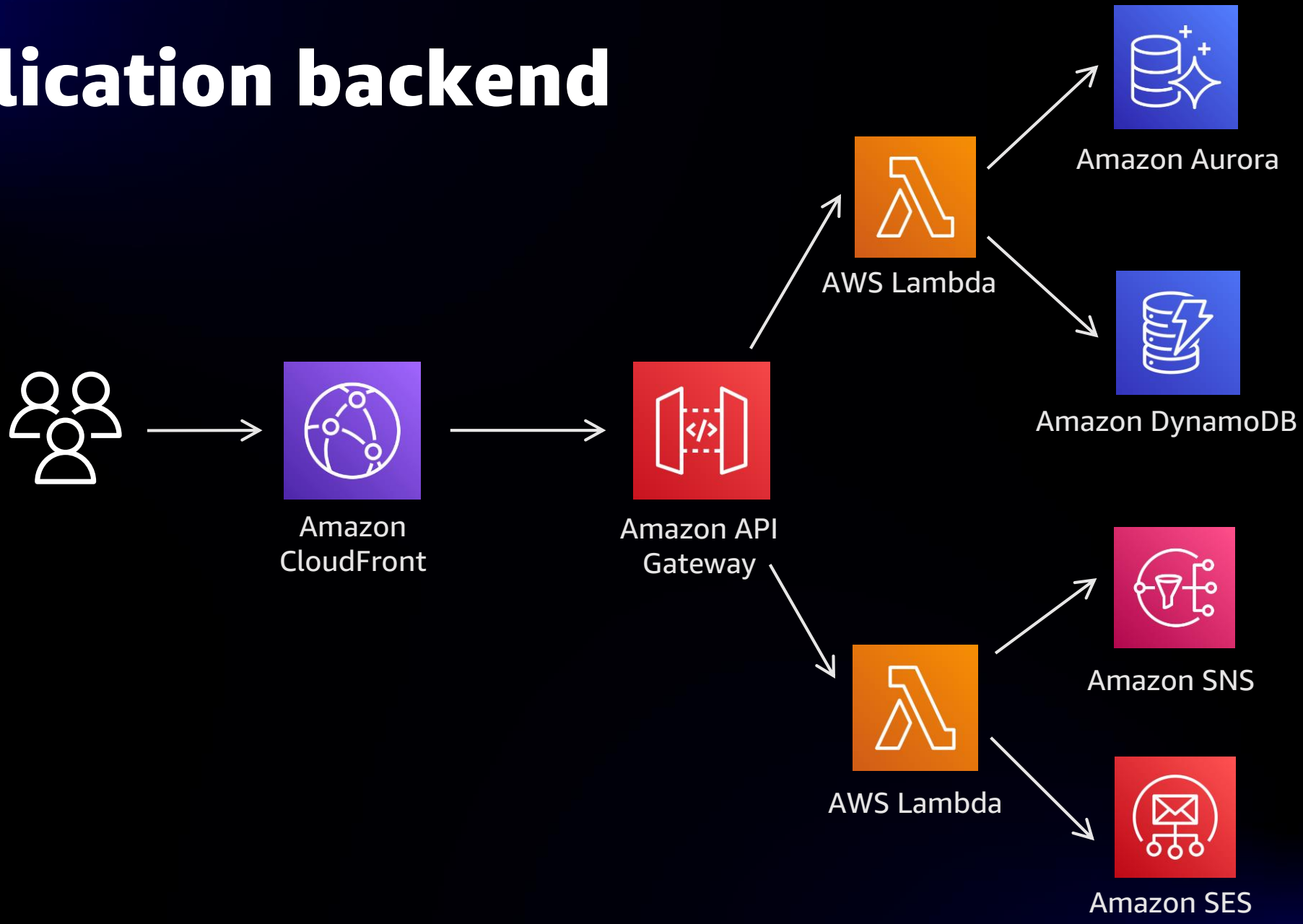
Application front end: Caching



Cache and accelerate all static content

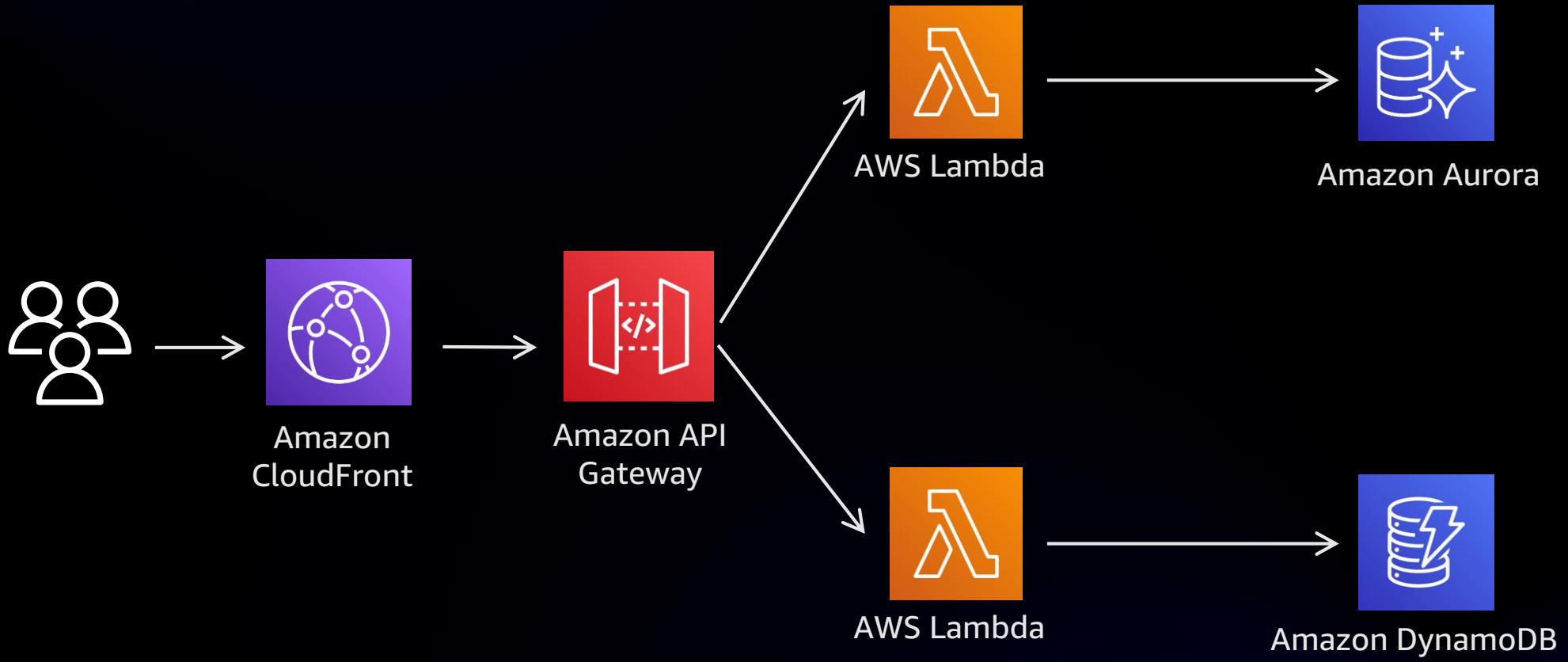
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend



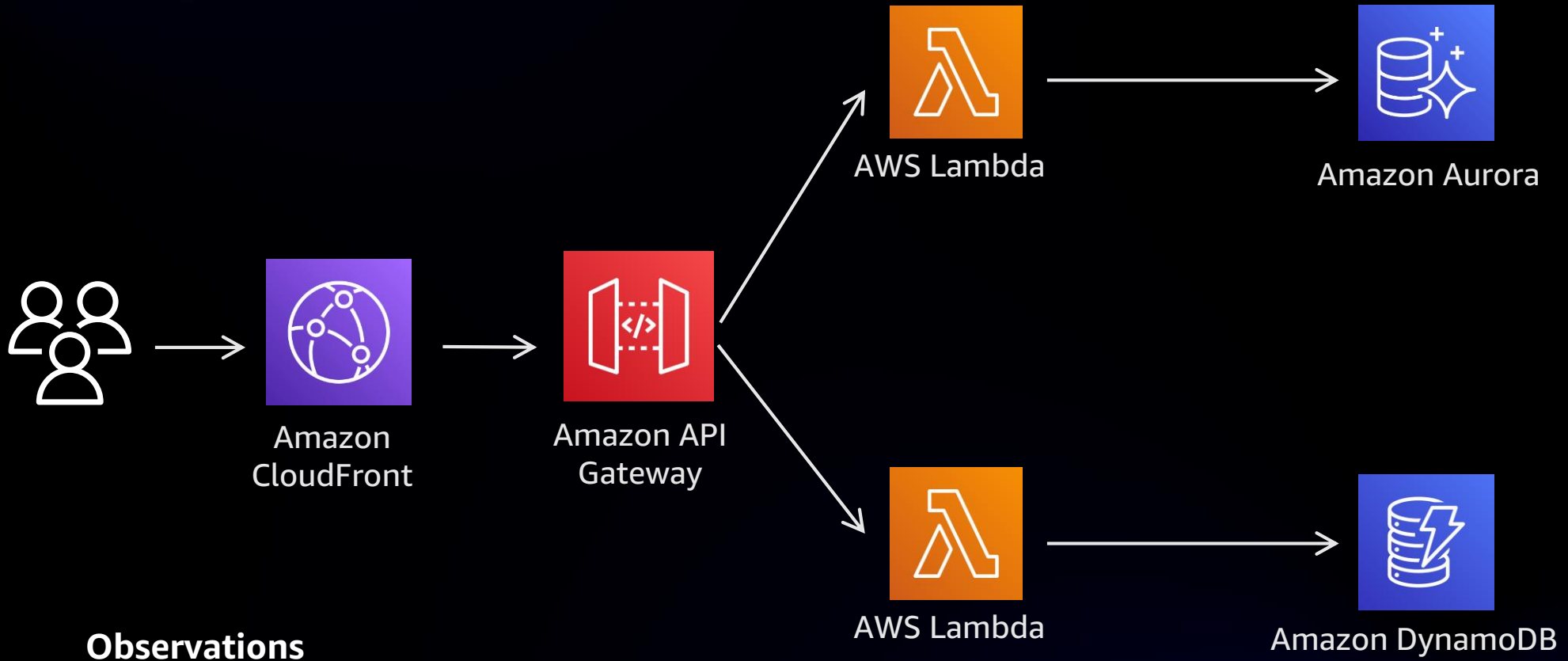
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Database interactions



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Database interactions

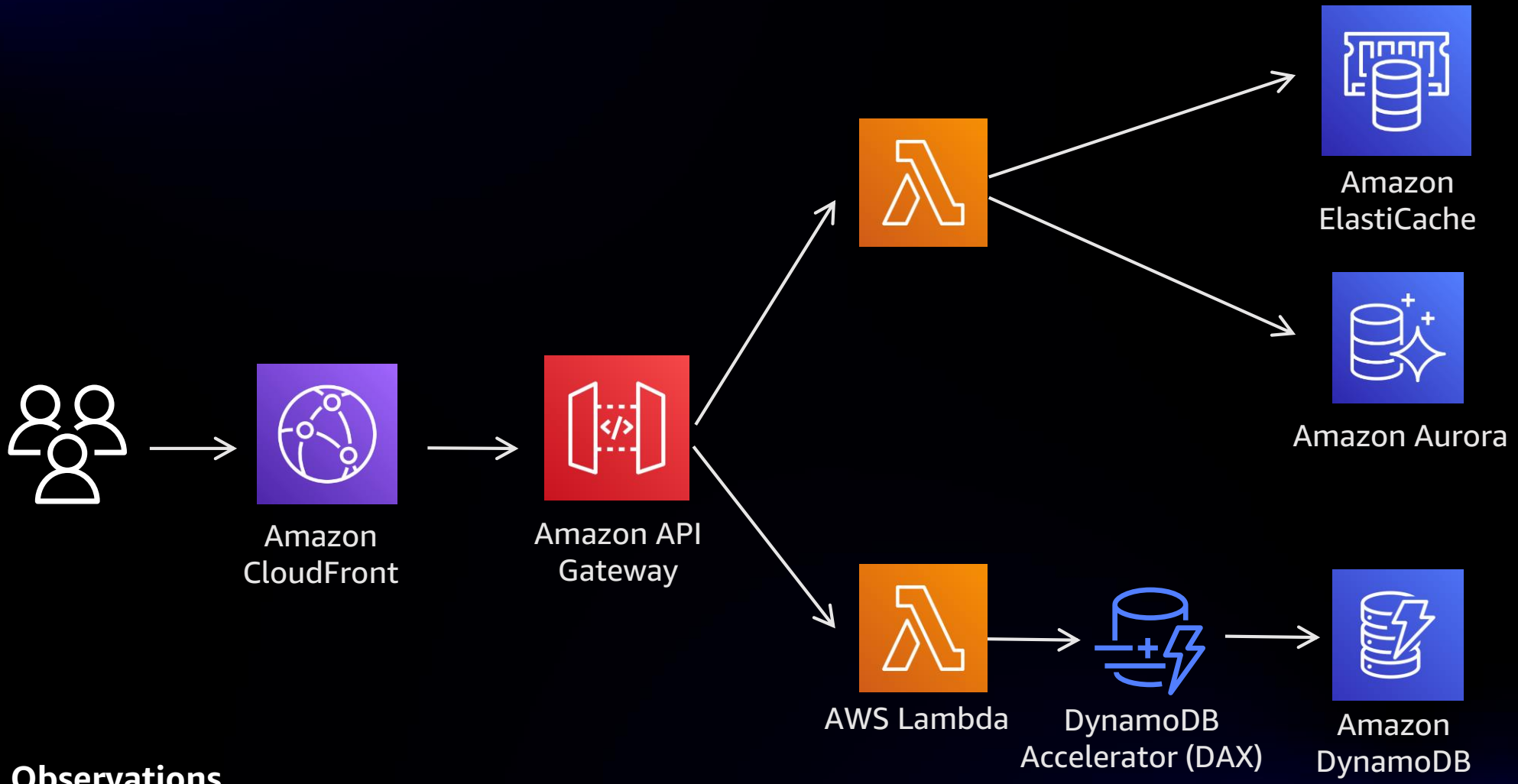


Observations

- **Initially ALL Lambda invocations were synchronous**
- Every request queried database
- Many lookups were cacheable responses

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Database caching

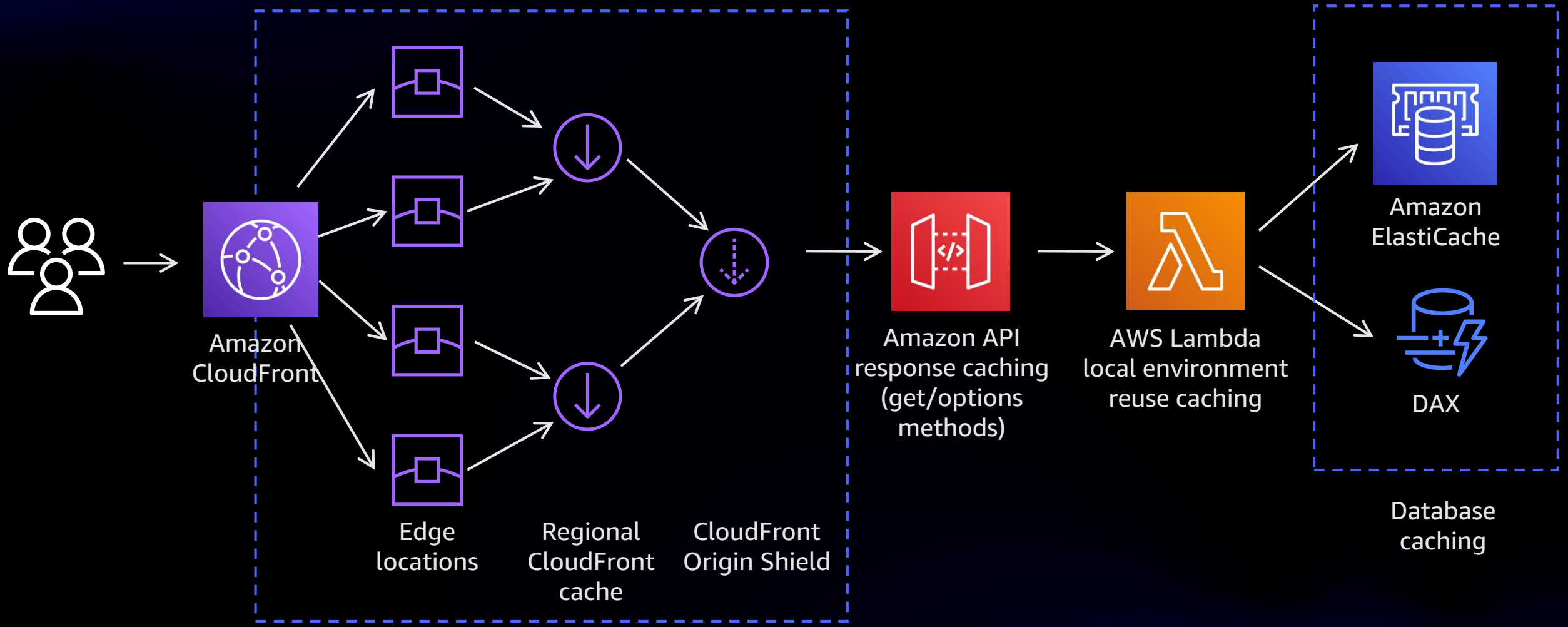


Observations

Database caching with DAX and/or Amazon ElastiCache

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Application backend: Multi-tiered caching



Caching closer to end user generally provides most benefit

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #6: Provision for peak when in doubt

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Estimating service quota needs

1. **Map user flows to expected API invocations** per API route
2. **Calculate Lambda concurrency** based on API Gateway route invocations and correlated function durations
   - Lambda concurrency = RPS x function duration in seconds
3. **Aggregate totals** across all API routes and all Lambda functions

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example: Calculating API route RPS

- **Marketing guidance:** 10,000 users expected in first 3 minutes for fire sale giveaway
- **Calculate average API route RPS**
  - 10,000/3 = 3,333 users per minute if steady traffic
  - 3,333/60 = 60 RPS for /checkout route
- **Round up RPS** for demand burst scenarios
  - With a free giveaway, users may be actively refreshing at 11:59 p.m.
- **Load-testing assumption: Invoke /checkout route at 120 RPS**

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example: Calculating Lambda concurrency

1. **Optimize cold-start and warm-start durations** for all functions
2. **Capture and document average durations** per function
3. **Calculate Lambda concurrency** for burst and expected states:
   - Peak concurrency = burst RPS x cold-start duration
   - Non-peak concurrency = non-peak RPS x warm-start duration

Checkout function	API route RPS	Function duration	Estimated concurrency
Burst/peak demand	120	500 ms (cold start)	60 concurrency
Non-peak demand	20	100 ms (warm start)	2 concurrency

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example: Aggregate totals

- **Aggregate RPS across all routes** for burst/peak and normal states
  - Compare to baseline 10,000 API Gateway RPS quota
- **Aggregate Lambda concurrency totals** across all functions
  - Compare to account's regional concurrency quota
  - Compare to Region-specific burst quota

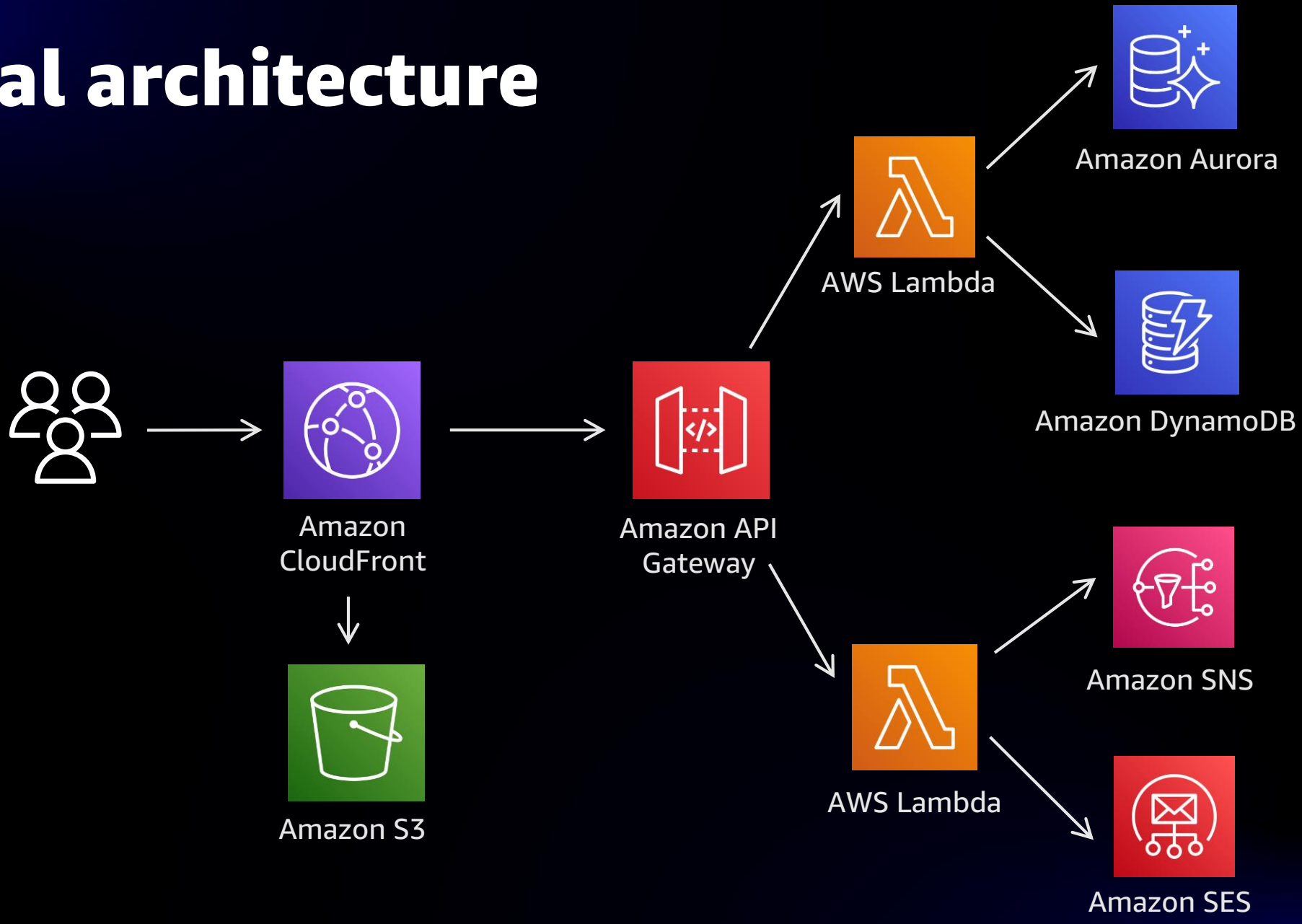
Summary	Estimated values
Lambda: expected concurrency (warm starts)	323
Lambda: max anticipated concurrency (warm starts)	2,015
Lambda: max 1-min burst concurrency (cold starts)	975
API Gateway: expected RPS	1,300
API Gateway: max anticipated RPS	9,000

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #7: Perform end-to-end load testing

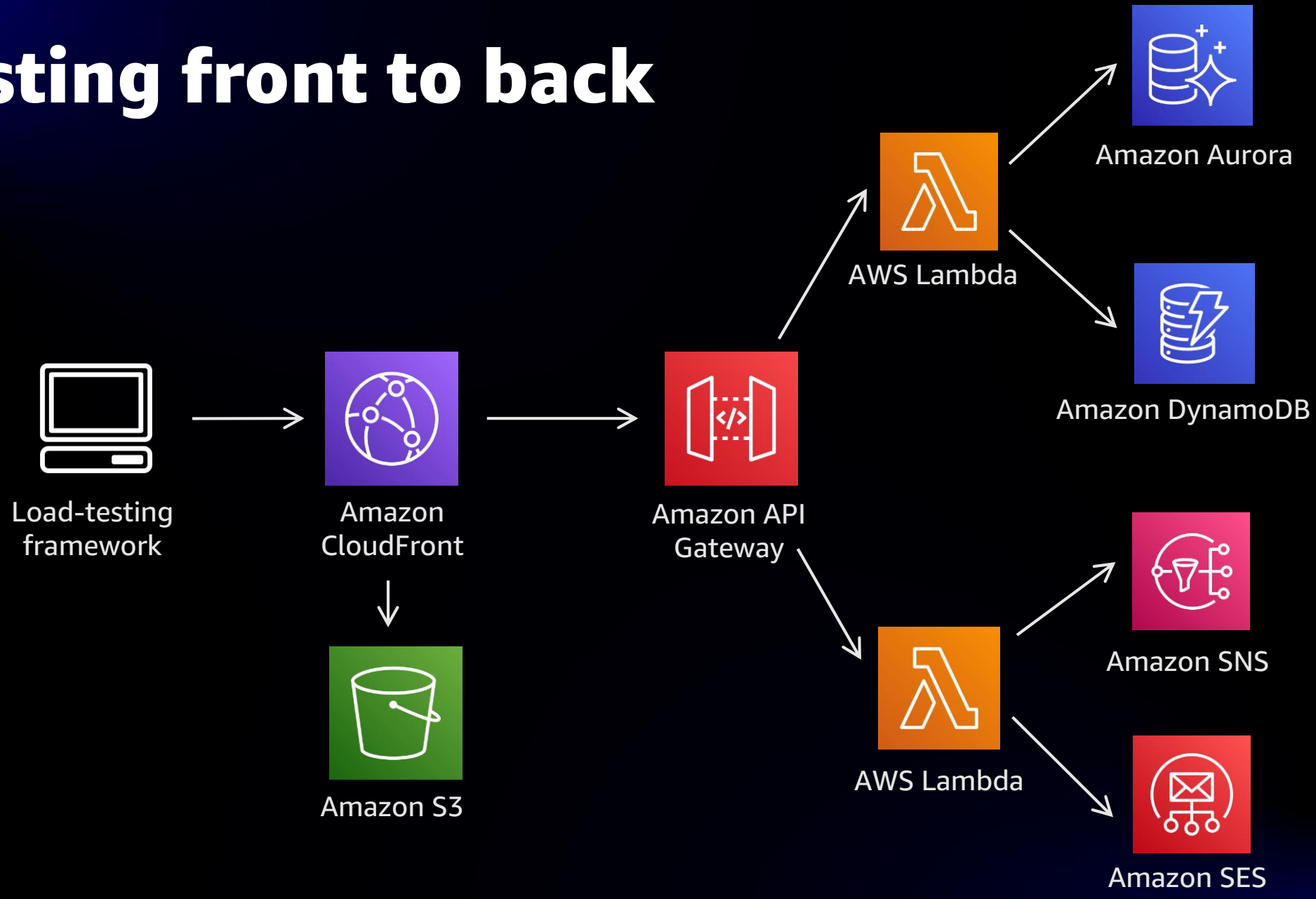
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



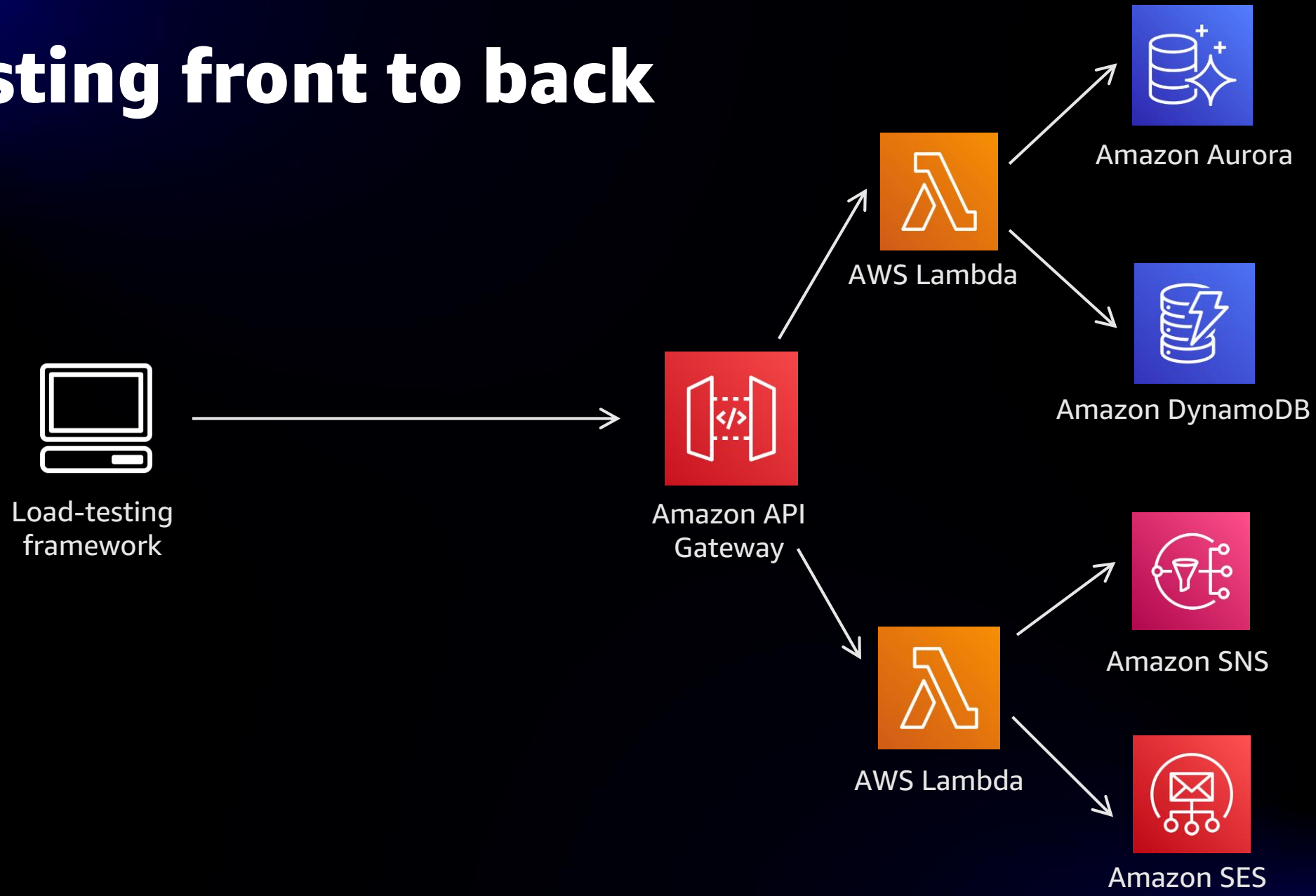
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Testing front to back



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Testing front to back



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Observe errors and latency with AWS X-Ray

Raise error from function to

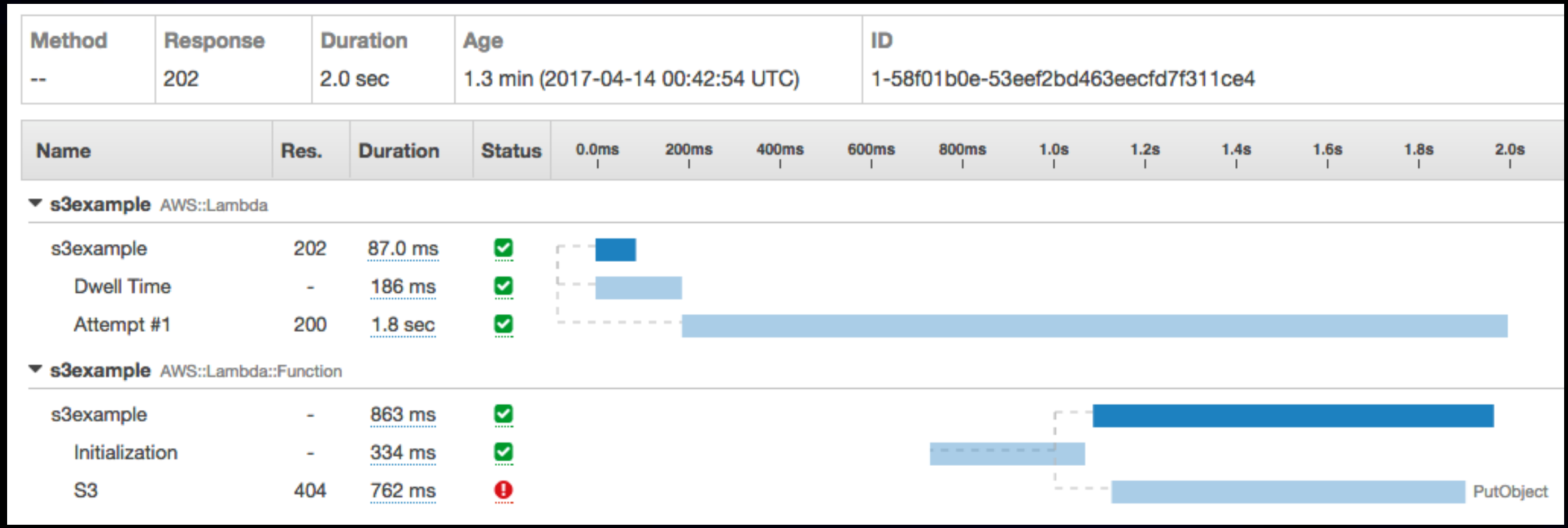
- Easily identify errors in traces
- Create error budgets via X-Ray trace groups

AWS X-Ray



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS X-Ray trace example



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Error handling and retry techniques

- **Set appropriate timeouts**
  - Set reasonable timeouts per Lambda function – do not set to 15 minutes
  - Configure API response timeout – default is 29 seconds
- **Retry with backoff**
  - Clients should retry, but with exponential backoff to not stampede the system
- **Implement idempotency**
  - Expect and account for retried operations
  - Application awareness and handling of scenarios is key

Dive deeper at https://s12d.com/reInvent2020-Error-Handling



aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #8: Mitigate traffic bursts through API rate limiting

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

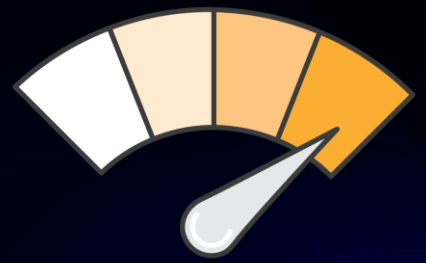
Technical implementation: API throttling

- **Protect backend systems through per-client API throttling**
- Prevents one customer from consuming all your backend system's capacity
- **Available natively as usage plans feature in API Gateway REST APIs**
- Lets you decide how to allocate capacity among your API consumers with quotas and request rates – example:

Tenant A users:
10 RPS, up to 100 calls/day

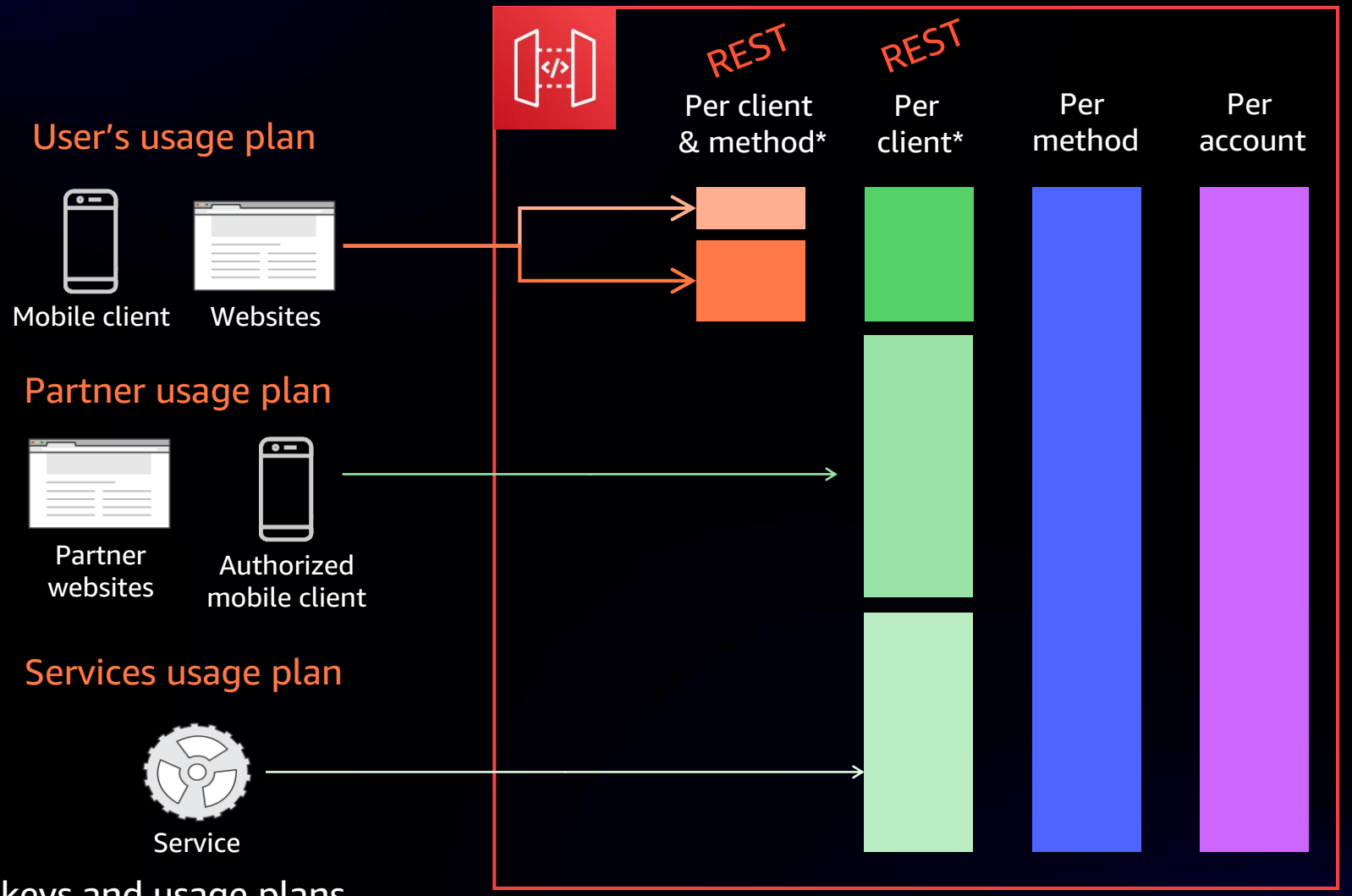


Tenant B users:
500 RPS, no limit on calls/day



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

API Gateway: Four levels of API throttling



REST

*Requires API keys and usage plans

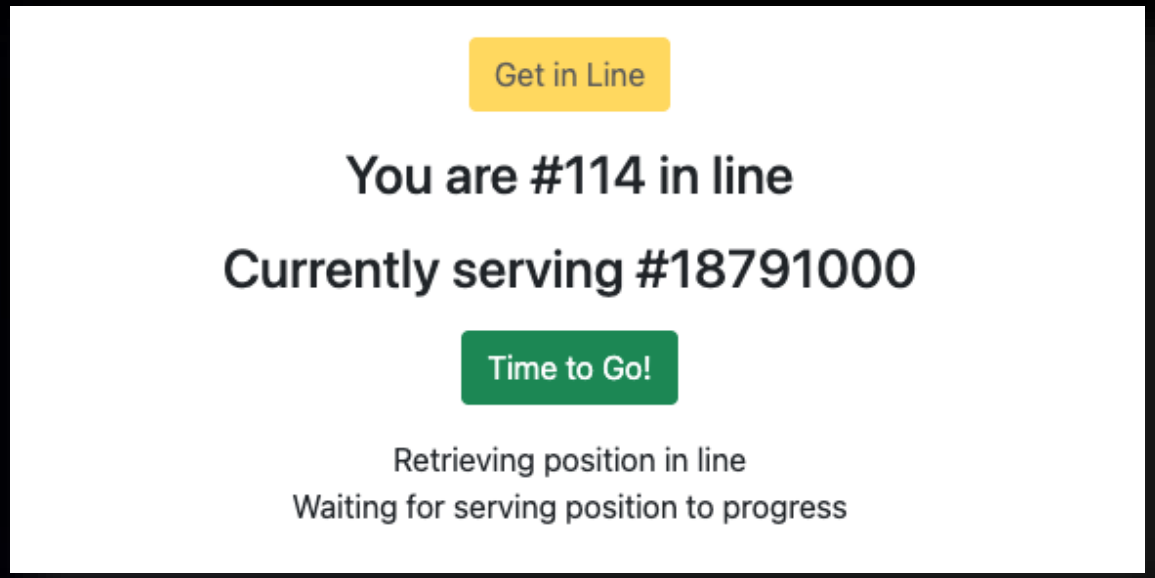
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practice #9: Integrate a virtual waiting room

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Virtual waiting room

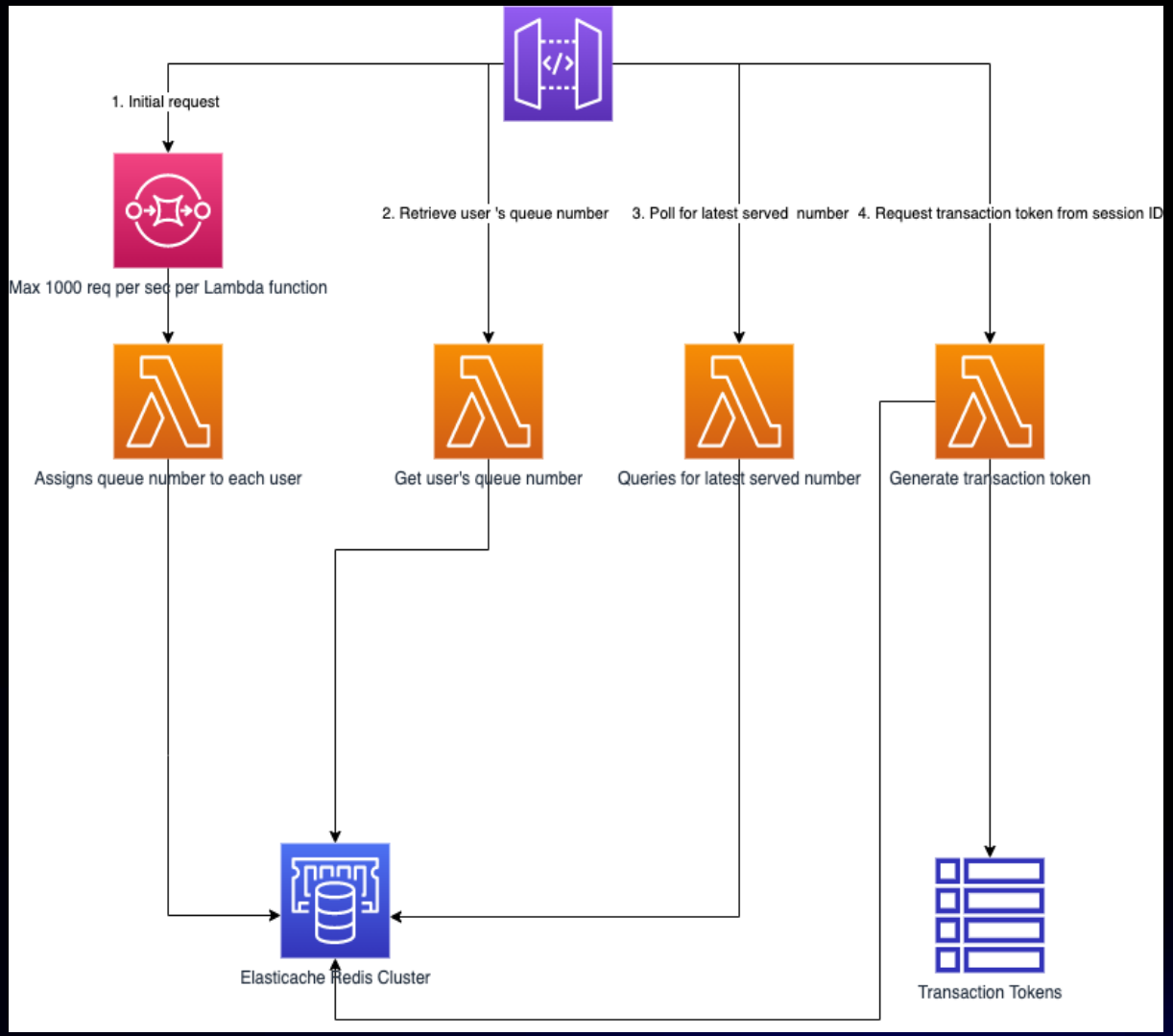
- Offer **customer-facing rate limiting**
- Provides customers **real-time insights into place in line**
- Simple idea, but can be difficult to build in cost-effective manner at scale



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Virtual Waiting Room

- AWS Virtual Waiting Room solution built with AWS serverless services and asynchronous patterns
- Load-tested to 100,000 concurrent waiting room users with entry of 1,000 users/sec
- Open-source customizable implementation under Apache 2.0 license
- Pluggable module that can be integrated into new and existing apps
- Available now at github.com/aws-solutions/aws-virtual-waiting-room



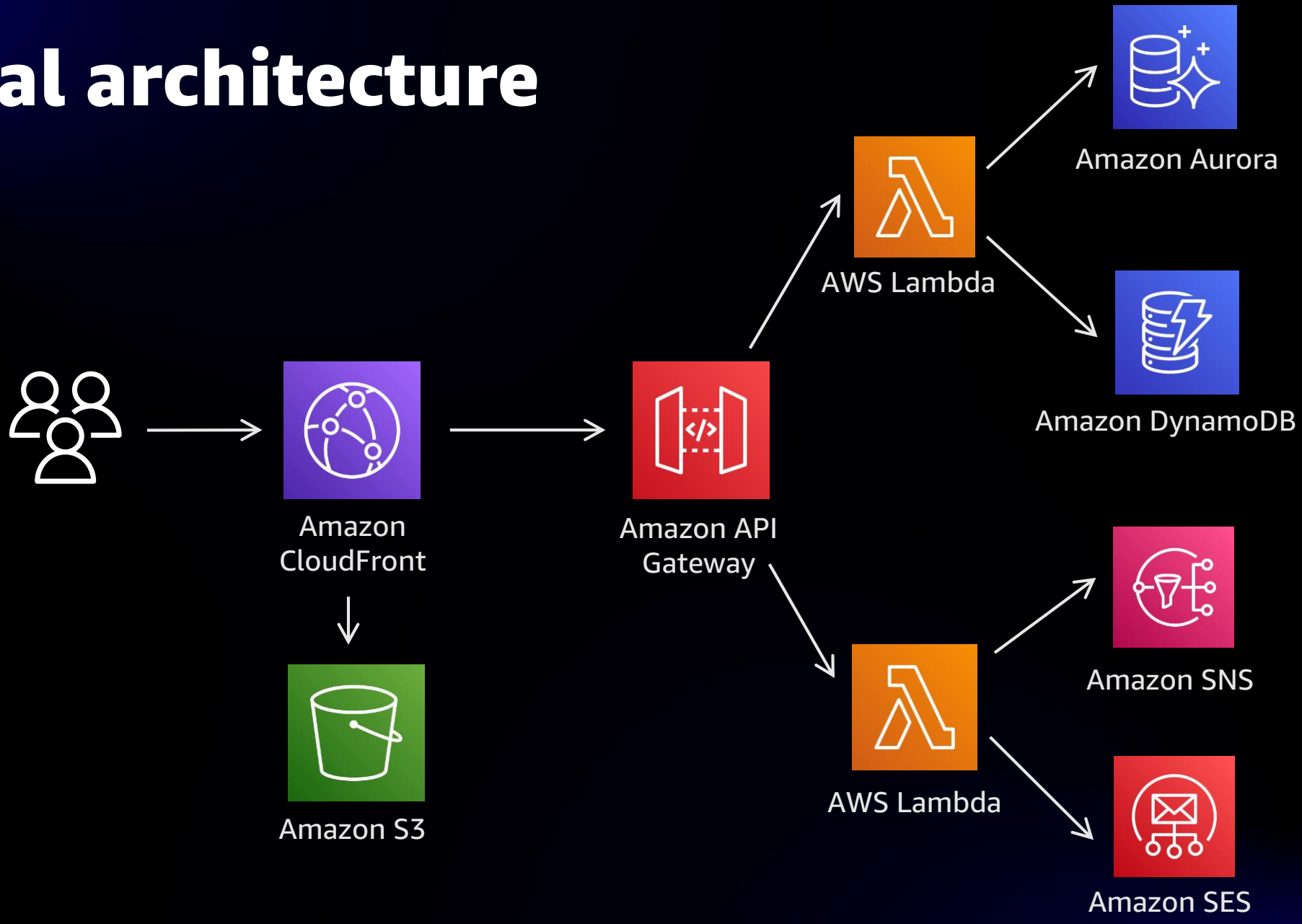
aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hyperscale practices summary

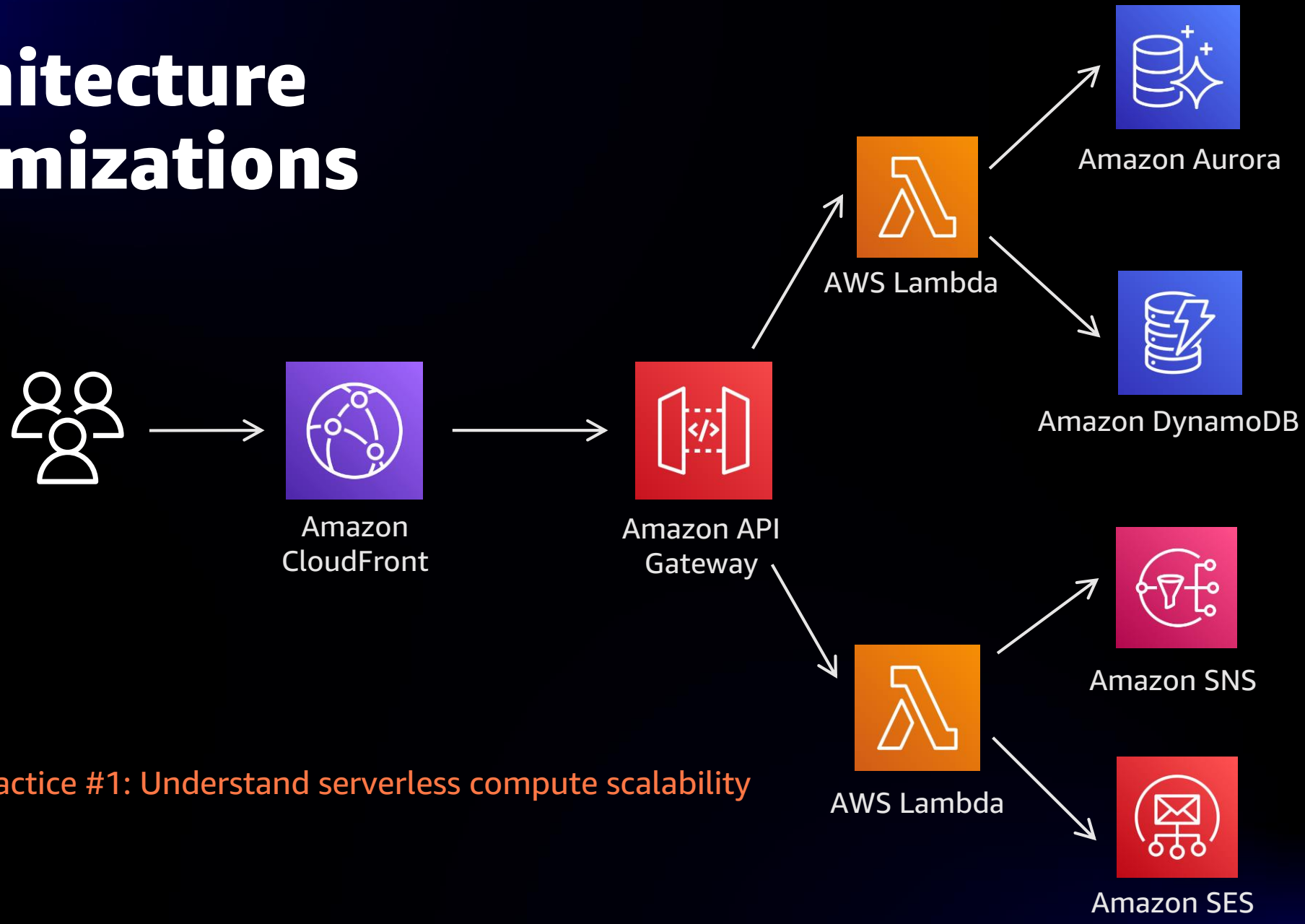
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Initial architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

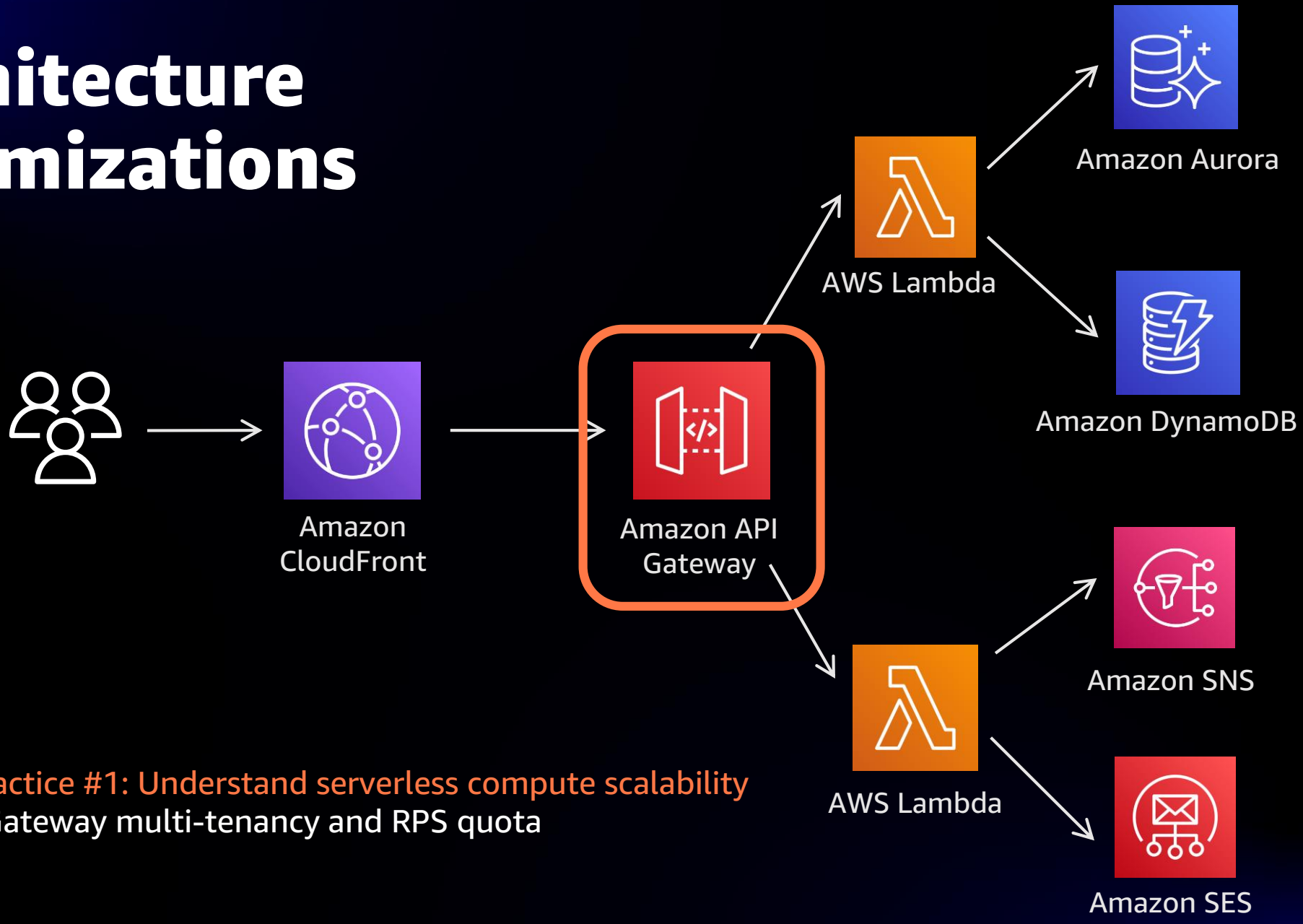
Architecture optimizations



Best practice #1: Understand serverless compute scalability

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations

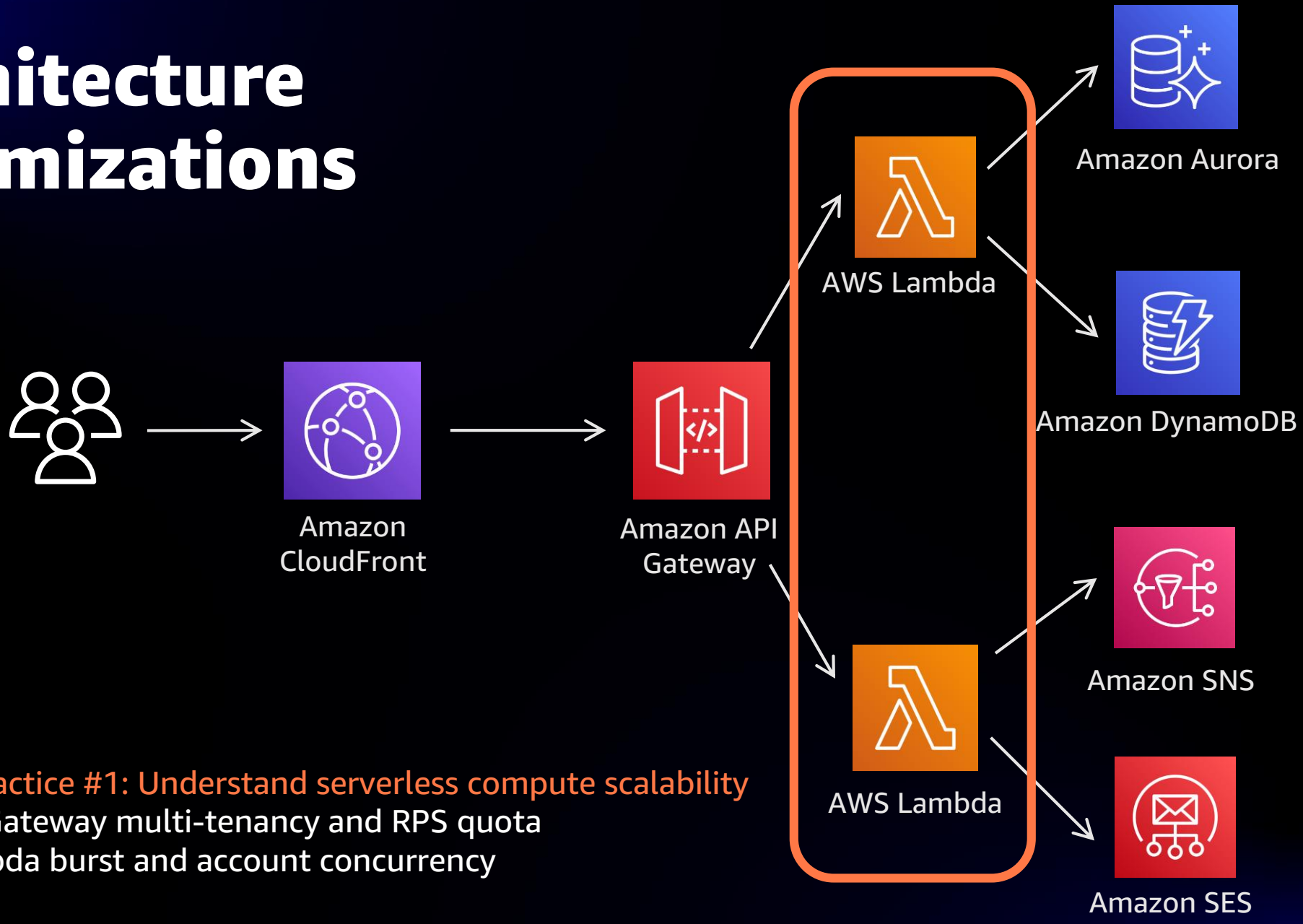


Best practice #1: Understand serverless compute scalability

- API Gateway multi-tenancy and RPS quota

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations

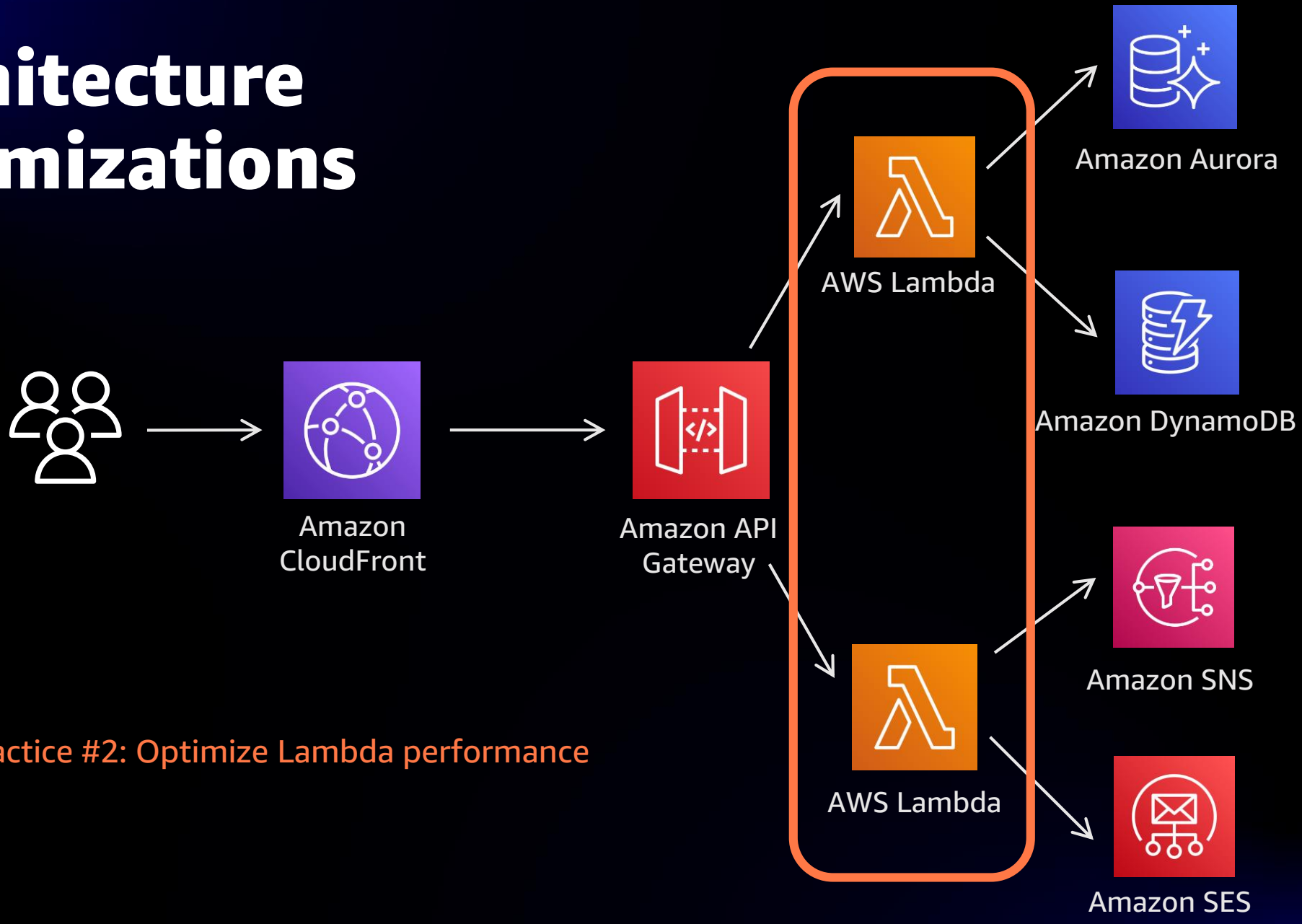


Best practice #1: Understand serverless compute scalability

- API Gateway multi-tenancy and RPS quota
- Lambda burst and account concurrency

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

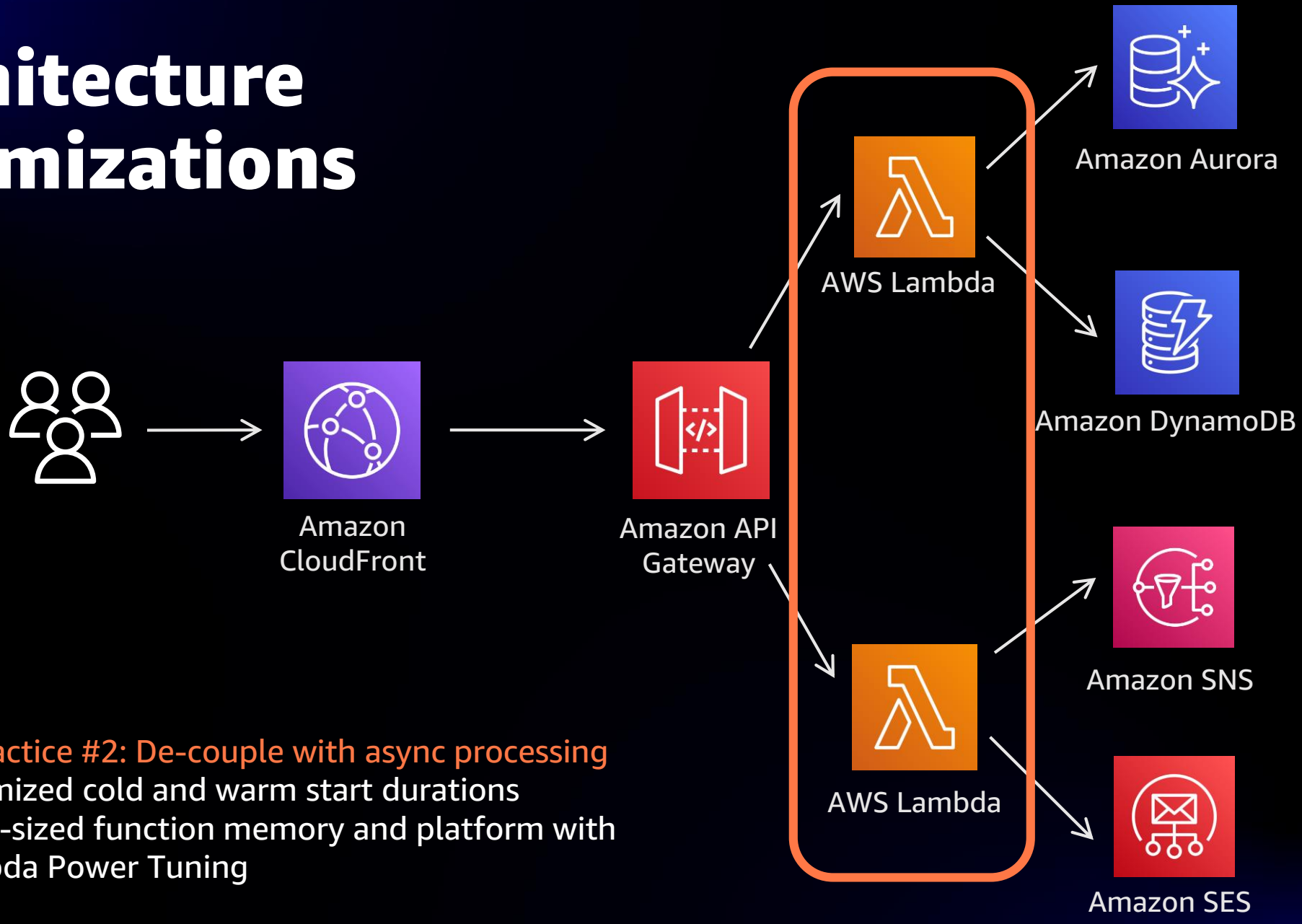
Architecture optimizations



Best practice #2: Optimize Lambda performance

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations

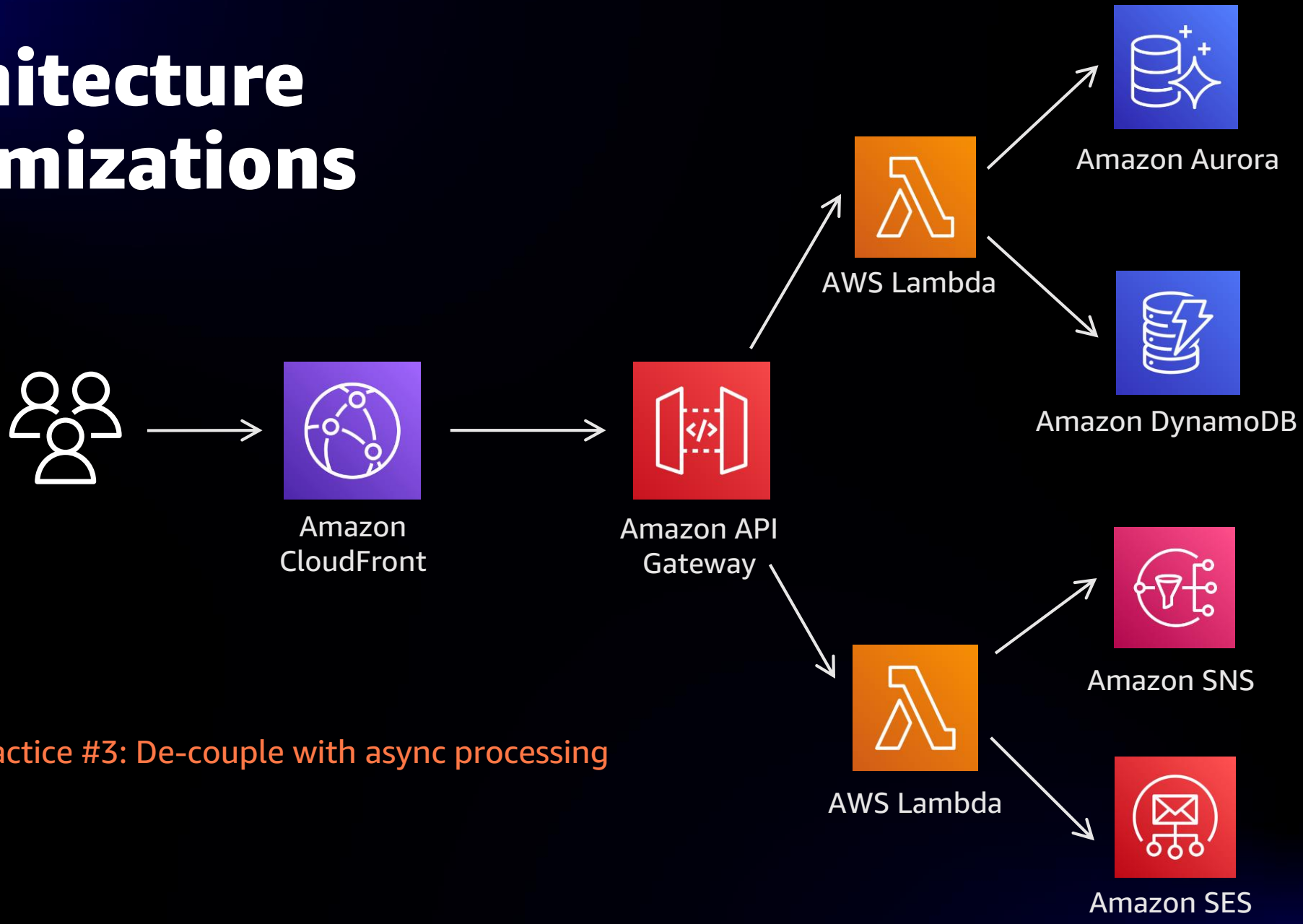


Best practice #2: De-couple with async processing

- Optimized cold and warm start durations
- Right-sized function memory and platform with Lambda Power Tuning

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

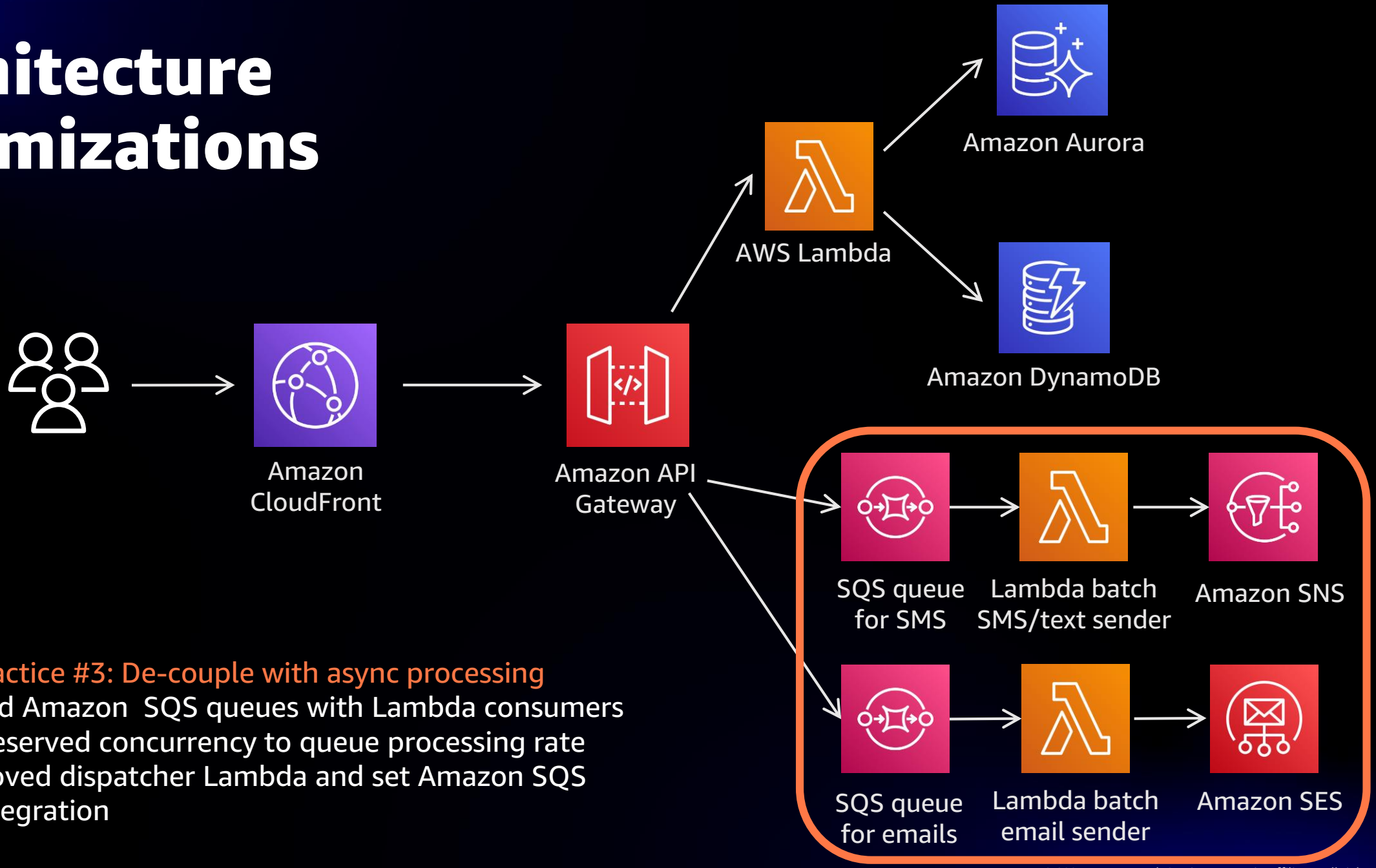
Architecture optimizations



Best practice #3: De-couple with async processing

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



Best practice #3: De-couple with async processing

- Added Amazon SQS queues with Lambda consumers
- Set reserved concurrency to queue processing rate
- Removed dispatcher Lambda and set Amazon SQS as integration

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

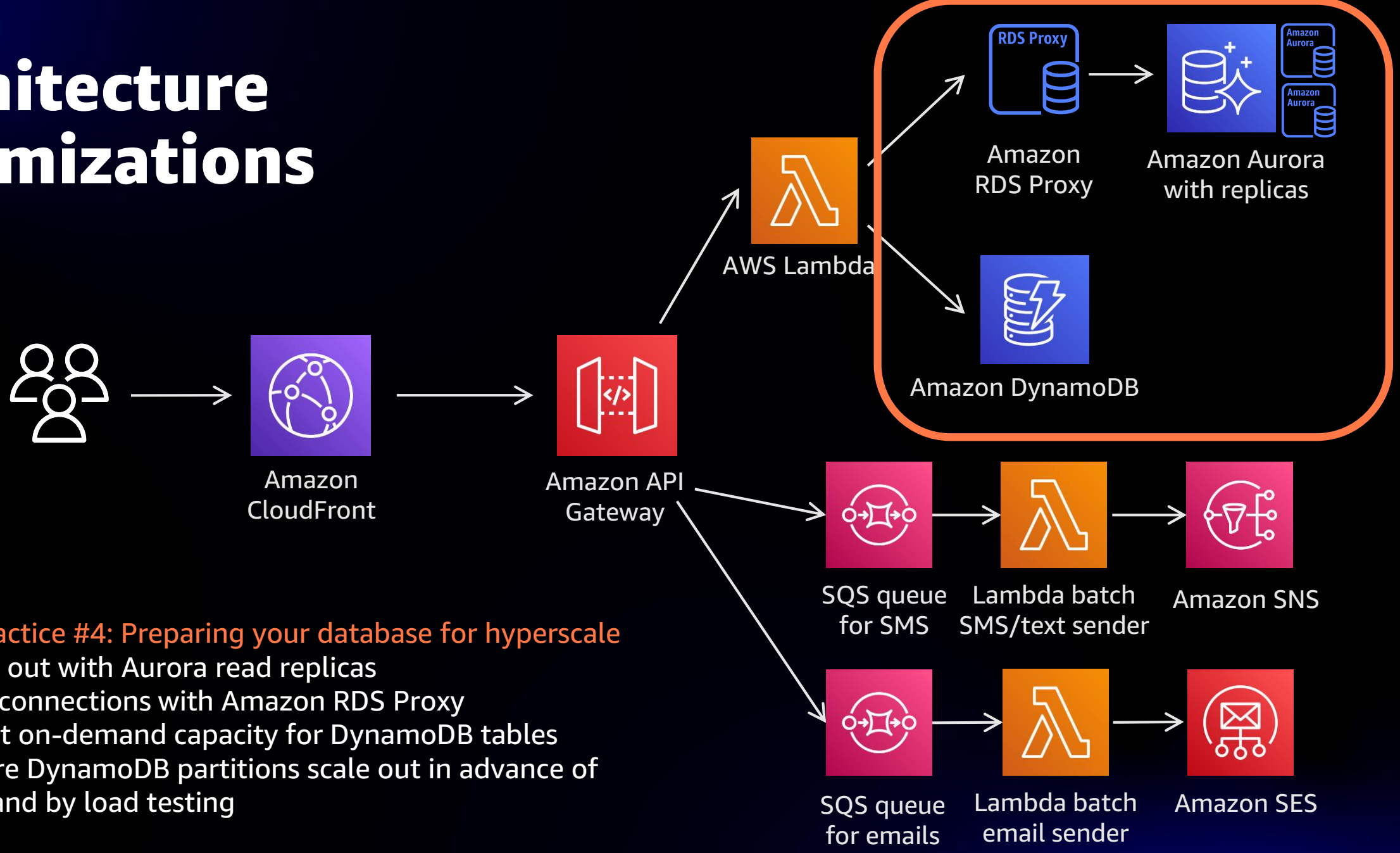
Architecture optimizations



Best practice #4: Preparing your database for hyperscale

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



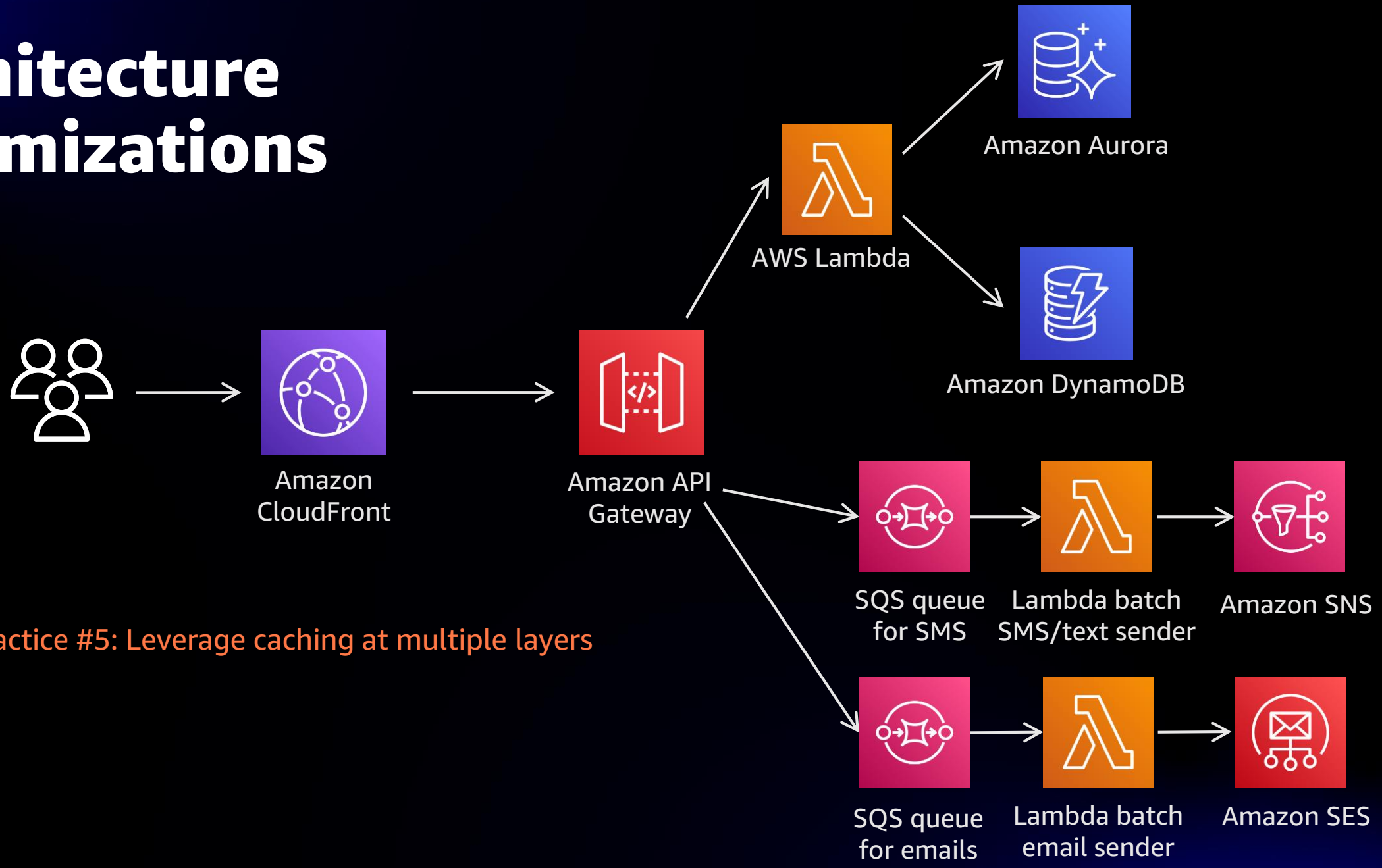
Best practice #4: Preparing your database for hyperscale

- Scale out with Aurora read replicas
- Pool connections with Amazon RDS Proxy
- Select on-demand capacity for DynamoDB tables
- Ensure DynamoDB partitions scale out in advance of demand by load testing

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

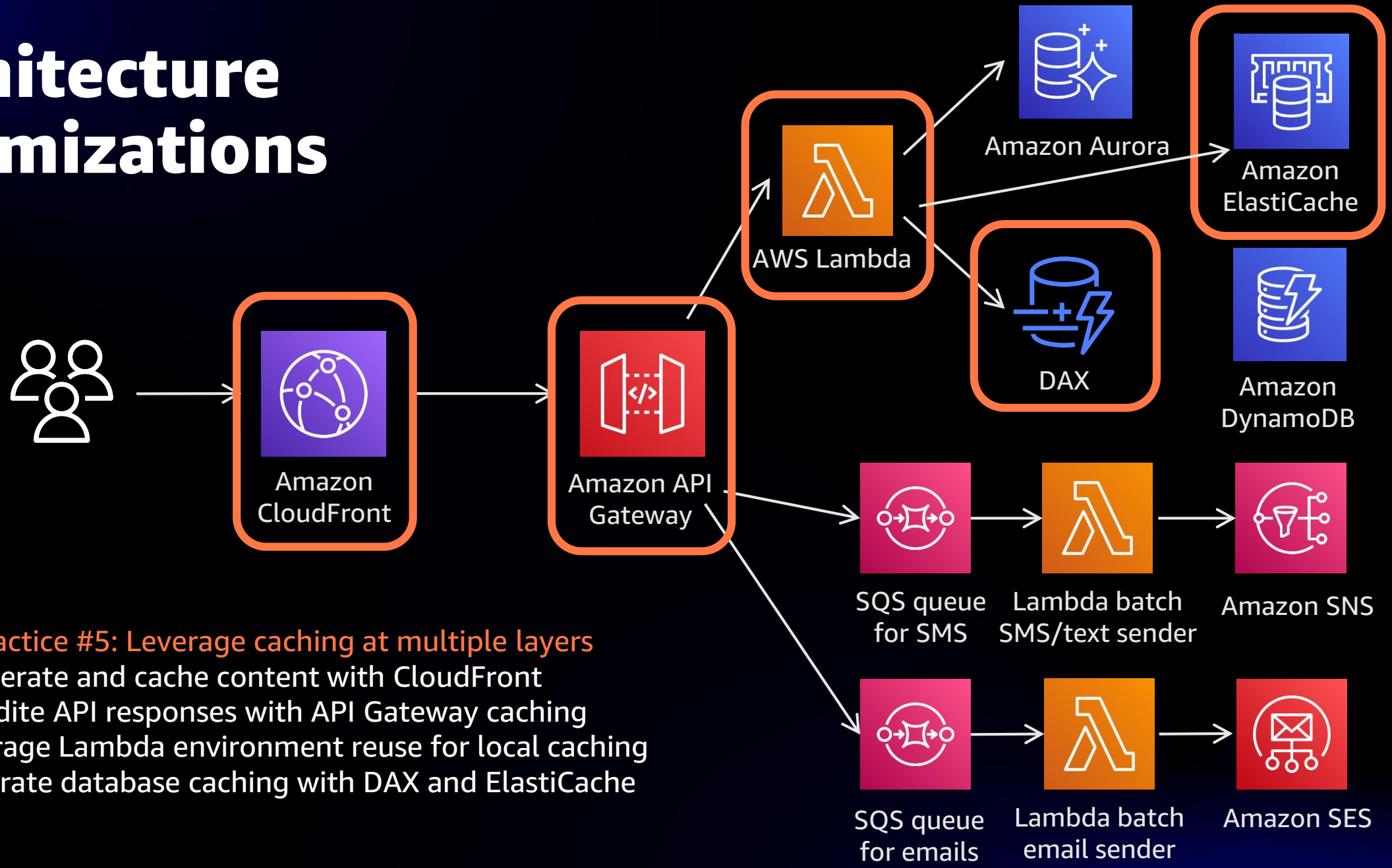
Architecture optimizations



Best practice #5: Leverage caching at multiple layers

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



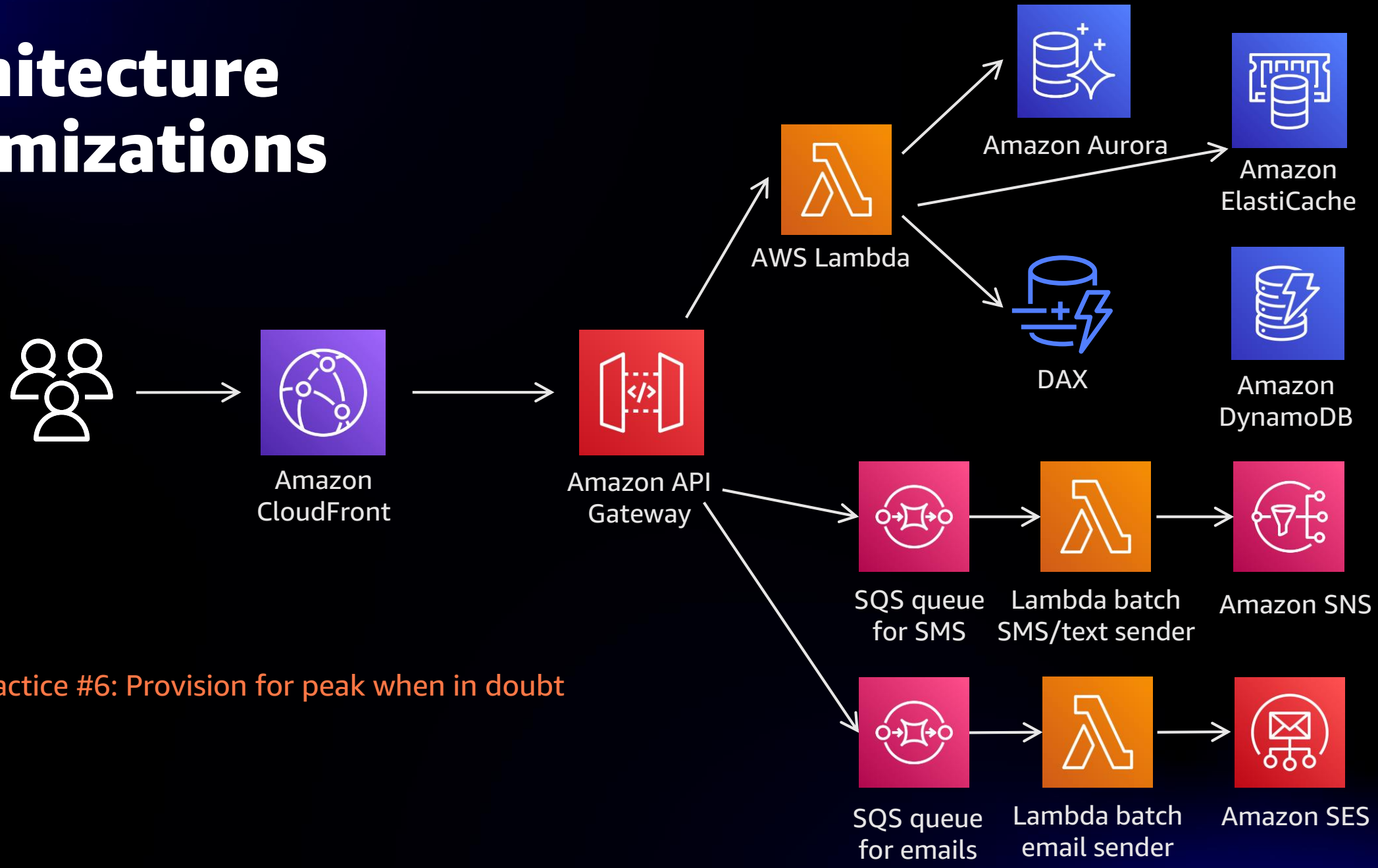
Best practice #5: Leverage caching at multiple layers

- Accelerate and cache content with CloudFront
- Expedite API responses with API Gateway caching
- Leverage Lambda environment reuse for local caching
- Integrate database caching with DAX and ElastiCache

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

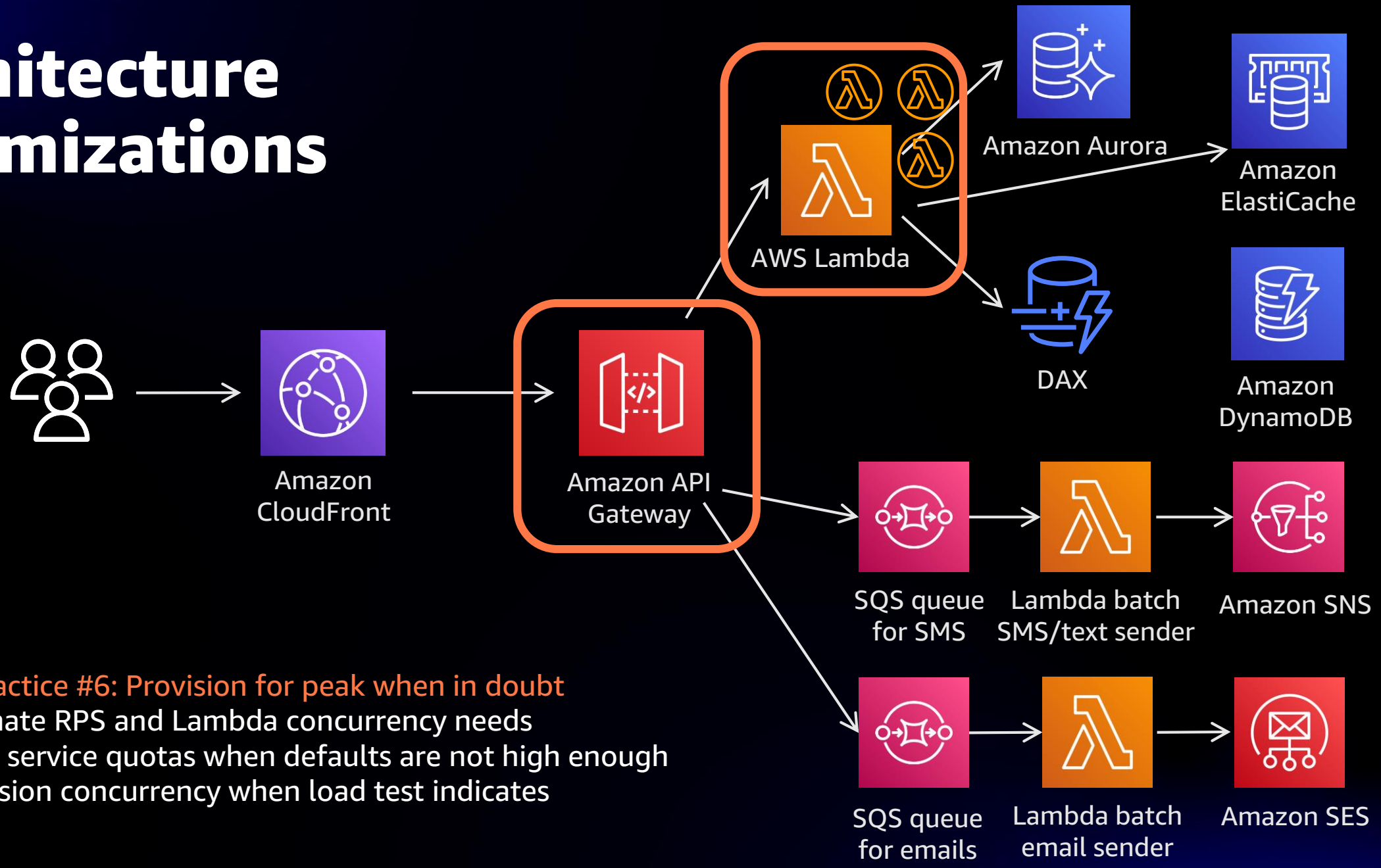
Architecture optimizations



Best practice #6: Provision for peak when in doubt

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations

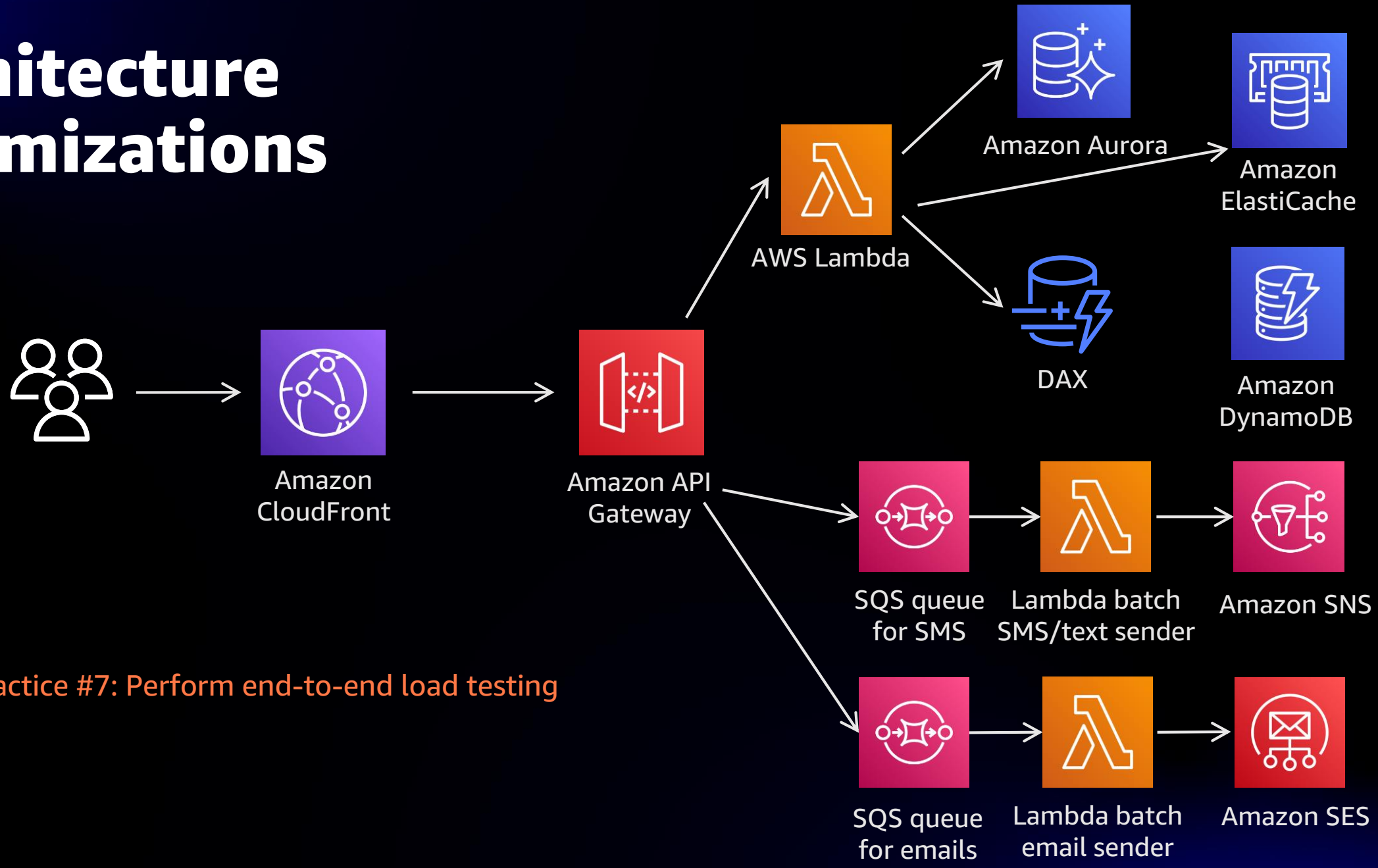


Best practice #6: Provision for peak when in doubt

- Estimate RPS and Lambda concurrency needs
- Raise service quotas when defaults are not high enough
- Provision concurrency when load test indicates

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

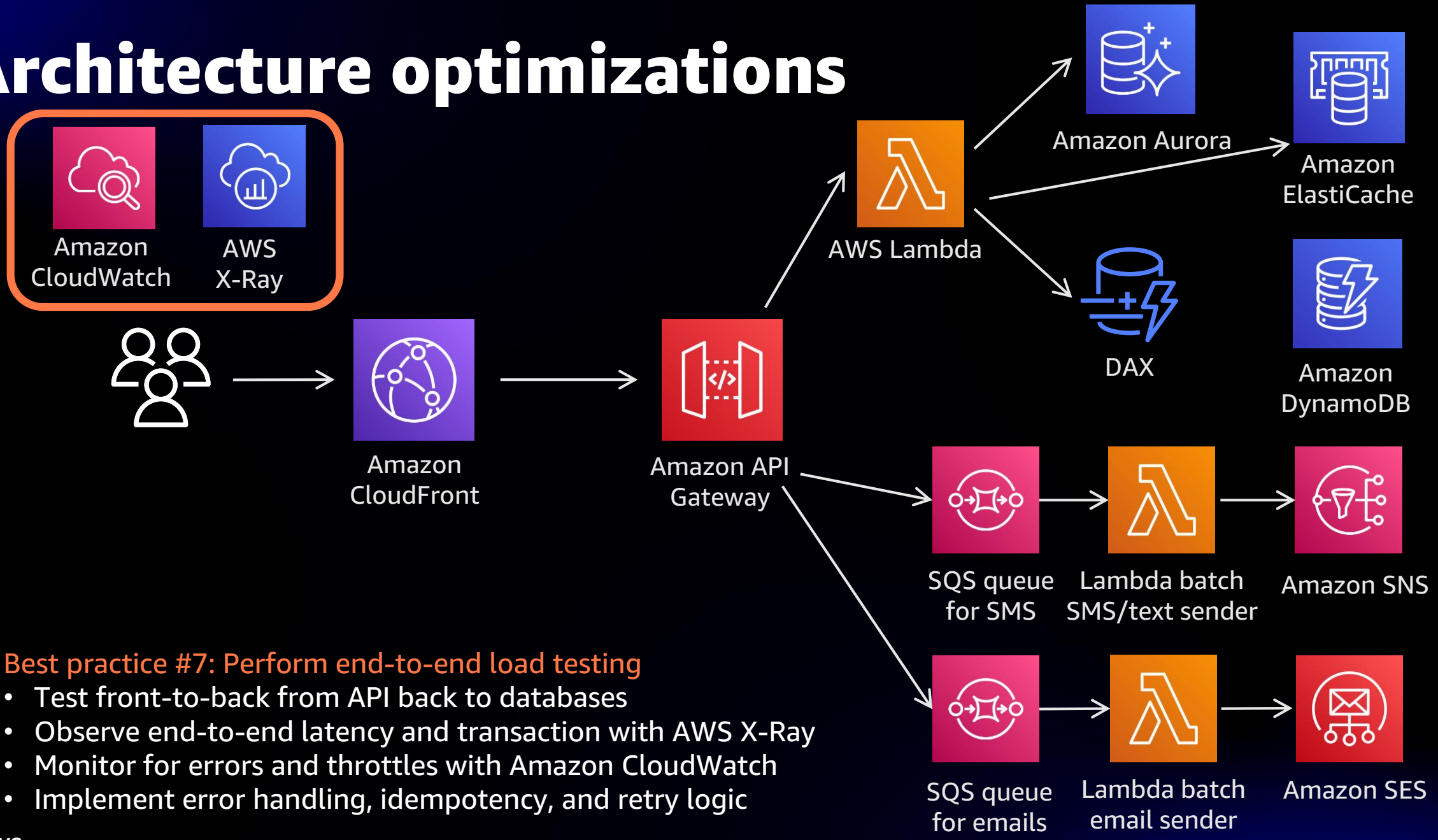
Architecture optimizations



Best practice #7: Perform end-to-end load testing

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



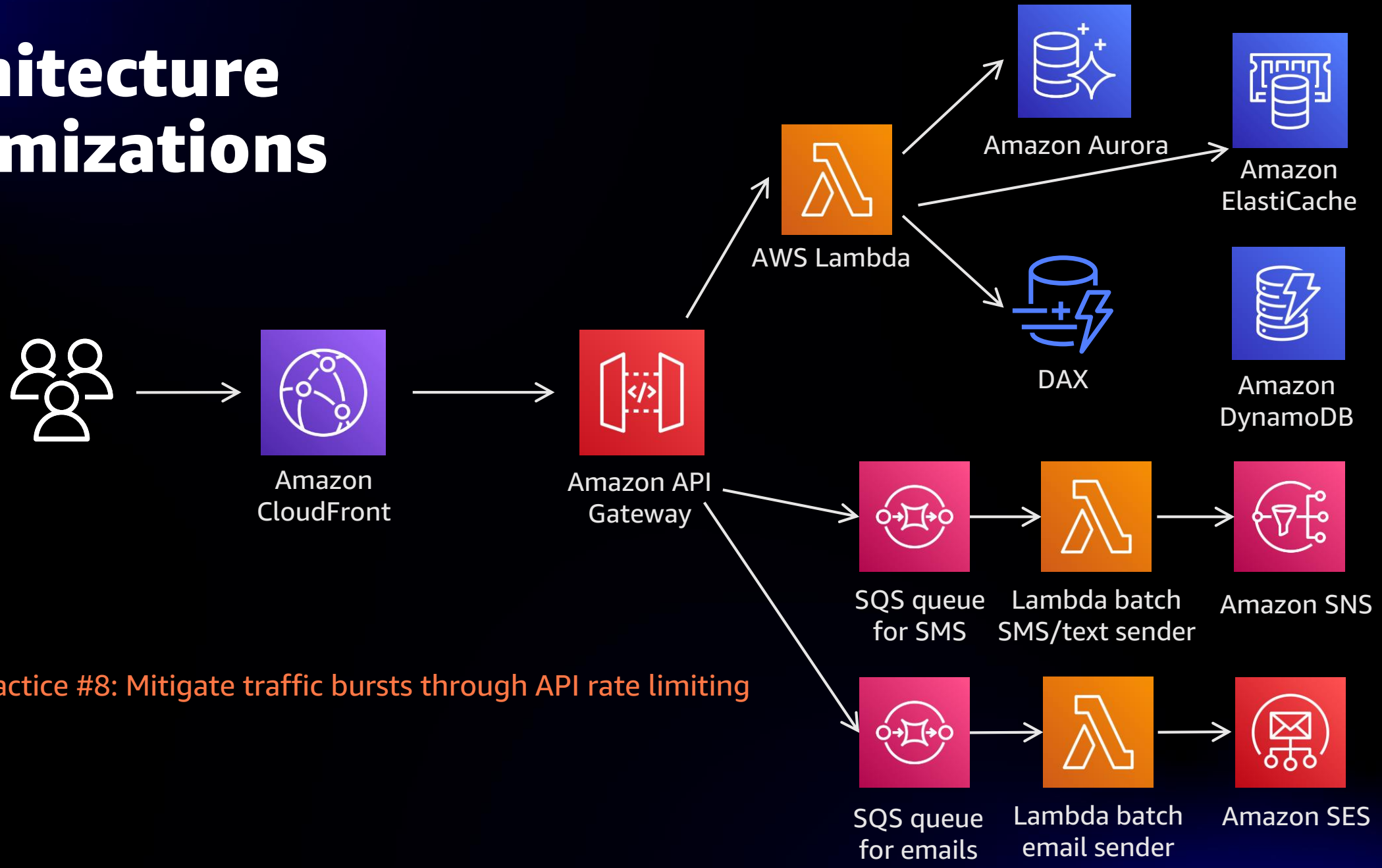
Best practice #7: Perform end-to-end load testing

- Test front-to-back from API back to databases
- Observe end-to-end latency and transaction with AWS X-Ray
- Monitor for errors and throttles with Amazon CloudWatch
- Implement error handling, idempotency, and retry logic

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations

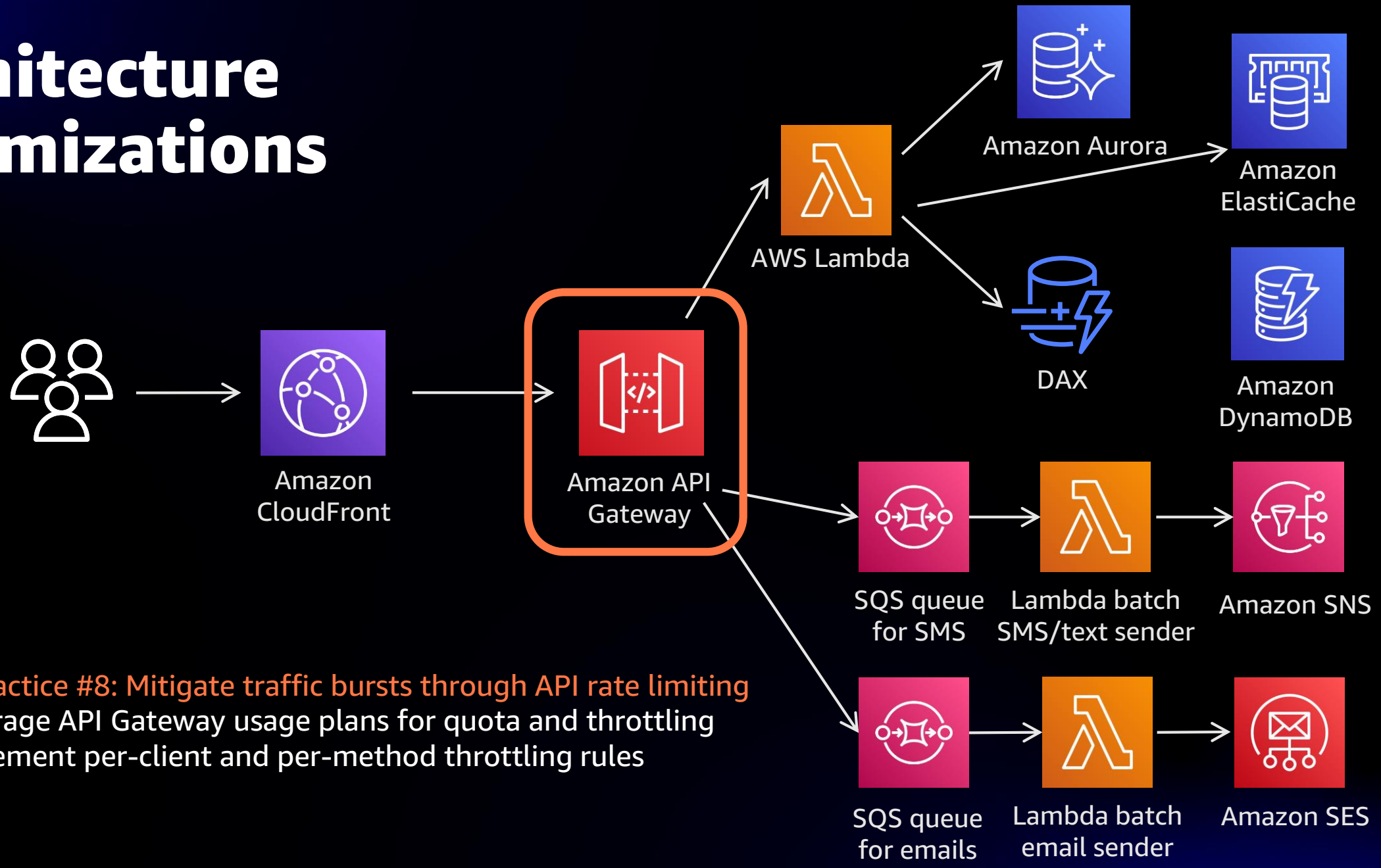


Best practice #8: Mitigate traffic bursts through API rate limiting

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



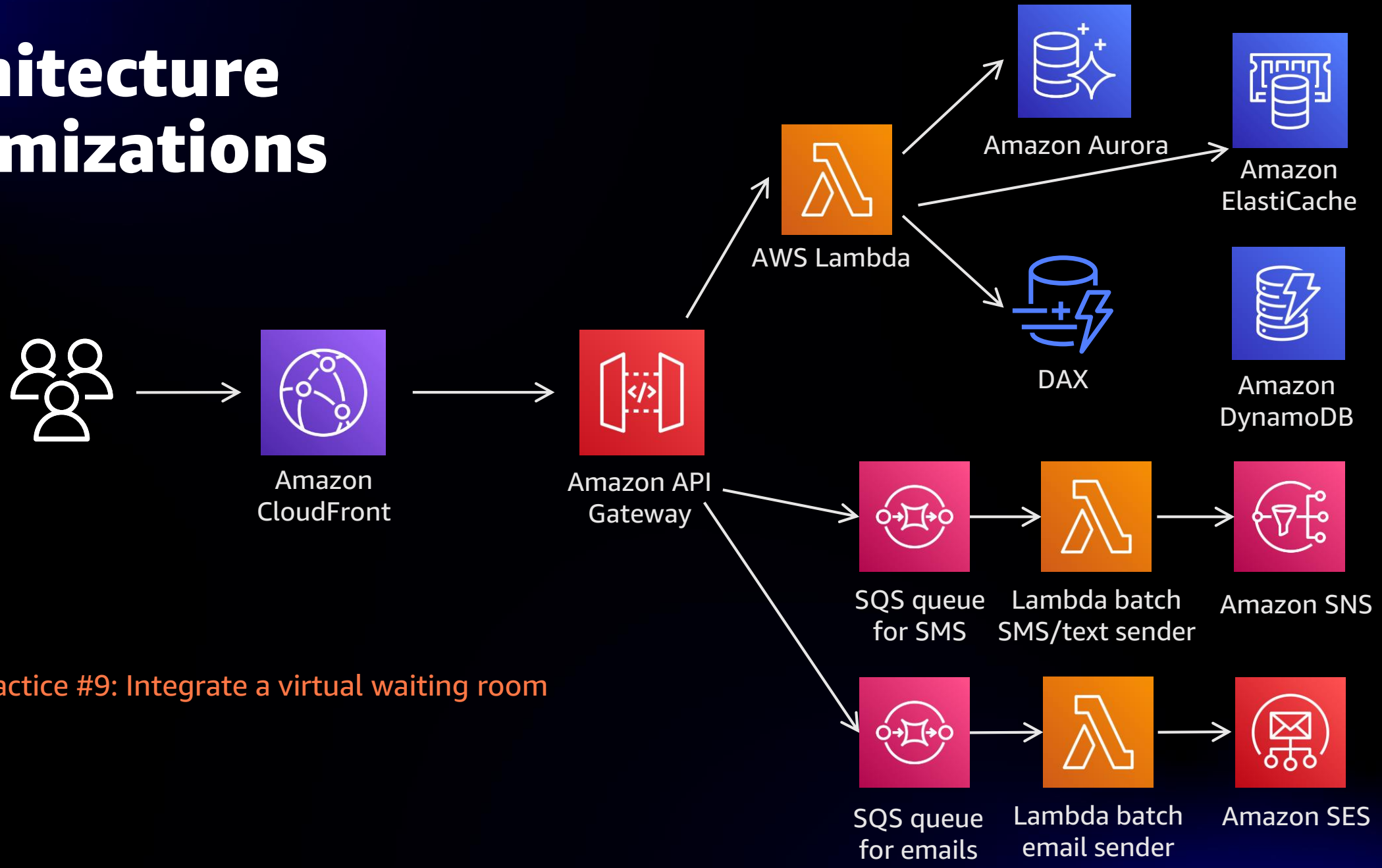
Best practice #8: Mitigate traffic bursts through API rate limiting

- Leverage API Gateway usage plans for quota and throttling
- Implement per-client and per-method throttling rules

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

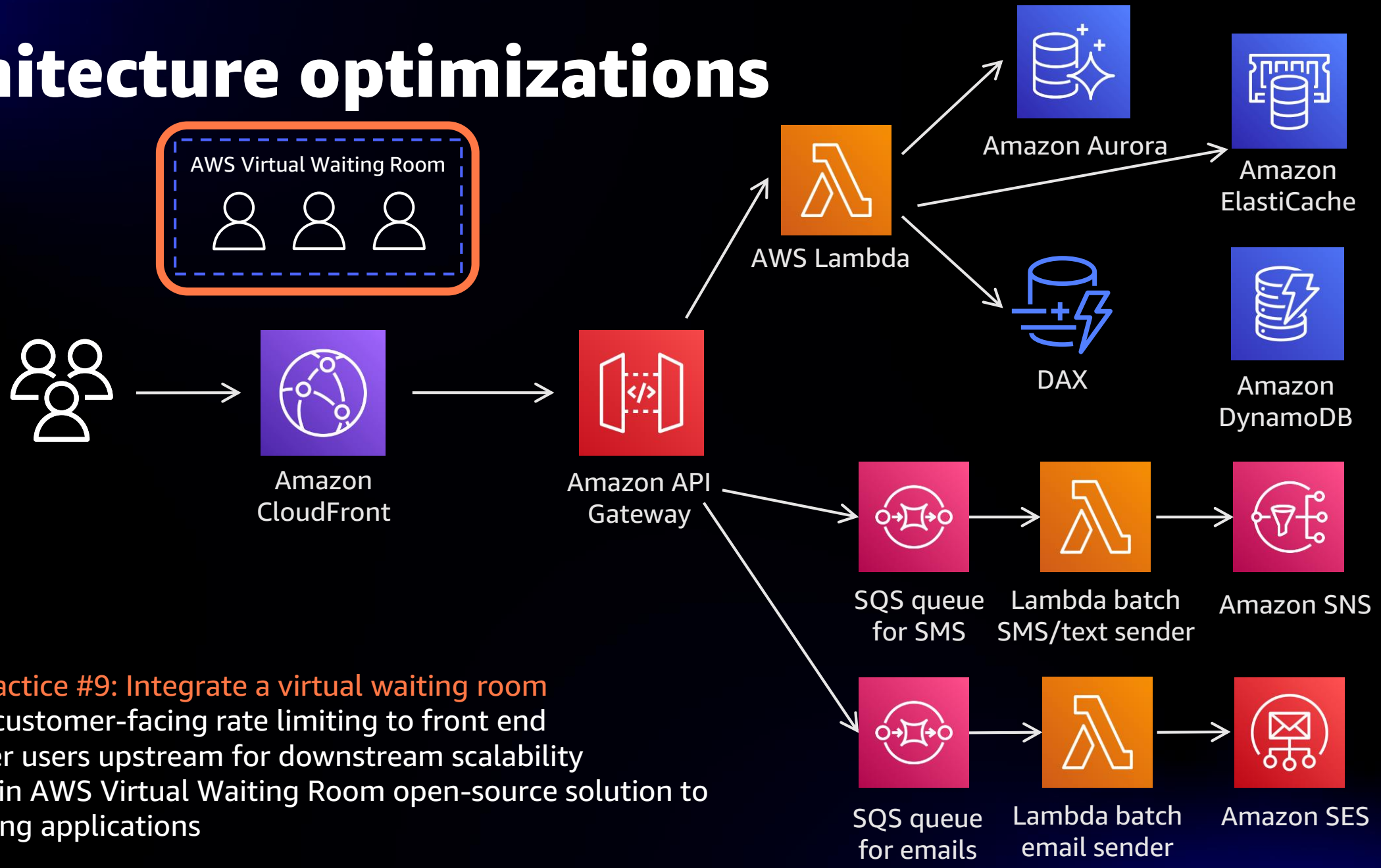
Architecture optimizations



Best practice #9: Integrate a virtual waiting room

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Architecture optimizations



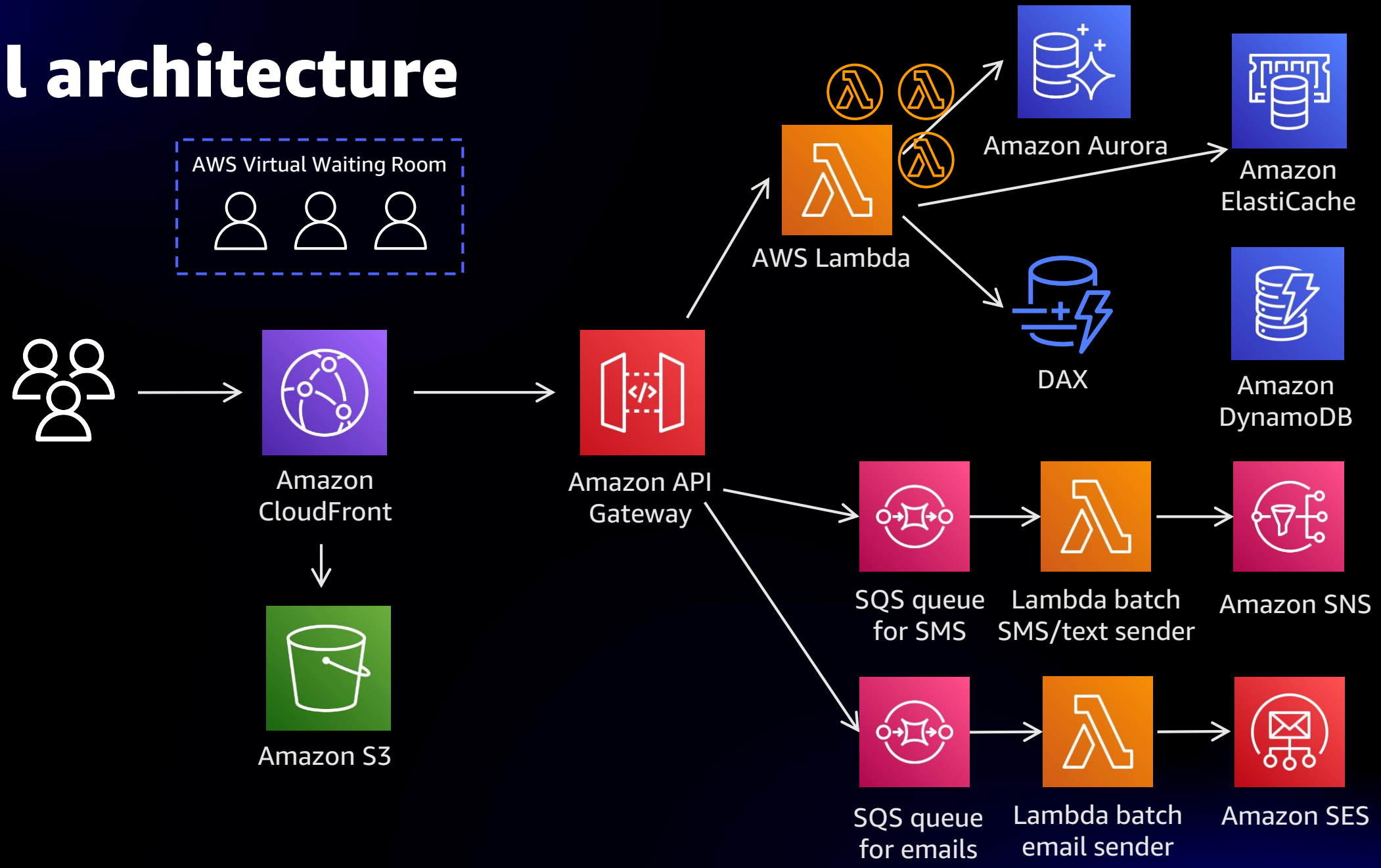
Best practice #9: Integrate a virtual waiting room

- Add customer-facing rate limiting to front end
- Buffer users upstream for downstream scalability
- Plug in AWS Virtual Waiting Room open-source solution to existing applications

aws

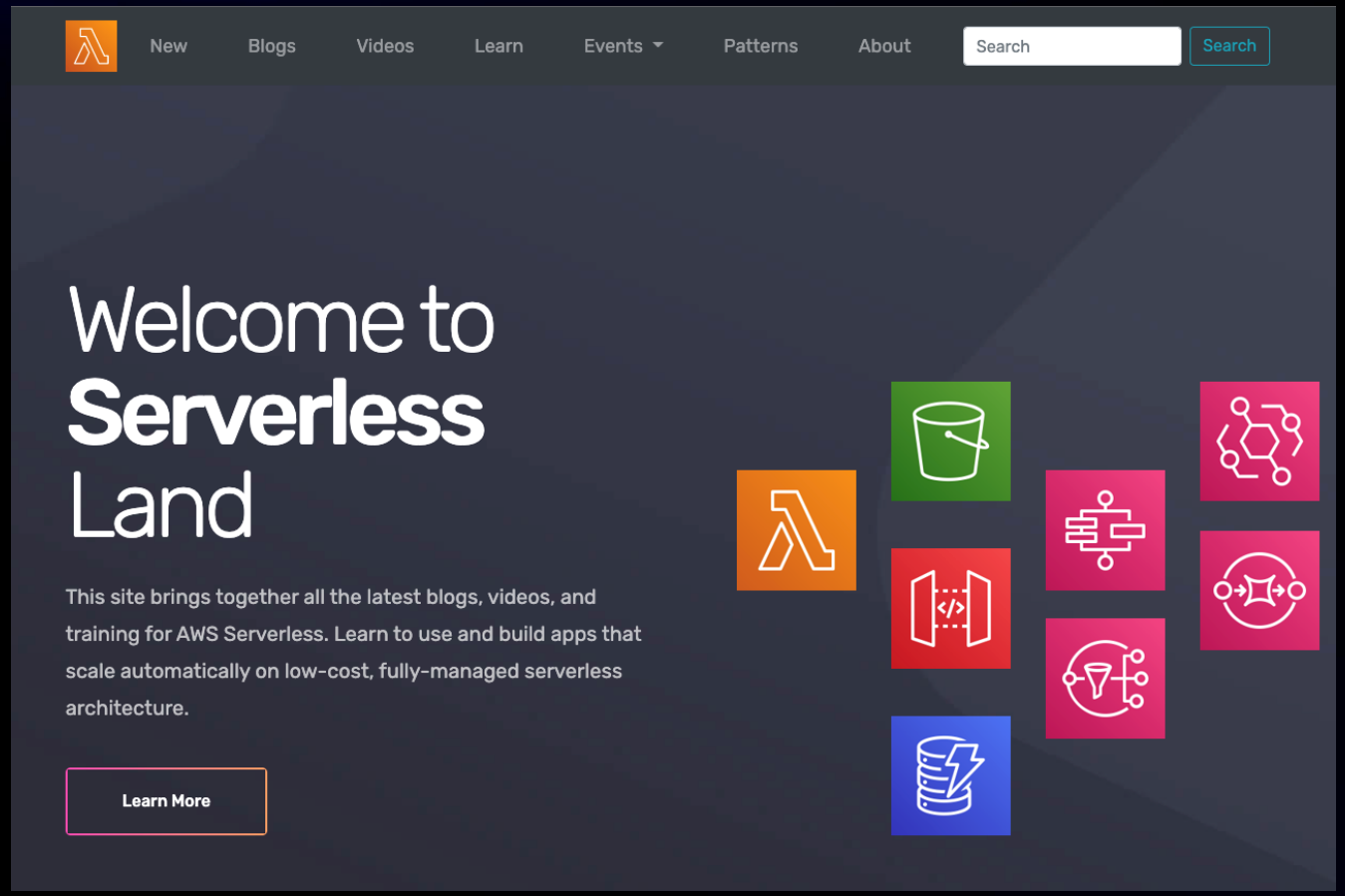
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Final architecture



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Want to learn more?



https://serverlessland.com/

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Justin Pirtle

@justinpirtle

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.