
AWS
re:Invent
DECEMBER 2 – 6, 2024 | LAS VEGAS, NV

SVS322

Optimize multi-tenant serverless architectures for agility and scale

Anand Bilgaiyan
(he/him)
Sr. Specialist PSA – Enterprise Transformation
Amazon Web Services

Nishant Dhiman
(he/him)
Sr. Solutions Architect
Amazon Web Services

Agenda

01 Multi-tenant challenges

02 Serverless for multi-tenant

03 Use case walkthrough

04 Best practices

Unique challenges

Tenant isolation

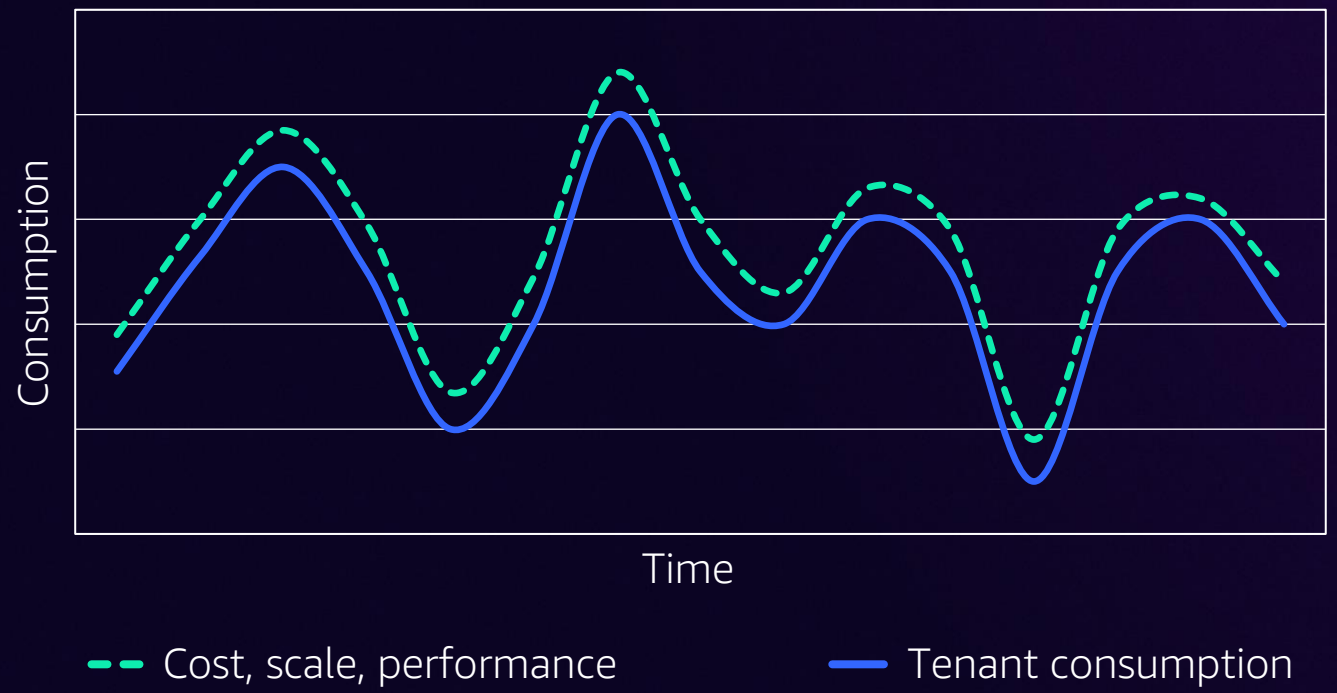
Scaling integration

Noisy neighbor

Cost attribution

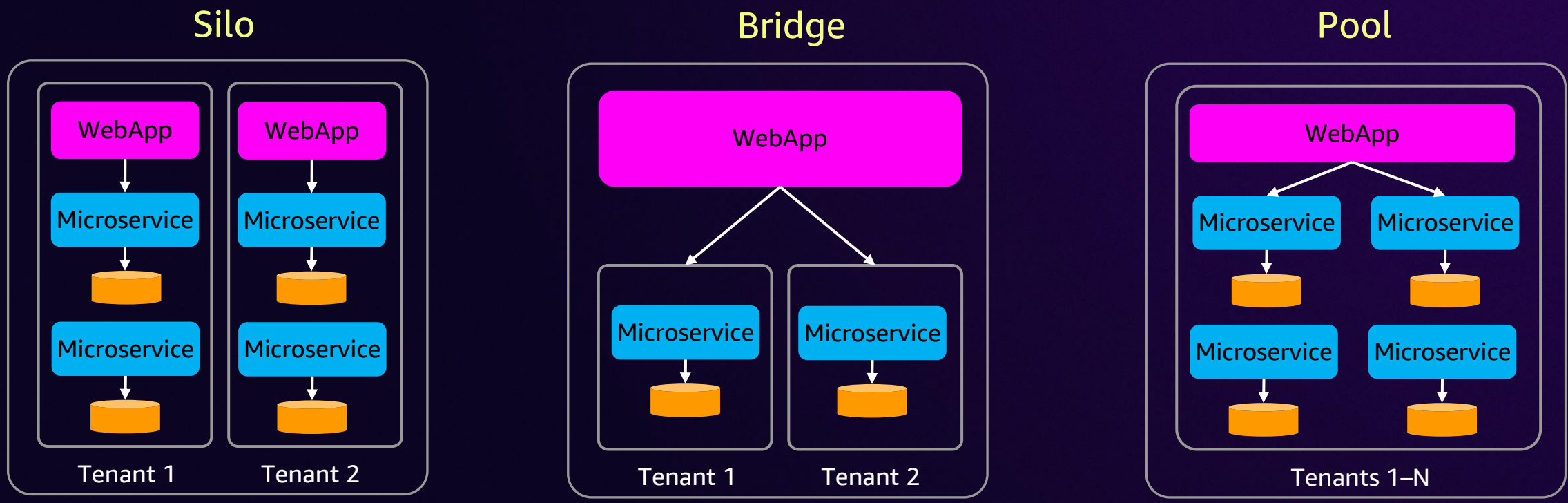
Inefficient resource utilization

Serverless-powered multi-tenant approach



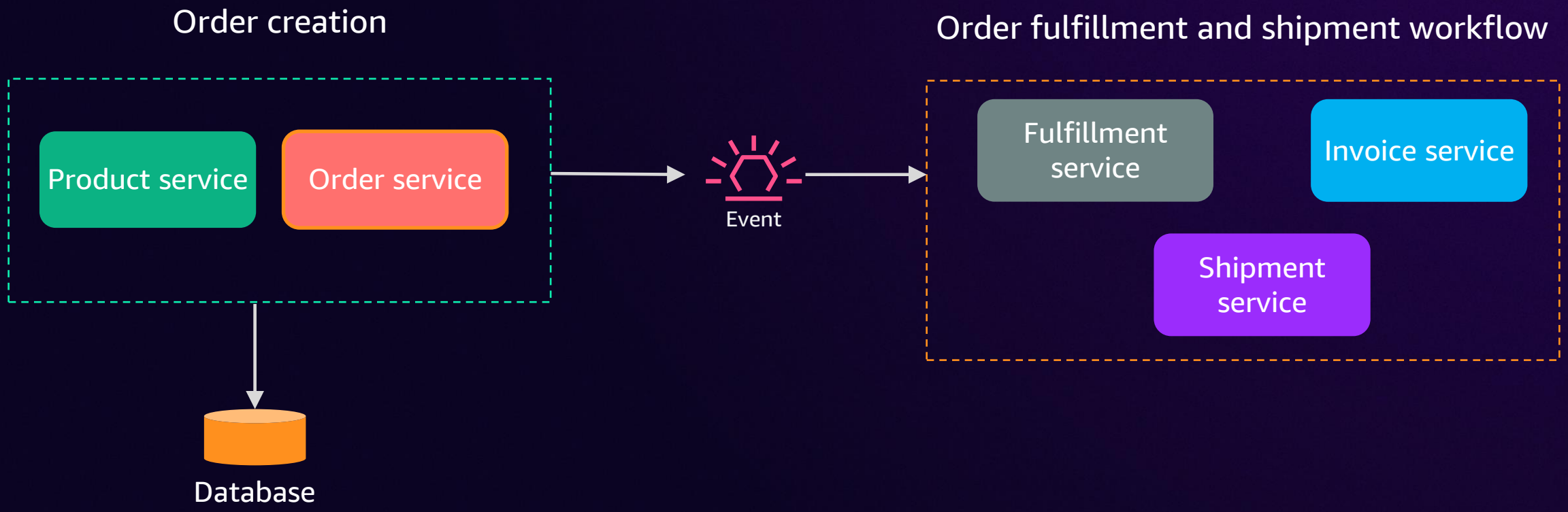
- No infrastructure provisioning, no management
- Automatic scaling
- Pay for value
- Highly available and secure

Multi-tenant models



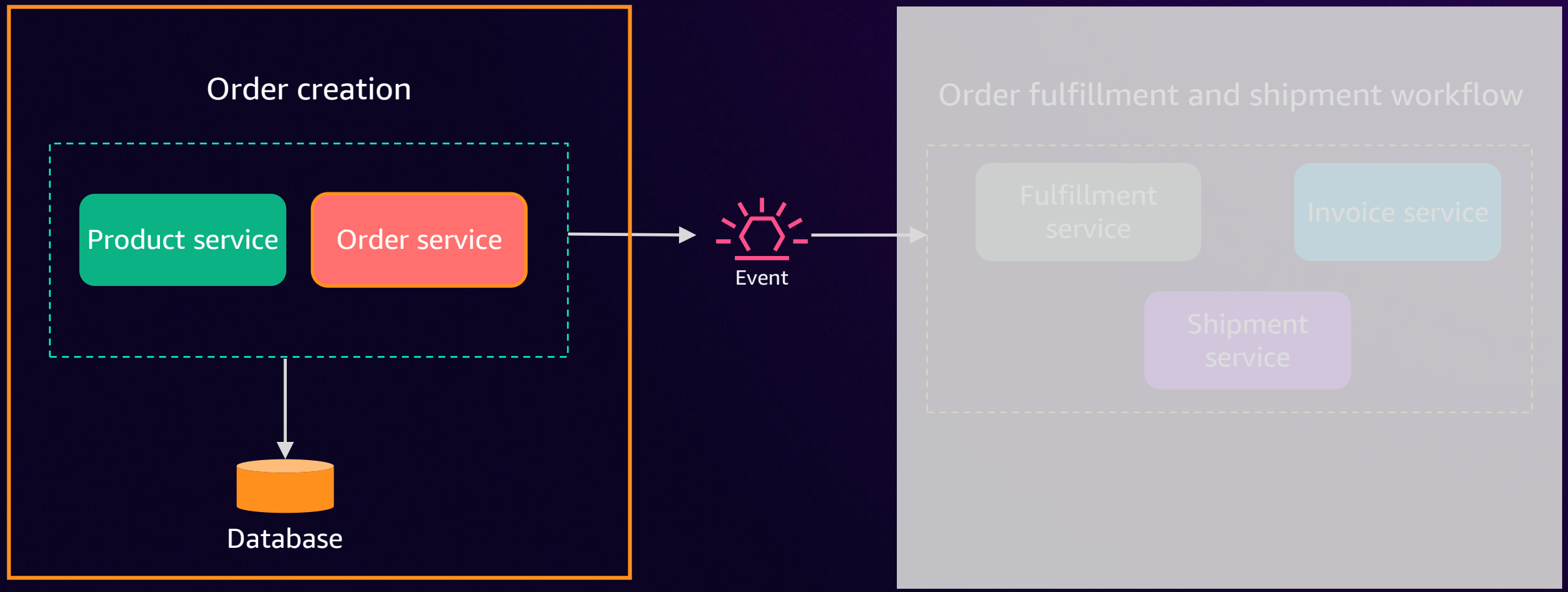
Find the right mix of scale, cost, and experience

OUR ECOMMERCE USE CASE

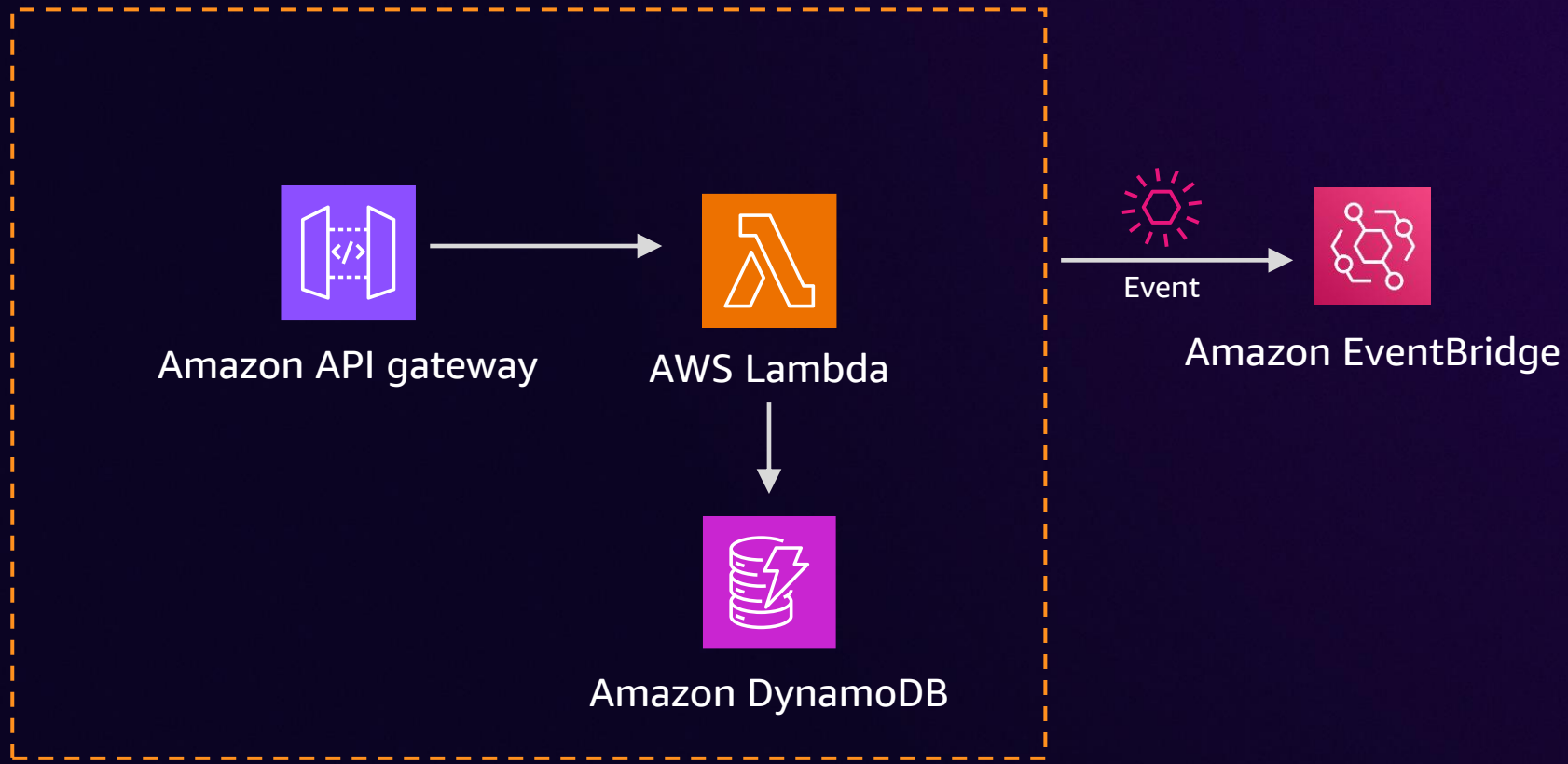


Find the right mix of scale, cost, and experience

OUR ECOMMERCE USE CASE

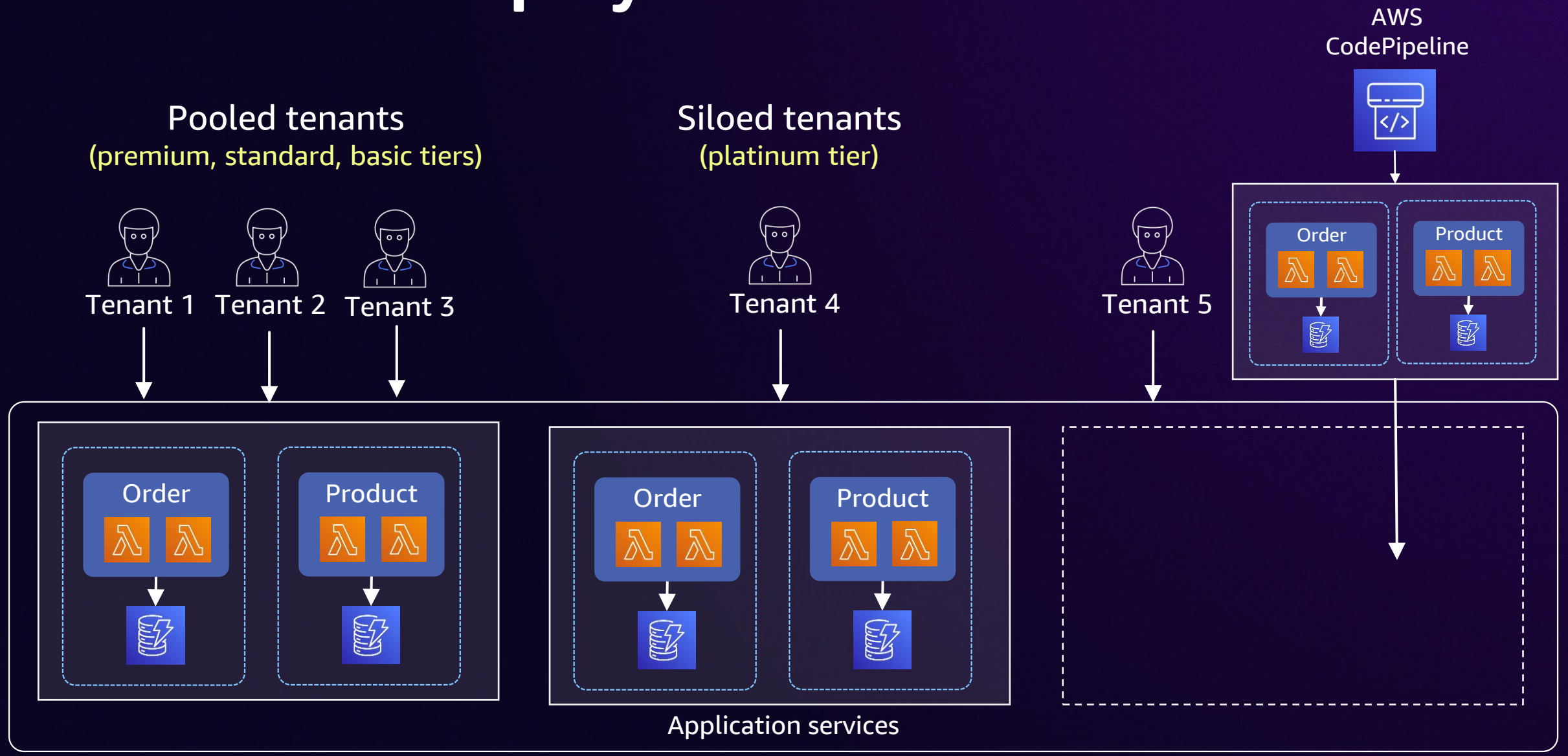


Serverless implementation: Order service



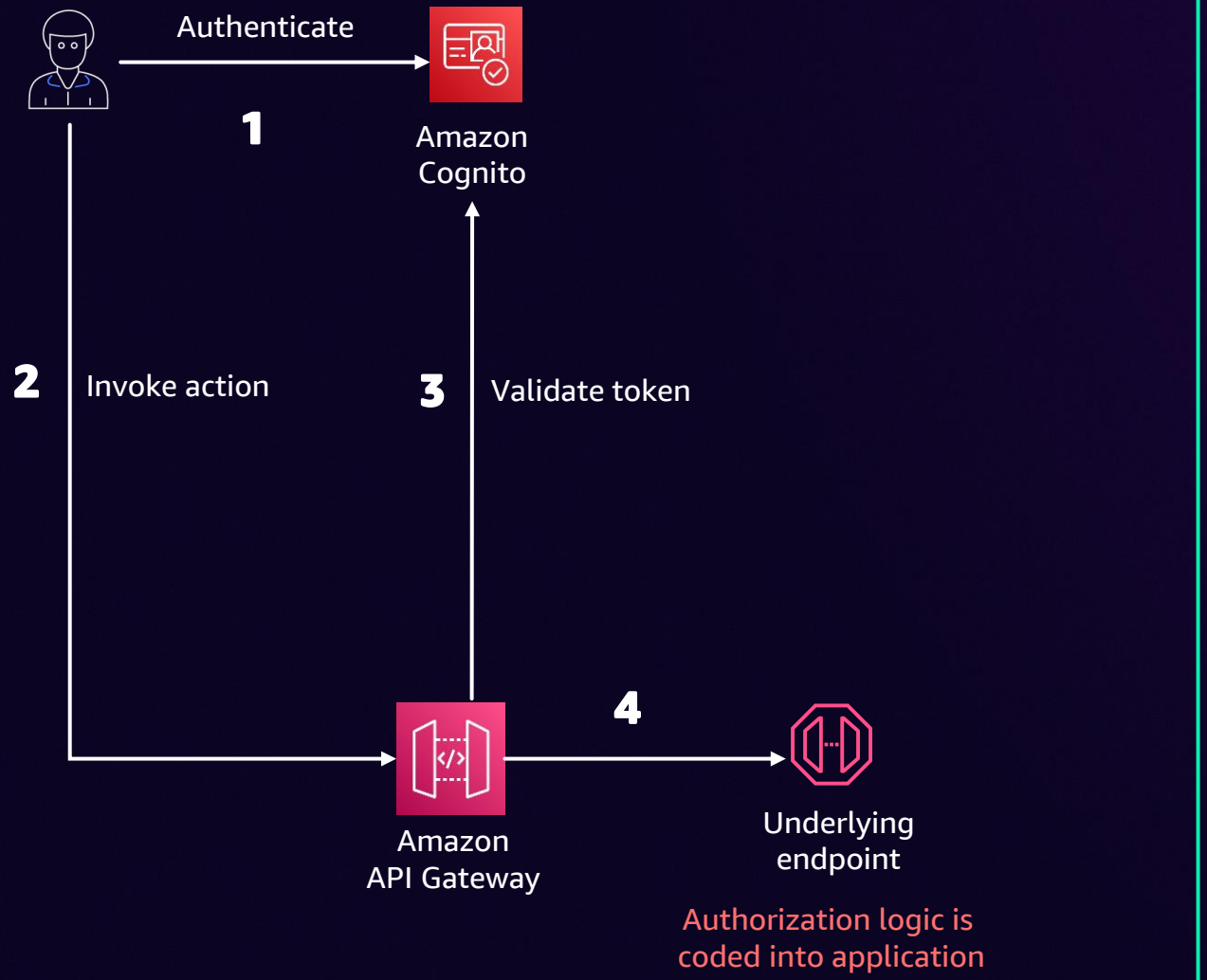
Multi-tenancy with AWS Lambda

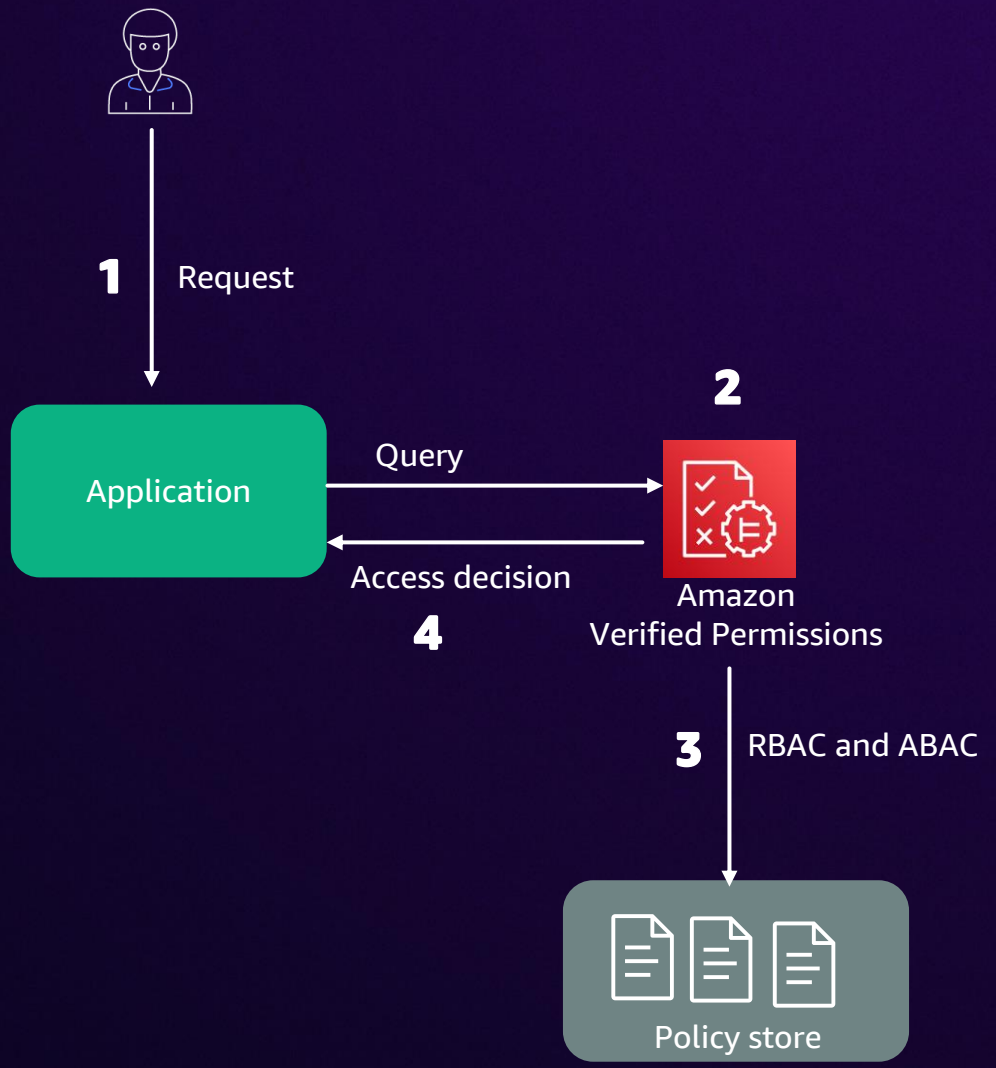
AWS Lambda deployment models



Data isolation

AWS Lambda isolation models: Amazon Verified Permissions





Declarative way: Cedar policy language

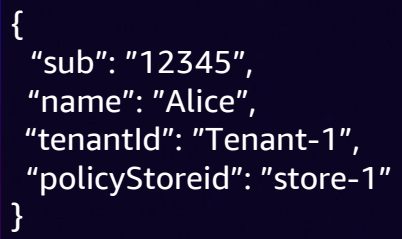
Verified Permissions policy example

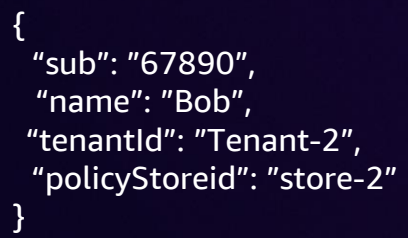
```
permit (
    principal in MultitenantApp::Role::"allAccessRole",

    action in [
        MultitenantApp::Action::"CreateOrder",
        MultitenantApp::Action::"ViewOrder"
    ],
    resource
)
when  {
    resource in principal.Tenant &&
    principal.account_lockout_flag == false &&
    context.uses_mfa == true
};
```

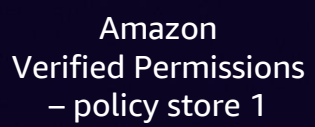
Implement **app permissions** as **Cedar** policies

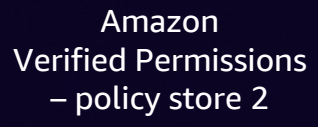
- Cedar is **easy to read and write**
- Separate policies are **easy to audit and change**
- Cedar validator **helps prevent policy mistakes**

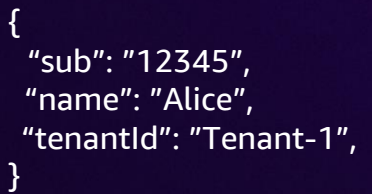
Verified Permissions: Multi-tenant approach

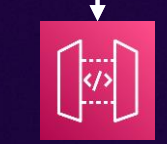


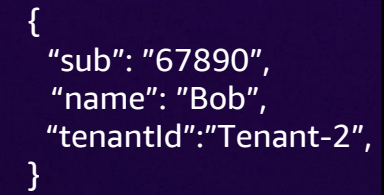


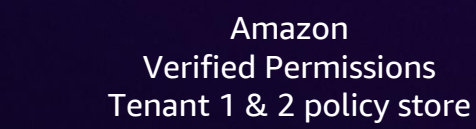






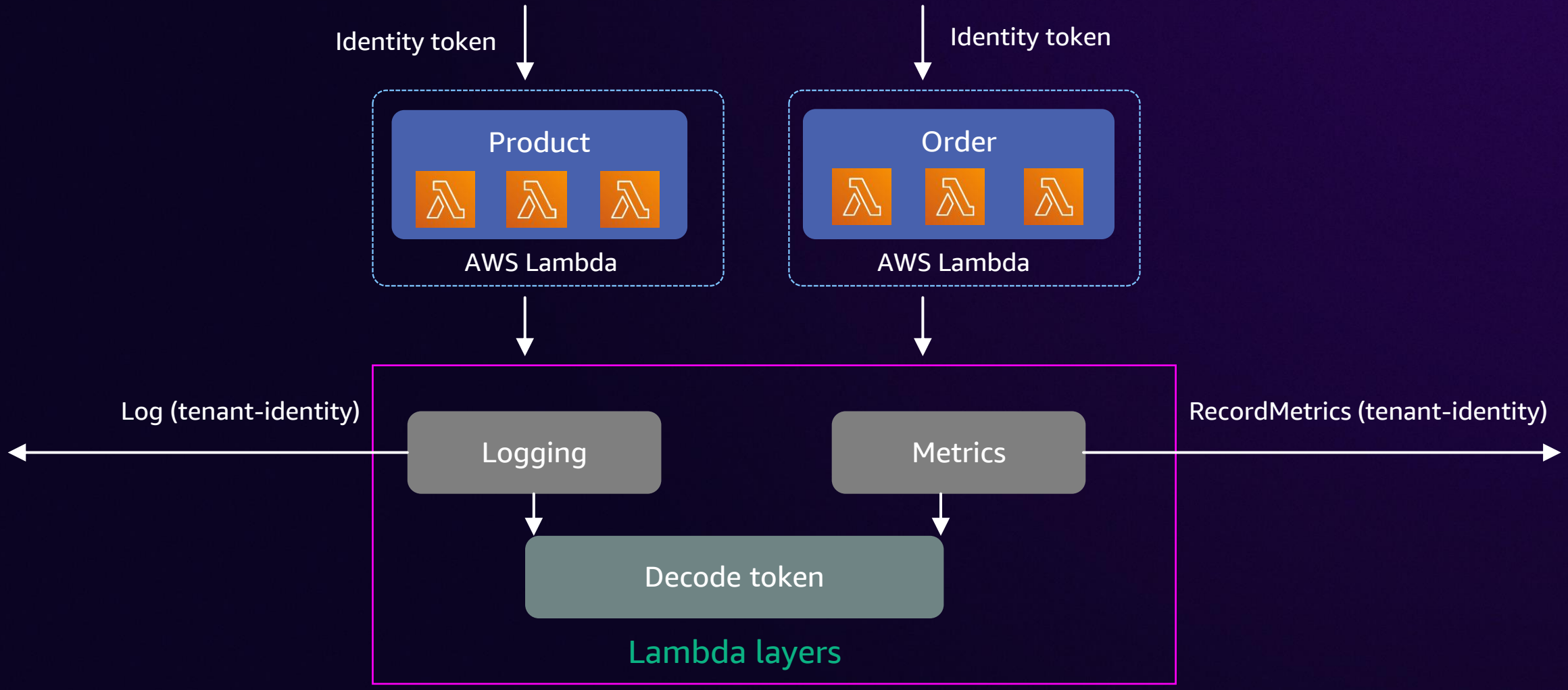






Cost per tenant

AWS Lambda layers for centralized logging and metrics collection



Serverless: Cost per tenant

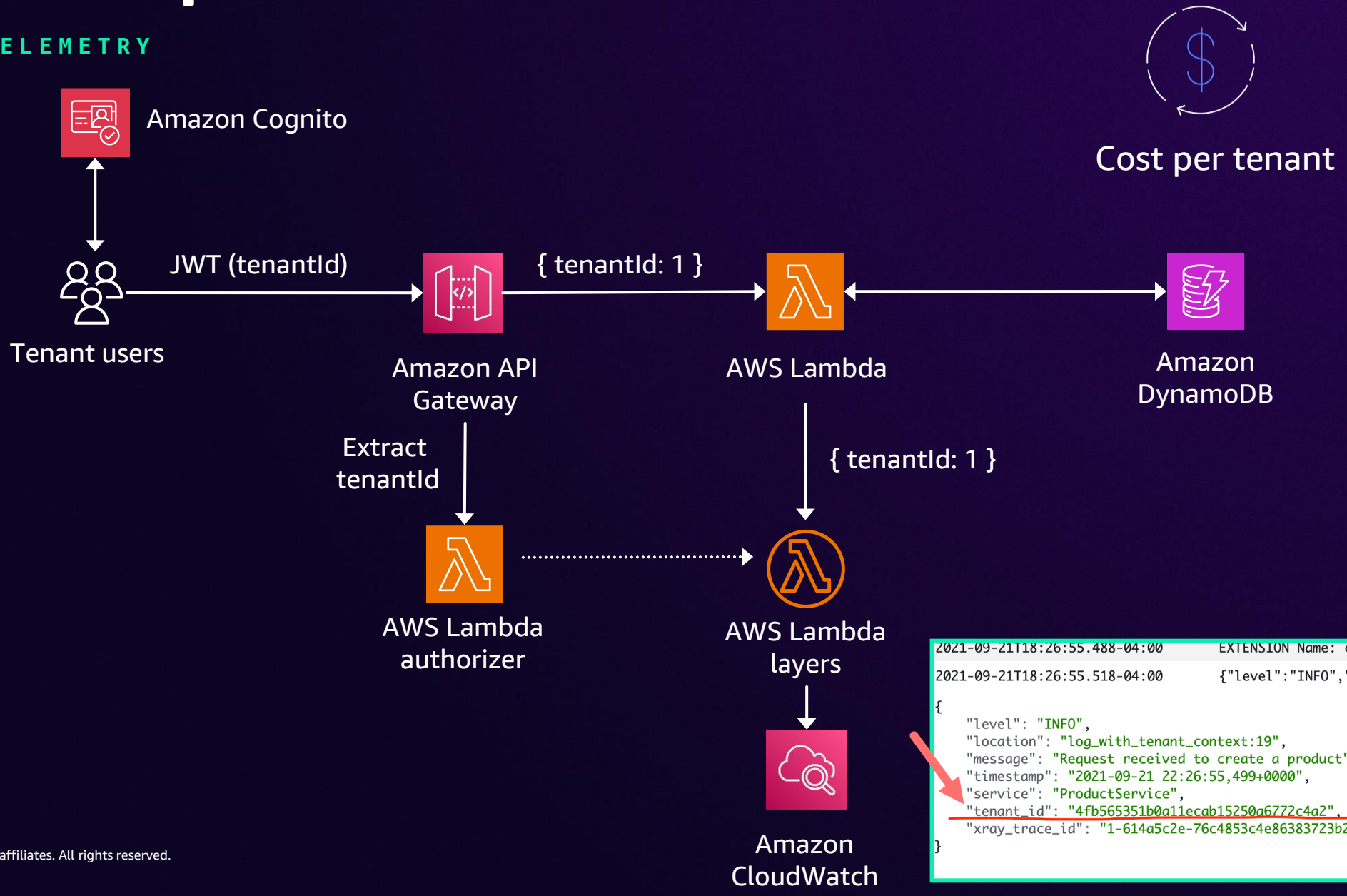
CAPTURE AND STORE TELEMETRY

Amazon API Gateway

- Number of requests

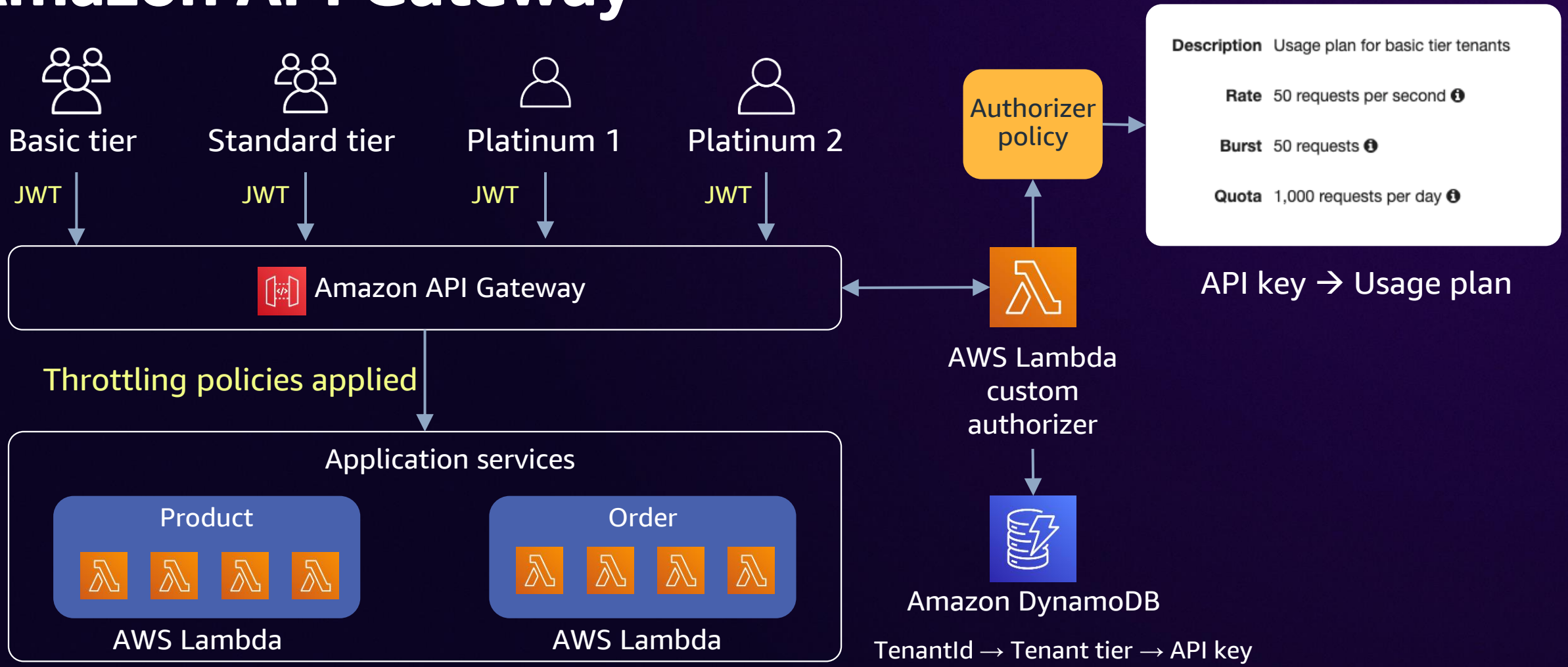
Lambda

- Duration
- Number of invocations



Noisy neighbor

Tier-based throttling with AWS Lambda and Amazon API Gateway



Scaling multi-tenant applications with AWS Lambda

Account concurrency

Maximum concurrency in a given Region across all functions

1,000 in all Regions

This can be increased

Function quota

Scaling rate per function, in each Region

1,000 new concurrent executions every 10 seconds

This can NOT be increased

What if you need more scale?

AWS Lambda concurrency controls

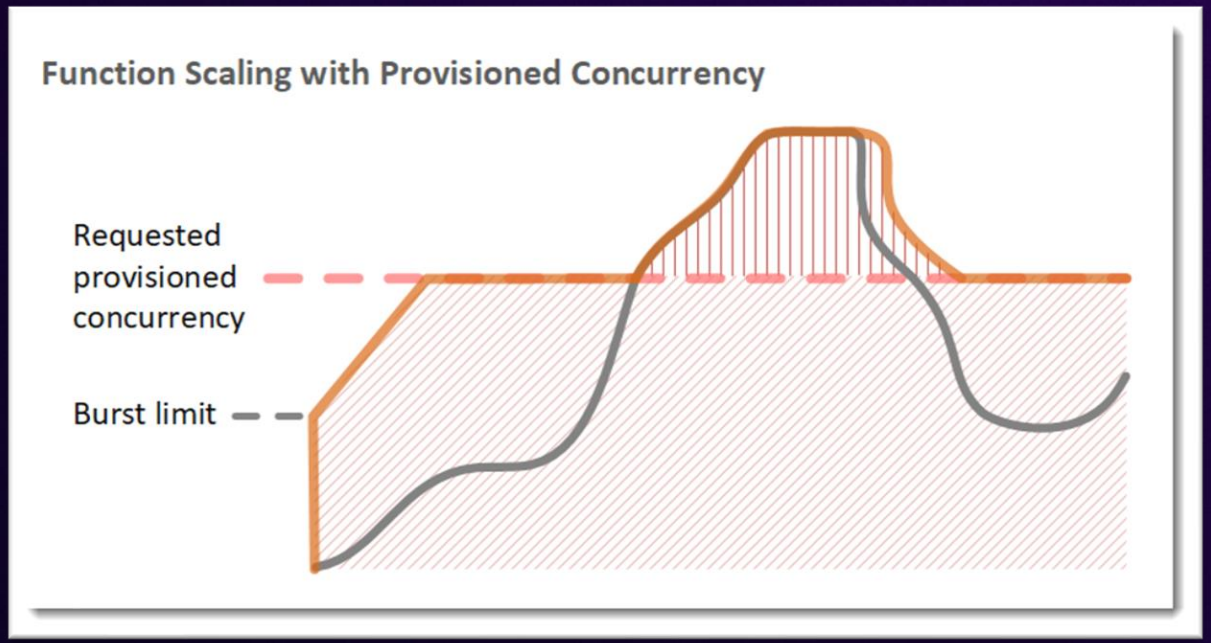
Provisioned concurrency

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



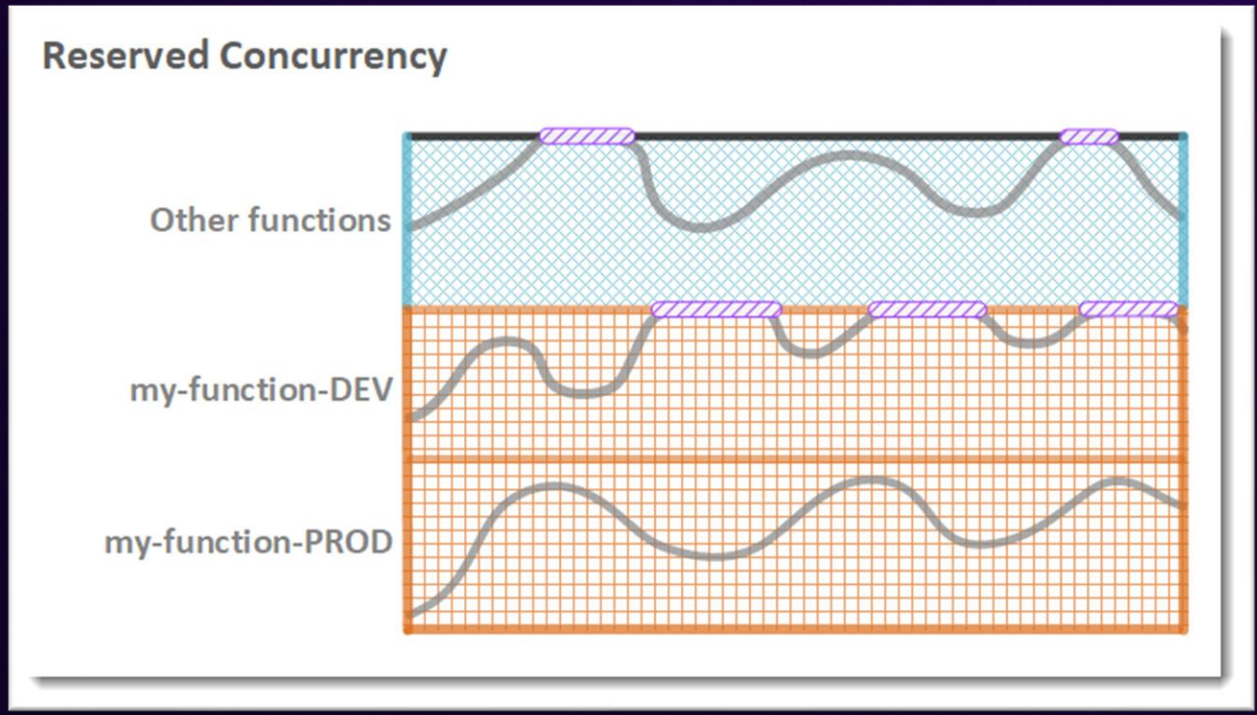
What if scaling too fast may overwhelm downstream systems?

AWS Lambda concurrency controls

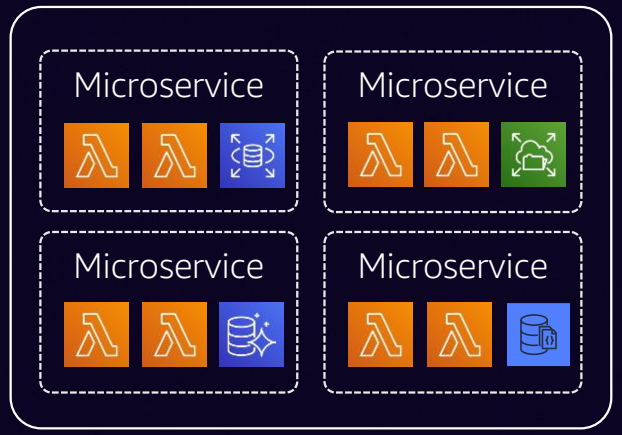
Reserved concurrency

Sets ceiling on maximum number of execution environments – **upper limit on maximum concurrency** for a given function

Also, **reserves that concurrency from the account's quota**

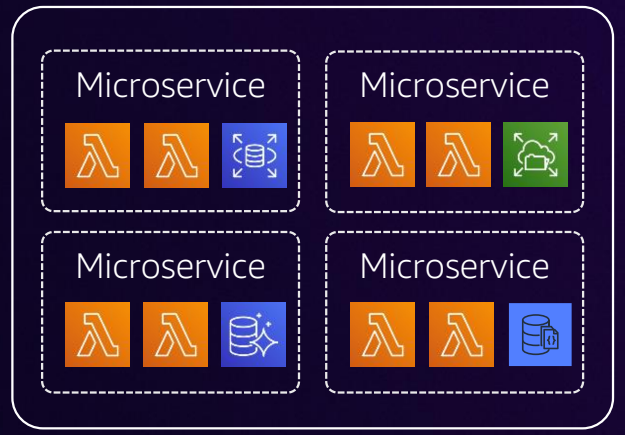


Noisy neighbor and AWS Lambda concurrency



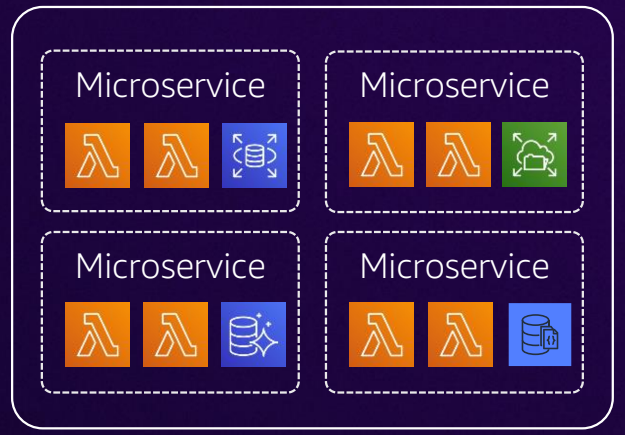
Basic tier

Reserve concurrency = 100



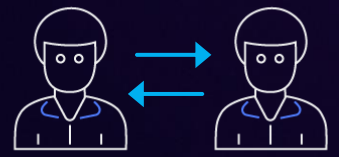
Advanced tier

Reserve concurrency = 300



Premium tier

Reserve concurrency = All unreserved



Noisy neighbor

Multi-tenancy best practices with AWS Lambda

Use **Amazon Verified Permissions** for isolation

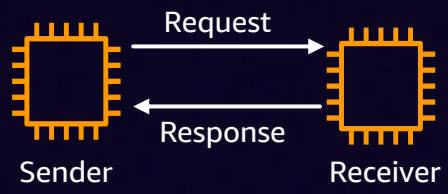
Control scale and noisy neighbors with AWS Lambda **reserved concurrency** and Amazon API Gateway **usage plan**

Leverage **Lambda layers** for logs and metrics consumption to determine cost per tenant

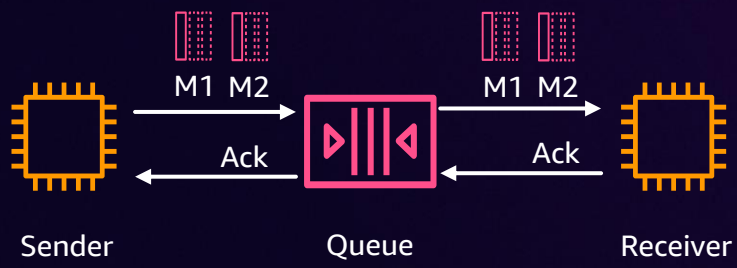
Integration use cases in multi-tenant solutions

Integration patterns

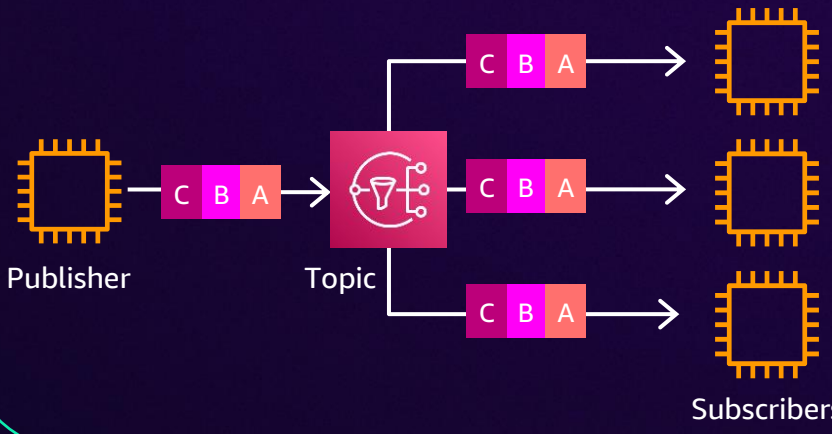
Synchronous



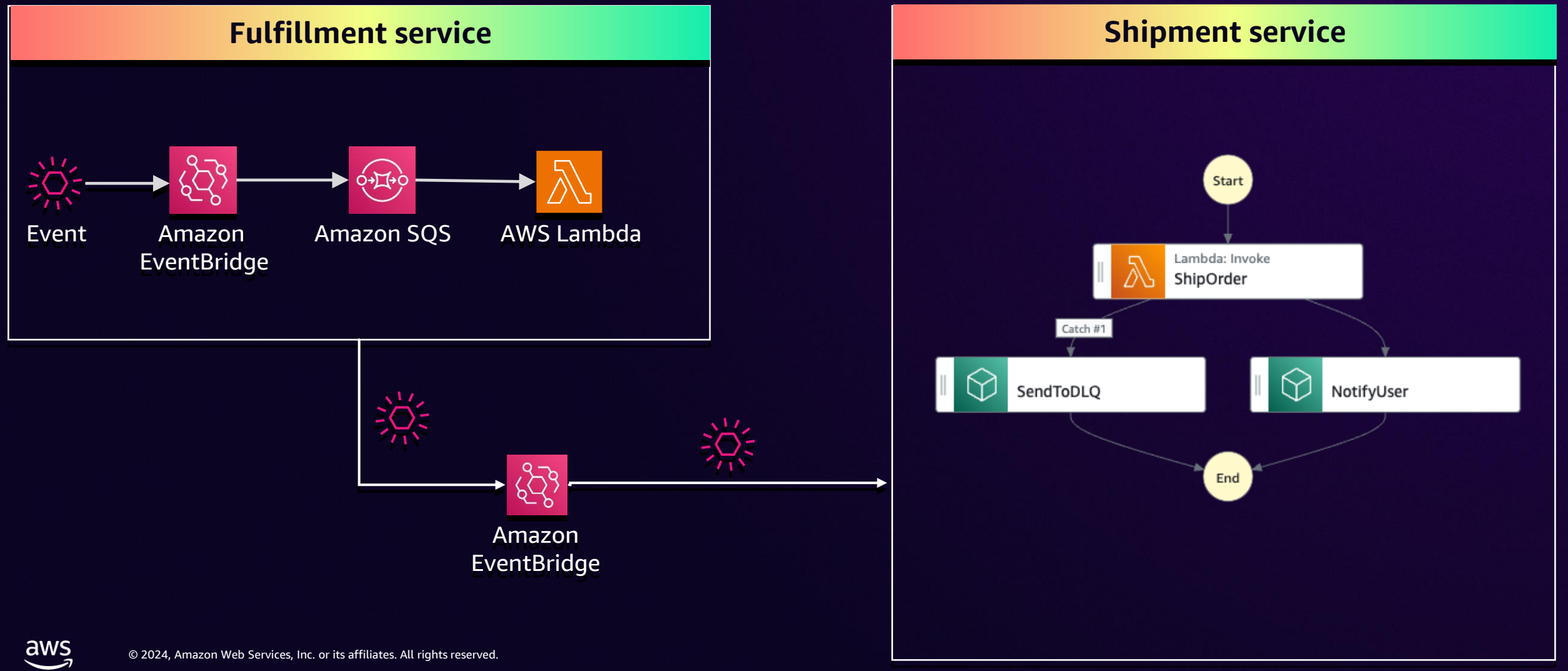
Asynchronous



Publish/subscribe



Serverless implementation: Fulfillment and shipment services



Common questions

Should I share my resources across tenants?

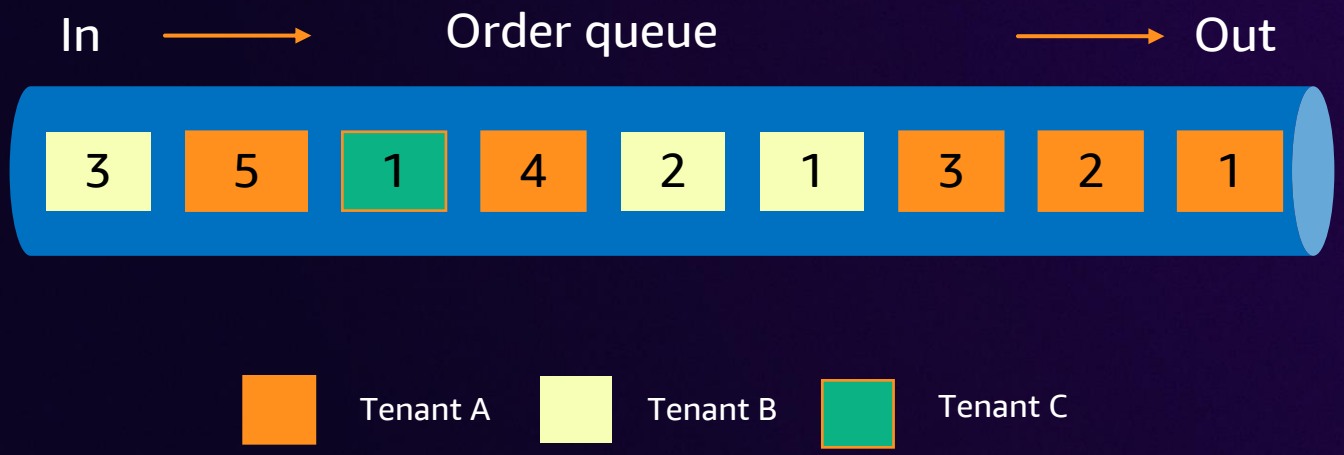
Operational complexity?

What if one tenant produces more messages?

What about data isolation?

How to handle errors?

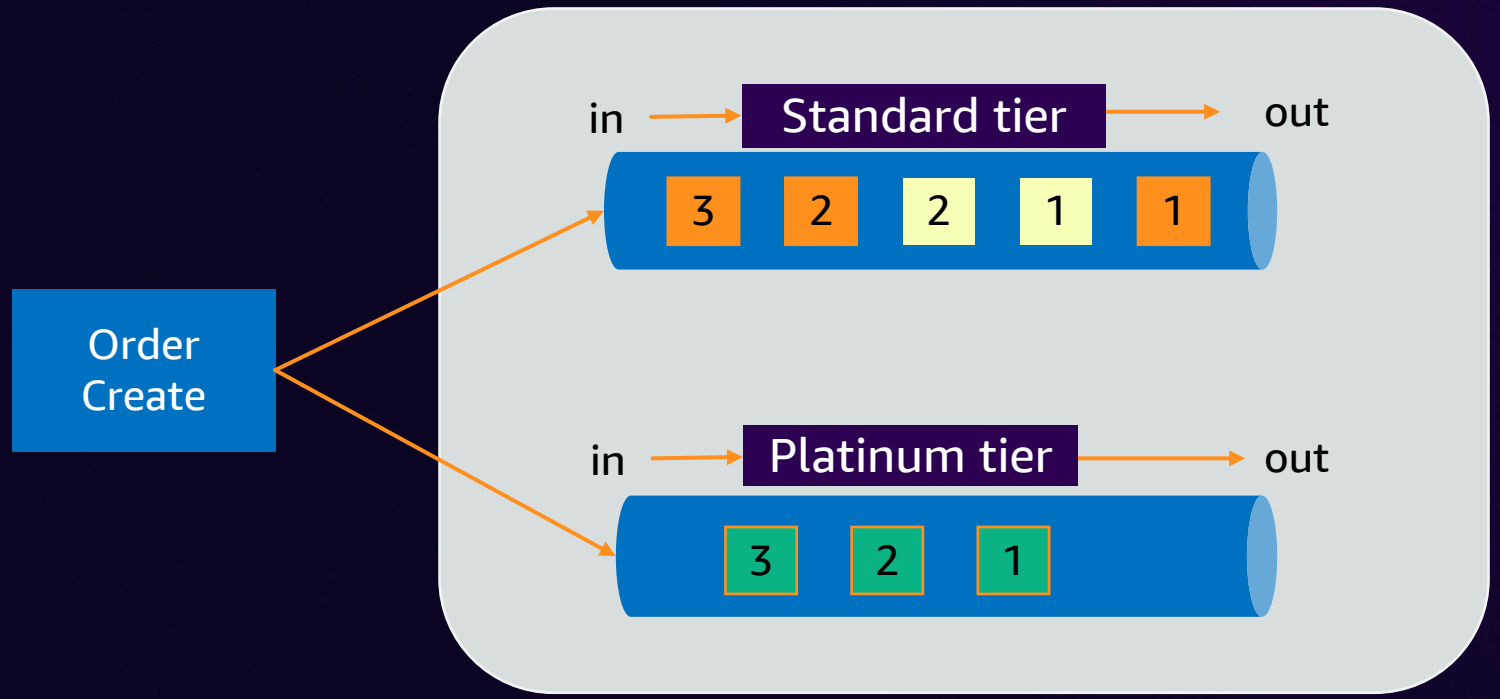
Noisy neighbor



Tenant A causing noisy neighbor

- Solve noisy neighbor problem while continuing to meet the isolation requirements of tenants
- At the same time, remain agile, simplify operations, and optimize costs

Queue sharing based on tier



Tenant A

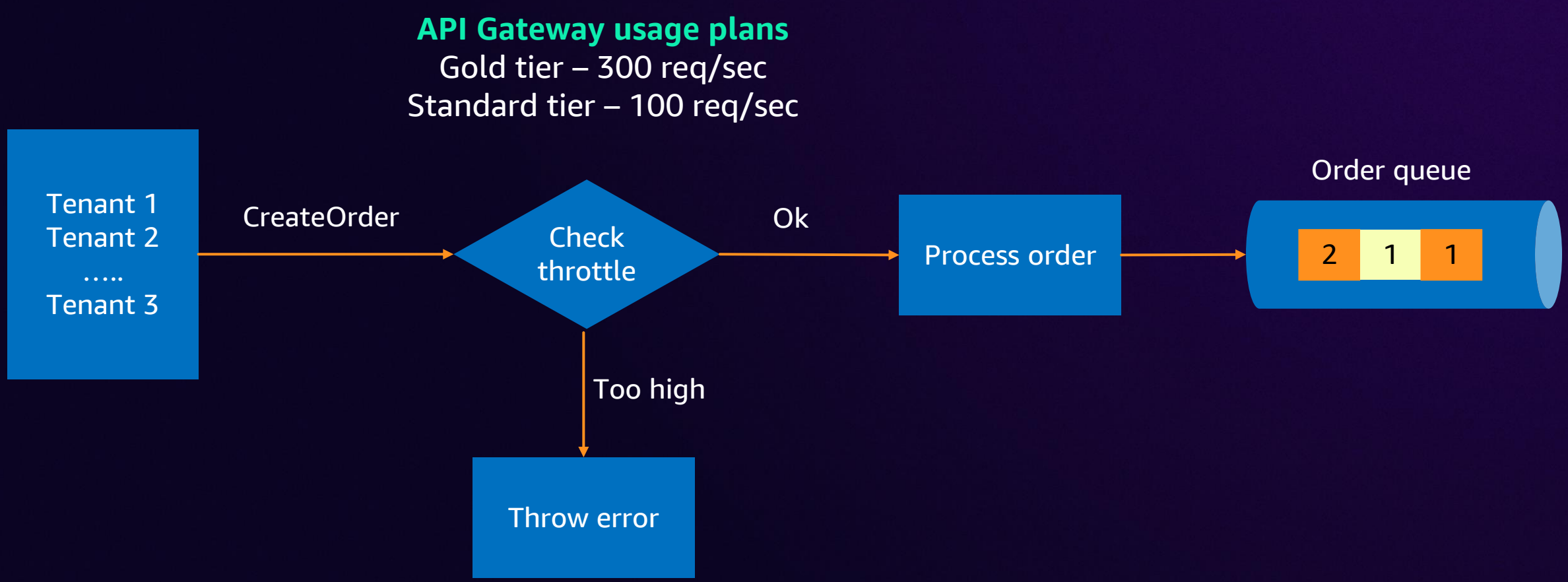
Tenant B

Belong to standard tier

Tenant C

Belongs to platinum tier

Handling a noisy neighbor effectively: Rate Limit



Configuring batch size for data isolation

Messages

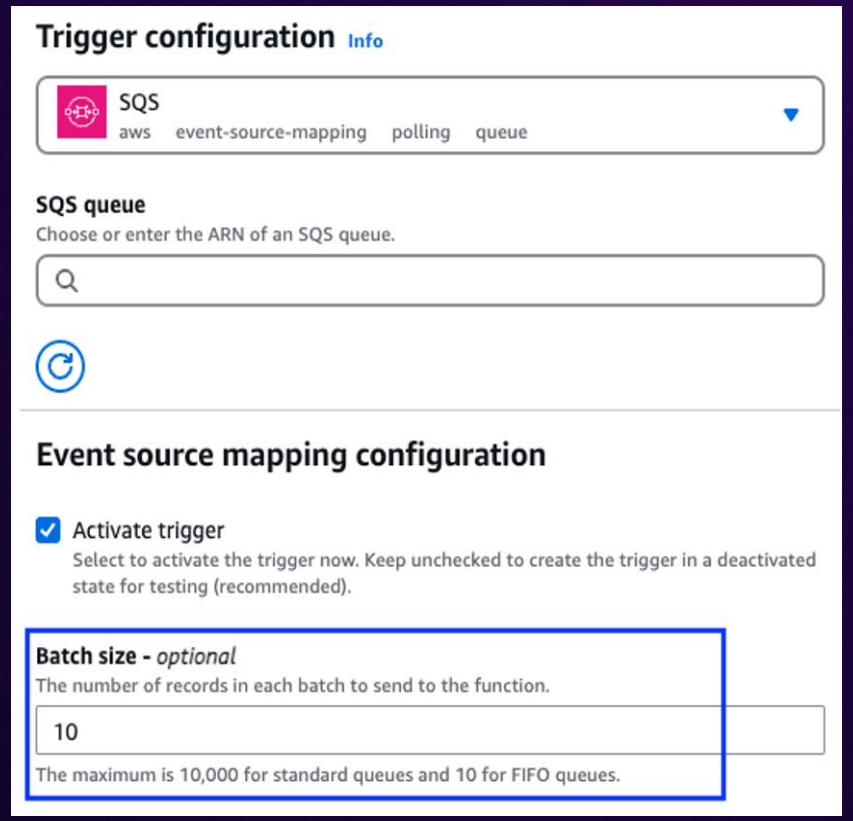
Batch size = 10

3 5

1 4 2

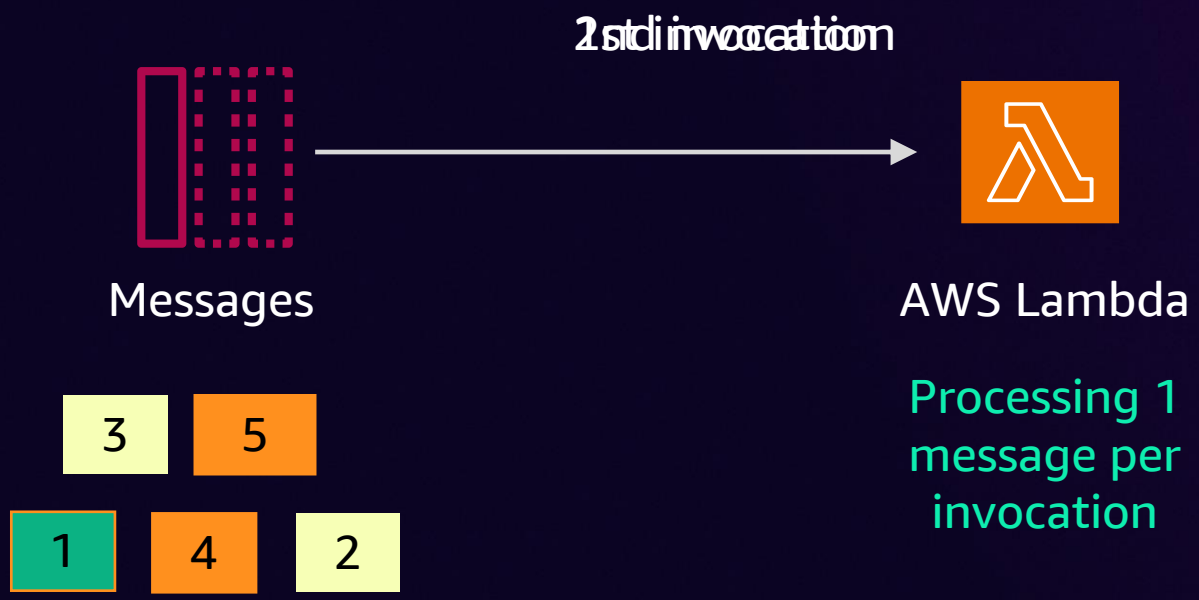
AWS Lambda

Single invocation has access to cross-tenants' messages

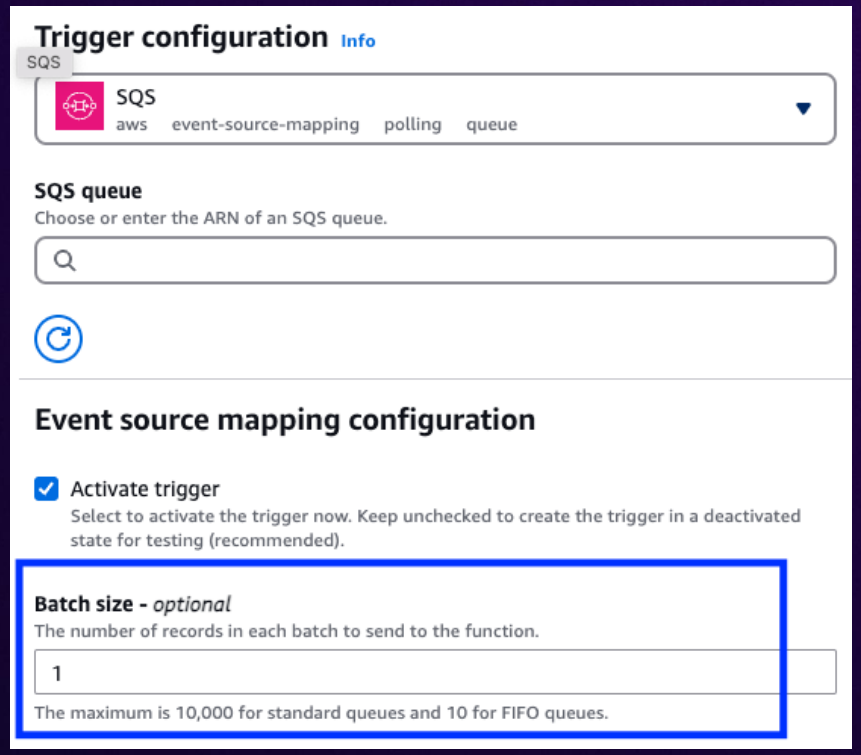


By default, batch size = 10

Configuring batch size for data isolation



Processing 1 message per invocation

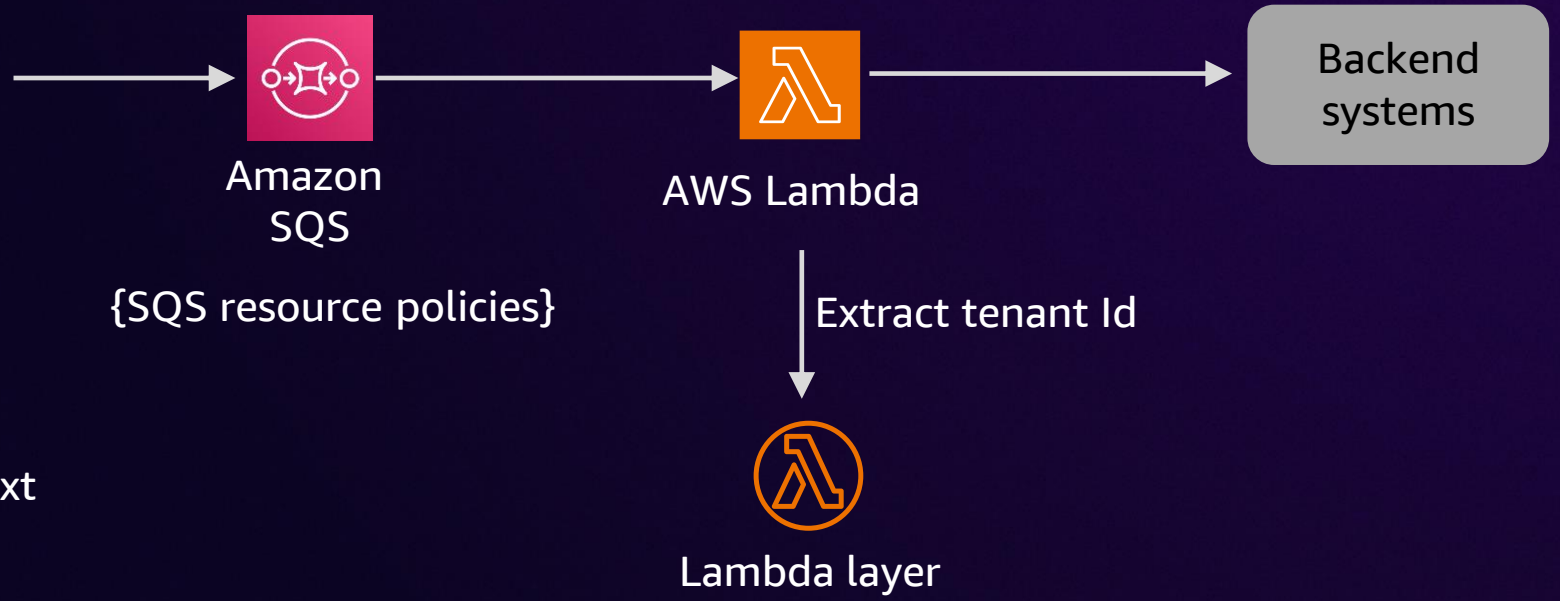


Batch size = 1

Data isolation and security: Message attributes

```
    MessageBody= message_body,
    MessageAttributes=
        {
        'tenant_id': {
            'StringValue': tenant_id,
            'DataType': 'String'
            },
        'message_version': {
            'StringValue': 'Version 1.0',
            'DataType': 'String'
            }
        })
```

Message attributes with tenant's context



Multi-tenancy best practices with Amazon SQS

Configure the queue to delay messages to put off work until later

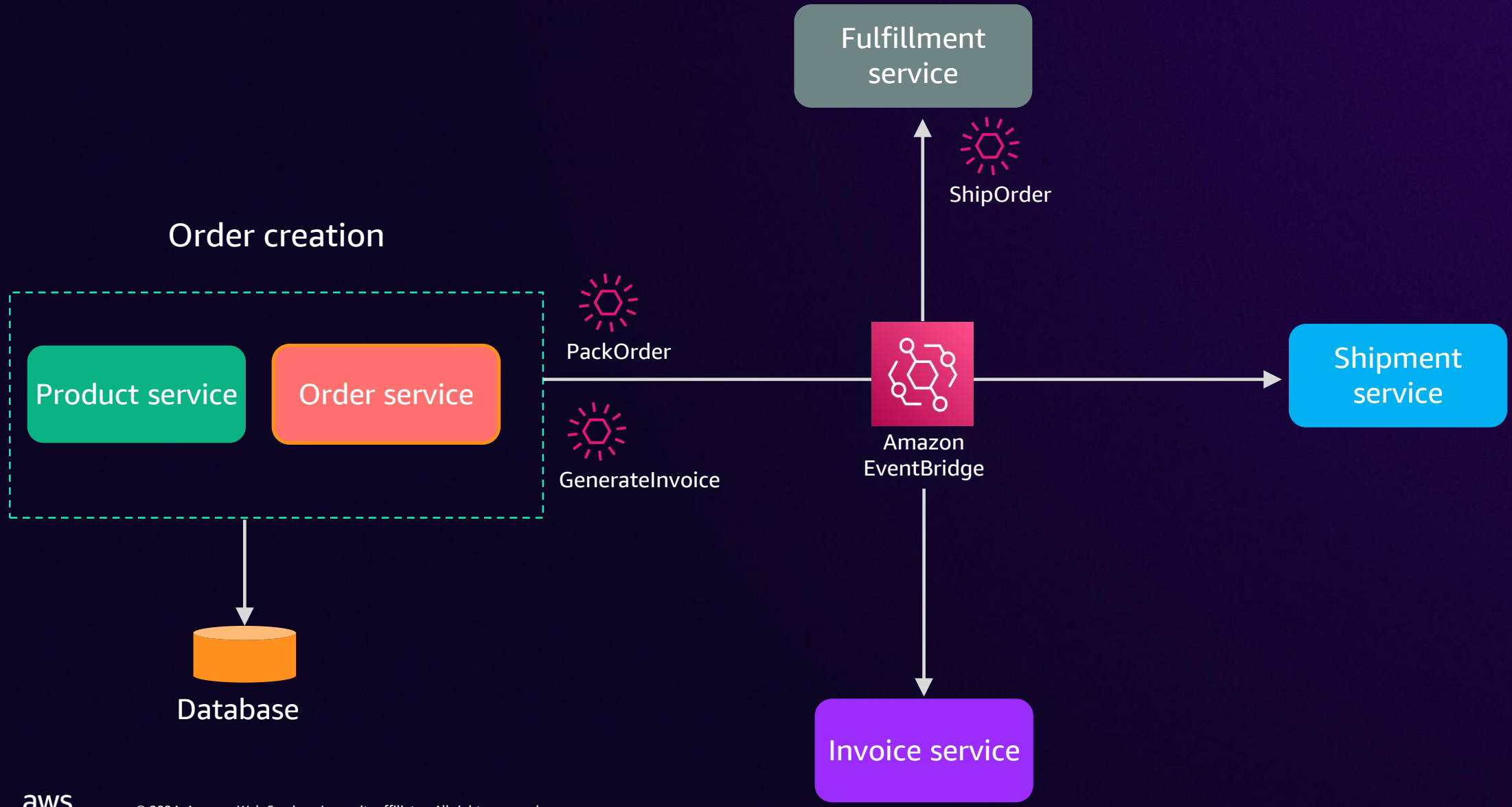
Avoid too many **in-flight** messages

Use **dead-letter queues** for messages that can’t be processed

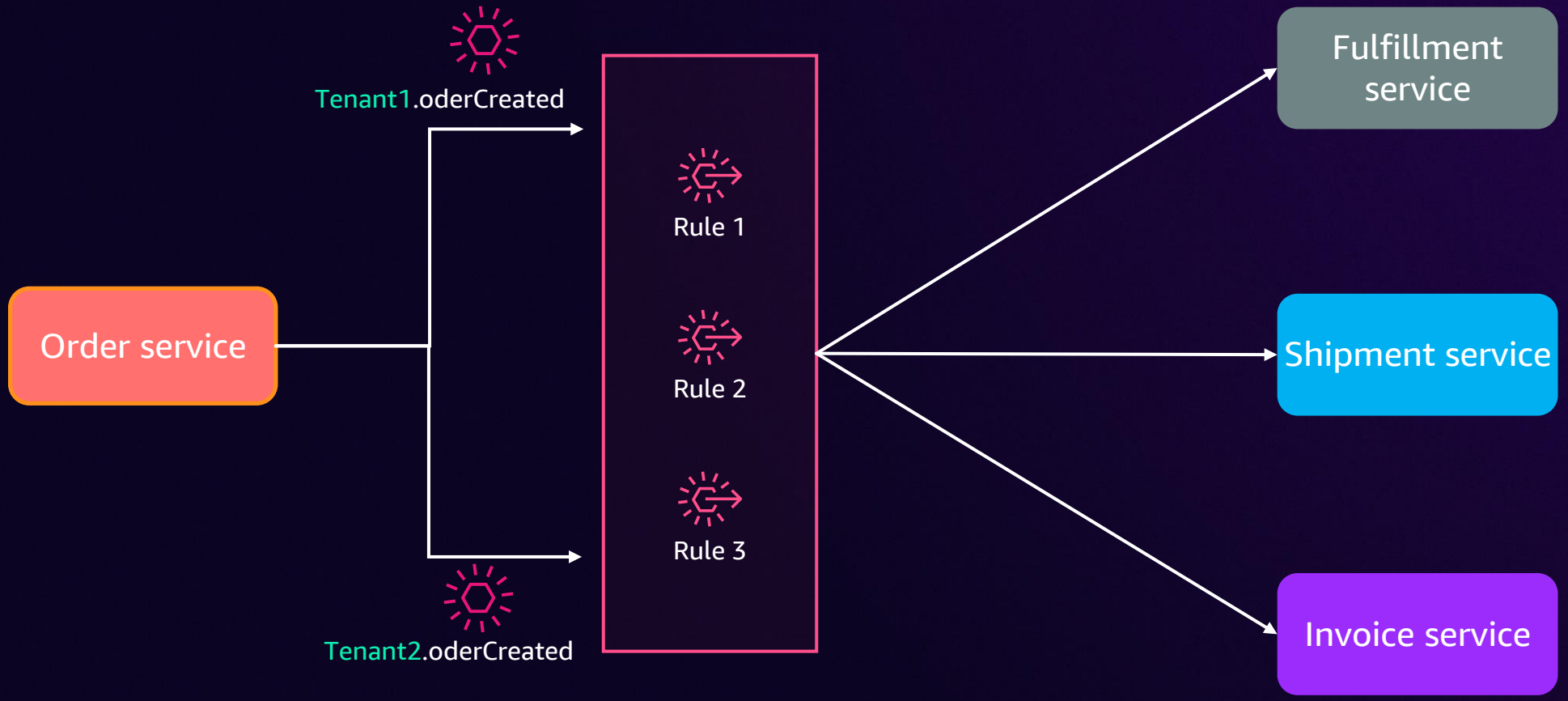
Pass tenant context as message attributes while sharing queues across tenants

Event-driven use case considerations for multi-tenancy

Event-driven ecommerce use case



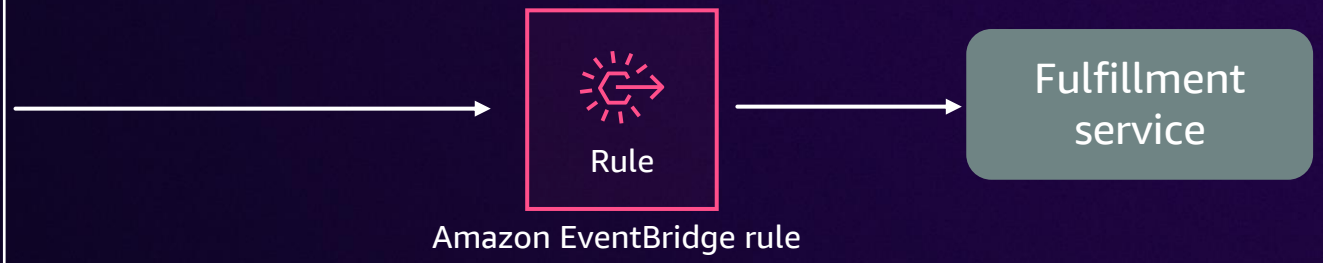
Event filtering and routing with Amazon EventBridge



Routing events based on tiers

An example event from **Order** Service

```
{
  "source": ["order.service"],
  "detail": {
    "tenantId": ["tenant-xxxxx"],
    "tier": ["Gold"]
  },
  "detail-type": ["OrderCreated"],
  "resources": [],

   //. . . additional attributes
}
```




```
{
  "source": ["order.service"],
  "detail": {
    "tier": ["Gold"]
  }
}
```

Optimizing the number of rules

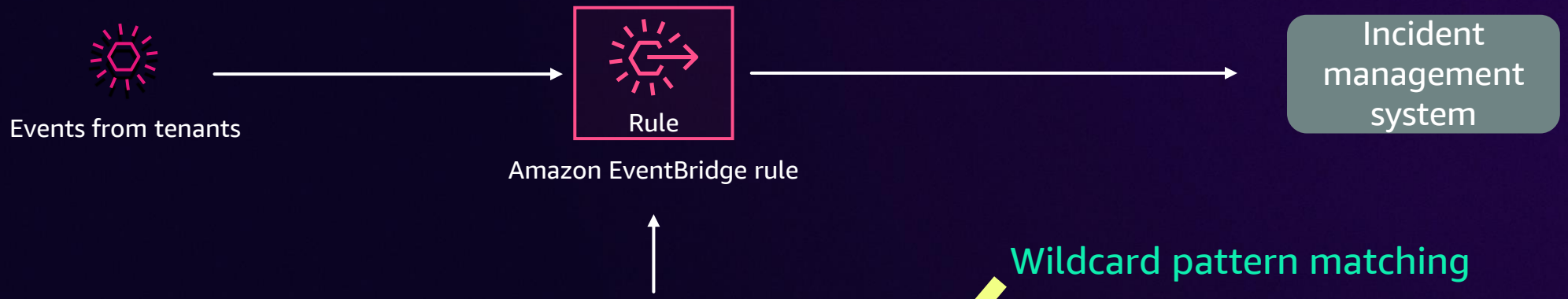
Event for **Tenant A**

```
{
  "source": ["shopping.cart.service"],
  "detail": {
    "tenantId": ["tenant-A"],
    "tier": ["Gold"]
  },
  "detail-type":
[”shopping.cart.error.timeout"],
  "resources": [],

   //. . . additional attributes
}
```

Event for **Tenant B**

```
{
  "source": ["shopping.inventory.service"],
  "detail": {
    "tenantId": ["tenant-B"],
    "tier": ["Gold"]
  },
  "detail-type”:
["shopping.inventory.error.outofstock"],
  "resources": [],

   //. . . additional attributes
}
```

Filtering events using wildcard pattern matching

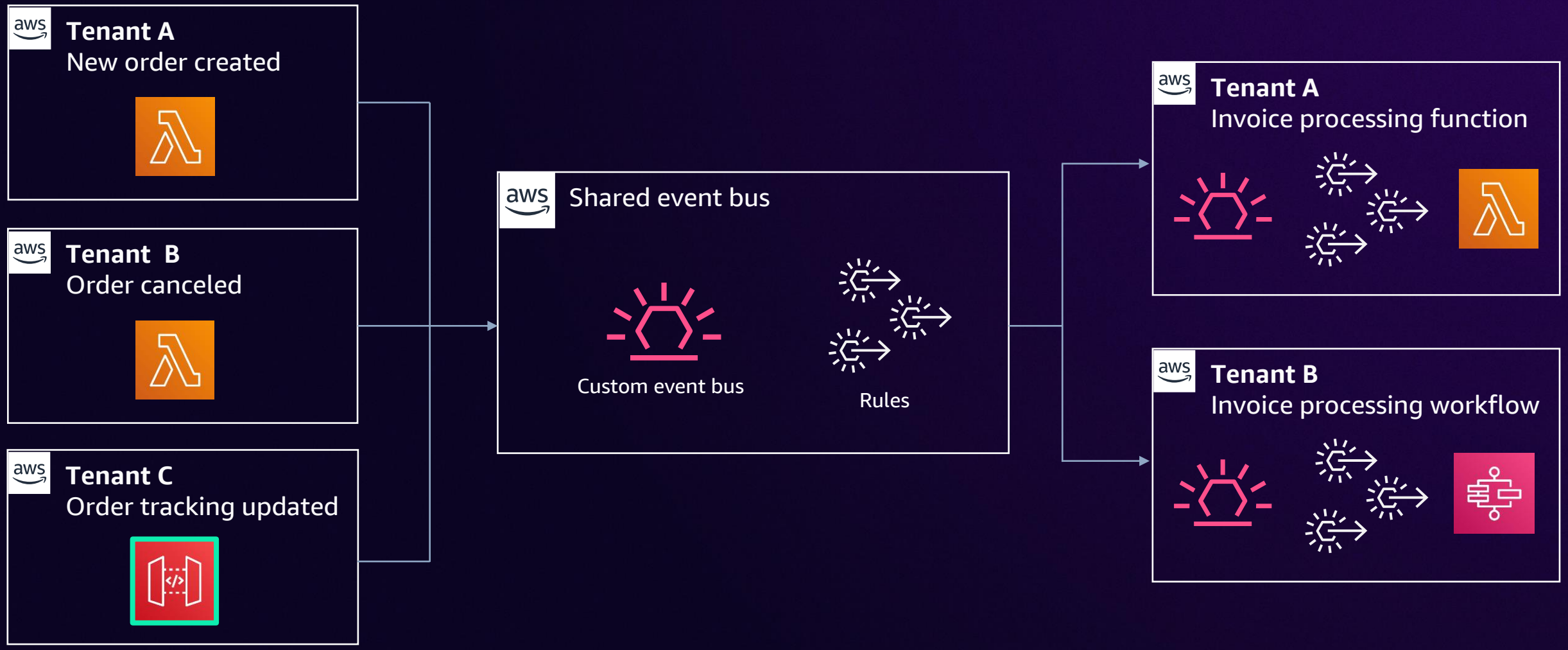


Wildcard pattern matching

```
{ "source": [{
"prefix": "shopping."
}],
"detail-type": [{
"wildcard": "*.error.*"
}],
"detail": {
"tenantId": [{ "exists": true }]
} }
```

Fewer rules results in increased operational excellence

Hybrid approach: Bridge model



Summary: Amazon EventBridge

- Use a **single rule per subscriber**
- Avoid using the **default event bus** for custom events
- Use **wildcard pattern matching** wherever feasible
- With a pool model, you get a centralized bus with resource limits applicable

Key takeaways

01 **Architectural design:** Select the right multi-tenant model, externalize data isolation

02 **Cost optimization:** Use Lambda layers, execute asynchronously when possible

03 **To avoid noisy neighbors:** Implement rate limiting, a tier-based strategy, and capacity reservation

04 **Scalability and agility:** Leverage low-code integration, serverless microservices

Check out these other sessions

API306 | Integration patterns for distributed systems

Wednesday (Dec. 4) at 09:00 AM – MGM Grand | Level 1 | Grand 116

SVS324 | Implementing security best practices for serverless applications

Wednesday (Dec. 4) at 10:30 AM – MGM Grand | Level 1 | Grand 122

API311 | Application integration for platform builders

Wednesday (Dec. 4) at 04:00 PM – MGM Grand | Level 3 | Premier 318

Continue your AWS serverless learning

Learn at your own pace

Expand your serverless skills with our learning plans on **AWS Skill Builder**

Increase your knowledge

Use our **Ramp-Up Guides** to build your serverless knowledge

Earn an AWS serverless badge

Demonstrate your knowledge by achieving **digital badges**

https://s12d.com/serverless-learning

Best practices, for everyone

Powertools for AWS Lambda

Python | TypeScript | Java | .NET

- Batch processing
- Config management
- Observability
- Secrets handling
- REST/GraphQL API
- Idempotency
- Input/output validation
- BYO middleware
- Self-documented schemas
- Feature flags
- Caching
- Streaming
- Data extraction

*Feature set may vary across languages

Thank you!

Please complete the session survey in the mobile app

Anand Bilgaiyan

/in/anand-bilgaiyan

Nishant Dhiman

/in/nishant-dhiman