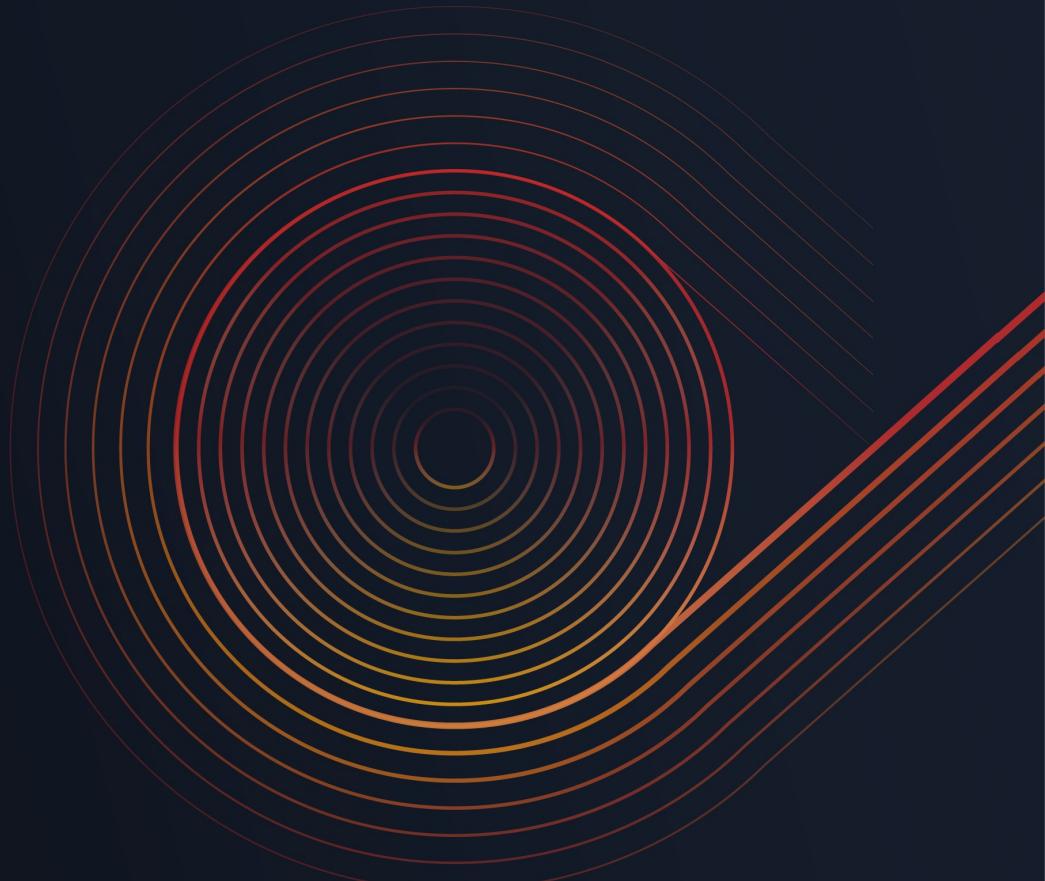


Building event-driven architectures on AWS

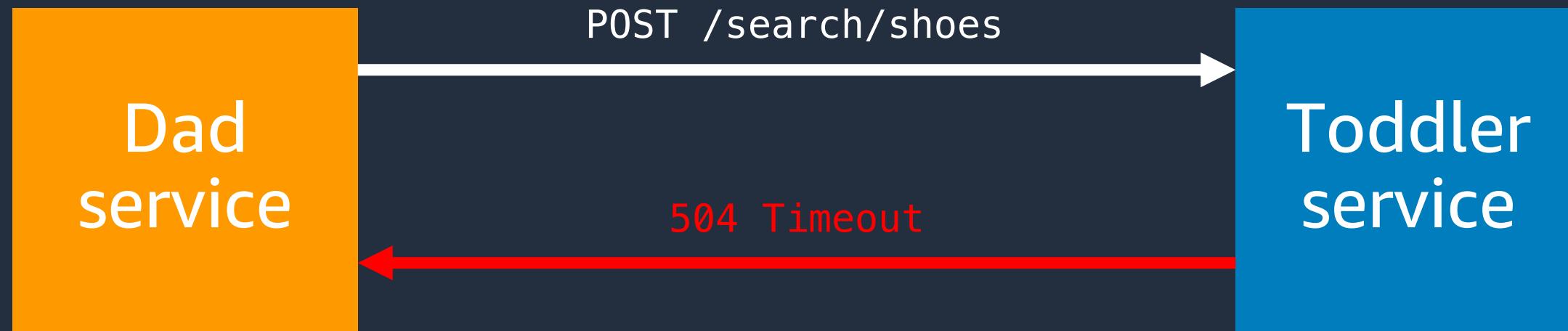
Iaroslav Ustinov, Enterprise Solutions Architect



Event-driven kids



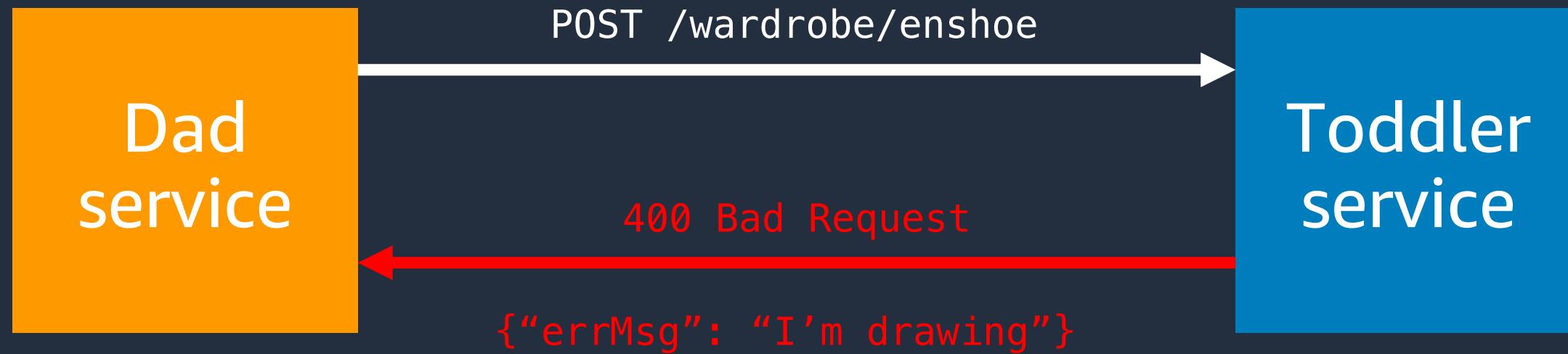
Event-driven kids



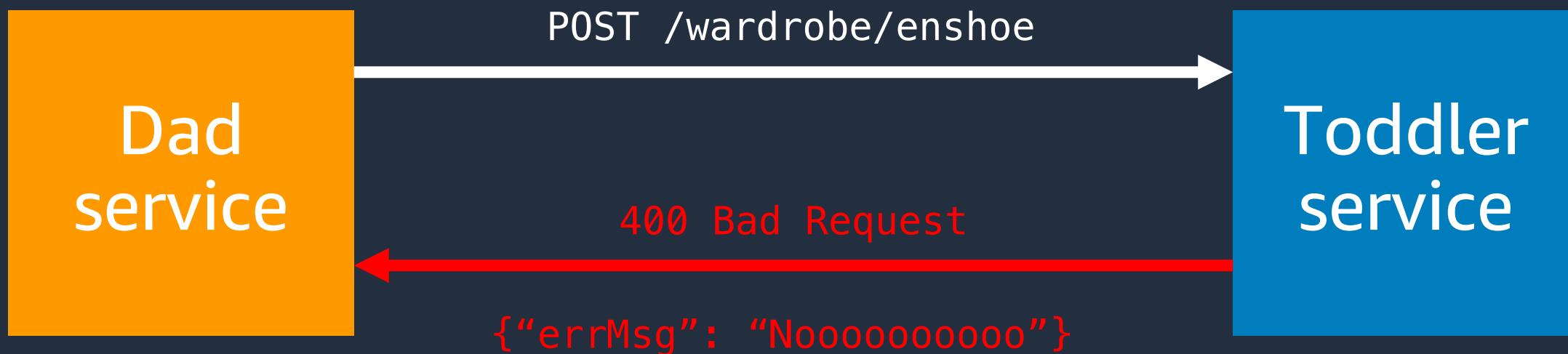
Event-driven kids



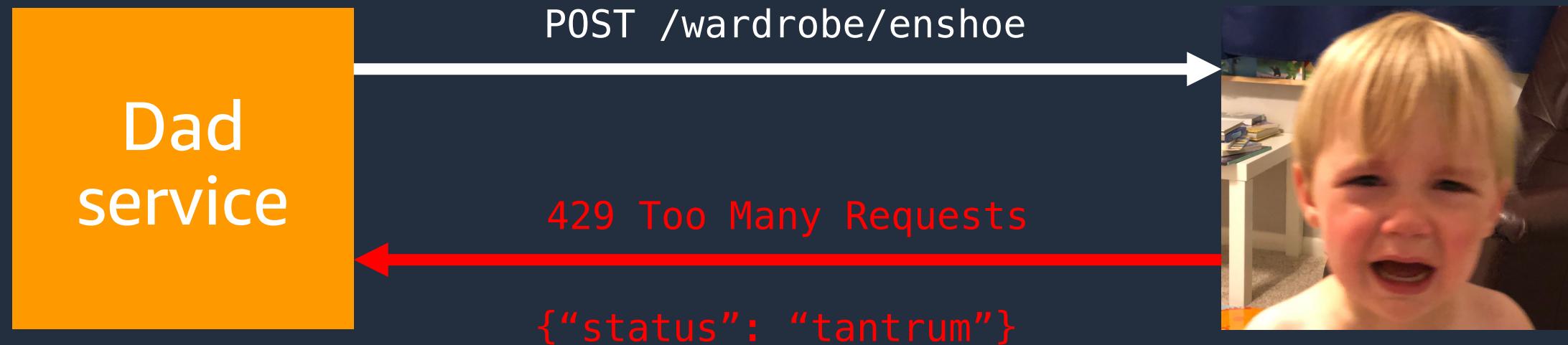
Event-driven kids



Event-driven kids



Event-driven kids



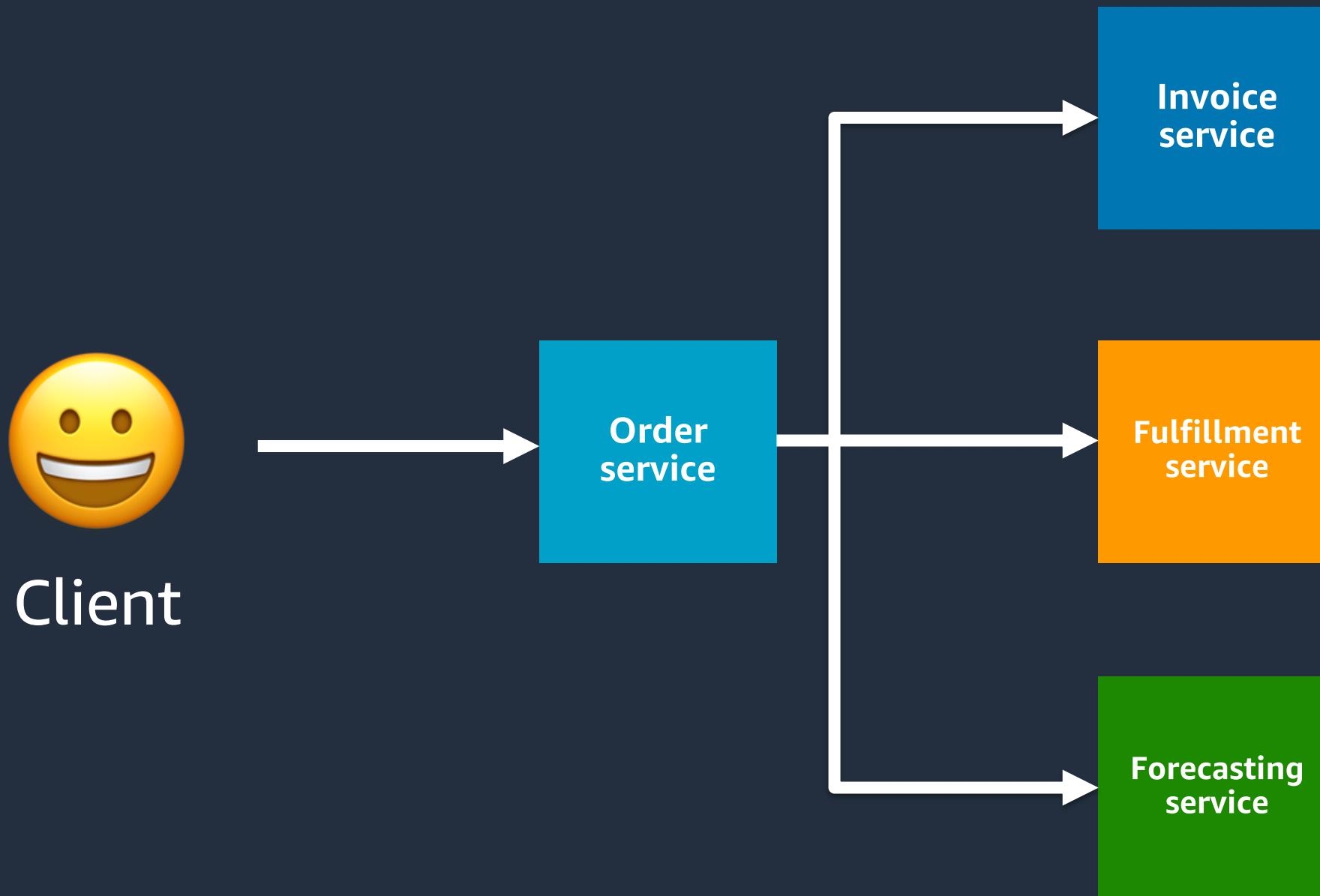
Challenges with distributed systems



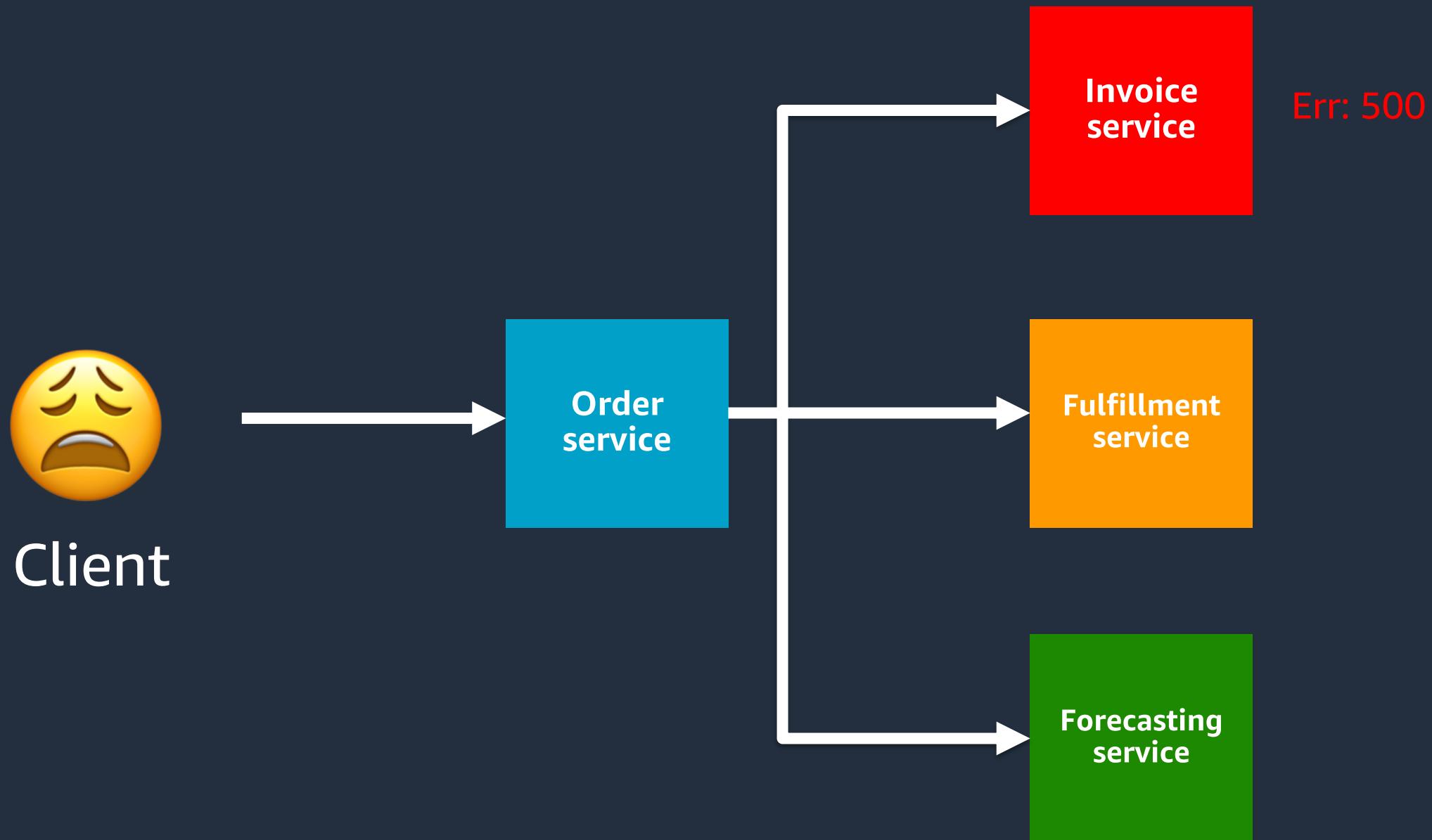
Microservices start simple



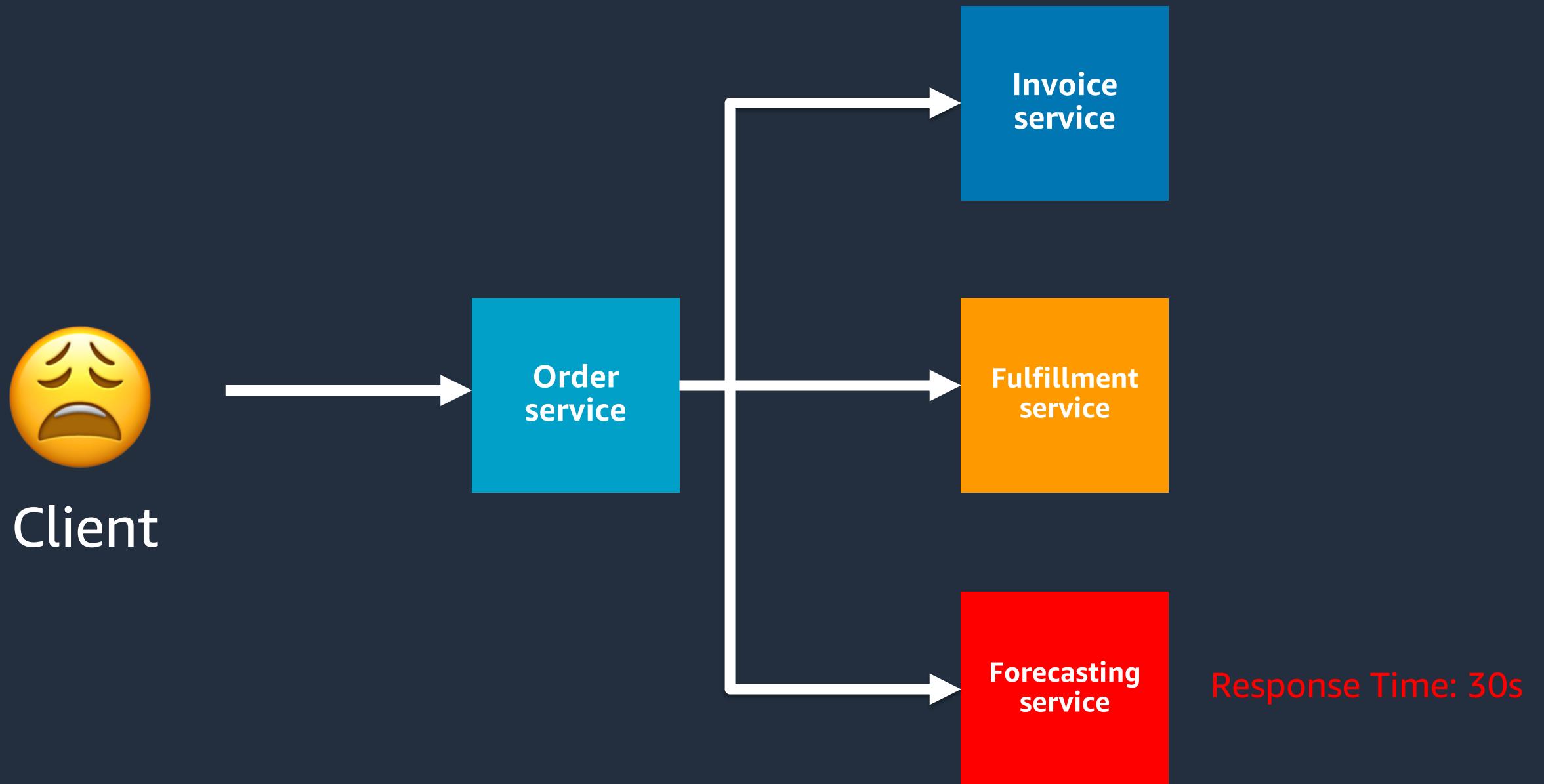
Synchronous API



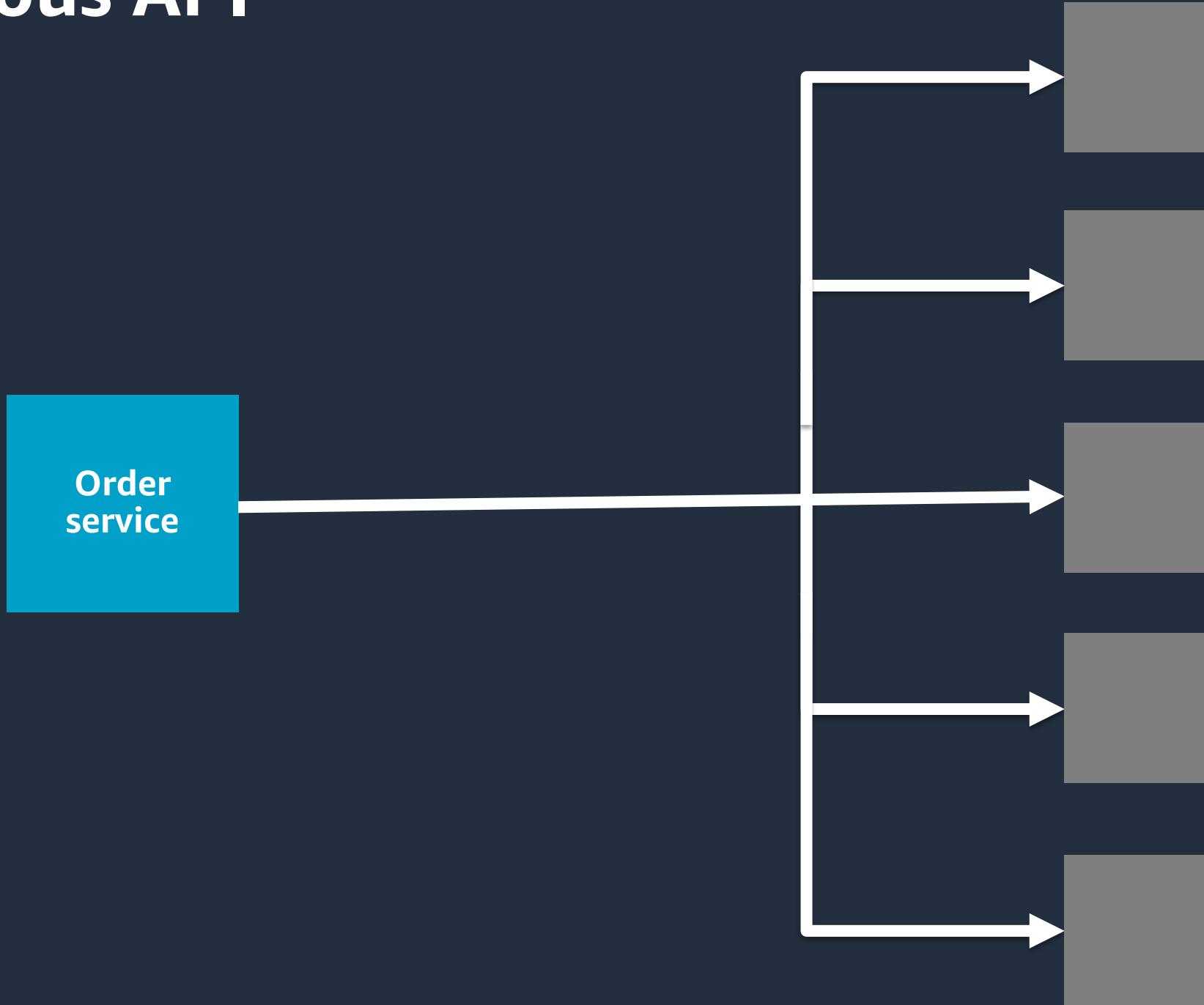
Synchronous API

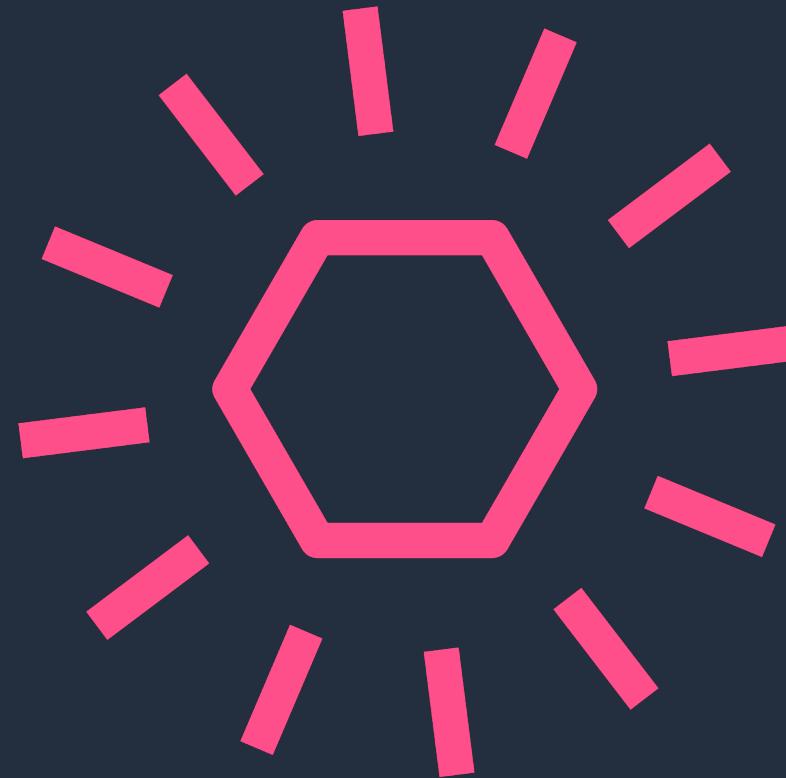


Synchronous API



Synchronous API



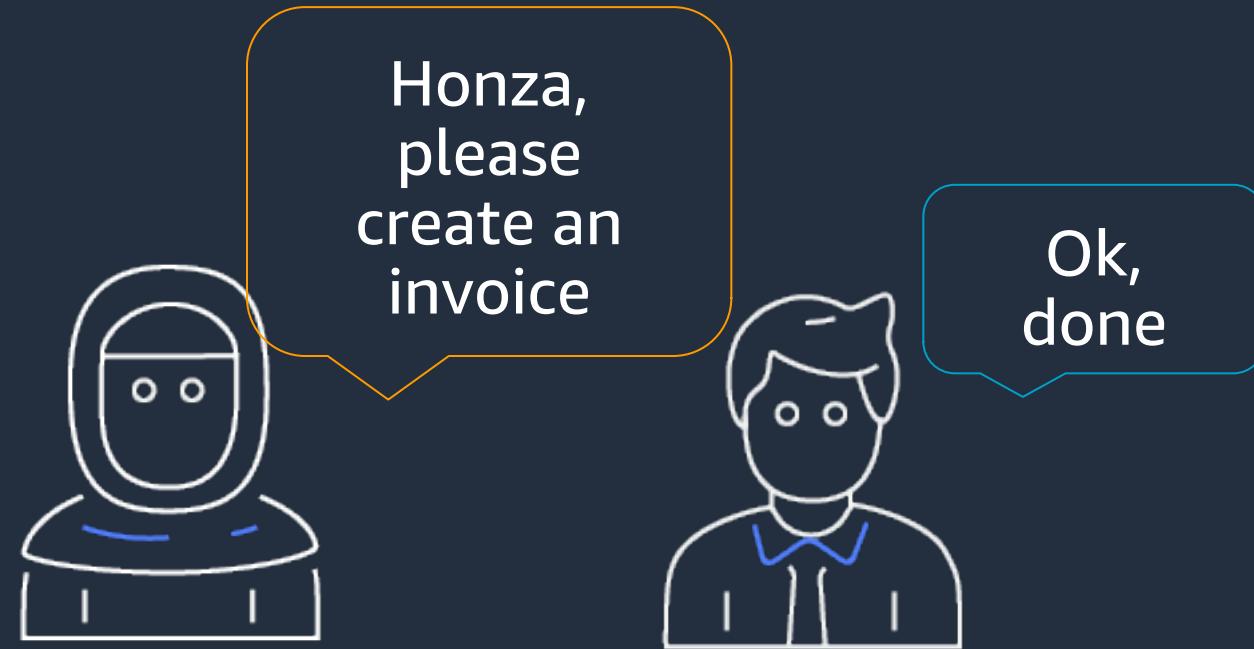


event

[i-'vent] noun

A signal that a system's state has changed.

Events are observable, not directed

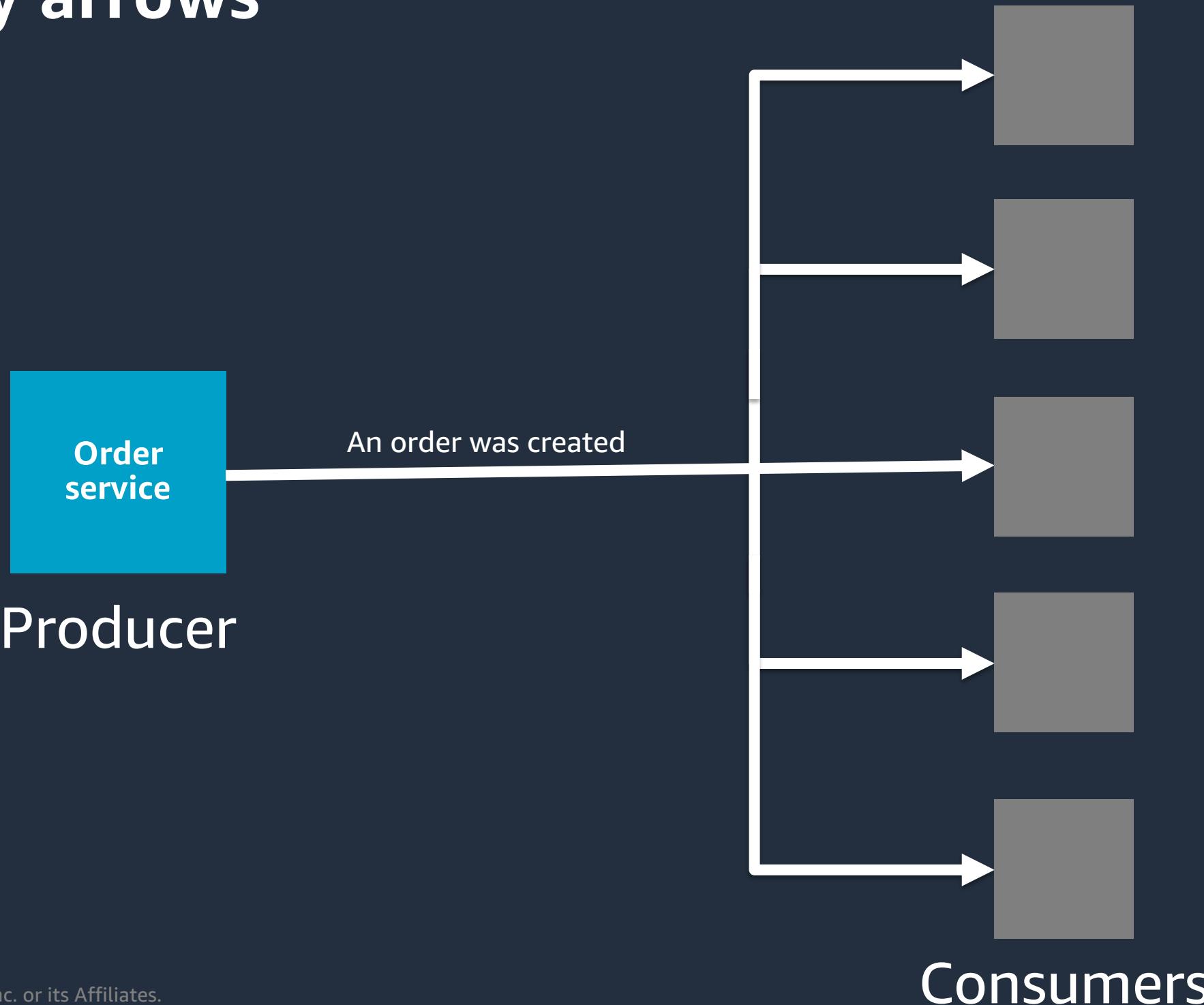


Directed commands

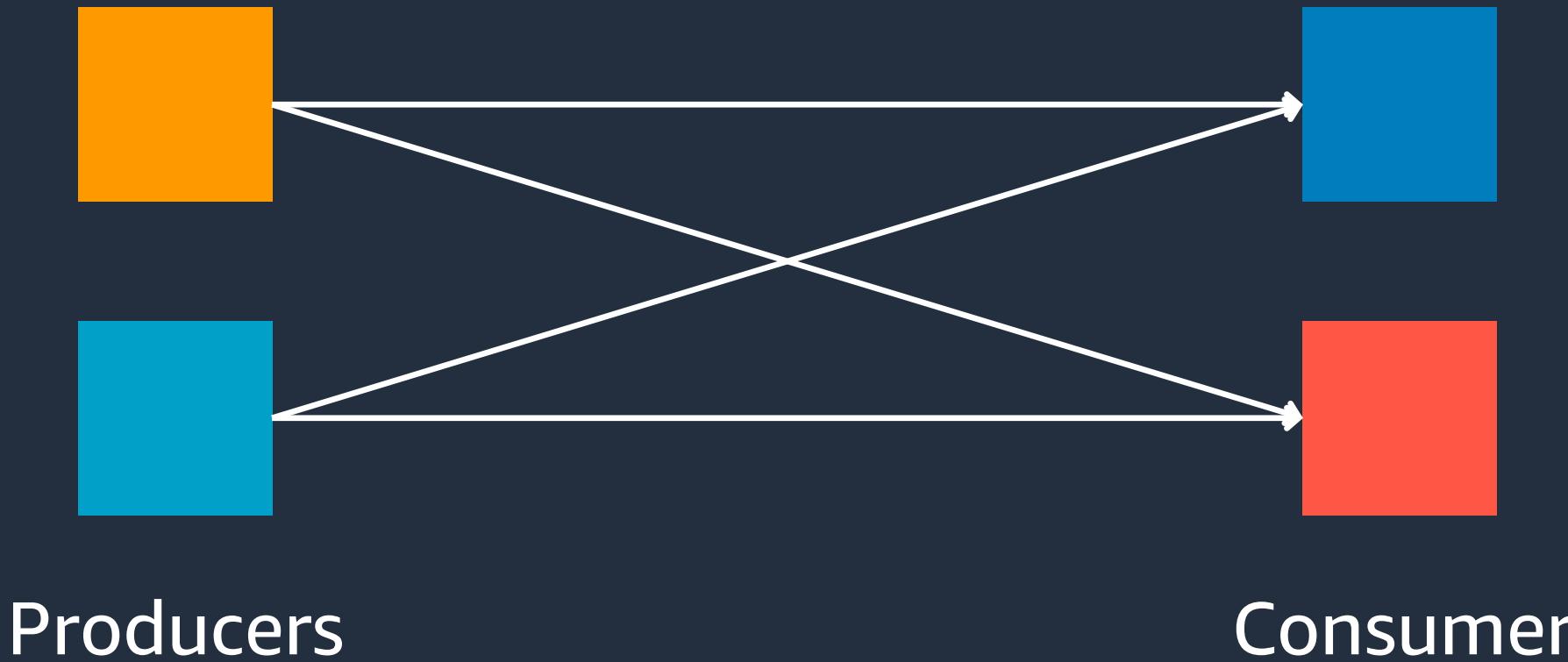


Observable events

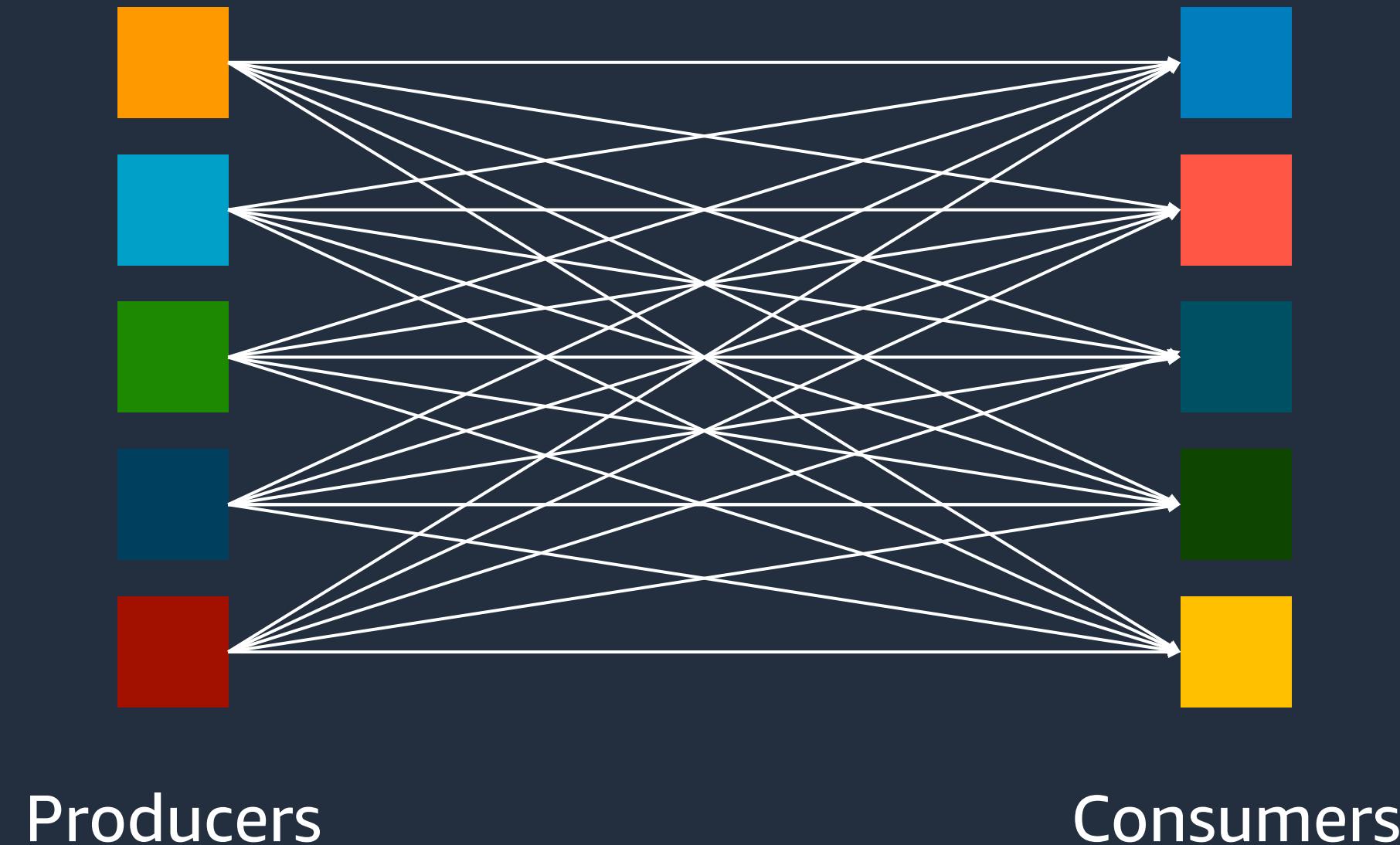
Still many arrows



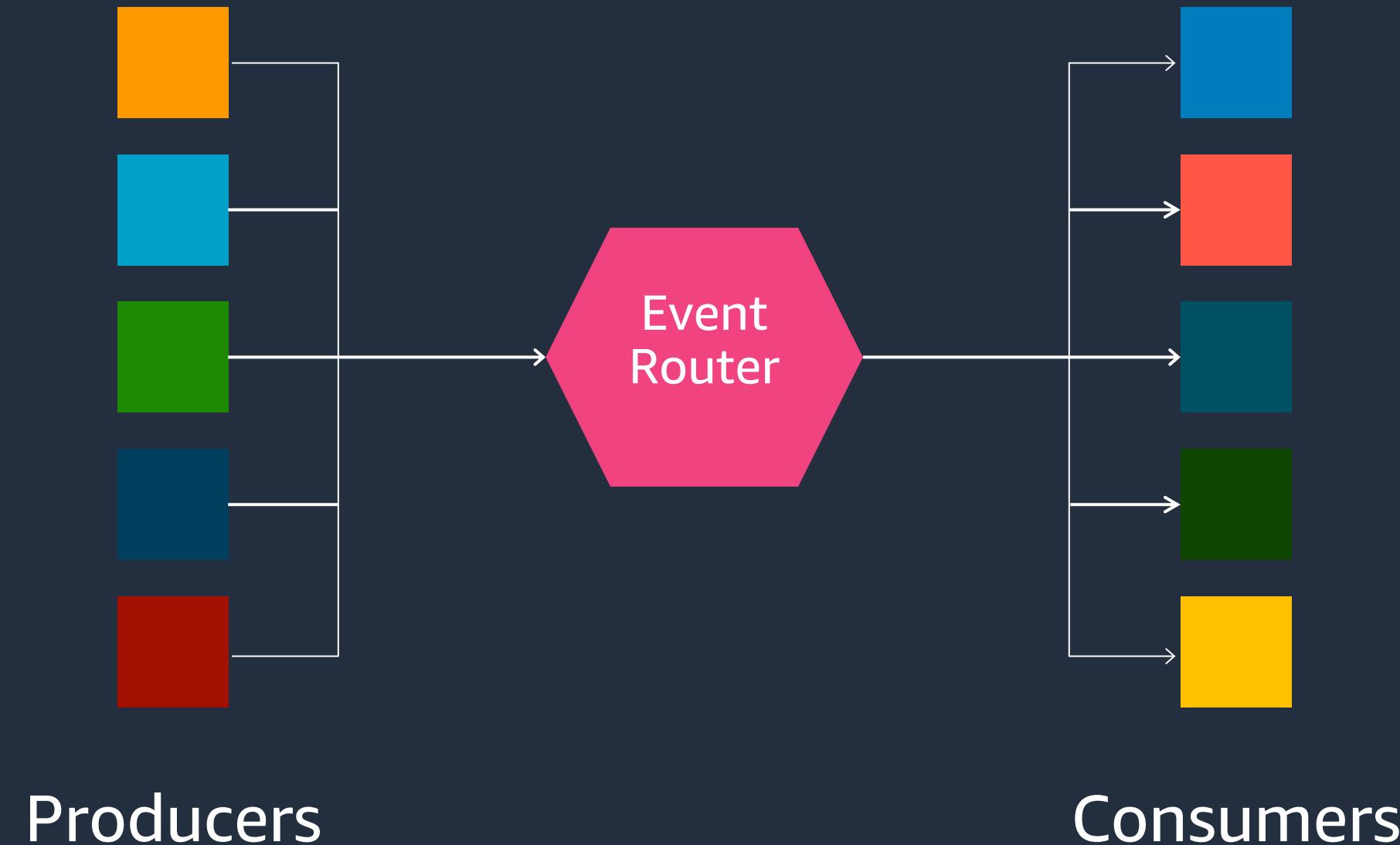
Why do you need a router?



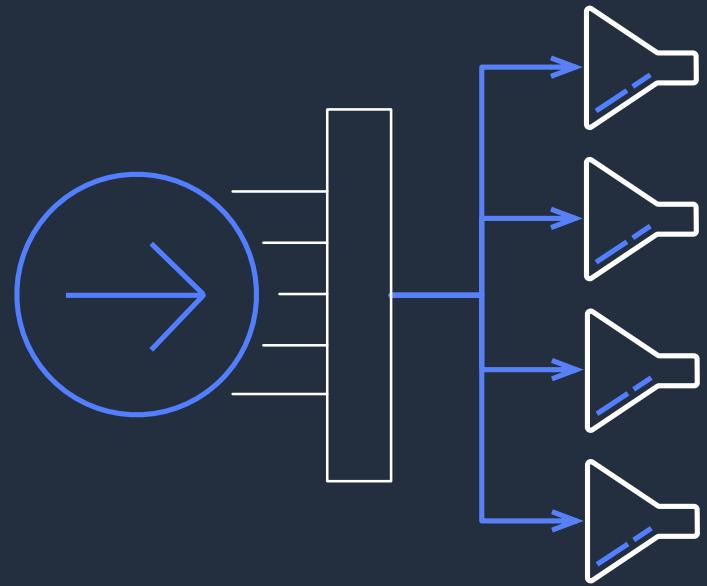
Why do you need a router?



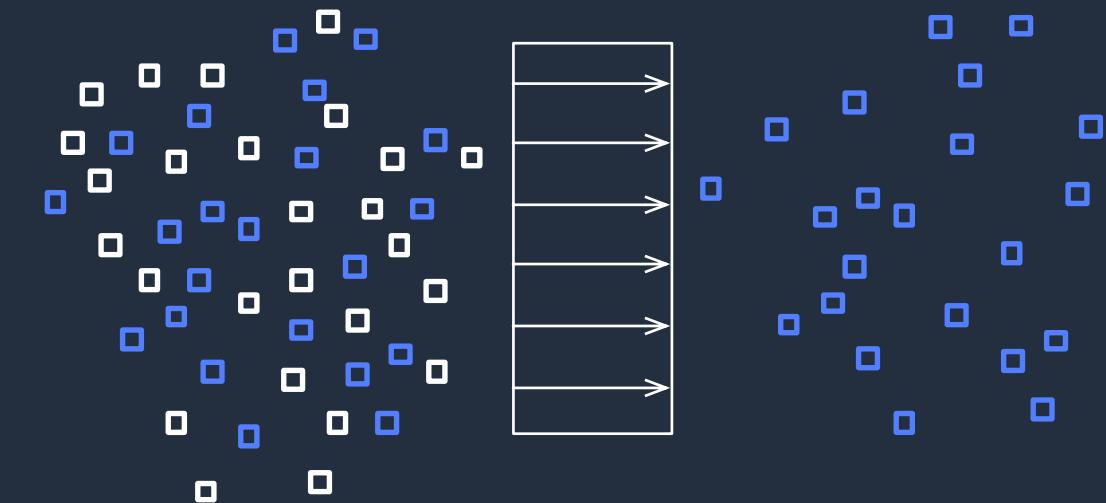
Why do you need a router?



Event Routers



Abstracts producers
and consumers



Selects and filters
events

Event Routers in AWS

Topics

AWS Service



Amazon Simple
Notification Service

Characteristics

AWS Lambda/Amazon SQS/HTTP targets
Millions of subscriptions
Filter on event metadata

Event buses



Amazon EventBridge

Over 35 targets
Native SaaS event sources
Metadata and payload routing

Amazon SNS Message Filters

Message Attributes

```
{  
    "location": "eu-west"  
}
```



Amazon SNS
"Orders" Topic

Filter Policy

```
{  
    "location":  
        ["us-west", "us-east"]  
}
```

Amazon SNS Subscription



Amazon SQS
"US Orders" Queue

```
{  
    "location":  
        ["eu-west", "eu-east"]  
}
```

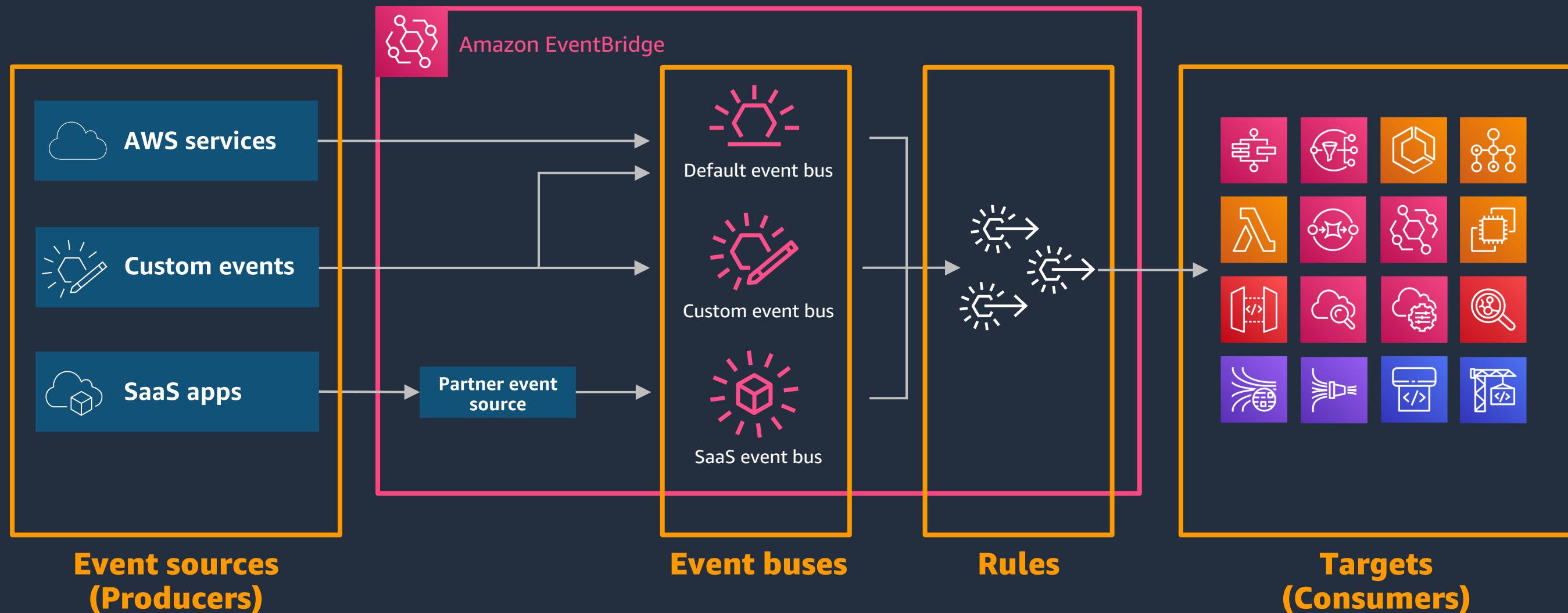


AWS Lambda



AWS Lambda

What is Amazon EventBridge?



Amazon EventBridge

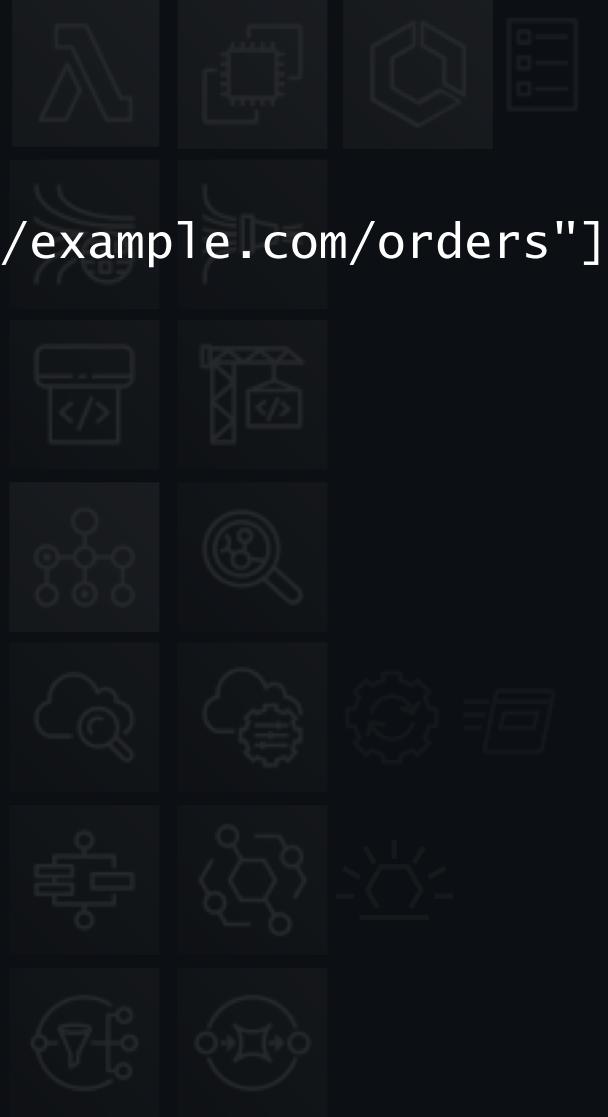
Example event:

The diagram illustrates the flow of events from SaaS applications to Amazon EventBridge. It features a central vertical timeline with three main stages: "SaaS apps" at the bottom, "Custom events" in the middle, and "Amazon EventBridge" at the top. Arrows indicate the progression of events from SaaS apps through custom events to the event bus. The "SaaS apps" stage contains code snippets for creating a ticket and updating its status. The "Custom events" stage contains code snippets for ticket creation and modification. The "Amazon EventBridge" stage shows the resulting event structure with fields like detail-type, source, and detail.

```
{  
  "detail-type": "Ticket Created",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "department": "billing",  
    "creator": "user12345"  
  }  
}  
  
...  
  
{  
  "detail-type": "Ticket Updated",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "status": "PENDING",  
    "department": "billing",  
    "creator": "user12345"  
  }  
}  
  
{  
  "detail-type": "Ticket Updated",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "status": "PENDING",  
    "department": "billing",  
    "creator": "user12345"  
  }  
}  
  
{  
  "detail-type": "Ticket Updated",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "status": "PENDING",  
    "department": "billing",  
    "creator": "user12345"  
  }  
}
```

Example rule:

```
{  
  "source": ["aws.partner/example.com/orders"]
```



Amazon EventBridge

Example event:

The diagram illustrates the flow of event data from SaaS applications to Amazon EventBridge. It features a central vertical column representing the event data structure, with arrows pointing from various sources to specific fields.

Event Data Structure:

```
{  
  "detail-type": "Ticket Created",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "department": "billing",  
    "creator": "user12345"  
  }  
  ...  
}
```

Sources and Fields:

- SaaS apps** (Cloud icon) → **ticketId**: "987654321"
- AWS services** (Cloud icon) → **source**: "aws.partner/example.com/orders"
- Custom events** (Lightbulb icon) → **detail-type**: "Ticket Created", **department**: "billing", **creator**: "user12345"
- Partner event source** (Cloud icon) → **detail** (indicated by an arrow pointing to the curly brace)

Example rule:

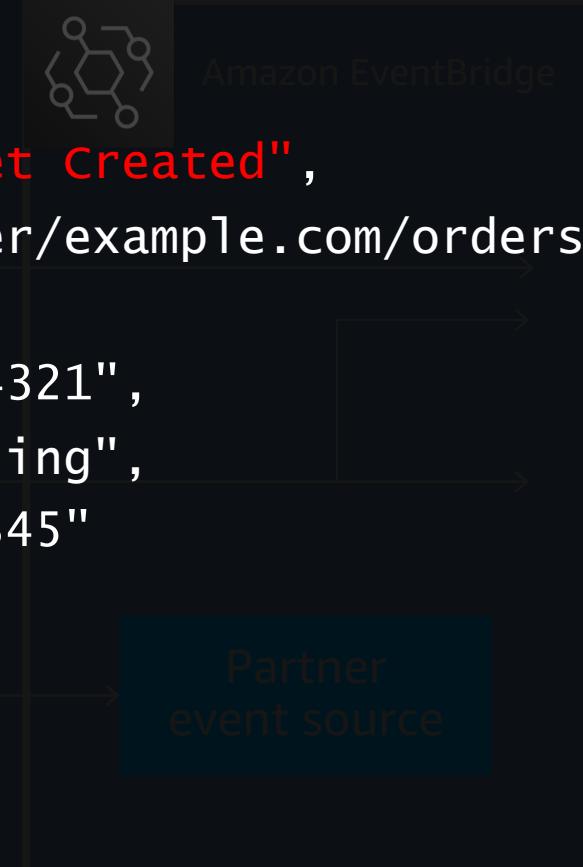
```
{  
  "detail": {  
    "department": ["billing", "fulfillment"]  
  }  
}
```



Amazon EventBridge

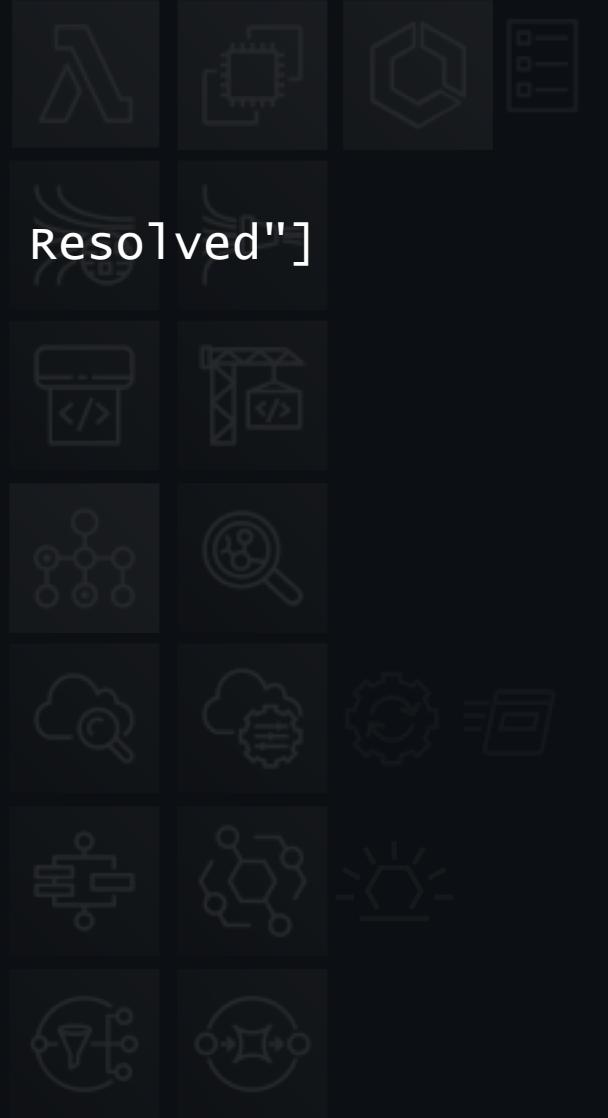
Example event:

```
{  
  "detail-type": "Ticket Created",  
  "source": "aws.partner/example.com/orders",  
  "detail": {  
    "ticketId": "987654321",  
    "department": "billing",  
    "creator": "user12345"  
  }  
}  
  
SaaS apps
```

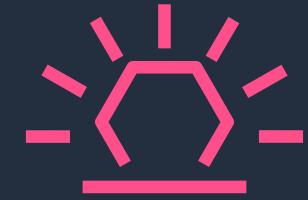


Example rule:

```
{  
  "detail-type": ["Ticket Resolved"]  
}
```

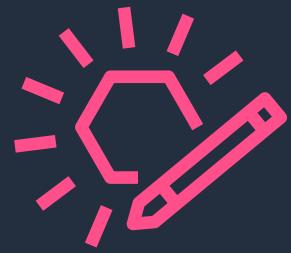


Type of Event Buses



Default Bus

Events from
AWS services
Scheduled events
Custom events



Custom Buses

Custom events

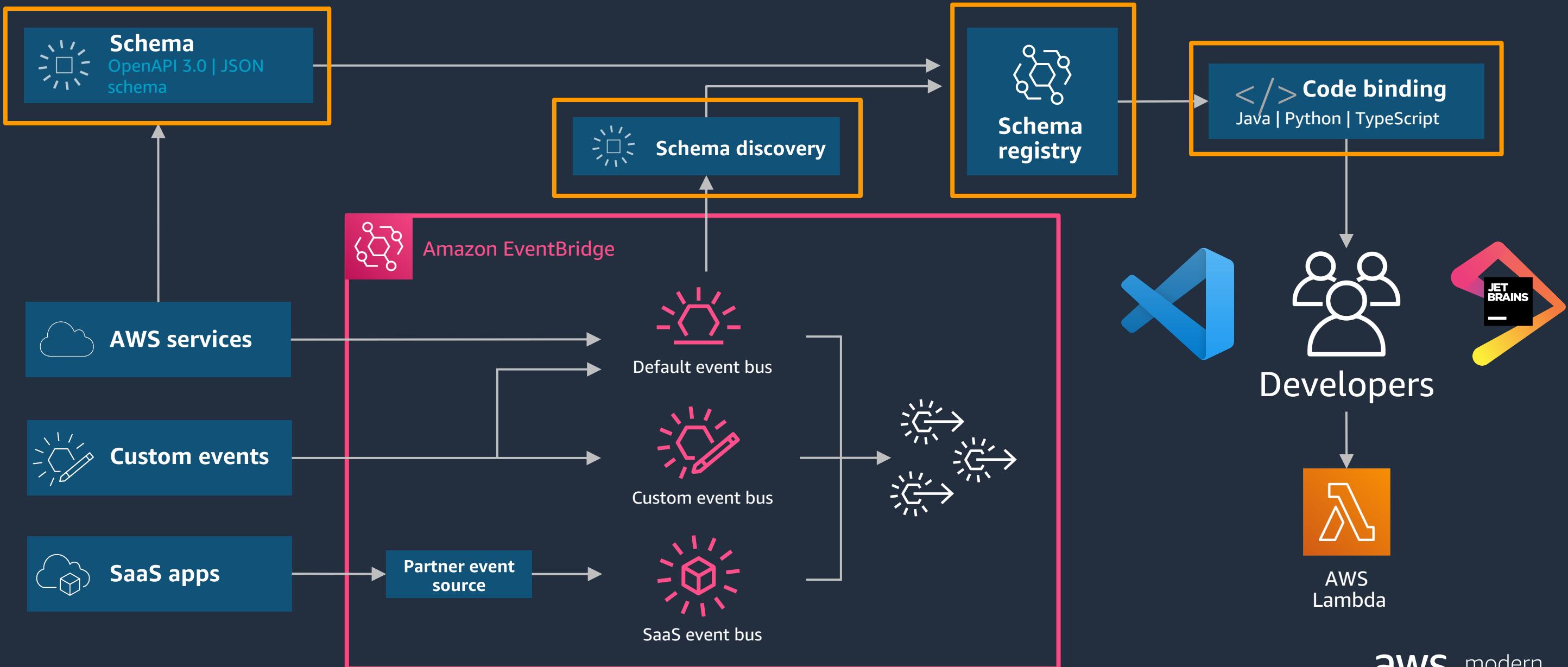


SaaS Buses

3rd party SaaS events
Datadog
New Relic
Zendesk

Amazon EventBridge architecture

Amazon EventBridge Schema Registry



Event Routers in AWS

Topics

AWS Service



Amazon Simple
Notification Service

Characteristics

AWS Lambda/Amazon SQS/HTTP targets
Millions of subscriptions
Filter on event metadata

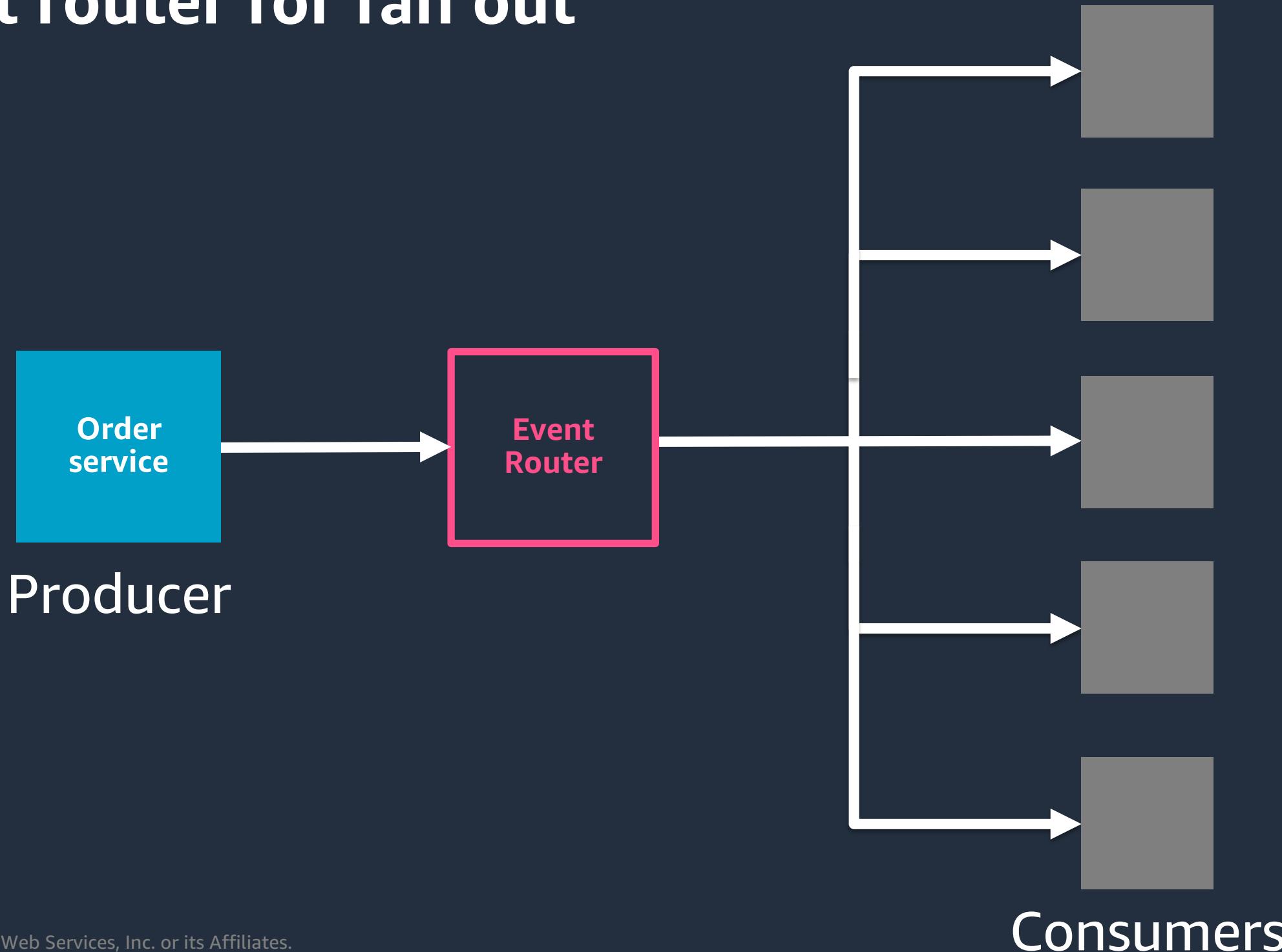
Event buses



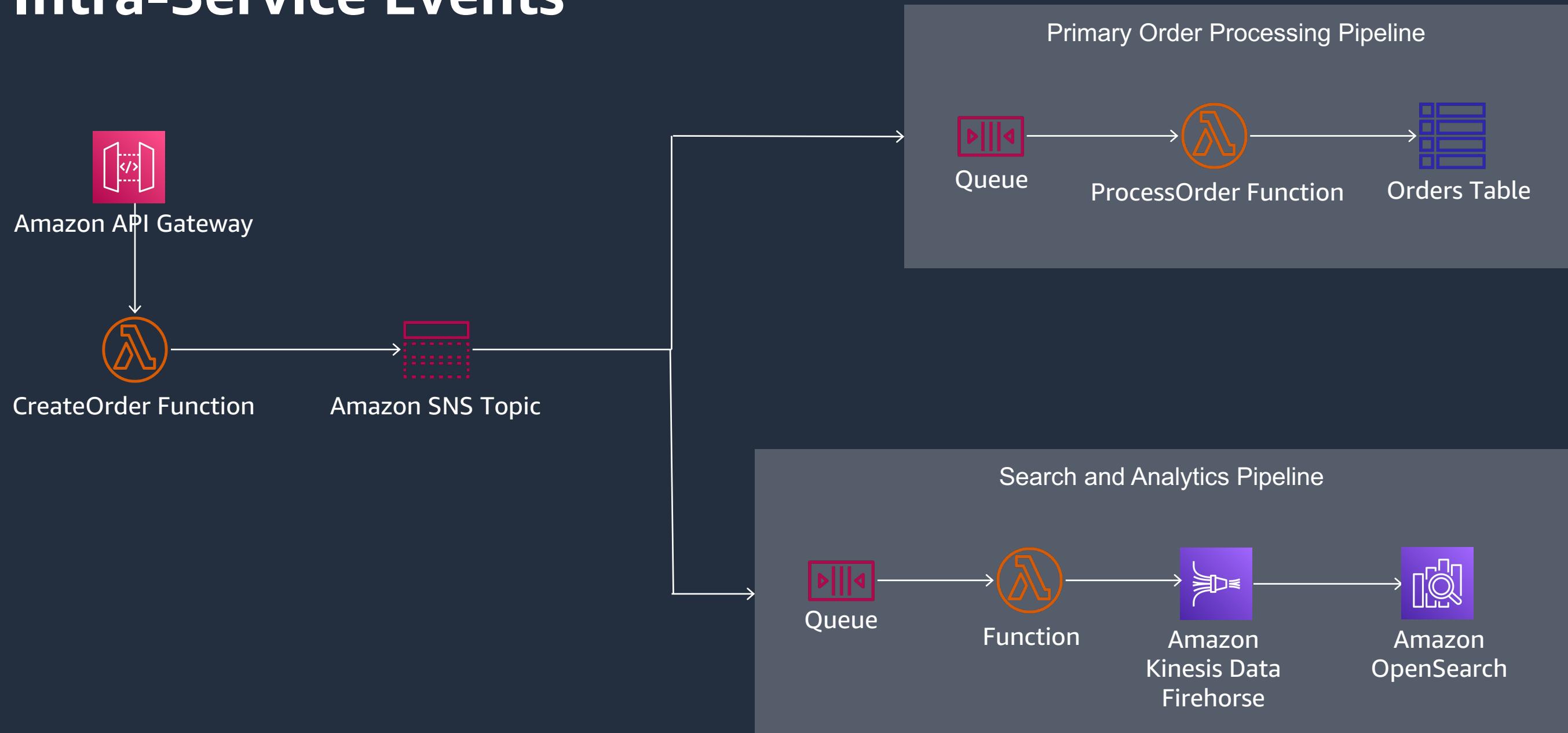
Amazon EventBridge

Over 35 targets
Native SaaS event sources
Metadata and payload routing

Event router for fan out



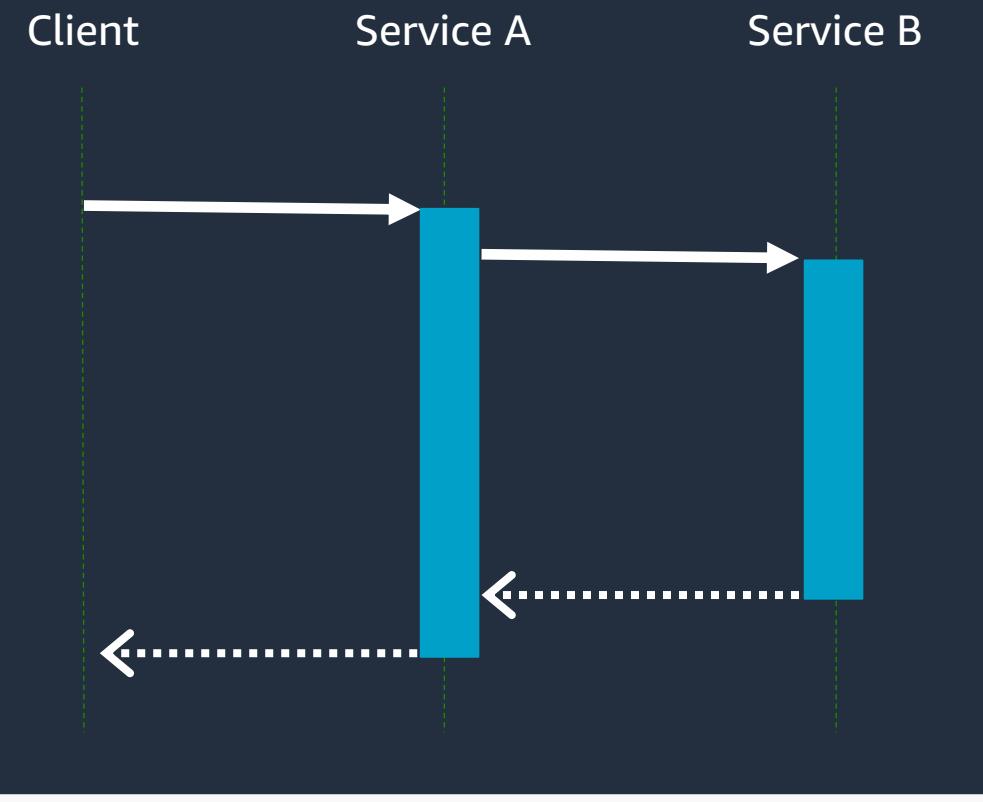
Intra-Service Events



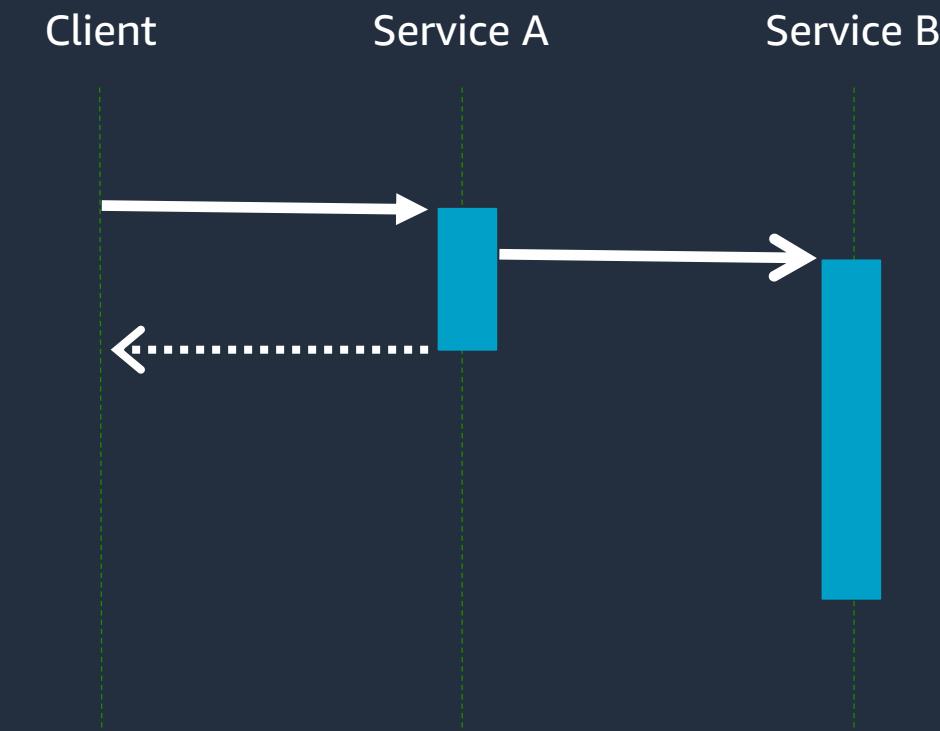
Downstream Failures



Events are asynchronous

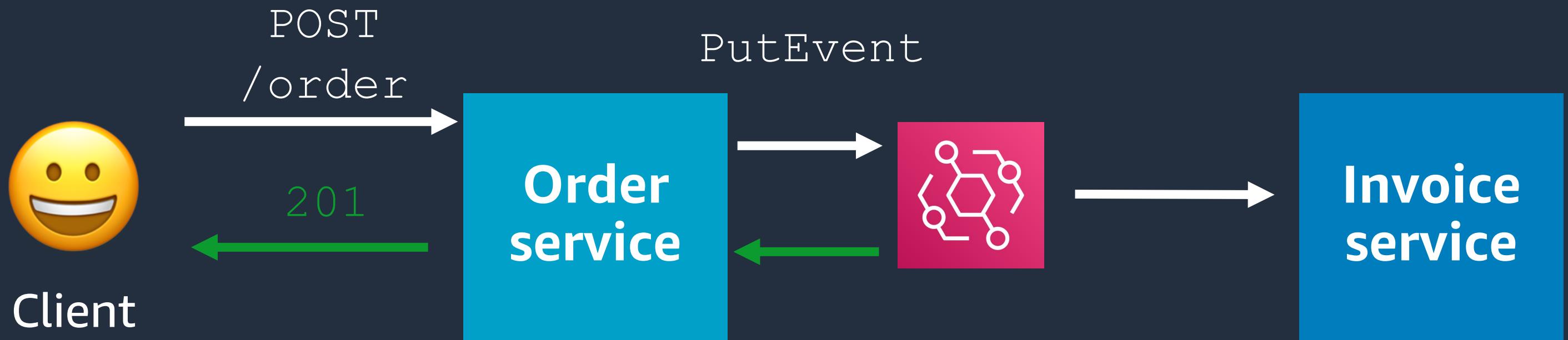


Synchronous
commands

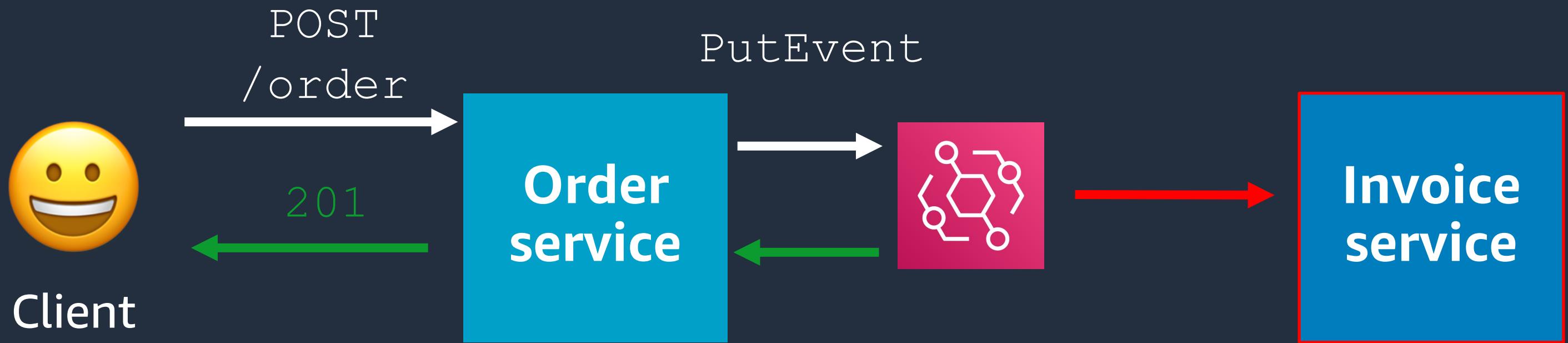


Asynchronous
events

Asynchronous events

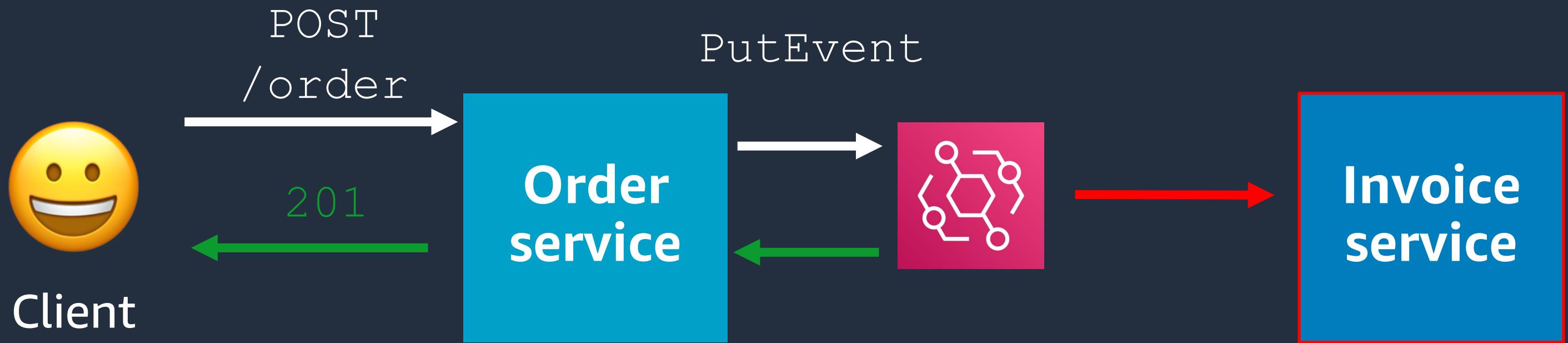


Asynchronous events

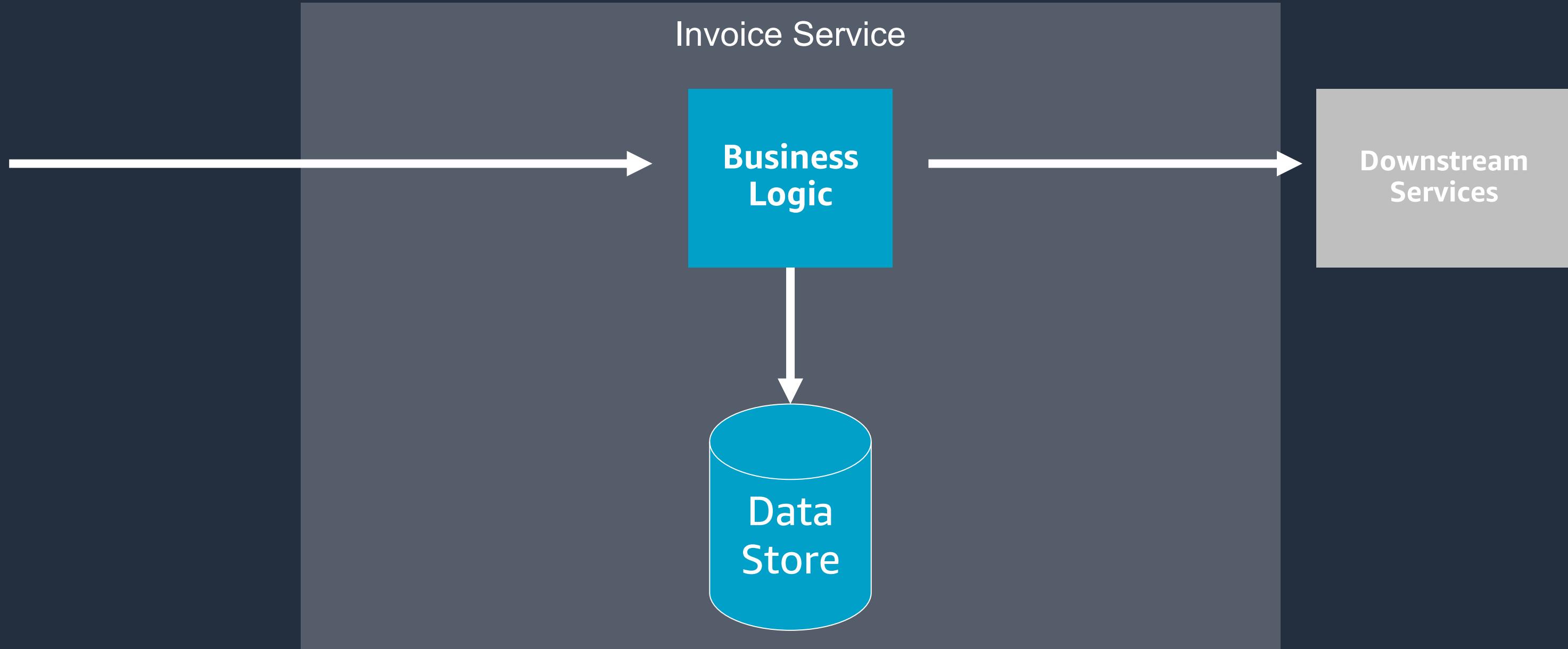


**But wait, now the customer
won't get their invoice!**

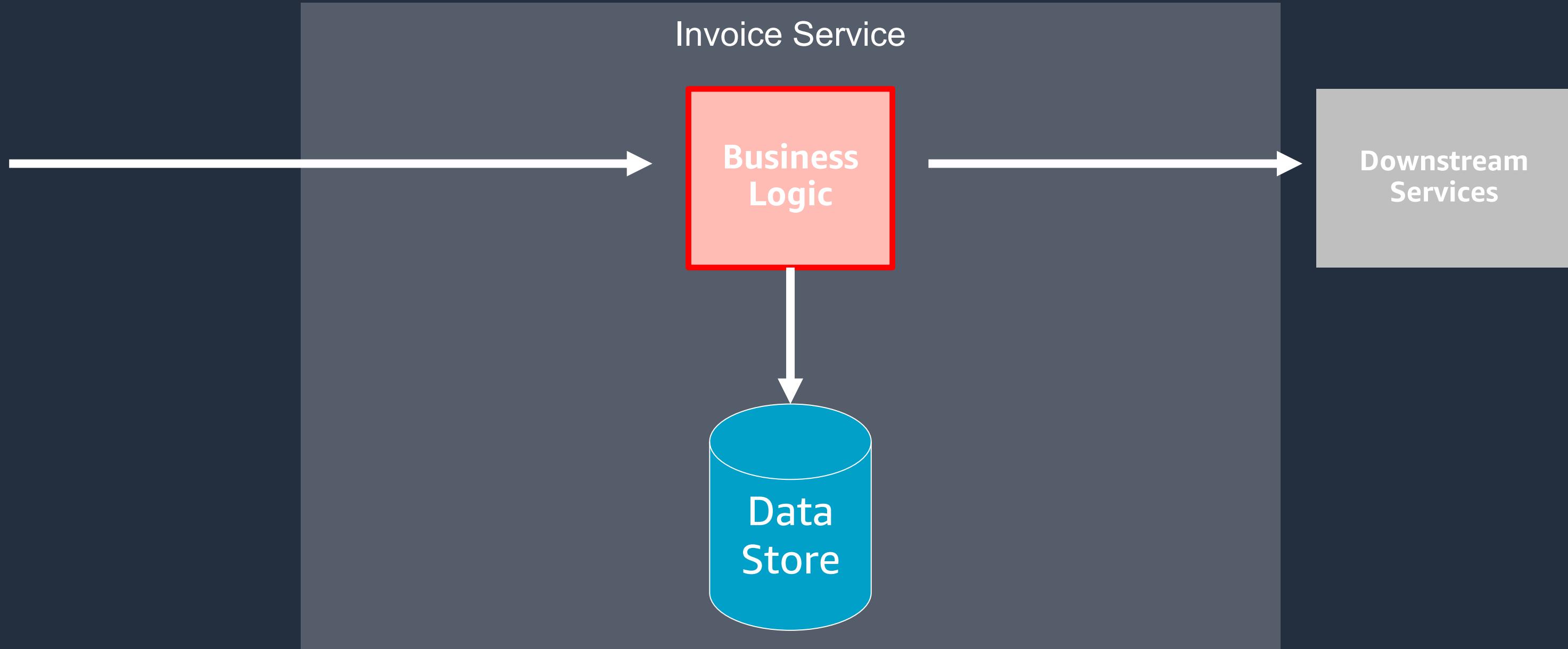
Asynchronous events



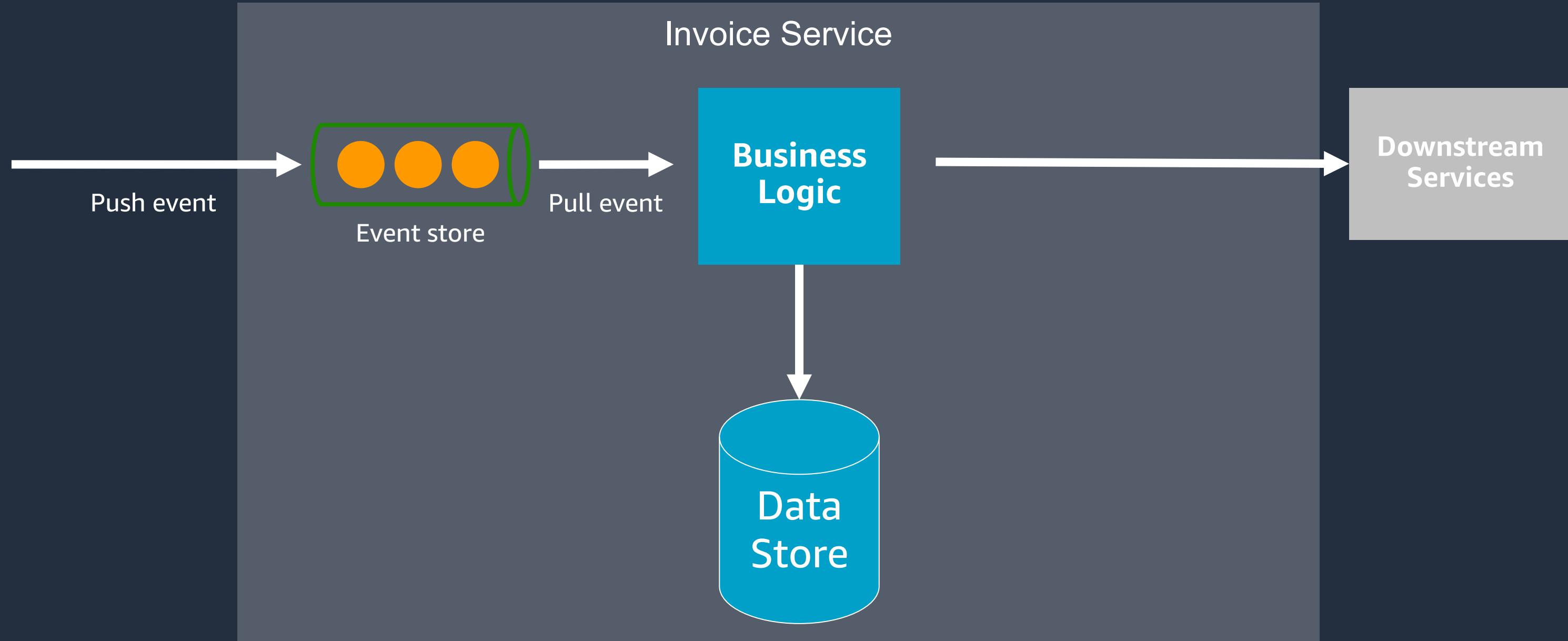
Inside the Invoice Service



Inside the Invoice Service



Inside the Invoice Service



Event Stores in AWS

AWS Native Service

Managed
Open Source

Queues



Amazon Simple
Queue Service



Amazon MQ

Streams

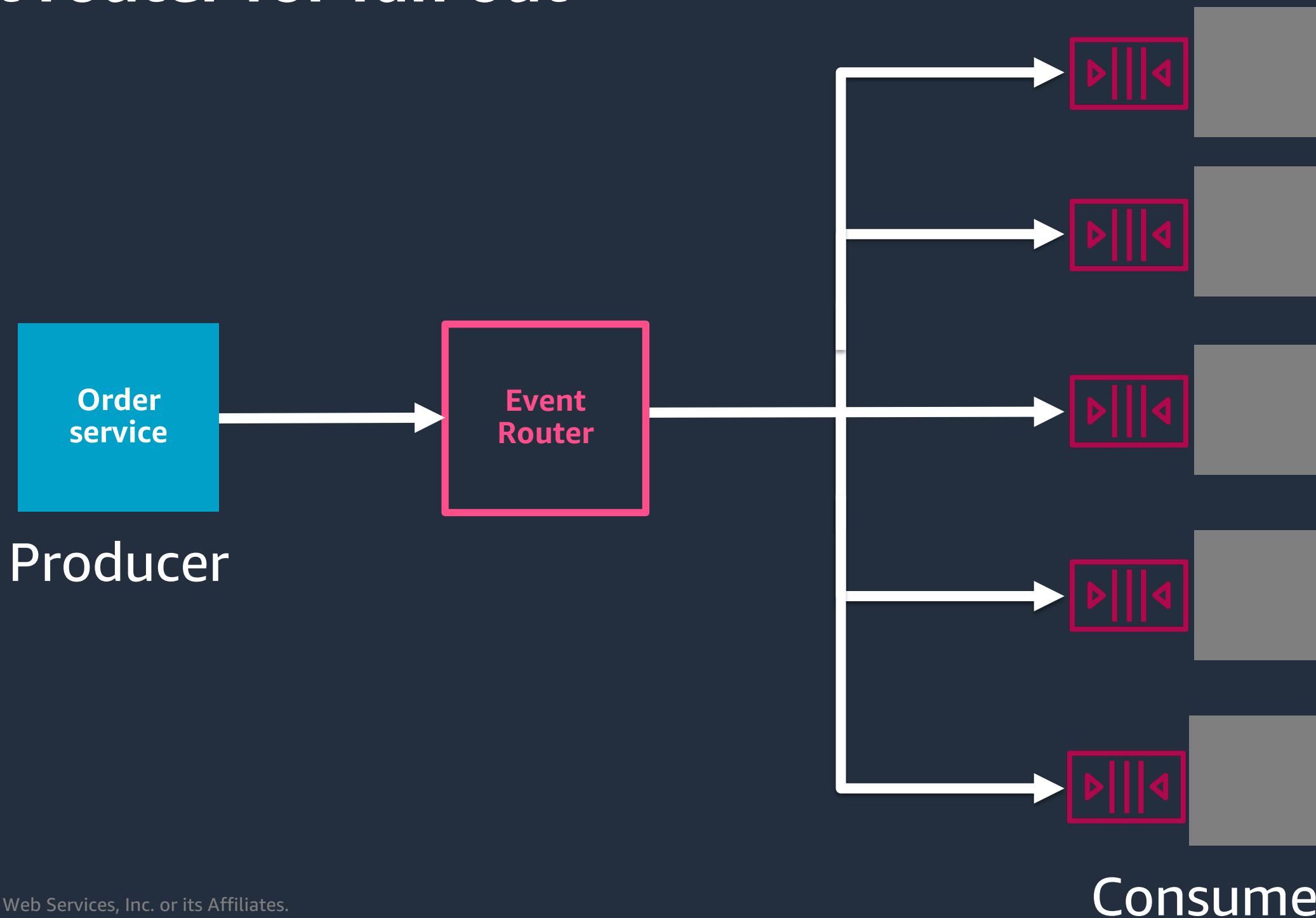


Amazon Kinesis
Data Streams



Amazon MSK

Event router for fan out



Structuring Events



Events vs. full state descriptions

Order 123 was created at 10:47 a.m. by customer 456



Events

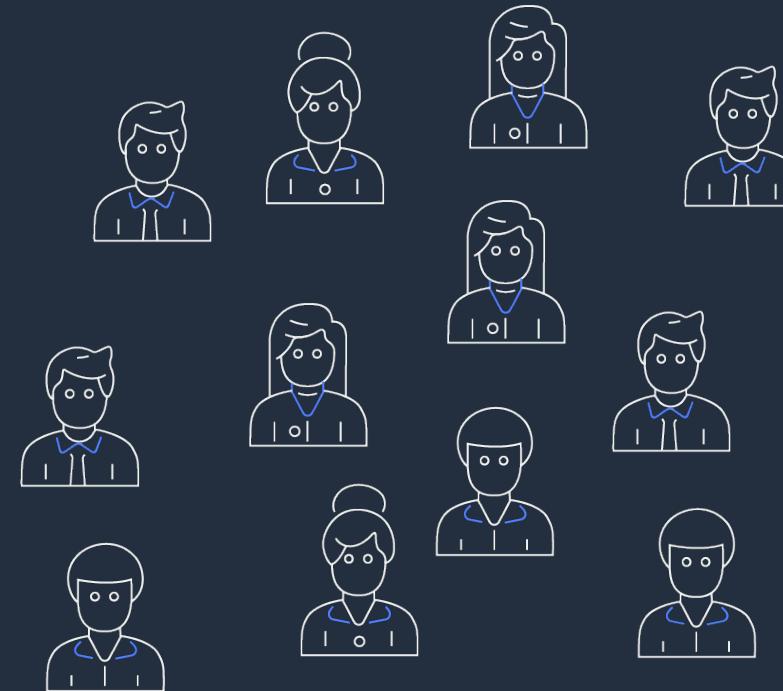
Order 123 was created at 10:47 a.m. by customer 456. The current status is Open, the total was \$237.51, the items were ...



Full state description

Considerations with sparse events

Order 123 was created by customer 456



What are the details for order 123?

Versioning events

```
{  
  "version": "1.0",  
  "productId": "7ba35e6b",  
  "salesPrice": 39.95,  
  "costPrice": 18.76  
}
```

Original Version

```
{  
  "version": "2.0",  
  "productId": "7ba35e6b",  
  "salesPrice": 3995,  
  "costPrice": 1876  
}
```

New Version

Anatomy of an event

```
{  
  "version": "0",  
  "id": "adeacade-c34c-ce58-c4a0-74f106398c4e",  
  "account": "123456789012",  
  "region": "us-east-1",  
  "time": "2019-12-02T21:46:19Z",  
  "source": "order-service",  
  "detail-type": "New Order",  
  "resources": [],  
  "detail": {  
    "orderId": "cfb2ae566f9b",  
    "customerId": "C12345",  
    ...  
  }  
}
```

The diagram illustrates the structure of an AWS Lambda event. It shows a JSON object with various fields. A blue curly brace on the right side groups the object into two main sections: 'Envelope metadata' (the top part containing fields like id, account, region, time, source, detail-type, resources, and detail) and 'Payload' (the bottom part containing the detail object with orderId and customerId).

Envelope metadata

Payload

Anatomy of an event

```
{  
  "version": "0",  
  "id": "adeacade-c34c-ce58-c4a0-74f106398c4e",  
  "account": "123456789012",  
  "region": "us-east-1",  
  "time": "2019-12-02T21:46:19Z",  
  "source": "order-service",  
  "detail-type": "New Order",  
  "resources": [],  
  "detail": {  
    "orderId": "cfb2ae566f9b",  
    "customerId": "C12345",  
    ...  
  }  
}
```

Managed attributes

Anatomy of an event

```
{  
  "version": "0",  
  "id": "adeacade-c34c-ce58-c4a0-74f106398c4e",  
  "account": "123456789012",  
  "region": "us-east-1",  
  "time": "2019-12-02T21:46:19Z",  
  "source": "order-service",  
  "detail-type": "New Order",  
  "resources": [],  
  "detail": {  
    "orderId": "cfb2ae566f9b",  
    "customerId": "C12345",  
    ...  
  }  
}
```

Service that created the event

Anatomy of an event

```
{  
  "version": "0",  
  "id": "adeacade-c34c-ce58-c4a0-74f106398c4e",  
  "account": "123456789012",  
  "region": "us-east-1",  
  "time": "2019-12-02T21:46:19Z",  
  "source": "order-service",  
  "detail-type": "New Order",  
  "resources": [],  
  "detail": {  
    "orderId": "cfb2ae566f9b",  
    "customerId": "C12345",  
    ...  
  }  
}
```

Event type

Anatomy of an event

```
{  
  "version": "0",  
  "id": "adeacade-c34c-ce58-c4a0-74f106398c4e",  
  "account": "123456789012",  
  "region": "us-east-1",  
  "time": "2019-12-02T21:46:19Z",  
  "source": "order-service",  
  "detail-type": "New Order",  
  "resources": [],  
  "detail": {  
    "orderId": "cfb2ae566f9b",  
    "customerId": "C12345",  
    ...  
  }  
}
```

Any valid JSON object

Design Considerations



Delivery semantics

At-least once delivery

Events can be delivered to a target more than once. Include logic to detect duplicate events by tracking the state of processed events (**idempotent**).

- Amazon EventBridge
- Amazon SNS Standard
- Amazon SQS Standard

Exactly-once delivery*

Specify an identifier used by the AWS service for deduplication.

- Amazon MQ
- Amazon MSK
- Amazon Kinesis
- Amazon SQS FIFO

** outside of application error handling*

Ordering semantics*

Unordered

Events can be delivered out of order.

- Amazon EventBridge
- Amazon SNS Standard
- Amazon SQS Standard

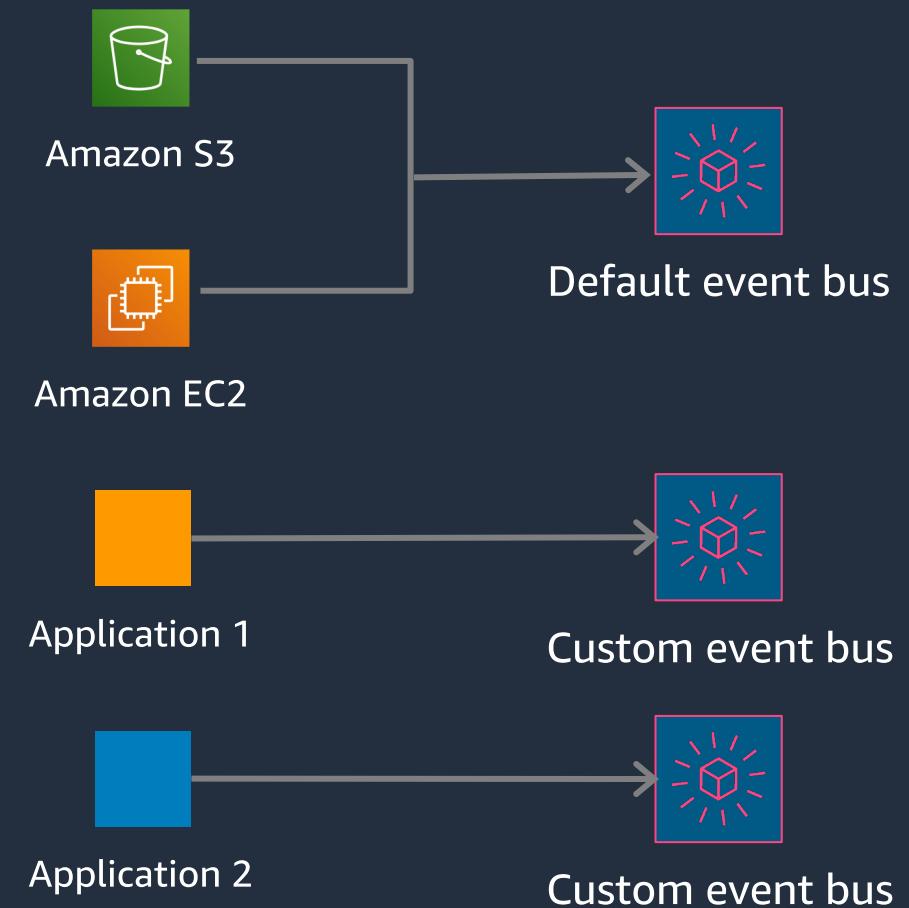
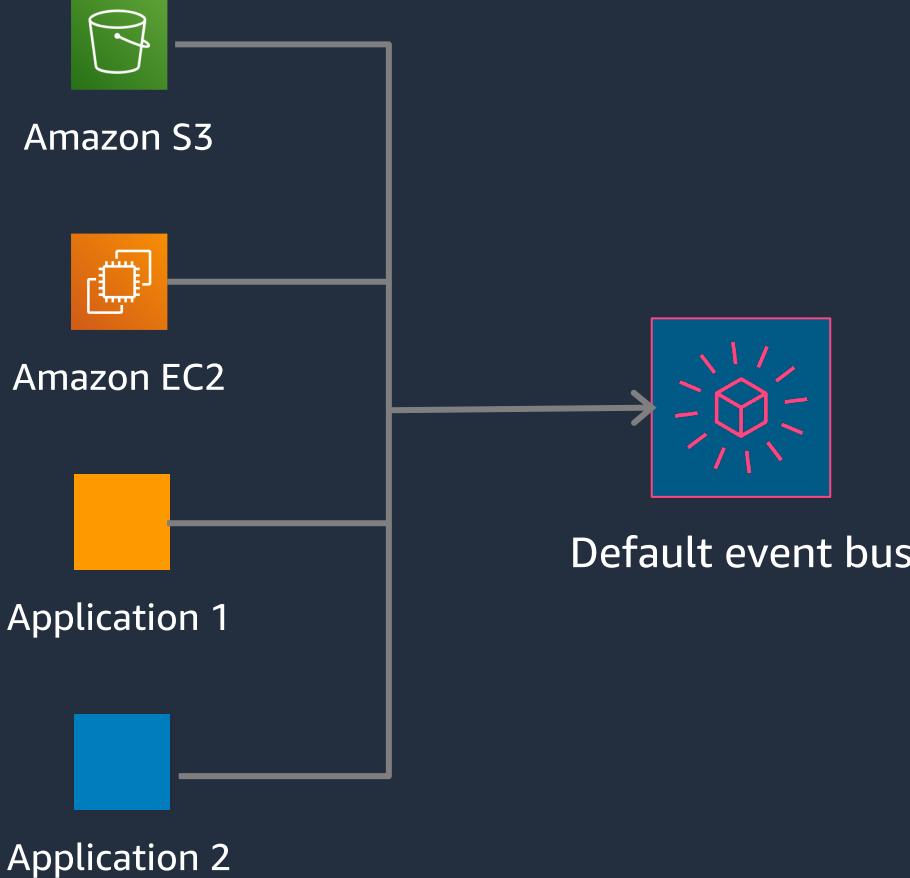
Ordered

Events are delivered in order within a partition, message group, etc (no global order).

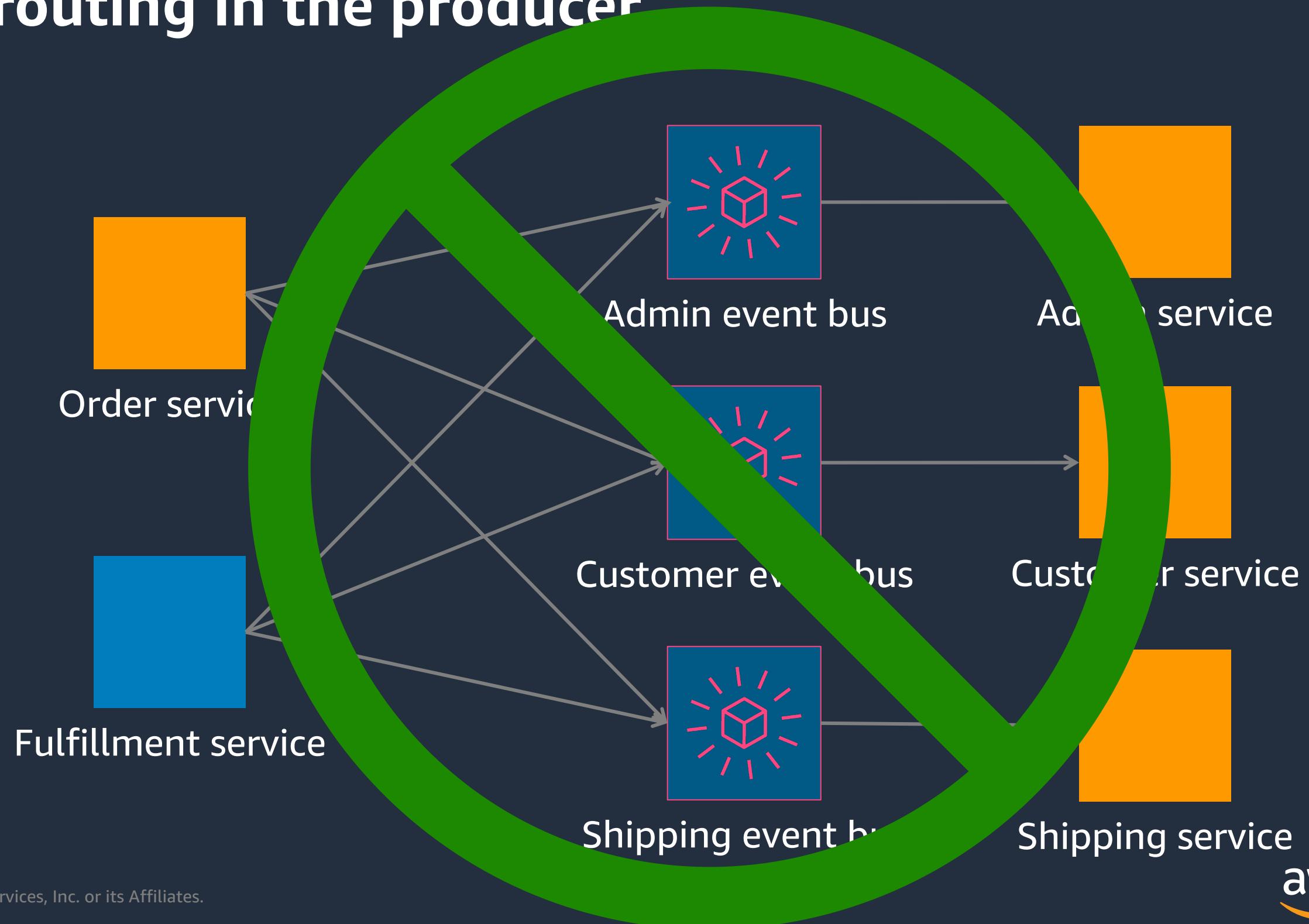
- Amazon MQ
- Amazon MSK
- Amazon Kinesis
- Amazon SQS FIFO

** Out-of-order event handling logic is highly application specific. If the application cannot be designed to handle out-of-order events, consider orchestration instead.*

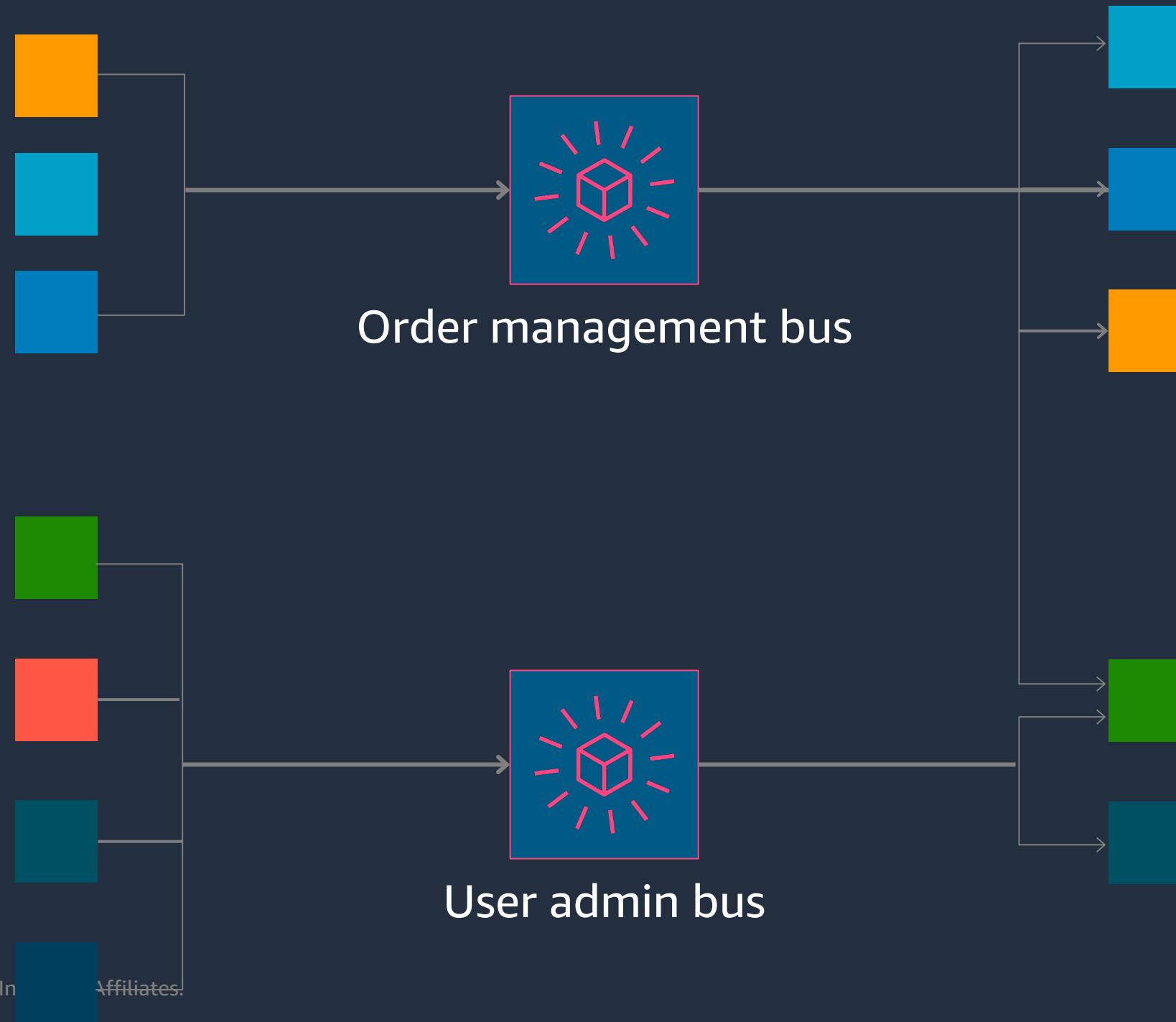
How many event buses should I have?



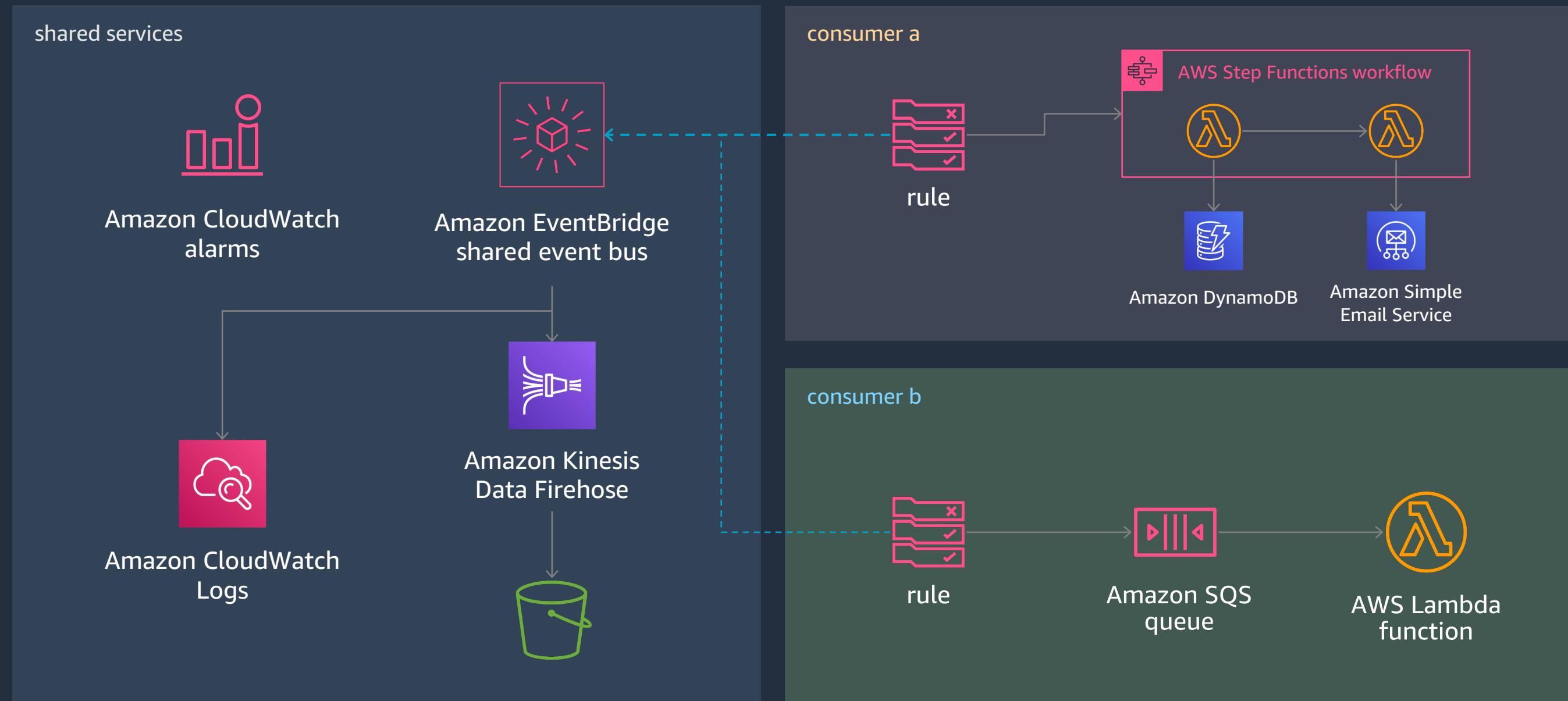
Avoid routing in the producer



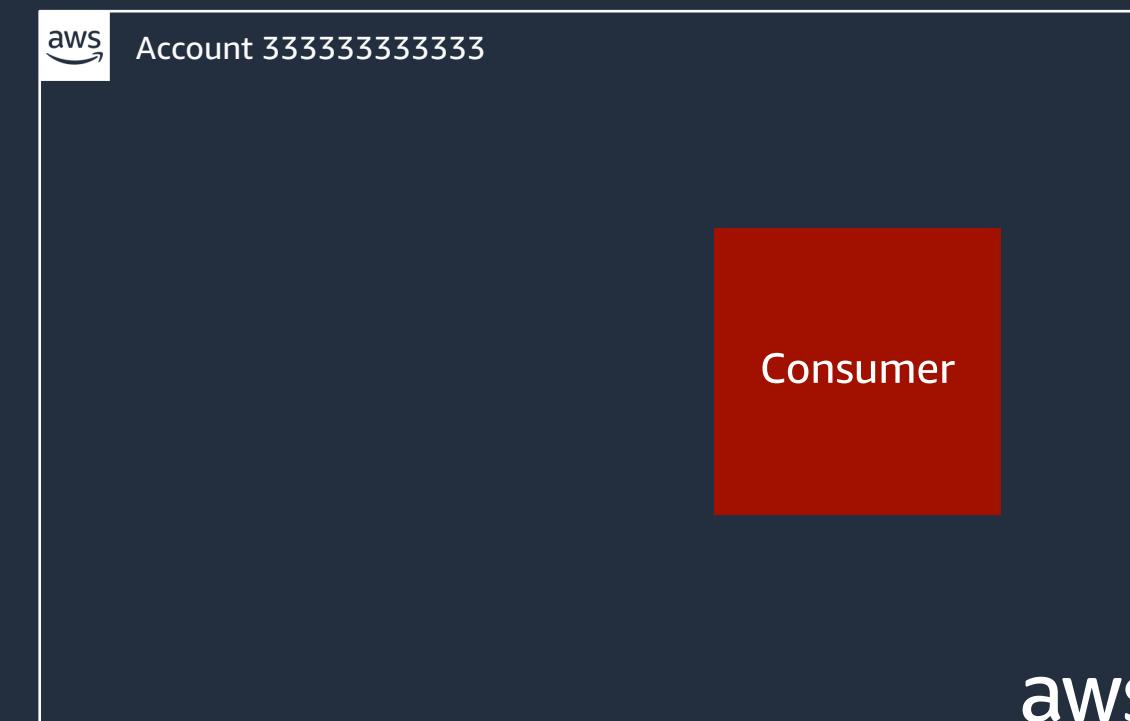
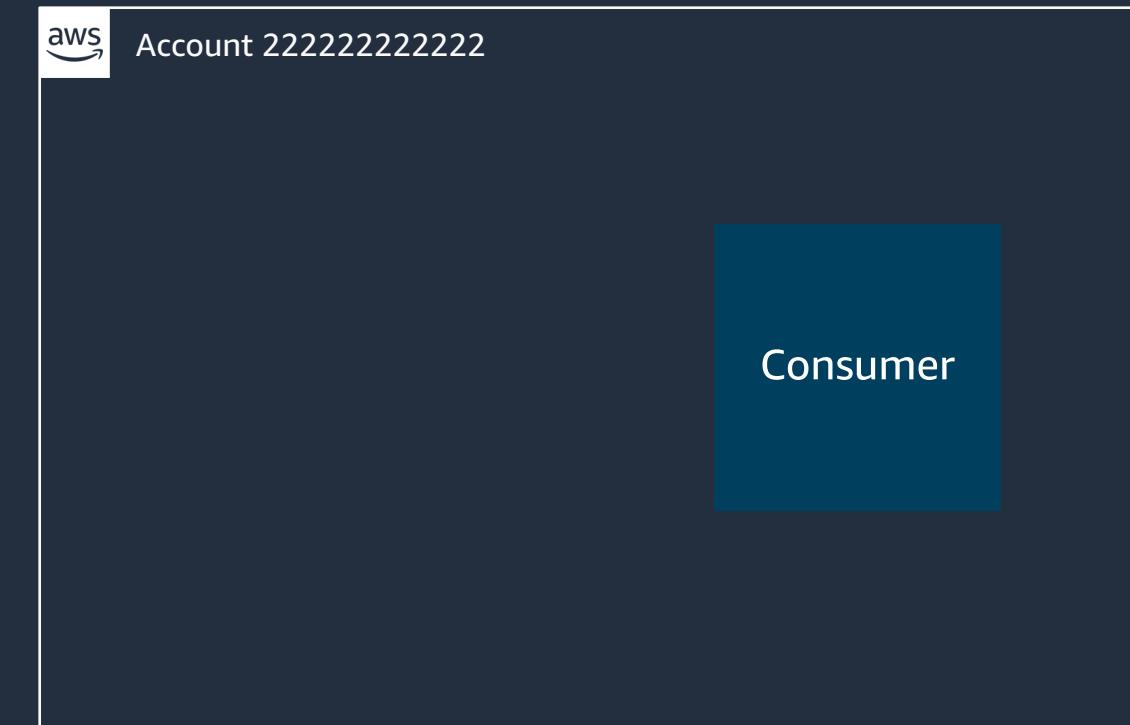
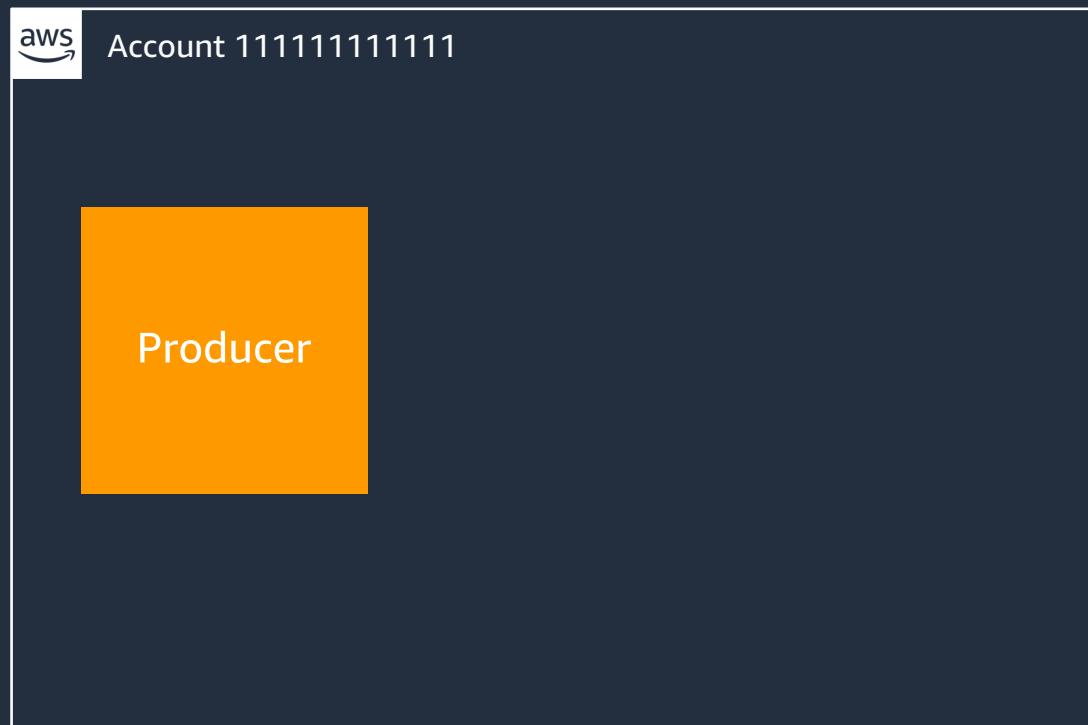
Event bus domain alignment



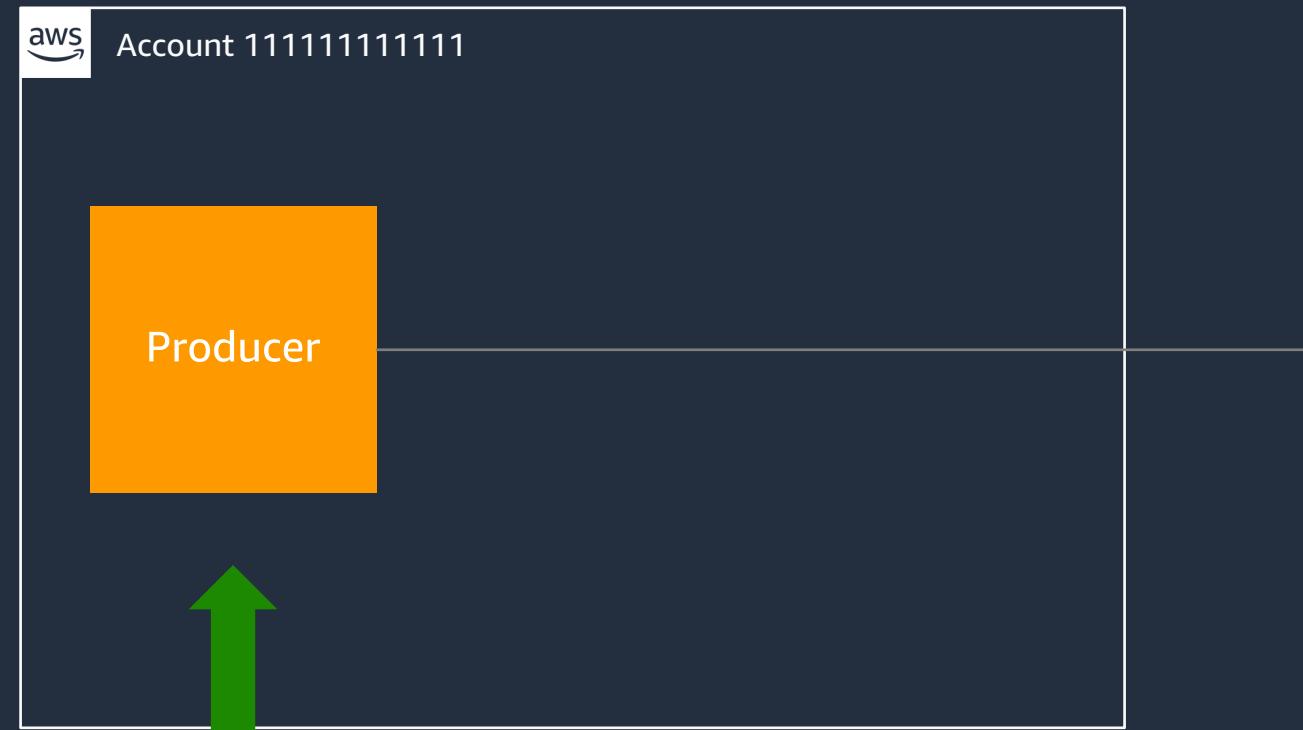
Event bus provisioning



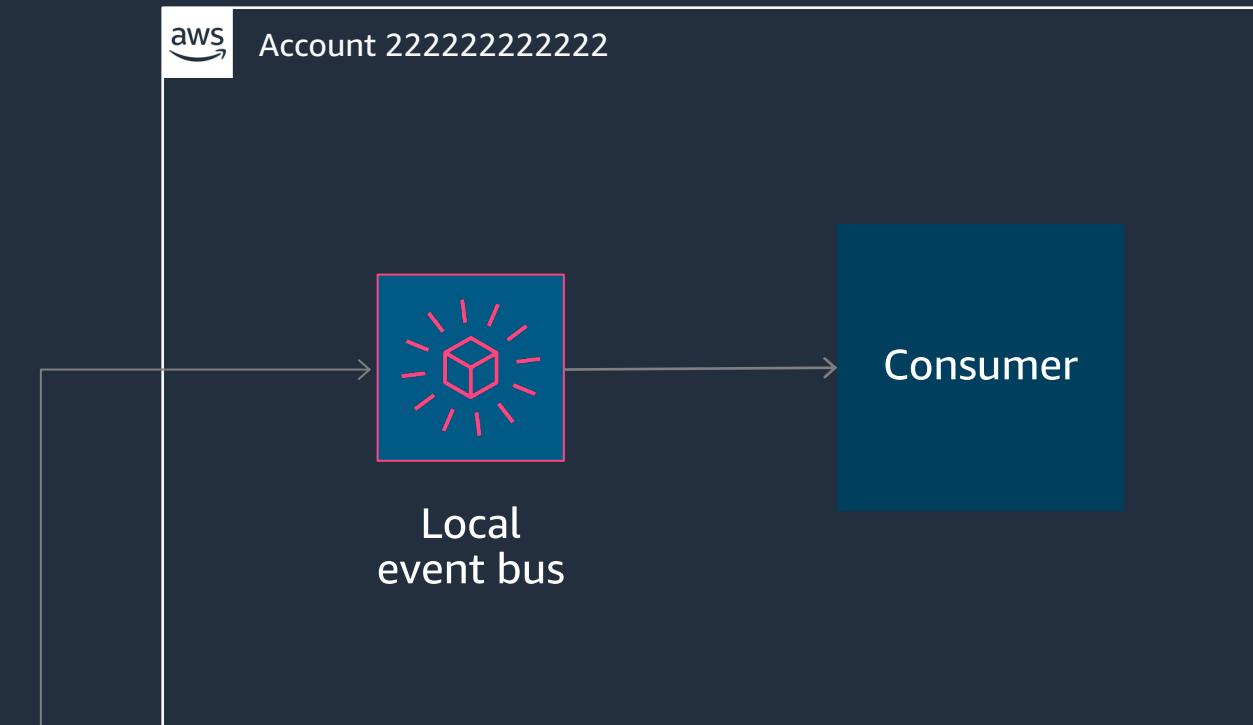
Cross-account events



Cross-account events

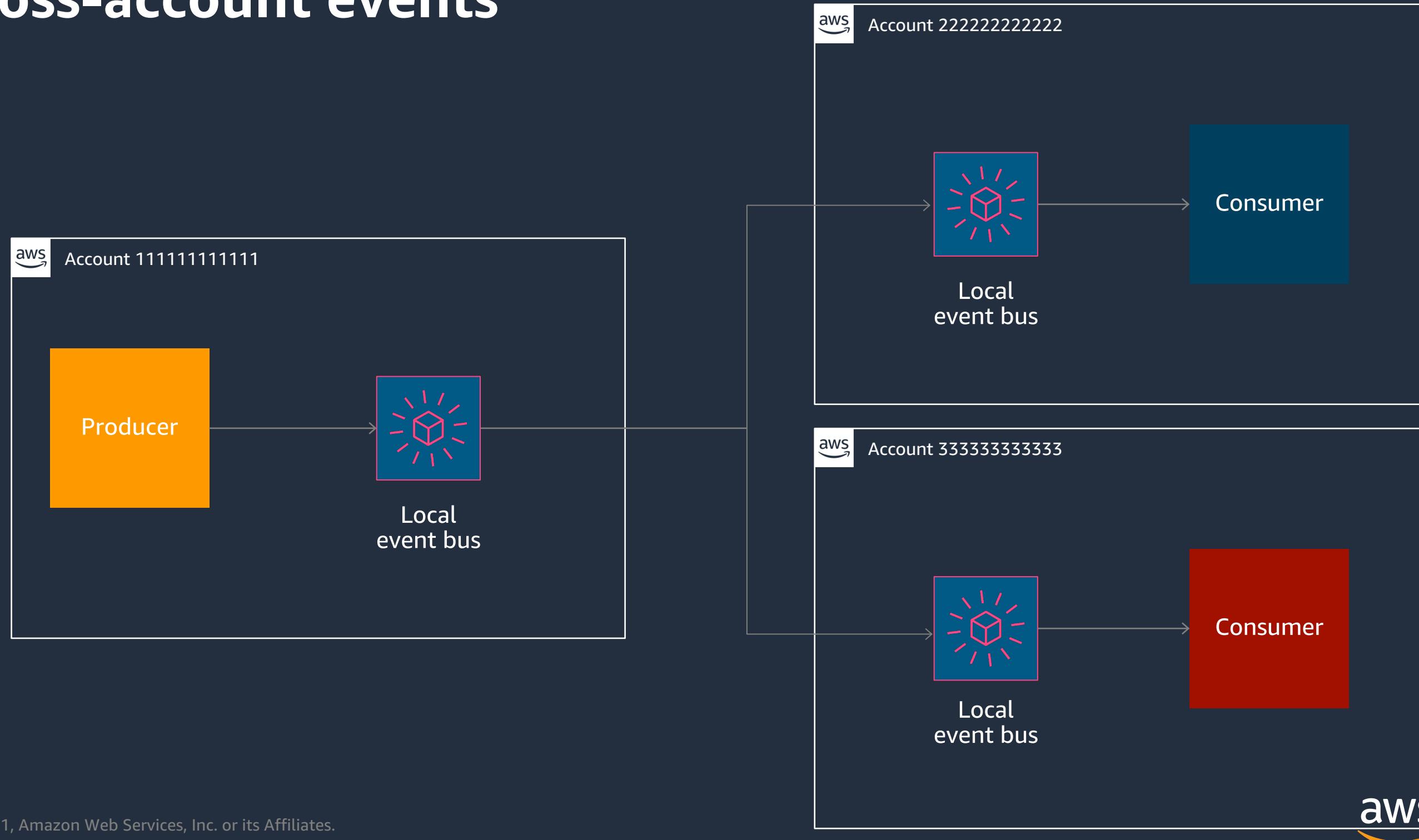


Requires routing in
the producer



Local
event bus

Cross-account events



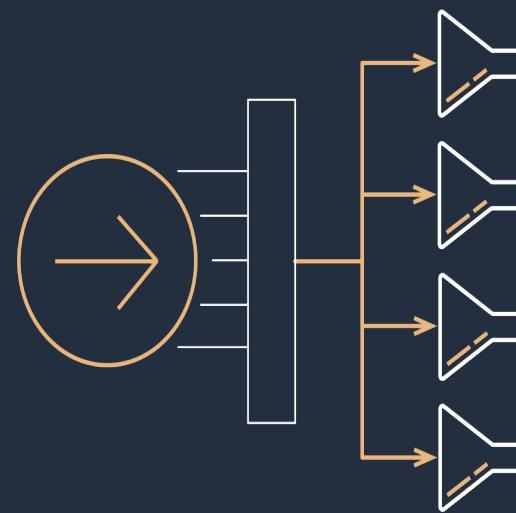
Demo



Conclusion

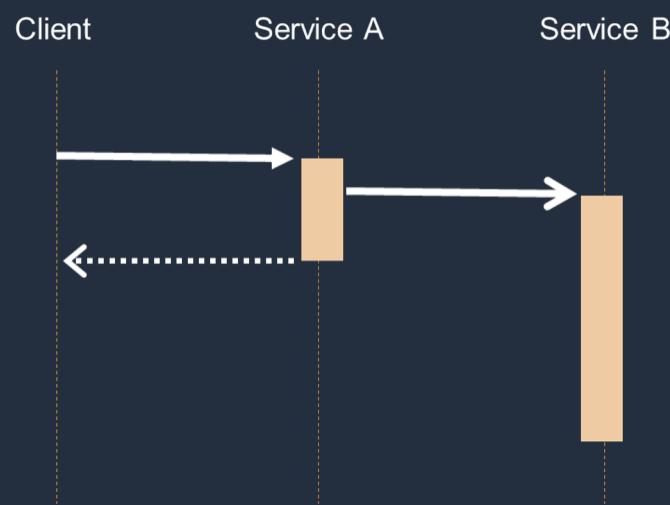


Event-driven architectures drive reliability and scalability



Event Routers

Abstract producers and consumers from each other



Asynchronous Events

Improve responsiveness and reduce dependencies



Event Stores

Buffer messages until services are available to process

Event-driven benefits



Scale and fail independently



Develop with agility



Audit with ease



Cut costs

Thanks!

