AWS Lambda

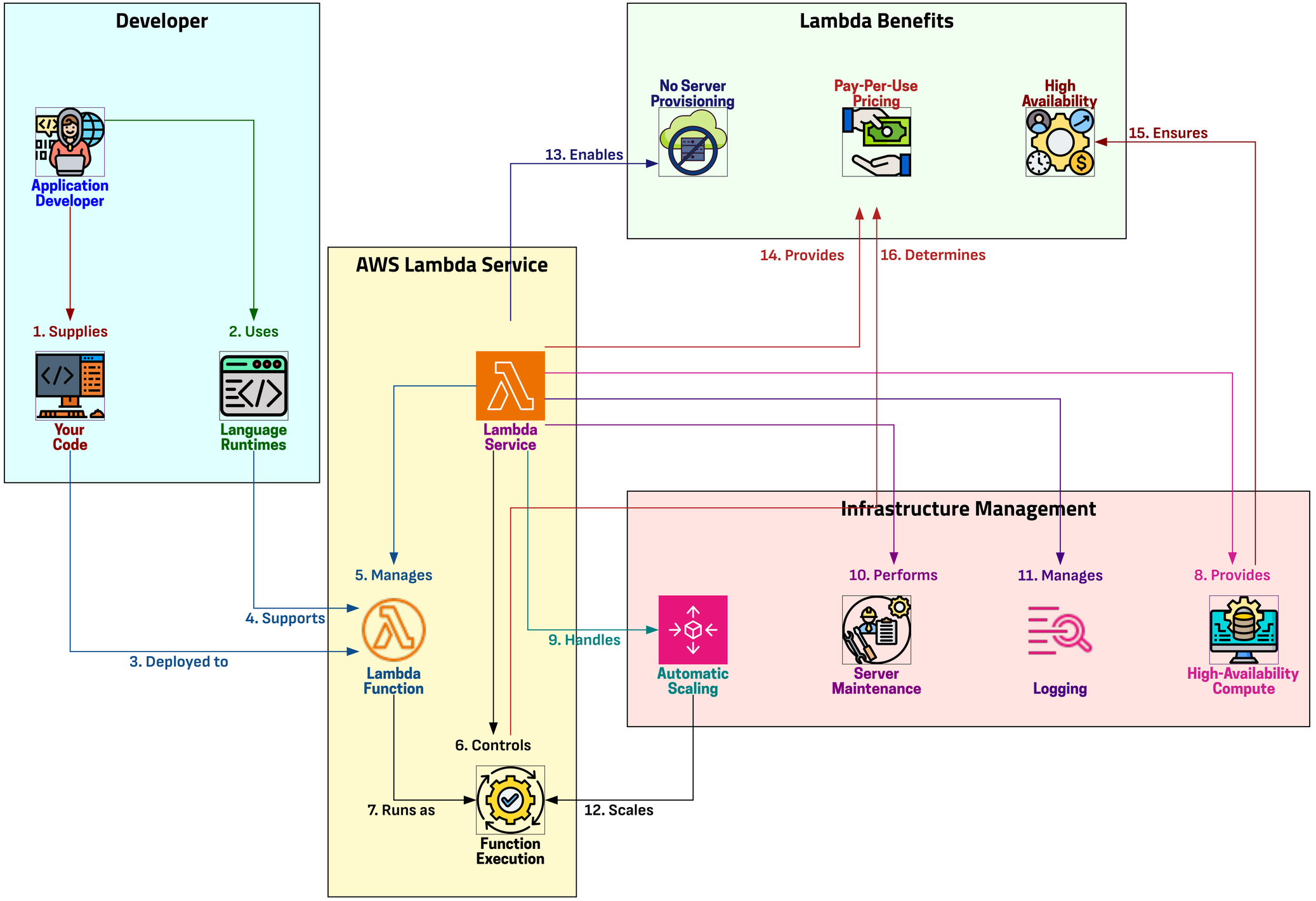
Table of Contents

What is AWS Lambda?

1️⃣. 💻 Serverless Code Execution

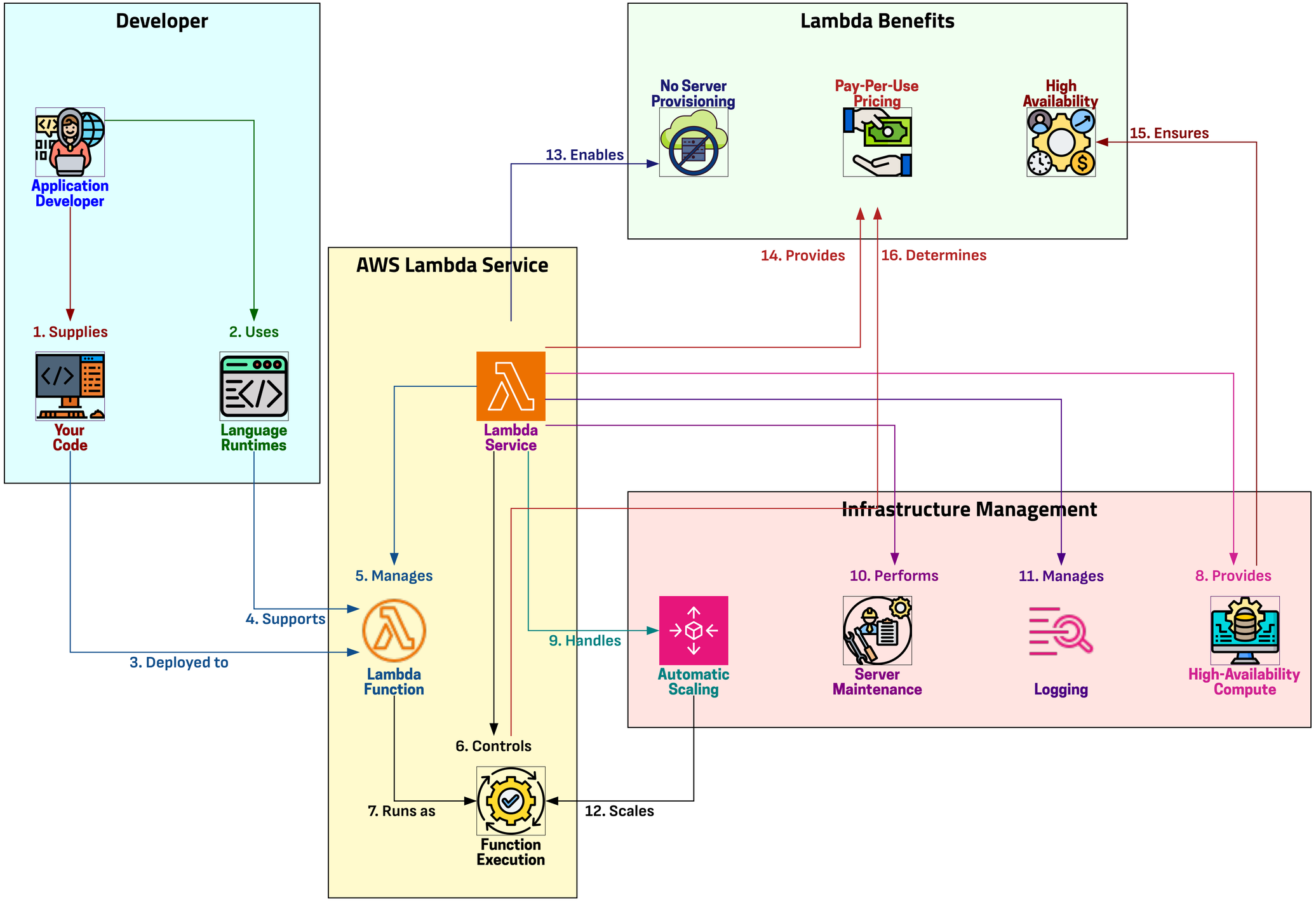
🖥️ Run code without servers

☁️ Serverless architecture



What is AWS Lambda?

2️⃣. 🛠️ Managed Infrastructure
⚙️ Runs on reliable systems
🔄 Handles server maintenance
🔼 Manages OS updates
📊 Provides capacity
⚖️ Automatic scaling
📝 Log tracking

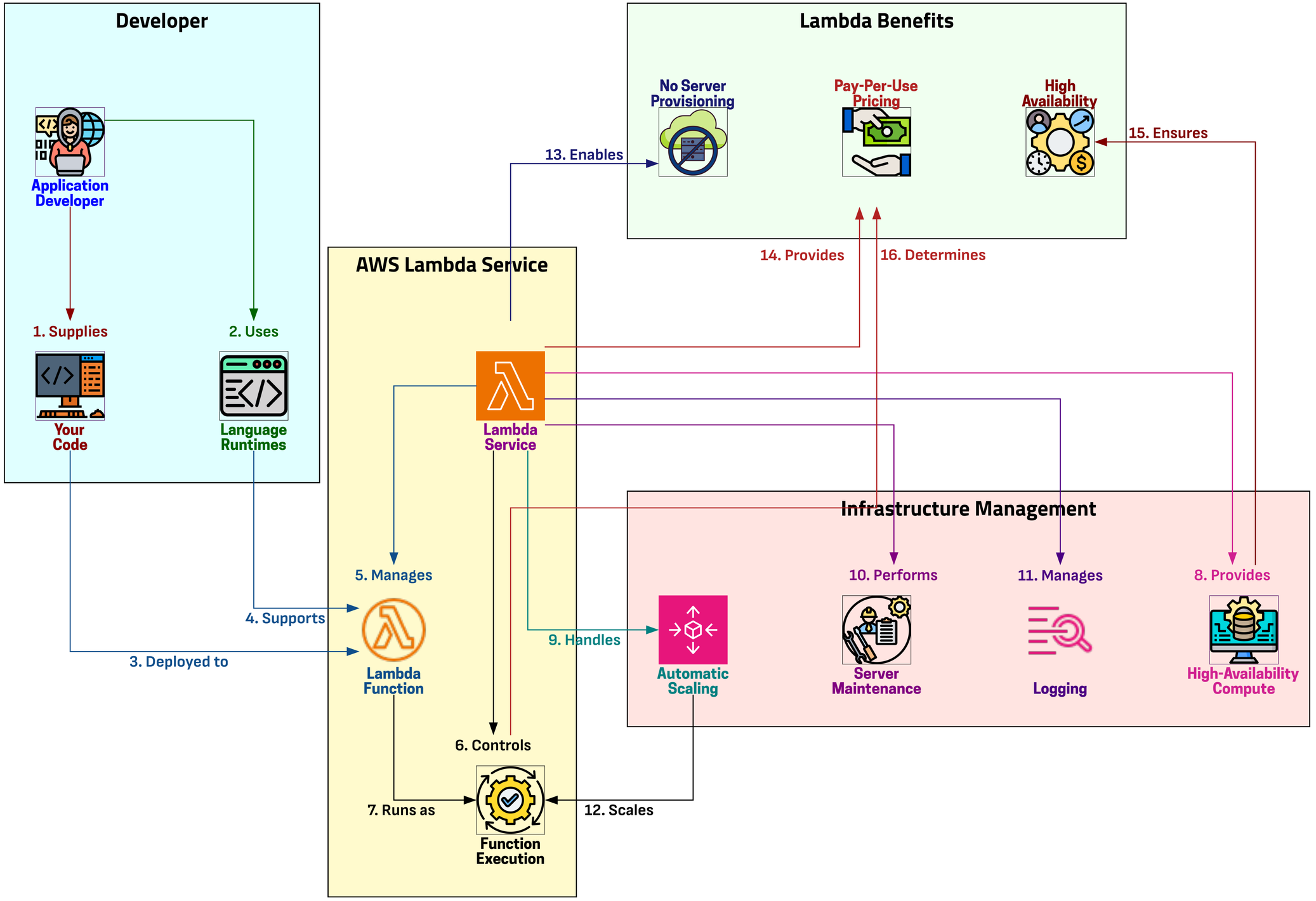


What is AWS Lambda?

3️⃣. ✍️ Provide Your Code

📄 Write in supported languages

📨 Upload your code

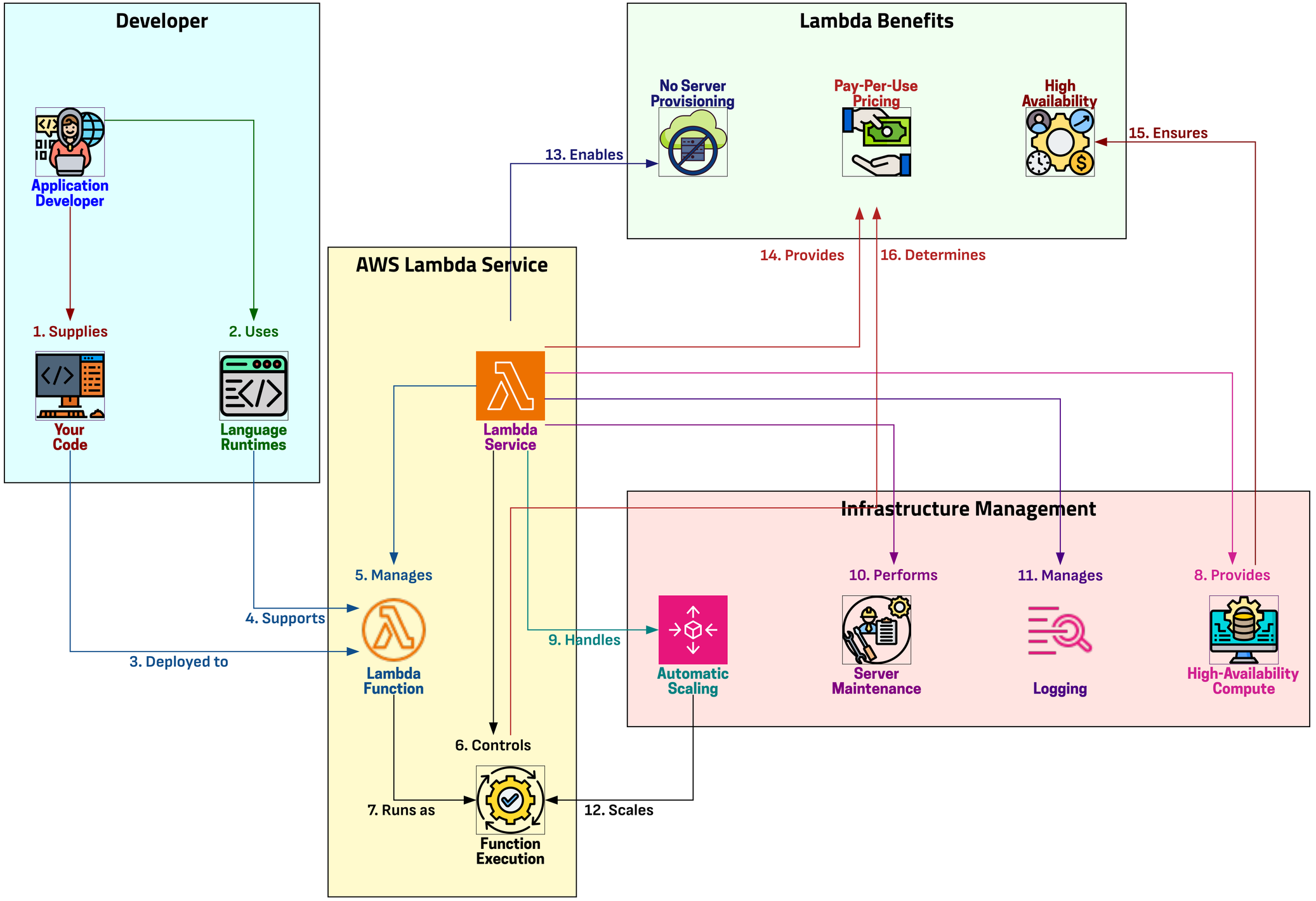


What is AWS Lambda?

4️⃣. 📦 Lambda Functions

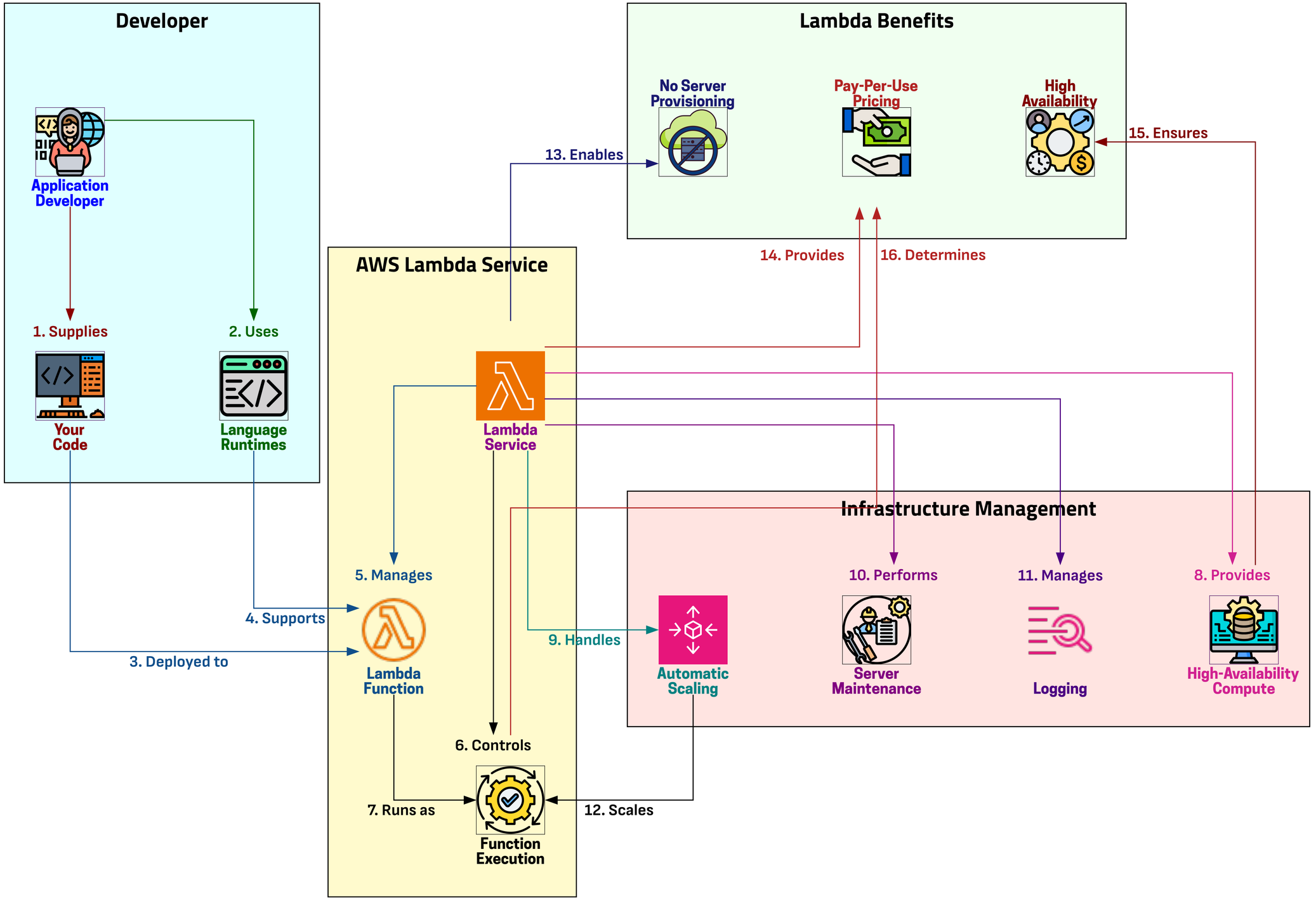
🧩 Organized code units

🏗️ Structured approach



What is AWS Lambda?

5️⃣. 🚀 On-Demand & Auto-Scaling
🔍 Runs when needed
📈 Adjusts to demand
🔧 Handles any load size

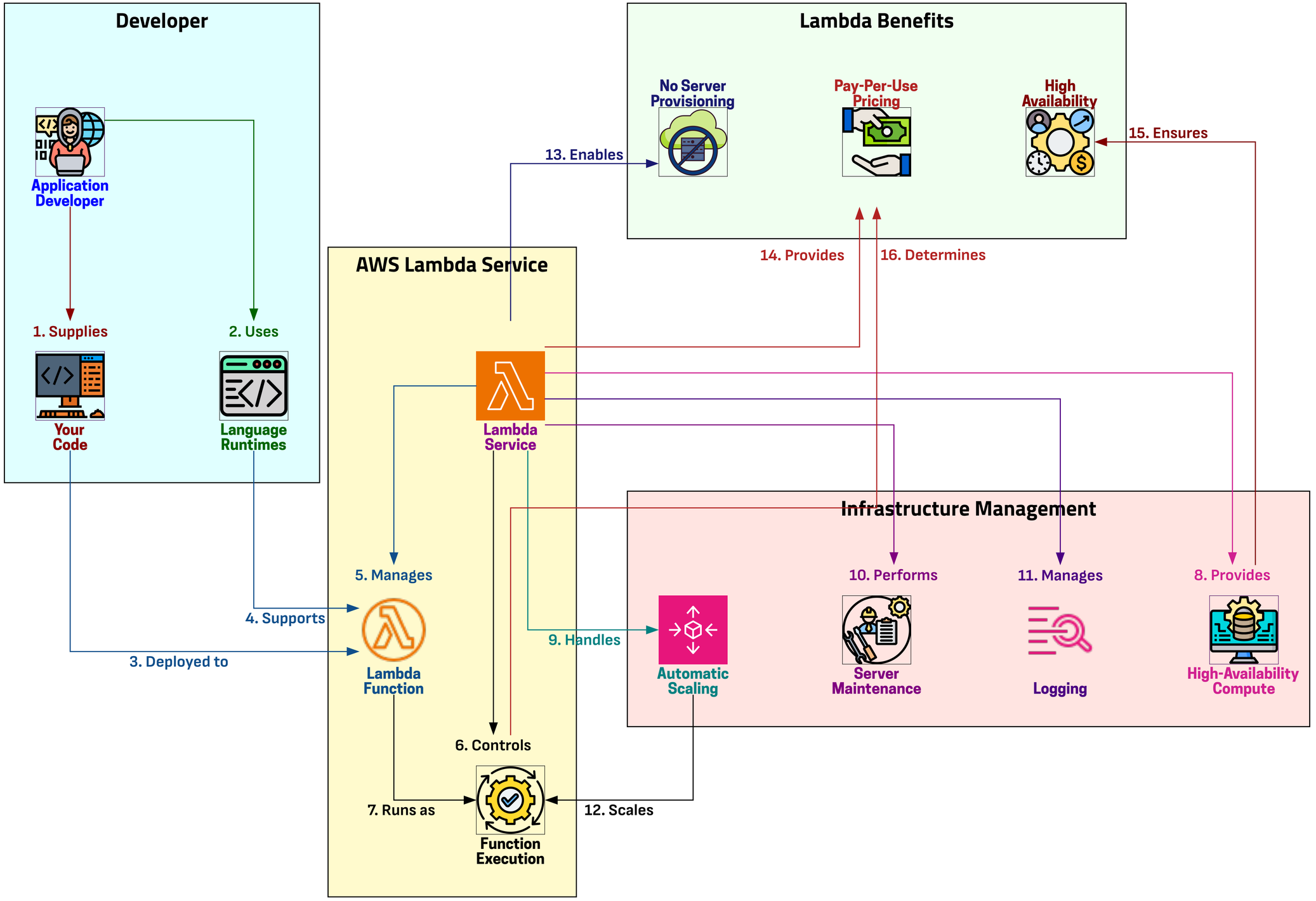


What is AWS Lambda?

6. 💸 Pay-Per-Use

⏱️ Charged for compute time

🆓 No cost when idle



When to use Lambda

1

When to use Lambda: Ideal for rapid scaling scenarios, Scales down to zero when not in demand

2

File processing: Uses Amazon S3 as trigger, Real-time data processing after upload

When to use Lambda

3

 **Stream processing**:  Works with Amazon Kinesis,  Processes real-time streaming data,  Handles clickstream analysis,  Performs data cleansing

4

 **Web applications**:  Combines with other AWS services,  Automatically scales up and down,  Runs in highly available configuration

When to use Lambda

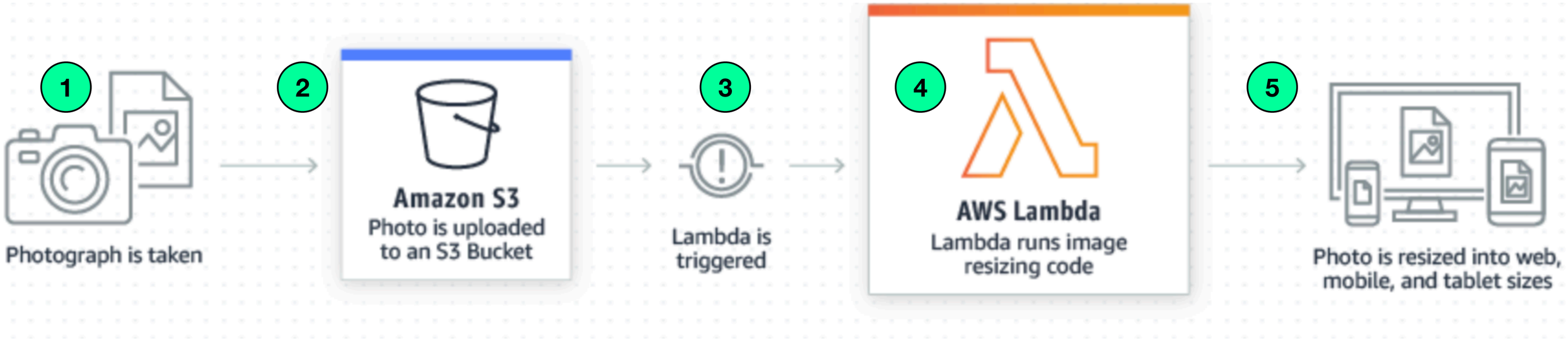
5

IoT backends: Builds serverless backends, Handles various request types, Processes third-party API requests

6

Mobile backends: Uses Amazon API Gateway for authentication, Integrates with AWS Amplify, Supports multiple frontend platforms

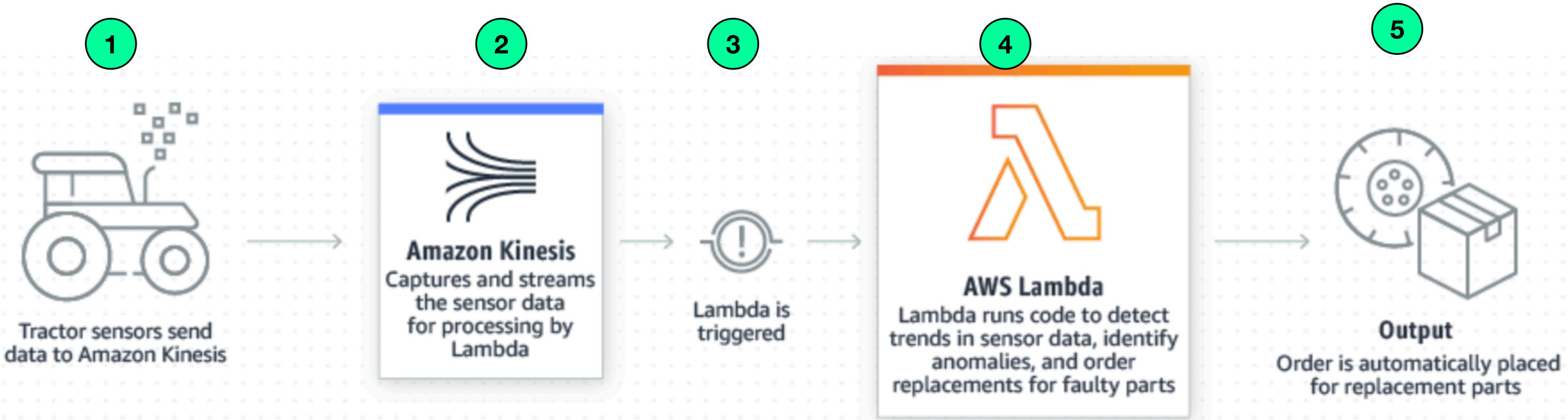
File Processing Example



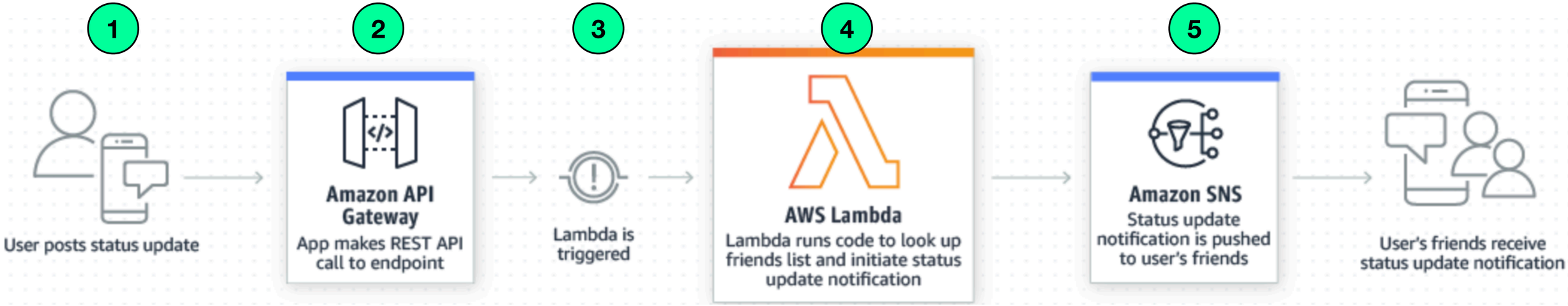
Streaming Example

Web applications Example

IoT Backends Example



Mobile Backends Example



Key Features

1. 🔧 Environment variables
2. 🔢 Versions
3. 🐋 Container images
4. 📚 Lambda layers
5. 🔌 Lambda extensions
6. 🔗 Function URLs
7. 📲 Response streaming
8. ⚖️ Concurrency and scaling controls
9. ✅ Code signing
10. 🔒 Private networking
11. 📁 File system
12. ⚡ Lambda SnapStart

```
import json
import logging
```

Sample Lambda Function

Sample Lambda Function

```
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

Sample Lambda Function

```
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):









    }
```

Sample Lambda Function

```
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    }
```

Sample Lambda Function

```
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    Returns:
    - dict: Response containing greeting and execution details
    """









    }
```

Sample Lambda Function

```python
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    Returns:
    - dict: Response containing greeting and execution details
    """
    logger.info(f"Received event: {json.dumps(event)}")









    }
```

Sample Lambda Function

```
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    Returns:
    - dict: Response containing greeting and execution details
    """
    logger.info(f"Received event: {json.dumps(event)}")

    # Extract name from event if provided, otherwise use default
    name = event.get('name', 'AWS Student')









    }
```

Sample Lambda Function

```python
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    Returns:
    - dict: Response containing greeting and execution details
    """
    logger.info(f"Received event: {json.dumps(event)}")

    # Extract name from event if provided, otherwise use default
    name = event.get('name', 'AWS Student')

    # Return a simple response
    return {
        'statusCode': 200,
        'body': f"Hello {name}! Welcome to AWS Lambda.",
        'functionName': context.function_name,
        'remainingTime': context.get_remaining_time_in_millis()
    }
```

Sample Lambda Function

```python
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Basic AWS Lambda function

    Parameters:
    - event (dict): Input data passed to the function
    - context (LambdaContext): Runtime information provided by AWS Lambda

    Returns:
    - dict: Response containing greeting and execution details
    """
    logger.info(f"Received event: {json.dumps(event)}")

    # Extract name from event if provided, otherwise use default
    name = event.get('name', 'AWS Student')

    # Return a simple response
    return {
        'statusCode': 200,
        'body': f"Hello {name}! Welcome to AWS Lambda.",
        'functionName': context.function_name,
        'remainingTime': context.get_remaining_time_in_millis()
    }
```

Configure test event

```
{
  "name": "John Doe"
}
```

Test Result


```
{
  "statusCode": 200,
  "body": "Hello John Doe! Welcome to AWS Lambda.",
  "functionName": "YourFunctionName",
  "remainingTime": 12345
}
```

Sample Lambda Function - 2

```
import json
import logging
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")




def calculate_area(length, width):
    return length*width
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")




def calculate_area(length, width):
    return length*width
```

Sample Lambda Function - 2

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

Sample Lambda Function - 2

```python
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

Test


```
{
  "length": 6,
  "width": 7
}
```

Test Result

```
Status: Succeeded
Test Event Name: myTestEvent

Response
"{\"area\": 42}"

Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO]  2024-08-31T23:43:26.428Z  2d0b1579-46fb-4bf7-a6e1-8e08840eae5b  CloudWatch logs group: /aws/lambda/myLambdaFunction
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b  Duration: 1.42 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 39 MB  Init Duration:
123.74 ms

Request ID
2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
```

Advanced Features in AWS Lambda

1. **Scaling**: Automatically scales with triggers, Ensures high availability, Increases or decreases execution as needed

2. **Concurrency Controls**: Manages simultaneous executions, Optimizes performance, Prevents resource overconsumption

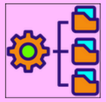
Advanced Features in AWS Lambda

3  **Function URLs**:  Dedicated HTTP URLs,  Direct function invocation,  Simplifies web application integration

4  **Asynchronous Invocation**:  Improves throughput,  Reduces latency,  Queues messages before processing,  Enhances reliability

Advanced Features in AWS Lambda

5  **Event Source Mappings**:  DynamoDB integration,  Kinesis integration,  SQS integration,  Enables real-time data processing

6  **Destinations**:  Post-execution data routing,  SNS integration,  Lambda function chaining,  Seamless data flow

Advanced Features in AWS Lambda

7

 **Function Blueprints**:  Predefined templates,  Quick start for common use cases,  Reduces development time

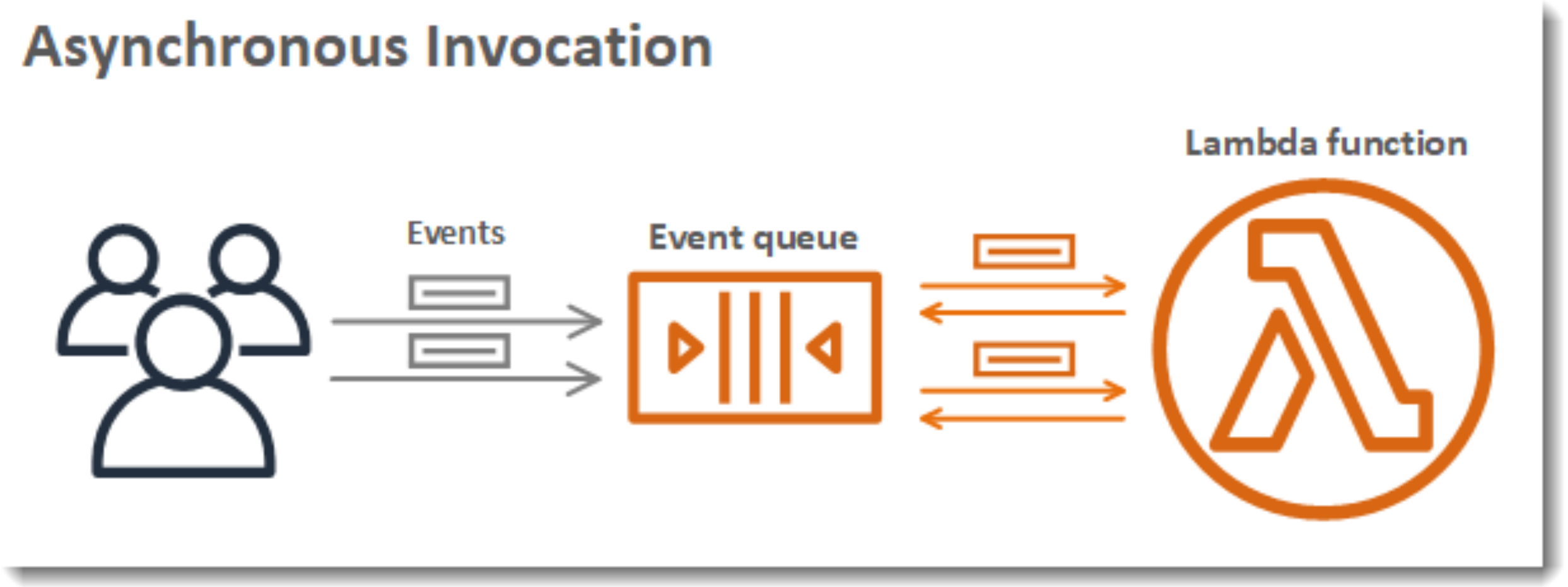
8

 **Testing and Deployment Tools**:  Integrated testing tools,  Efficient deployment capabilities,  Ensures reliability,  Promotes scalability

Advanced Features in AWS Lambda

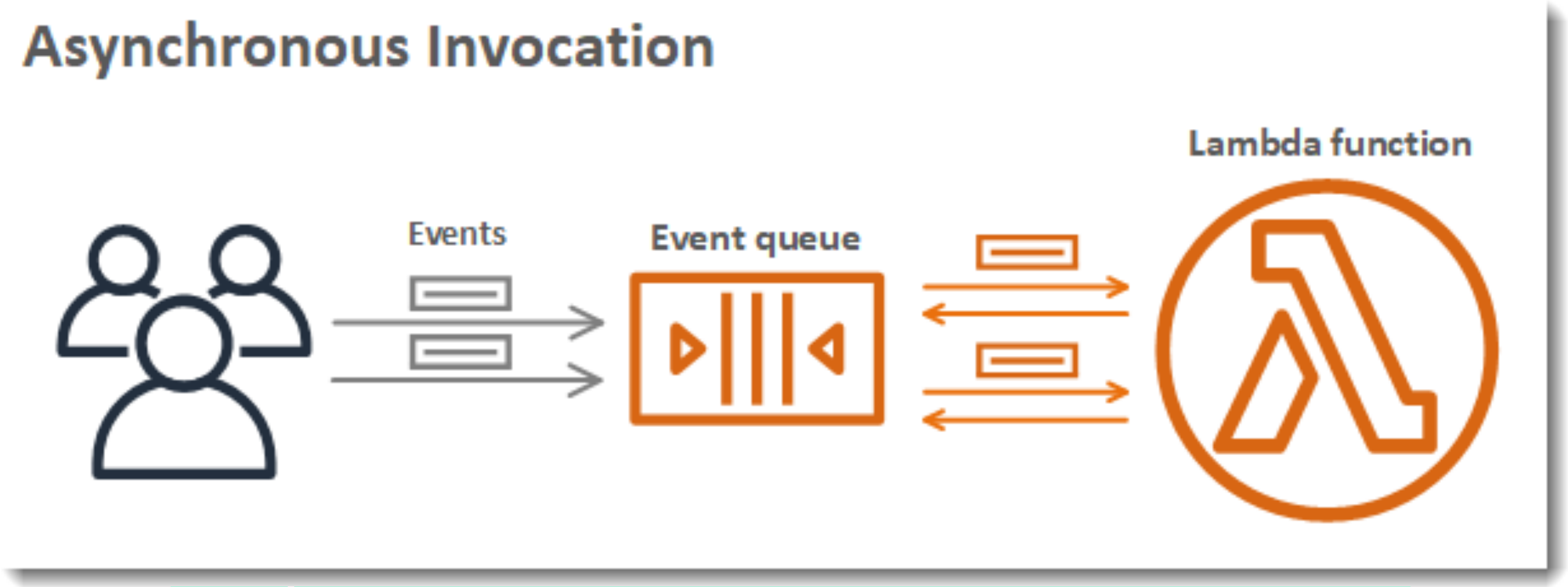
9	**Application Templates**: Pre-configured resource sets, Ready-to-use Lambda functions, Accelerates development, Speeds up deployment

Asynchronous invocation



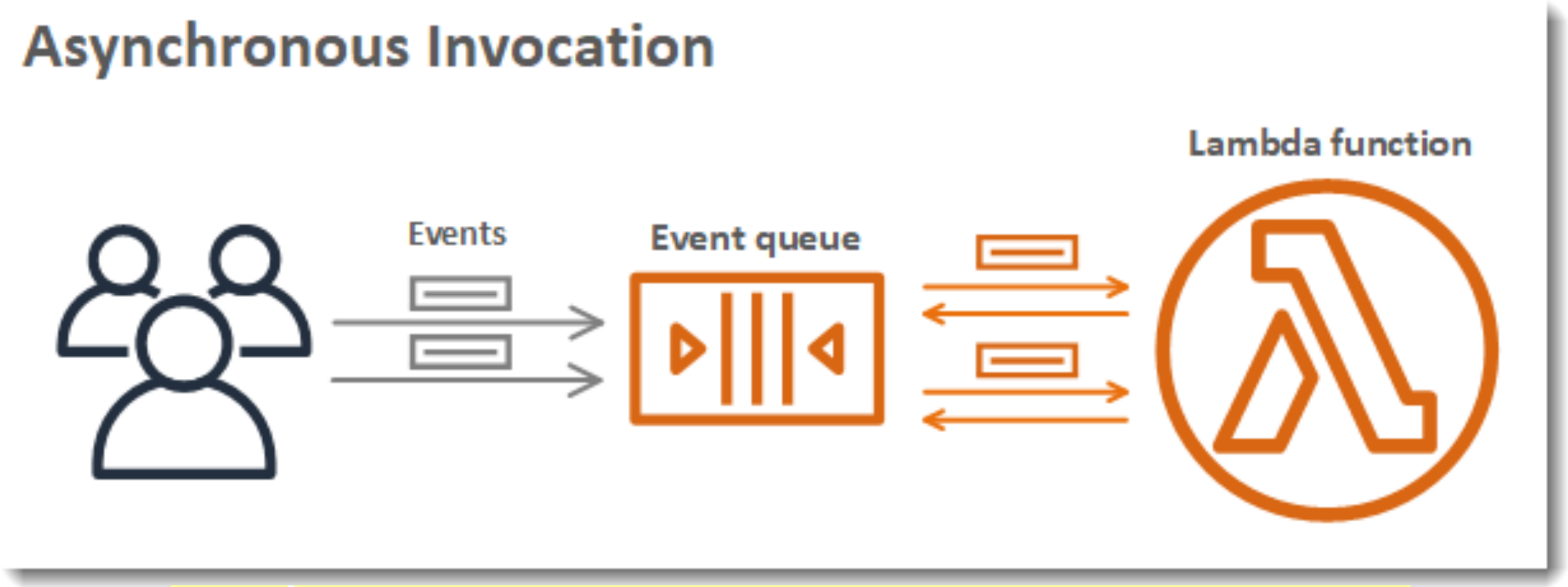
1	**Queue-based processing**: Queues events for processing, Different from synchronous invocation, Doesn't wait for function completion

Asynchronous invocation



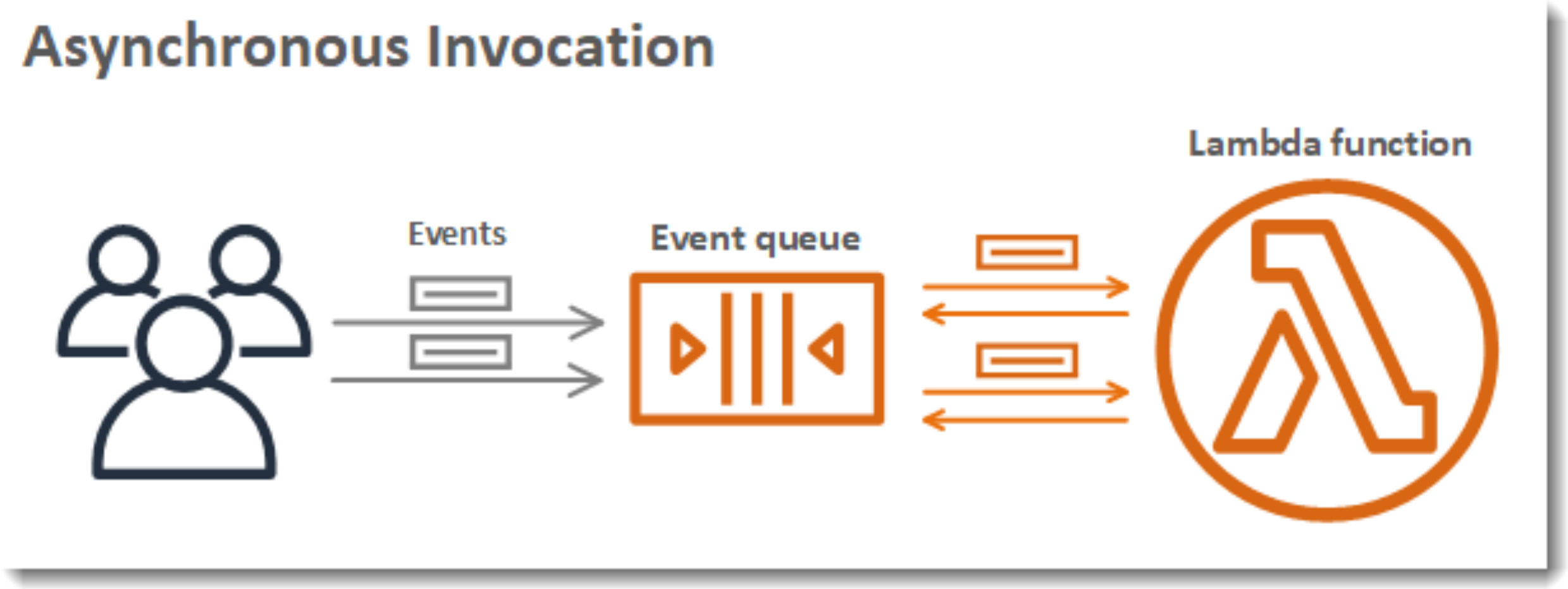
2	**Immediate response:** Lambda queues the event, Returns success status immediately, Caller continues without waiting

Asynchronous invocation



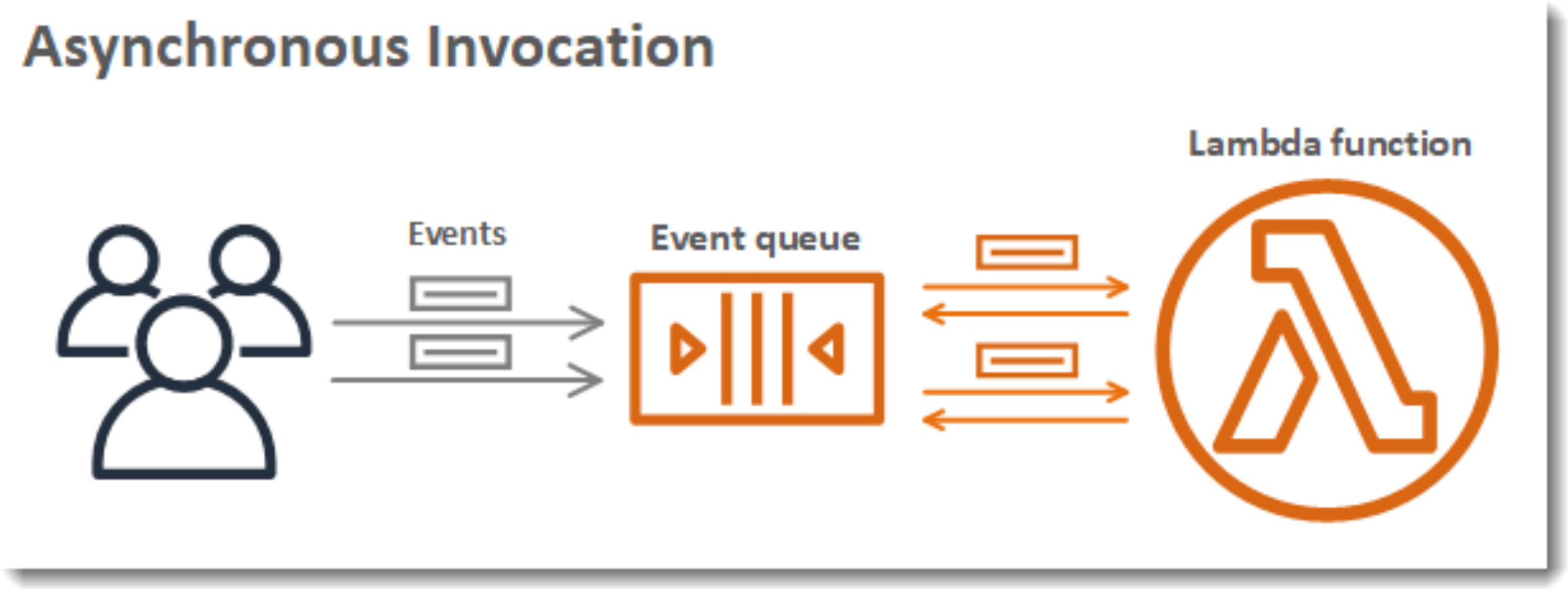
3	🔄 **Automatic retry mechanism**: 🚨 Handles errors automatically, ⚙️ Manages throttling situations, 📜 Lambda retries failed invocations

Asynchronous invocation



4	**Customizable error handling**: User-defined error behavior, Configurable retry attempts, Flexible handling options

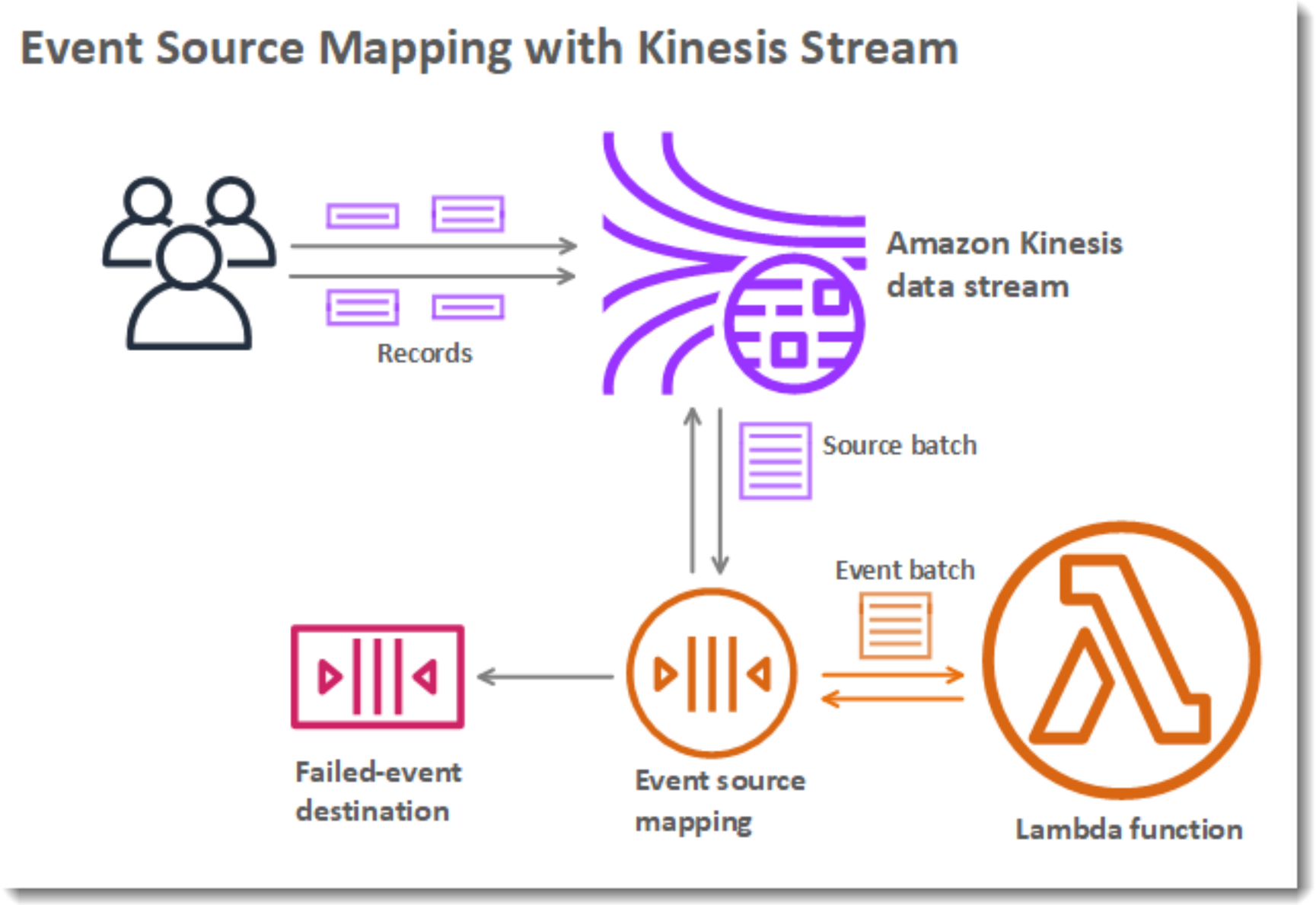
Asynchronous invocation



5

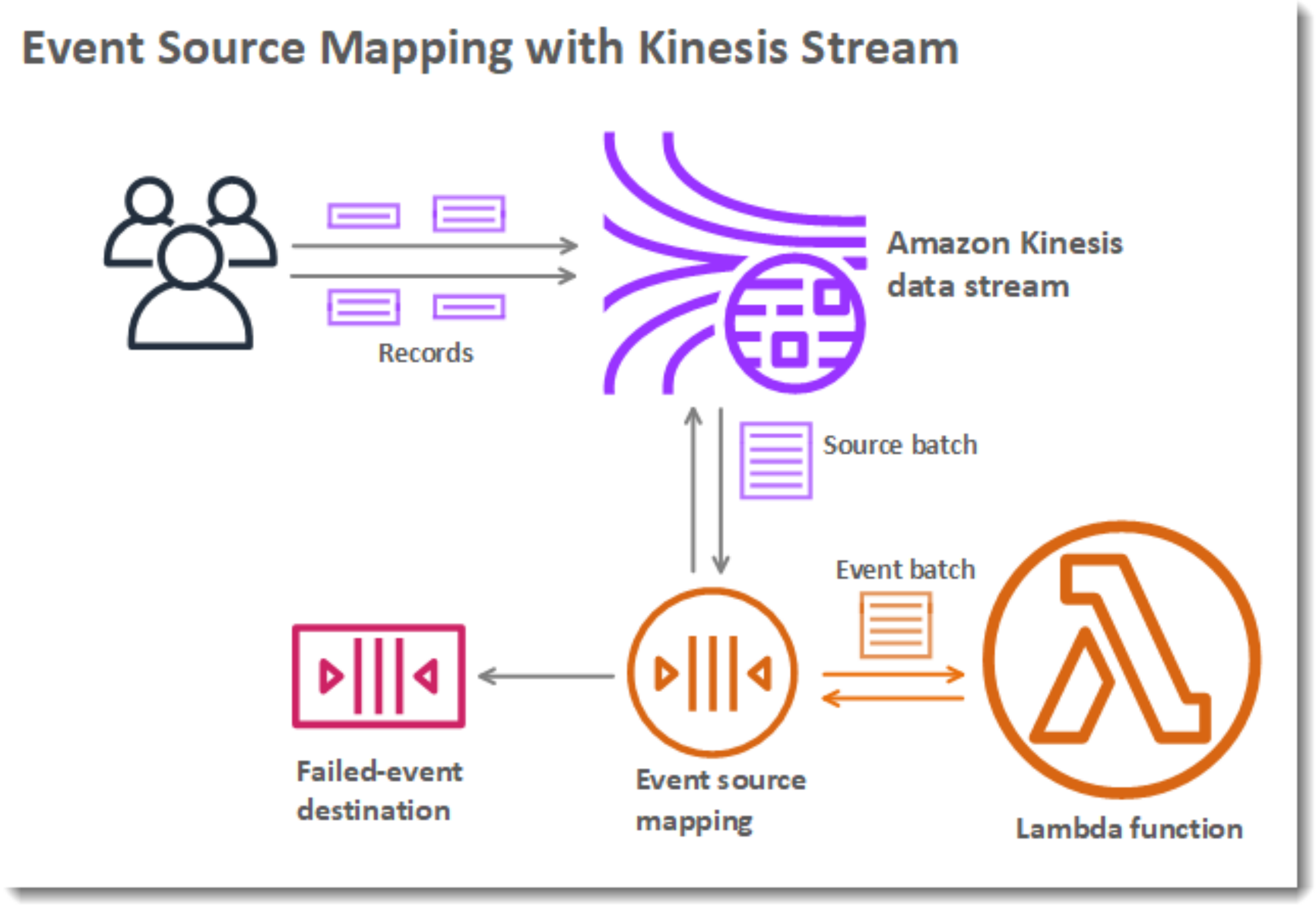
Failed event destinations: Dead-letter queue (DLQ) support, Alternative destinations possible, Enables later analysis, Allows for event reprocessing

Event source mappings



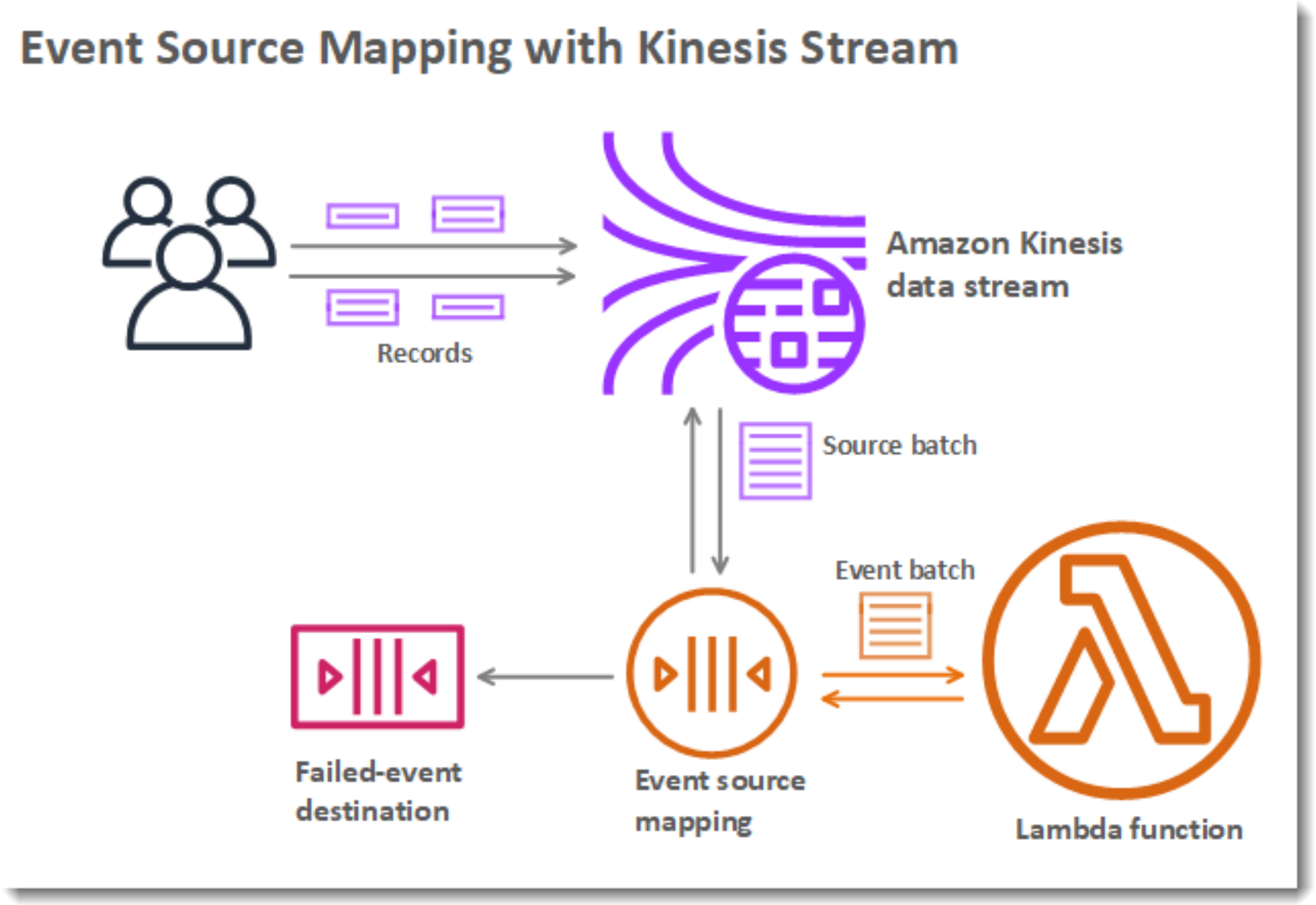
1️⃣. 🔄 Links Lambda to stream-based services
🧱 Processes SQS queue items
📊 Handles Kinesis streams
📋 Works with DynamoDB streams

Event source mappings



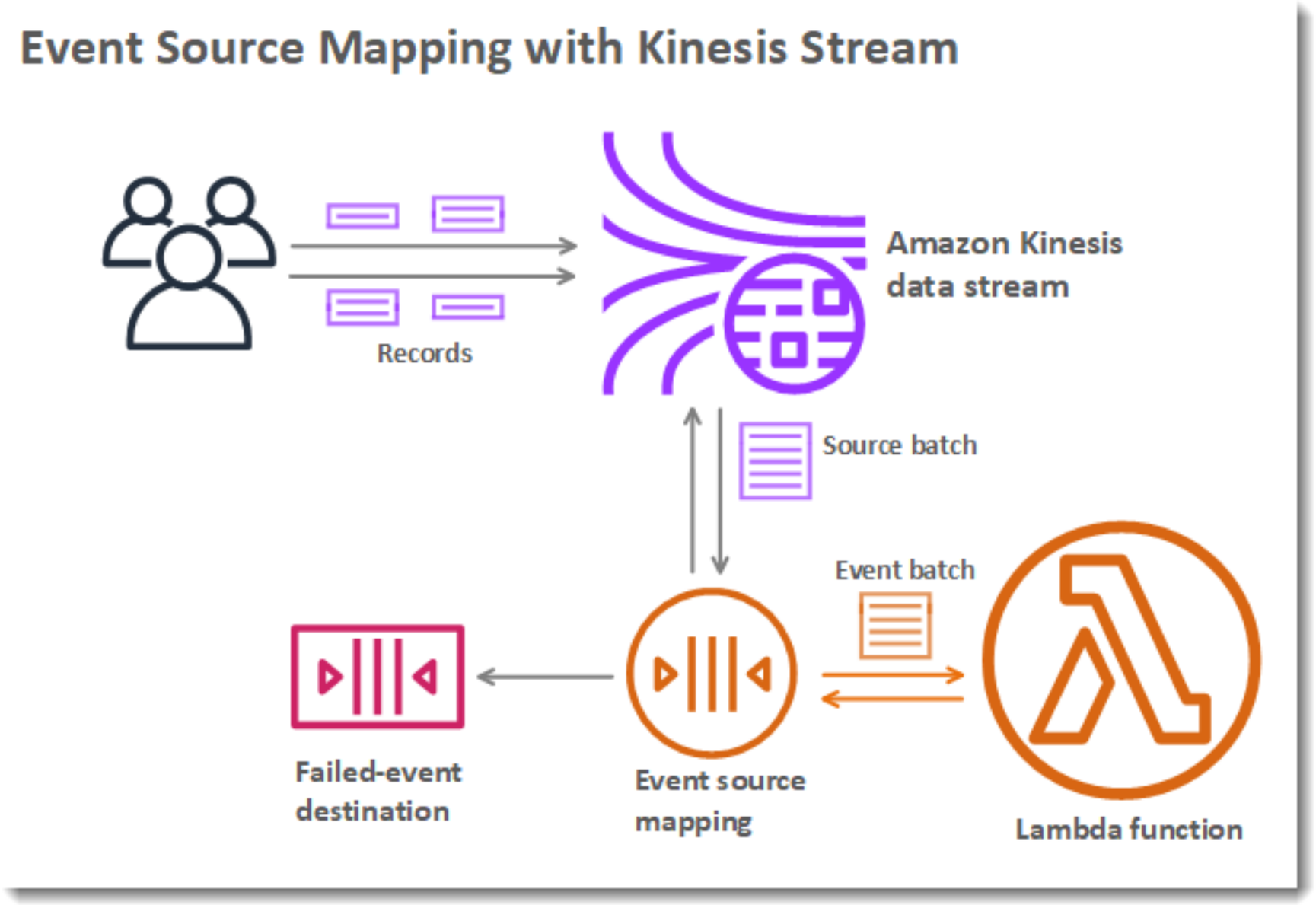
2️⃣. 📩 Lambda reads from source services
📩 Pulls data from mapped sources
🔄 Continuous data ingestion

Event source mappings



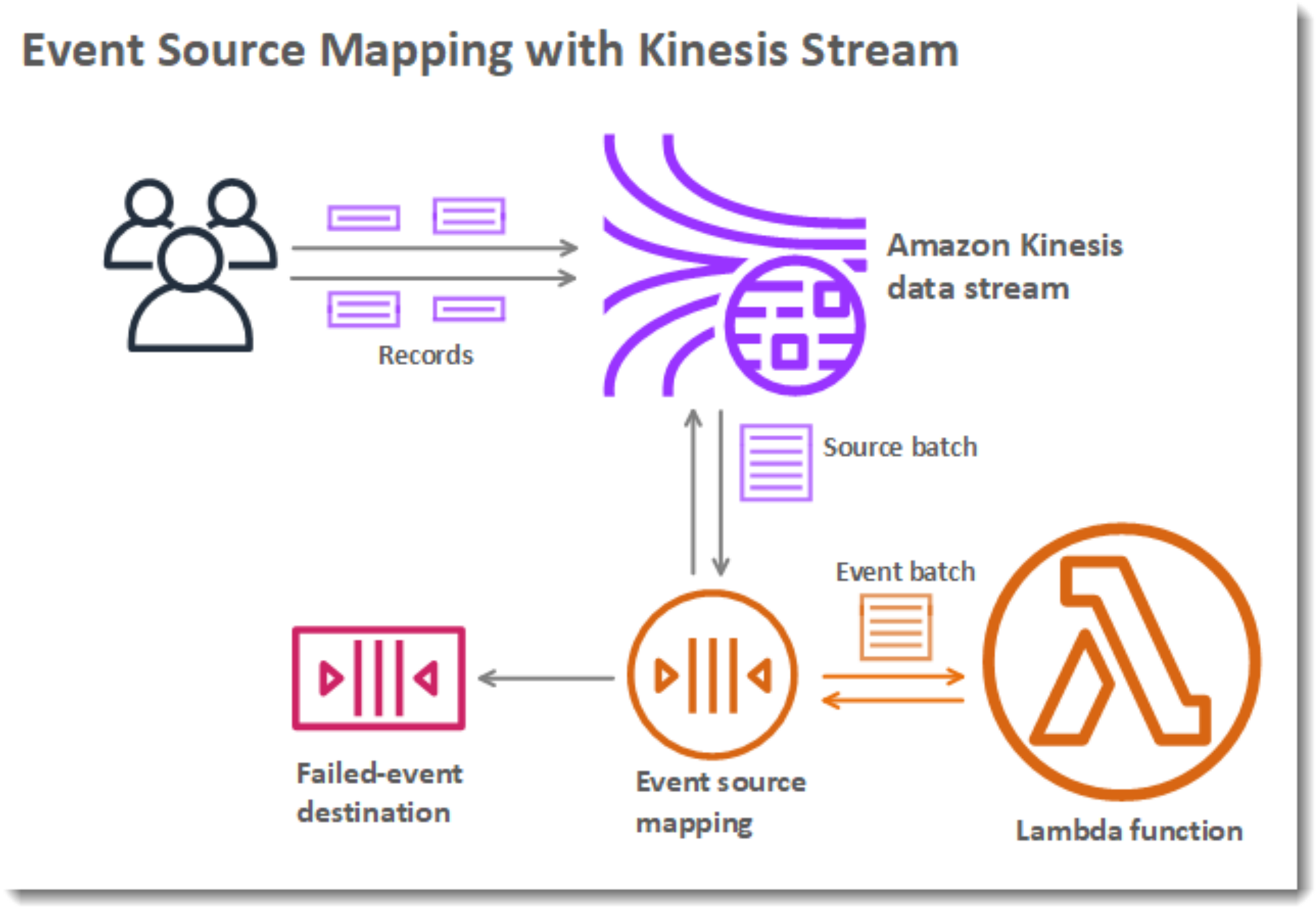
3️⃣. 📦 Batch processing of items
🗃️ Groups items into batches
🔢 Handles hundreds or thousands per batch
📩 Sends batches to Lambda function

Event source mappings



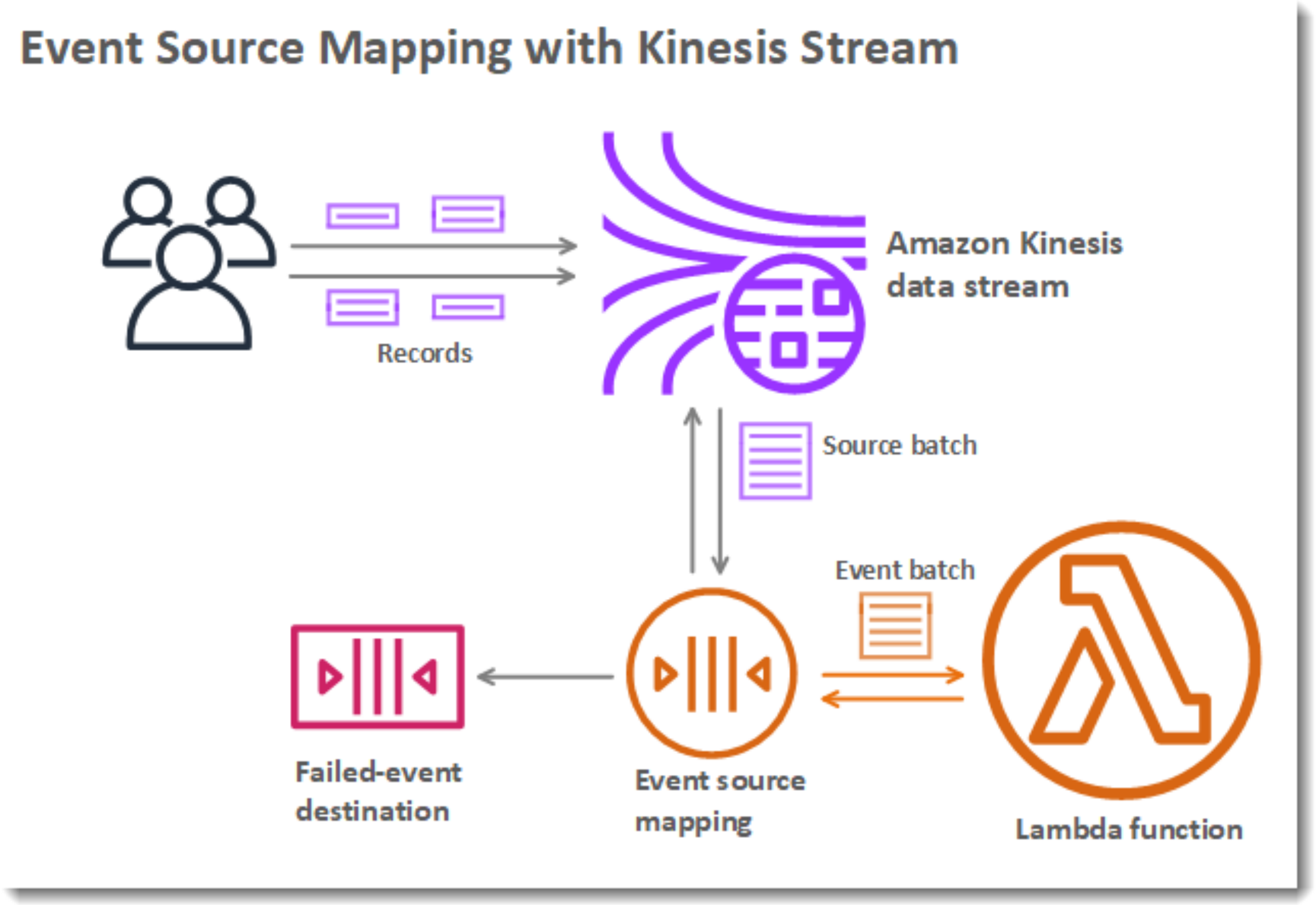
4️⃣. 🗒️ Built-in queue and retry mechanism
🗃️ Maintains local queue
🔁 Automatic retry on errors
🛑 Handles throttling gracefully

Event source mappings



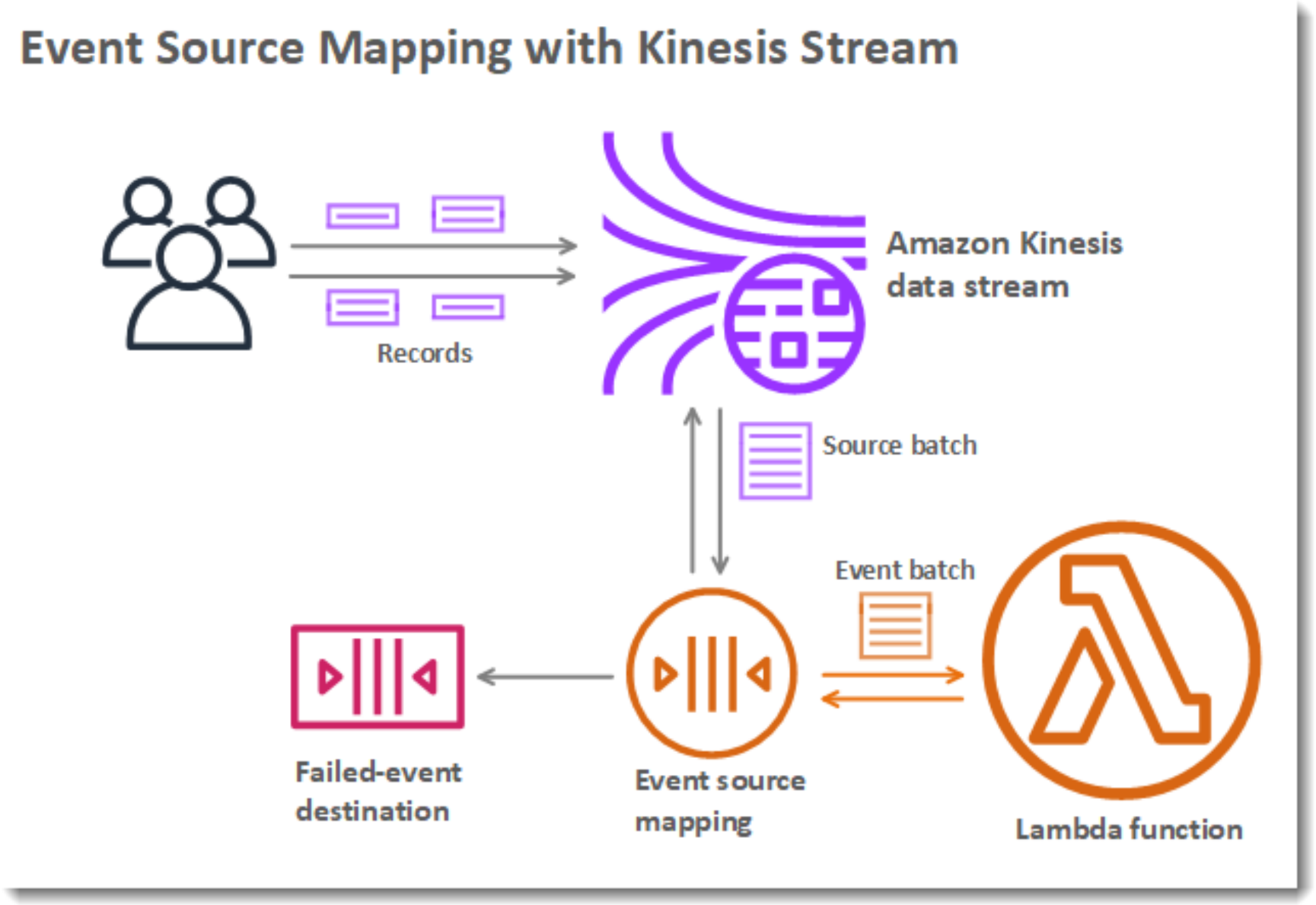
5️⃣. ⚙️ Customizable batching behavior
🔢 Adjustable batch size
⏱️ Configurable time window
🔧 Fine-tune for performance

Event source mappings



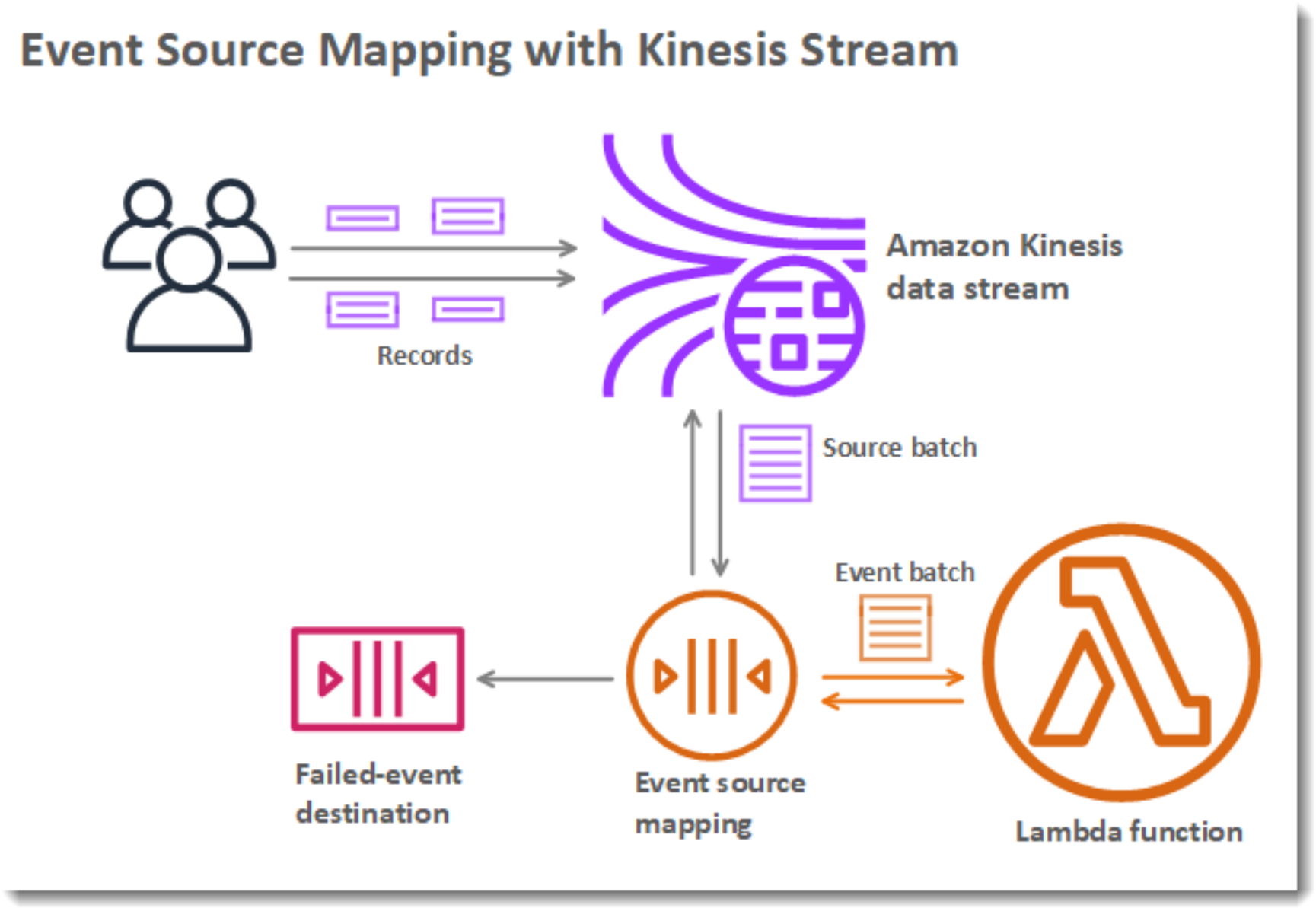
6️⃣. 🔀 Error handling destinations
📩 Failed items routing
📋 SQS queue destination option
📣 SNS topic destination option
🔍 Enables analysis of failures

Event source mappings



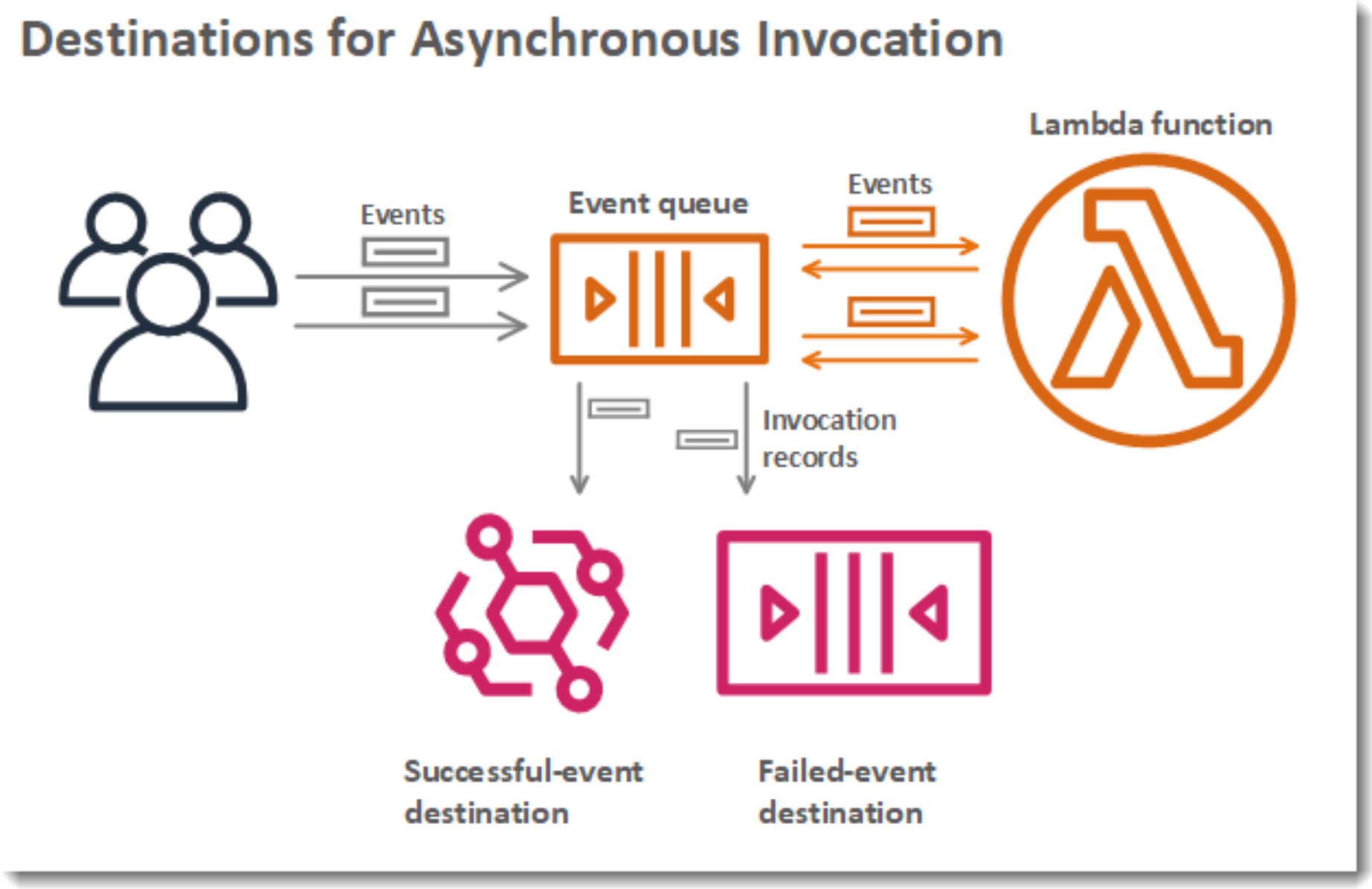
7️⃣. 🔍 Active polling mechanism
🔄 Continuously checks for new items
🔎 Monitors SQS for messages
📊 Watches Kinesis for records
🗃️ Scans DynamoDB for changes

Event source mappings



8️⃣. 🔄 Synchronous function invocation
📦 Receives batches of records
⏱️ Waits for function completion
🔄 Processes one batch at a time

Destinations



1️⃣. 🔀 Directing asynchronous invocation results

📦 Routes function results

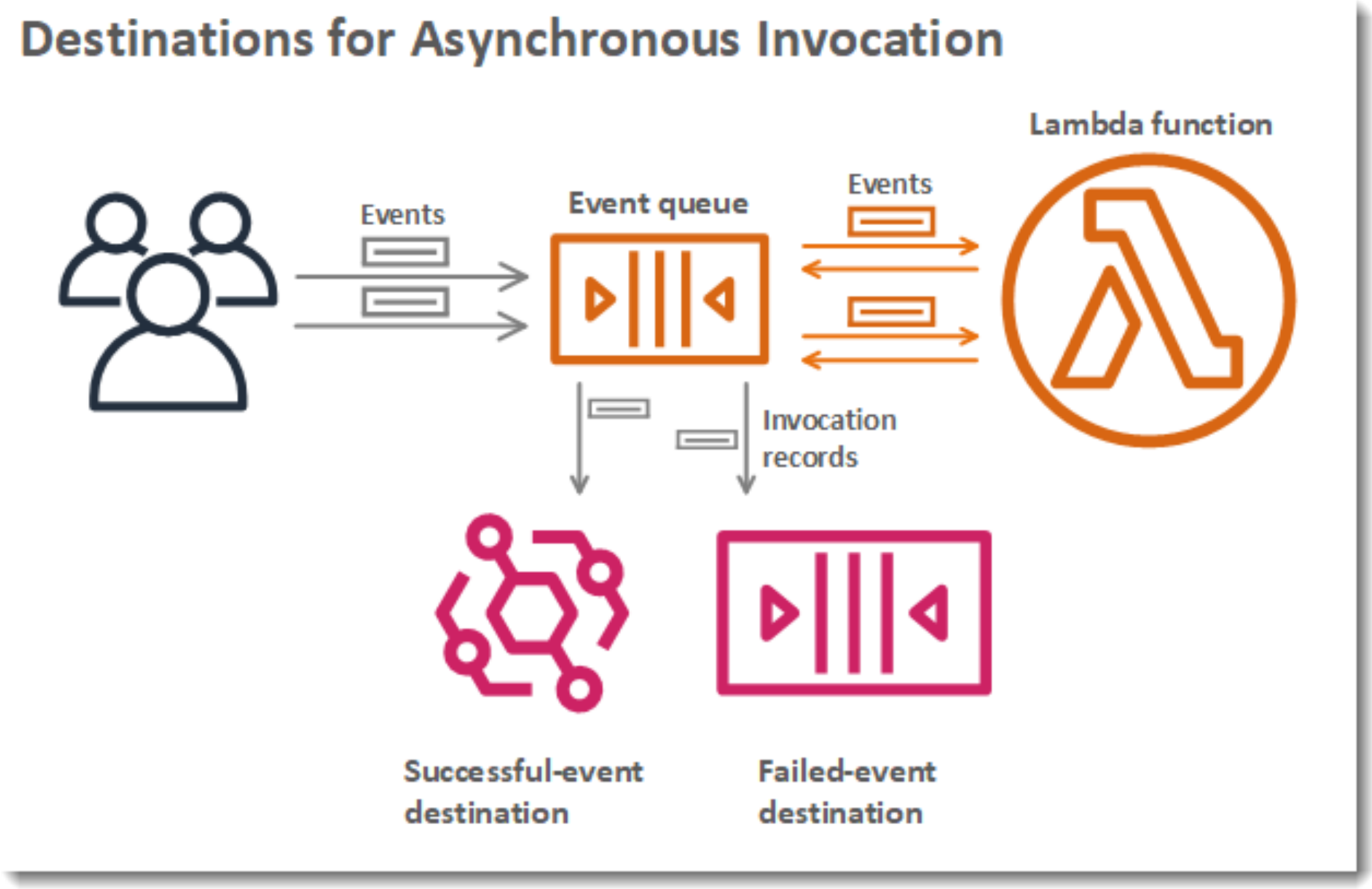
📋 Supports queue destinations

📣 Supports topic destinations

⚙️ Supports function destinations

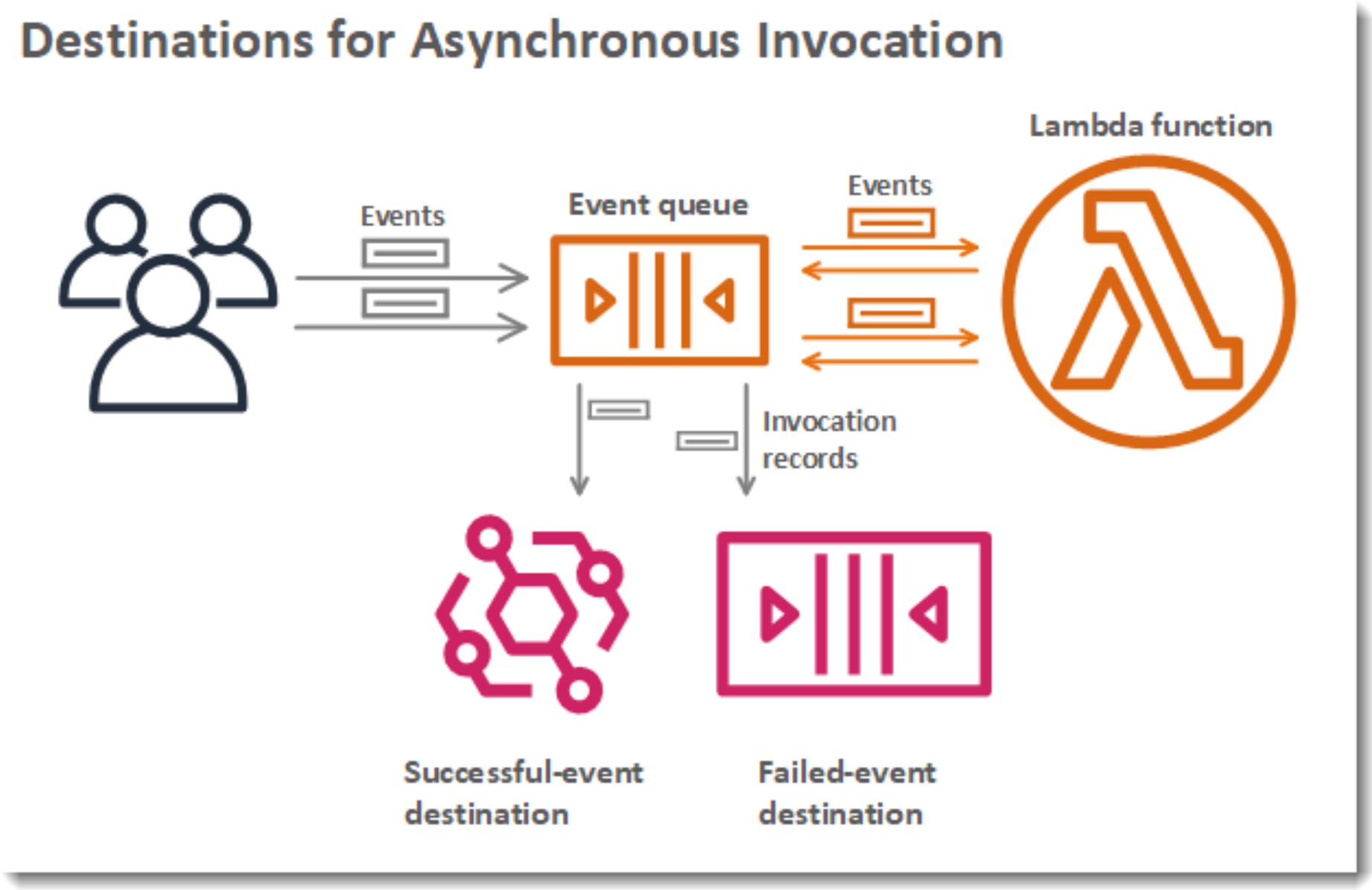
🚌 Supports event bus destinations

Destinations



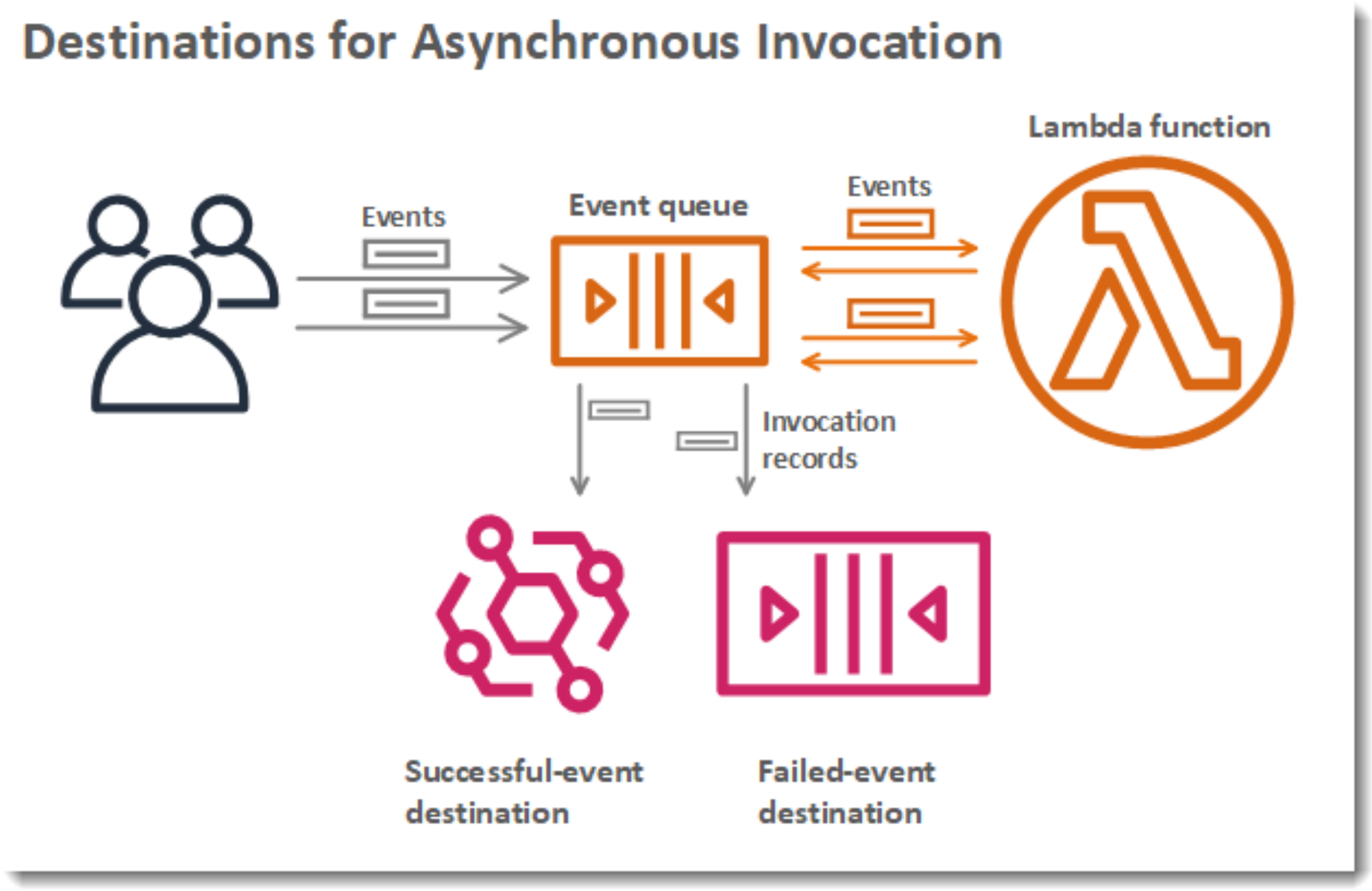
2️⃣. 🔀 Success and failure routing
✅ Configurable success destinations
✕ Separate failure destinations
🔁 Independent routing paths

Destinations



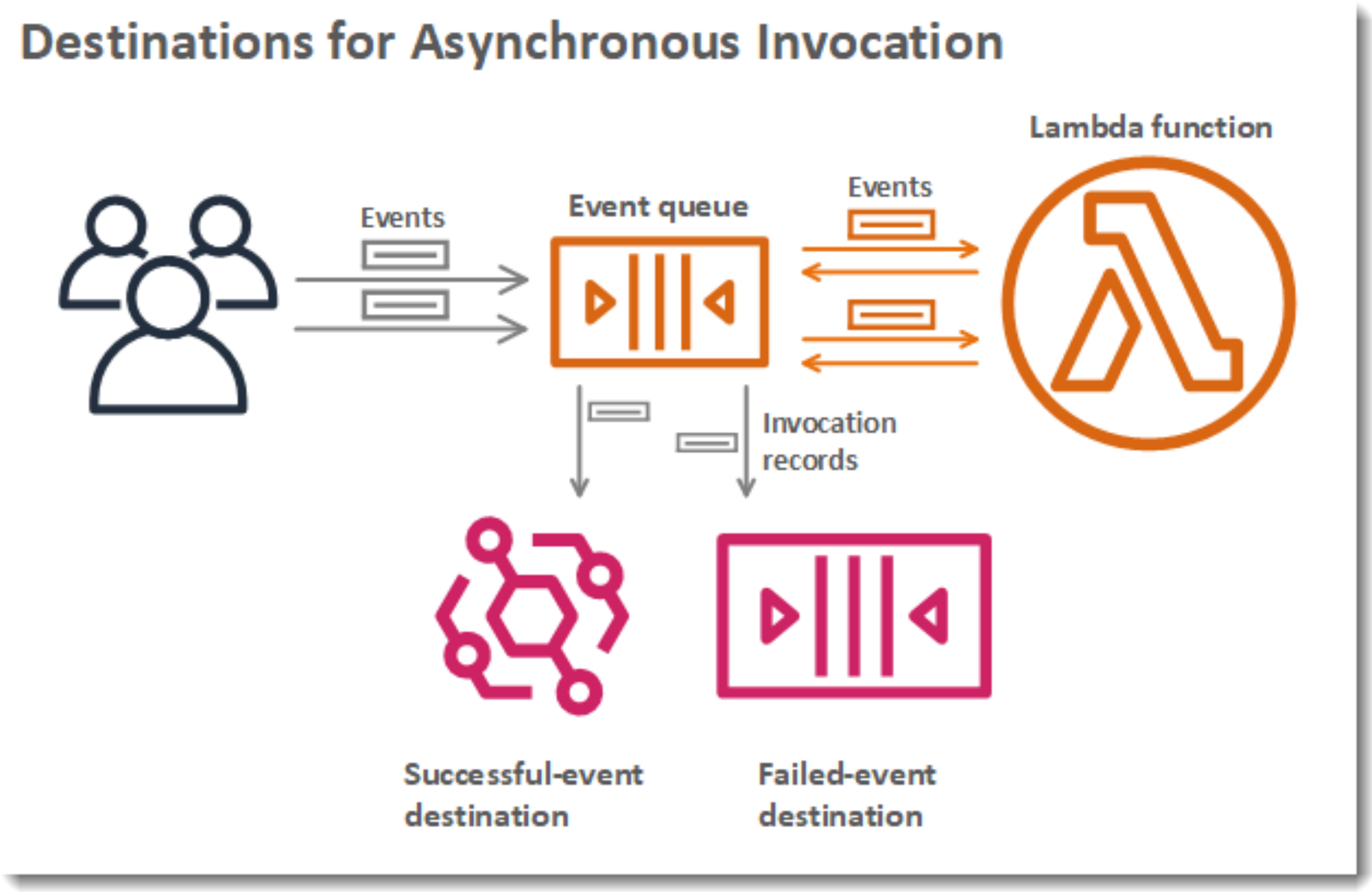
3️⃣. 📋 Detailed invocation records
📄 Contains event details
💬 Includes function response
❓ Provides reason for sending
📊 Complete execution metadata

Destinations



4️⃣. 🌊 Stream-based failures handling
🔄 Works with Kinesis
📋 Supports DynamoDB Streams
📤 Sends failed batches to queue
📣 Or sends failures to topics

Destinations



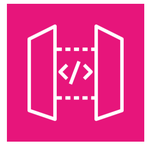
5️⃣. 📝 Failure metadata records
ℹ️ Contains batch metadata
🔍 Points to specific failure items
🔎 Helps identify failure cause
🔄 Facilitates reprocessing

Lambda Integrations

1. Amazon Alexa

2. Amazon Managed Streaming for Apache Kafka

3. Self-managed Apache Kafka



4. Amazon API Gateway

5. AWS CloudFormation

6. Amazon CloudFront (Lambda@Edge)

7. Amazon EventBridge (CloudWatch Events)

8. Amazon CloudWatch Logs

9. AWS CodeCommit

10. AWS CodePipeline

11. Amazon Cognito

12. AWS Config

13. Amazon Connect

14. Amazon DynamoDB

15. Amazon Elastic File System

16. Elastic Load Balancing (Application Load Balancer)

17. AWS IoT

18. AWS IoT Events

19. Amazon Kinesis

20. Amazon Data Firehose

21. Amazon Lex

22. Amazon MQ

23. Amazon Simple Email Service

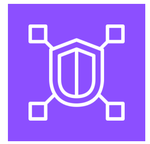
24. Amazon Simple Notification Service

25. Amazon Simple Queue Service

26. Amazon Simple Storage Service (S3)

27. Amazon S3 Batch

28. Secrets Manager



29. Amazon VPC Lattice

30. AWS X-Ray

Thanks for Watching