The Art of the State

Fully managed service orchestration powered by state machines

Gabe Hollombe
Sr. Technical Evangelist, AWS

@gabehollombe

What we'll cover in this session

- Getting things done with distributed services
- Coordination patterns: Choreography vs. Orchestration
- Service orchestration made easy using state machines
- AWS Step Functions: state machines in the cloud
- Examples from the real world
- Where to learn more

Getting Things Done

In a Monolith, everything gets deployed together

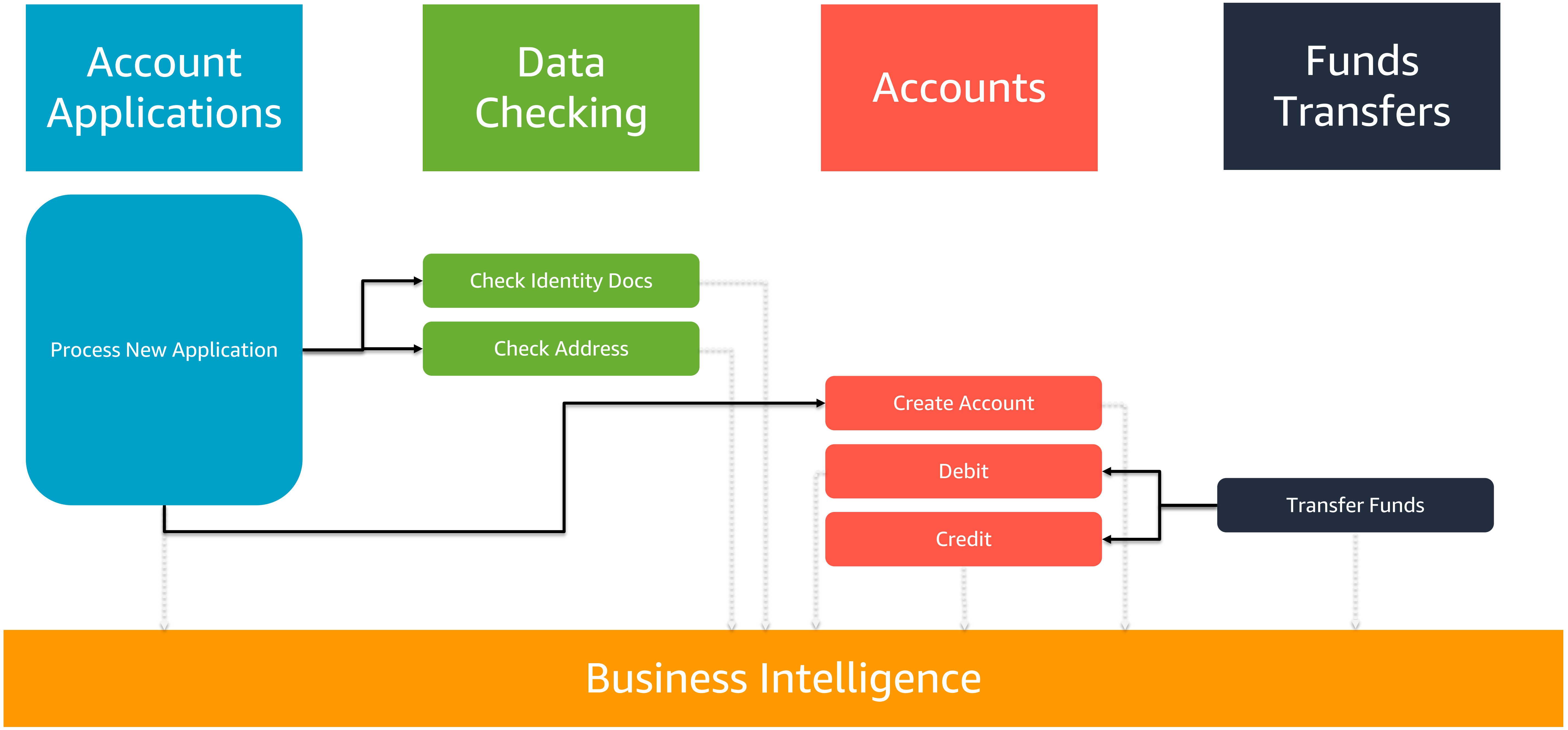
With Microservices, we split the work between multiple systems

Microservices can give us increased agility and scalability

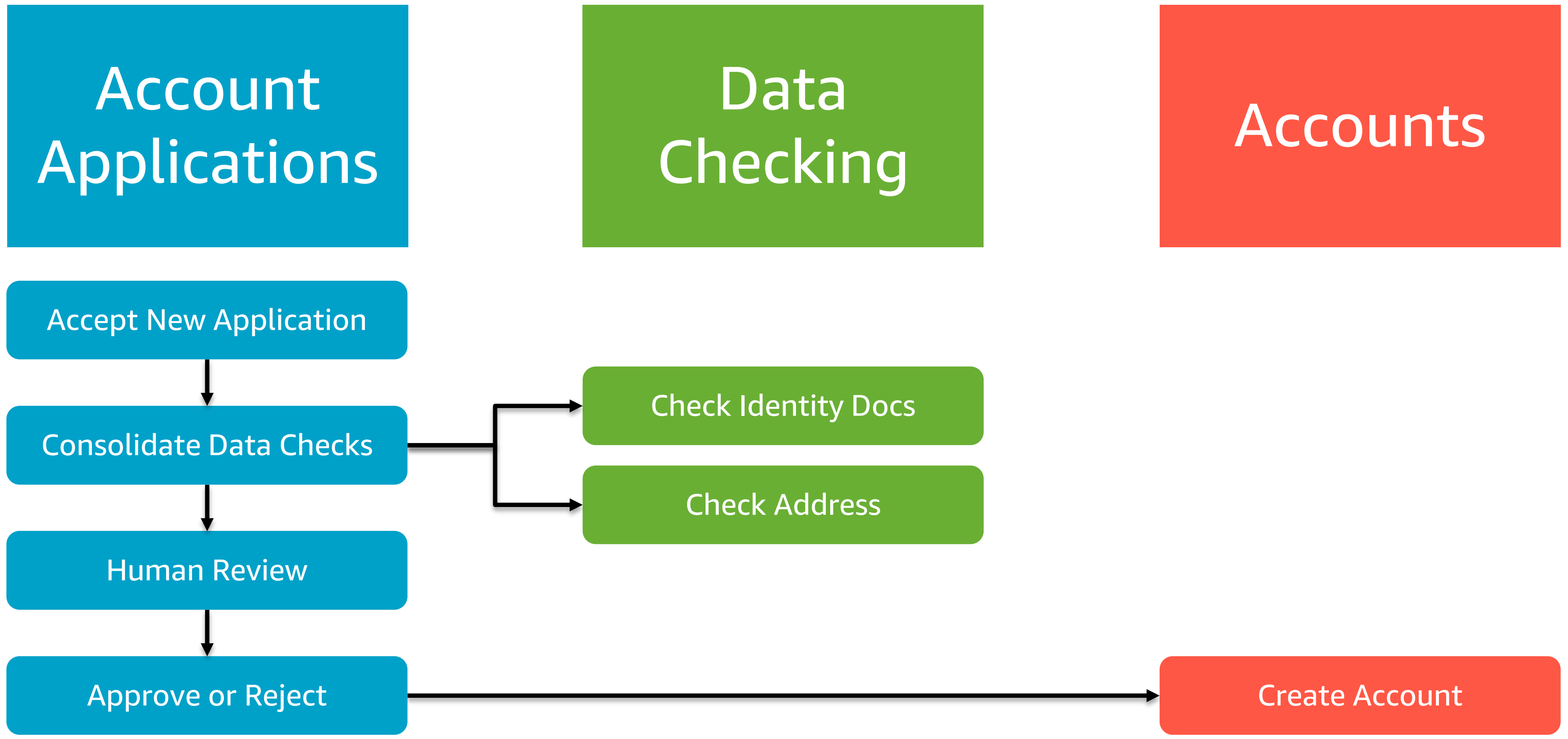
But distributed systems can be harder to coordinate and debug

Coordination Patterns
Choreography & Orchestration

Here's a simplified banking system



Processing a new account application requires some coordination



In *Choreography* services emit and respond to chains of events

Account Applications

Emits
- Application Submitted
- Identity Check Requested
- Address Check Requested
- Application Approved
- Application Rejected

Listens For
- Identity Checked
- Address Checked
- Application Reviewed

Data Checking

Emits
- Identity Checked
- Address Checked

Listens For
- Identity Check Requested
- Address Check Requested

Event Bus

Accounts

Emits
- Account Opened

Listens For
- Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

aws

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

aws

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

Accounts

Emits	**Listens For**
Account Opened	Application Approved

In *Choreography* services emit and respond to chains of events

Account Applications

Emits	Listens For
Application Submitted	Identity Checked
Identity Check Requested	Address Checked
Address Check Requested	Application Reviewed
Application Approved	
Application Rejected	

Data Checking

Emits	Listens For
Identity Checked	Identity Check Requested
Address Checked	Address Check Requested

Event Bus

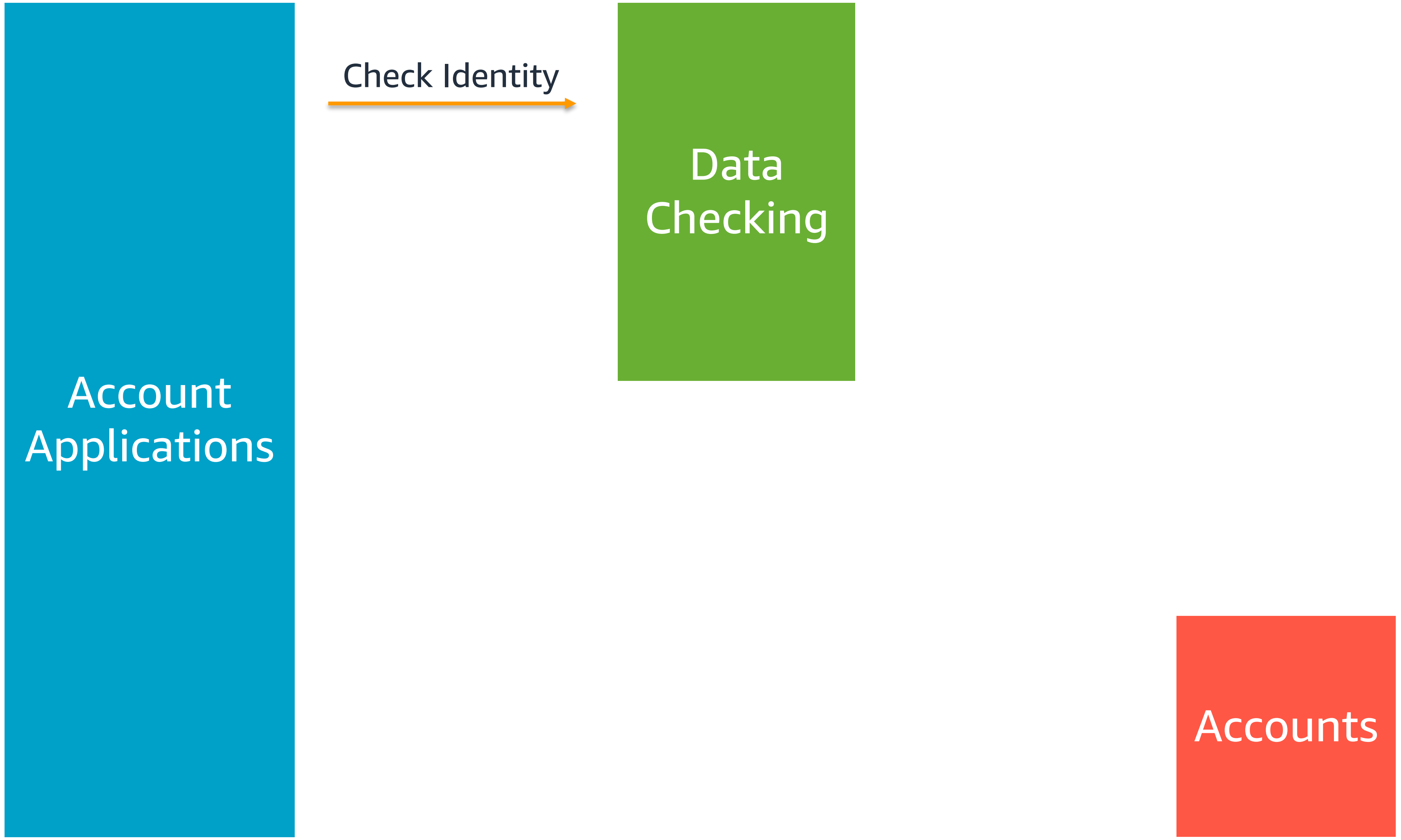
Accounts

Emits	**Listens For**
Account Opened	Application Approved

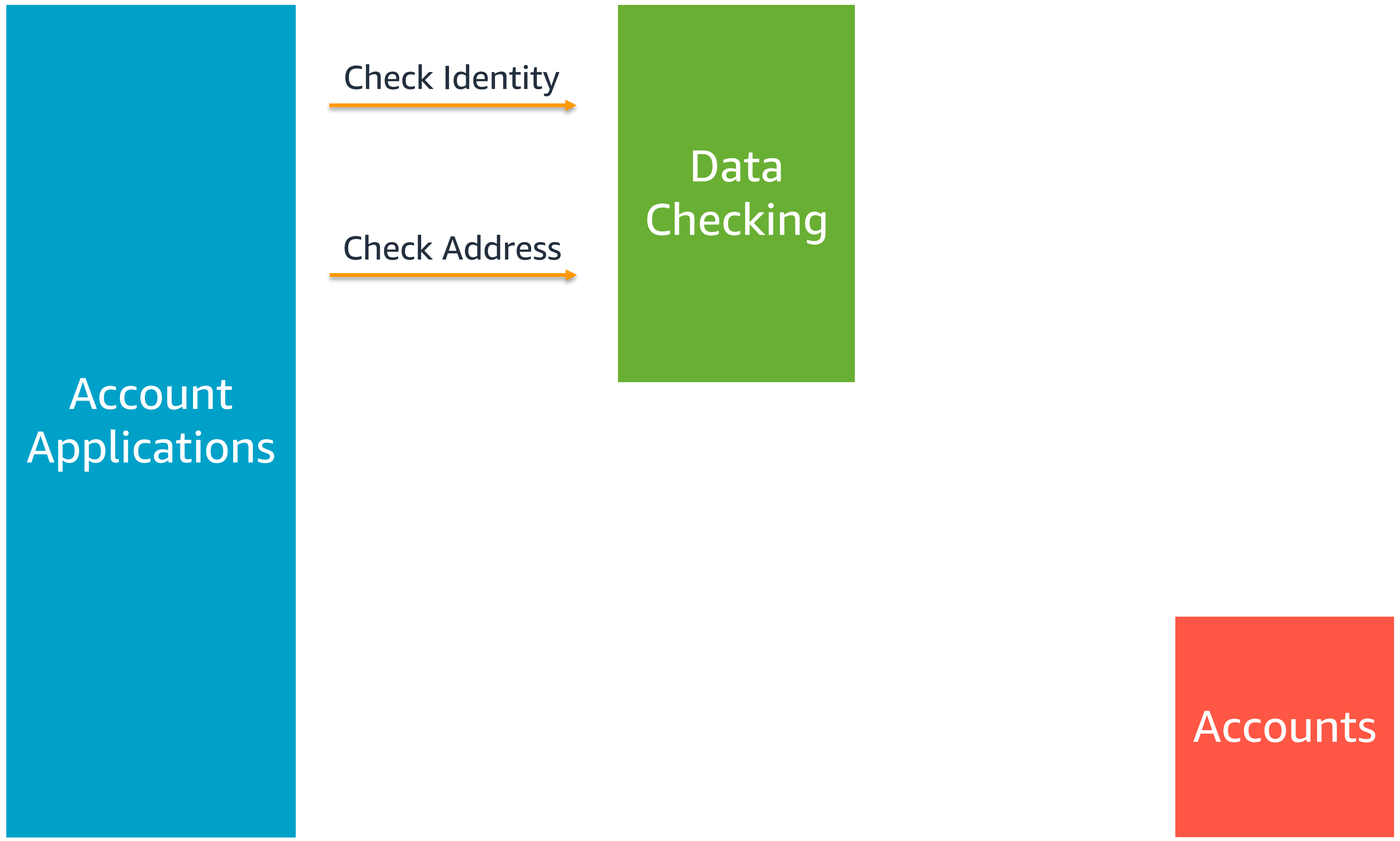
In *Orchestration* one process manages state and calls appropriate services in turn

Account Applications
Data Checking
Accounts

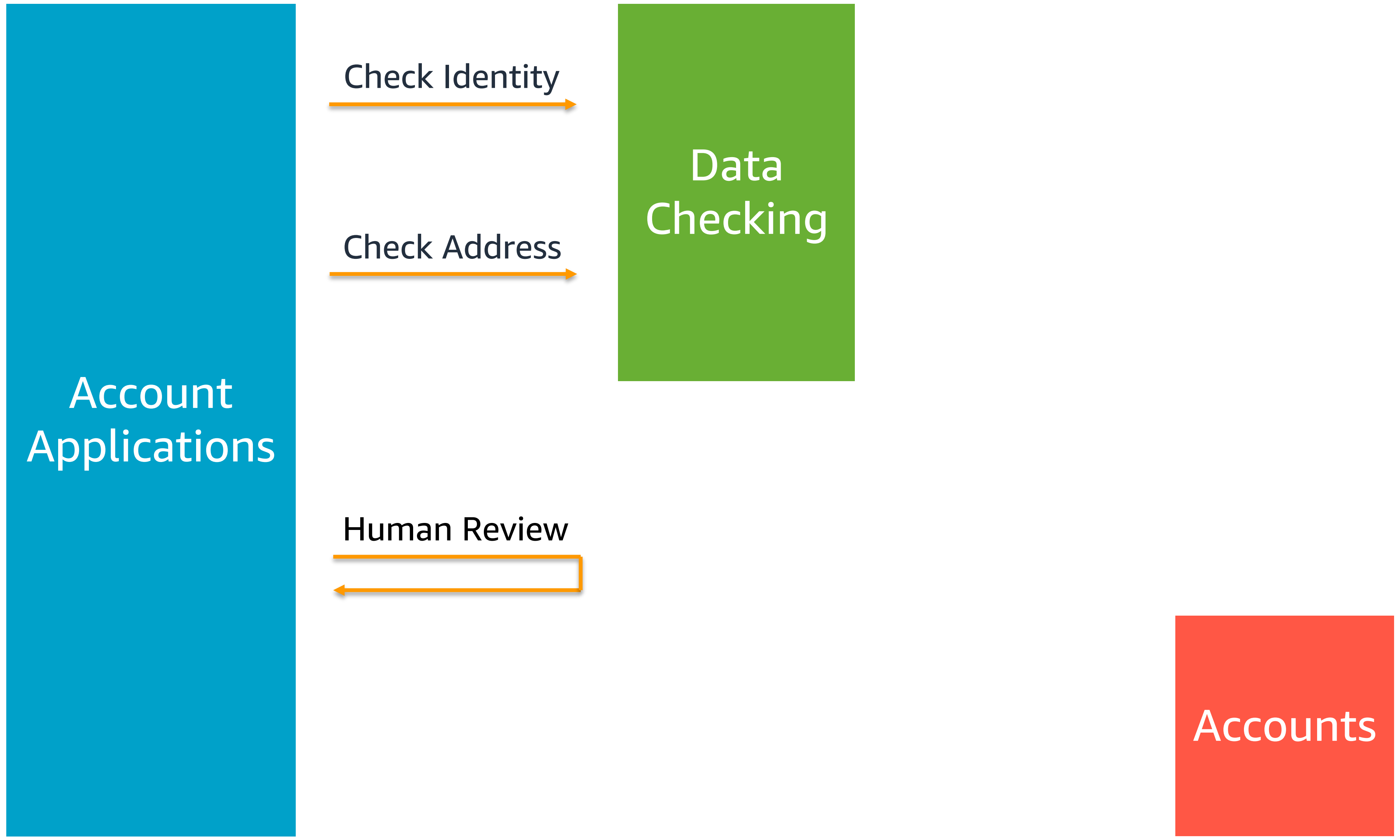
In *Orchestration* one process manages state and calls appropriate services in turn



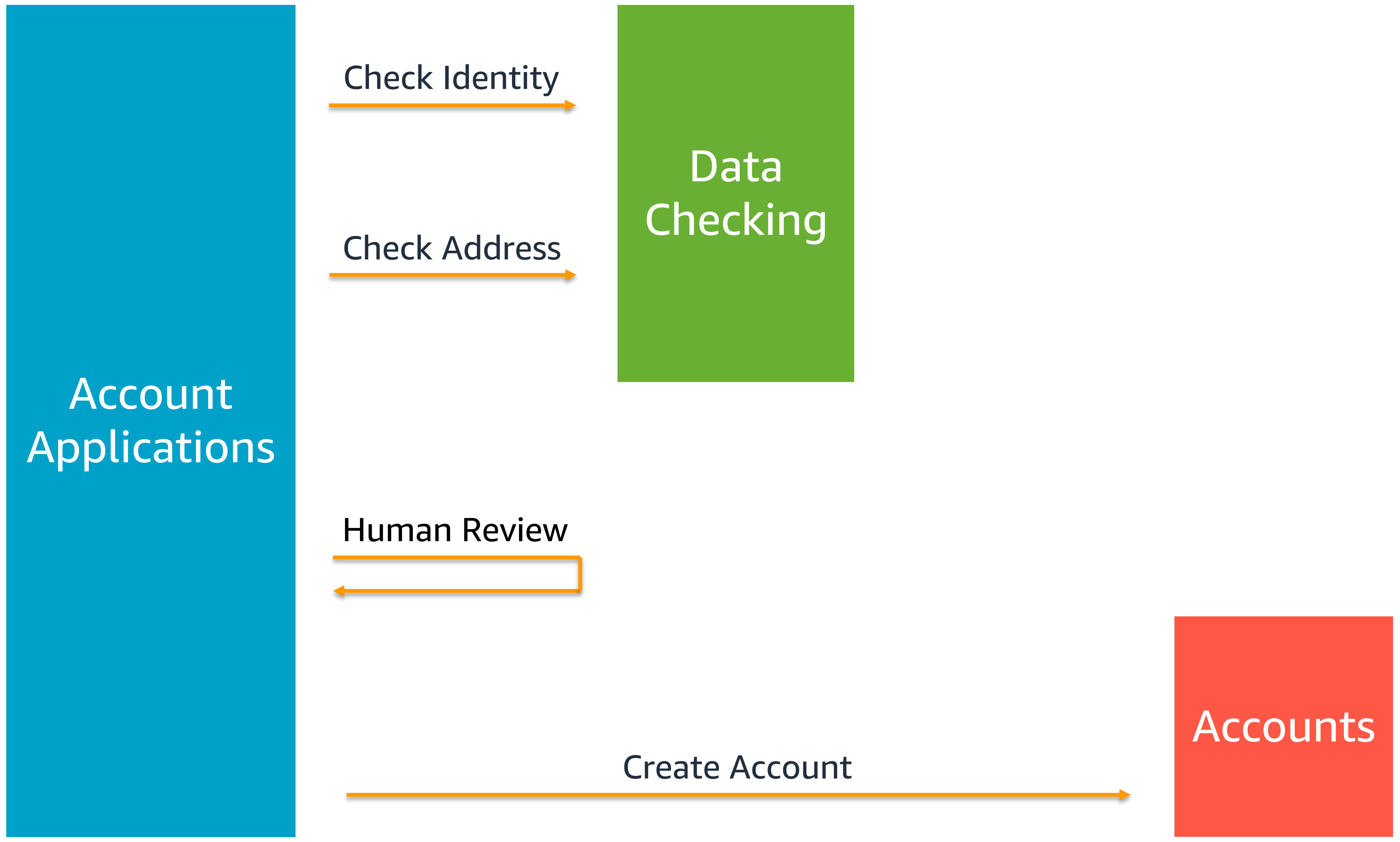
In *Orchestration* one process manages state and calls appropriate services in turn



In *Orchestration* one process manages state and calls appropriate services in turn



In *Orchestration* one process manages state and calls appropriate services in turn



When should I use Choreography vs. Orchestration?

<u>Choreography</u>

Simple workflows without a lot of logic

Broadcast style flows where services don't depend on what events other services emit

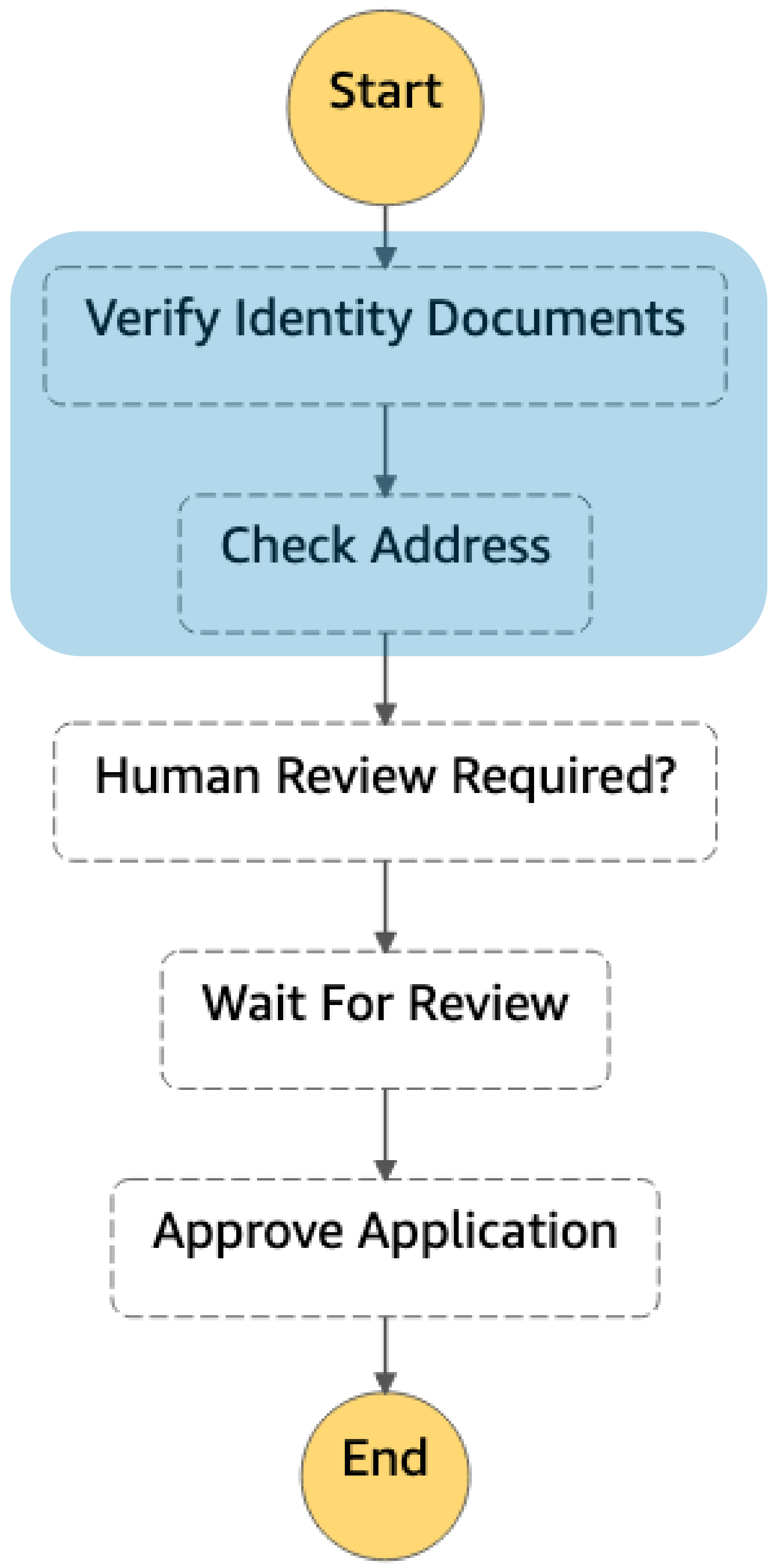
<u>Orchestration</u>

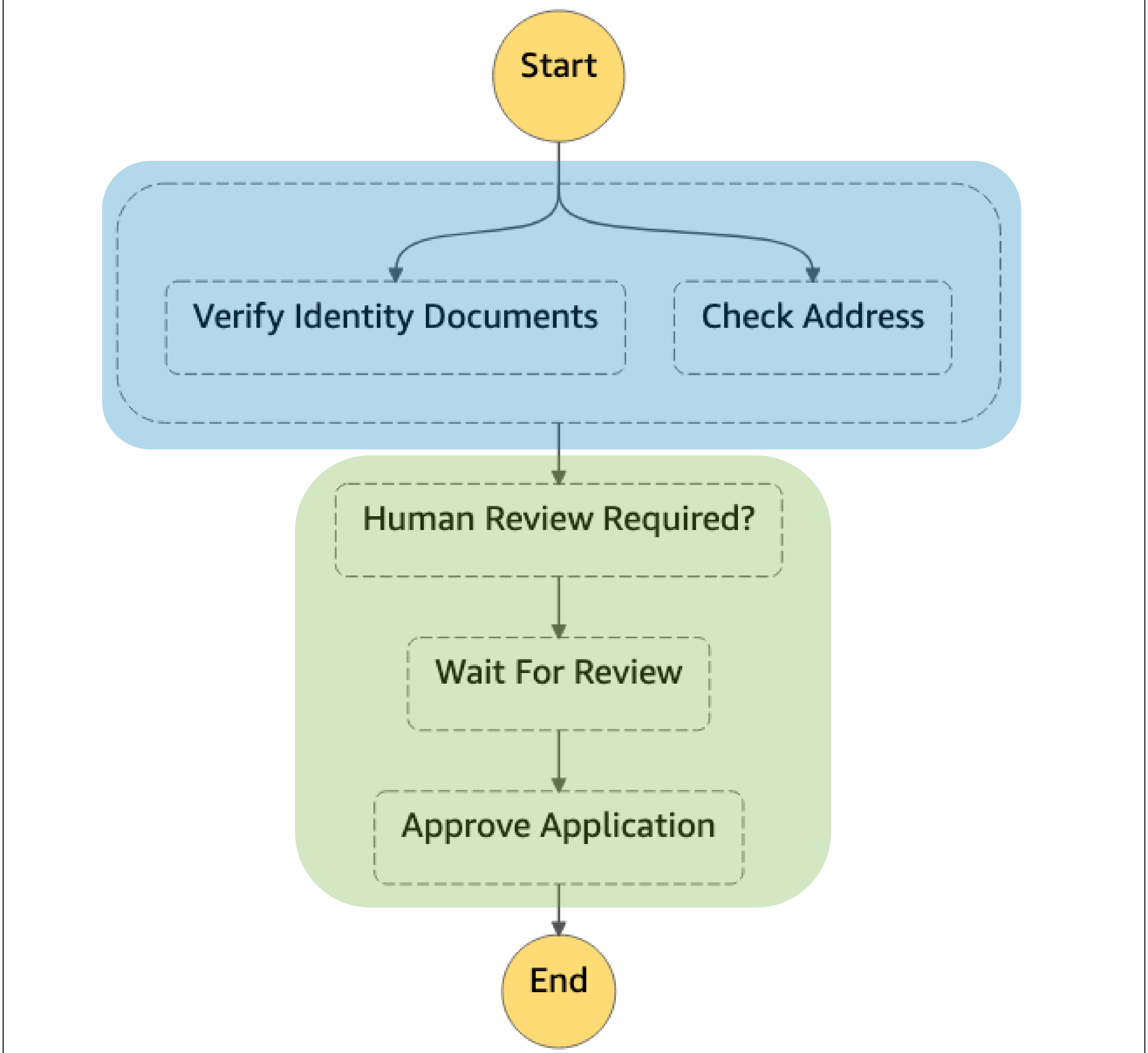
Workflow execution auditability

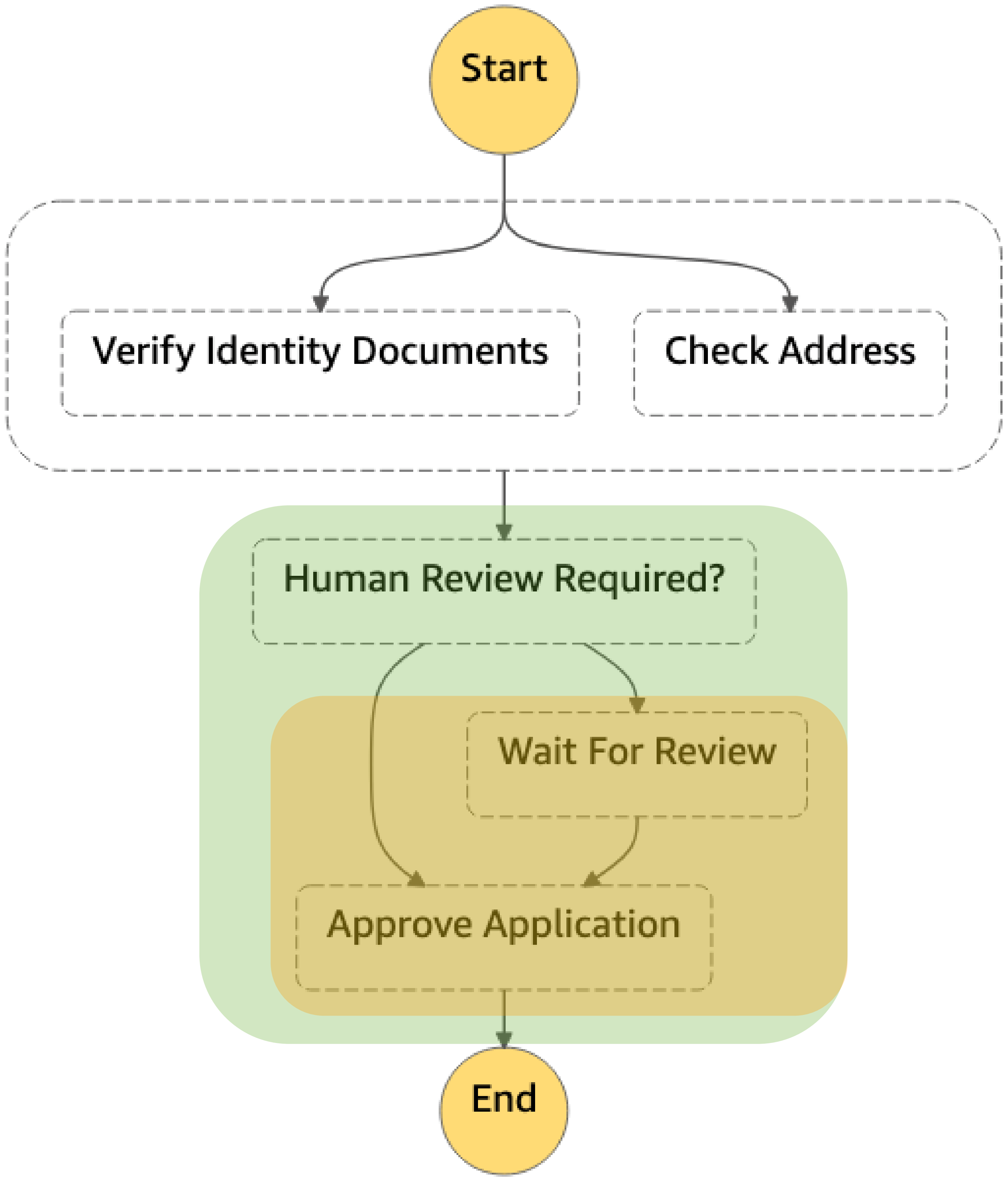
Robust retries & error handling

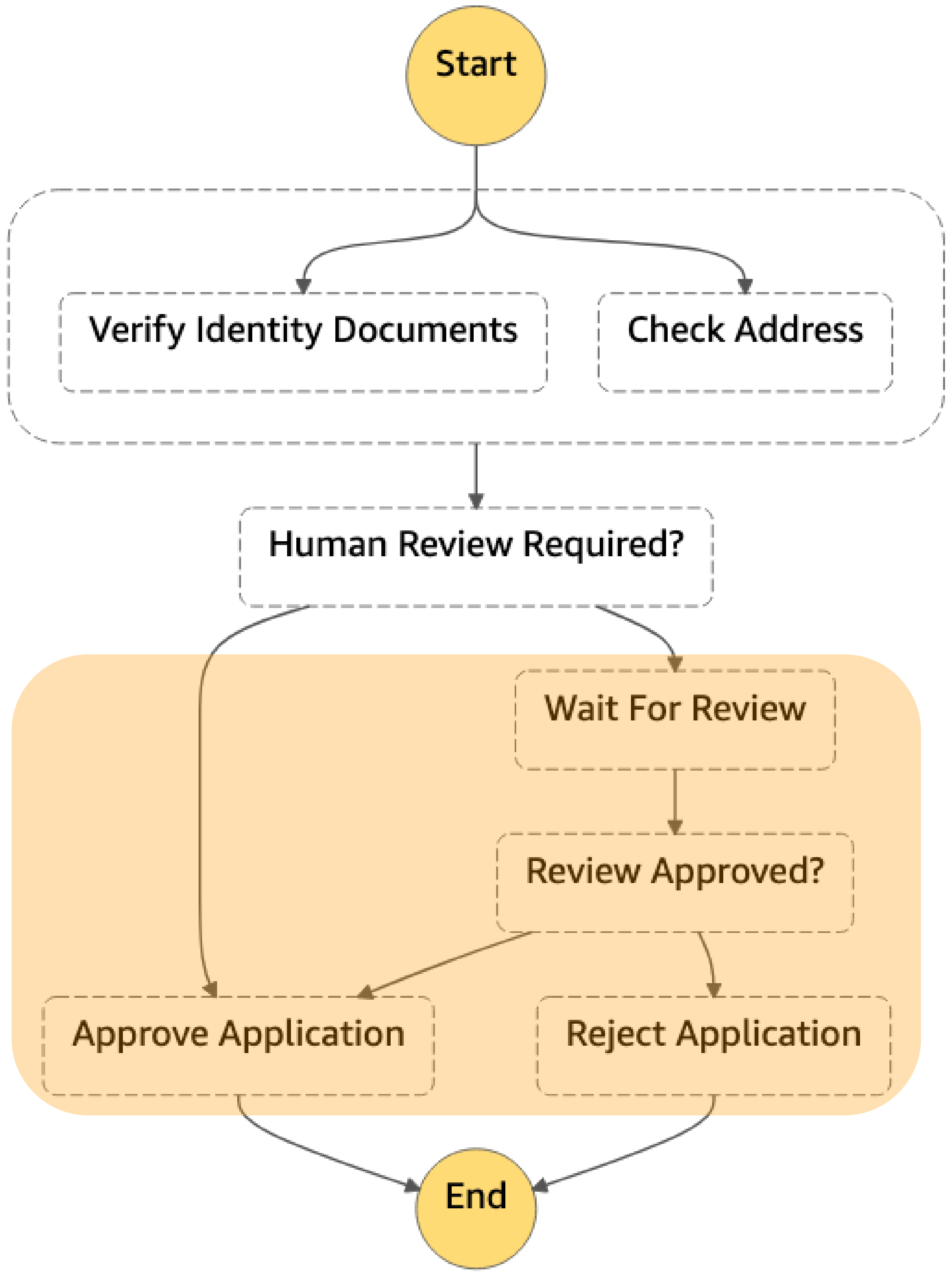
Manage a workflow's business logic in one place

Example Orchestration
Processing new bank account applications


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Review Approved?
Approve Application
Reject Application
End

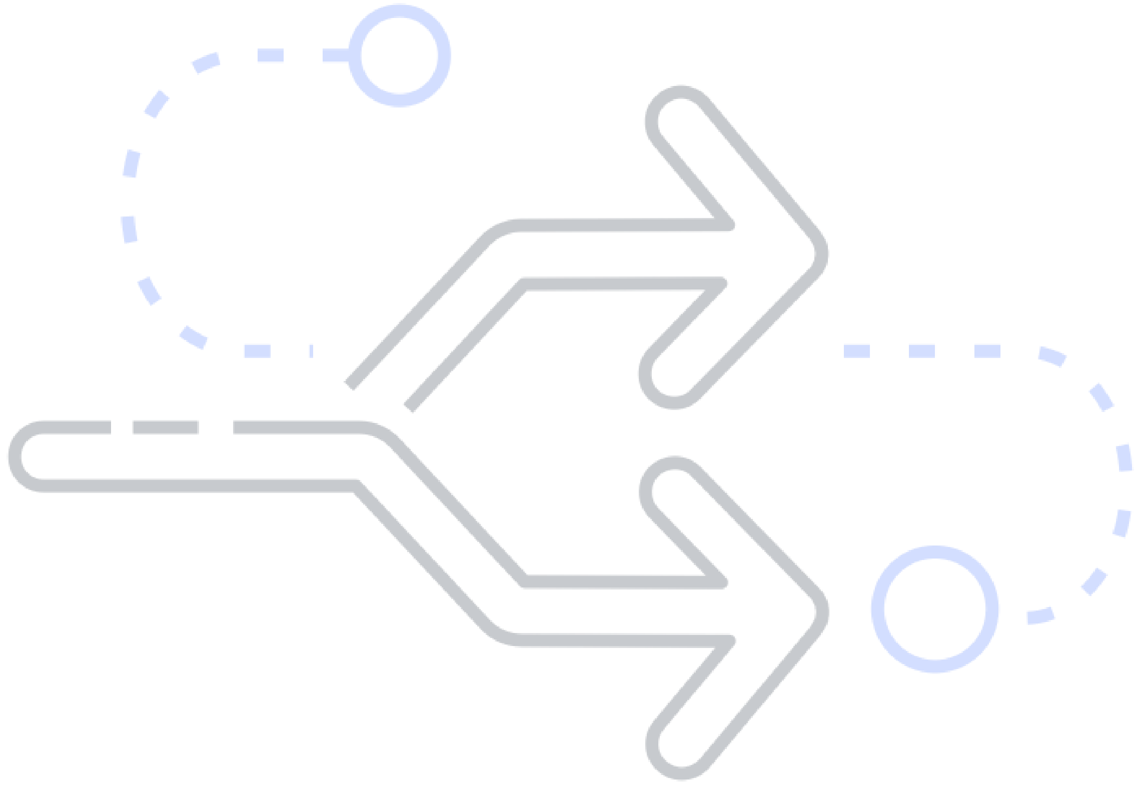
A State Machine

Describes a collection of computational steps split into discrete states

Has one starting state and
always one active state (while executing)

The active state receives input,
takes some action, and generates output

Transitions between states are based on state outputs and rules that we define



AWS Step Functions: Fully-managed state machines on AWS

Resilient workflow automation

Built-in error handling

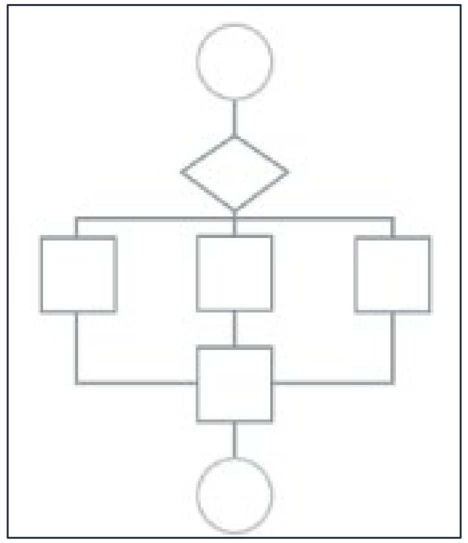
Powerful AWS service integration

First-class support for integrating with your own services

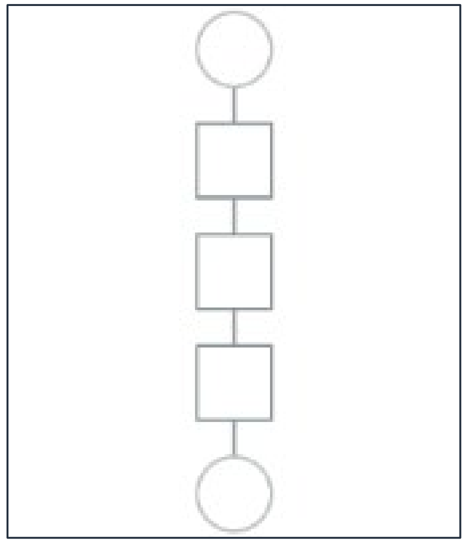
Auditable execution history & visual monitoring

Step Functions
The Basics

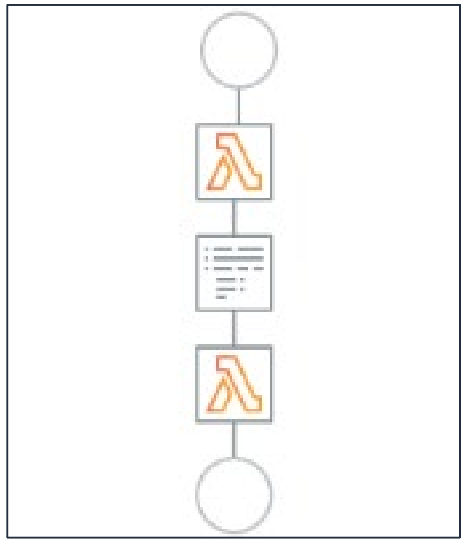
How AWS Step Functions work



Coordinate individual tasks into a visual workflow, so you can build and update apps quickly.

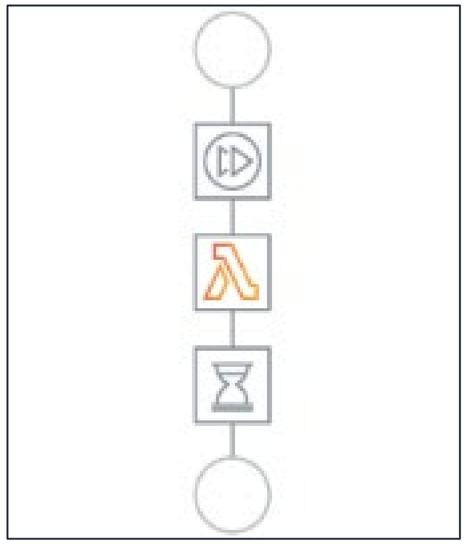


The workflows you build with Step Functions are called **state machines**, and each step of your workflow is called a **state**.

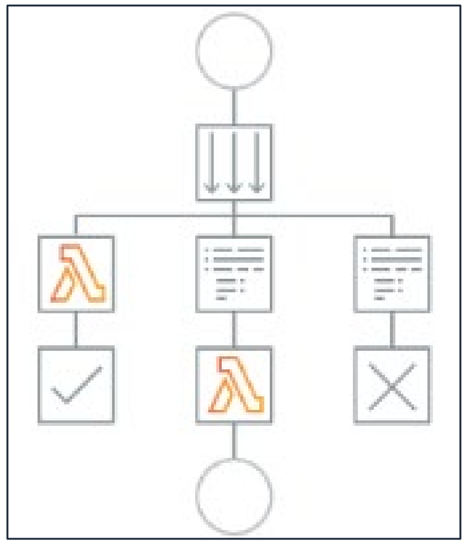


Tasks perform work, either by coordinating another AWS service or an application that you can host basically anywhere.

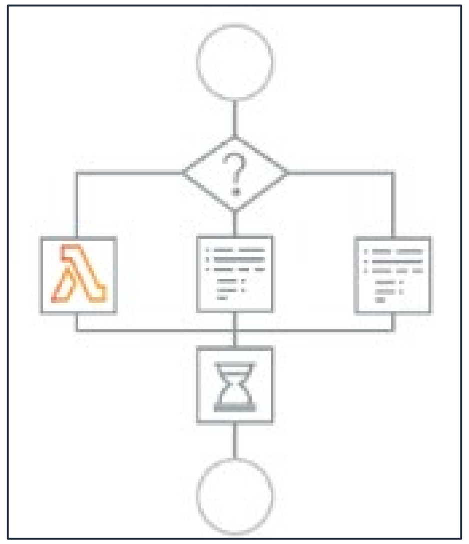
How AWS Step Functions work (continued)



Pass states pass their input as output to the next state. You can also delay execution when you need to using **wait states**.

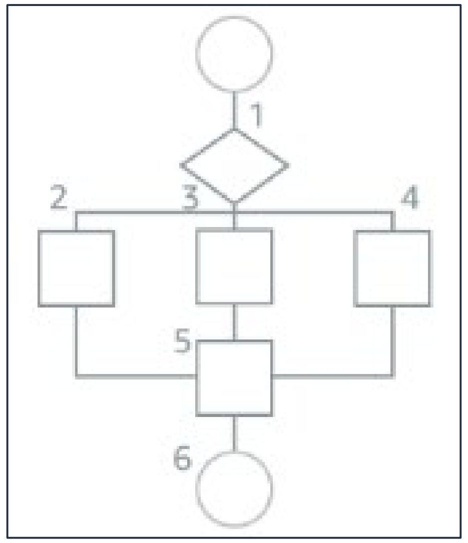


Parallel states begin multiple branches of execution at the same time, such as running multiple Lambda functions at once.

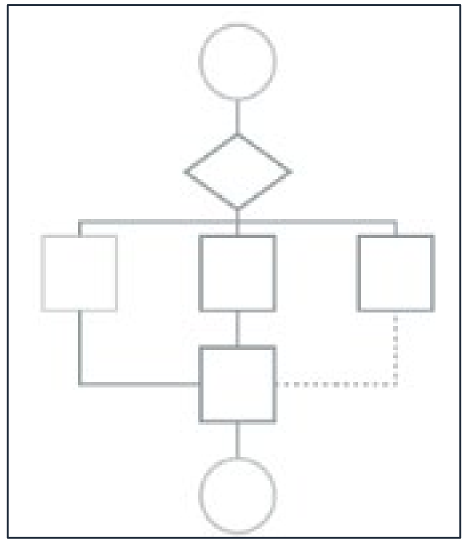


Choice states add branching logic to your state machine, and make decisions based on their input.

How AWS Step Functions work (continued)



When you execute your state machine, each move from one state to the next is called a **state transition**.



You can reuse components, easily edit the sequence of steps or swap out the code called by task states as your needs change.

Amazon States Language

https://states-language.net/spec.html

```
{
    "Comment": "A simple minimal example",
    "StartAt": "Hello World",
    "States": {
      "Hello World": {
        "Type": "Task",
        "Resource": "arn:aws:lambda...HelloWorld",
        "End": true
      },
      [. . .]
  }
}
```

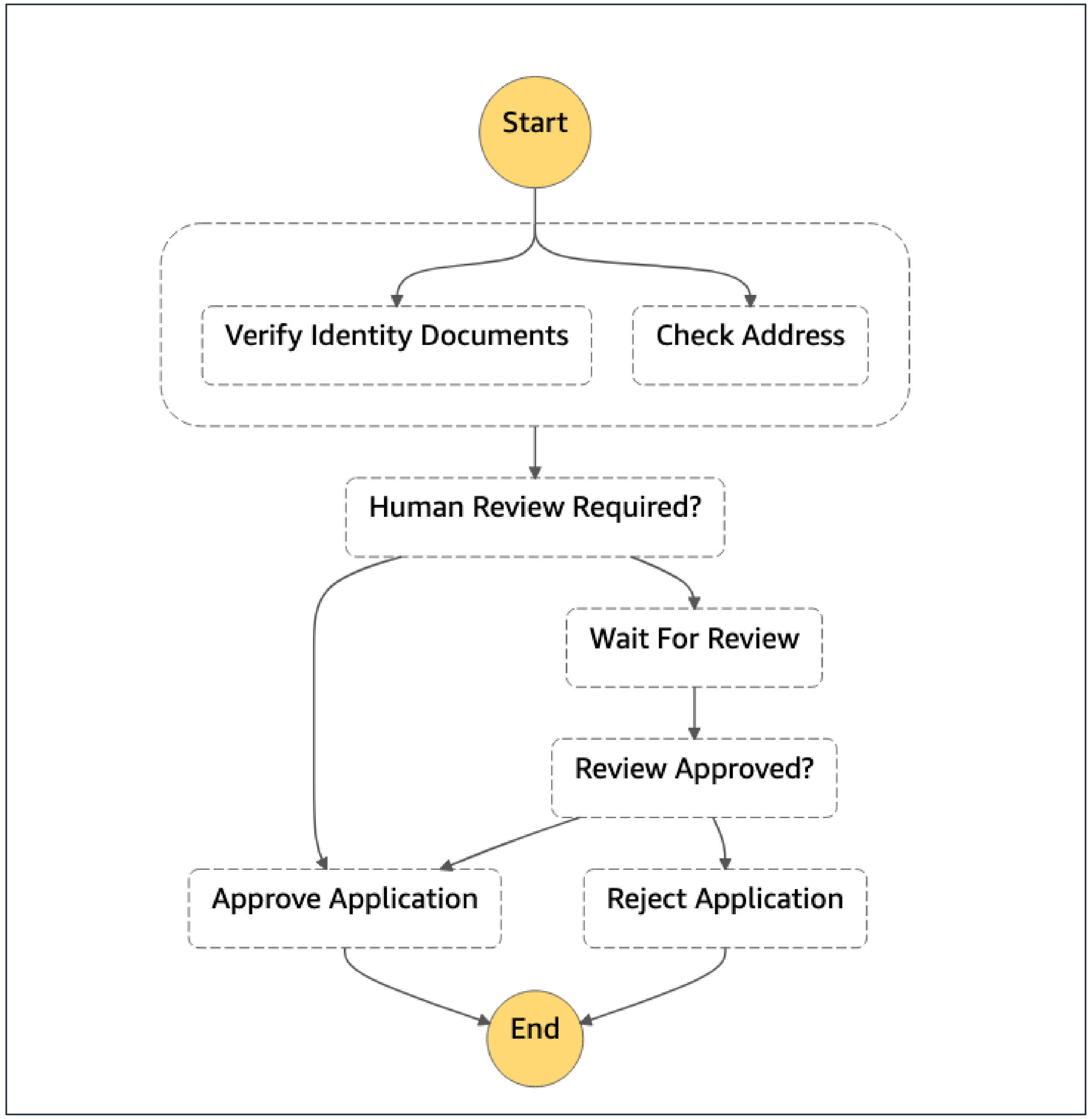
Back to our example new account workflow

Tasks

Parallel Steps

Branching Choice

Wait for a callback

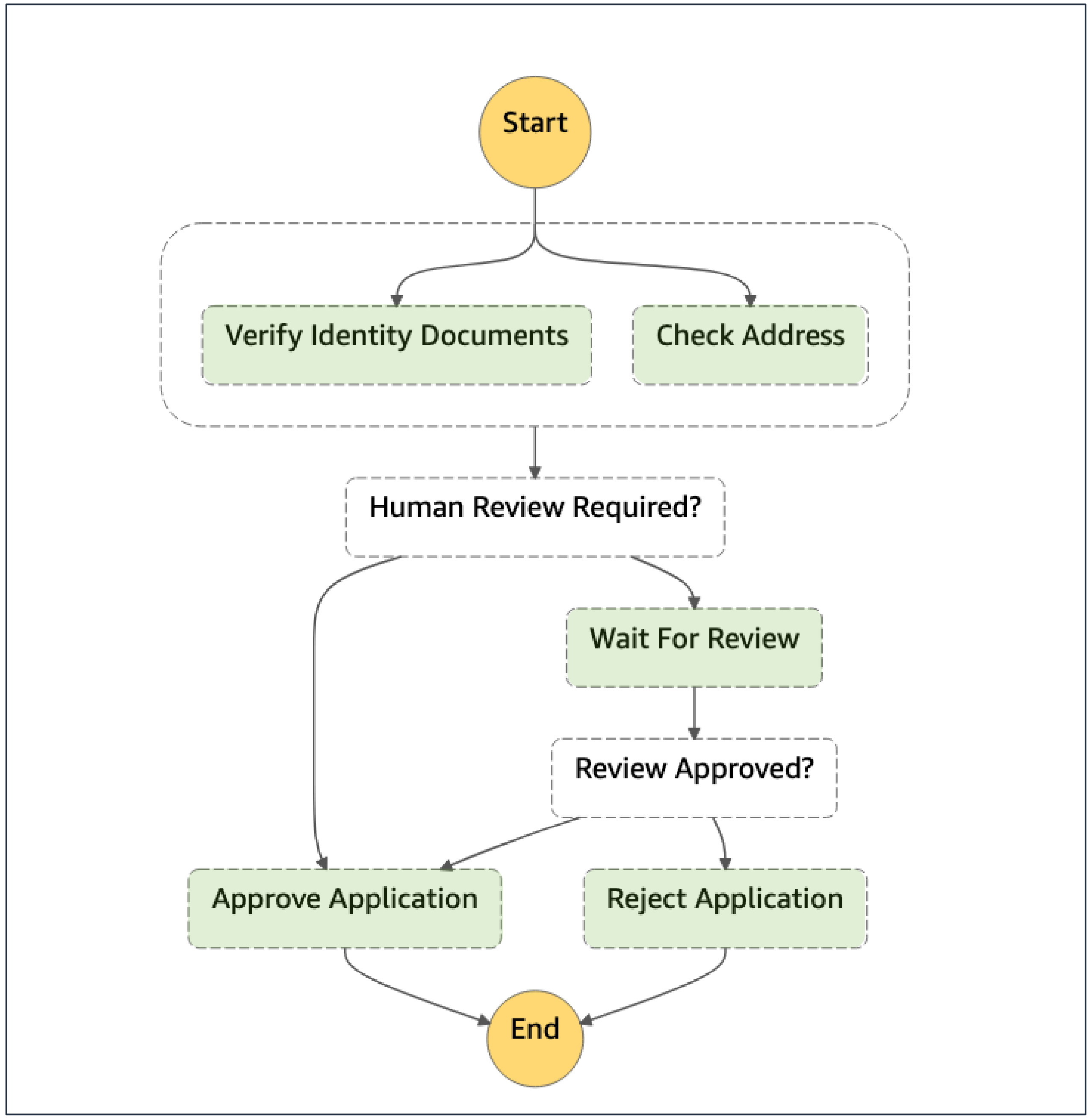


Performing a *Task*

Call an AWS Lambda Function

Wait for a polling worker to perform an activity

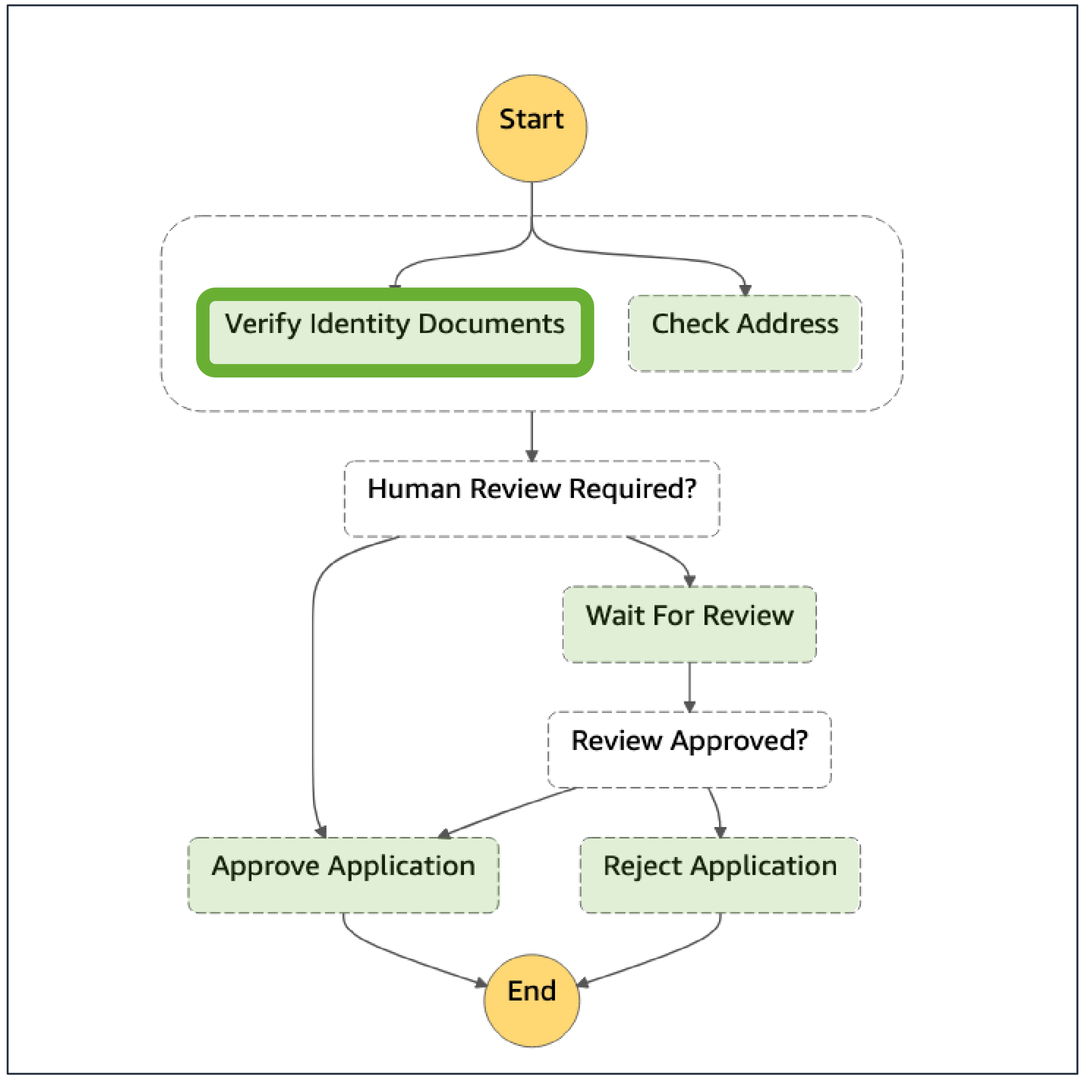
Pass parameters to an API of an integrated AWS Service



Performing a *Task*

Example: Execute a Lambda Function

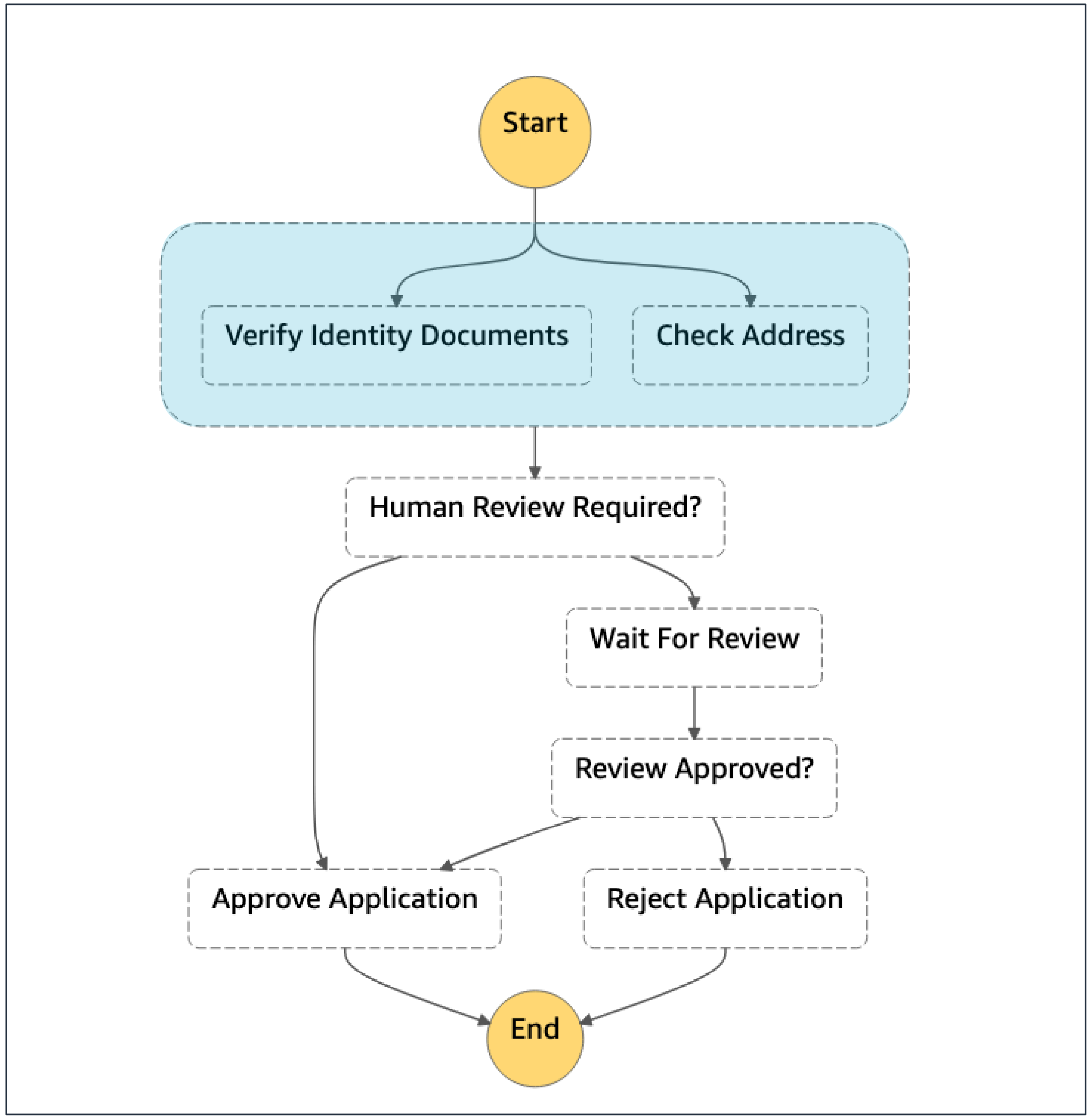
```
"Verify Identity Documents": {
  "Type": "Task",
    "Parameters": {
      "name.$": "$.application.name"
      "identityDoc.$": "$.application.idDocS3path"
    },
    "Resource": "arn:aws:lambda...VerifyIdDocs",
    "End": true
}
```



Executing branches in *Parallel*

Contains an array of state machines *branches* to execute in parallel

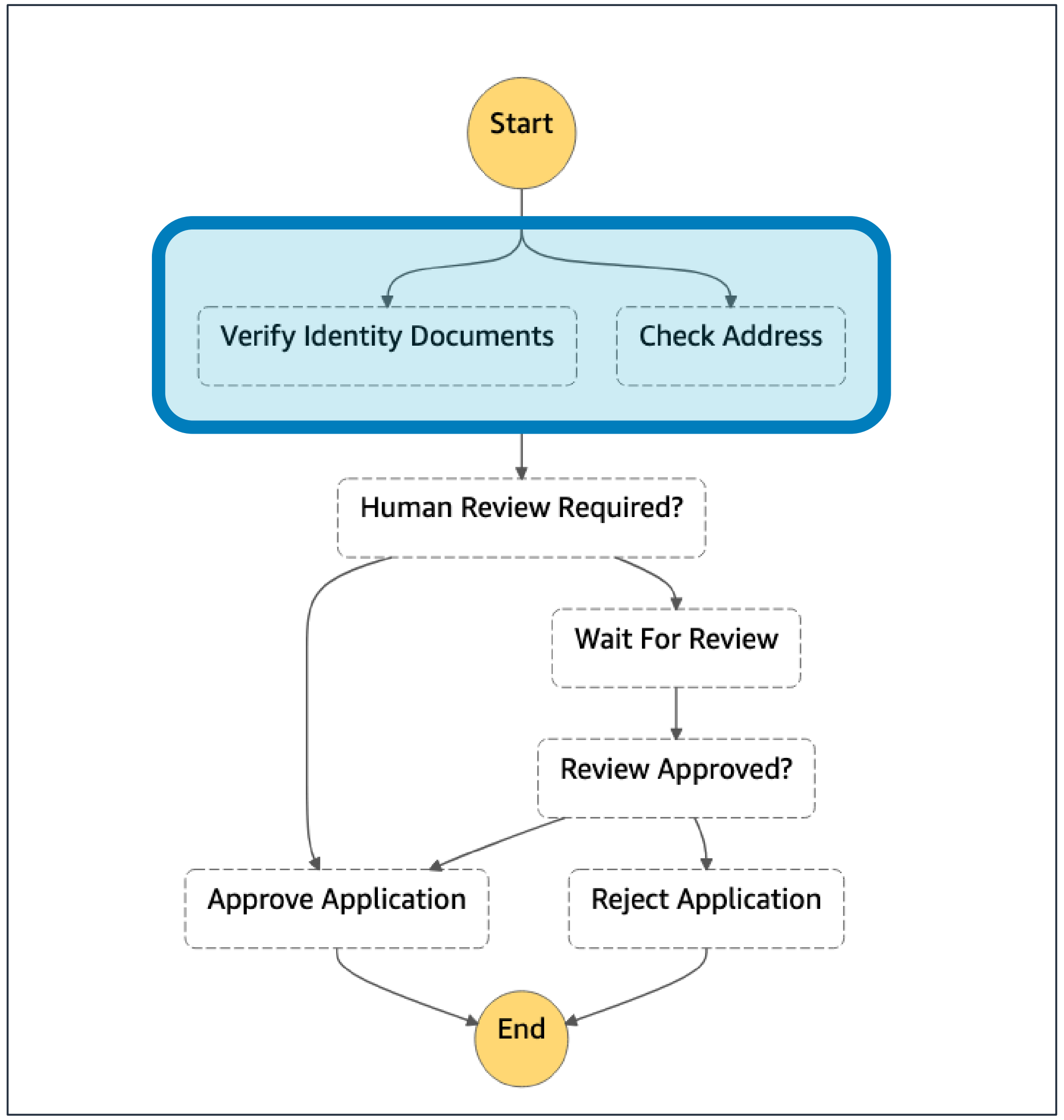
Outputs an array of outputs from each state machine in its *branches*



Executing branches in *Parallel*

Example: Run two branches in parallel

```
"Perform Automated Checks": {
  "Type": "Parallel",
    "Branches": [
      {
        "StartAt": "Verify Identity Documents",
        "States": { "Verify Identity Documents": { … } }
      },
      {
        "StartAt": "Check Address",
        "States": { "Check Address": { … } }
      }
    ]
  },
  "ResultPath": "$.checks",
  "Next": "Human Review Required?"
}
```

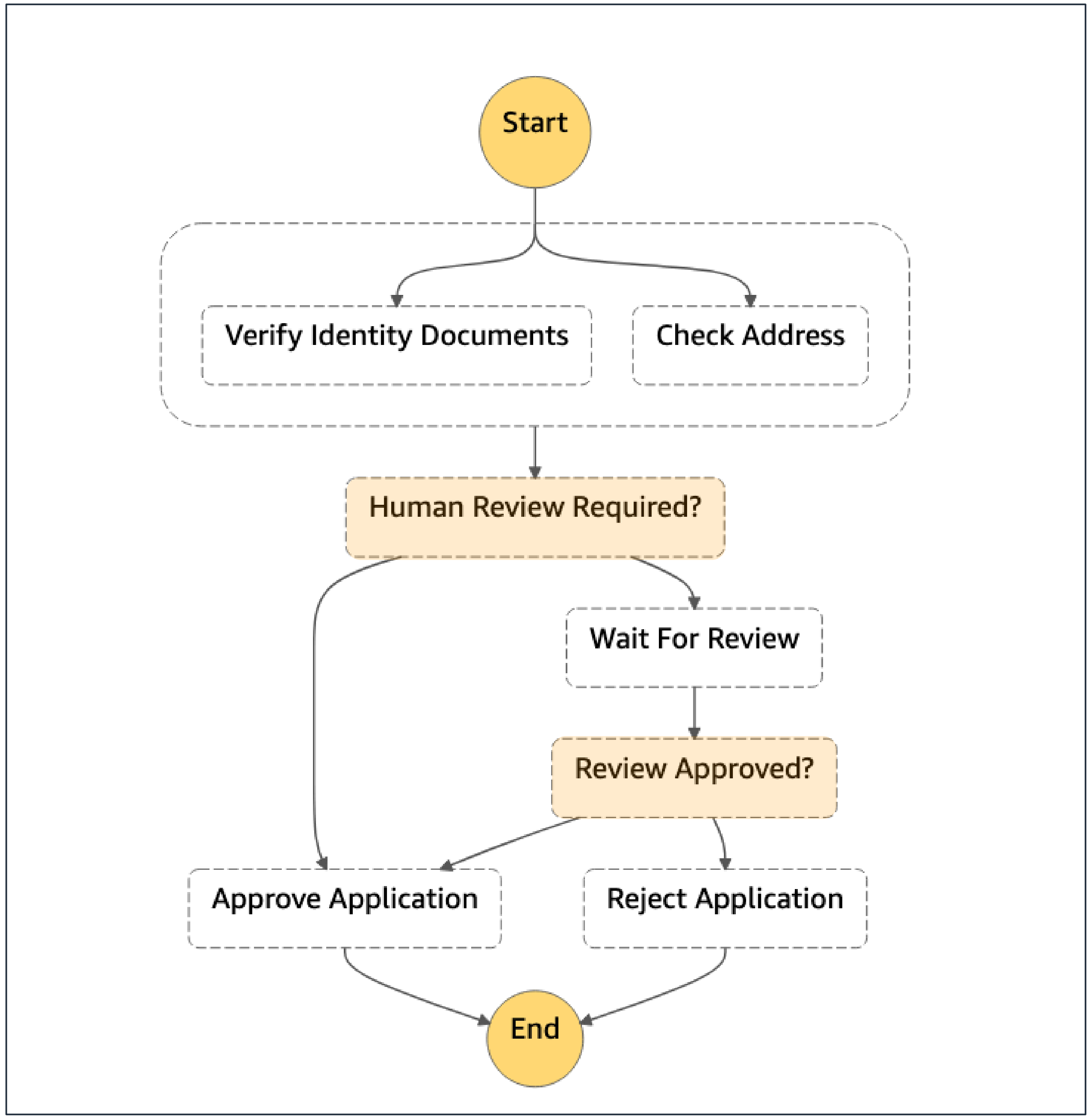


Making a *Choice*

Like a switch statement in programming

Inspects an array of *choice* expressions, comparing variables to values

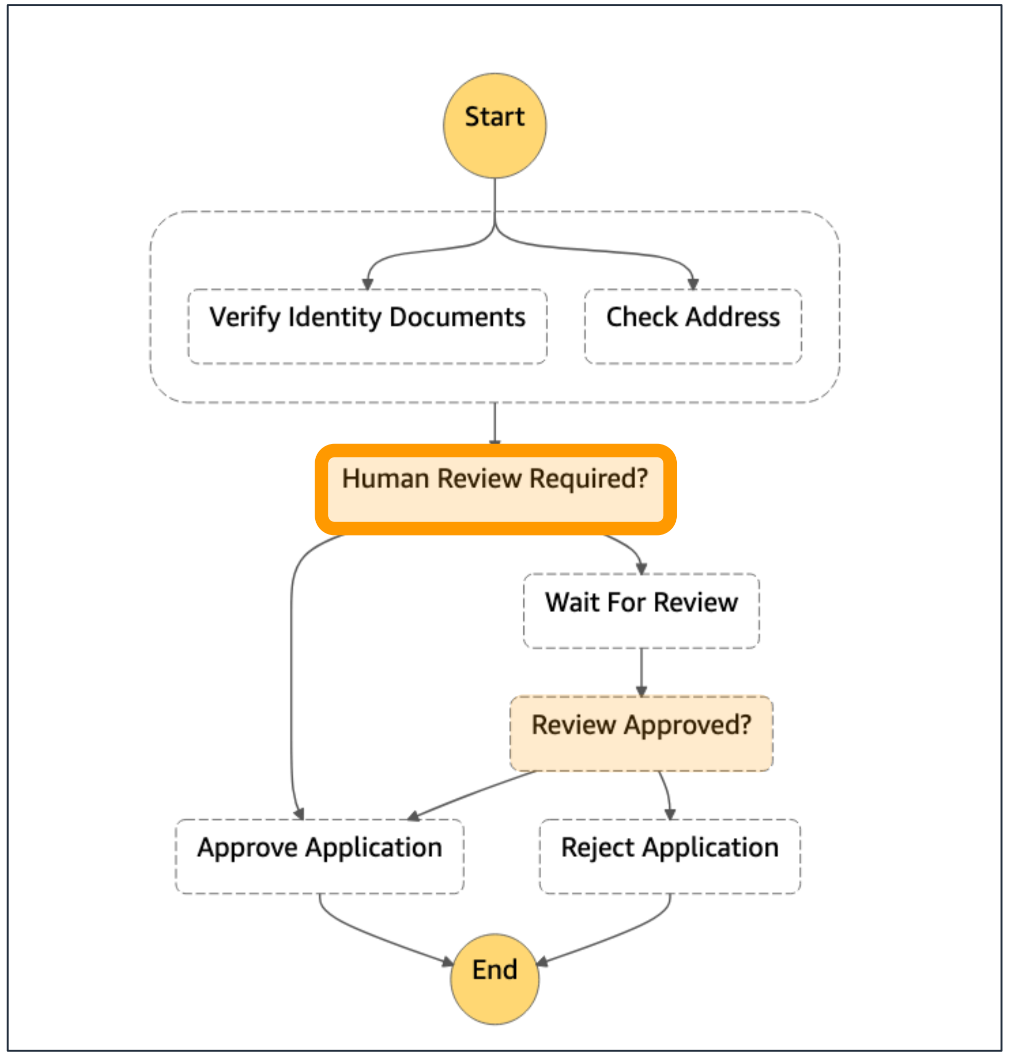
Determines which state to transition to next



Making a *Choice*

Example: Choose next step based on state outputs

```
"Human Review Required?": {
  "Type": "Choice",
  "Choices": [
    {
      "Variable": "$.checks[0].flagged",
      "BooleanEquals": true,
      "Next": "Wait For Review"
    },
    {
      "Variable": "$.checks[1].flagged",
      "BooleanEquals": true,
      "Next": "Wait For Review"
    }
  ],
  "Default": "Approve Application"
}
```

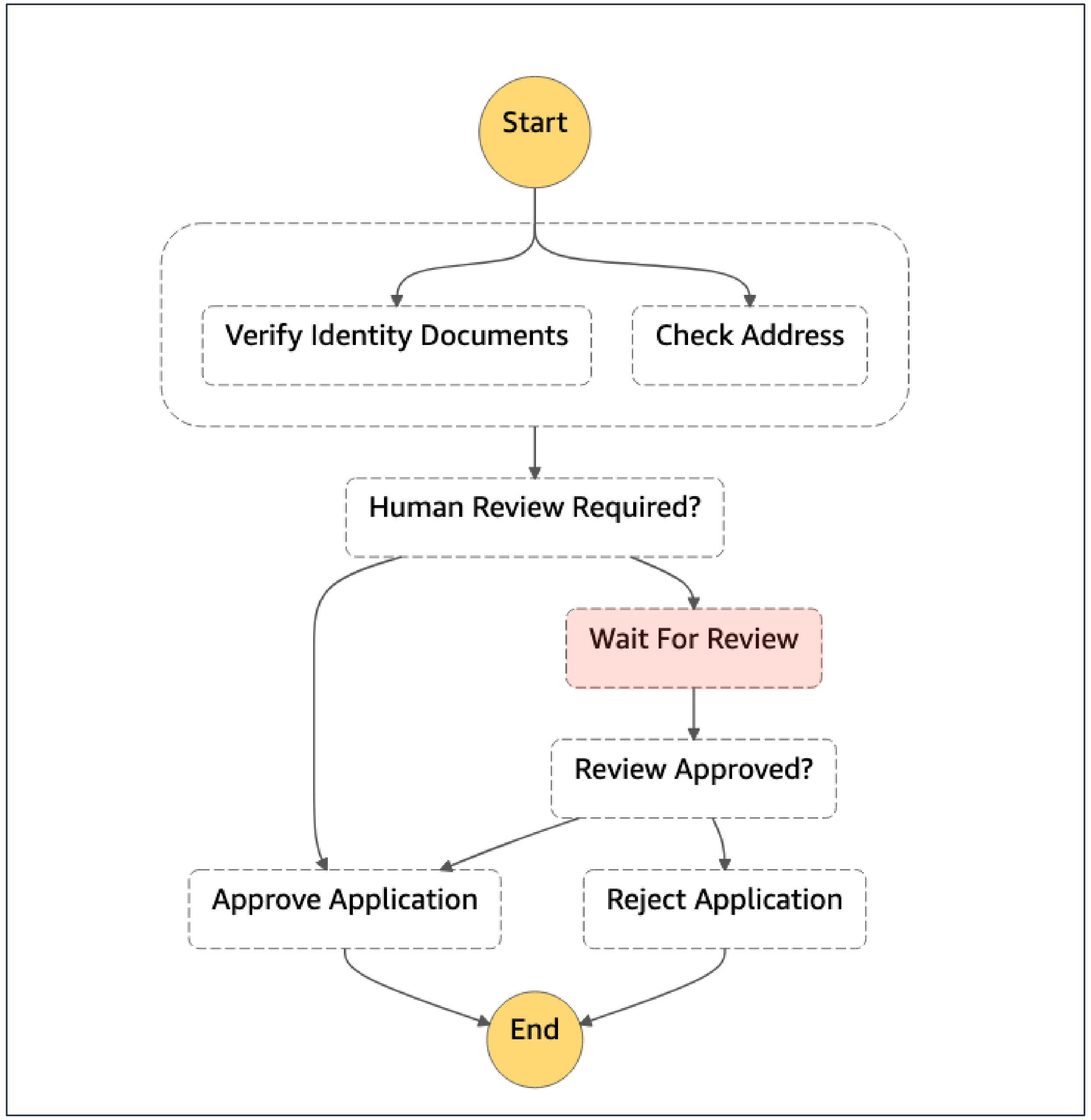


Waiting for a callback

Generates a *Task Token* and passes it to an integrated service

When the recipient process is complete, it calls *SendTaskSuccess or SendTaskFailure* with the *Task Token*

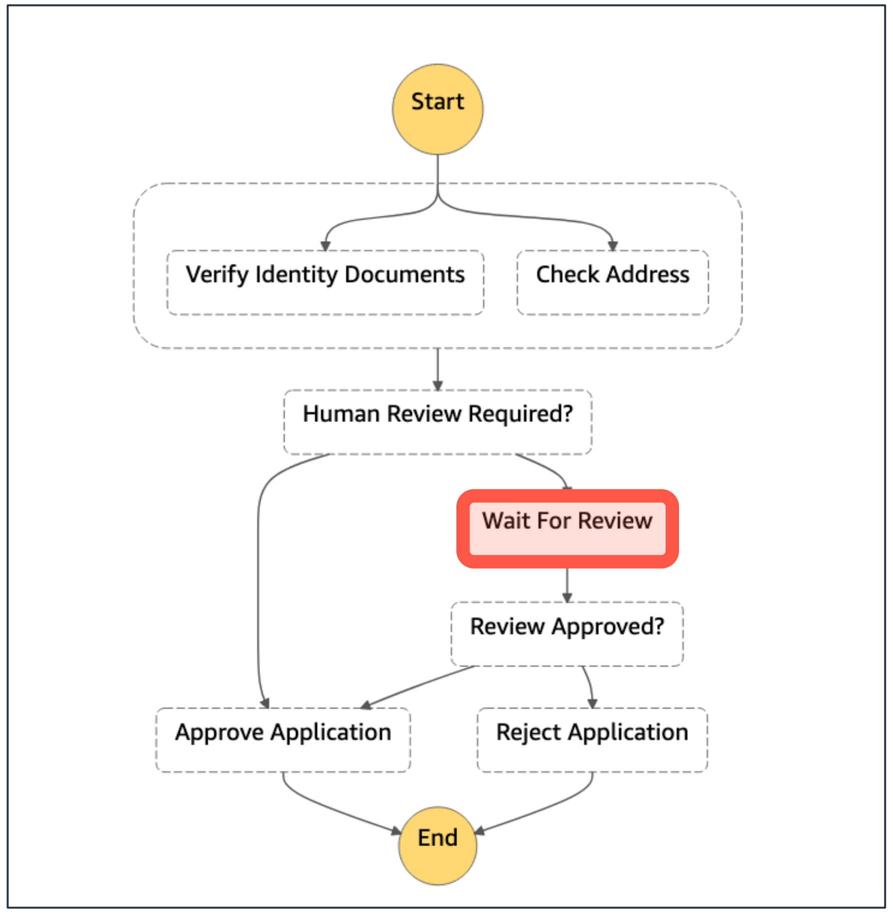
Workflow resumes its execution



Waiting for a callback

Example: Pause and wait for an external callback

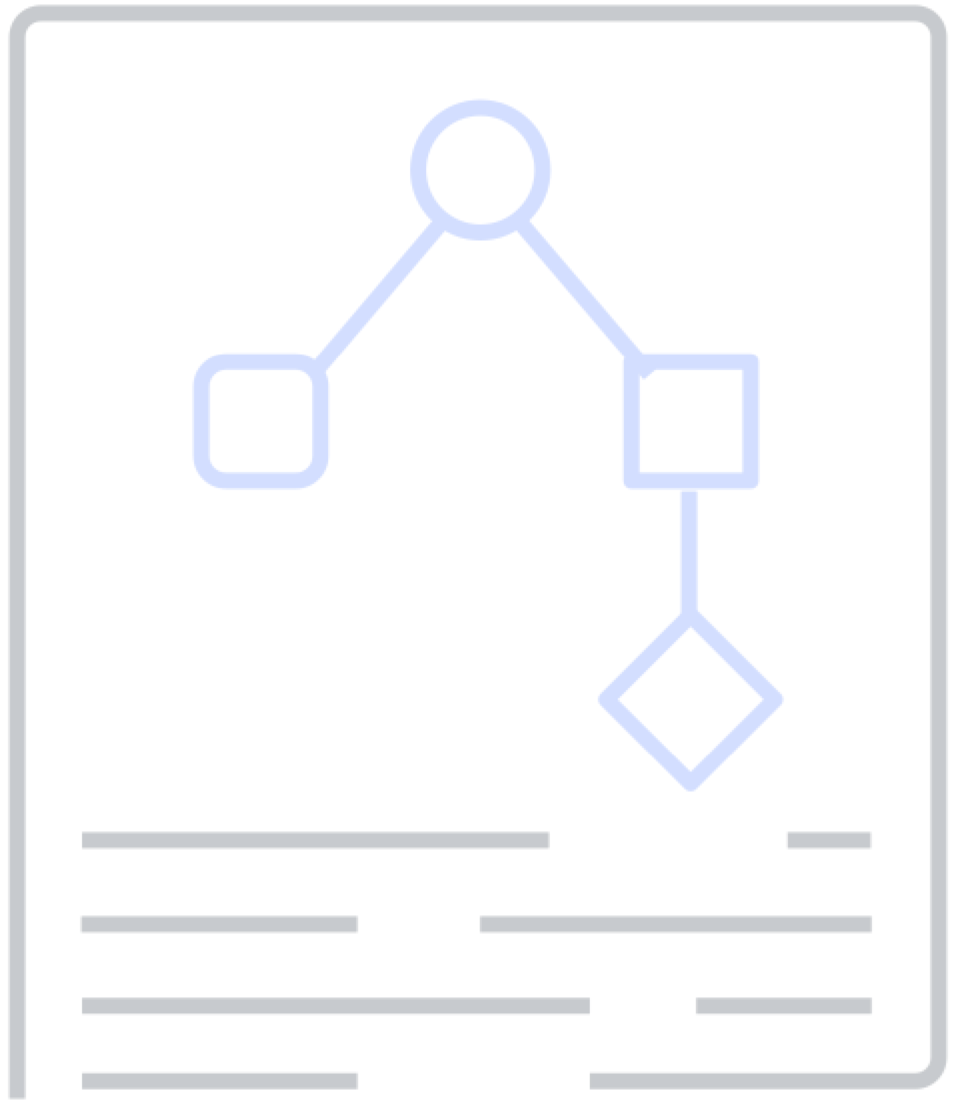
```
"Type": "Task",
"Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
"Parameters": {
    "FunctionName": "FlagApplicationForReview",
    "Payload": {
        "applicationId.$": "$.application.id",
        "taskToken.$": "$$.Task.Token"
    }
},
"ResultPath": "$.reviewDecision",
"Next": "ReviewApproved?"
```



Step Functions Diving Deeper

State Types

Task	*Execute work*
Choice	*Add branching logic*
Wait	*Add a timed delay*
Parallel	*Execute branches in parallel*
Map	*Process each of an input array's items with a state machine*
Succeed	*Terminate successfully or ends a branch of Parallel or an iteration of Map*
Fail	*Terminate the state machine and mark execution as a failure*
Pass	*Passes input to output*



Step Functions service integrations

AWS
Lambda

Amazon
Elastic Container Service

AWS
Batch

Amazon
DynamoDB

AWS
Glue

Amazon
SageMaker

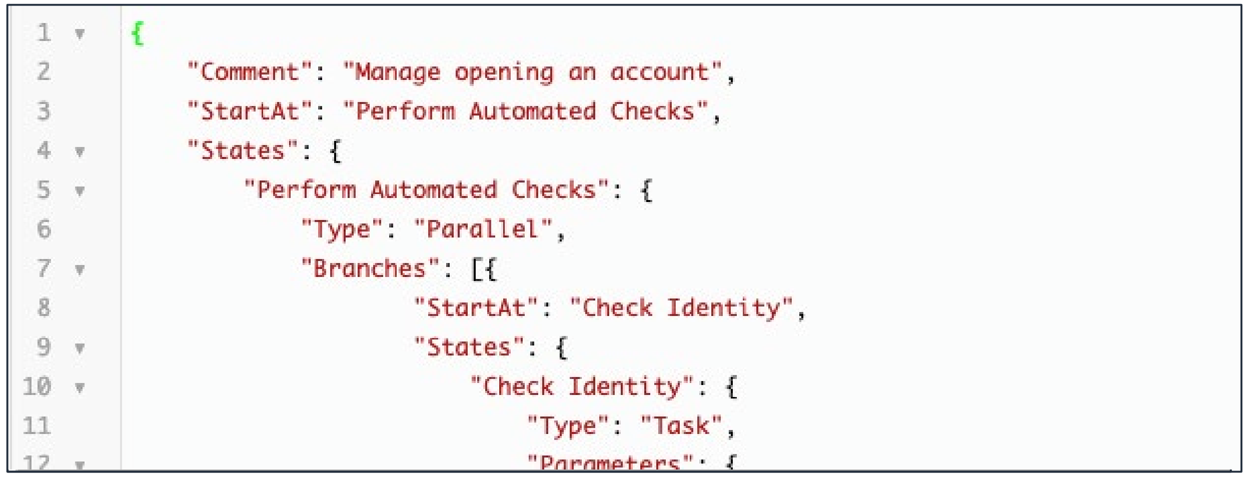
AWS
Step Functions

Amazon
Simple Notification Service

Amazon
Simple Queue Service

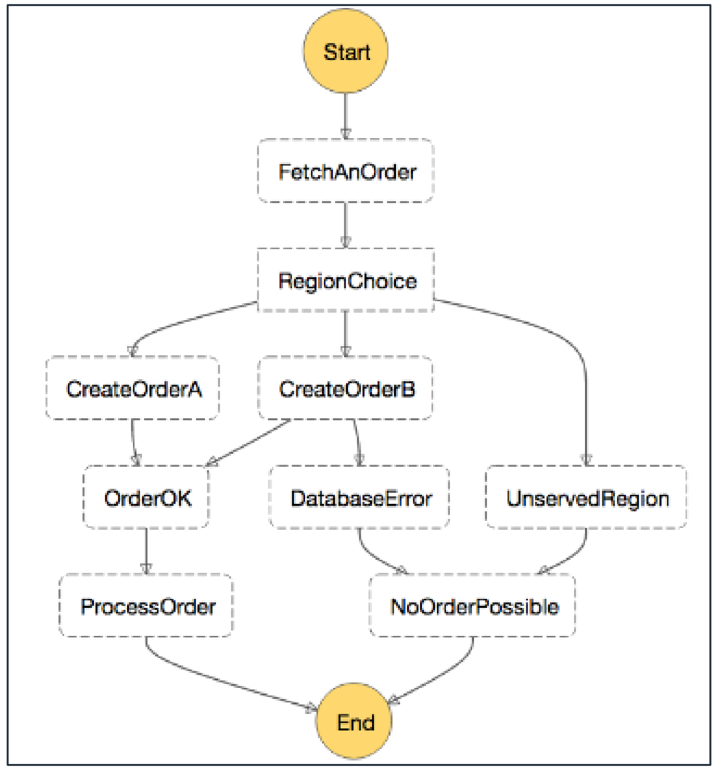
Working with Step Functions

Define in JSON

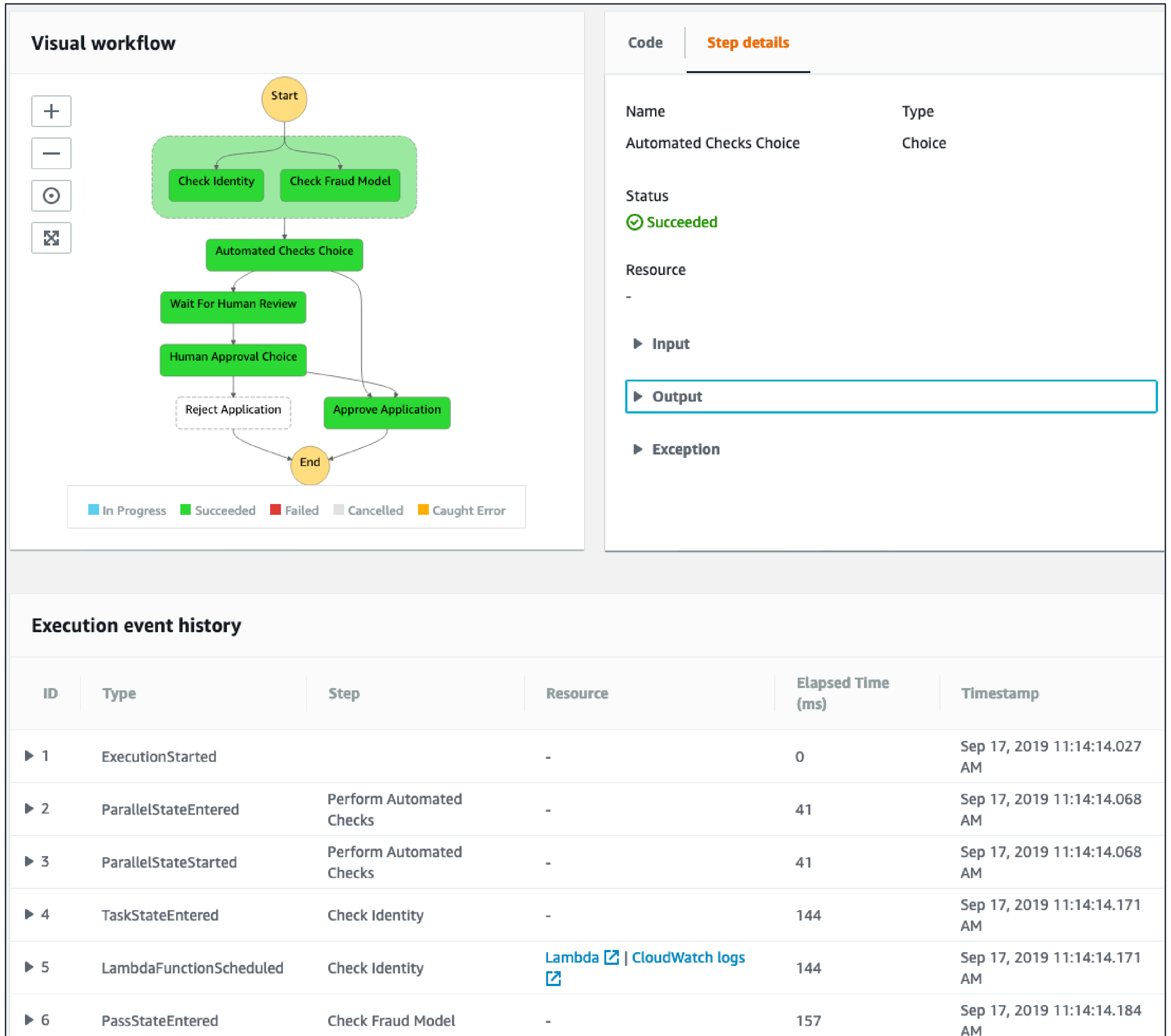


```
{
    "Comment": "Manage opening an account",
    "StartAt": "Perform Automated Checks",
    "States": {
        "Perform Automated Checks": {
            "Type": "Parallel",
            "Branches": [{
                    "StartAt": "Check Identity",
                    "States": {
                        "Check Identity": {
                            "Type": "Task",
                            "Parameters": {
```

Visualize in the Console



Monitor Executions



Name: Automated Checks Choice
Type: Choice
Status: Succeeded
Resource: -

Execution event history

ID	Type	Step	Resource	Elapsed Time (ms)	Timestamp
1	ExecutionStarted		-	0	Sep 17, 2019 11:14:14.027 AM
2	ParallelStateEntered	Perform Automated Checks	-	41	Sep 17, 2019 11:14:14.068 AM
3	ParallelStateStarted	Perform Automated Checks	-	41	Sep 17, 2019 11:14:14.068 AM
4	TaskStateEntered	Check Identity	-	144	Sep 17, 2019 11:14:14.171 AM
5	LambdaFunctionScheduled	Check Identity	Lambda | CloudWatch logs	144	Sep 17, 2019 11:14:14.171 AM
6	PassStateEntered	Check Fraud Model	-	157	Sep 17, 2019 11:14:14.184 AM

Error Handling

Failures can happen due to *Timeouts, Failed Tasks, or Insufficient Permissions*

Tasks can *Retry* when errors occur using a *BackoffRate* up to *MaxAttempts*

Tasks can *Catch* specific errors and transition to other states

Development Tips

Step Functions Local
https://docs.aws.amazon.com/step-functions/latest/dg/sfn-local.html

Statelint
https://github.com/awslabs/statelint

Serverless Framework Plug-in
https://github.com/horike37/serverless-step-functions

Visual Studio Code
aws-step-functions-constructor extension
https://marketplace.visualstudio.com/items?itemName=paulshestakov.aws-step-functions-constructor

Step Functions In Action

"AWS Step Functions gives us a **reliable, automated way of orchestrating very complex queries and processes between all our distributed systems**," Brown says. "We saved time and money by making it easy for our developers to build applications using AWS Lambda functions, giving them **more productivity and agility**. We also get a visual representation of the logic for each workflow, which makes it **easier when discussing the solution with nontechnical stakeholders** at the company."

Paul Brown
Senior Developer Manager

Workflows managed with Step Functions

Automating subscriber account deletions across many distributed systems

Receiving customer orders while external billing and payment services are offline

Running an extract, transform, and load (ETL) newspaper-fulfillment pipeline through a series of Lambda functions

https://aws.amazon.com/solutions/case-studies/the-guardian/

Shortened processing time for updating nutrition labels from 36 hours down to 10 seconds

Data validation and transformation steps are designed visually with non-technical personnel

Validation and transformation steps verified in real-time as data flows through the state machine in real time

Process optimizations are identified and implemented on the spot

https://www.youtube.com/watch?v=sMaqd5J69Ns

AWS Step Functions Key Benefits

Fully-managed service

High availability & automatic scaling

Visual monitoring & state management

Auditable history of each execution

Built-in error handling

Pay per use

Where to learn more

Get started building with AWS Step Functions

Create a Serverless Workflow ~10 minutes

https://aws.amazon.com/getting-started/tutorials/create-a-serverless-workflow-step-functions-lambda

Developer Guide ~2 hours

https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

Reference Architectures

https://aws.amazon.com/step-functions/resources/

Your modern application development journey starts with AWS Training and Certification

Training
Developing on AWS is where you will learn how to use the AWS SDK to develop secure and scalable cloud applications. We will explore how to interact with AWS using code and discuss key concepts, best practices, and troubleshooting tips.

AWS Certified Developer – Associate

Developers with one or more years of hands-on experience on AWS

This exam validates an understanding of core AWS services, uses, and basic AWS architecture best practices. Examinees must demonstrate proficiency in developing, deploying, and debugging cloud-based applications using AWS.

Visit https://www.aws.training/

AWS Certified DevOps Engineer – Professional

DevOps engineers with two or more years of experience on AWS

This exam tests an engineer's experience provisioning, operating, and managing AWS environments. Examinees will show an understanding of how to build highly scalable, available, and self-healing systems on the AWS platform and to design, manage, and maintain tools to automate operational processes.

Thank You for Attending
AWS Online Event: Modern Application Development

We hope you found it interesting! A kind reminder to **complete the survey.**
Let us know what you thought of today's event and how we can improve the event experience for you in the future.

Gabe Hollombe
Sr. Technical Evangelist, AWS
@gabehollombe

aws-apac-marketing@amazon.com
twitter.com/AWSCloud
facebook.com/AmazonWebServices
youtube.com/user/AmazonWebServices
slideshare.net/AmazonWebServices
twitch.tv/aws