12

Orchestration Scopes and Fault Handling

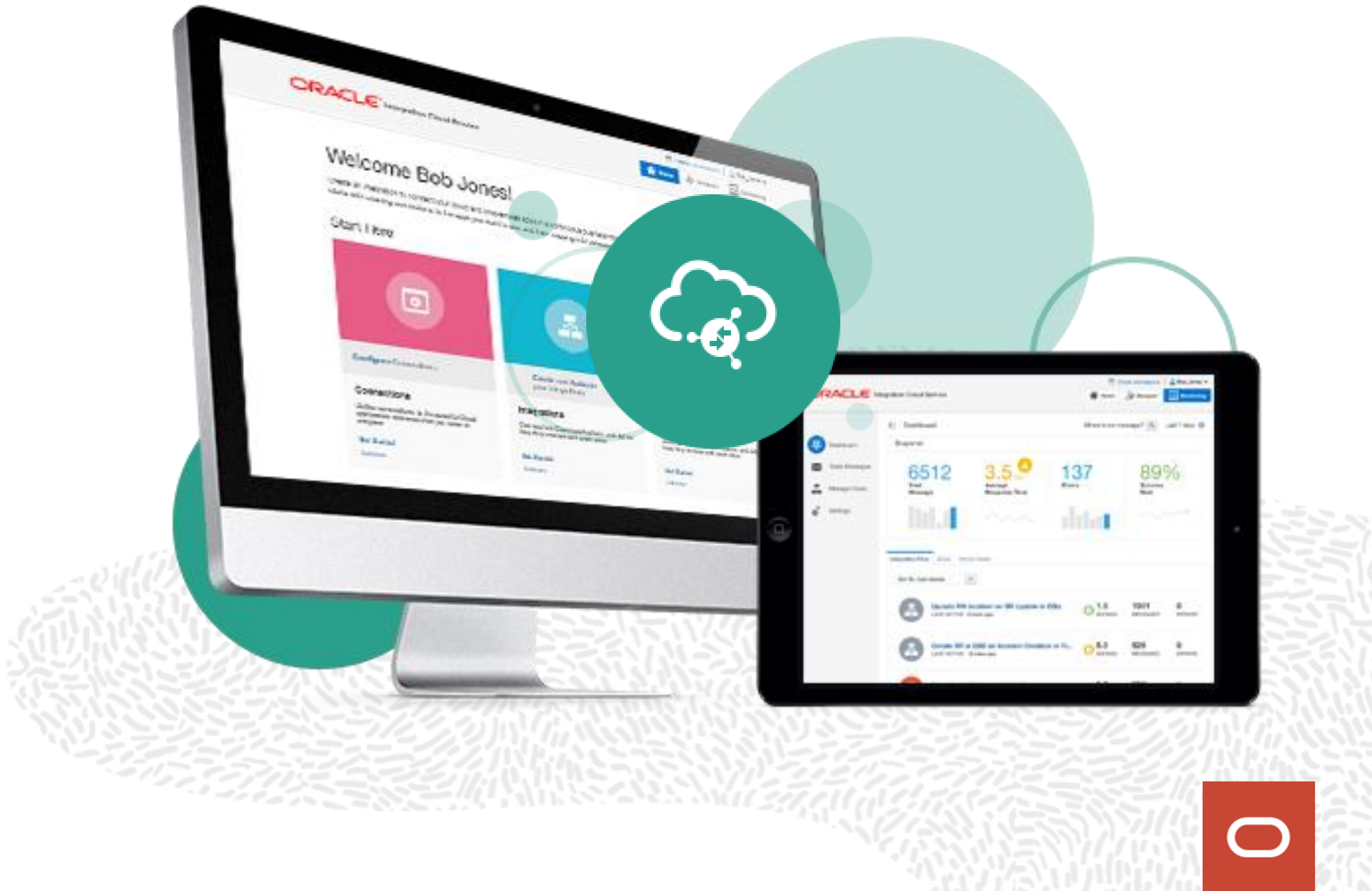
Objectives

After completing this lesson, you should be able to:

- Leverage Scope and nested Scope containers
- Describe the role and behavior of the OIC error hospital
- Implement error handling logic in the Global fault handler
- Design error handling strategies using Scope fault handlers
- Access fault information within fault handlers
- Explain the strategies for extended error handling use cases

Agenda

- Understanding Scope Containers
- Using Fault Handlers
- Managing Failed Instances



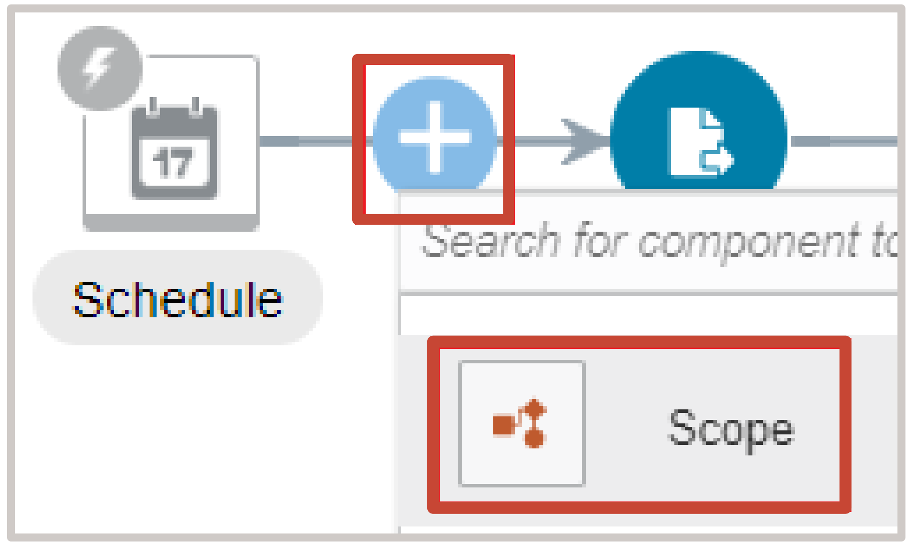
Scope Containers

Use a **Scope** within an orchestration style integration flow to:

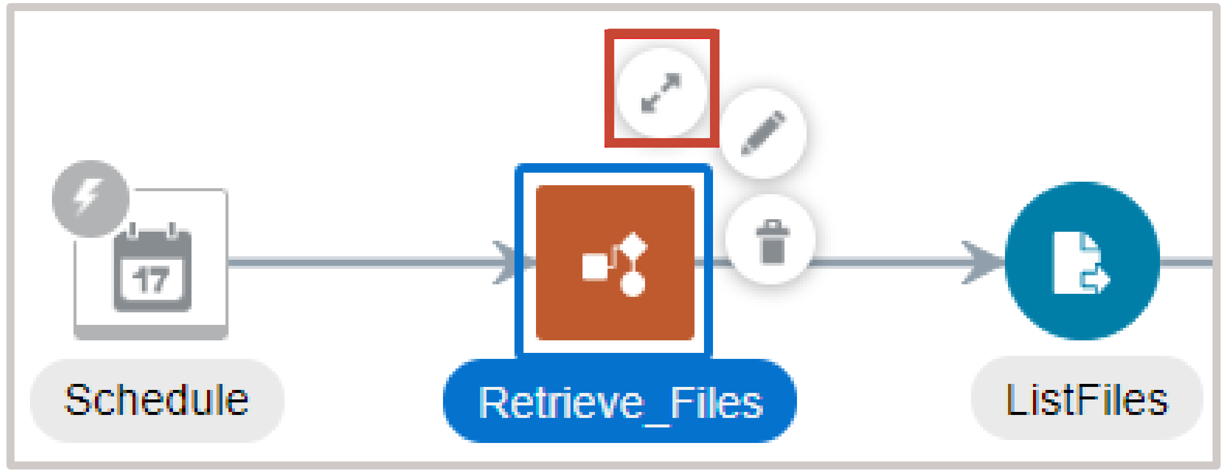
- Organize related invokes and actions
- Provide for scope-level fault handling logic

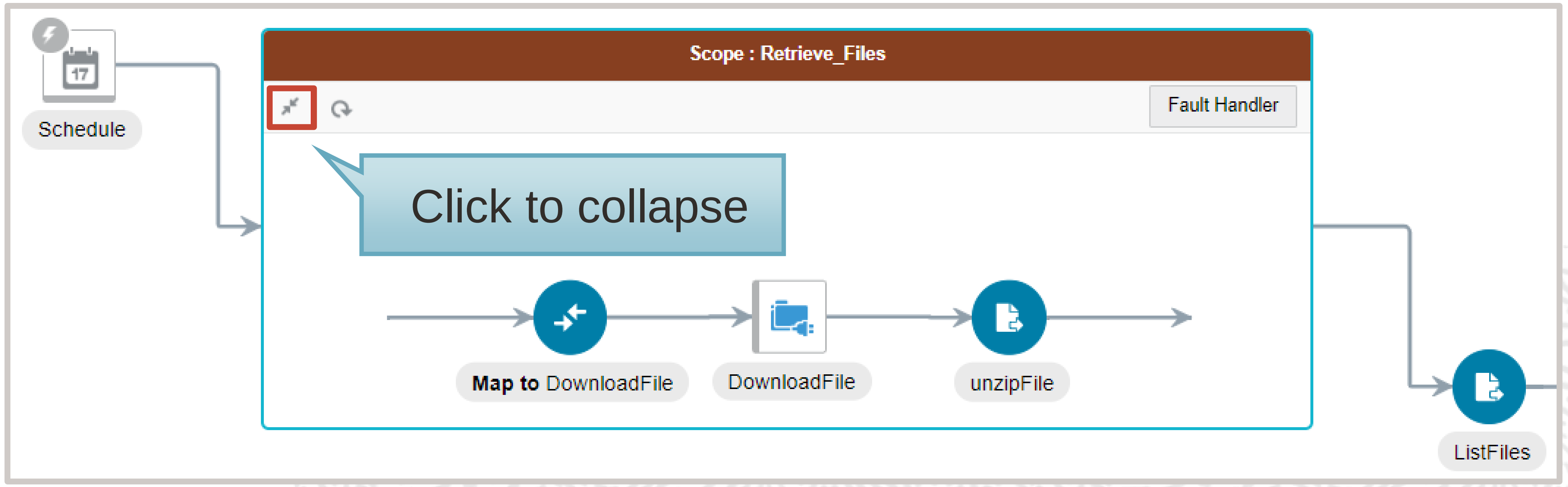
Scope containers can be expanded or collapsed as needed for visibility.

- Must be collapsed if you need to reposition to another location



Click to expand

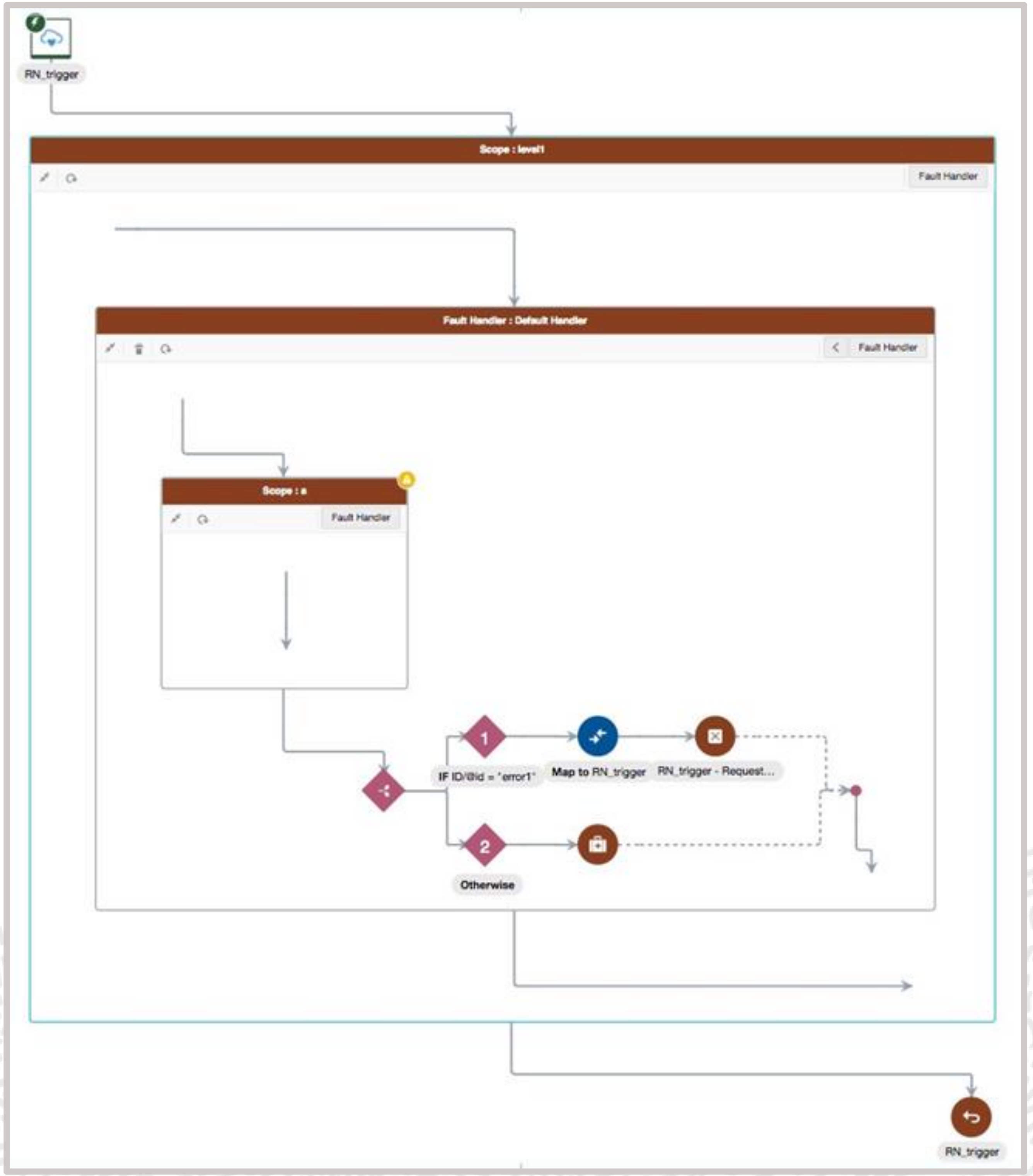




Nested Scopes

You can add nested (child) scope actions to a basic scope action.

- Provides a more sophisticated way of organizing or separating actions into a subsection of the integration
- Allows for grouping nested child activities, which have their own variables as well as fault handlers
- No limitation to the levels of nesting; even the scope's fault handlers can have nested scopes



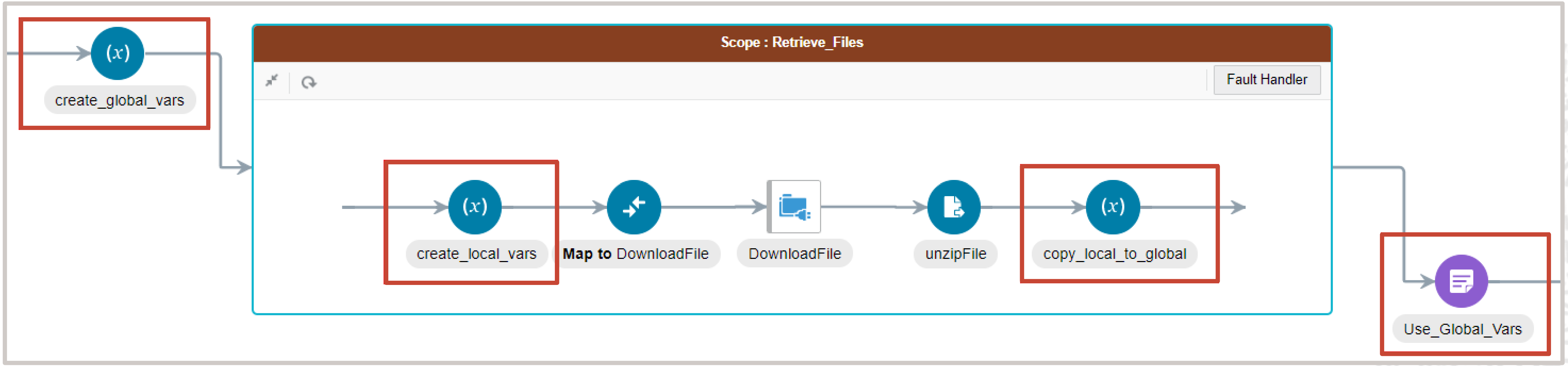
Scope Context Considerations

Your design logic needs to consider the following:

- Returned data from an invoked connection will not be available outside the scope's container.
- New variables created within a scope will not be visible outside the scope.
- Runtime or business faults will be caught by that scope's fault handlers (if configured).

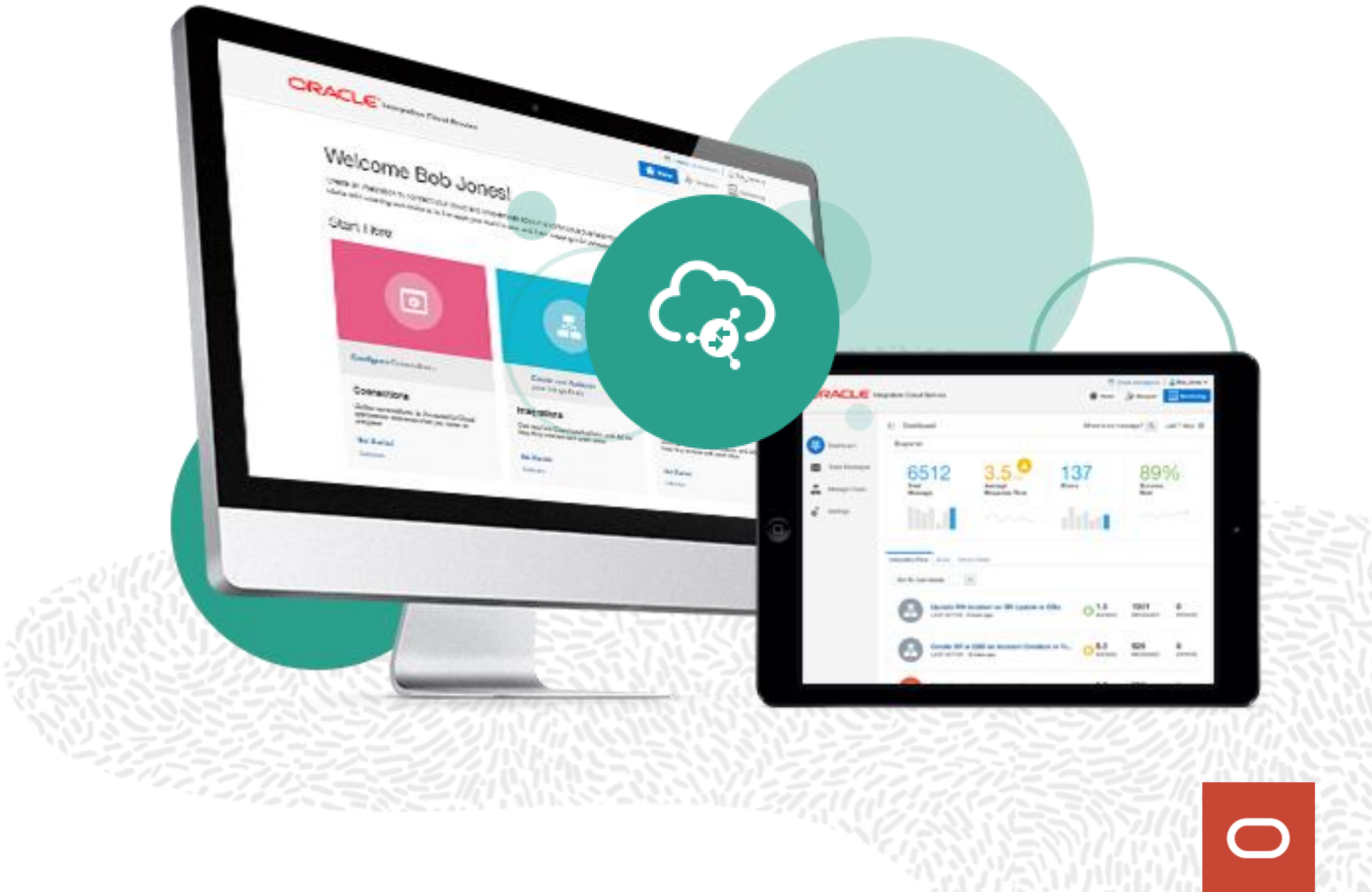
If needed, create one or more global variables.

- Used to "pass" local variables and/or invoked responses outside the scope's container



Agenda

- Understanding Scope Containers
- Using Fault Handlers
- Managing Failed Instances



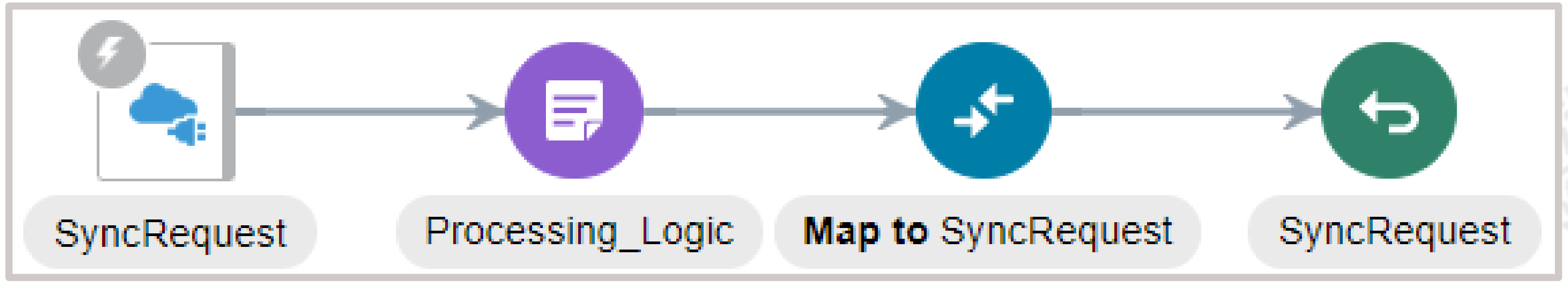
Faults in Integration Flows

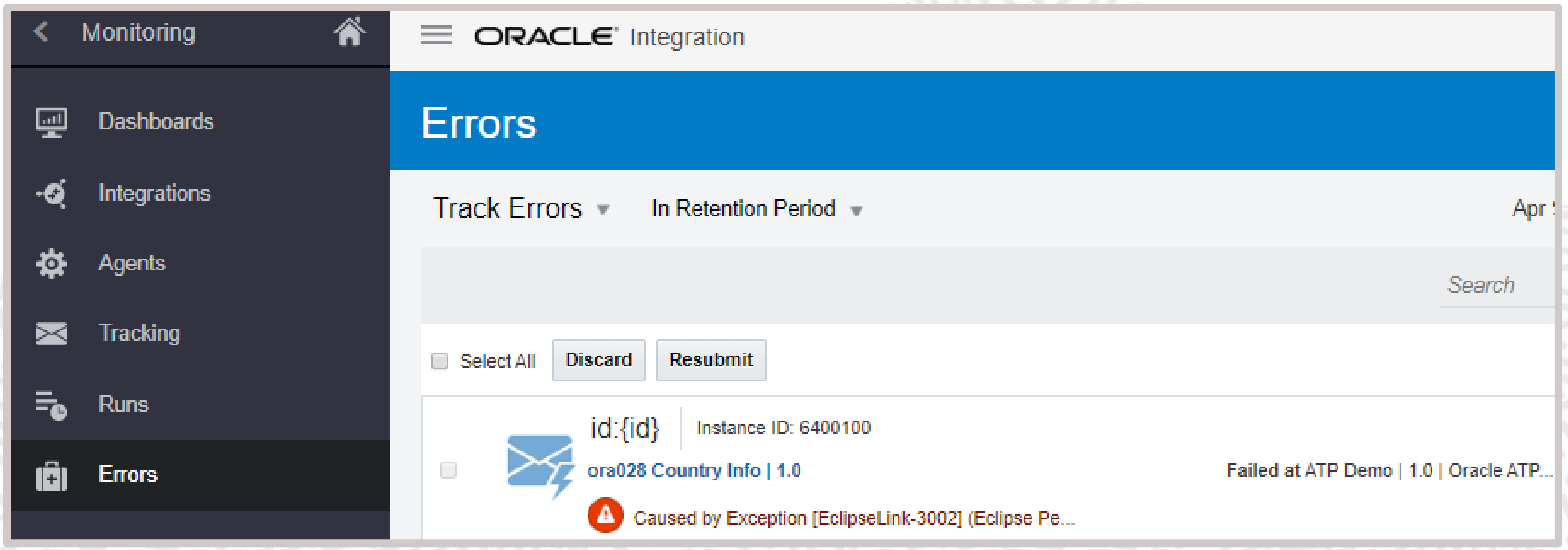
By default, all faults are caught by the OIC error hospital. These include:

- Explicit runtime or business faults returned from invoking an external service or system
- Other runtime faults encountered by OIC
  - *(for example, Request timeout, Invalid inbound trigger payload)*
- Internal runtime errors executing orchestration actions
  - *(for example, Data mapping, calling JavaScript functions)*

Faults caught by the error hospital are visible in the Monitoring portal on the **Errors** page.

- In addition, synchronous integration flow clients will receive that fault as a response.





Errors

Track Errors ▾ In Retention Period ▾ Apr

id:{id} | Instance ID: 6400100

ora028 Country Info | 1.0

Failed at ATP Demo | 1.0 | Oracle ATP...

Caused by Exception [EclipseLink-3002] (Eclipse Pe...

Designing Beyond the “Happy Path”

Instead of allowing the error hospital to catch all faults, you should:

- Intentionally catch all faults in either of the following:
  - The integration’s Global fault handler
  - One or more Scope fault handlers
- Add fault handling logic within the fault handlers based on use case requirements. For example:
  - Invoke a secondary service for backup processing
  - Log the error but continue on with the integration flow
  - Log the error and then terminate the integration flow
  - Invoke another service for notification or error handling processing
  - Reply to the integration flow’s client with a custom error response
  - Send an email notification to an external stakeholder or an internal administrator
  - Invoke an OIC process to initiate a workflow involving manual intervention

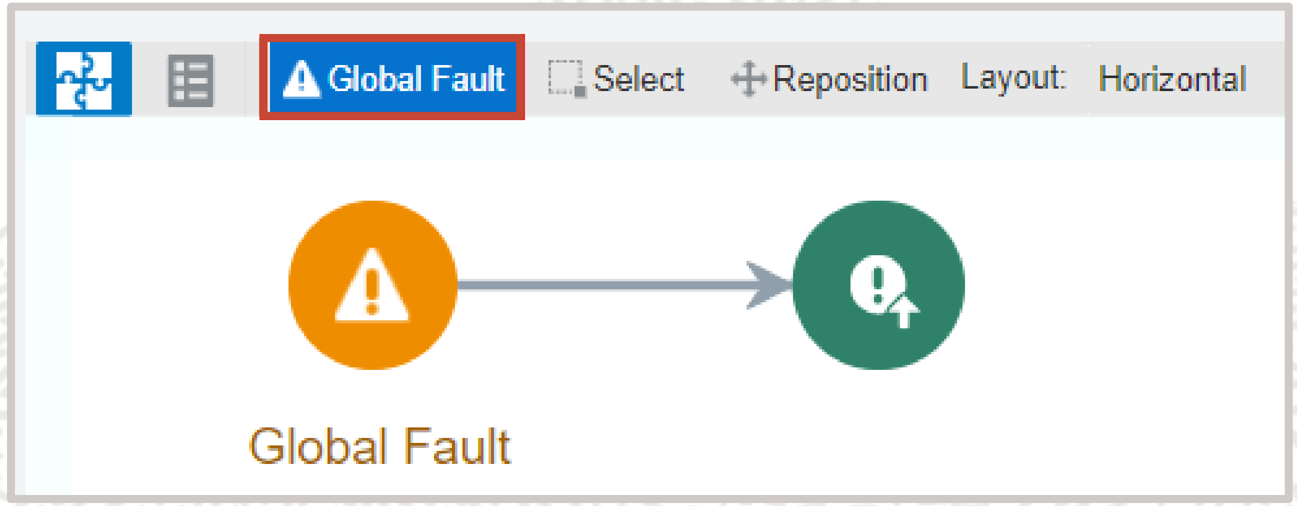
Global Fault Handler

The Global fault handler will catch all faults that are:

- Not caught by a Scope fault handler
- Explicitly thrown with a **Throw New Fault** action
  - From the main flow or from a Scope fault handler
- Explicitly thrown with a **Re-throw Fault** action
  - This action is available only within fault handers

Click the **Global Fault** button to enter the Global Fault Handler edit canvas:

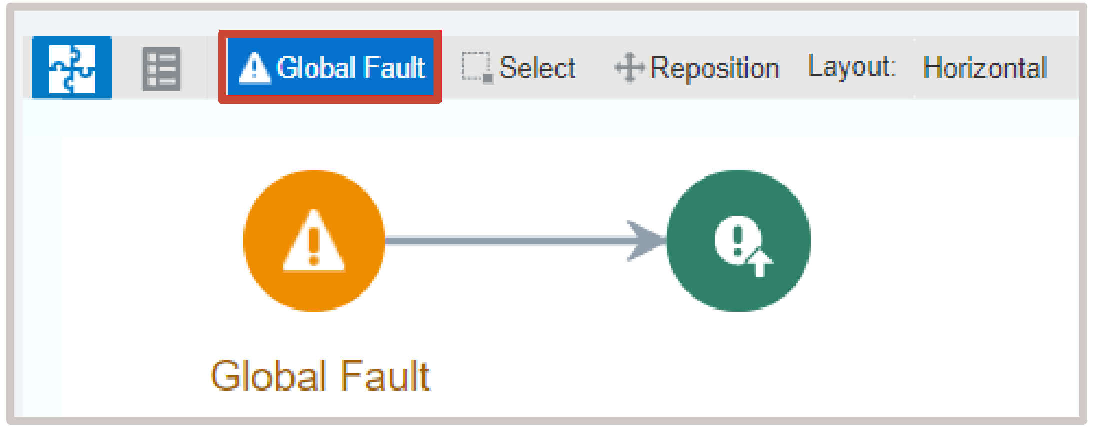
- The default implementation rethrows the fault to the error hospital.

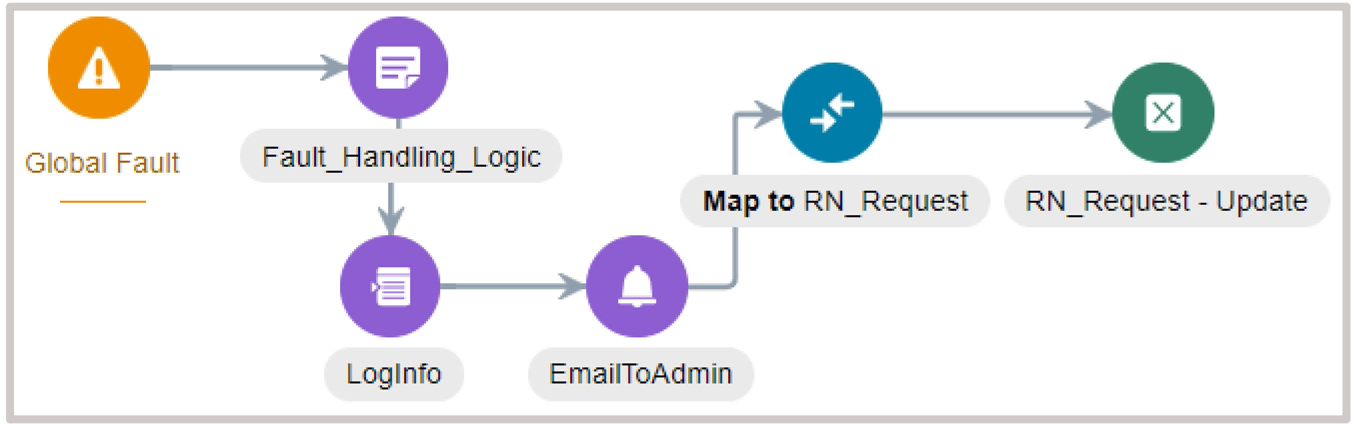


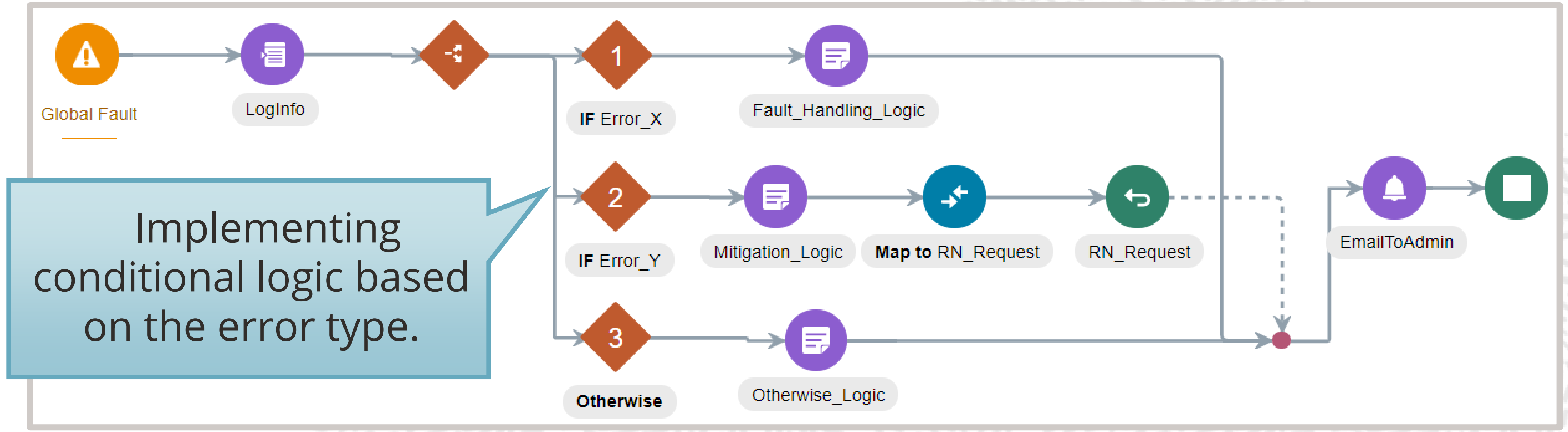
Fault Handling Logic

Change the default fault handling logic appropriate for your integration. Examples include:

- Invoke an “error handling” service such as an OIC process, another OIC integration, external web service, etc.
- Add one more action, such as an email notification.
- Map custom data for a fault return or callback *(not available for one-way integrations)*.
- Mitigate the error condition and return a successful response.







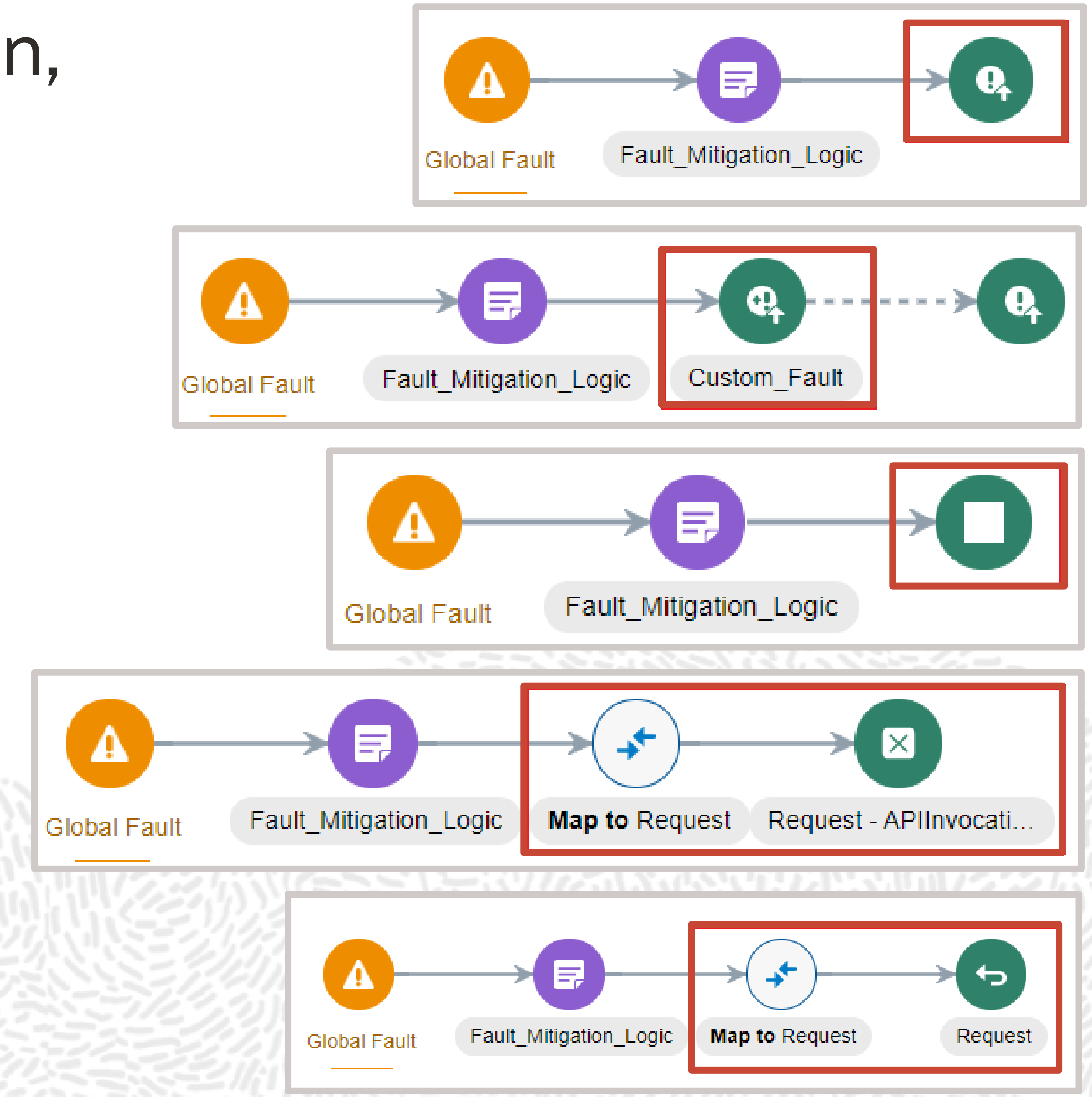
Defining How to End the Integration Flow

End action options will depend on the integration's message exchange pattern.

- *(sync, async w/ callback, async 1-way or scheduled orchestration)*

Instead of using the default **Re-Throw Fault** end action, you can:

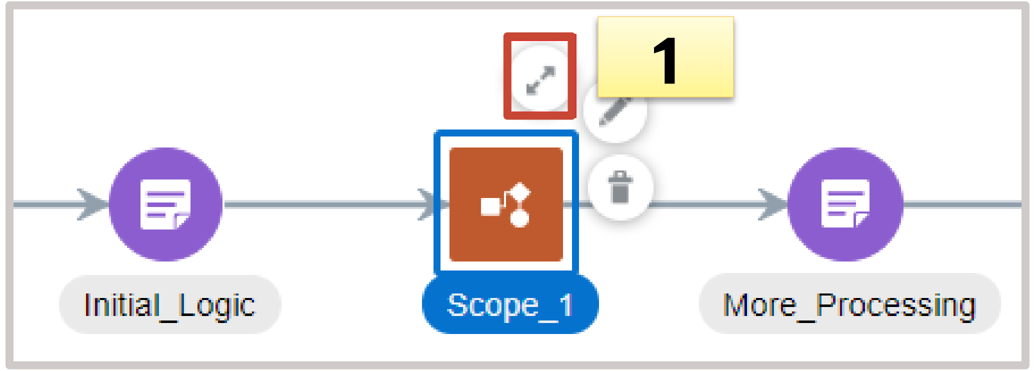
- **Throw New Fault** *(conditionally throw a custom fault to the error hospital)*
- **Stop** *(no fault)*
- **Fault Return** *(map data for a fault response to be sent to the client)*
- **Return** or **Callback** *(map data for a "success" response to the client)*

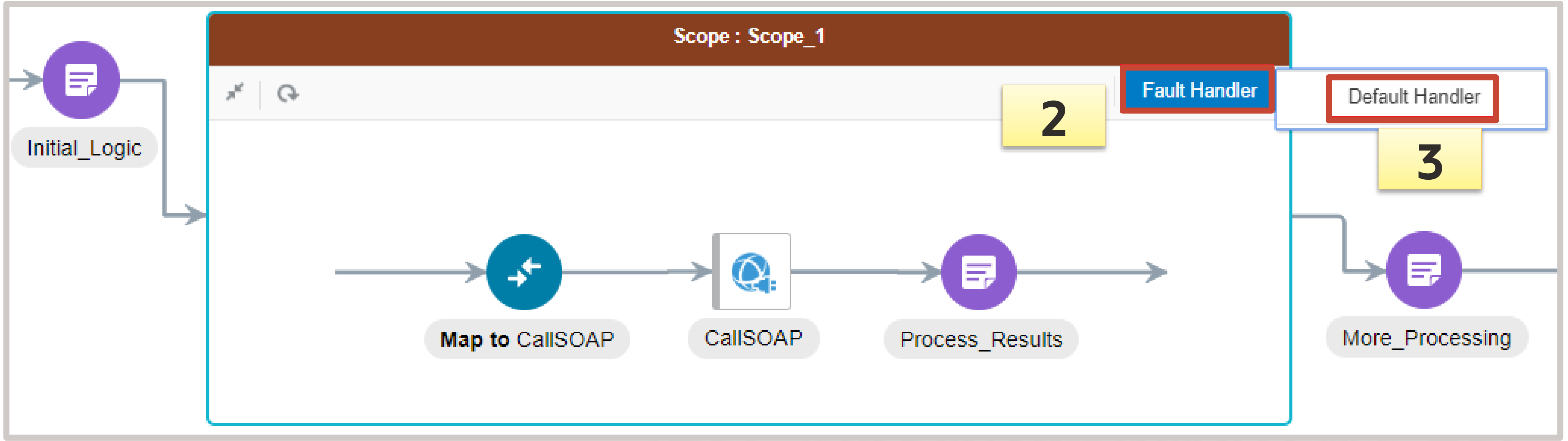


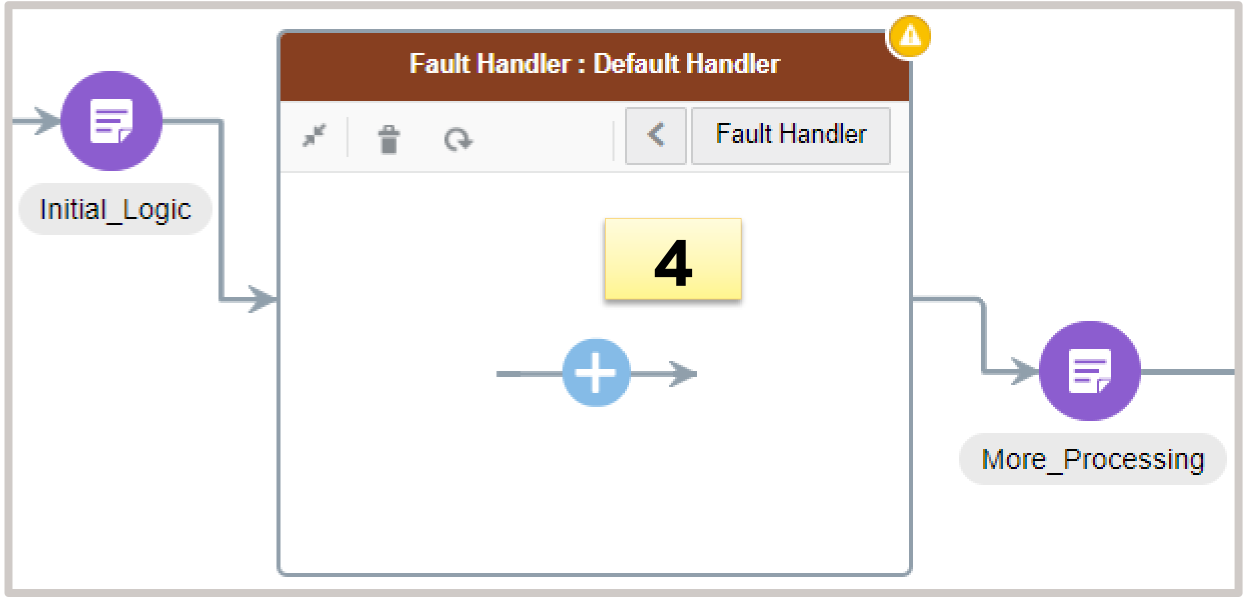
Scope Fault Handlers

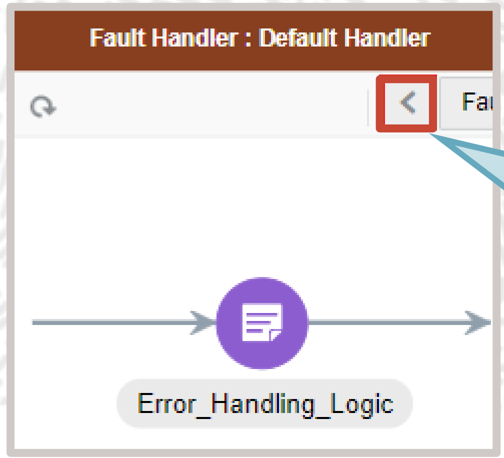
To implement error handling logic within a **Scope** container, open a scope fault handler as follows:

1. Expand the Scope *(if collapsed)*.
2. Click the **Fault Handler** button.
3. Select the handler to edit.
4. Add actions or invokes.





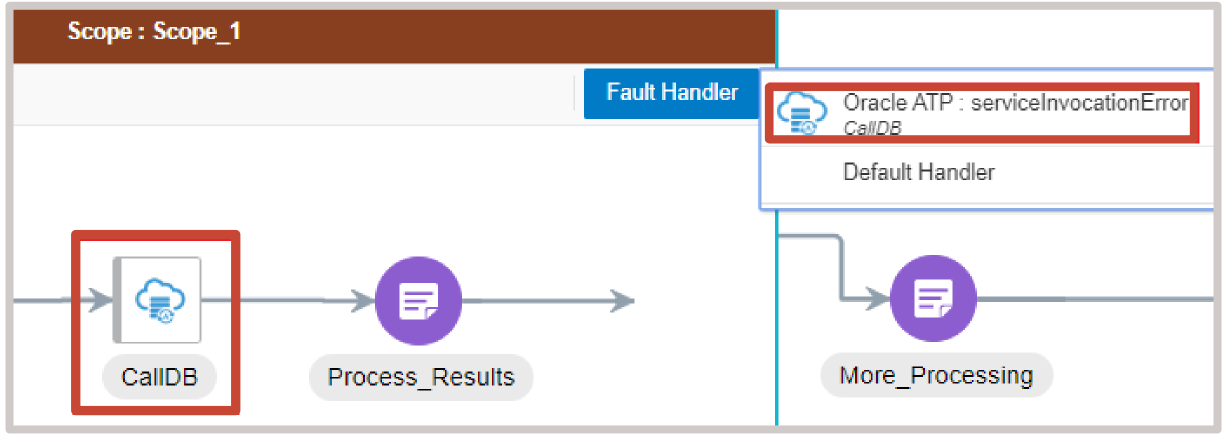


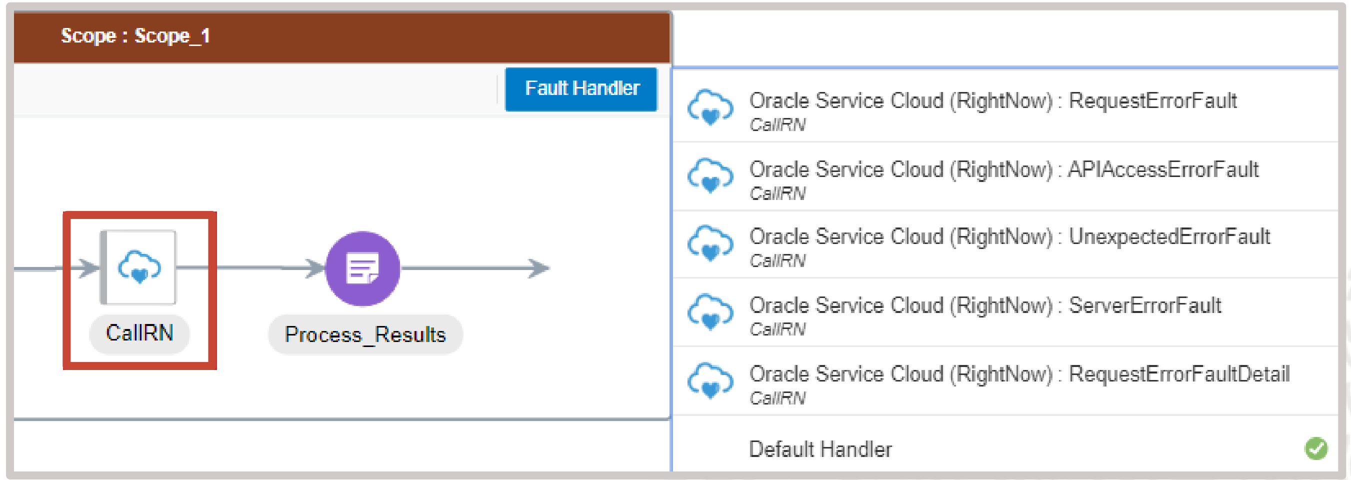


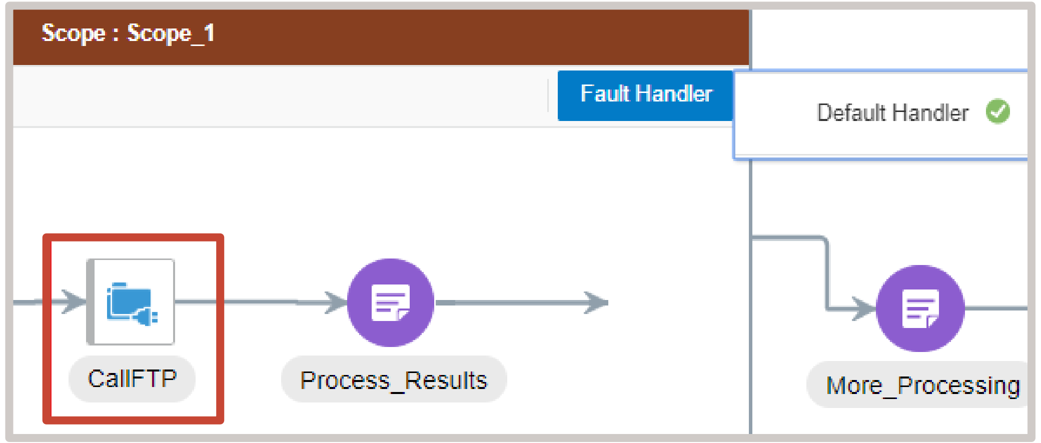
Click to return to the main Scope canvas

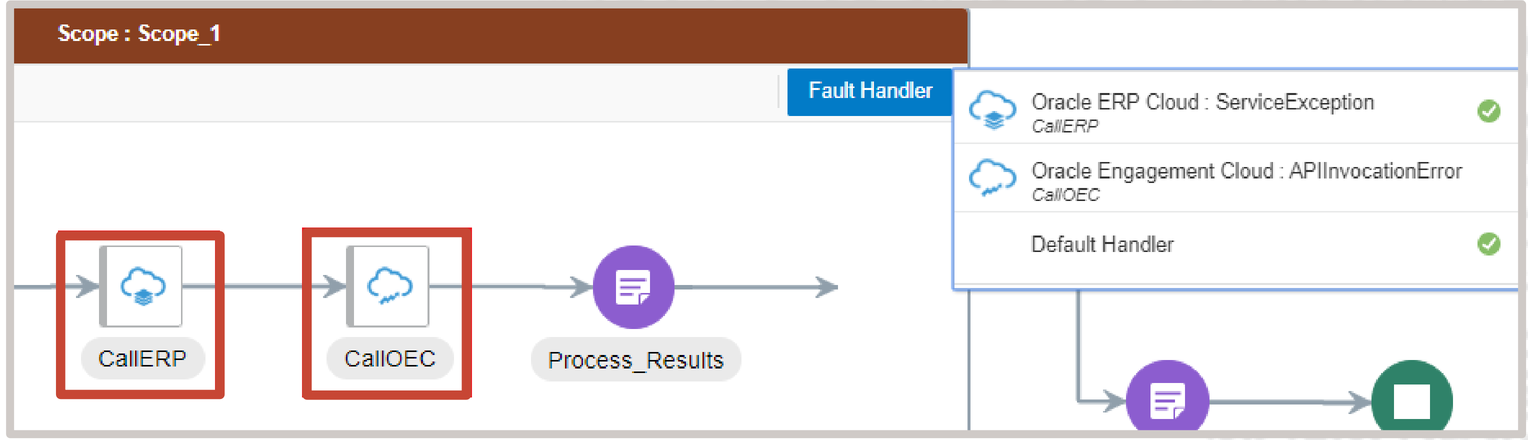
Additional Fault Handlers

- Along with the scope's default handler, additional handlers may be available based on the Invoke connection(s) configured within the Scope.
  - Many application adapter types provide a **serviceInvocationError** fault handler.
  - Typically, only the default handler is available for most technology adapter types.
- Implemented handlers are indicated with a green check mark.
- The Default Handler serves as a *"CatchAll"*.




Scope : Scope_1
Fault Handler
Oracle Service Cloud (RightNow) : RequestErrorFault
CallRN
Oracle Service Cloud (RightNow) : APIAccessErrorFault
CallRN
Oracle Service Cloud (RightNow) : UnexpectedErrorFault
CallRN
Oracle Service Cloud (RightNow) : ServerErrorFault
CallRN
Oracle Service Cloud (RightNow) : RequestErrorFaultDetail
CallRN
Default Handler ✓
CallRN
Process_Results

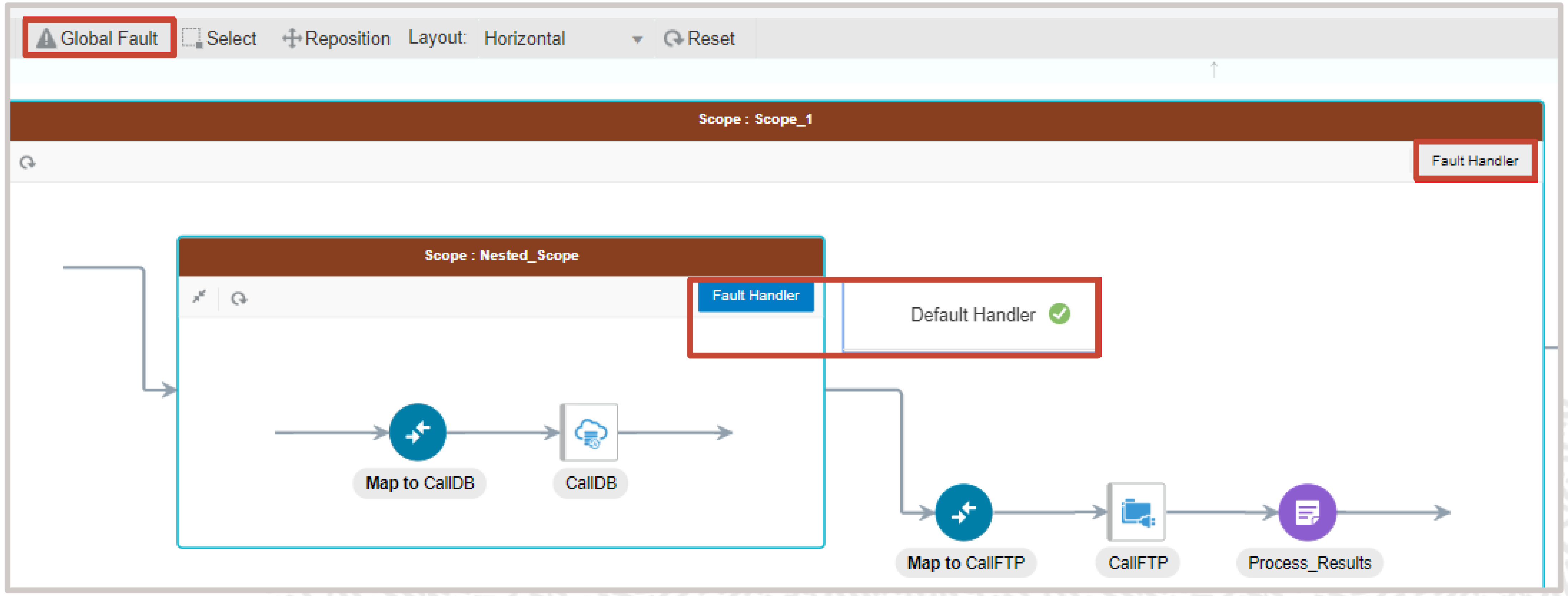




Catching Faults

- Faults are caught by the "closest" fault handler to where the fault occurred.
  - If there are no fault handlers configured for a **Scope** container, all faults will be caught by the next higher level fault handler.
  - Any faults occurring *within* a fault handler will be caught by the next higher level fault handler.

- Fault handler hierarchy:
  - Nested scope to its parent scope fault handler
  - Main flow scope to the Global fault handler
  - Global fault handler to the OIC Error Hospital

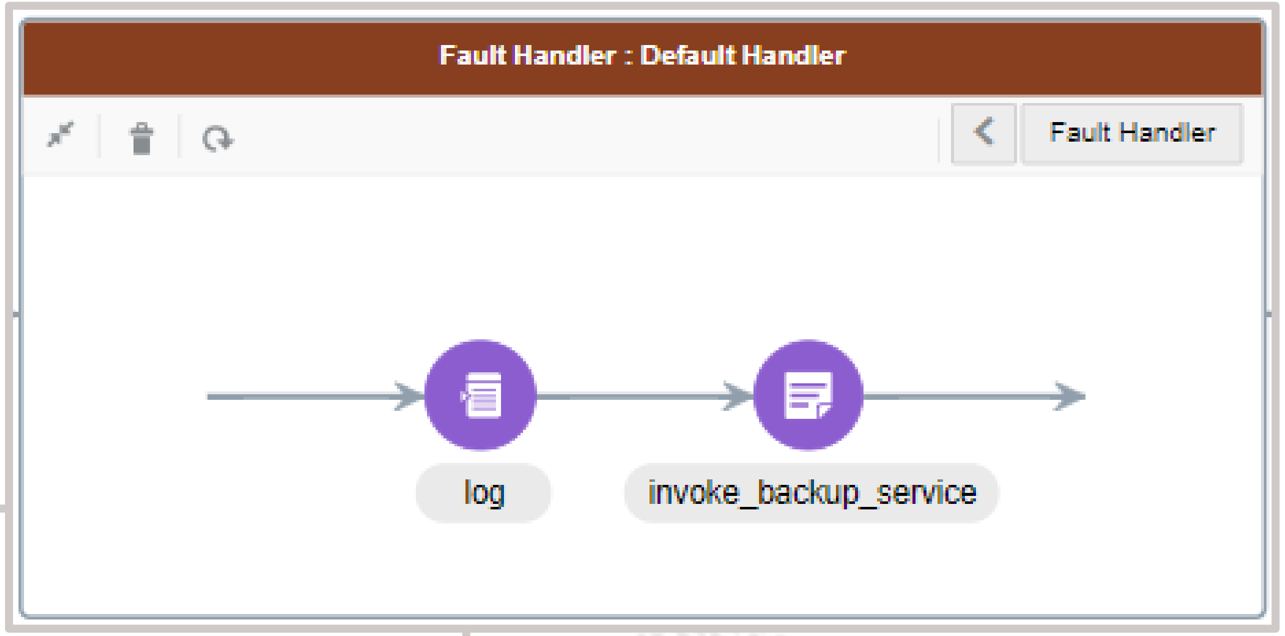


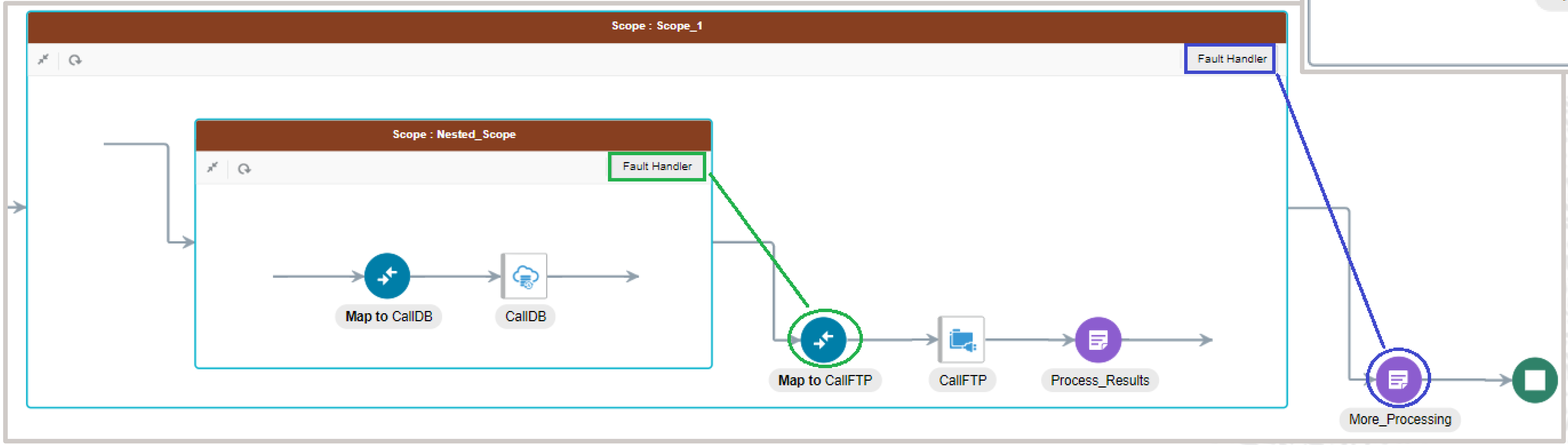
Fault Mitigation

Any fault that can be mitigated or is very minor need not necessarily terminate the integration flow. To continue processing after your fault handling or mitigation logic:

- Do not add an **End** action in the scope's fault handler
- The next action outside the scope will be executed.

Note: You must have an End action in the Global Fault Handler.

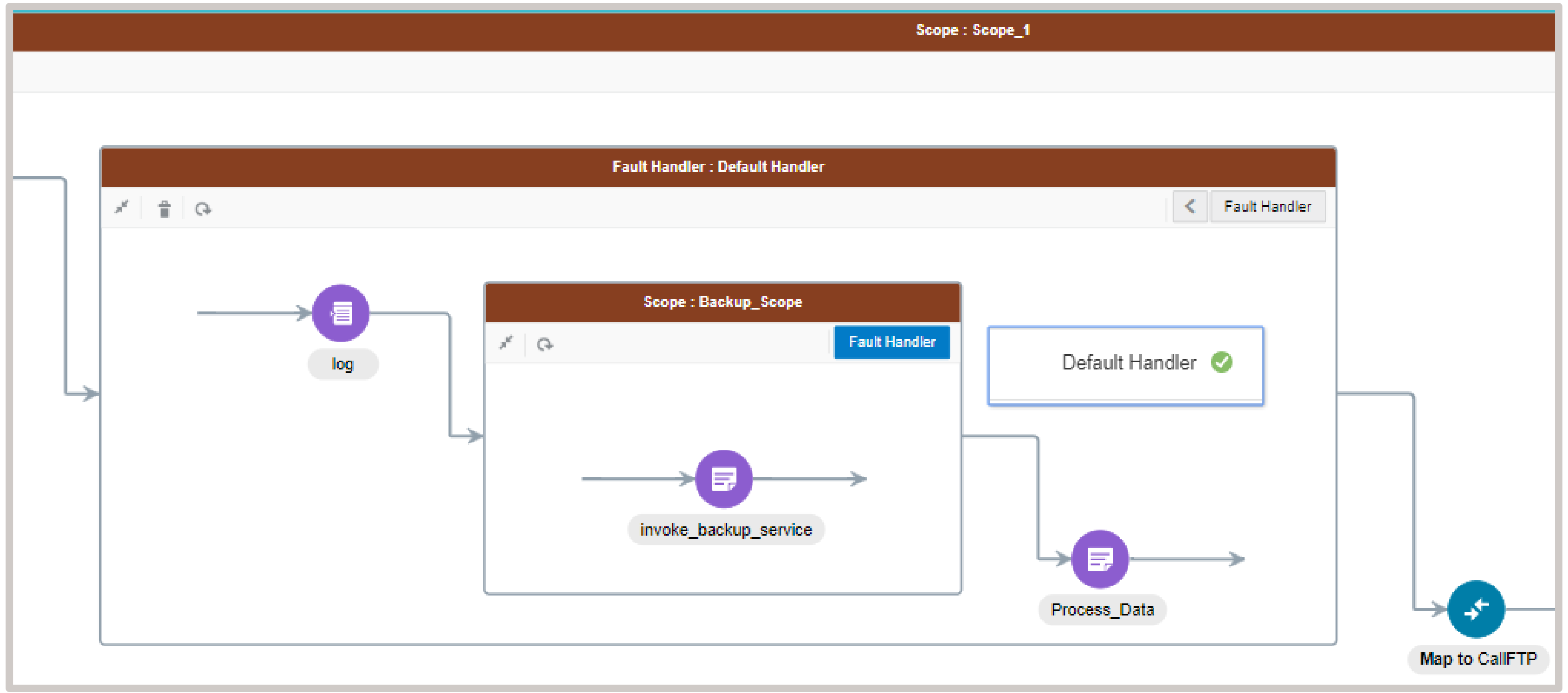


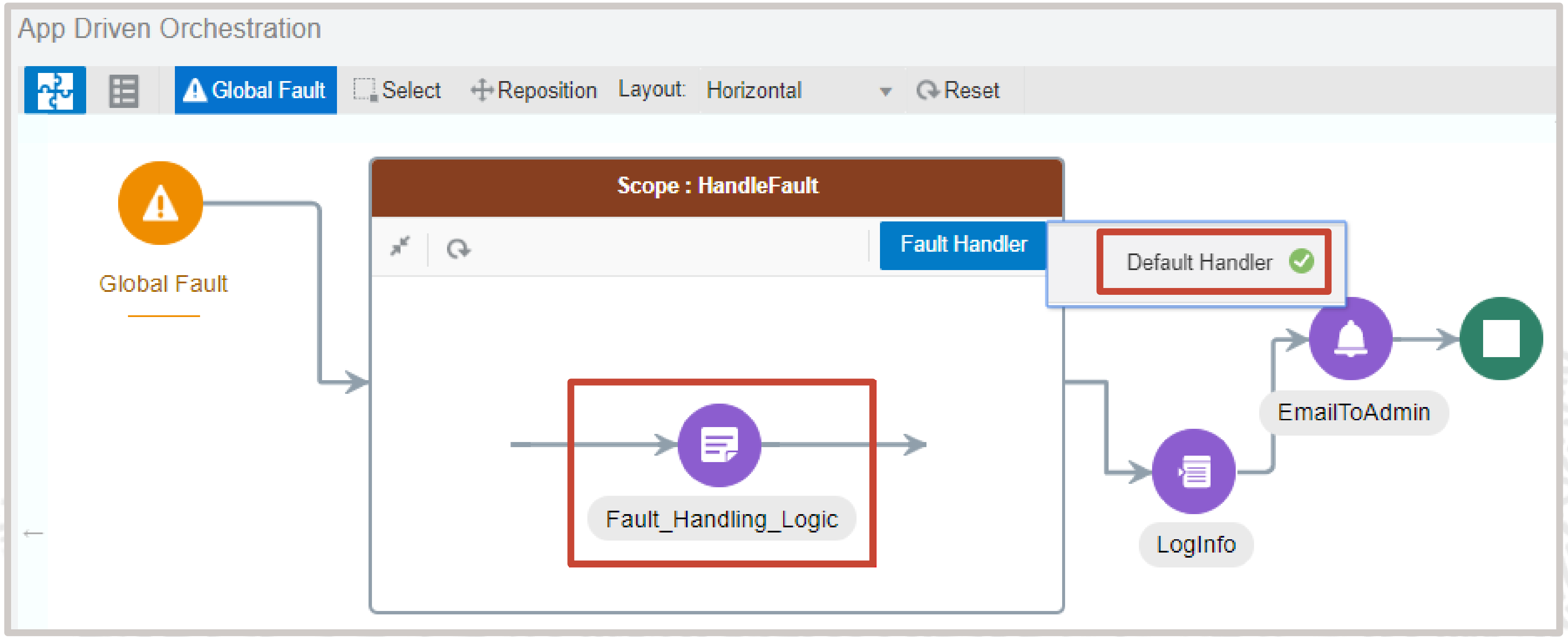


Adding Scopes Inside of Fault Handlers

If needed, you can handle more complex error handling logic by adding Scope containers inside of a Scope fault handler or even in the Global fault handler.

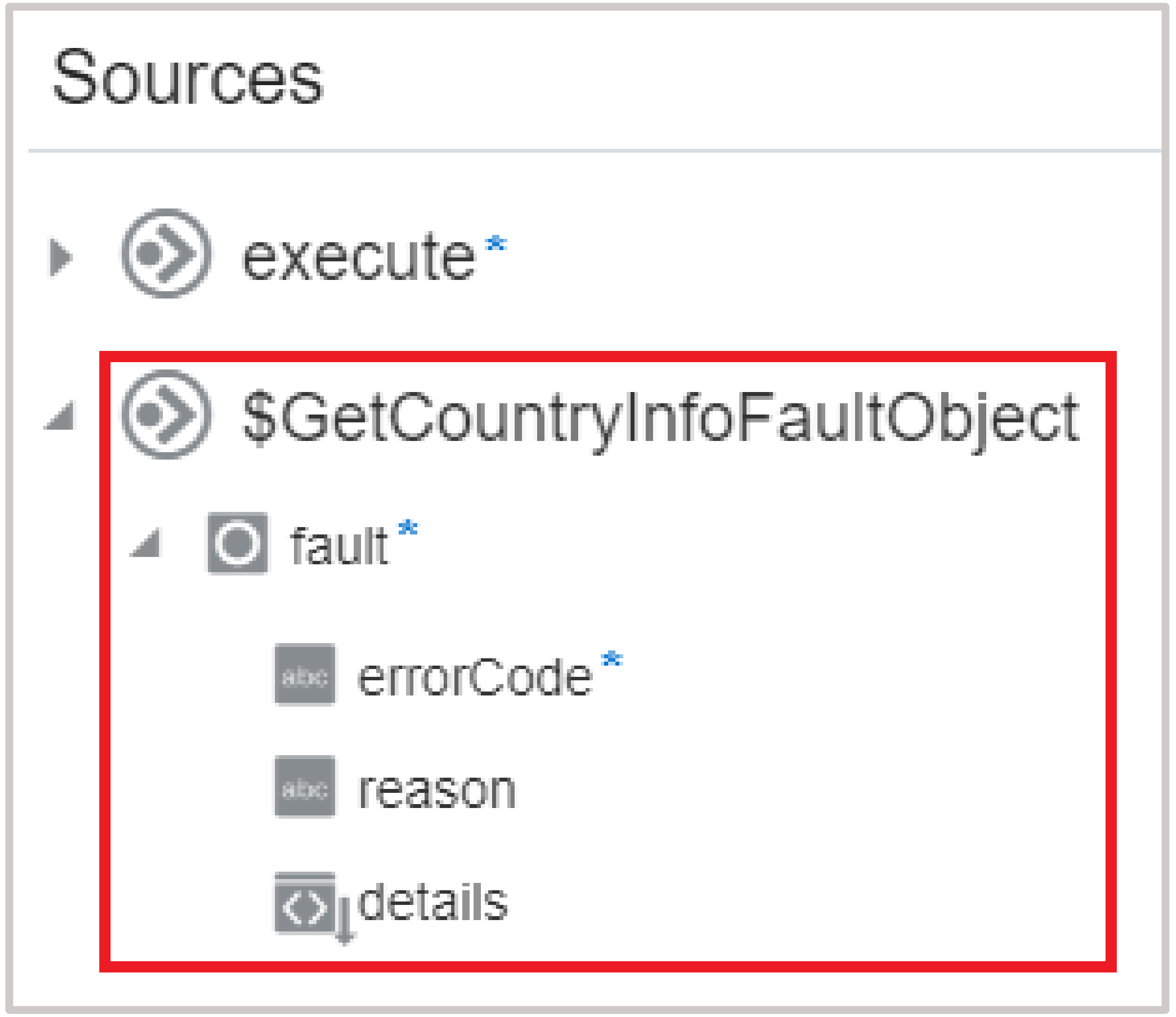
- Each new "nested" scope can then have its own fault handler logic as well.
- However, try to avoid overly complex nested fault handling logic.
  - A best practice is to design separate error-handling integration flows to be reused and invoked as needed.

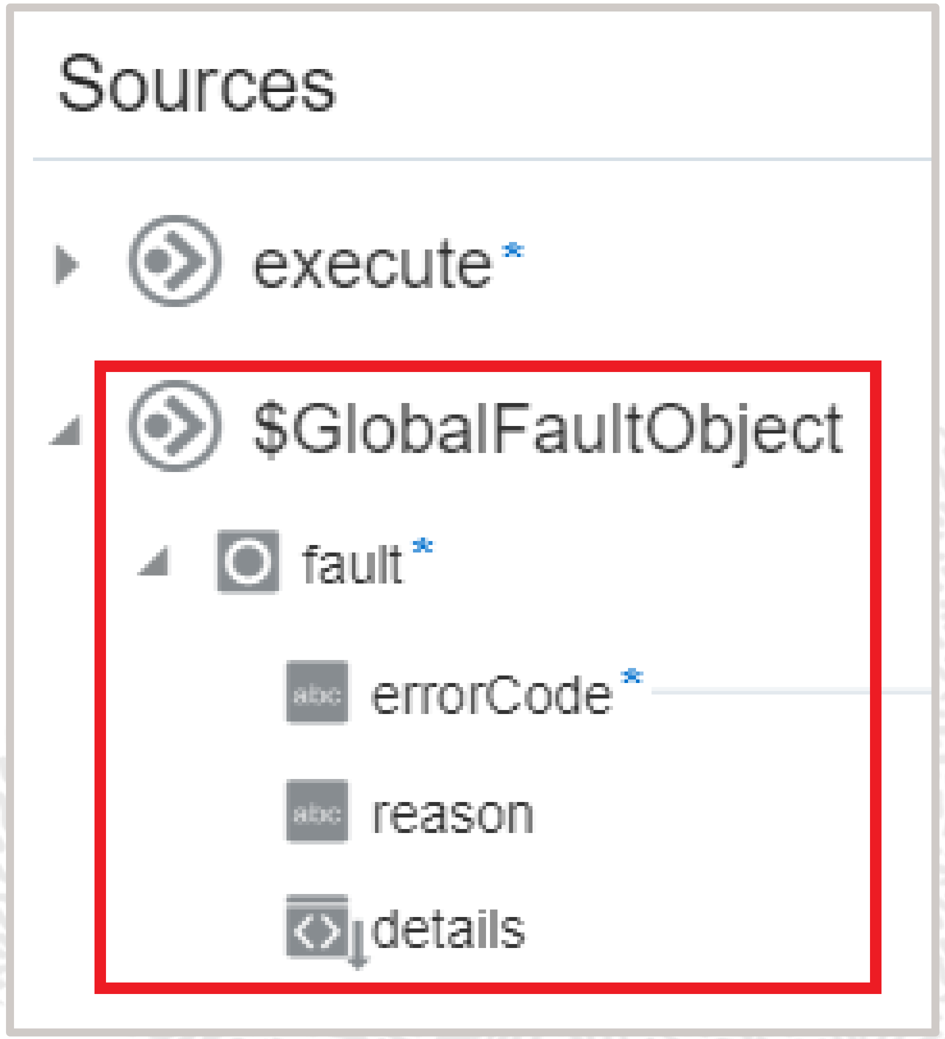




Accessing Fault Information

- You have access to the **Fault** object only while inside of a fault handler.
  - It is added as a **Source** data object in the Expression builder.
  - Data includes a required **errorCode** and optional **reason** and **details** field values.



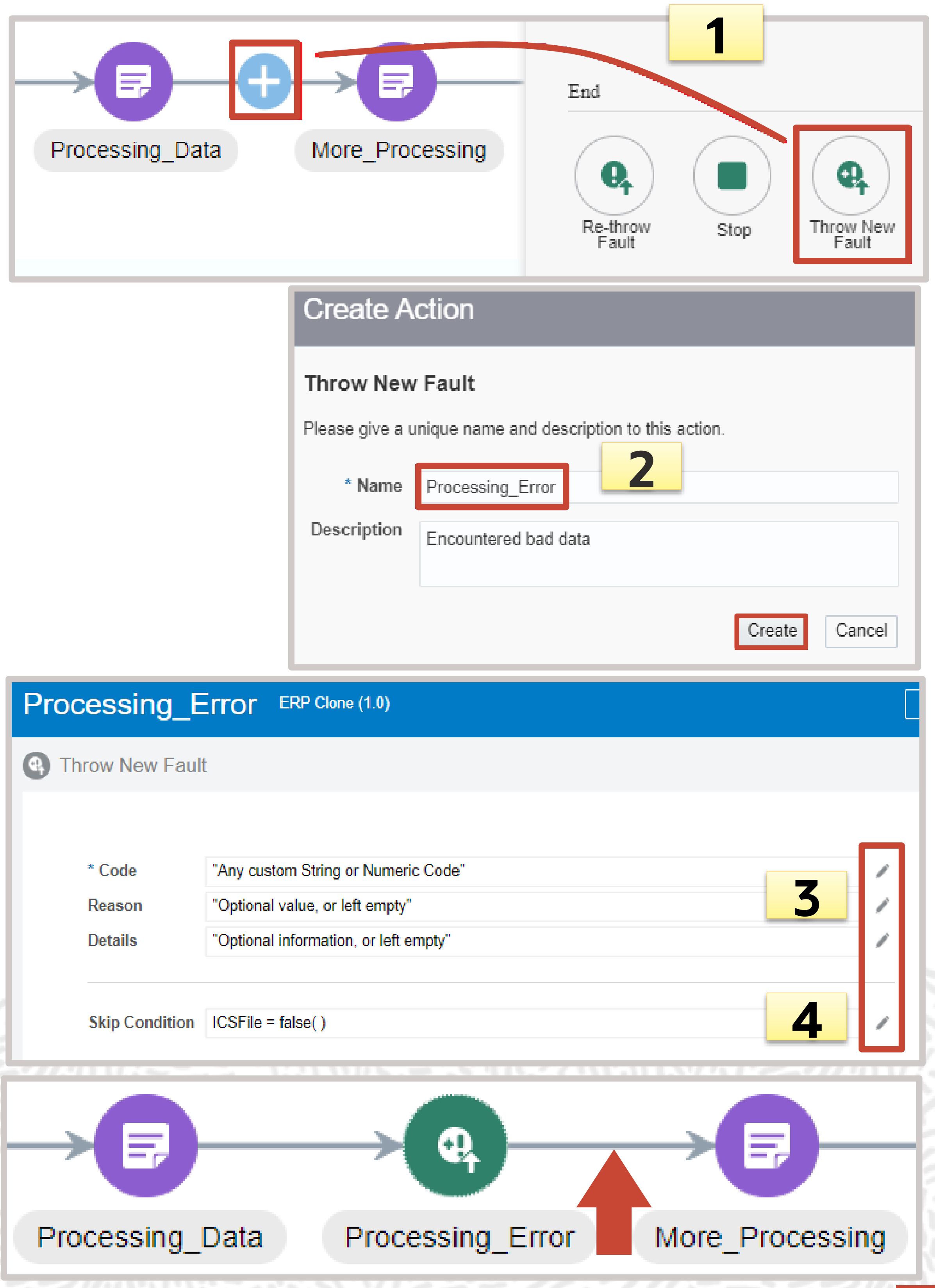


Leveraging the Throw New Fault Action

- The **Throw New Fault** action can be used anywhere within the integration flow:
  - Inside the Global fault handler
  - Inside a **Scope** fault handler
  - Within a **Scope** container, **Switch** branch, **For Each** or **While** loop
  - On the main flow path
- It can be used for defining a processing logic error to be used internally, or for sending a custom fault to external synchronous clients.

Configuring the Throw New Fault Action

1. Drag the **Throw New Fault** action to the desired location.
2. Create an action name and optional description.
3. Click each pencil icon to define:
   – **Code** (required)
   – **Reason**
   – **Details**
4. Optionally, configure the **Skip Condition**.
   – If the condition is false, the fault is thrown.
   – If the condition is true, processing continues on to the next action.



Create Action

Throw New Fault

Please give a unique name and description to this action.

* Name	Processing_Error
Description	Encountered bad data

Create Cancel

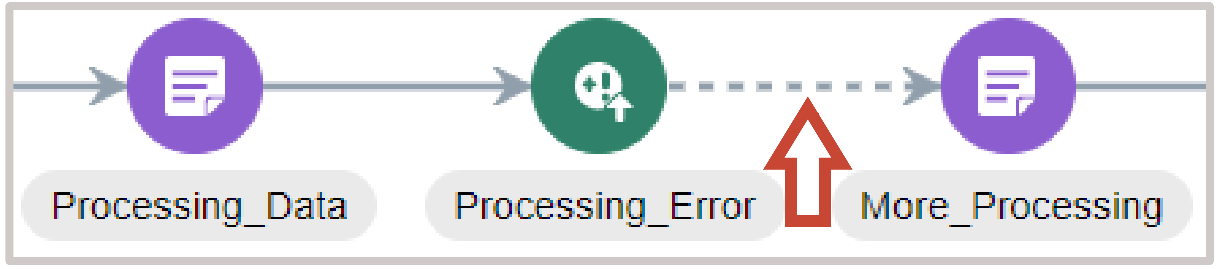
Processing_Error ERP Clone (1.0)

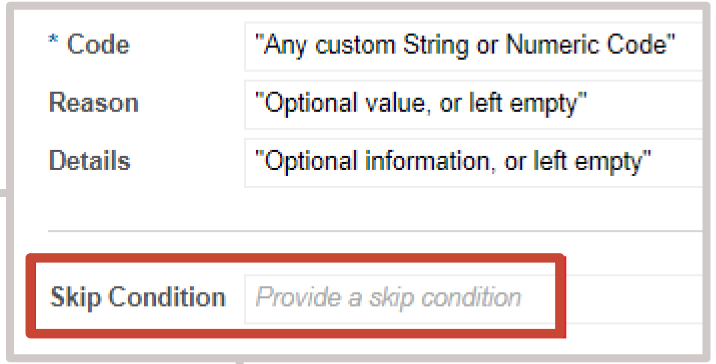
Throw New Fault

* Code	"Any custom String or Numeric Code"
Reason	"Optional value, or left empty"
Details	"Optional information, or left empty"
Skip Condition	ICSFile = false( )

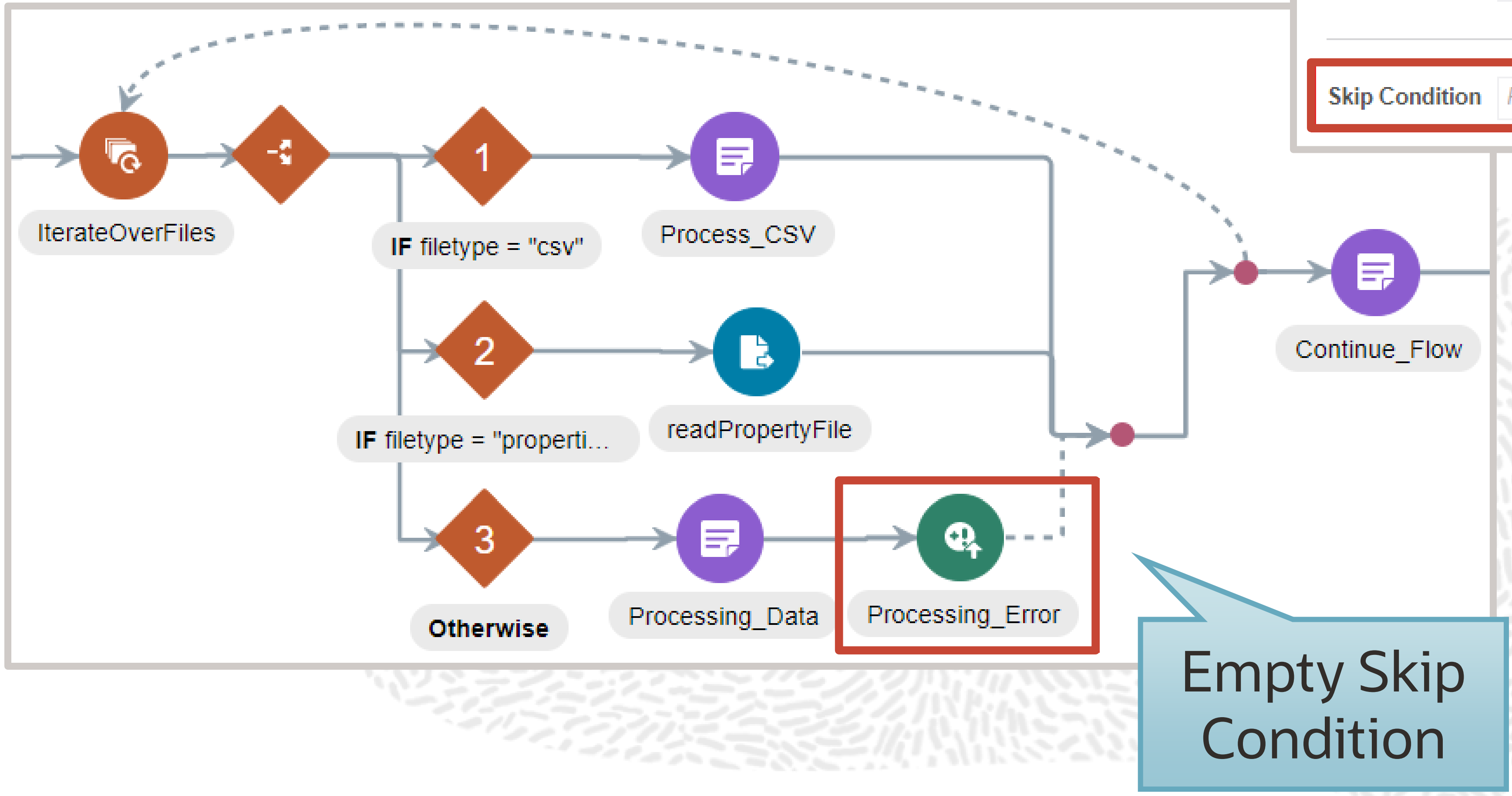
Skip Condition Considerations

- If the conditional expression for the skip condition is empty:
  - The fault will always be thrown
  - No additional actions in the flow will ever be executed
- Therefore, logically, you should always configure the skip condition unless it is the last action within a “final” execution thread.
  - **Scope** fault handler
  - **Switch** branch



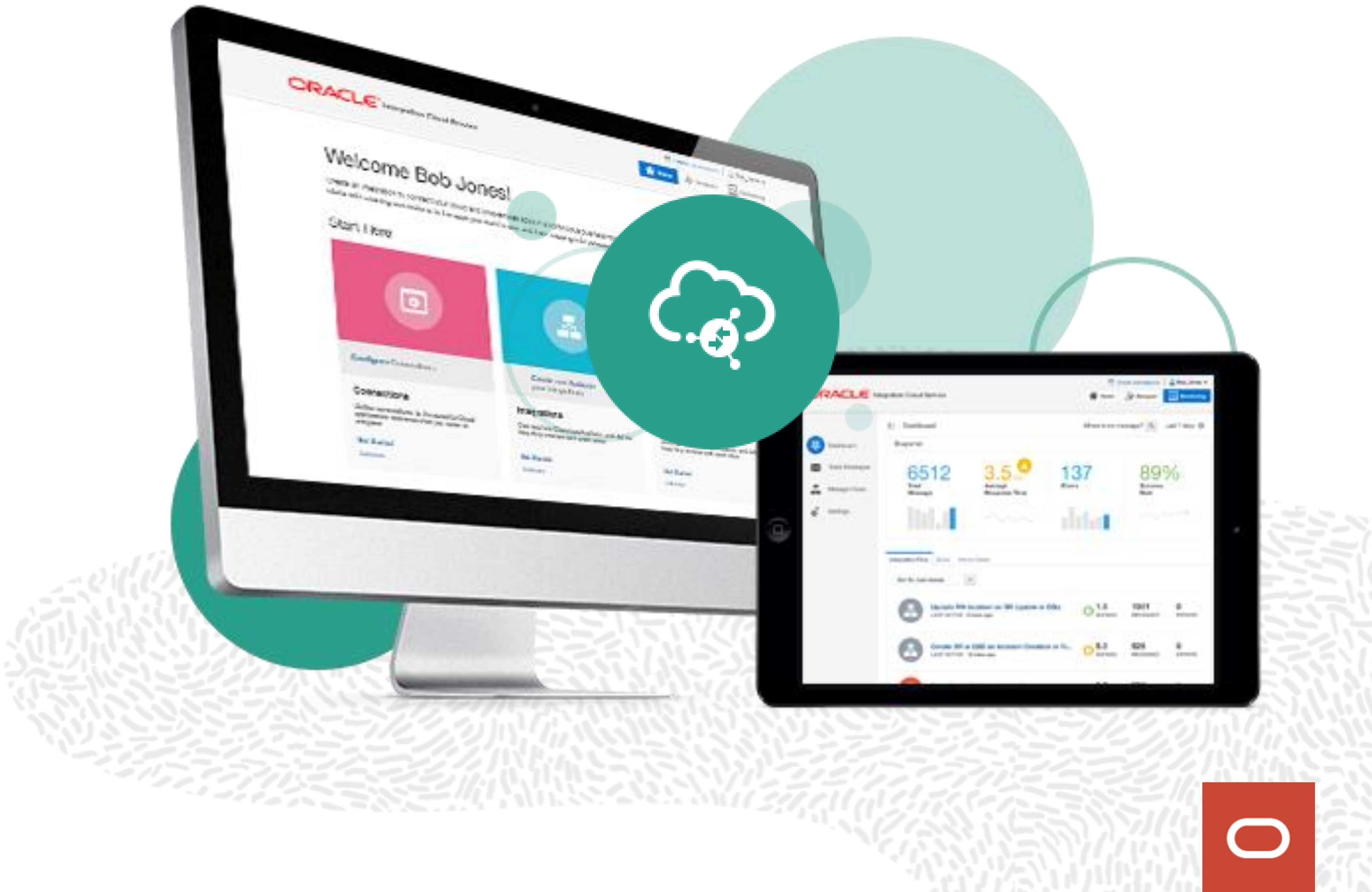


* Code	"Any custom String or Numeric Code"
Reason	"Optional value, or left empty"
Details	"Optional information, or left empty"
Skip Condition	Provide a skip condition



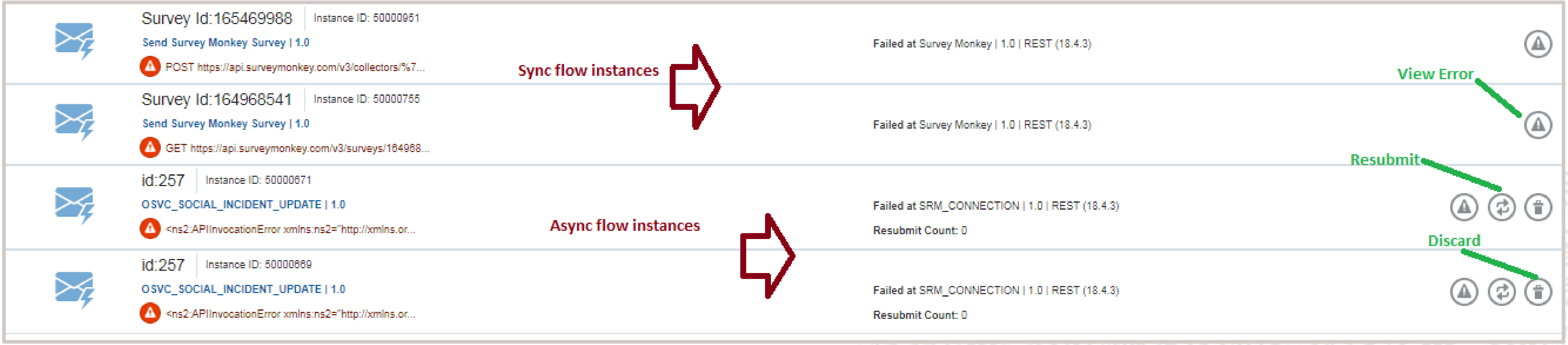
Agenda

- Understanding Scope Containers
- Using Fault Handlers
- Managing Failed Instances



Resubmitting Failed Instances

- It is the client’s responsibility to resubmit after receiving a fault from synchronous integration instances.
- However, asynchronous integration instances can be resubmitted in the event of failure.
  - When possible, design for idempotency (*repeatable with no side effects*).
  - Manual analysis may be required to determine if there are any external dependency issues that need to be resolved prior to resubmission.



Survey Id:165469988 | Instance ID: 50000951
Send Survey Monkey Survey | 1.0
POST https://api.surveymonkey.com/v3/collectors/%7...
Failed at Survey Monkey | 1.0 | REST (18.4.3)

Survey Id:164968541 | Instance ID: 50000755
Send Survey Monkey Survey | 1.0
GET https://api.surveymonkey.com/v3/surveys/164968...
Failed at Survey Monkey | 1.0 | REST (18.4.3)

id:257 | Instance ID: 50000671
OSVC_SOCIAL_INCIDENT_UPDATE | 1.0
<ns2:APIInvocationError xmlns:ns2="http://xmlns.or...
Failed at SRM_CONNECTION | 1.0 | REST (18.4.3)
Resubmit Count: 0

id:257 | Instance ID: 50000669
OSVC_SOCIAL_INCIDENT_UPDATE | 1.0
<ns2:APIInvocationError xmlns:ns2="http://xmlns.or...
Failed at SRM_CONNECTION | 1.0 | REST (18.4.3)
Resubmit Count: 0

Extended Error Handling

Example use cases requiring the need for additional components to maintain failed instance state and/or message persistence include:

- A requirement that upon a failure, the flow retries from the last point of failure instead of starting over from the start
- Scenarios in which the completed tasks are not repeatable or require expensive compensation to make them repeatable
- Resubmission of faulted instances are required beyond the OIC 3-day retention period.
- There is need to change the request payload prior to resubmitting the faulted instance.

We'll look at two example approaches for implementing extended fault handling logic…

Parking Lot Pattern

Use a database as a persistent “parking lot” to retain:

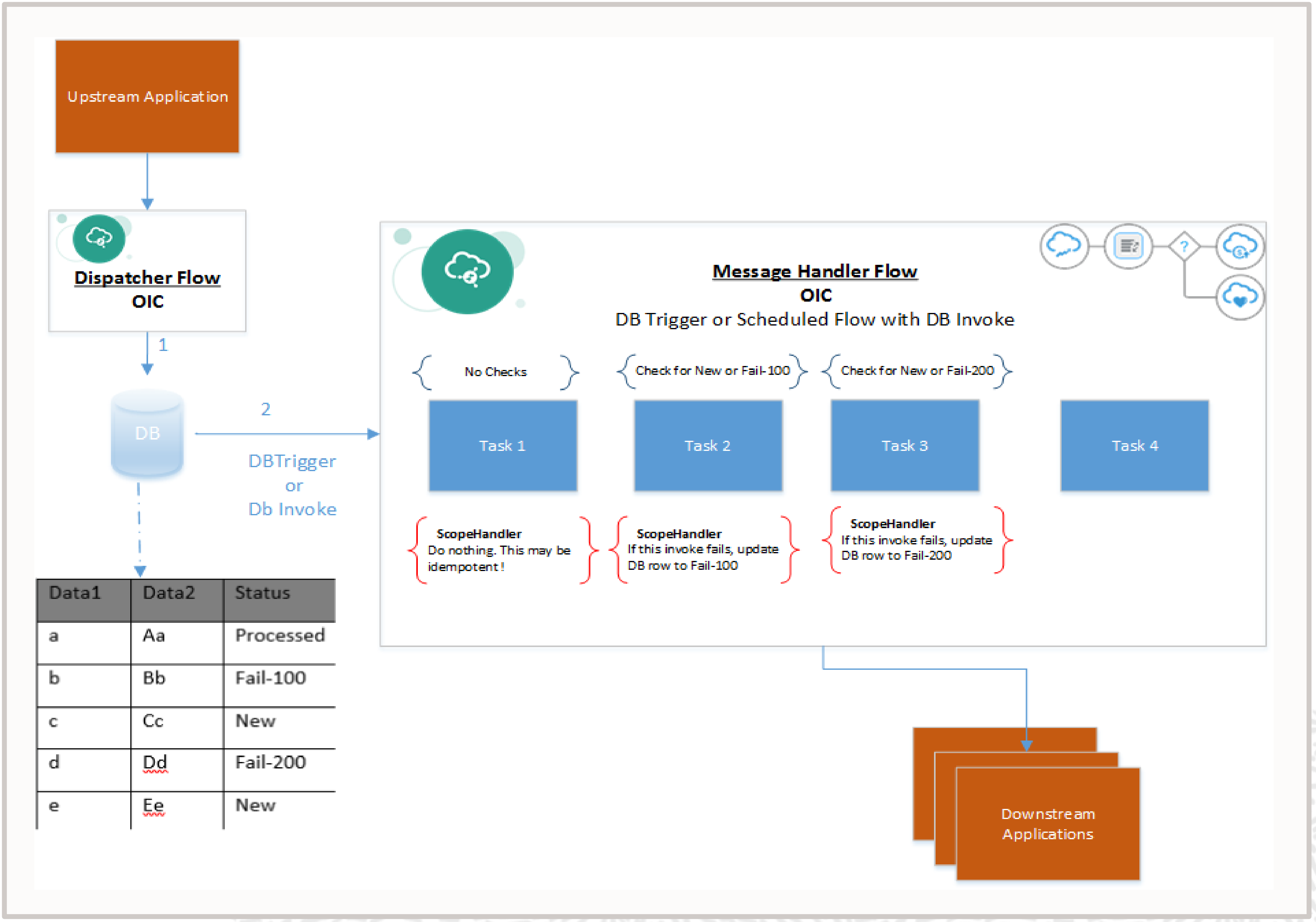
- New request messages
- Instance data and state at key milestones

Dispatcher Integration:

- Receives requests
- Creates a new database record

Message Handler Integration:

- Updates the instance data and state at each task milestone
- Changes status if failure occurs
- Processes each “record” until the full processing is completed



ScopeHandler
Do nothing. This may be idempotent!

ScopeHandler
If this invoke fails, update DB row to Fail-100

ScopeHandler
If this invoke fails, update DB row to Fail-200

Data1	Data2	Status
a	Aa	Processed
b	Bb	Fail-100
c	Cc	New
d	Dd	Fail-200
e	Ee	New

File-based Integrations

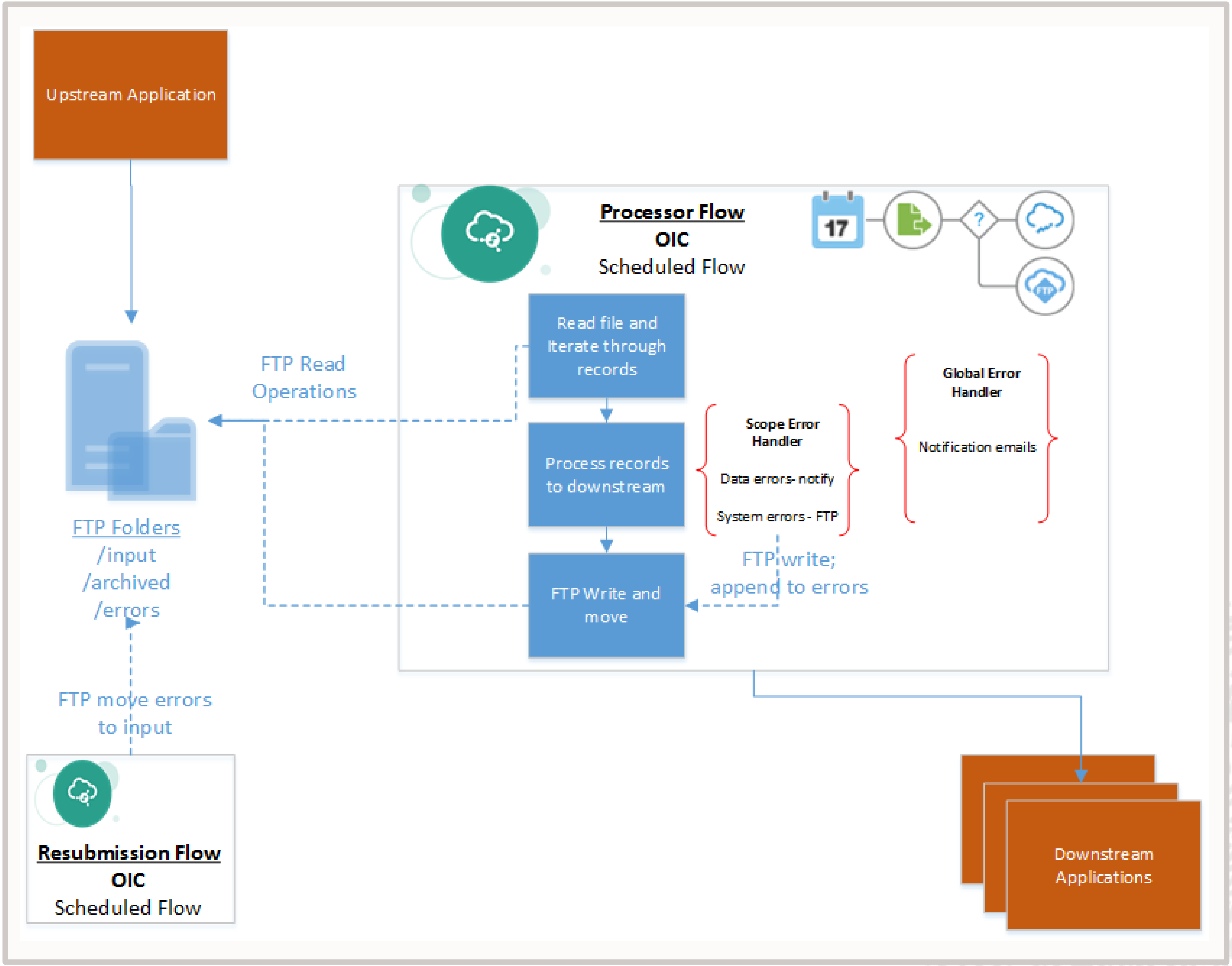
A simple strategy for reprocessing records from batch files involves two separate Scheduled Orchestration style integration flows.

Processor Integration:

- Retrieves files and processes records
  - Failed records *(due to system failures)* are appended to an errors file.
  - Error file is sent to FTP server.
  - Successful files are archived on FTP.

Resubmission Integration:

- Moves error files from error folder to the input location – *(to be reprocessed by the* ***Processor*** *integration)*



Fault Handling Best Practices

- Organize related invokes and actions within separate Scope containers. Handle or mitigate faults within the scope's fault handlers.
- Define generic fault handling logic *(such as notifications or logging)* within the Global fault handler. Throw *(or rethrow)* faults from Scope fault handlers as needed.
- Create one or more "error-handling" integration flows that can be reused and invoked from other integration's fault handlers.
- Consider creating an OIC Process Application process for fault handling logic that may require manual intervention or human-centric workflow tasks.
- Plan for monitoring failed integration instances for further analysis, review, and/or resubmission.
- When needed, implement extended error handling strategy design patterns.

Summary

In this lesson, you should have learned how to:

- Leverage Scope and nested Scope containers
- Describe the role and behavior of the OIC error hospital
- Implement error handling logic in the Global fault handler
- Design error handling strategies using Scope fault handlers
- Access fault information within fault handlers
- Explain strategies for extended error handling use cases

Practice 12-1: Implementing Scope Fault Handler Logic

This practice includes:

- PART 1: Create the Integration and Define the Trigger Interface
- PART 2: Add a Scope and Configure Invoke Logic
- PART 3: Configure the Scope's Default Fault Handler
- PART 4: Configure the Global Fault Handler
- PART 5: Activate and Test the Integration Flow