Temporal 102

Java™

20240906a

Temporal 102

Your Instructor

Mason Egger

Austin, TX, USA

Current: **Sr. Developer Advocate @ Temporal**

Past: Sr. Technical Curriculum Developer at Temporal (Wrote these courses)

Sr. Developer Advocate at DigitalOcean

Sr. Site Reliability Engineer at Vrbo (Expedia Group)

Software Engineer at Forcepoint (Raytheon)

Logistics

- **Introductions**
- **Schedule**
- **Facilities**
- **WiFi**
- **Asking questions and providing feedback**
- **Course conventions: “Activity” vs “activity”**
- **Prerequisite: Did *everyone* already complete Temporal 101?**

Network: Replay2025
Password: Durable!

We welcome your feedback

`t.mp/replay25ws`

During this workshop, you will

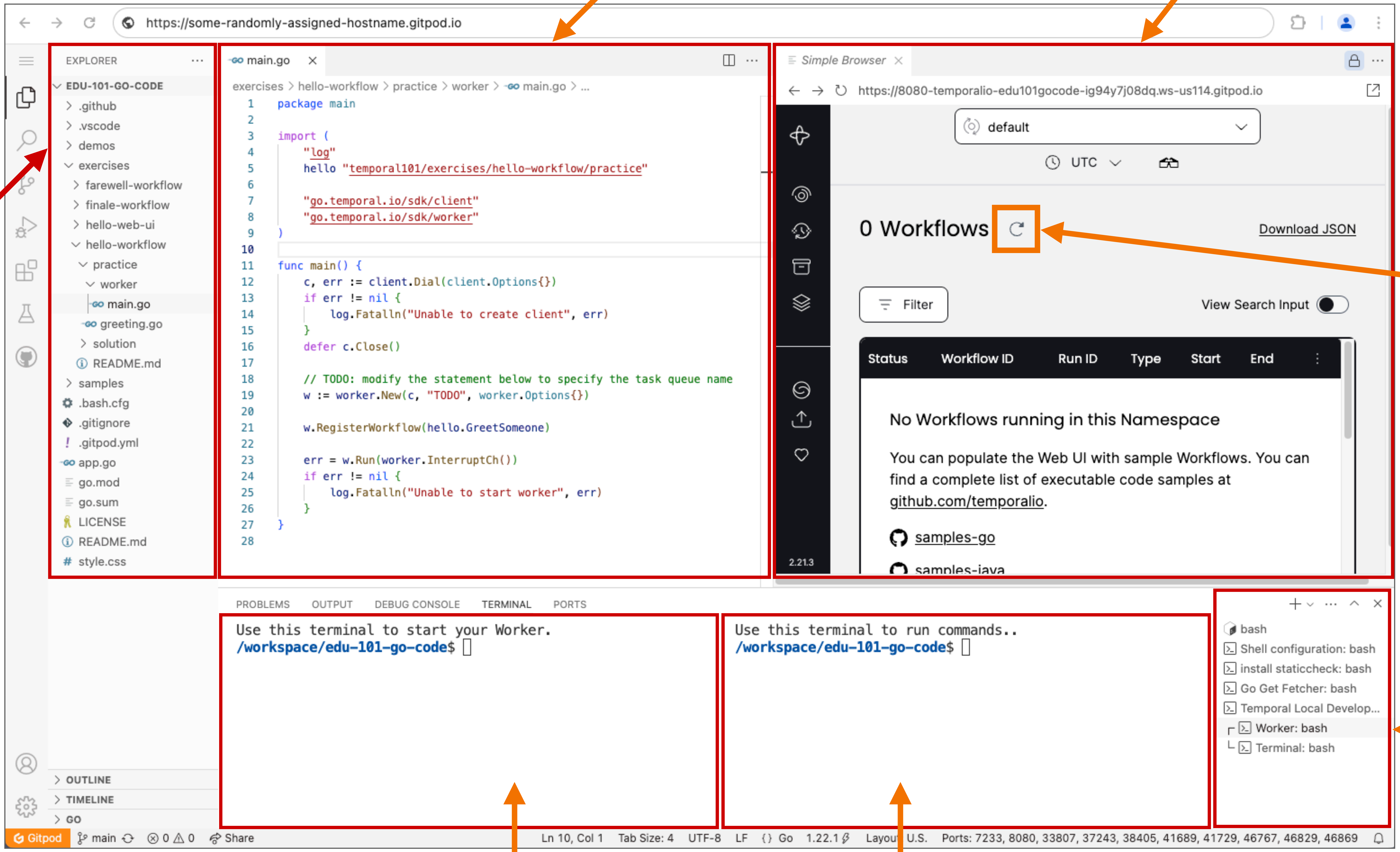
- Evaluate what a **production deployment** of Temporal looks like
- Use **Timers** to introduce delays in Workflow Execution
- Capture runtime information through **logging** in Workflow and Activity code
- Interpret **Event History** and debug problems with Workflow Execution
- Recognize **how Workflow code maps to Commands and Events** during Workflow Execution
- Differentiate **completion, failure, cancelation, and termination** of Workflow Executions
- Consider **why Temporal requires determinism** for Workflow code
- Observe **how Temporal uses History Replay** to achieve durable execution of Workflows
- Leverage the SDK's **testing support** to validate application behavior

Exercise Environment

- **We provide a development environment for you in this workshop**
  - It uses the GitPod service to deploy a private cluster, plus a code editor and terminal
  - You access it through your browser (may require you to log in to GitHub)
  - Your instructor will now demonstrate how to access and use it

https://t.mp/102-java-exercise-env

GitPod Overview



```go
package main

import (
	"log"
	hello "temporal101/exercises/hello-workflow/practice"

	"go.temporal.io/sdk/client"
	"go.temporal.io/sdk/worker"
)

func main() {
	c, err := client.Dial(client.Options{})
	if err != nil {
		log.Fatalln("Unable to create client", err)
	}
	defer c.Close()

	// TODO: modify the statement below to specify the task queue name
	w := worker.New(c, "TODO", worker.Options{})

	w.RegisterWorkflow(hello.GreetSomeone)

	err = w.Run(worker.InterruptCh())
	if err != nil {
		log.Fatalln("Unable to start worker", err)
	}
}
```

No Workflows running in this Namespace

You can populate the Web UI with sample Workflows. You can find a complete list of executable code samples at github.com/temporalio.

samples-go

samples-java

```
Use this terminal to start your Worker.
/workspace/edu-101-go-code$
```

```
Use this terminal to run commands..
/workspace/edu-101-go-code$
```

Temporal 102

Temporal: A Durable Execution System

- **What is a durable execution system?**
    - Ensures that your application runs reliably despite adverse conditions
    - Automatically maintains application state and recovers from failure
    - Improves developer productivity by making applications easier to develop, scale, and support

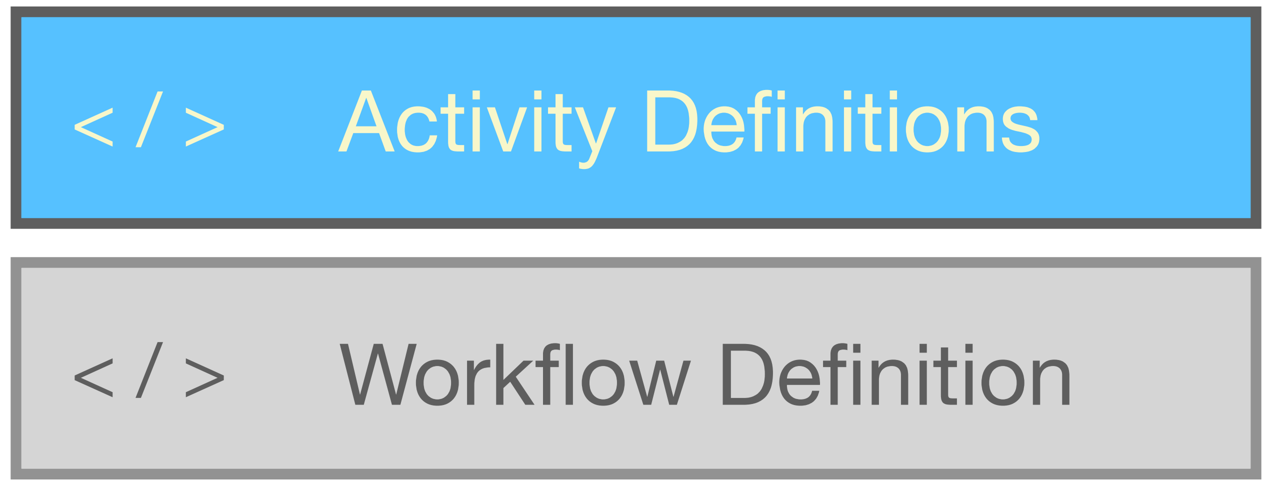
Temporal Workflows

- **Workflows are the core abstraction in Temporal**
    - It represents the sequence of steps used to carry out your business logic
    - They are durable: Temporal automatically recreates state if execution ends unexpectedly
    - In the Java SDK, a Temporal Workflow is defined as an Interface and its Implementation
    - Temporal requires that Workflows are *deterministic*

< / > Workflow Definition

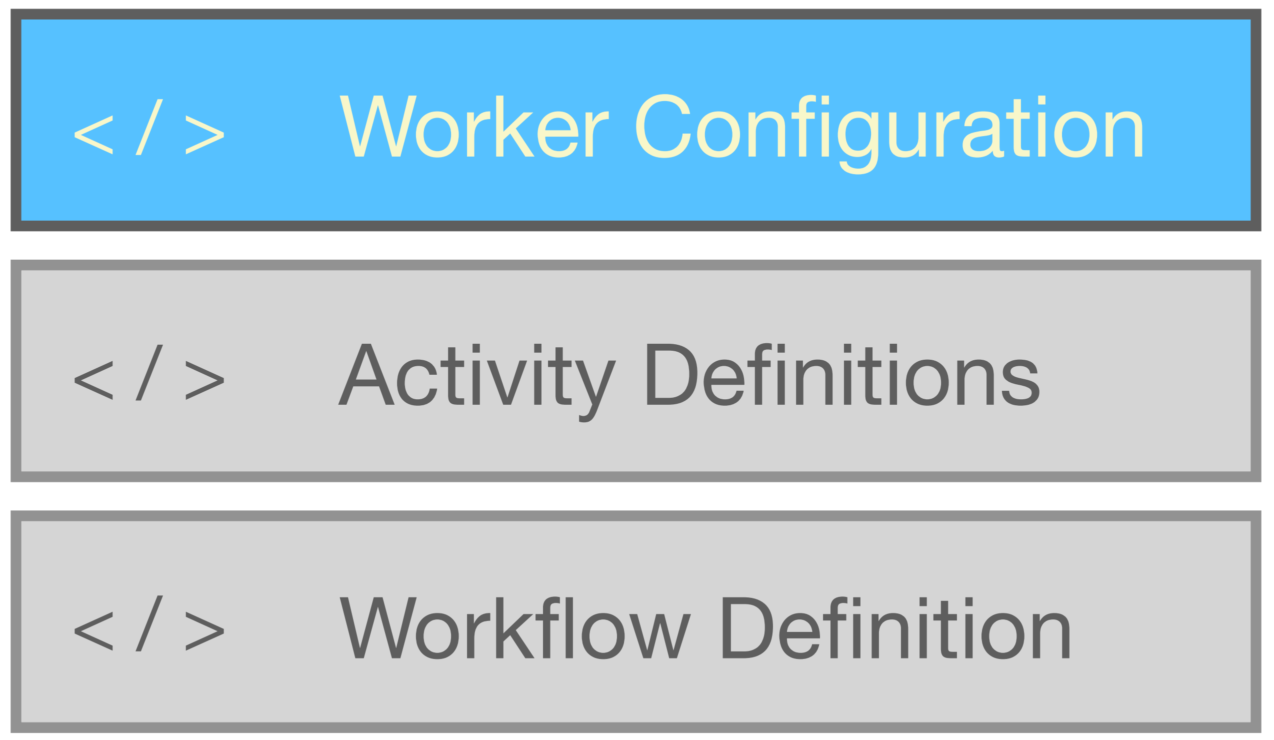
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Java SDK, Activities are defined as an Interface and its Implementation
  - Activities should be idempotent
    - A failed Activity may be retried, which means its code will be executed again
    - Protect against scenarios where re-running an Activity results in duplicate records or other undesirable side-effects.



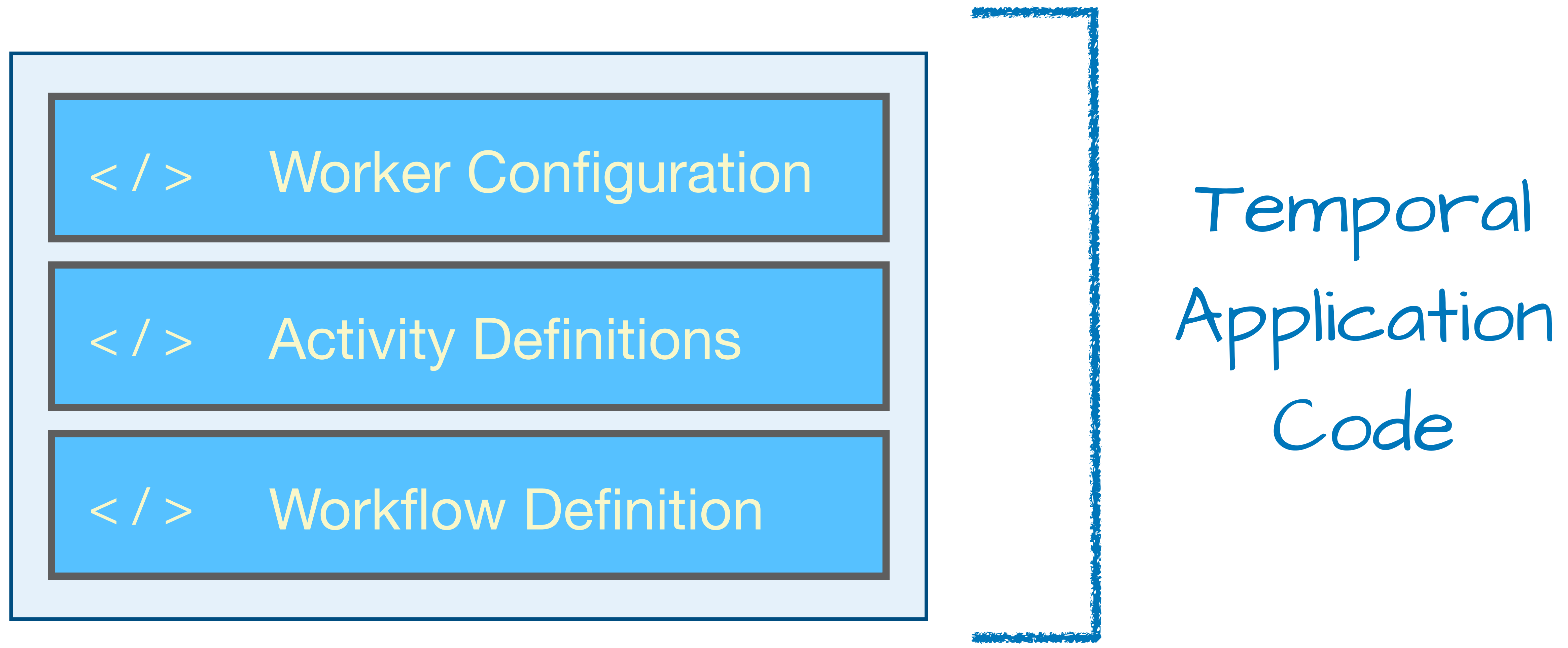
Temporal Workers

- **Workers are responsible for executing Workflow and Activity Definitions**
  - They poll a Task Queue maintained by the Temporal Service
- **The Worker implementation is provided by the Temporal SDK**
  - Your application will configure and start the Workers

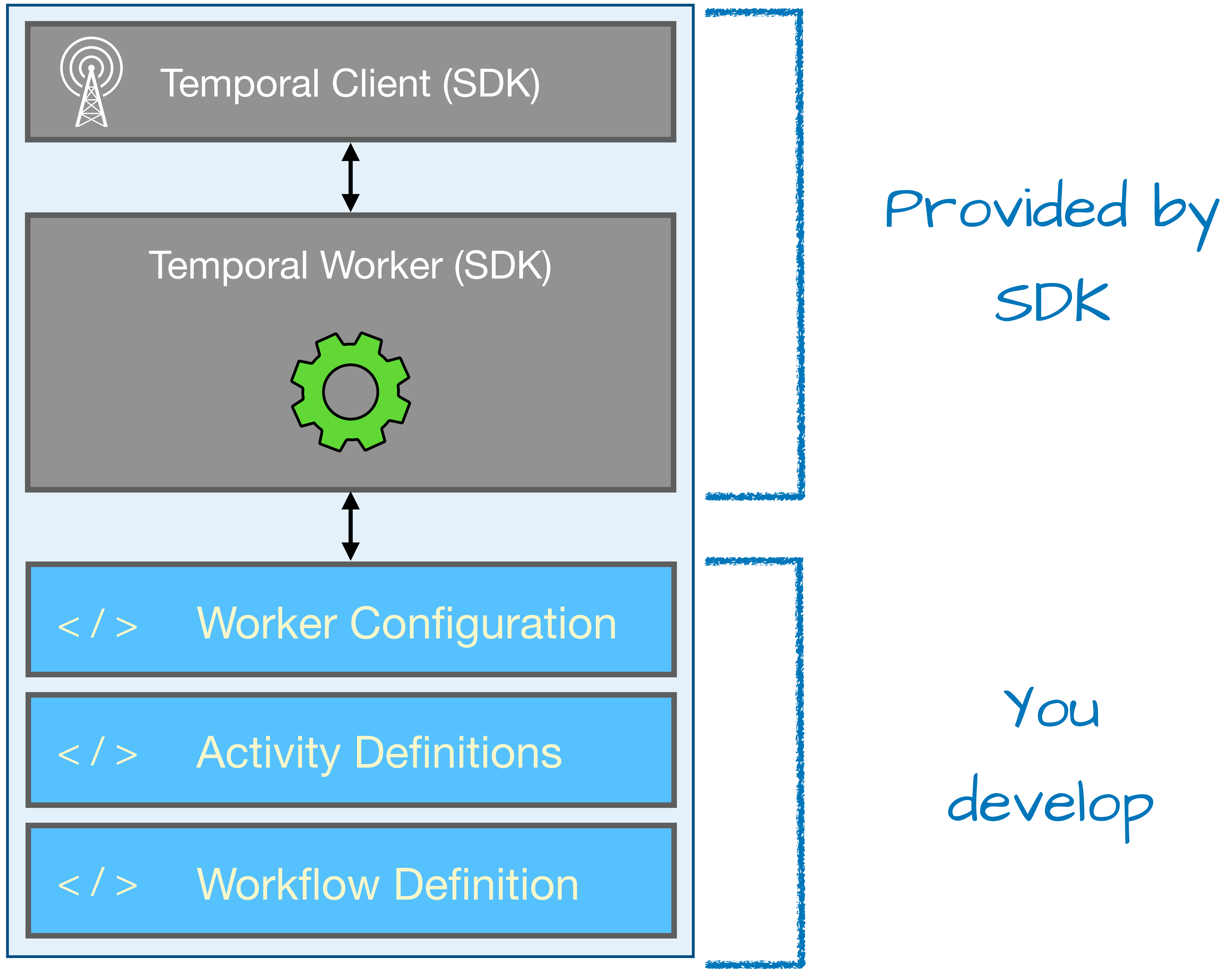


< / > Worker Configuration
< / > Activity Definitions
< / > Workflow Definition

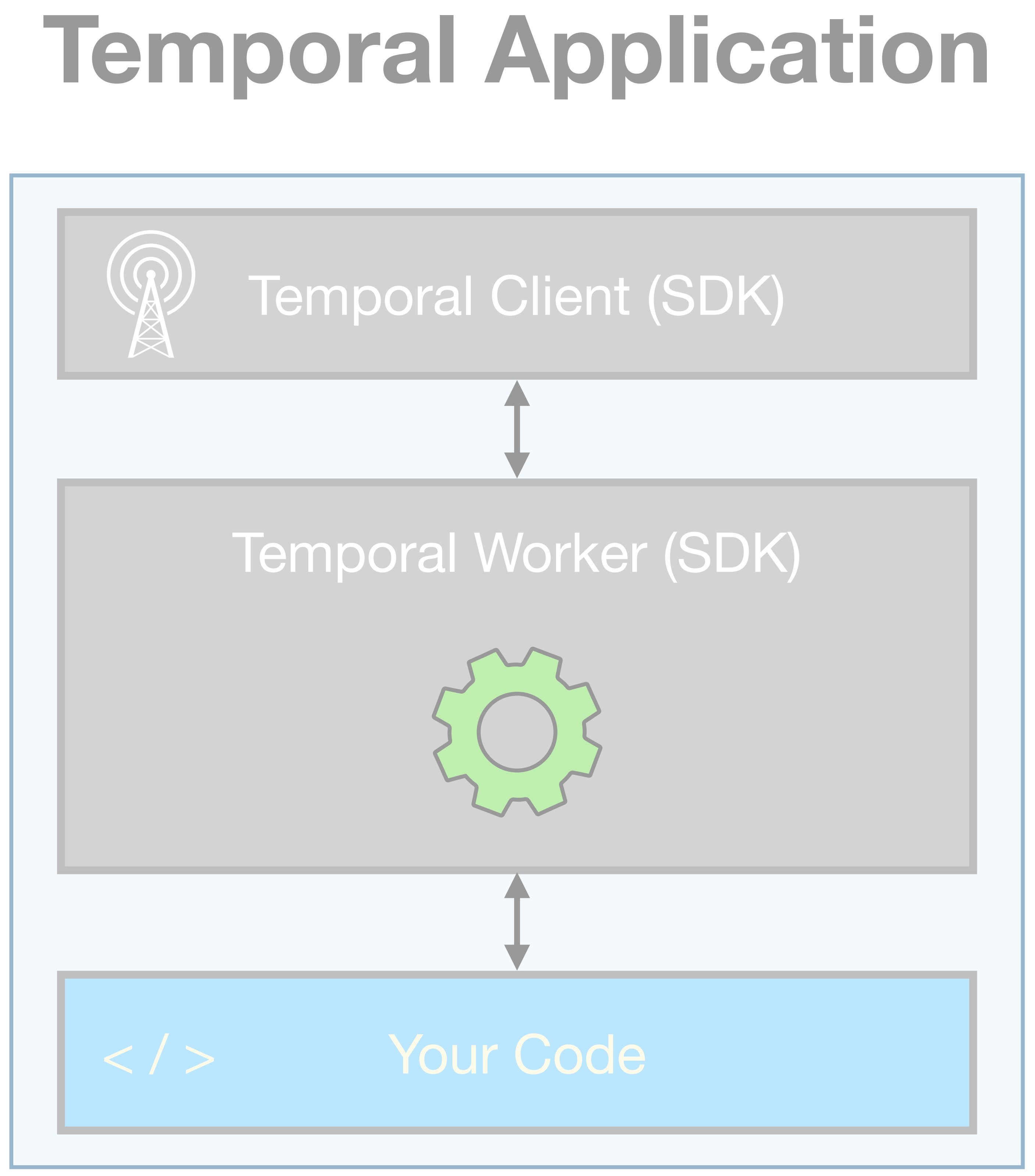
Code You Develop

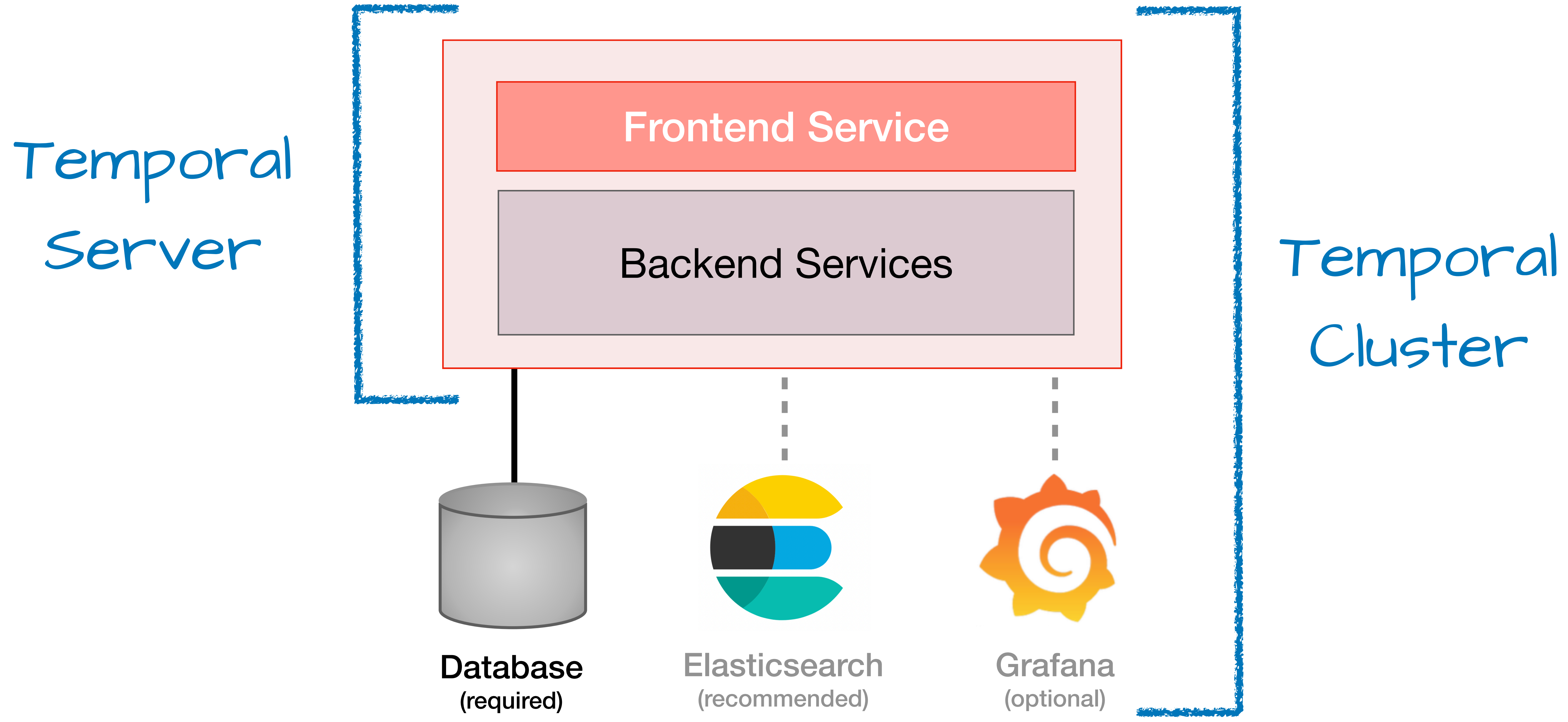


A Complete Temporal Application

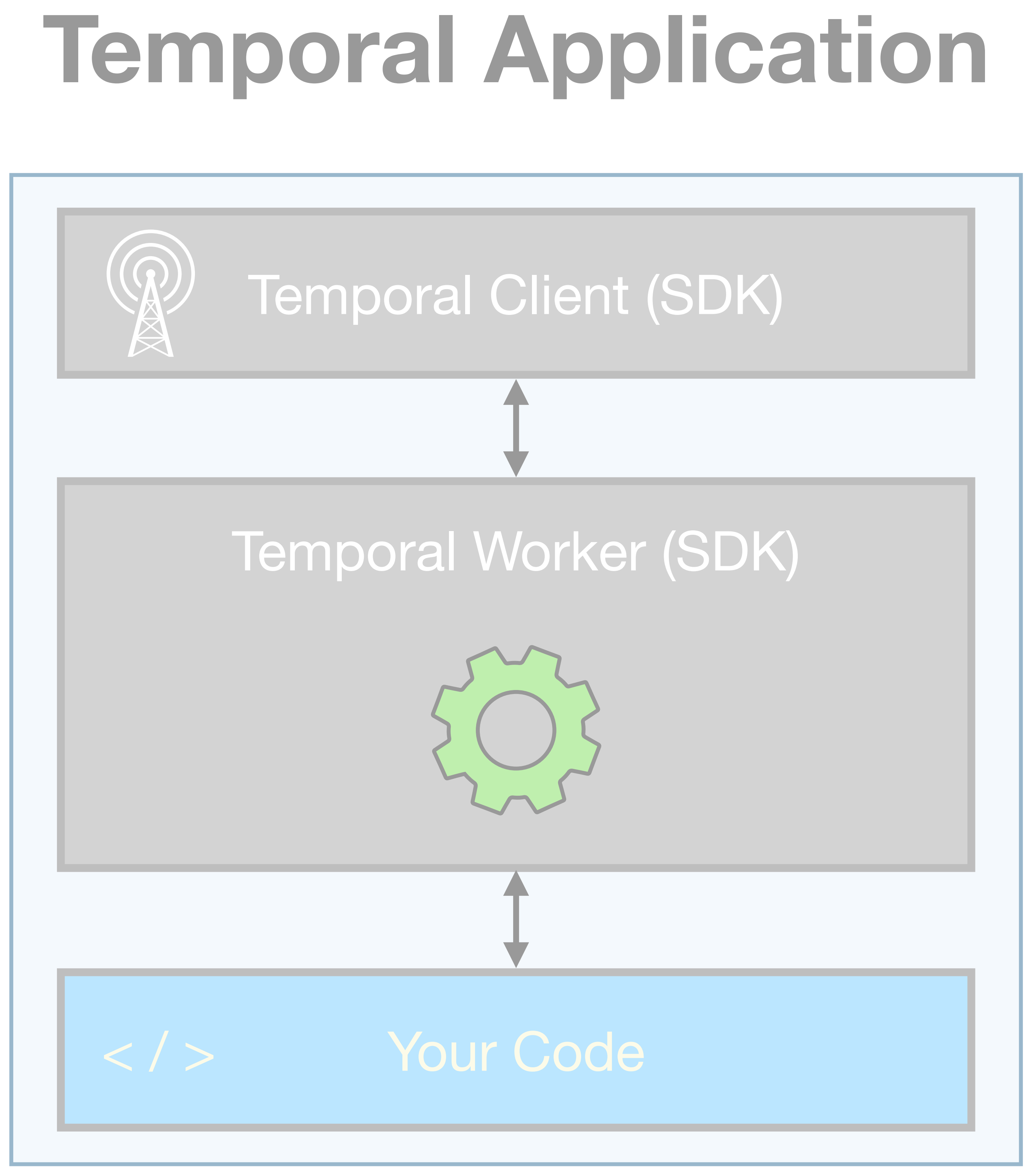

Temporal Client (SDK)
Temporal Worker (SDK)
< / > Worker Configuration
< / > Activity Definitions
< / > Workflow Definition
Provided by SDK
You develop

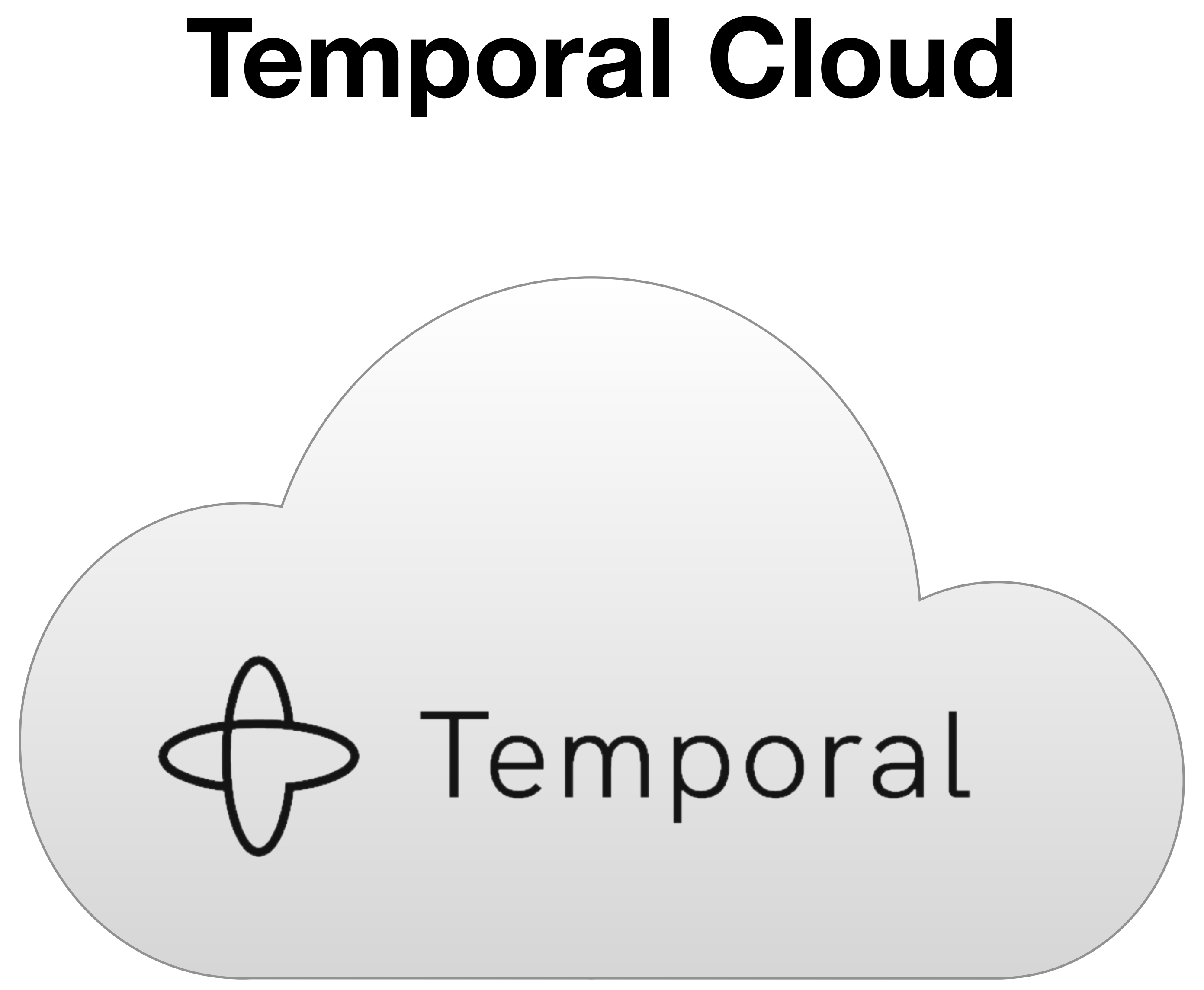
The Role of a Local Temporal Cluster



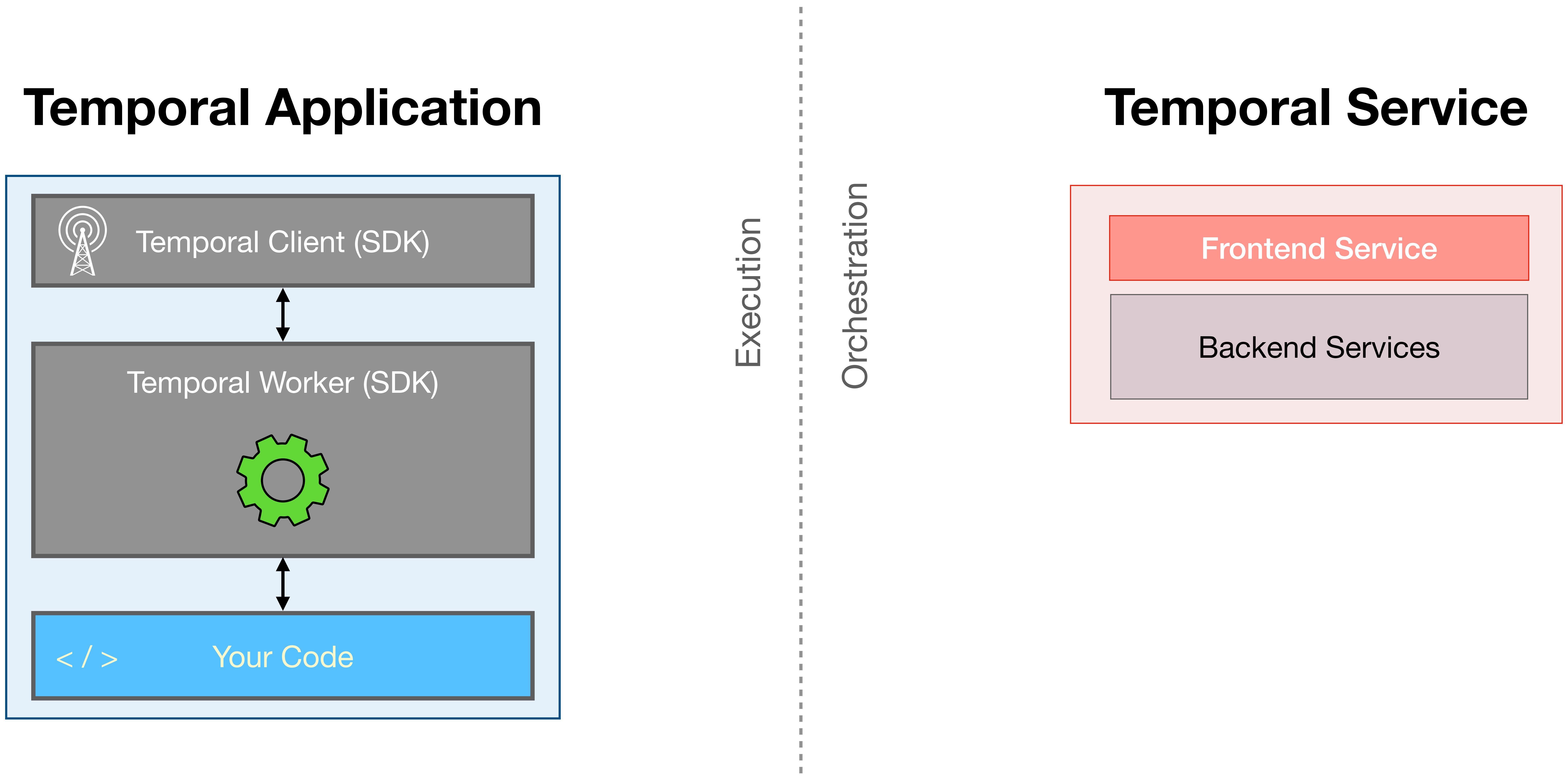


The Role of Temporal Cloud

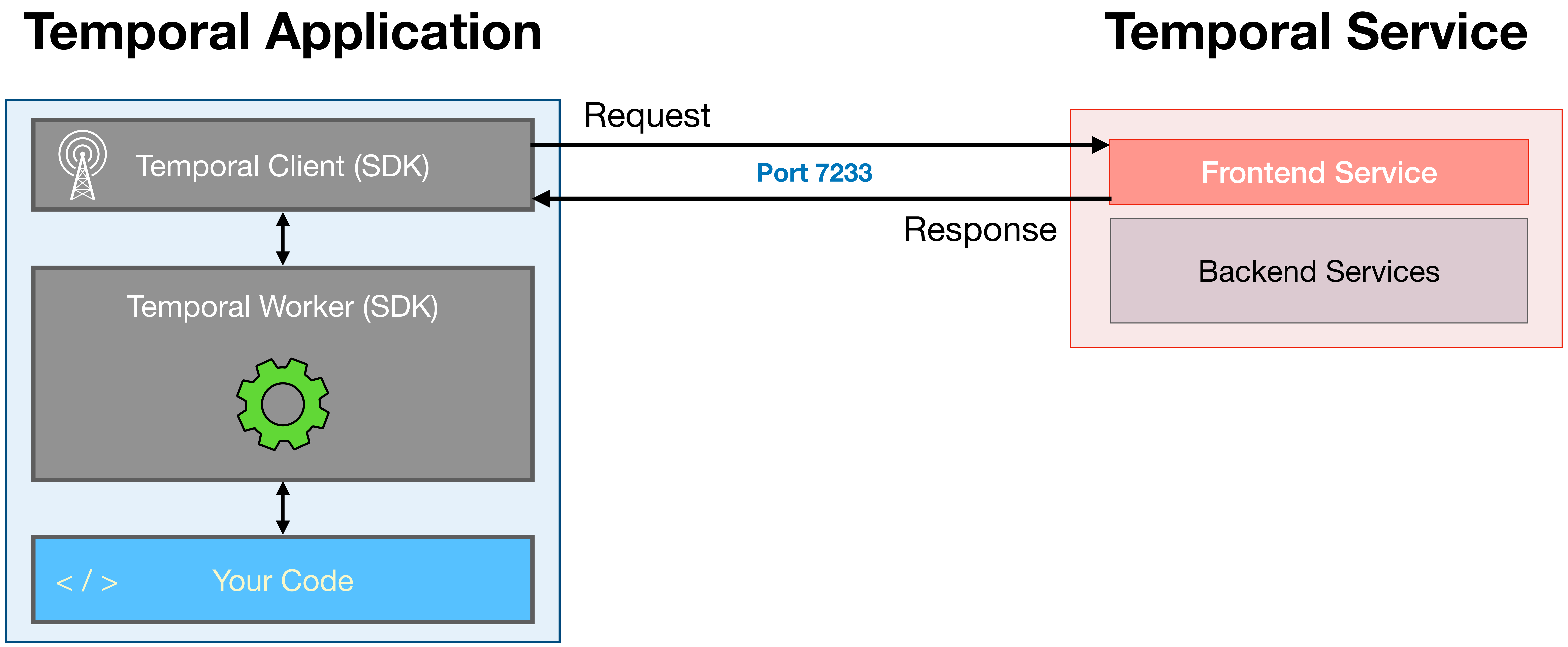




Applications Are External to the Service



Temporal Uses gRPC for Communication



Review

- **Temporal is a Durable Execution system**
  - Ensures that your application runs reliably despite adverse conditions
  - Automatically maintains application state and recovers from failure
- **Workflows represent the sequence of steps used to carry out your business logic. They must be deterministic**
- **Activities encapsulate unreliable or non-deterministic code. They should be idempotent because they can be retried**
- **Workers execute Workflow and Activity Definitions by polling a Task Queue**
- **Your Workers, Workflows, and Activities make up a Temporal Application and are separate from the Temporal Service**

Temporal 102

Compatible Evolution of Input Parameters

- **Workflows and Activities can take any number of parameters as input**
  - Changing the number, position, or type of these parameters can affect backwards compatibility
- **It is a best practice to pass all input in a single `Object`**
  - Define your own class
  - Changes to the composition of this class does not affect the method signature
- **This is also the recommended approach for return values**
  - Using classes in both places allows for evolution of input and output data

Example: Using a `class` in an Activity (1)

- **Imagine that you have the following Activity**

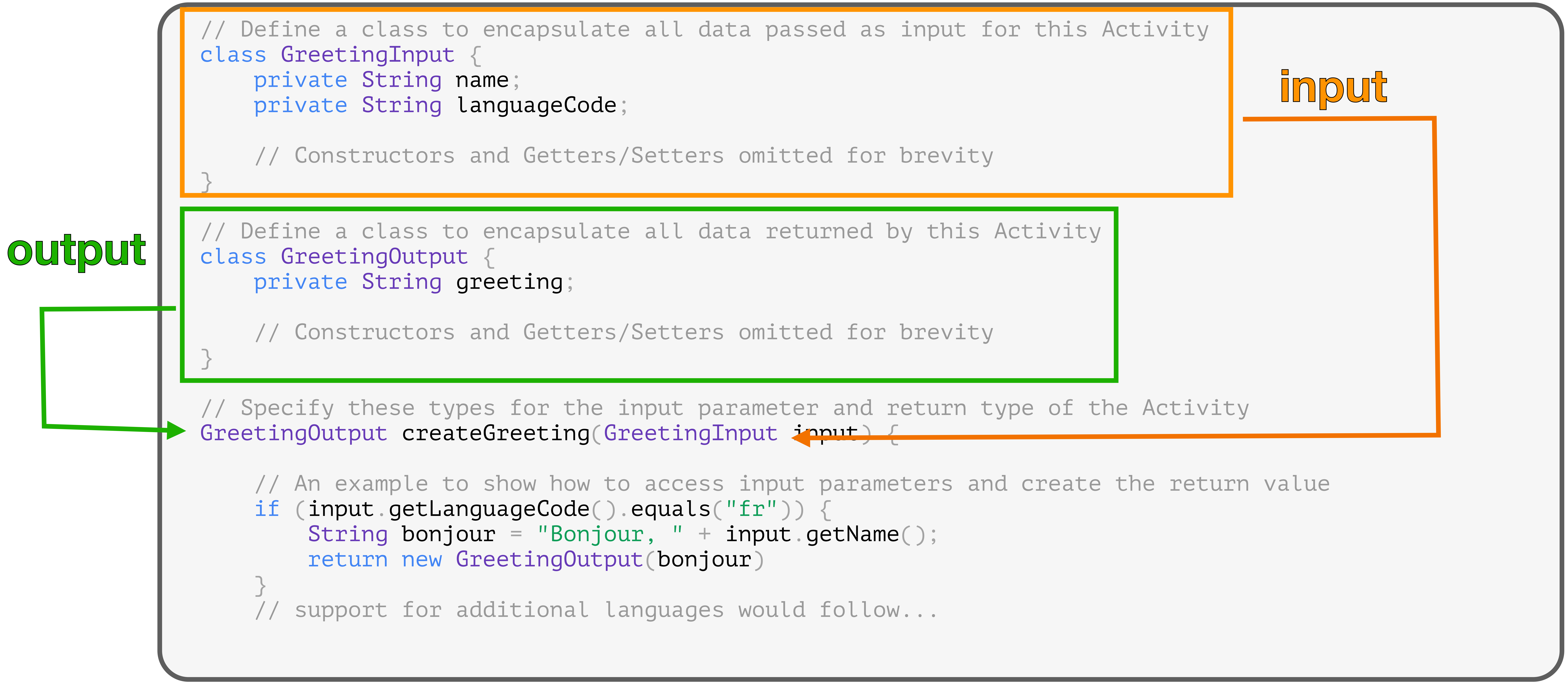
```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

output input

- **You later need to update it to support other languages, such as Spanish**
  - Changing what is passed into or returned from the method changes its signature
  - Changes to the class composition don't affect the signature of the methods that use it

Example: Using a `class` in an Activity (2)

- **The following code sample illustrates how you could support this**



```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**
  - Do not hardcode the name in multiple places: Use a shared constant if possible
  - Avoid mixed case: Task Queue names are case sensitive
  - Use descriptive names, but make them as short and simple as practical
- **Plan to run *at least* two Worker Processes per Task Queue**

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

- **Must be unique among all *running* Workflow Executions in the namespace**
  - This constraint applies across *all* Workflow Types, not just those of the *same Type*
  - This is an important consideration for choosing a Workflow ID

Workflow ID Reuse Examples

- **You can specify the Workflow ID Reuse Policy through the SDK or via command line**

```
package app

// The following import is needed to reference the Workflow ID Reuse Policy value
import io.temporal.api.enums.v1.WorkflowIdReusePolicy;

// other code removed for brevity

// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("example-workflow-id")
        .setTaskQueue(Constants.taskQueueName)
        .setWorkflowIdReusePolicy(
            WorkflowIdReusePolicy.WORKFLOW_ID_REUSE_POLICY_ALLOW_DUPLICATE)
        .build();

MyWorkflow workflow = client.newWorkflowStub(MyWorkflow.class, options);
// additional code would follow
```

How Exceptions Affect Activity Execution

- **An Activity that raises an exception is considered as failed**
  - It may or may not retried, based on the Retry Policy associated with its execution
  - By default, Activity Execution is associated with a Retry Policy
    - The default policy results in retrying until execution succeeds or is canceled

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Raising an exception in a Workflow causes a Workflow Task Failure
    - This failure will be retried
  - Raising an exception that extends `TemporalFailure` causes a Workflow Execution Failure
    - The Workflow will be marked as Failed and not retried

Logging in Temporal Applications

- **The recommended way of logging in Workflows is via an `slf4j` implementation provide by the Temporal Java SDK**
  - Is replay aware
- **The standard log levels are present, in increasing order of importance**
  - `debug`
  - `info`
  - `warn`
  - `error`

Using the WorkflowLogger

- **Accessing and using the Workflow logger using `Workflow.getLogger`**

```
import org.slf4j.Logger;
import io.temporal.workflow.Workflow;

// instance variable
public static final Logger logger = Workflow.getLogger(TranslationWorkflowImpl.class);

// code within a method
logger.debug("Preparing to execute an Activity")
logger.info("Calculated cost of order. Tax {}, Total {}", tax, total)
```

Logging in Activities

- **Activity logging needs no special Temporal Logger and can be done normally**

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

// instance variable
private static final Logger logger = LoggerFactory.getLogger(TranslationActivitiesImpl.class);

// code within a method
logger.info("getDistance invoked; determining distance to customer address");
logger.error("Database connection failed");
```

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**
  - Synchronous calls block, awaiting the Workflow or Activity result before continuing
  - Asynchronous calls do not block, so the result must be retrieved later
- **Workflows run until all Tasks yield and resume when new ones appear**
  - Example: If a Worker encounters a synchronous Activity call, it halts the current Workflow Task and requests Activity Execution. Once that completes, the Temporal Service adds a new Workflow Task to the queue.

Activity Execution

- **Synchronous Execution**

```
private final GreetingActivities activities =
  Workflow.newActivityStub(GreetingActivities.class, options);

// Synchronous Activity Method call
String greeting = activities.createGreeting(bill);
```

- **Asynchronous Execution**

```
private final GreetingActivities activities =
  Workflow.newActivityStub(GreetingActivities.class, options);

// Asynchronous Activity Method call
Promise<String> hello = Async.function(activities::createGreeting, name);

// Later in the program
String result = hello.get();
```

Workflow Execution

- **Synchronous Execution**

```java
// Use a client to request Workflow Execution.
GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);
String greeting = workflow.greetSomeone(name);
```

- **Asynchronous Execution**

```java
import java.util.concurrent.CompletableFuture;
import io.temporal.client.WorkflowClient;

// Options defining code omitted for brevity
GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

// Workflow will be started at this point but the call doesn't block.
CompletableFuture<String> greeting = WorkflowClient.execute(workflow::greetSomeone, "World");

// This line will block, waiting on the result from the Workflow.
String result = greeting.get();
```

Deferring Access to Execution Results

- **Deferring access to results *may* reduce overall execution time**
  - This is a good strategy when a Workflow needs to call unrelated Activities
  - It allows these Activities to execute in parallel, blocking only while accessing their results

```
Promise<String> hello = Async.function(activities::greetInSpanish, name);
Promise<String> goodbye = Async.function(activities::farewellInSpanish, name);
Promise<String> thanks = Async.function(activities::thankInSpanish, name);

// The following lines block until their respective executions have finished

String hello_result = hello.get();
String goodbye_result = goodbye.get();
String thanks_result = thanks.get();
// You could also collect them all at once
```

Temporal 102

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration
  - The duration is fixed and may range from seconds to years
  - Once the time has elapsed, the Timer fires, and execution continues
- **Workflow code must not use Java's built-in timers or sleep (non-deterministic)**

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**
  - Delay calling the next Activity based on a value returned by a previous one
- **Allow time for offline steps to complete**
  - Wait five business days for a check to clear before proceeding

Timer APIs Provided by the Java SDK

- **Java SDK offers two Workflow-safe, replay-aware ways to start a Timer**
  - There are synchronous and asynchronous versions
  - A Workflow-safe replacement for `Thread.sleep` and `java.util.Timer` are available
  - Workflow code must not use Java's methods for timers (non-deterministic)

Pausing Workflow Execution for a Specified Duration

- **Use the `Workflow.sleep` method for this**
  - This is an alternative to Java's `Thread.sleep` method
  - It blocks until the Timer is fired (or is canceled)

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// This will pause Workflow Execution for 10 seconds
Workflow.sleep(Duration.ofSeconds(10));
```

Running Code a Specific Point in the Future

- **Use the `Workflow.newTimer` method for this**
  - This is an alternative to Java's `java.util.NewTimer` method
  - This returns a `Promise`, which becomes ready when the Timer fires (or is canceled)

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// Workflow.newTimer is a Workflow-safe counterpart to java.util.Timer
Promise timerPromise = Workflow.newTimer(Duration.ofSeconds(30))
logger.info("The timer was set")

// Unlike Workflow.sleep, waiting for the timer is a separate operation
timerPromise.get()
logger.Info("The timer has fired")
```

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires
    - It will be as if the Worker had never crashed at all
  - If Worker is restarted 5 *minutes* later, the Timer will have already fired
    - In this case, the Worker will resume executing the Workflow code without delay

Exercise #1: Observing Durable Execution

https://t.mp/102-java-exercise-env

- **During this exercise, you will**
  - Create Workflow and Activity loggers
  - Add logging statements to the code
  - Add a Timer to the Workflow Definition
  - Launch two Workers, run the Workflow, and kill one of the Workers, observing that the remaining Worker completes the execution
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Essential Points

- Timers introduce delays into a Workflow Execution
- Timers are durable, meaning they can survive a crash of the Worker who invoked it
- Timers are maintained by the Temporal Service and recorded in the Event History
- Example Timer Use Cases:
  - Execute an Activity multiple times at predefined or calculated intervals
  - Allow time for offline steps to occur

Temporal 102

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

Workflow Definition

combined with

+

Execution Request

```java
MyWorkflowOutput greeting = workflow.greetSomeone("Brian");
```

results in

=

Workflow Execution

Running Workflow

```
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

1 Workflow Definition

combined with

+ \+

n Execution Requests

```
workflow.greetSomeone("Brian");
```

```
workflow.greetSomeone("Tom");
```

results in

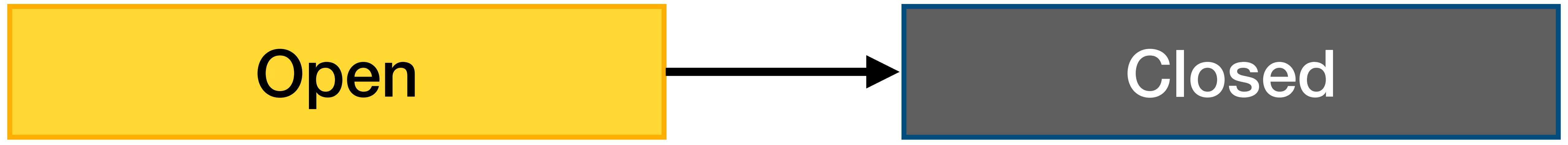
= =

n Workflow Executions

Workflow Execution 1

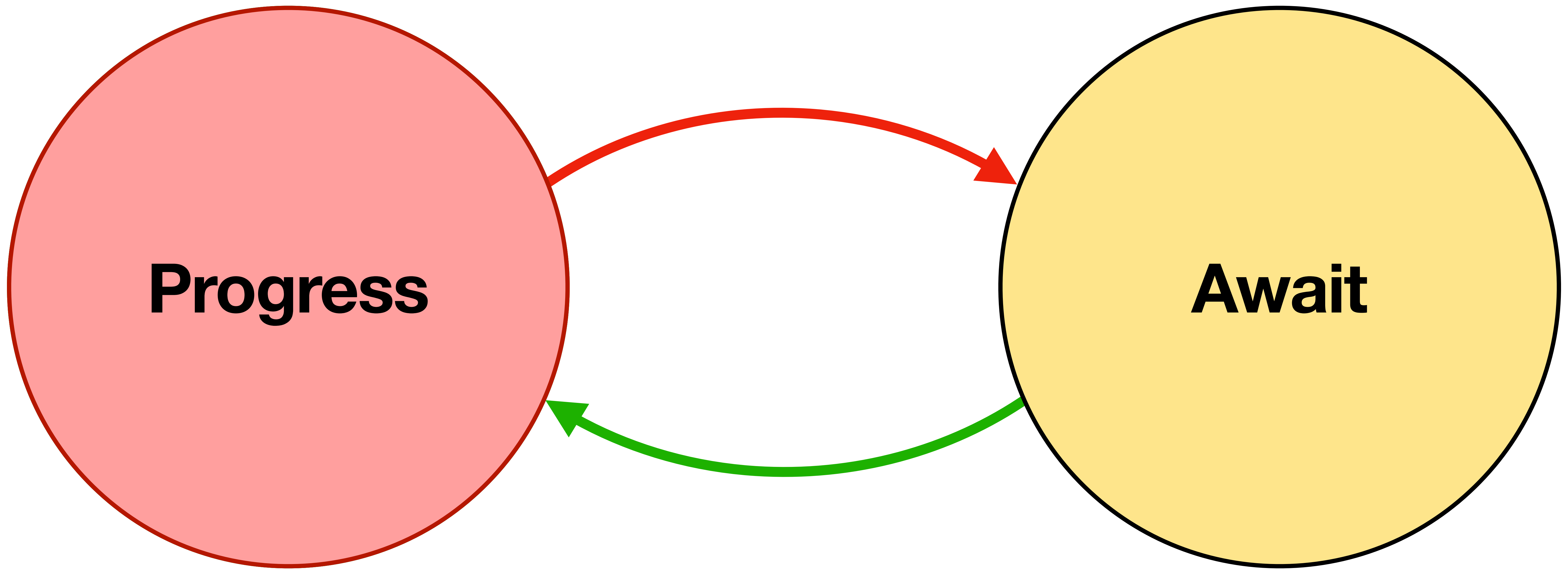
Workflow Execution 2

Workflow Execution States



This is a one-way transition

What Happens During Workflow Execution

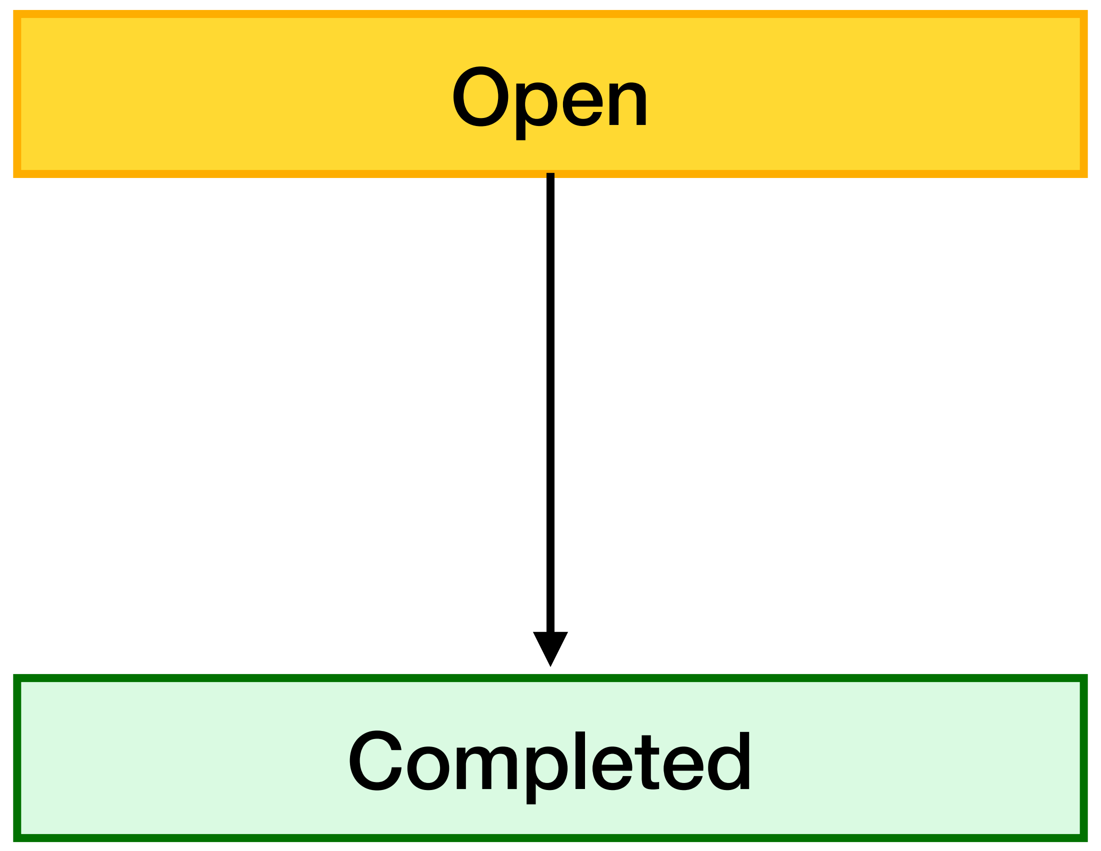


This cycle continues throughout Workflow Execution

Workflow Execution States

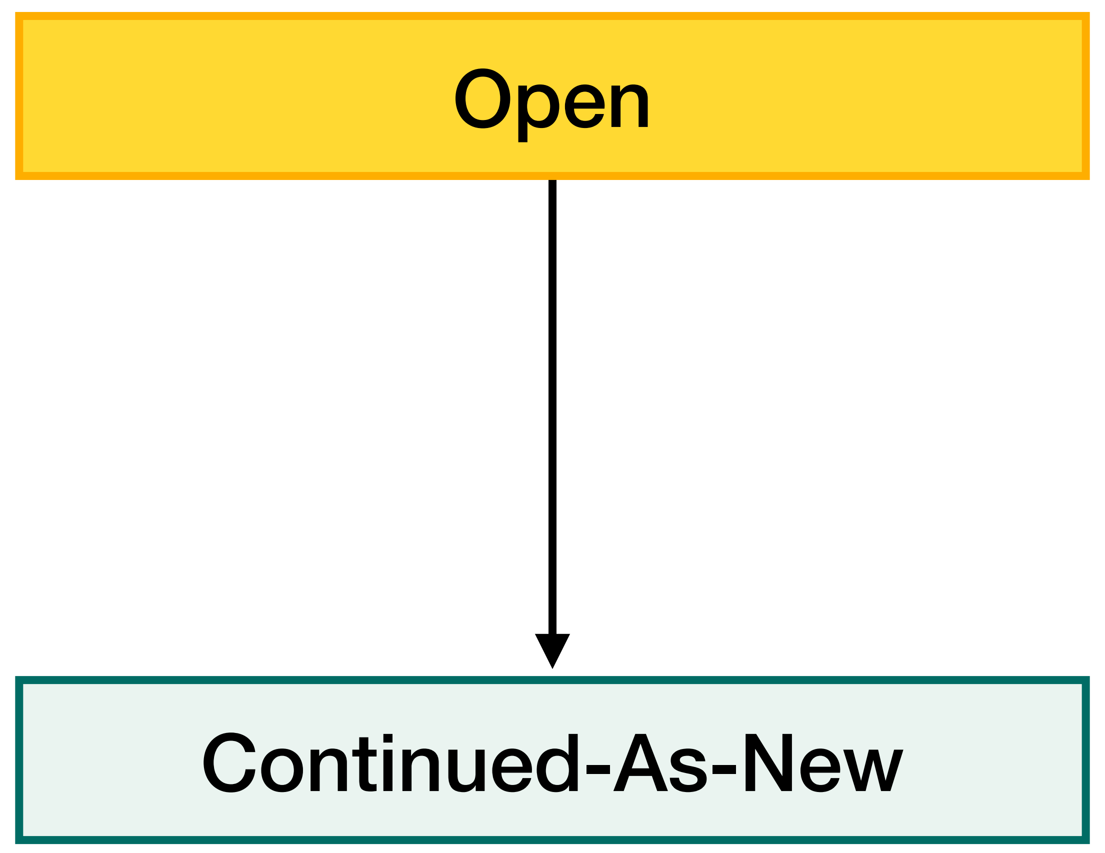
Completed

Meaning: The Workflow method returned a result



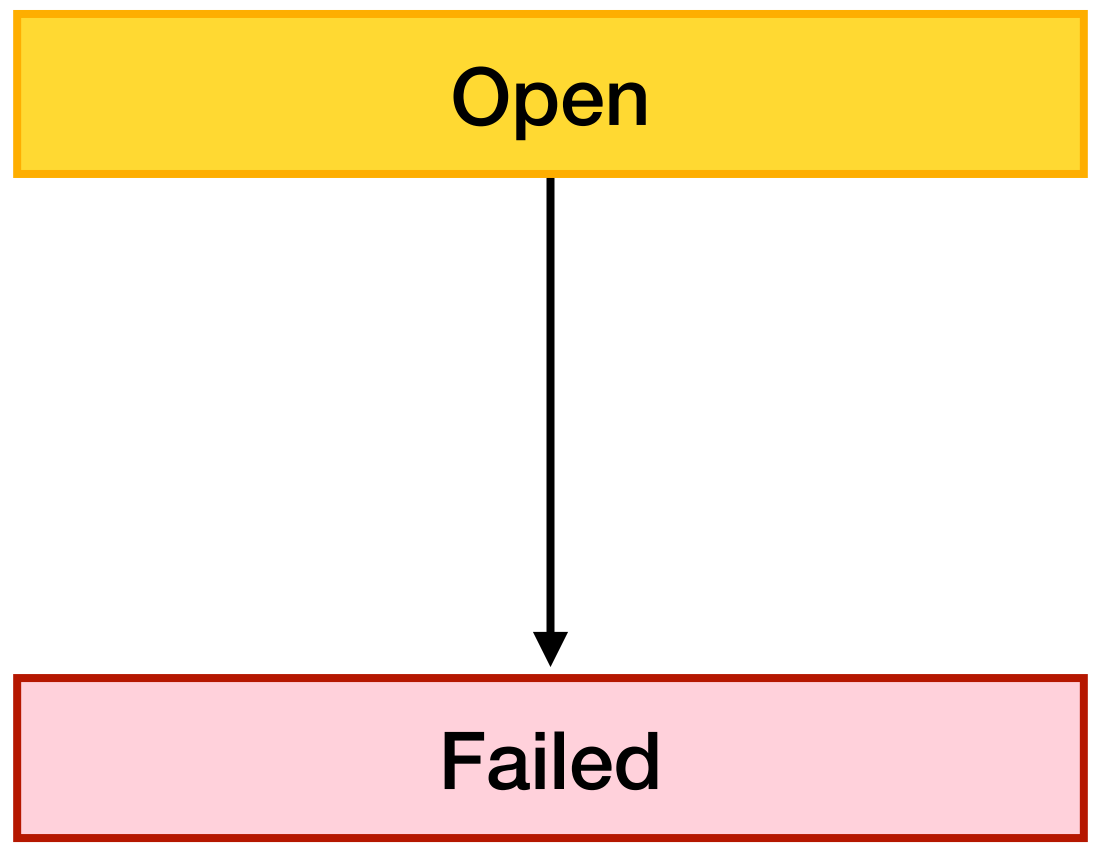
Continued-As-New

Meaning: Future progress will take place in a new Workflow Execution



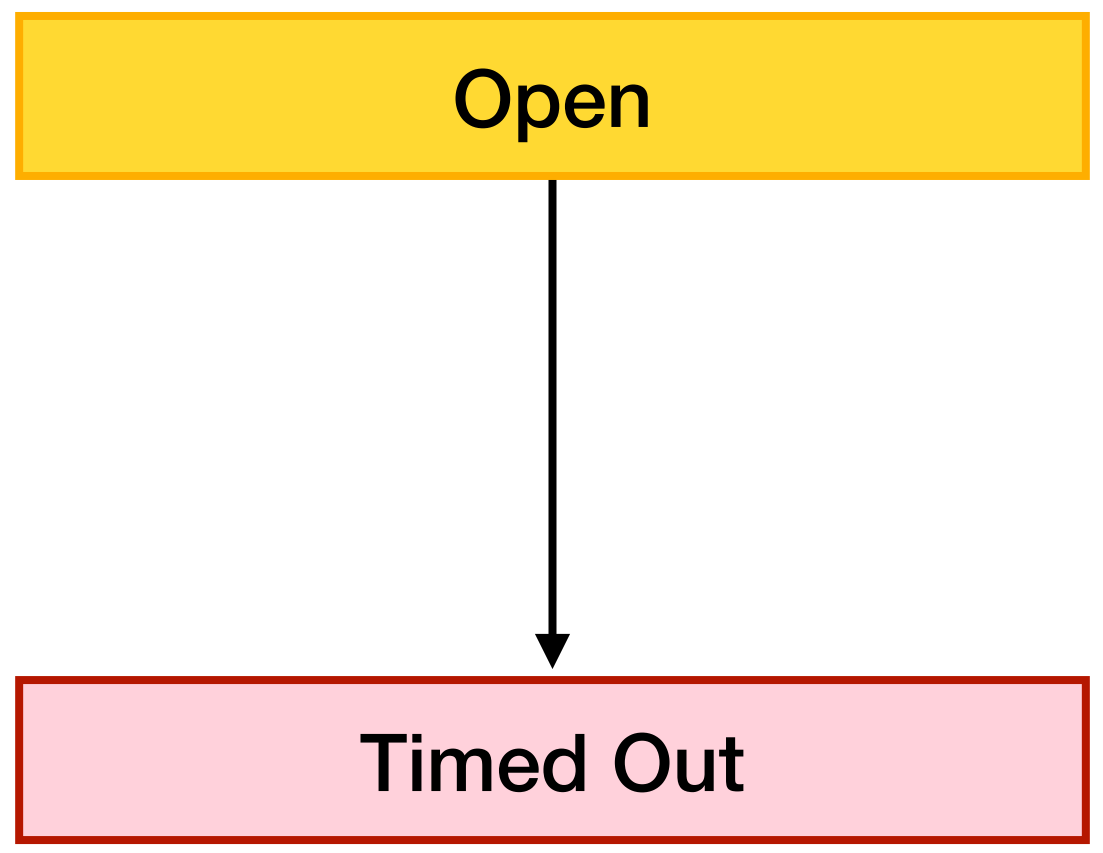
Failed

Meaning: The Workflow method raised an exception



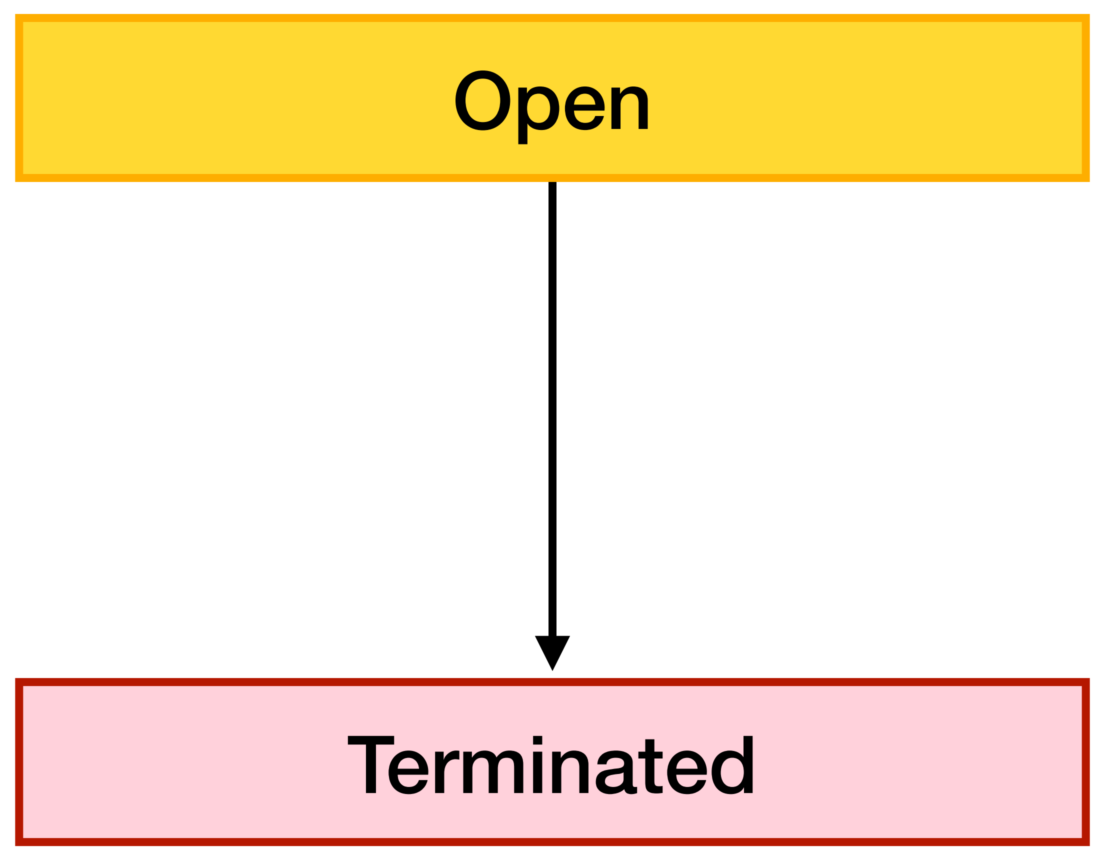
Timed Out

Meaning: Execution exceeded a specified time limit



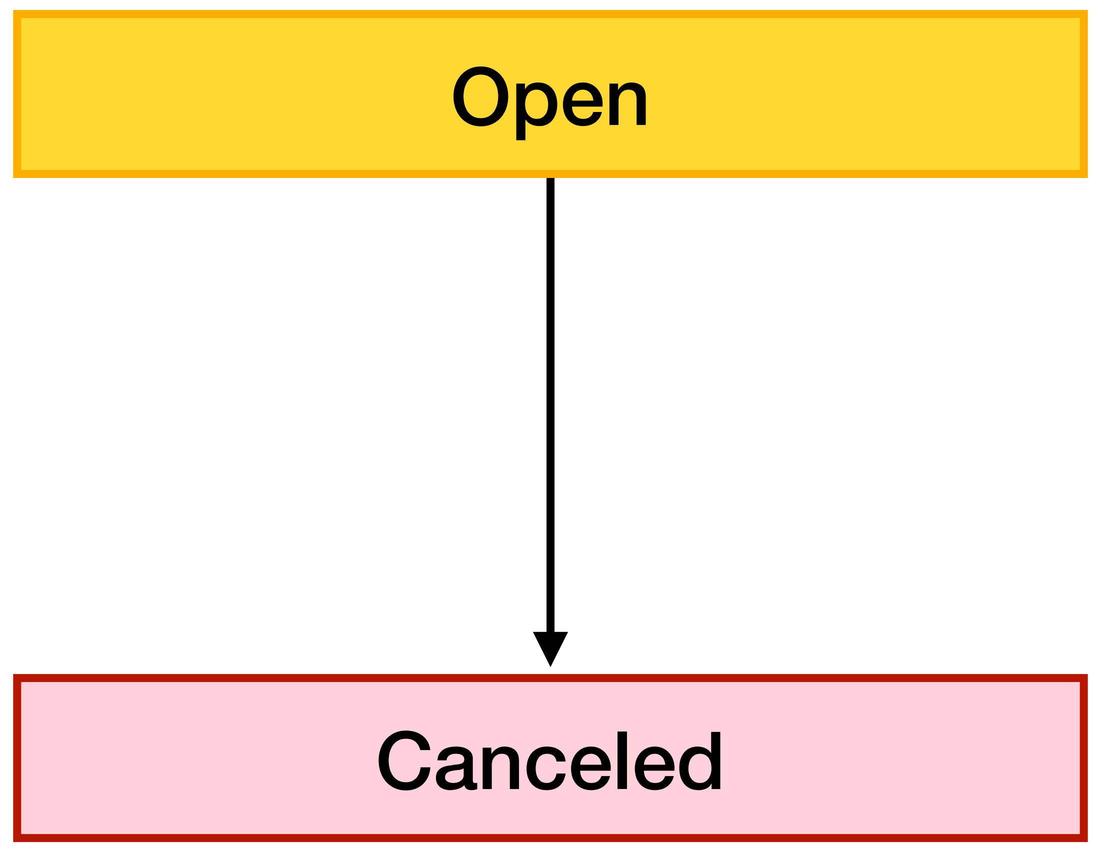
Terminated

Meaning: Temporal Service acted upon a termination request

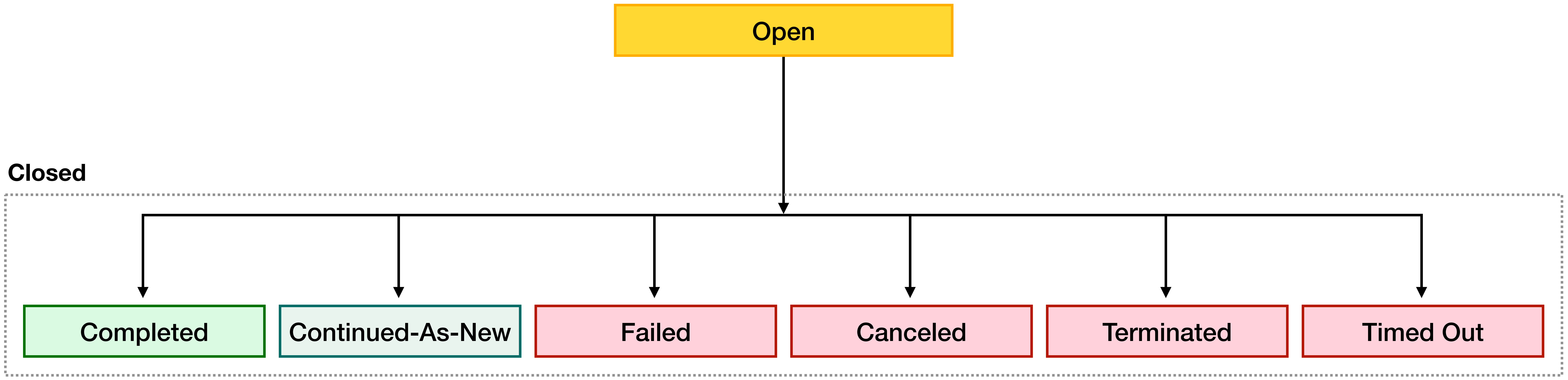


Canceled

Meaning: Temporal Service acted upon a request to cancel execution

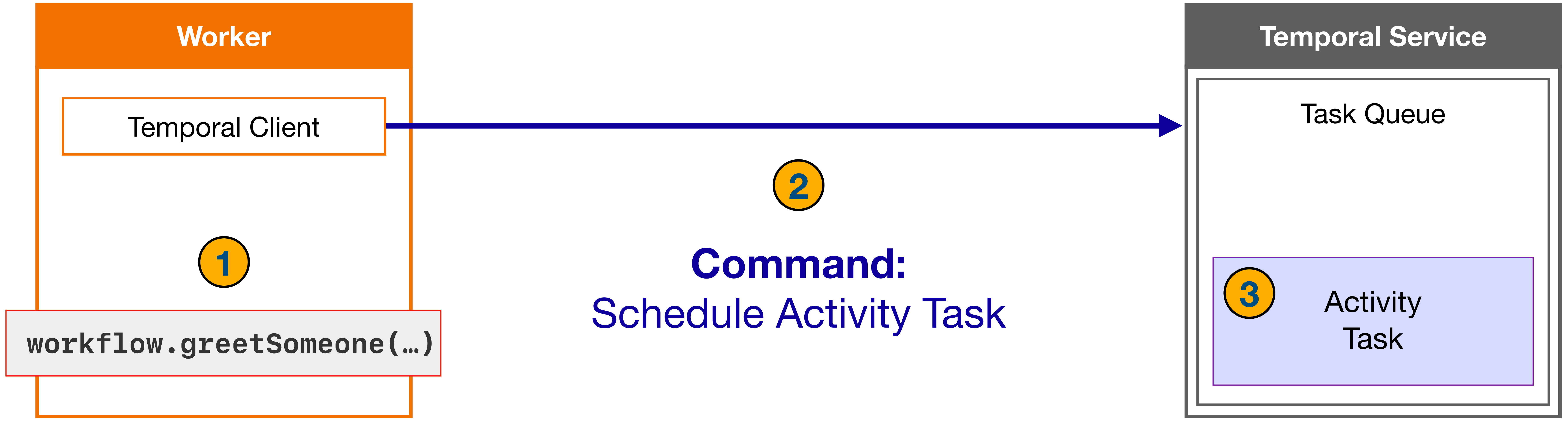


Summary of Workflow Execution States



How Workflow Code Maps to Commands

Commands



- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Service acts on these commands, but also stores them
- This allows the Worker to recreate the state of a Workflow Execution following a crash

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
       .setStartToCloseTimeout(Duration.ofSeconds(5))
       .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery
- Sleeps for 30 minutes
- Populates a class with billing information
- Sends a bill to the customer

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters
  - Evaluating variables or expressions
  - Performing calculations
  - Populating data structures
- These internal steps are highlighted in white

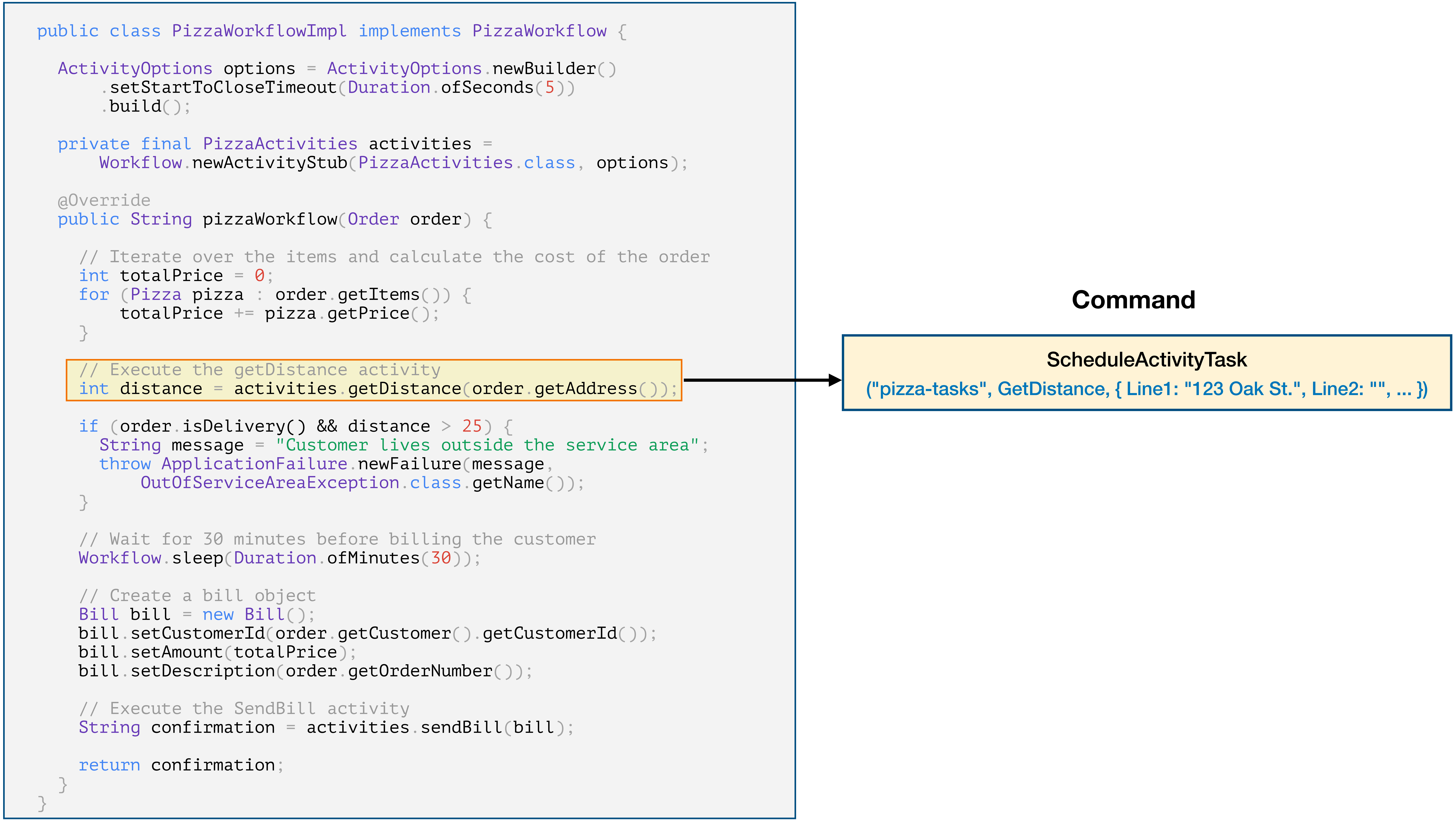
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity
  - Throwing an exception from the Workflow
  - Setting a Timer
  - Returning a value from the Workflow
- These external steps are highlighted in yellow

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {
  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);
  @Override
  public String pizzaWorkflow(Order order) {
    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command
ScheduleActivityTask
("pizza-tasks", GetDistance, { Line1: "123 Oak St.", Line2: "", ... })

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

StartTimer
(30 minutes)

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

ScheduleActivityTask

("pizza-tasks", SendBill, { Amount: 2750, Description: "Pizzas", ... }

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

CompleteWorkflowExecution

({ConfirmationNumber: "TPD-26074139"})

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**
  - Allow reconstruction of Workflow state following a crash
  - Enable developers to investigate both current and past executions
- **You can access them from code, command line, and Web UI**

Event History Content

- **An Event History acts as an ordered append-only record of Events**
  - Begins with the `WorkflowExecutionStarted` Event
  - New Events are appended as Workflow Execution progresses
  - Ends when the Workflow Execution closes

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution
- **Workflow Execution will be *terminated* after exceeding additional limits**
  - If its Event History exceeds 50K (51,200) Events
  - If its Event History exceeds 50 MB of storage

Event Structure and Characteristics

- **Every Event always contains the following three attributes**
  - ID (uniquely identifies this Event within the History)
  - Time (timestamp representing when the Event occurred)
  - Type (the kind of Event it is)

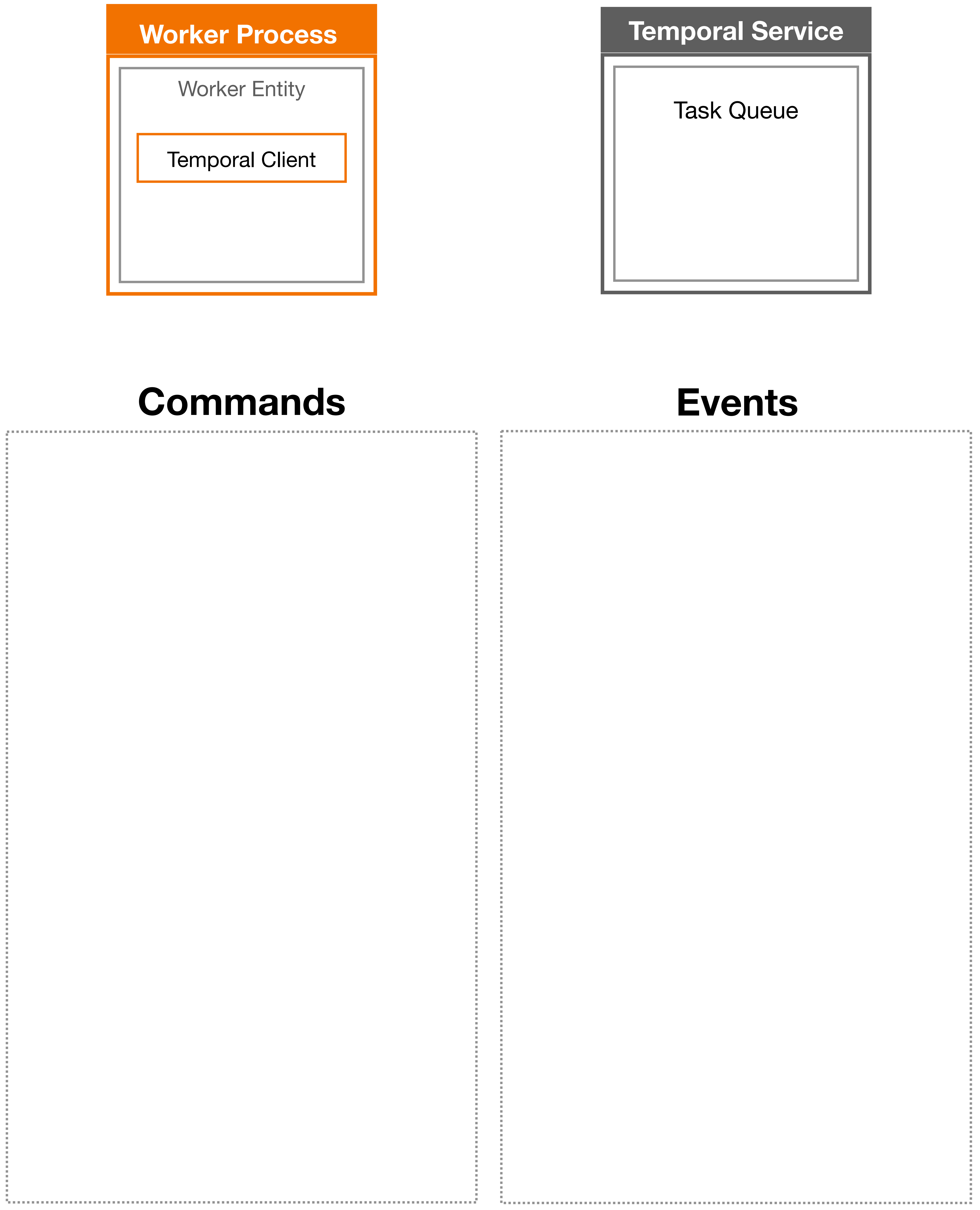
Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **`WorkflowExecutionStarted`** contains the Workflow Type and input parameters
  - **`WorkflowExecutionCompleted`** contains the result returned by the Workflow method
  - **`WorkflowExecutionFailed`** contains the exception thrown by the Workflow method
  - **`ActivityTaskScheduled`** contains the Activity Type and input parameters
  - **`ActivityTaskCompleted`** contains the result returned by the Activity method

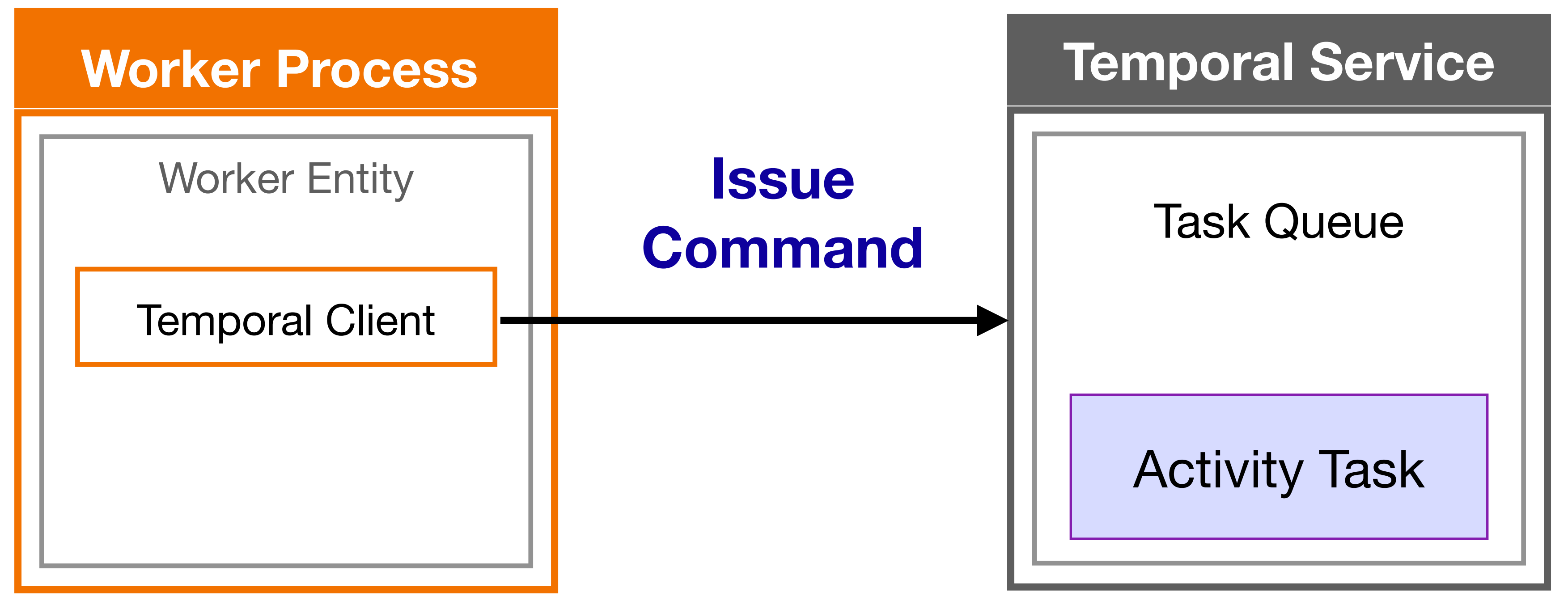
How Commands Map to Events

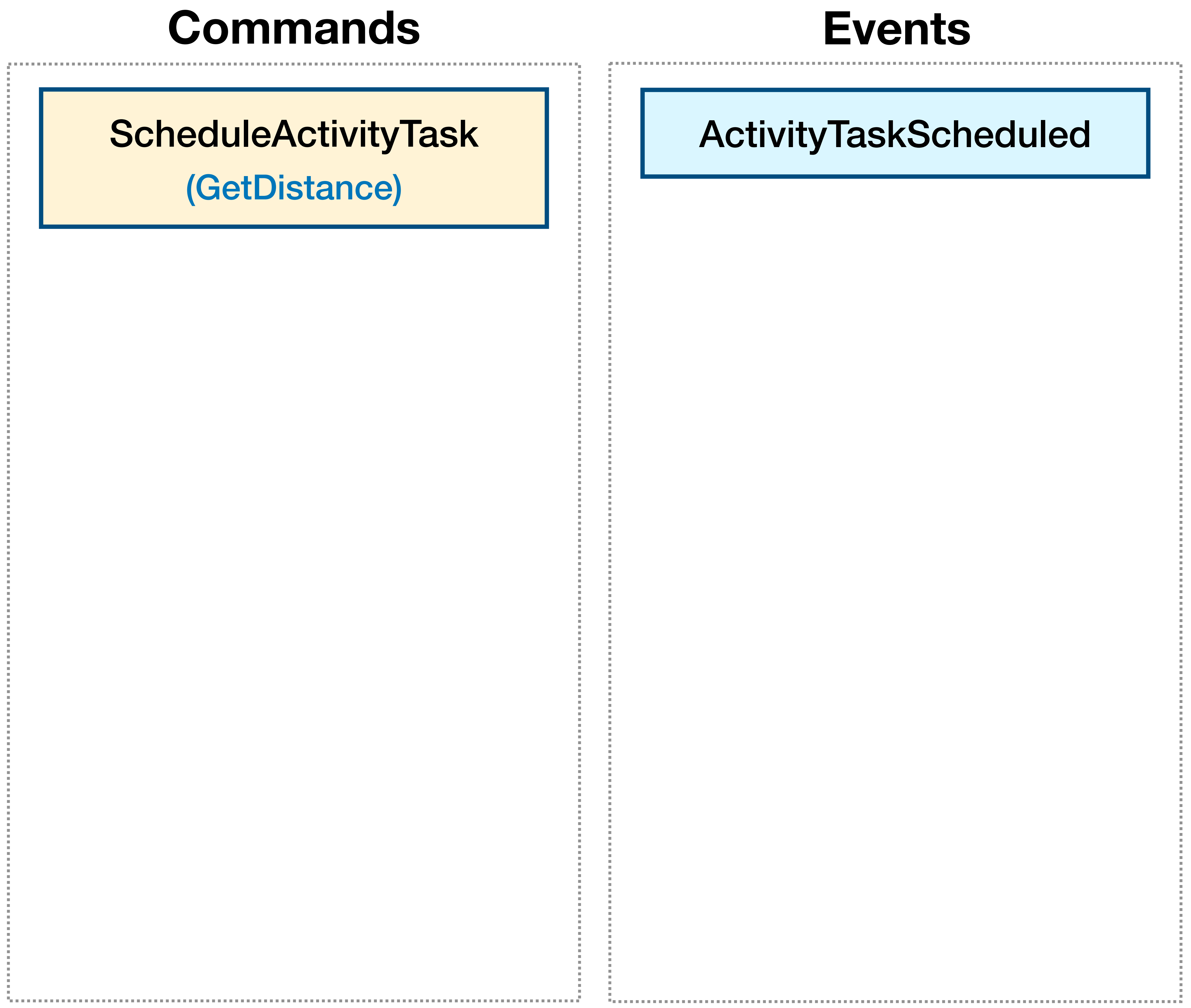
pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

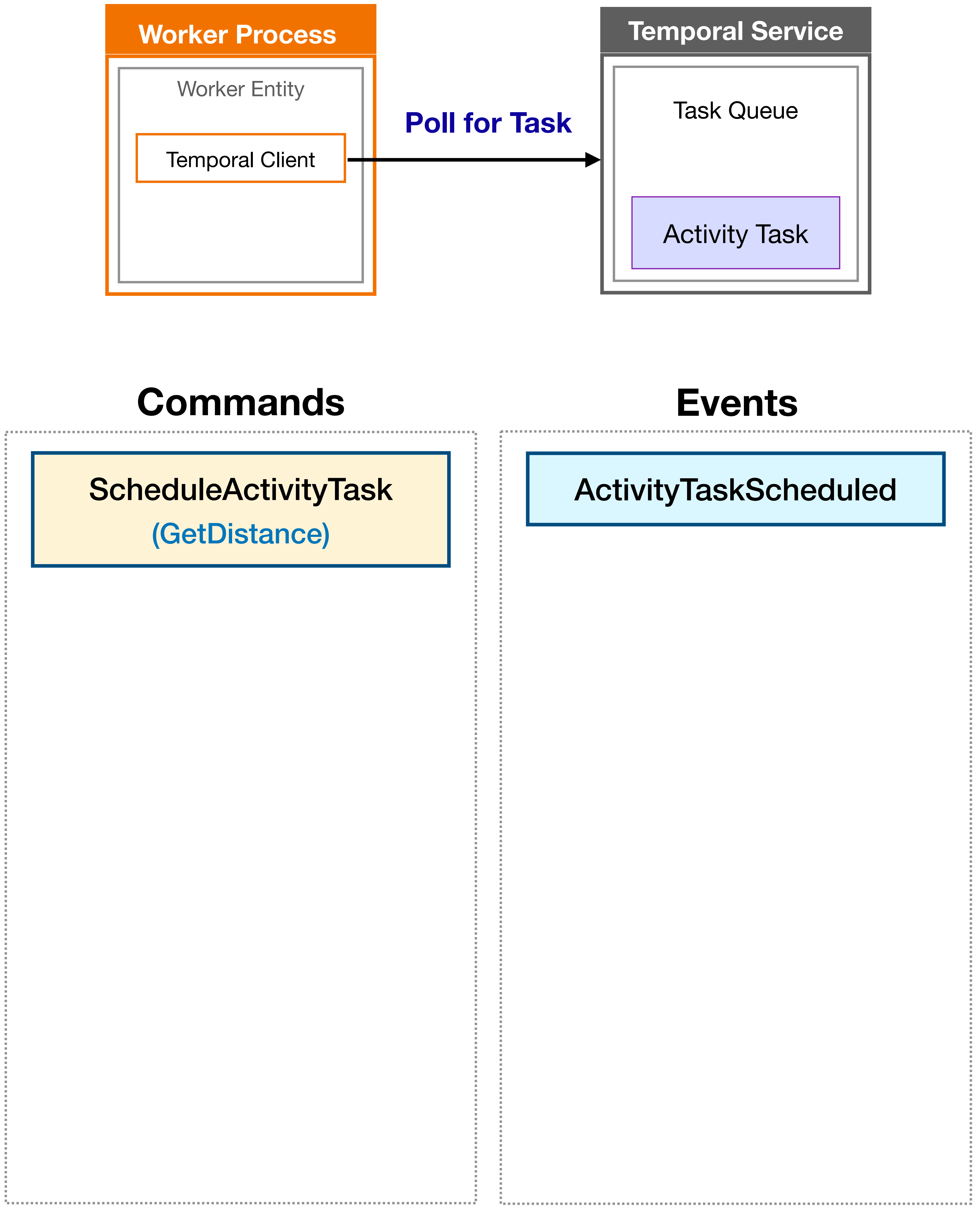


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

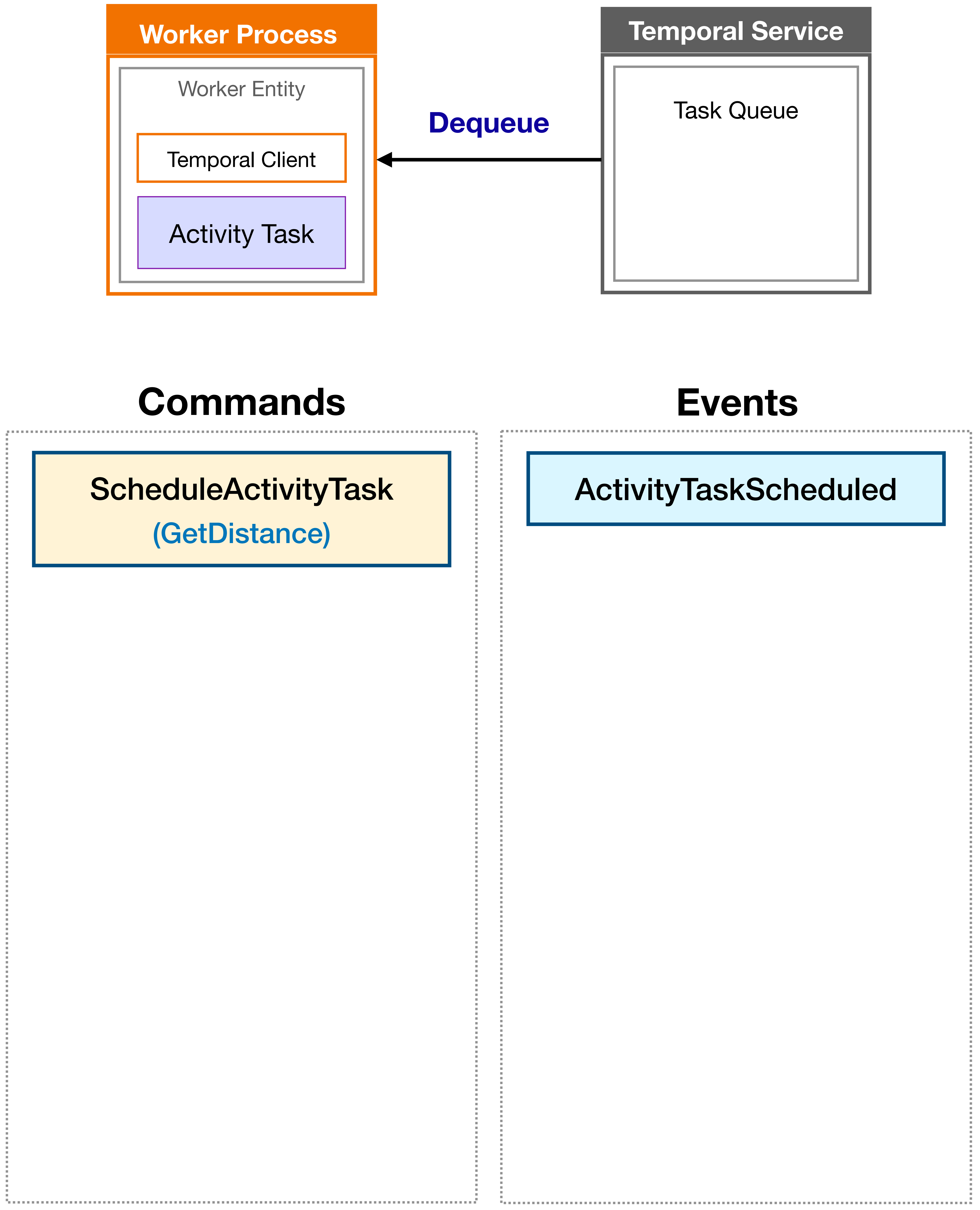




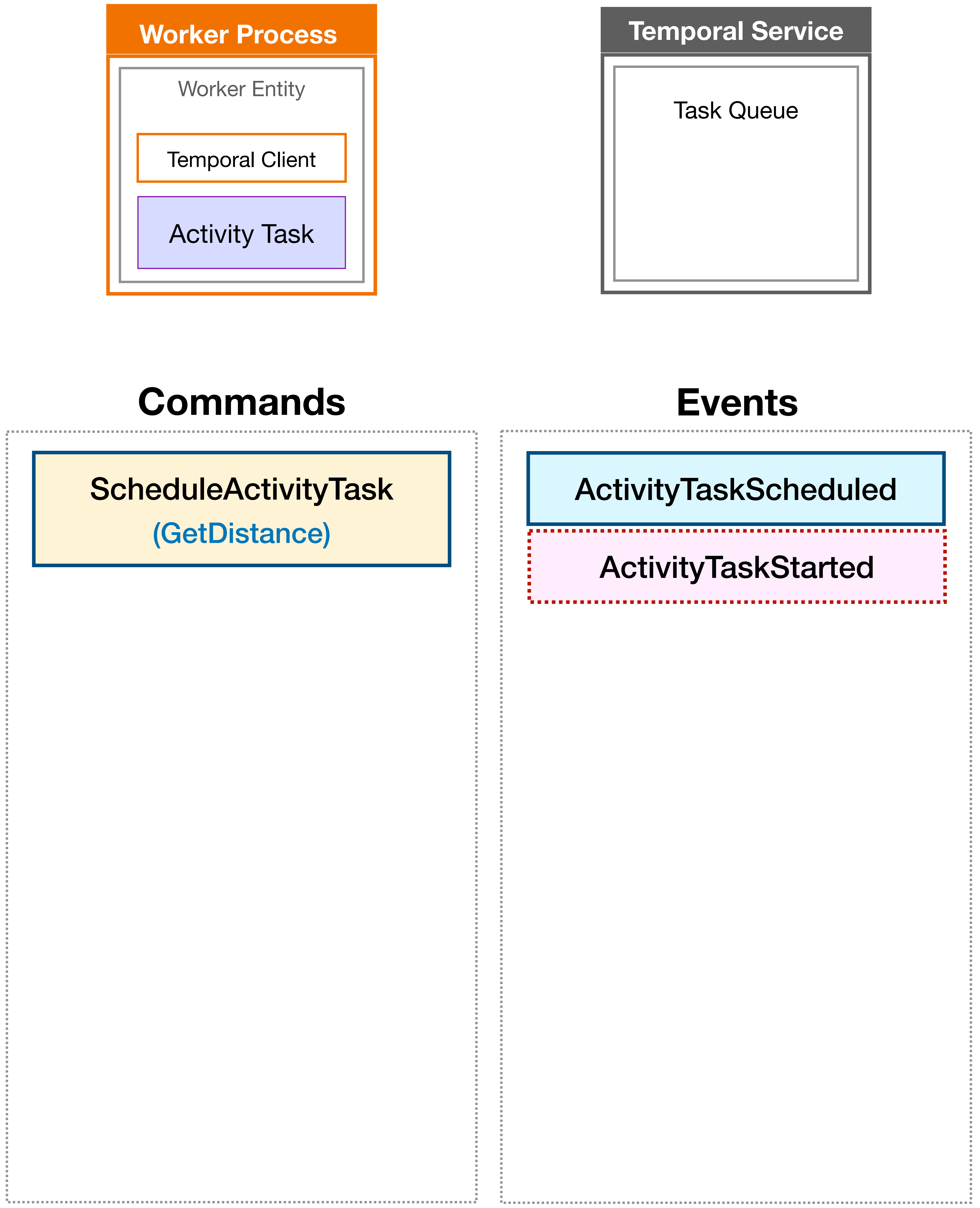
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



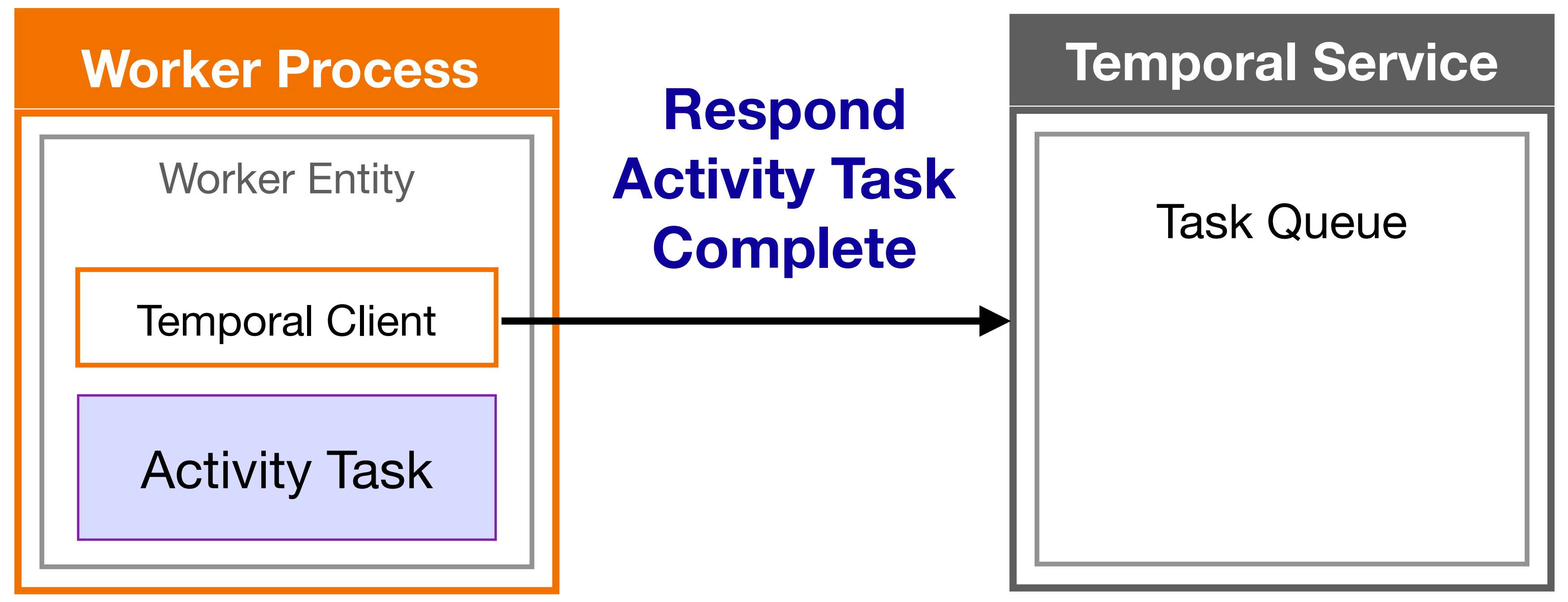
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

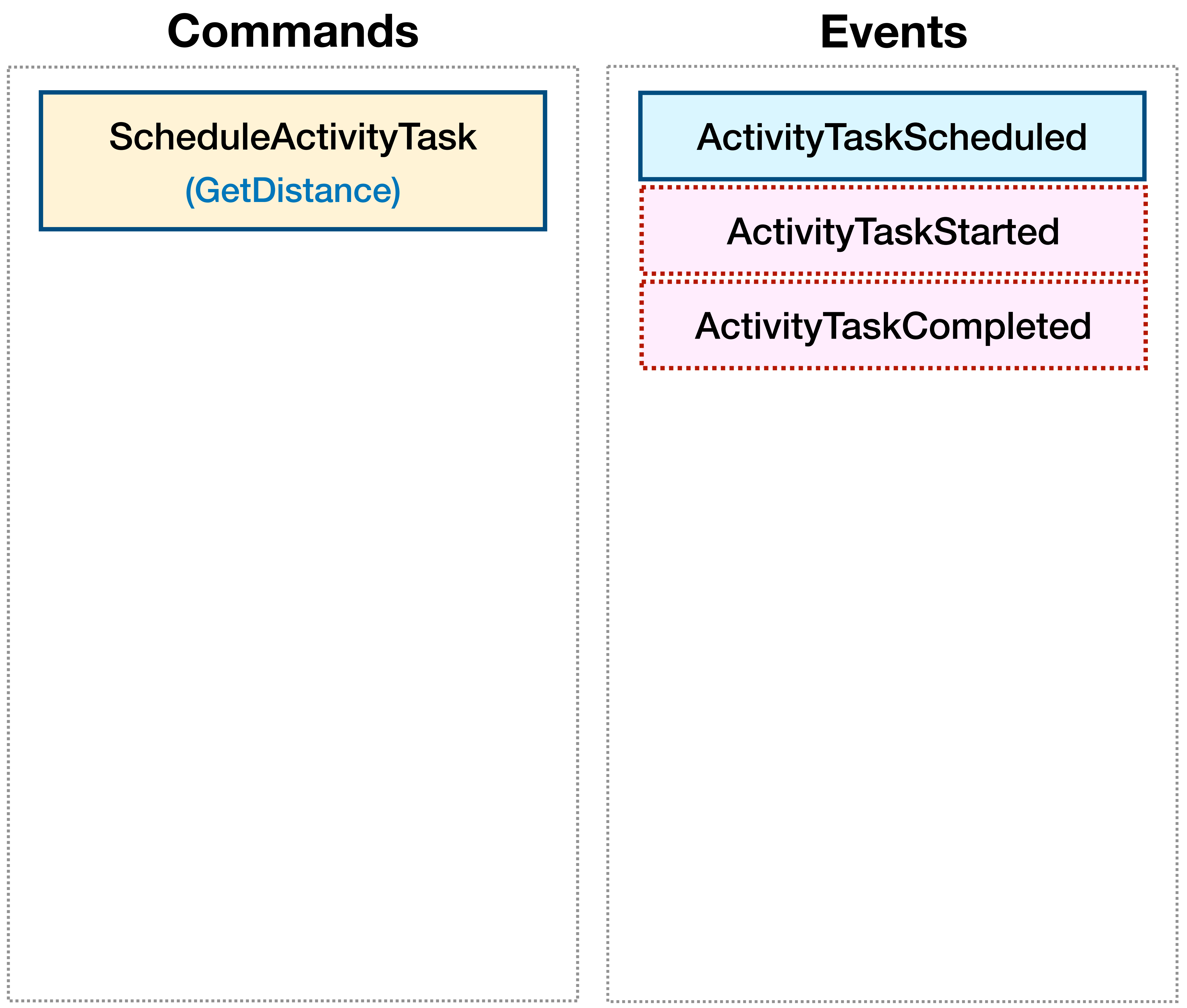


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

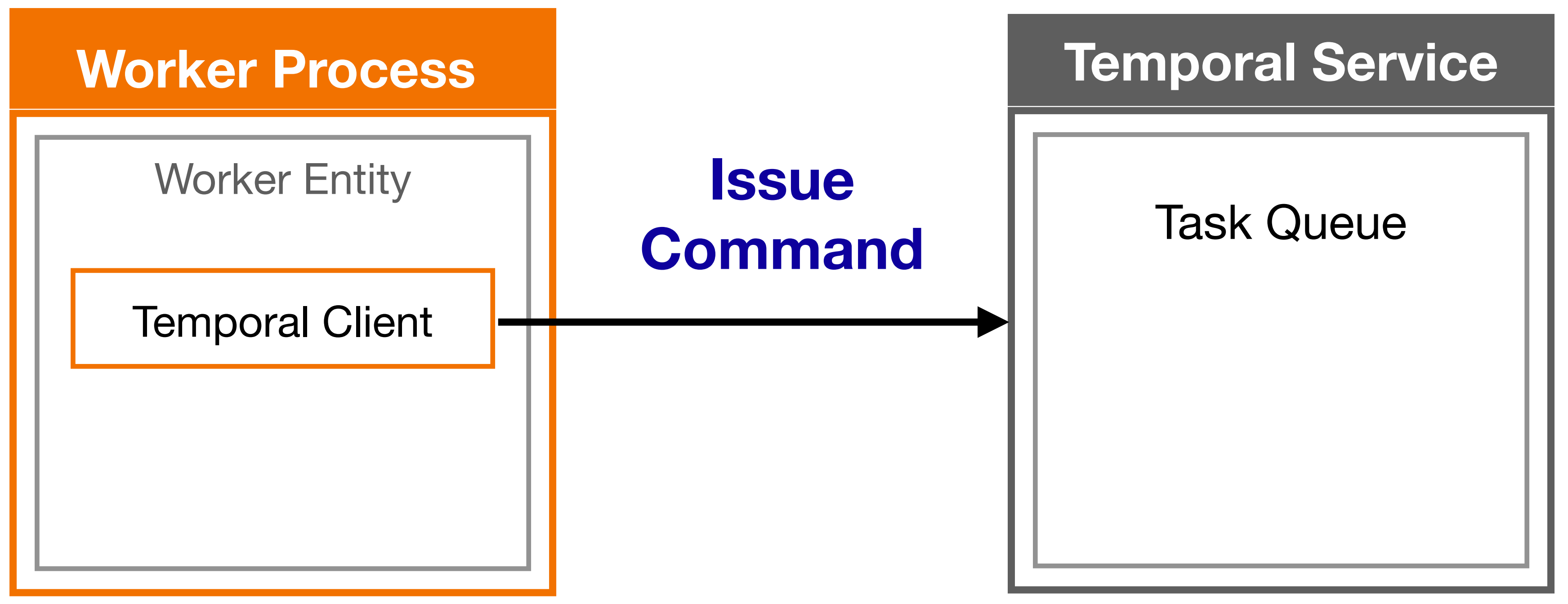


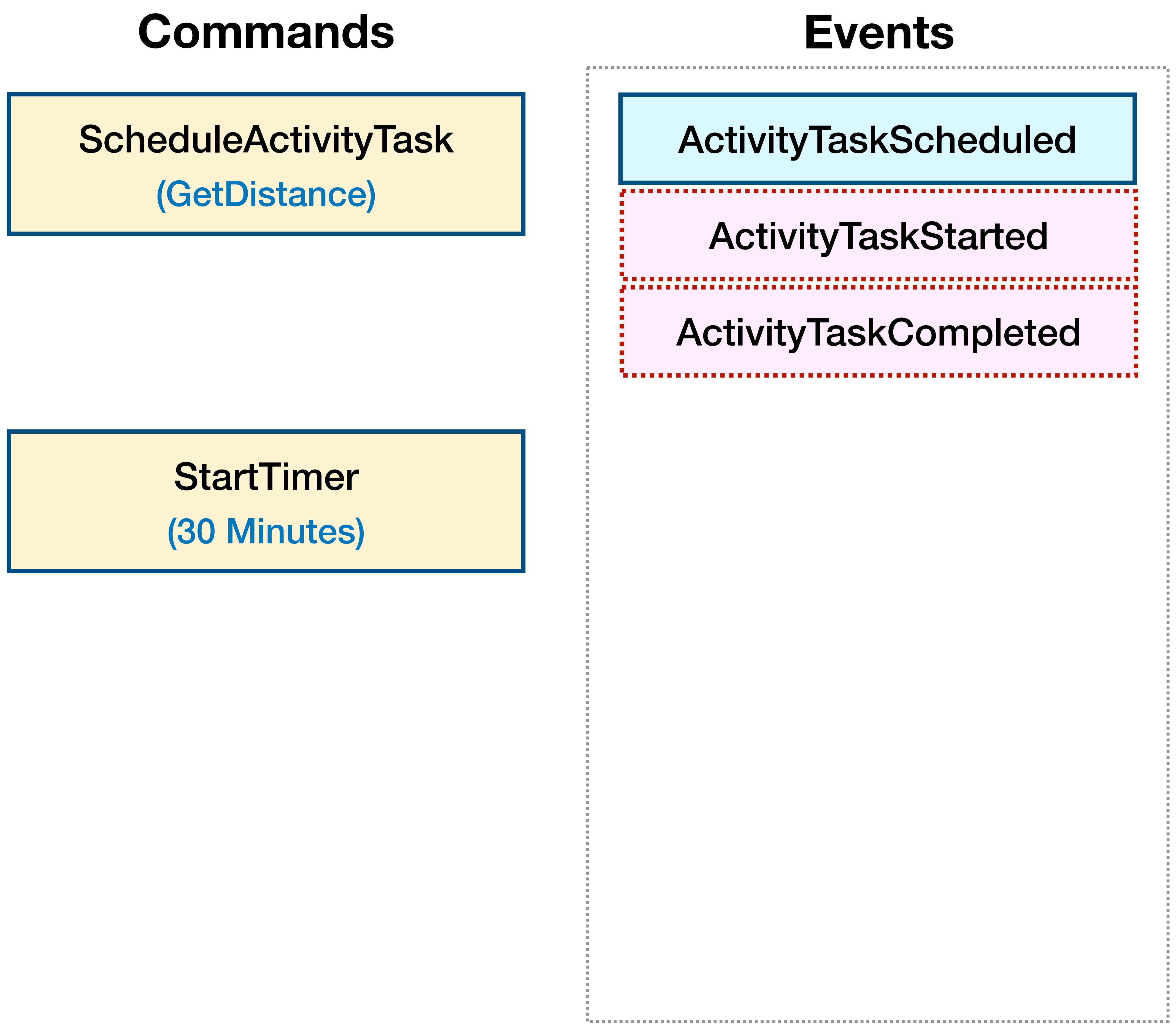
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



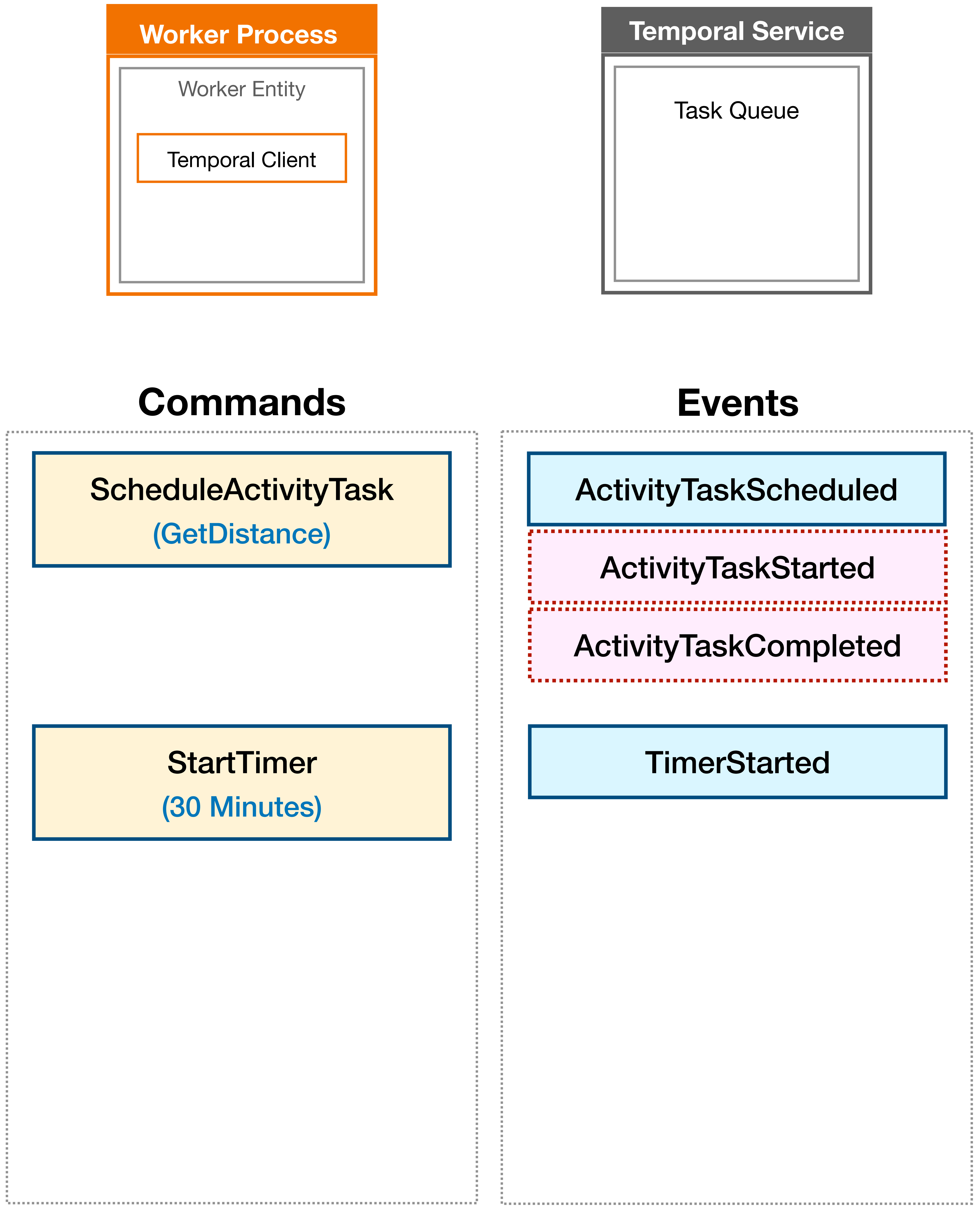


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

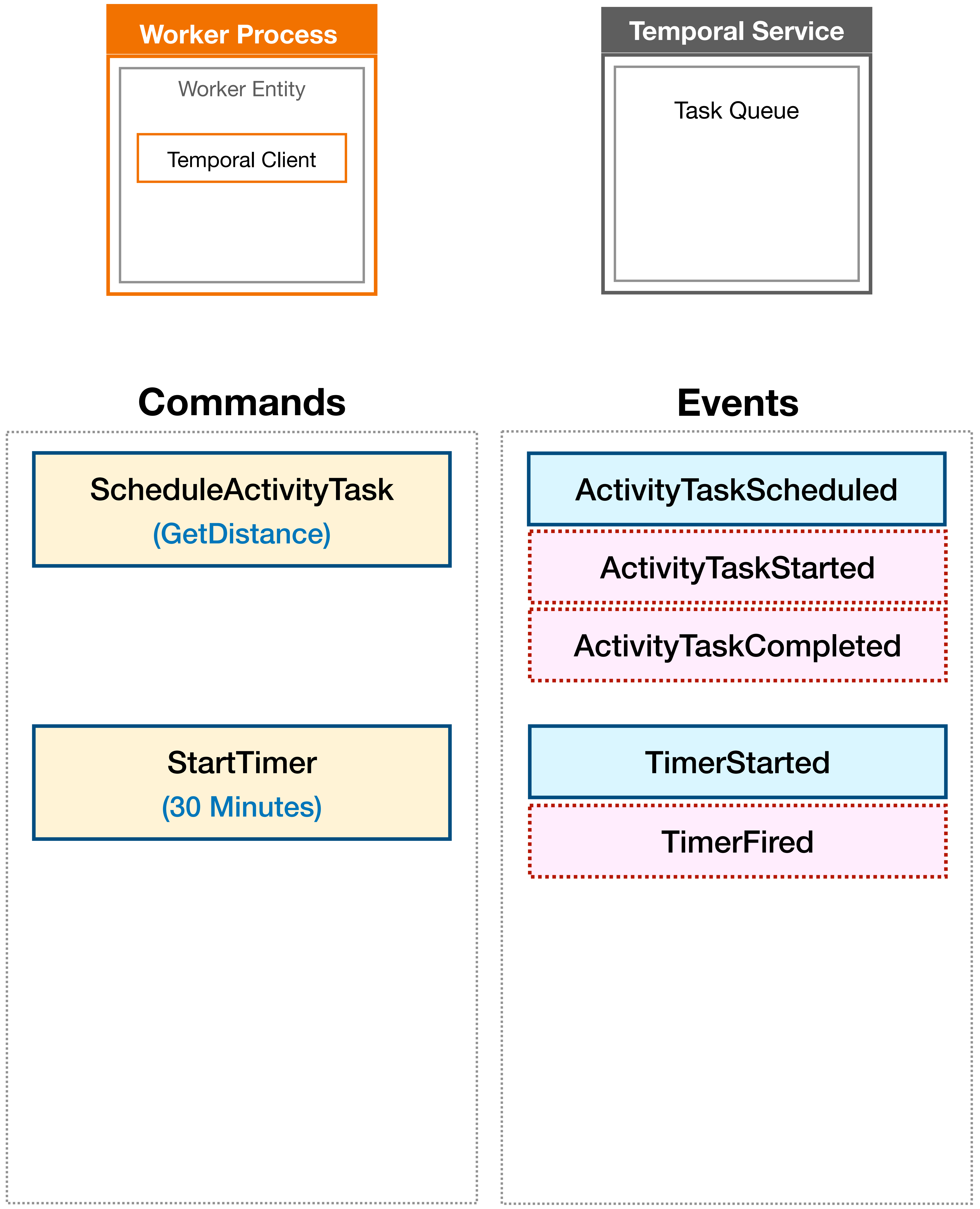




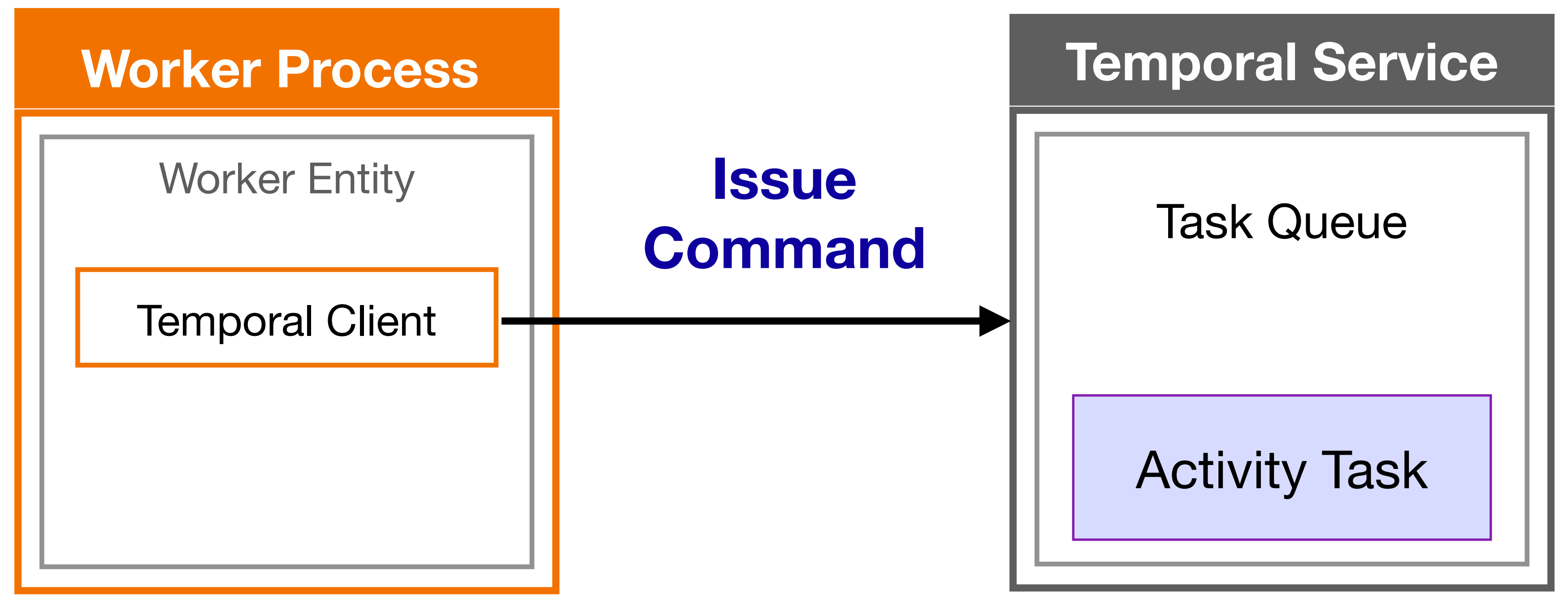
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

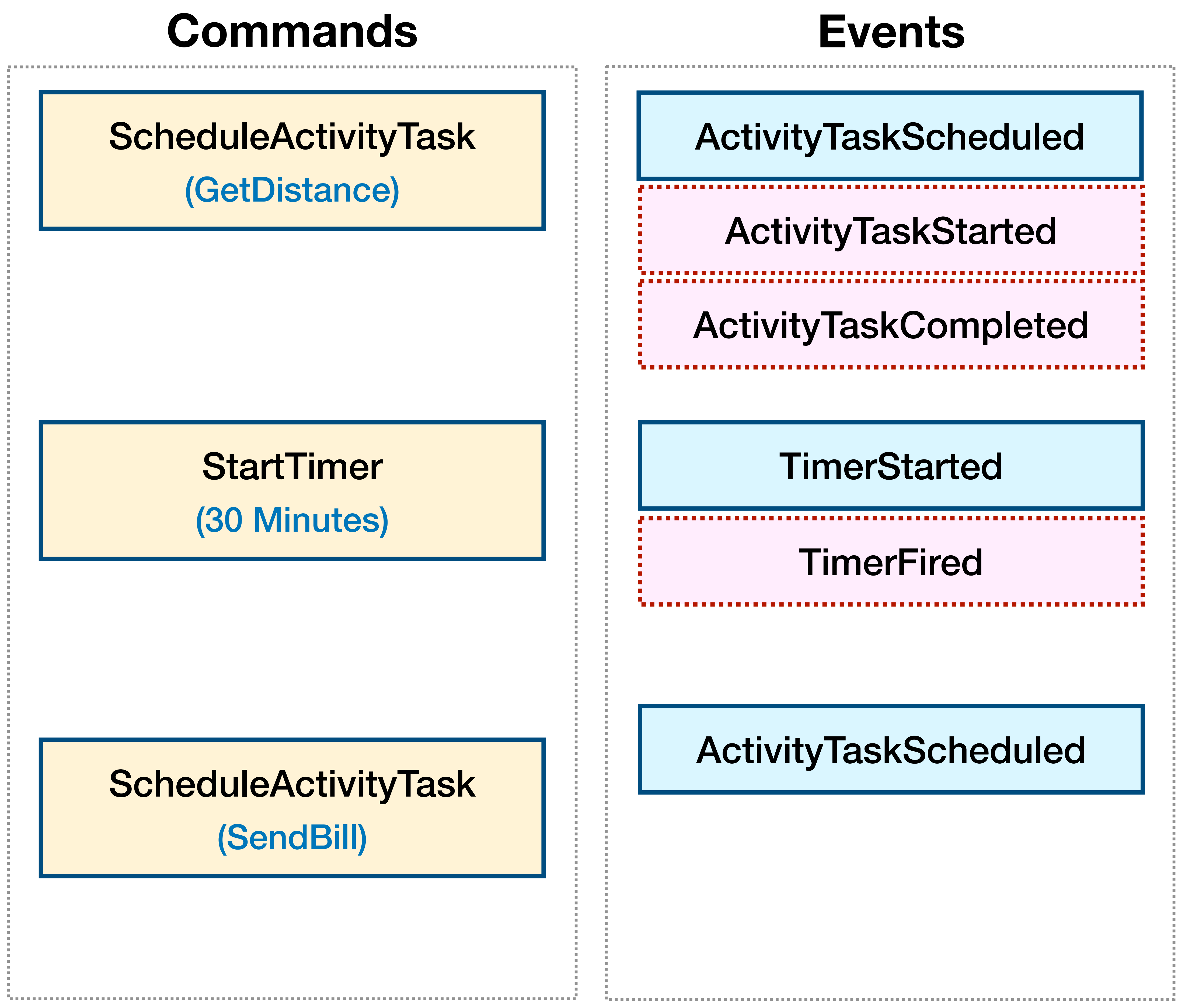


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

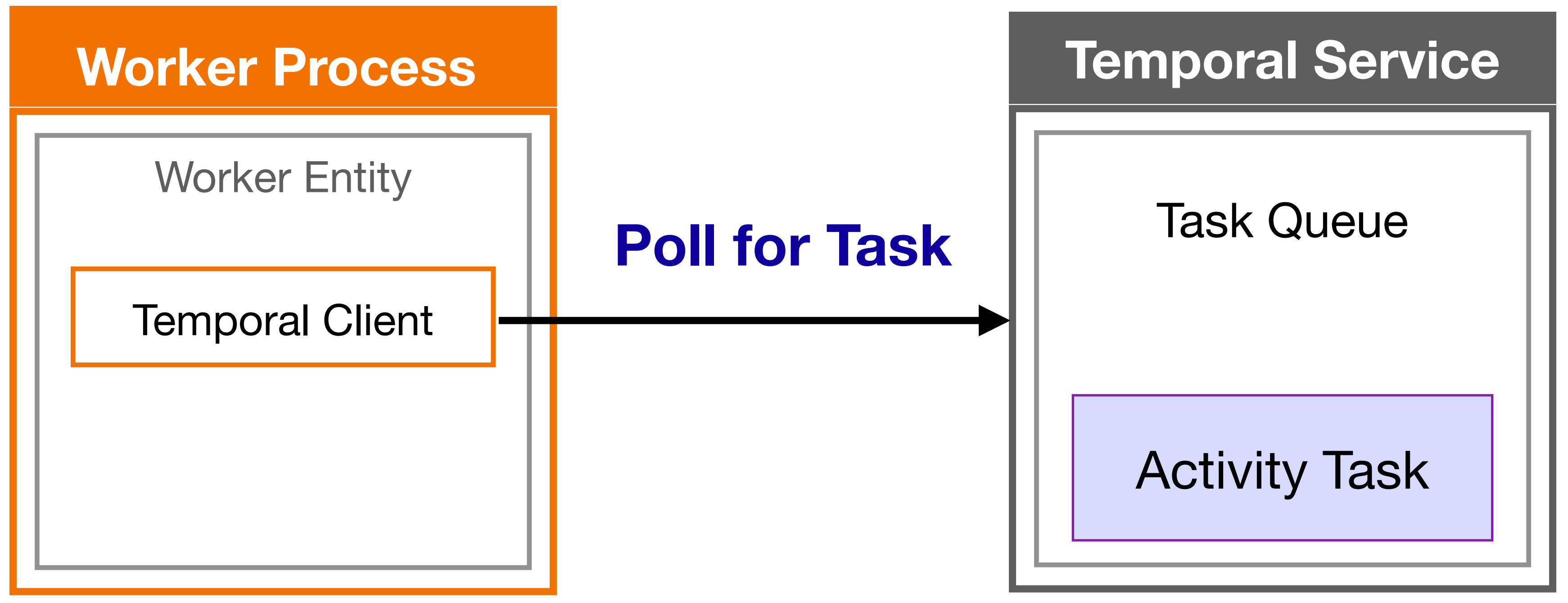


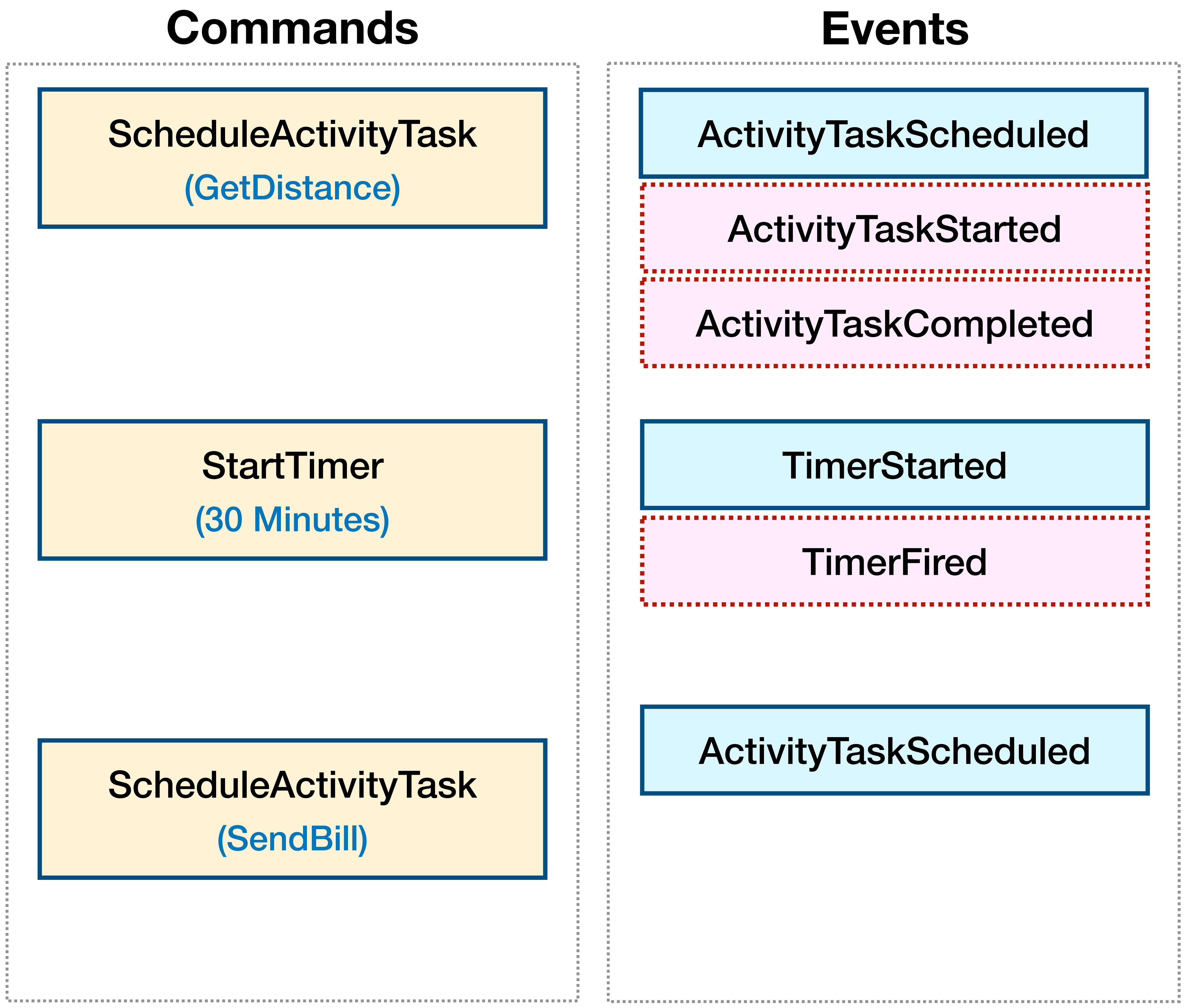
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



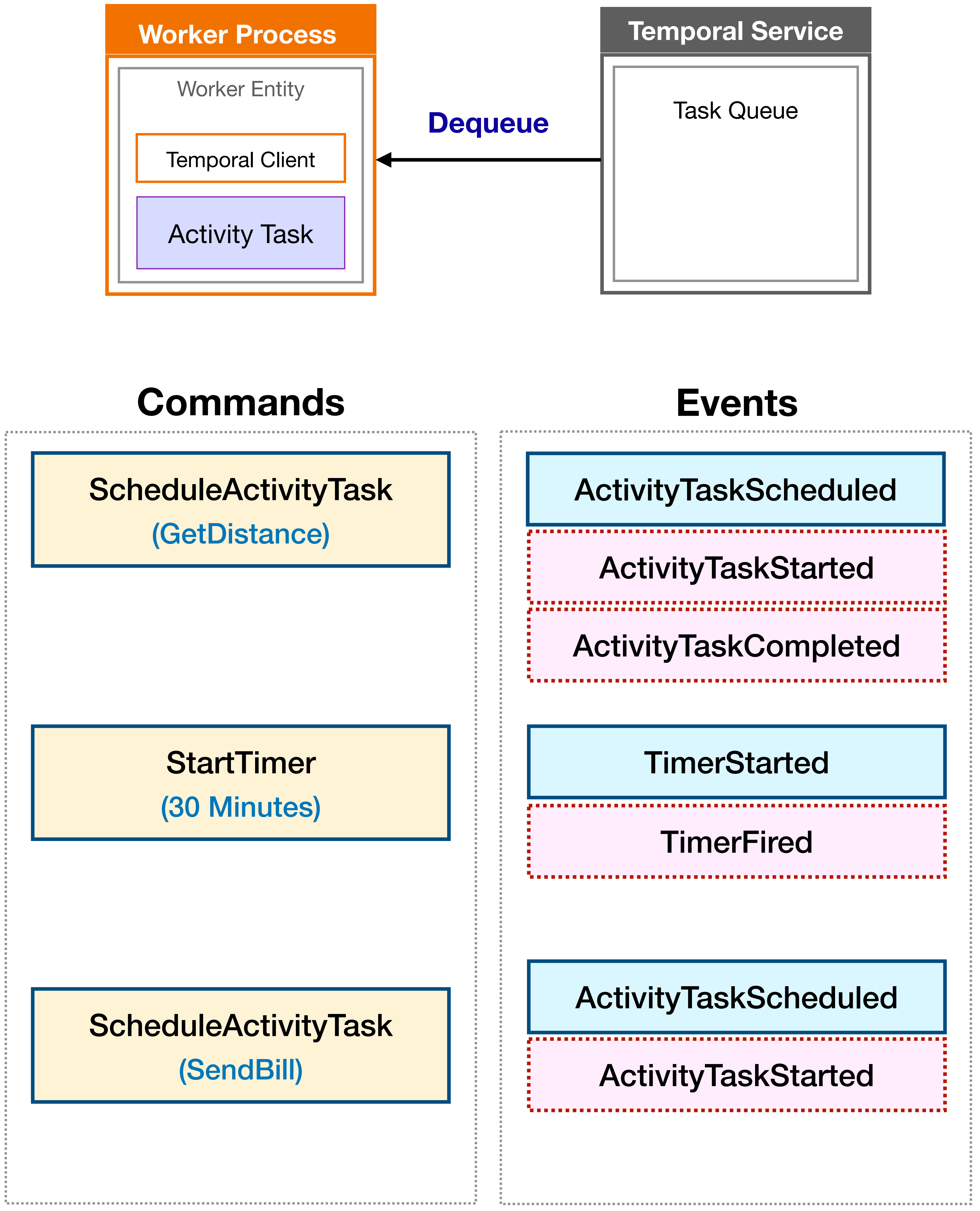


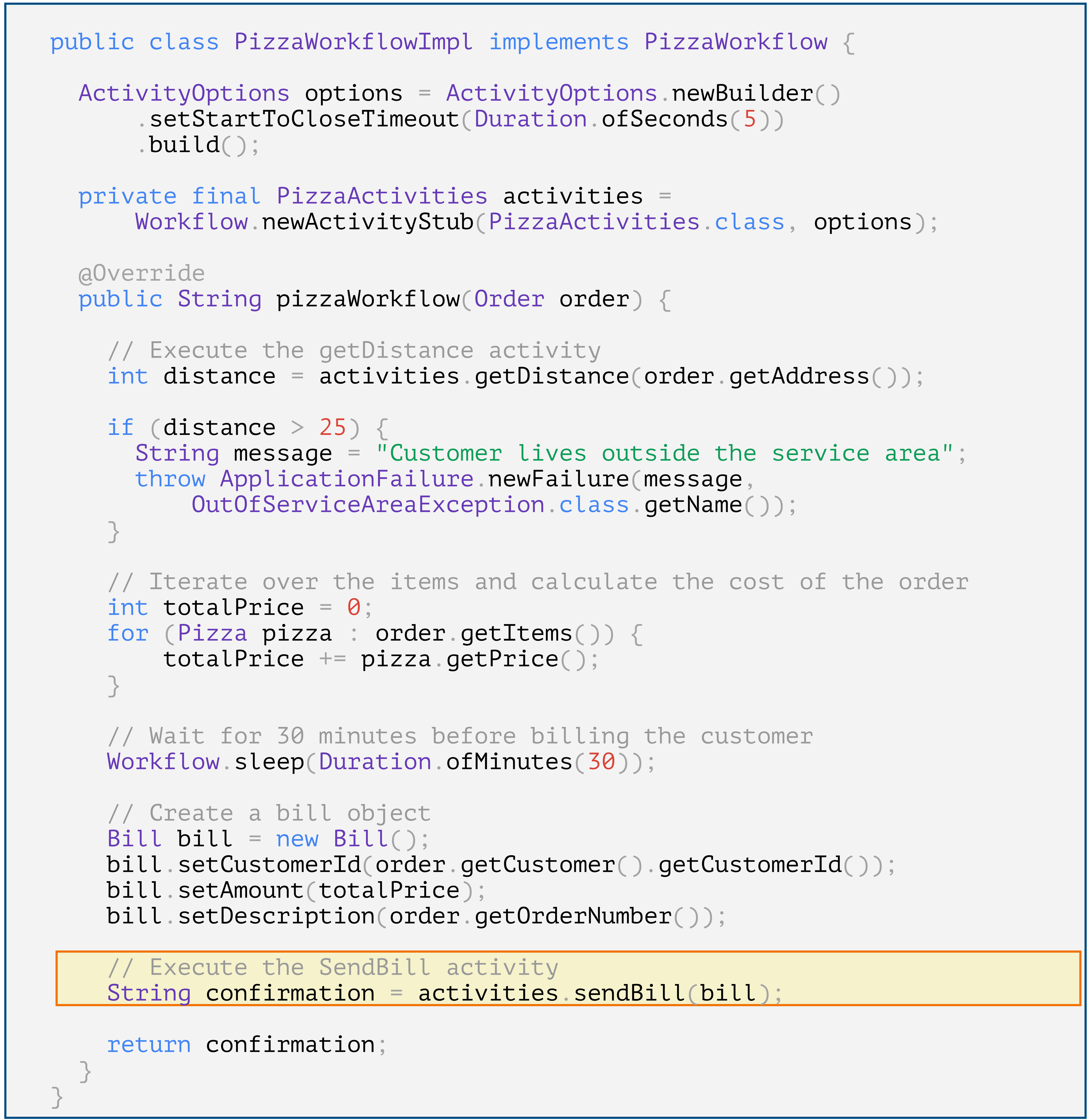
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



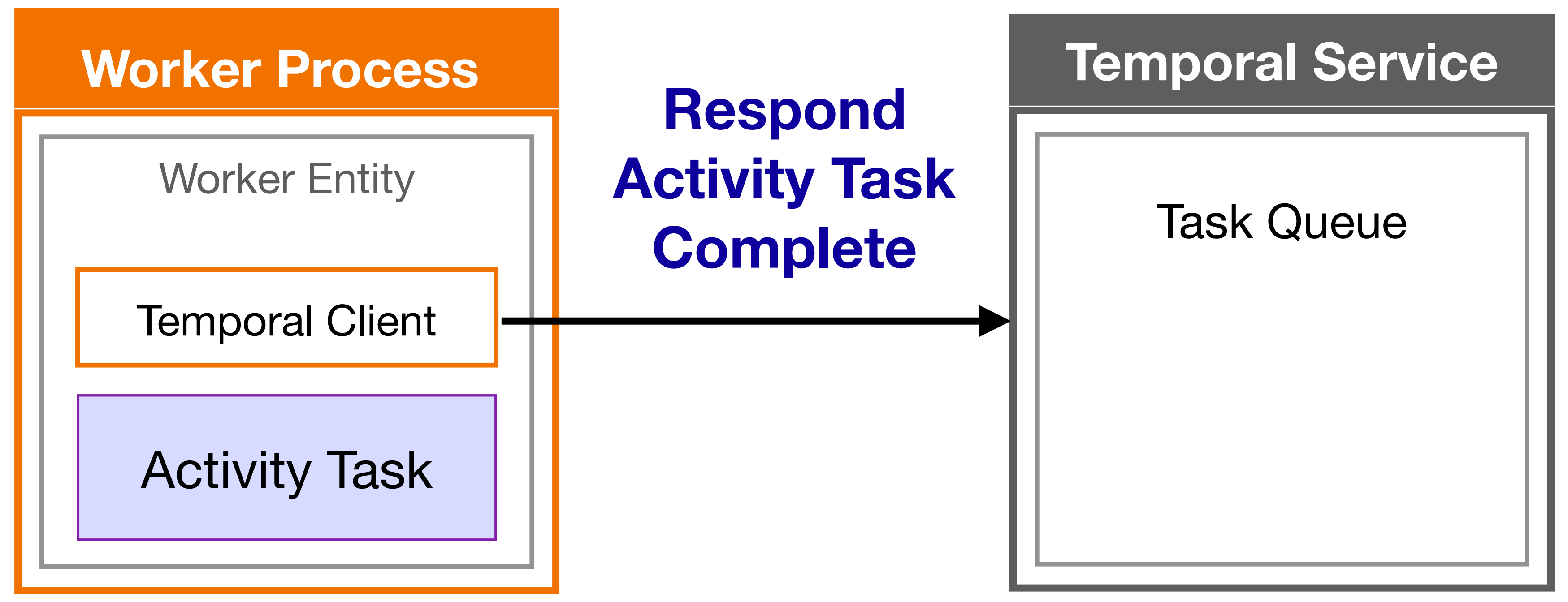


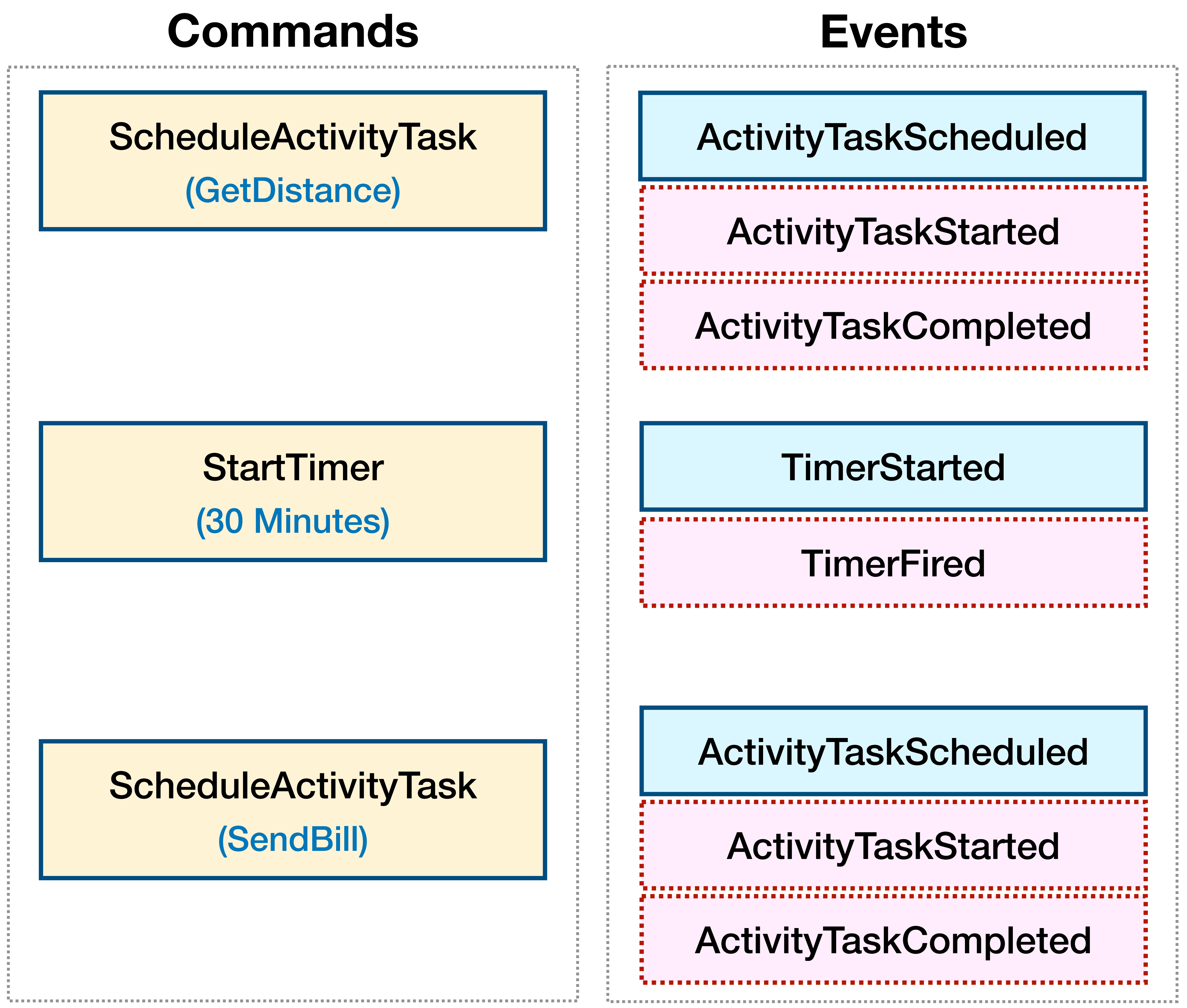
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



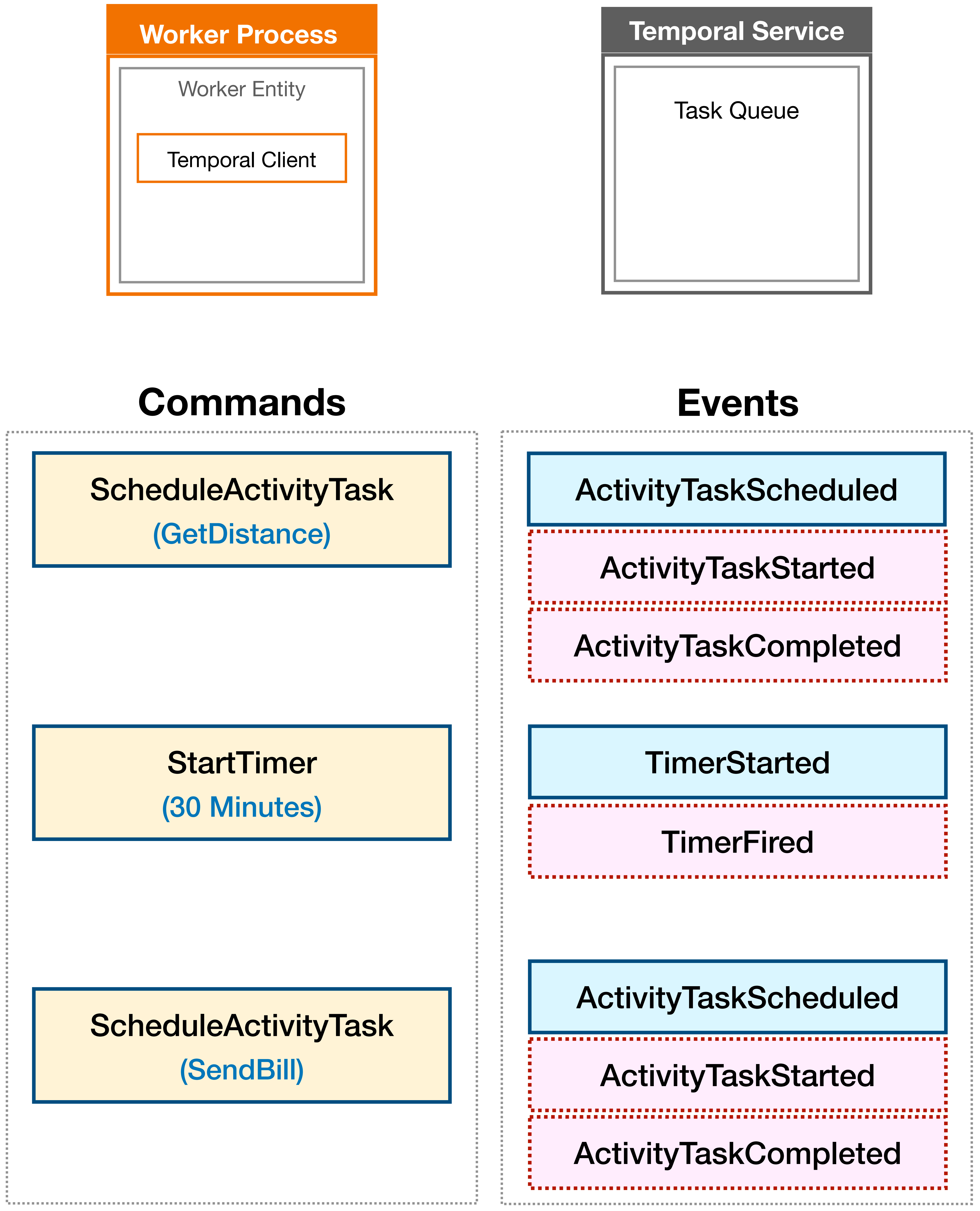


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Workflow and Activity Task States

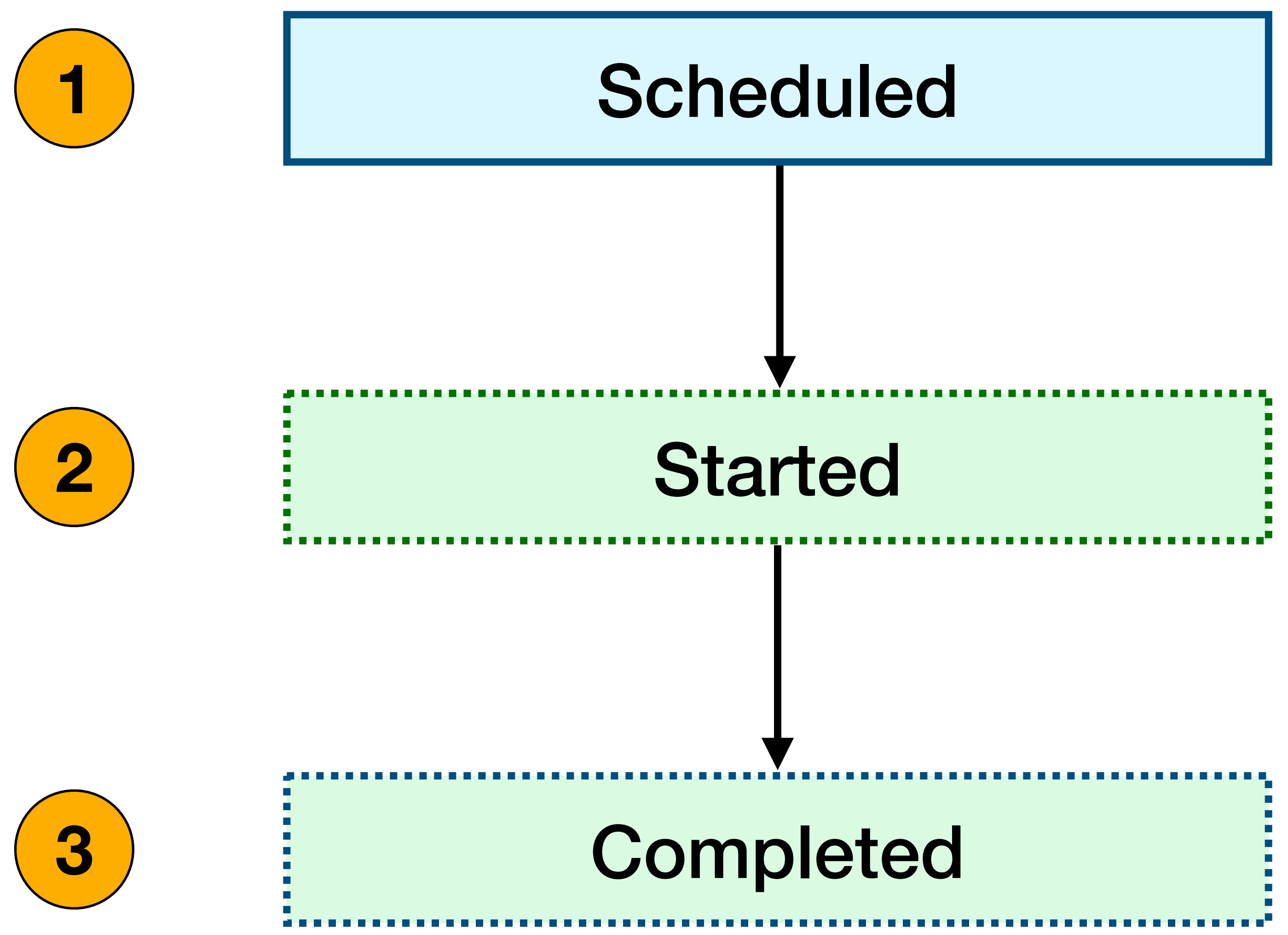
Activity Task Event Sequence

ActivityTaskScheduled

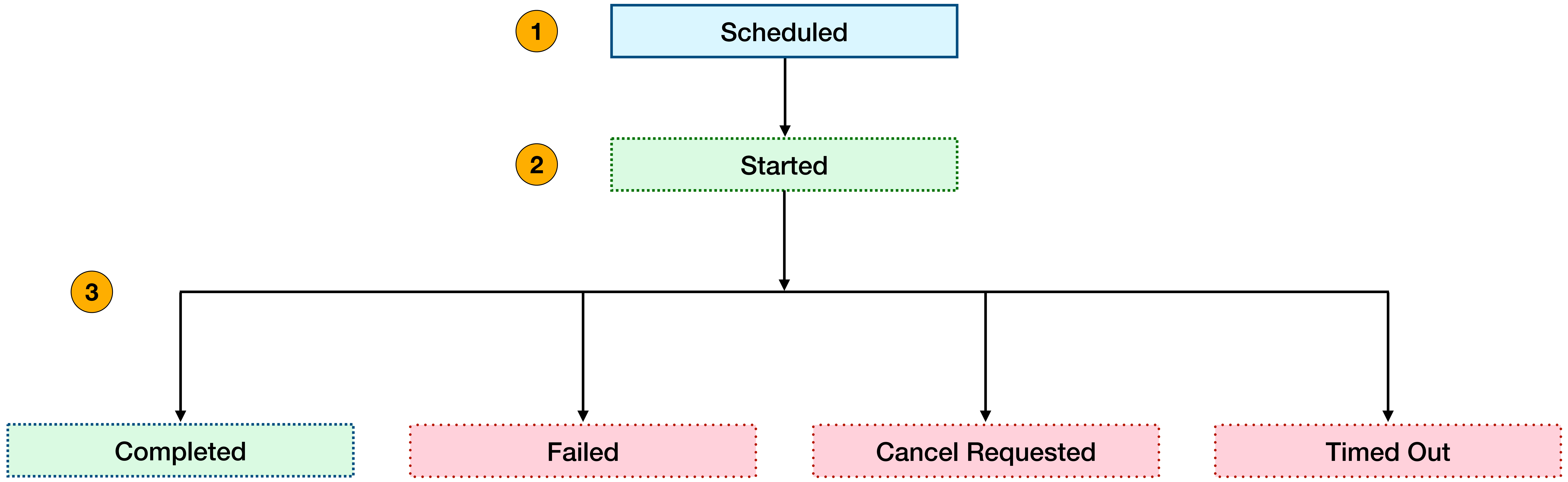
ActivityTaskStarted

ActivityTaskCompleted

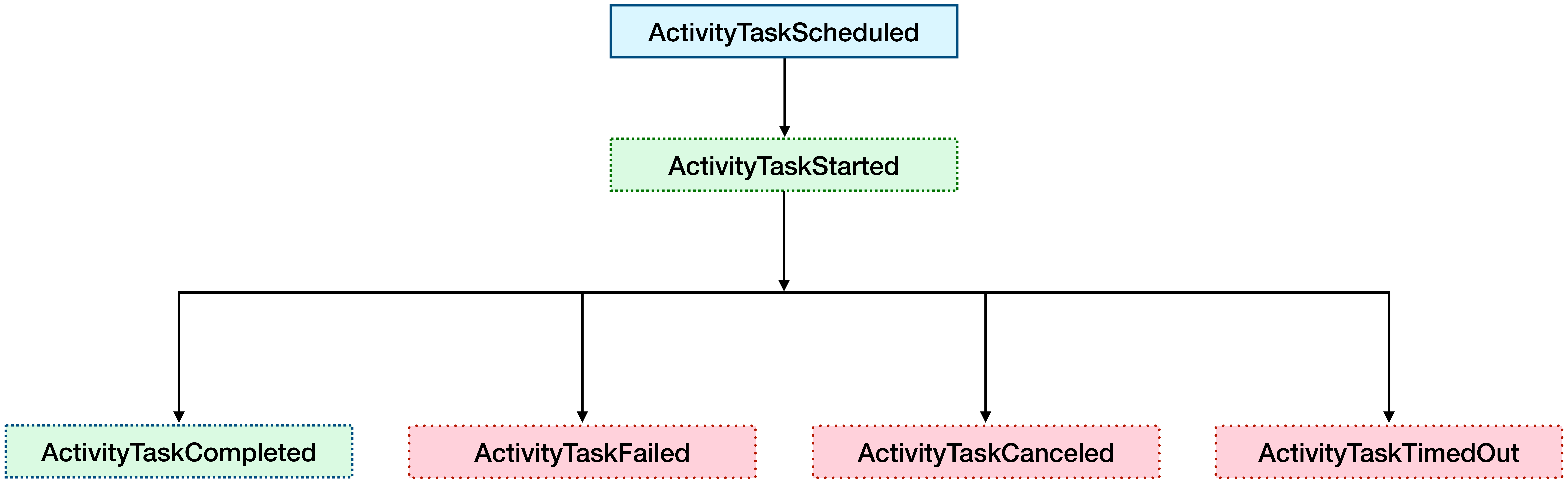
Activity States in that Sequence



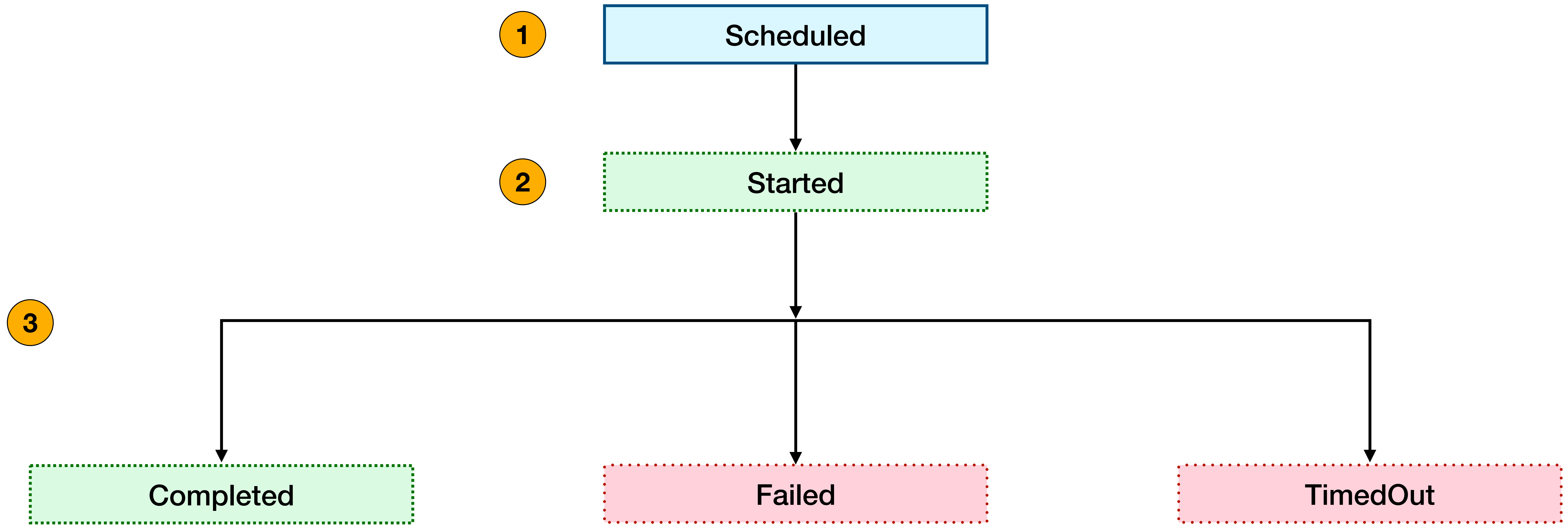
Activity Task States



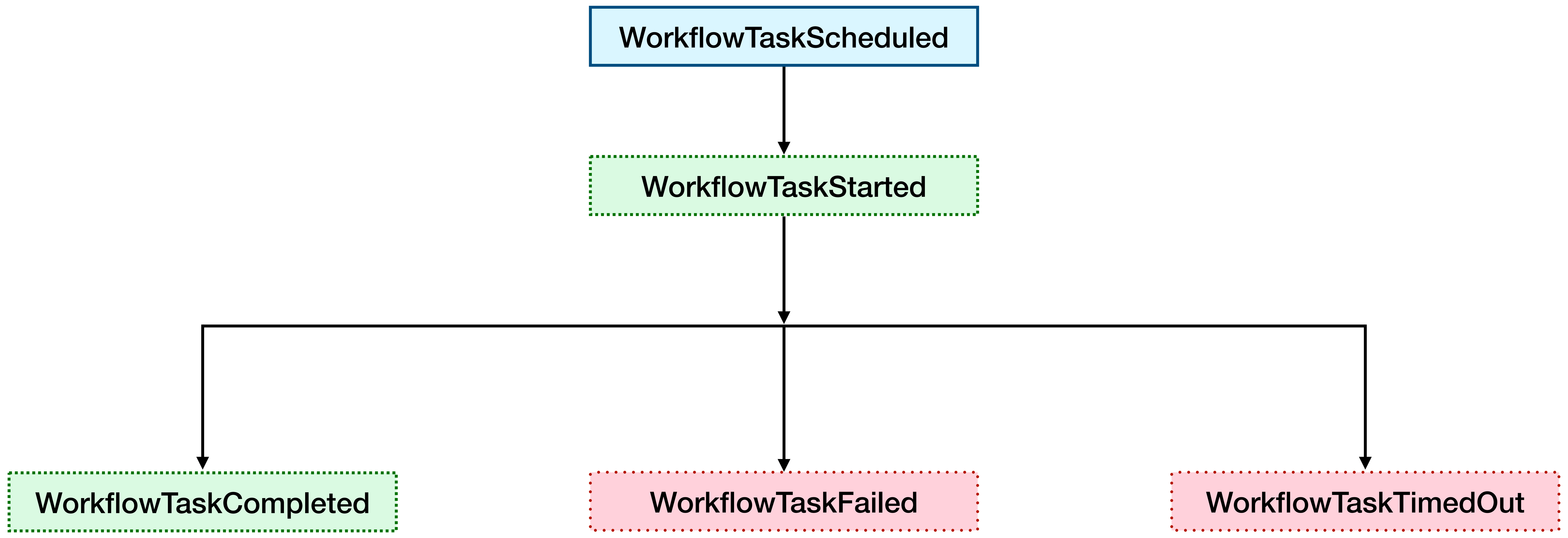
Activity Task Events



Workflow Task States



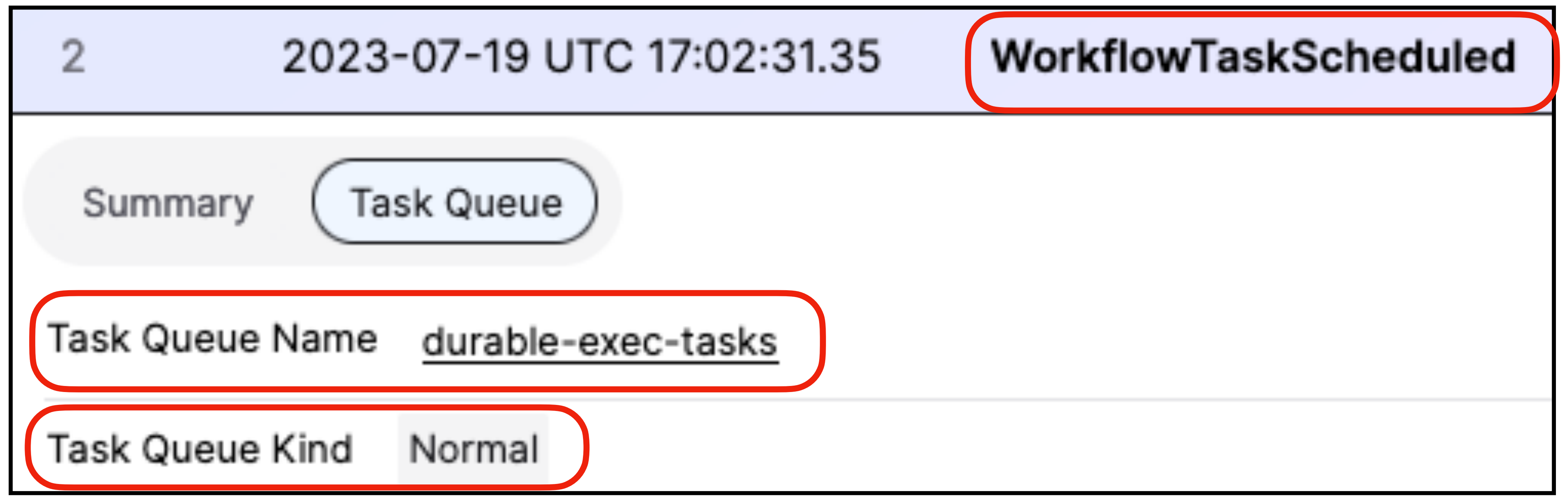
Workflow Task Events



Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
    - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
    - See the Task Queue Name / Kind fields
- **This does not apply to Activity Tasks**

First Workflow Task

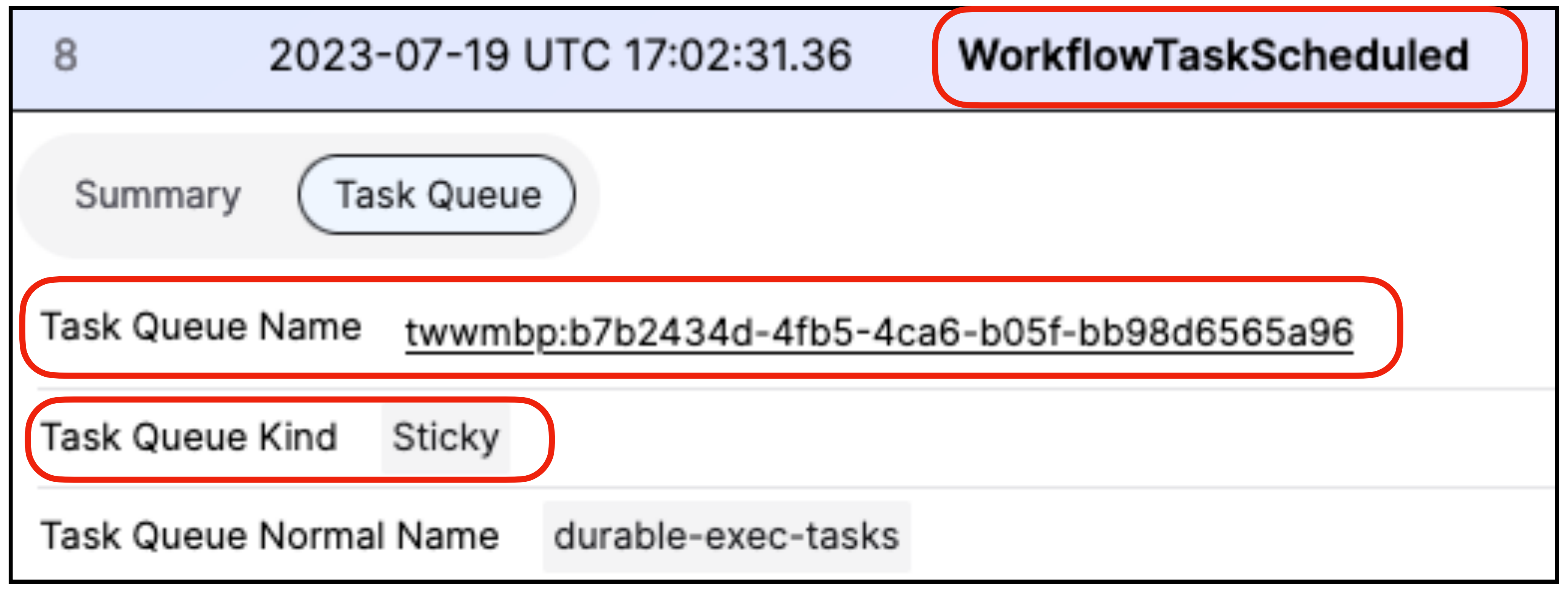


2 2023-07-19 UTC 17:02:31.35 WorkflowTaskScheduled

Task Queue Name durable-exec-tasks

Task Queue Kind Normal

Later Workflow Task



8 2023-07-19 UTC 17:02:31.36 WorkflowTaskScheduled

Task Queue Name twwmbp:b7b2434d-4fb5-4ca6-b05f-bb98d6565a96

Task Queue Kind Sticky

Task Queue Normal Name durable-exec-tasks

Review

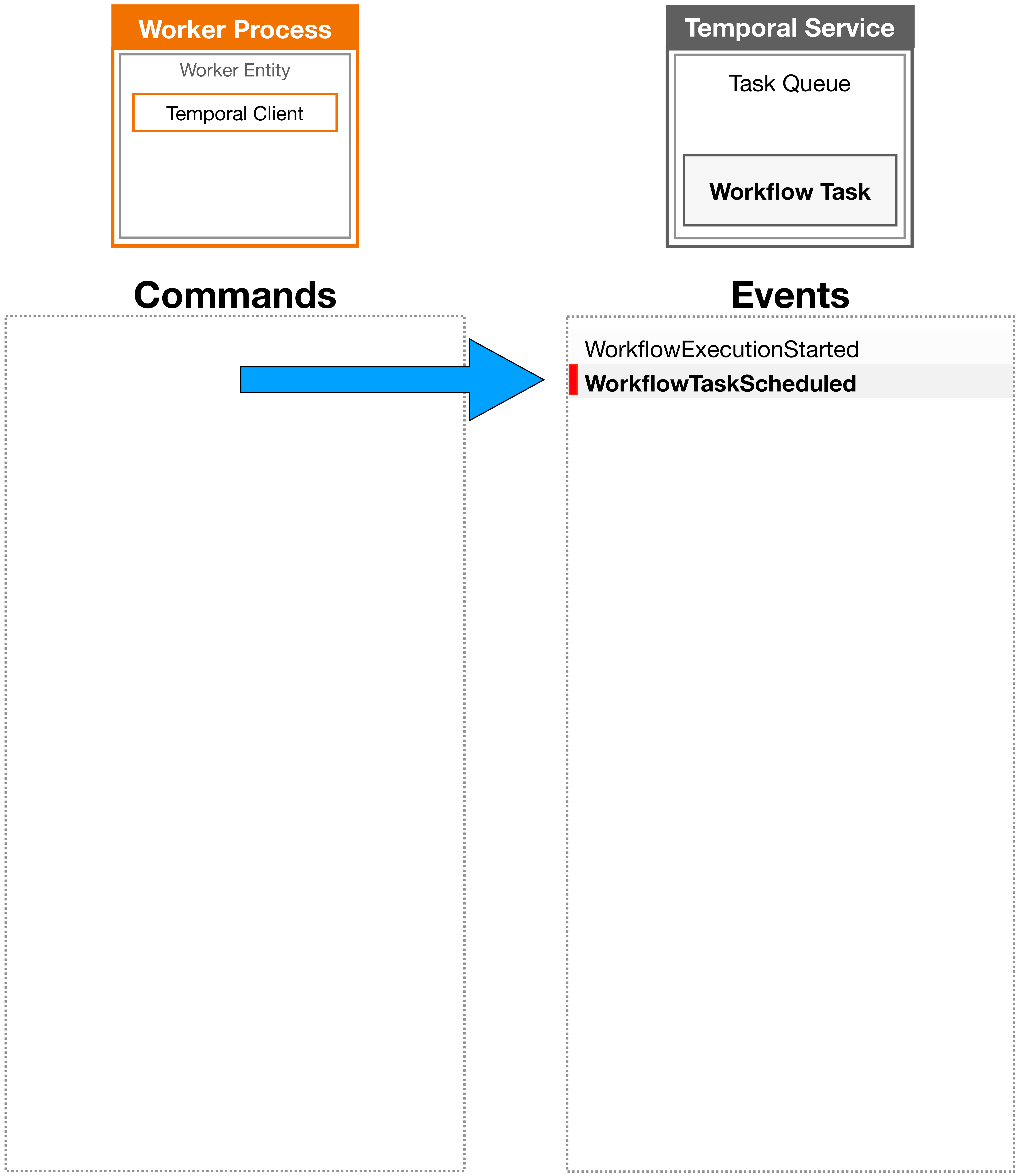
- Workflow Definition + Execution Request = Workflow Execution
- Each Workflow Execution is associated with an Event History that is the source of truth
- Executing Activities or creating Timers issues Commands to the Temporal Service, which creates Tasks, and adds Events to the Event History.
- Workflow Execution States can be Open or Closed
    - Closed means Completed, Continue-As-New, Failed, Timed Out, Cancelled, or Terminated
- Workflow and Activity Tasks can be Scheduled, Started, or Completed. They can also fail or time out.
- Sticky Execution directs Workflow Tasks to the same Worker that accepted them earlier in the same Workflow Execution

Temporal 102

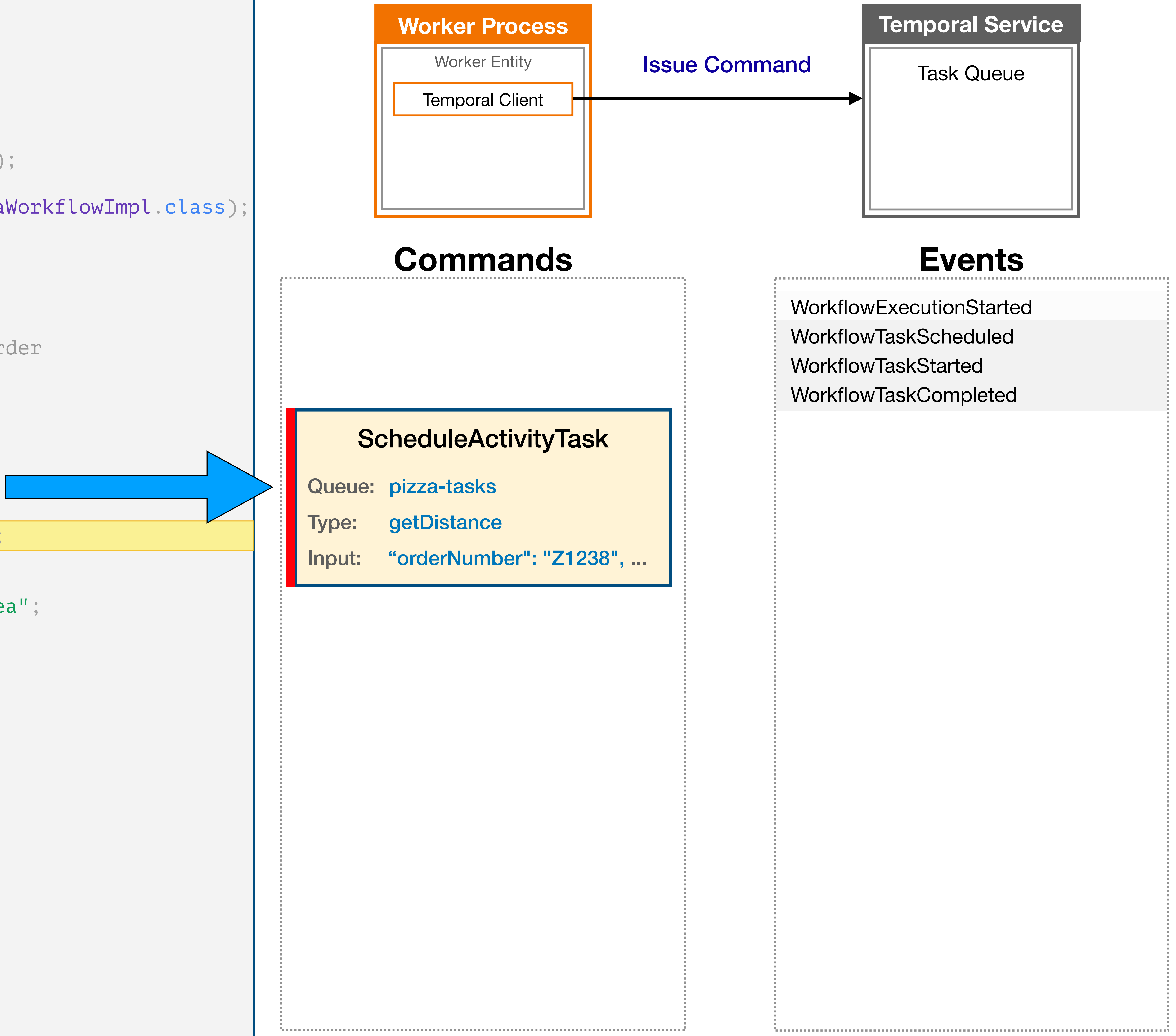
History Replay:

How Temporal Provides Durable Execution

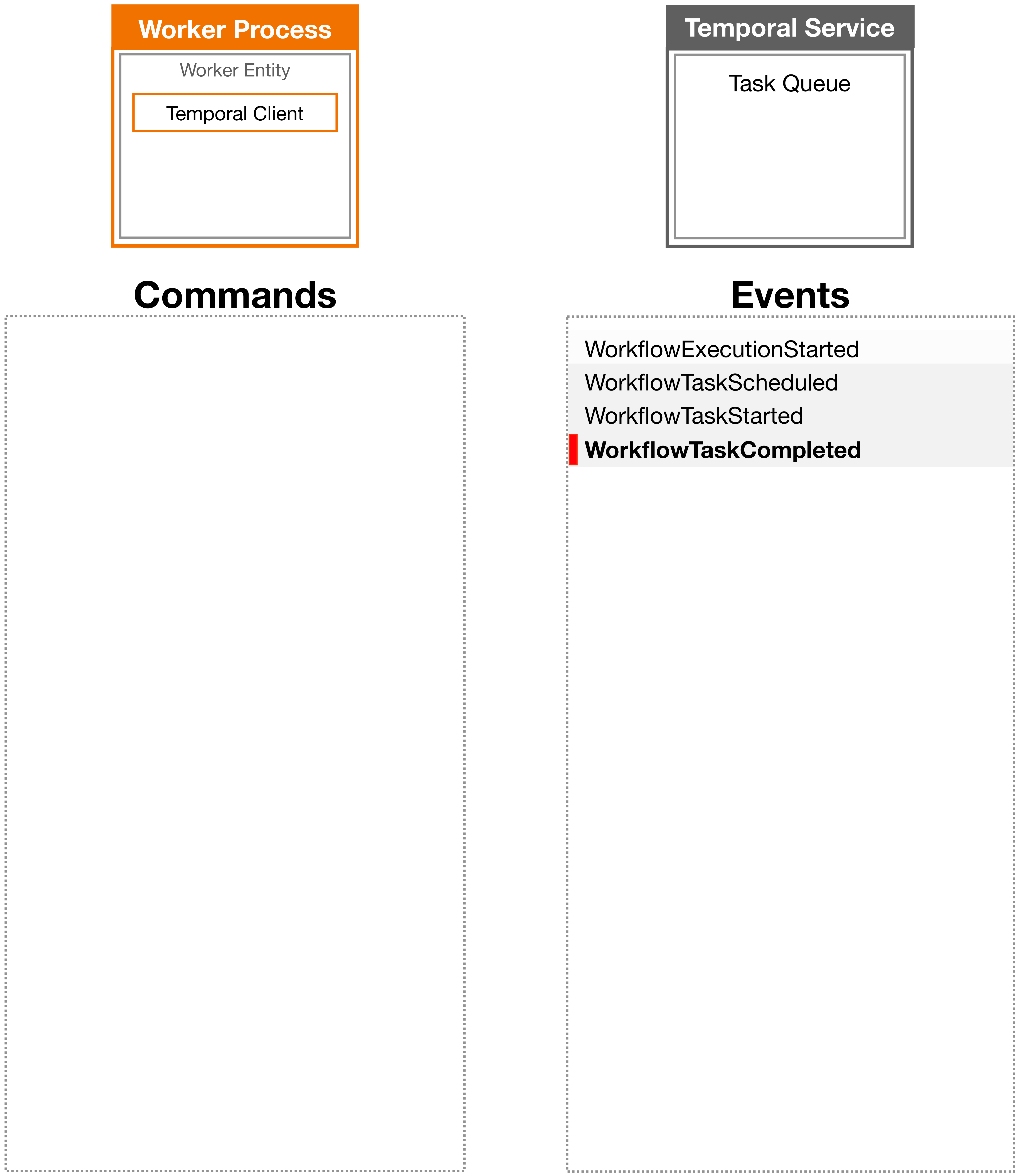
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



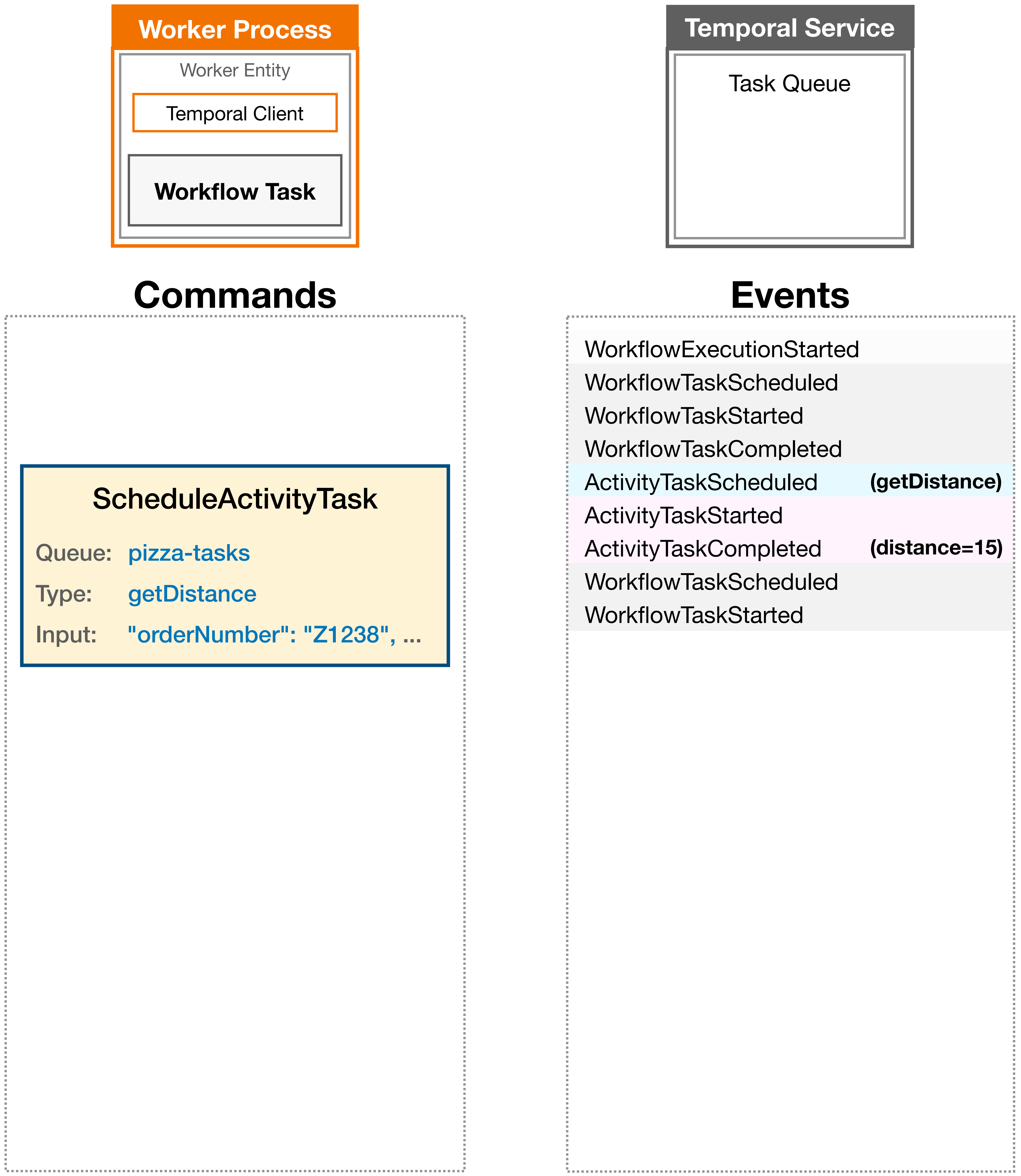
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

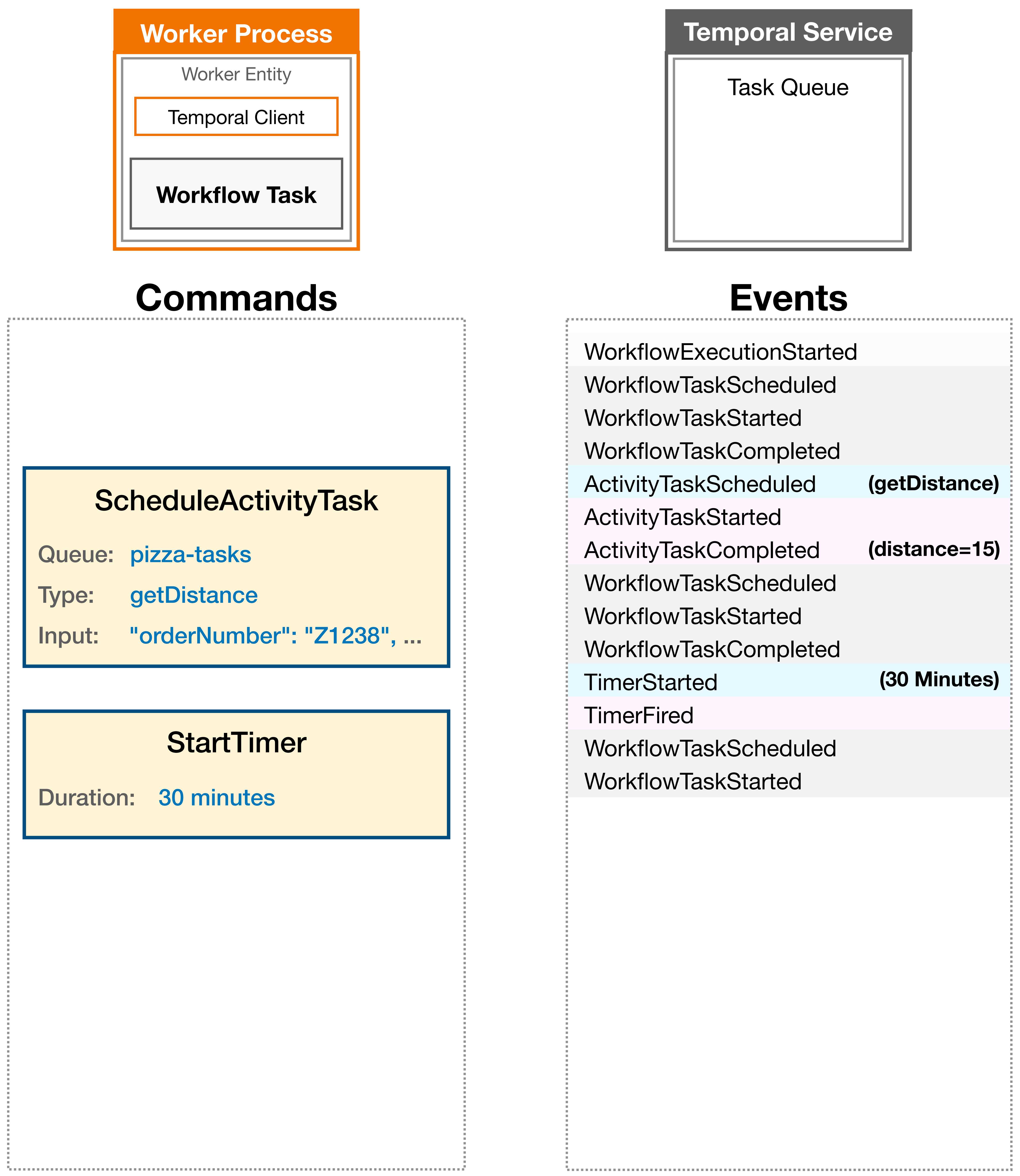


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker crashes here



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Start Workflow Execution

```java
String result = workflow.pizzaWorkflow(input);
```

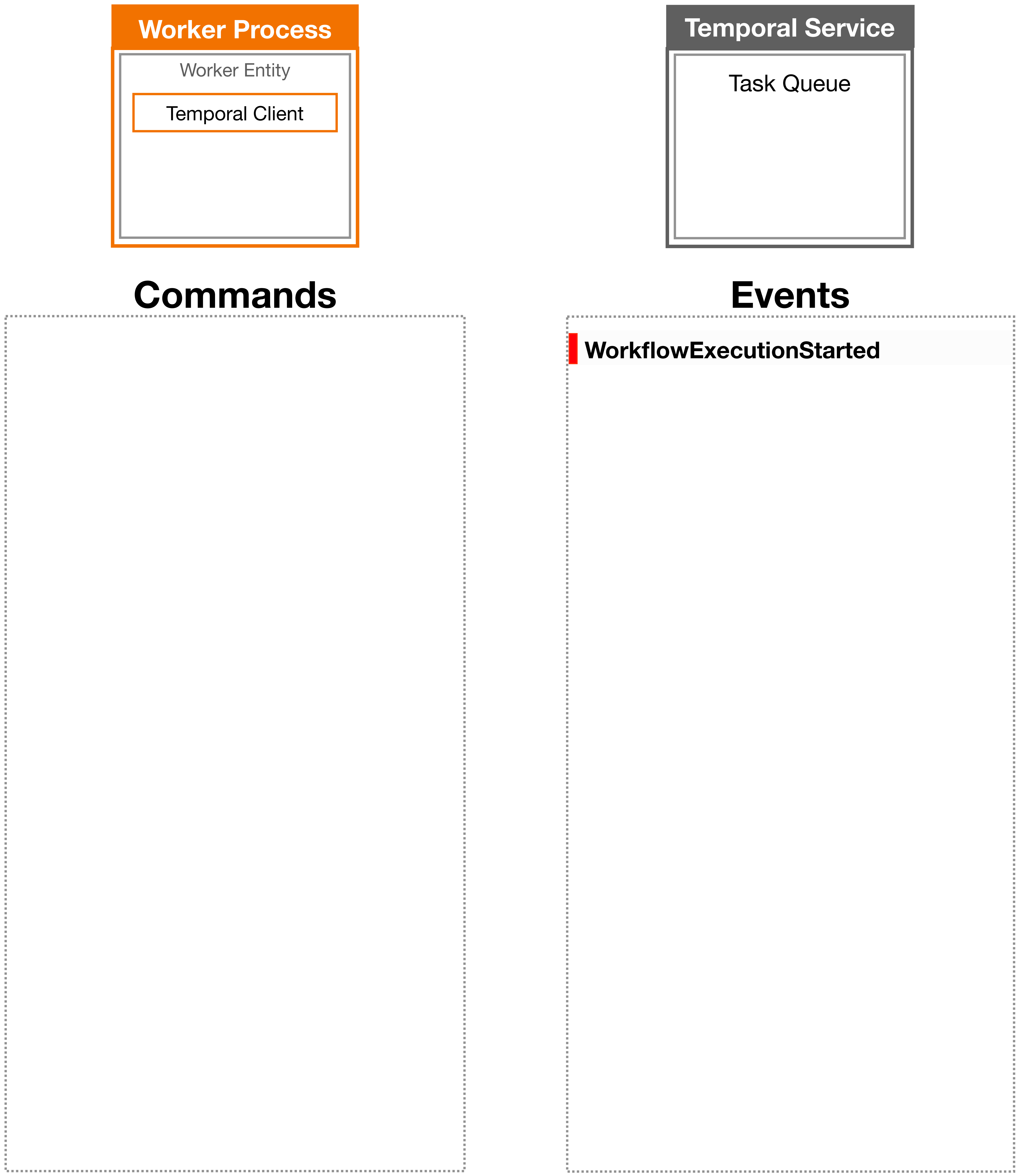
Start Workflow Execution

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

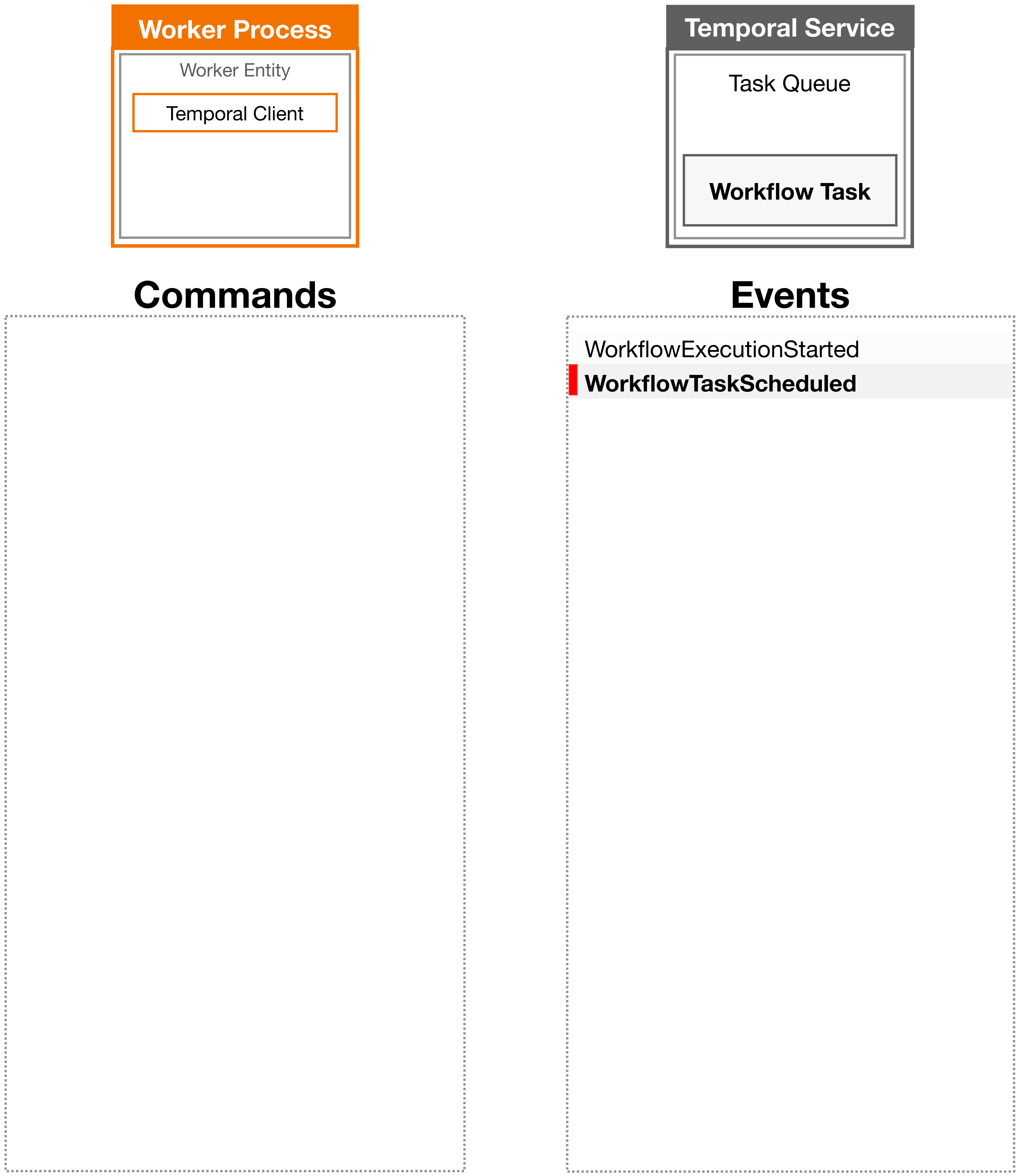
```java
String result = workflow.pizzaWorkflow(input);
```

```json
[
    {
      "orderNumber": "Z1238",
      "customer": {
        "customerID": 12983,
        "name": "María García",
        "email": "maria1985@example.com",
        "phone": "415-555-7418"
      },
      "items": [
        {
          "description": "Large, with pepperoni",
          "price": 1500
        },
        {
          "description": "Small, with mushrooms and onions",
          “price": 1000
        }
      ],
      "isDelivery": true,
      "address": {
        "line1": "701 Mission Street",
        "line2": "Apartment 9C",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94103"
      }
    }
]
```

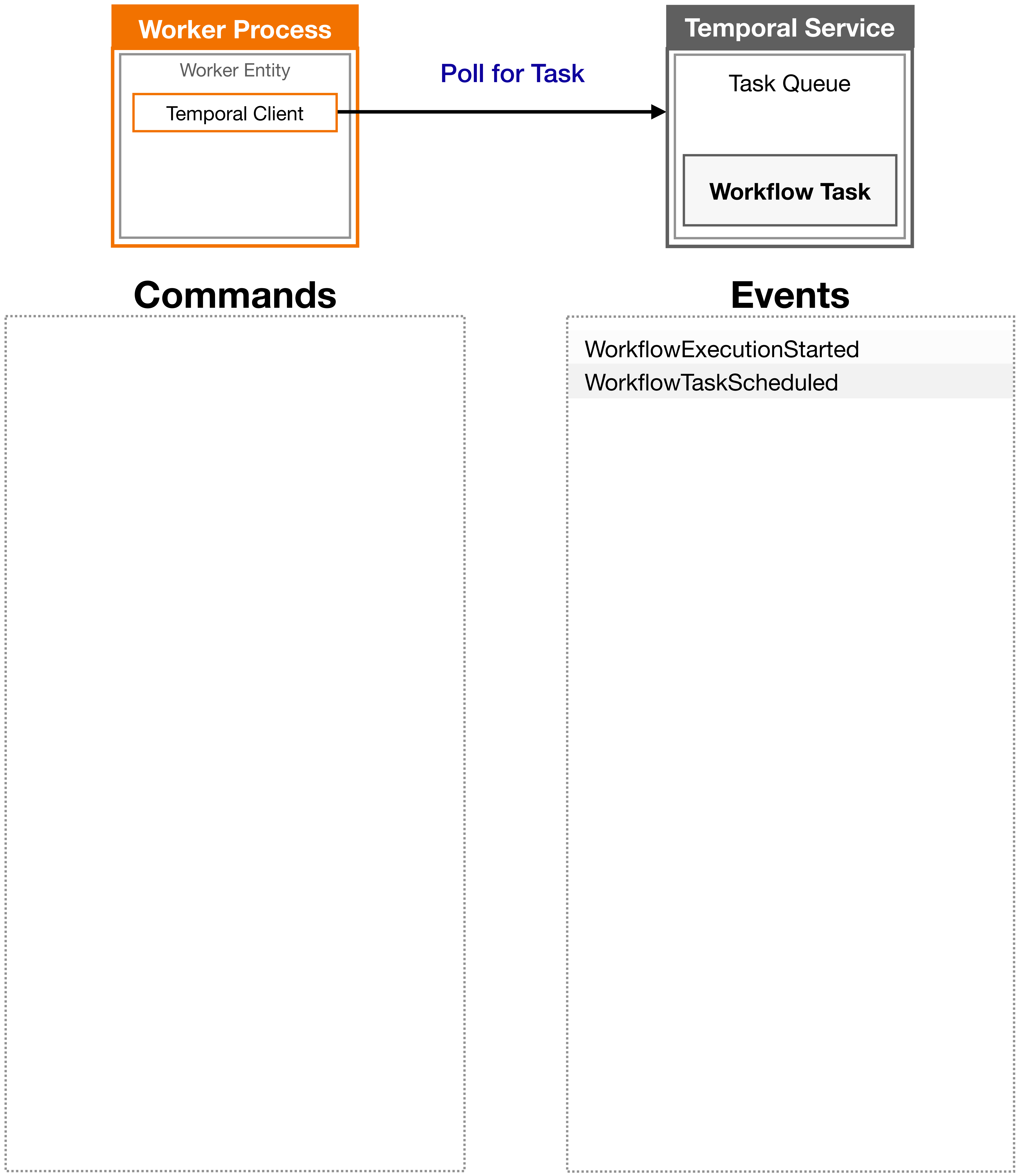
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



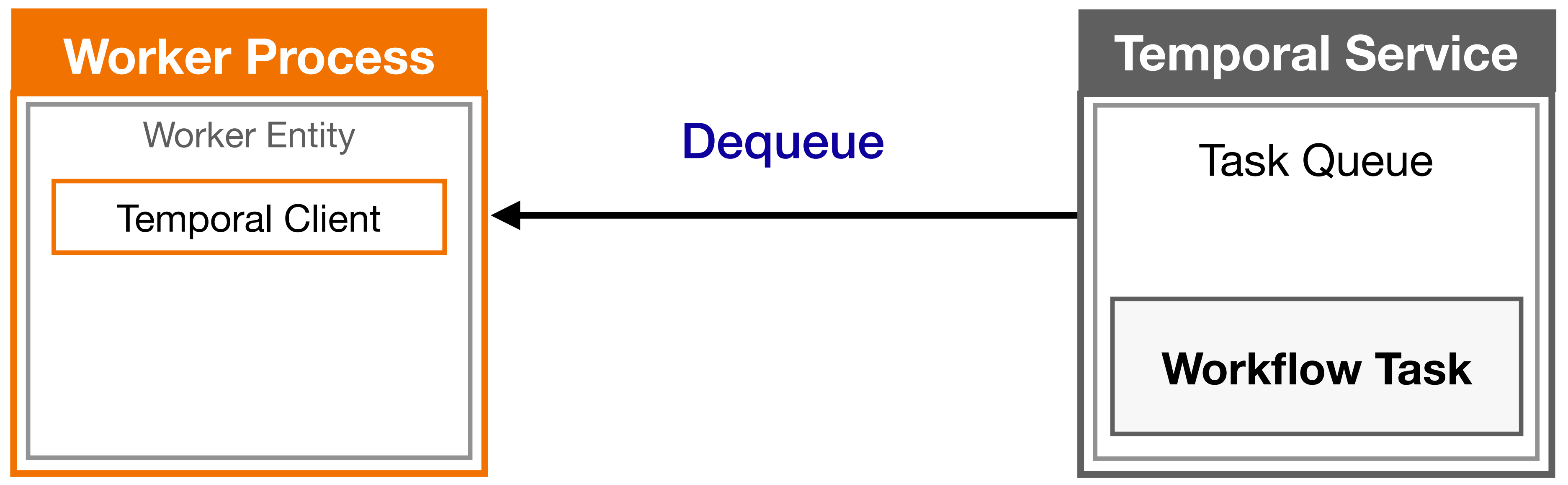
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

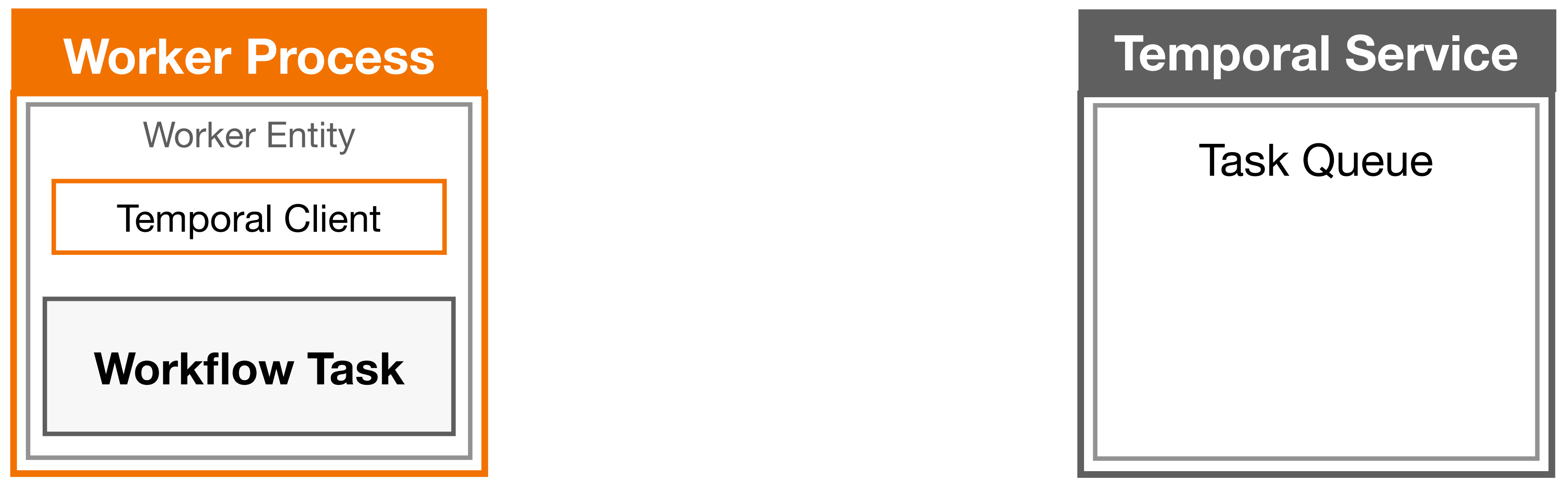


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

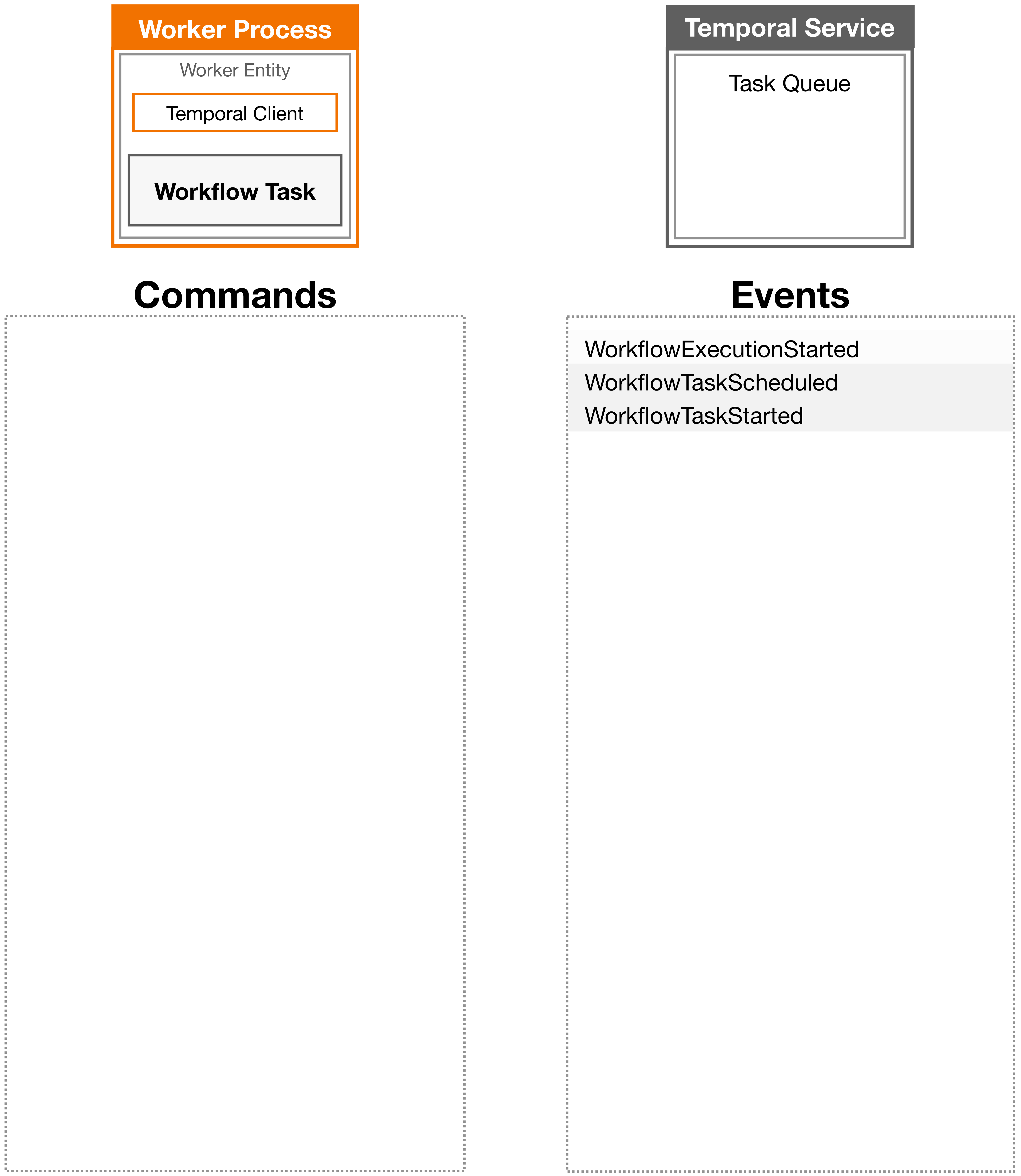


Commands

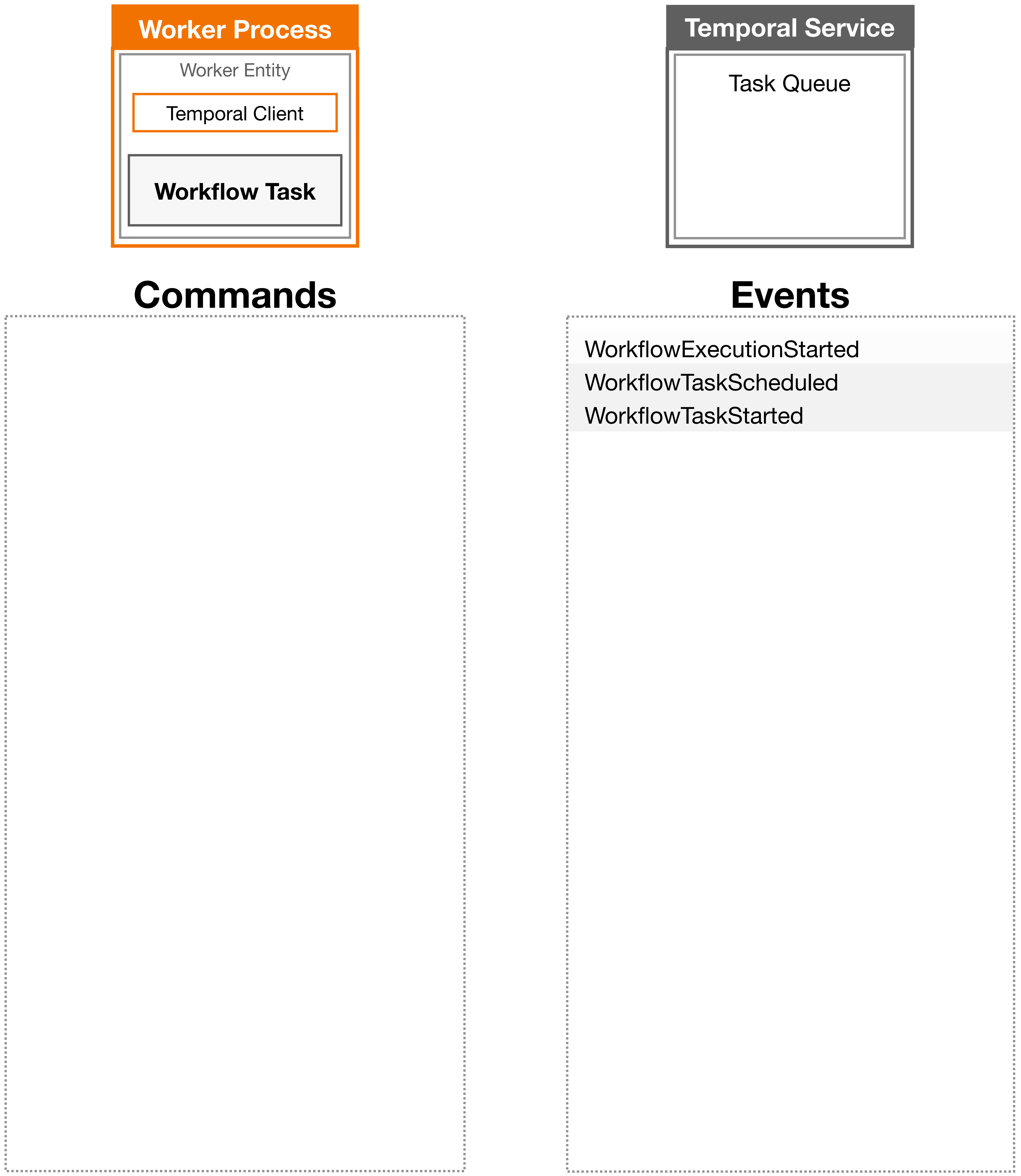
Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

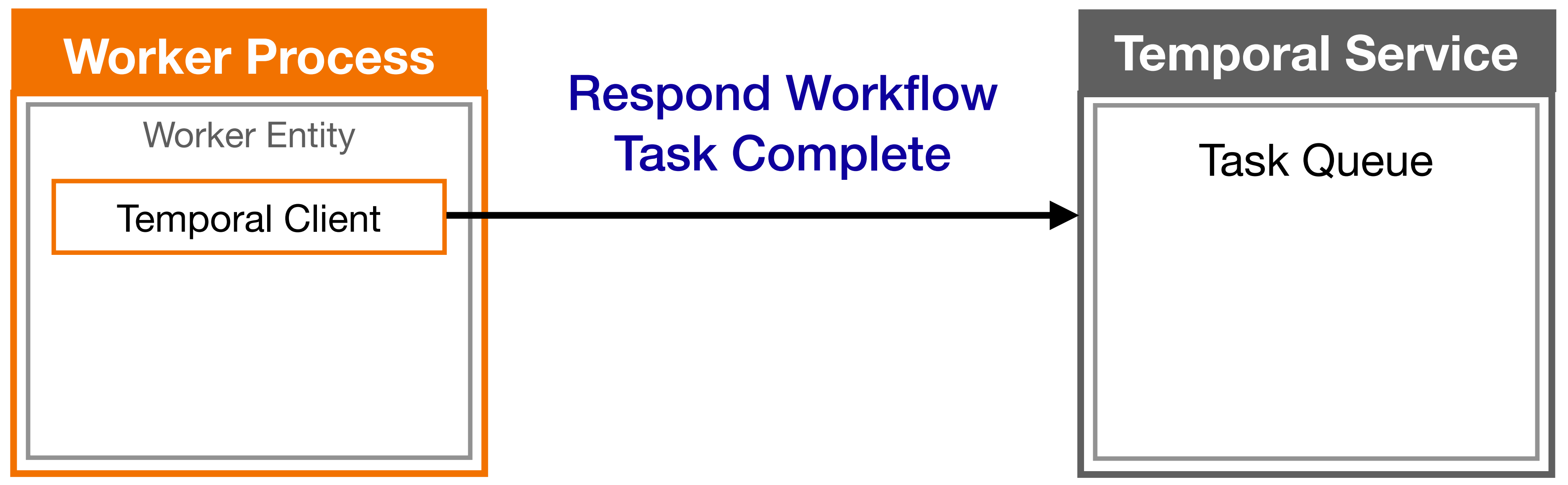
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);
    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

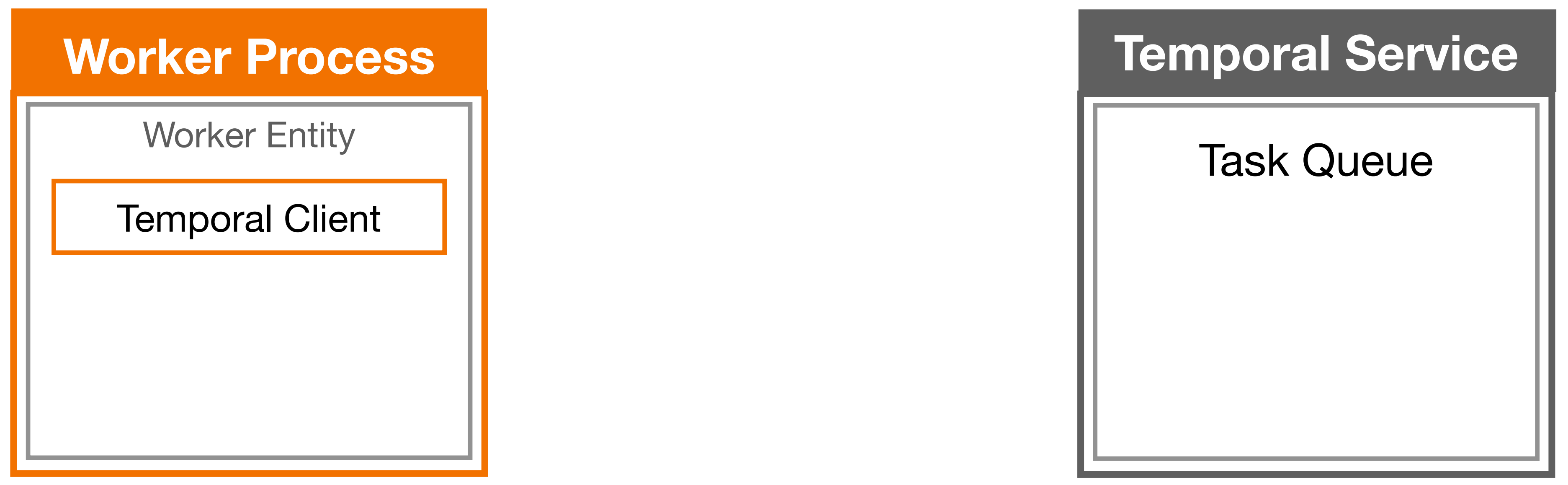


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

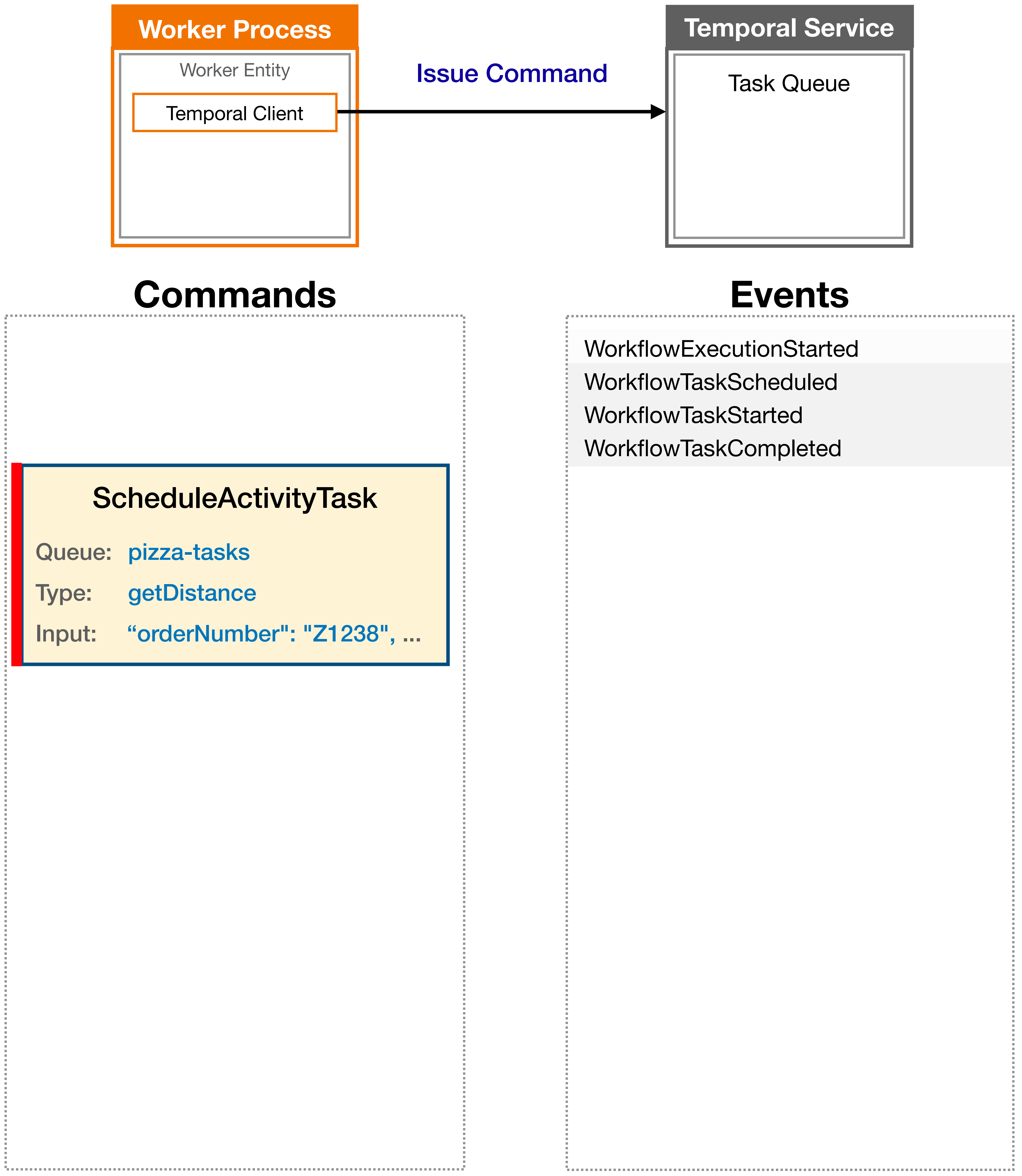


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

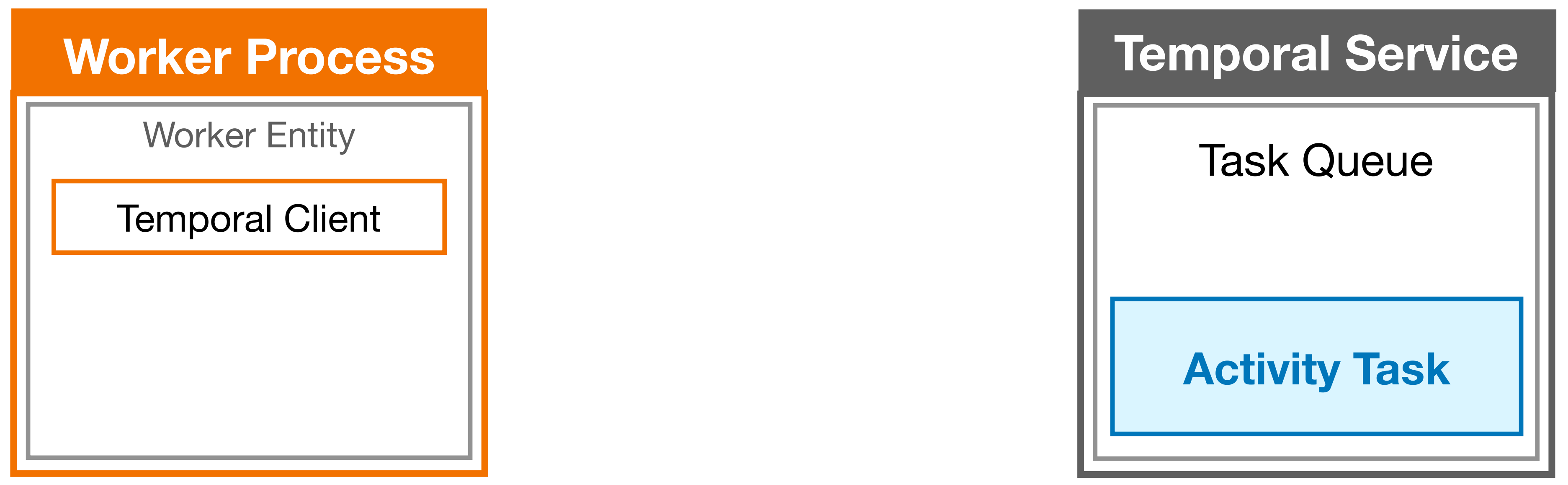


WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	“orderNumber": "Z1238", ...

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

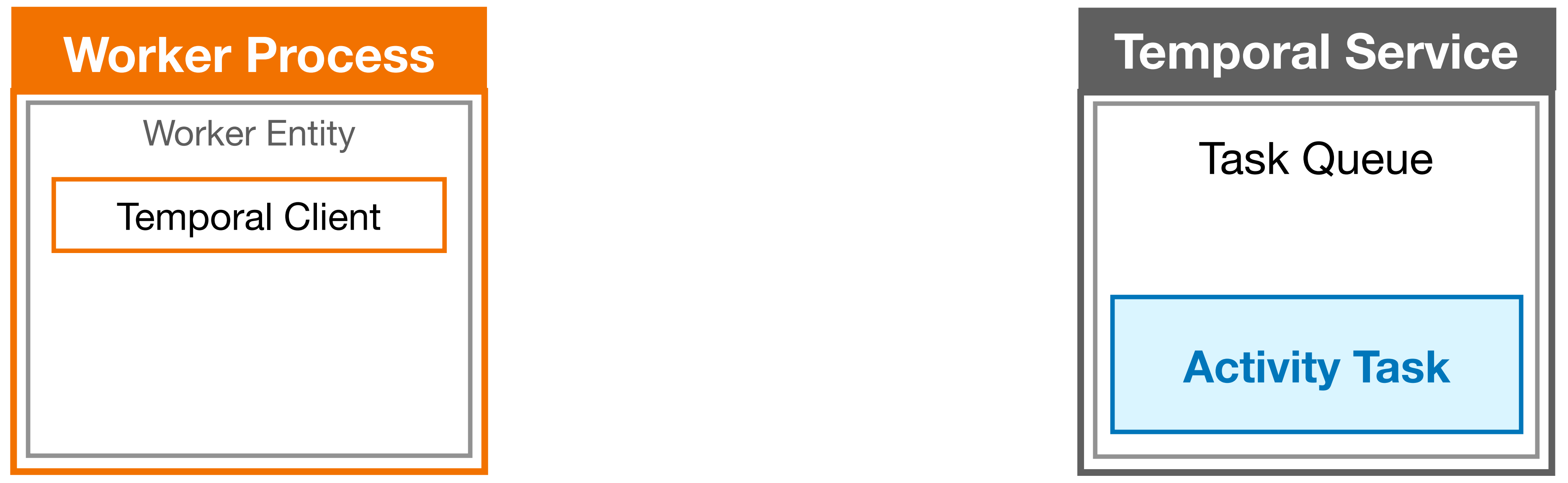
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

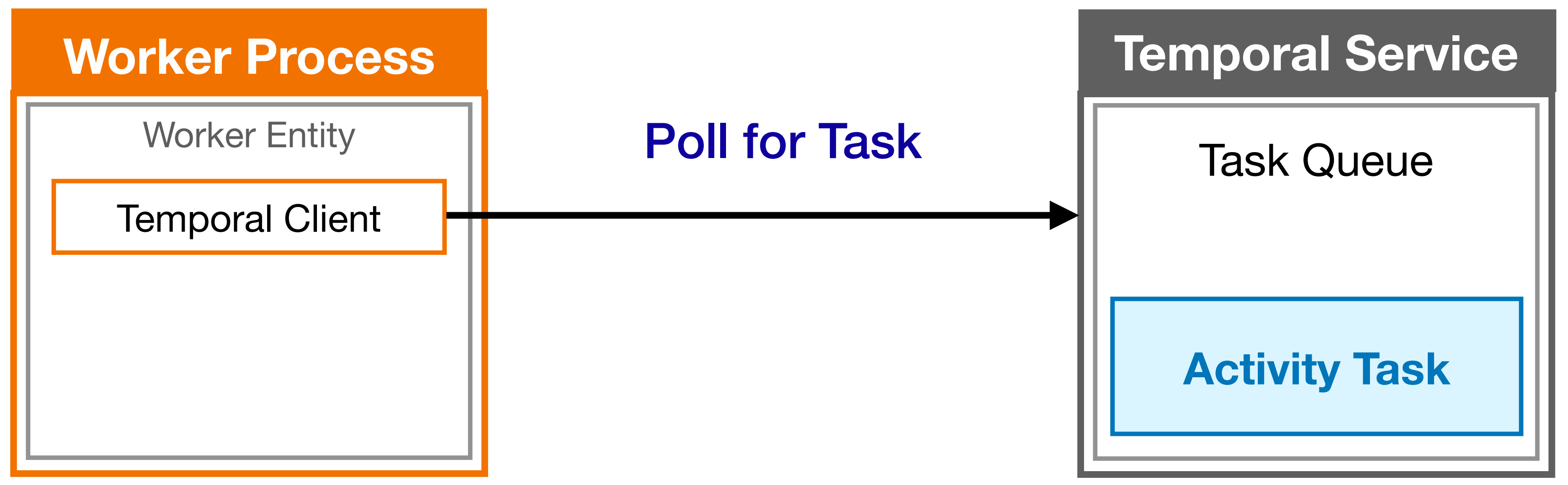
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

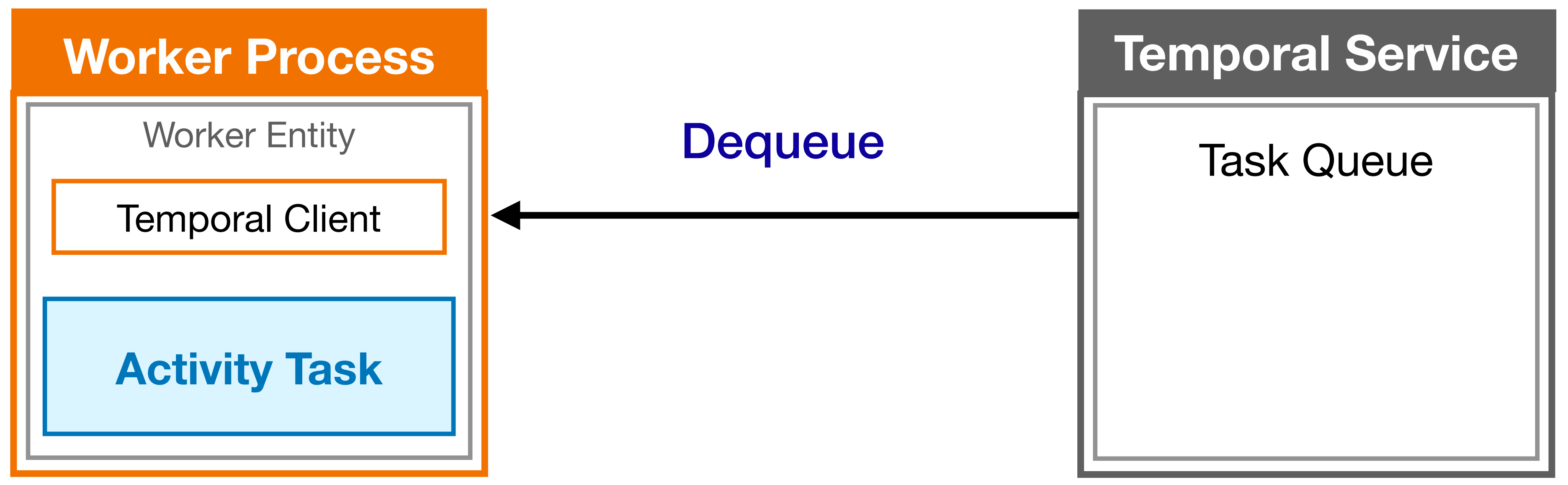
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

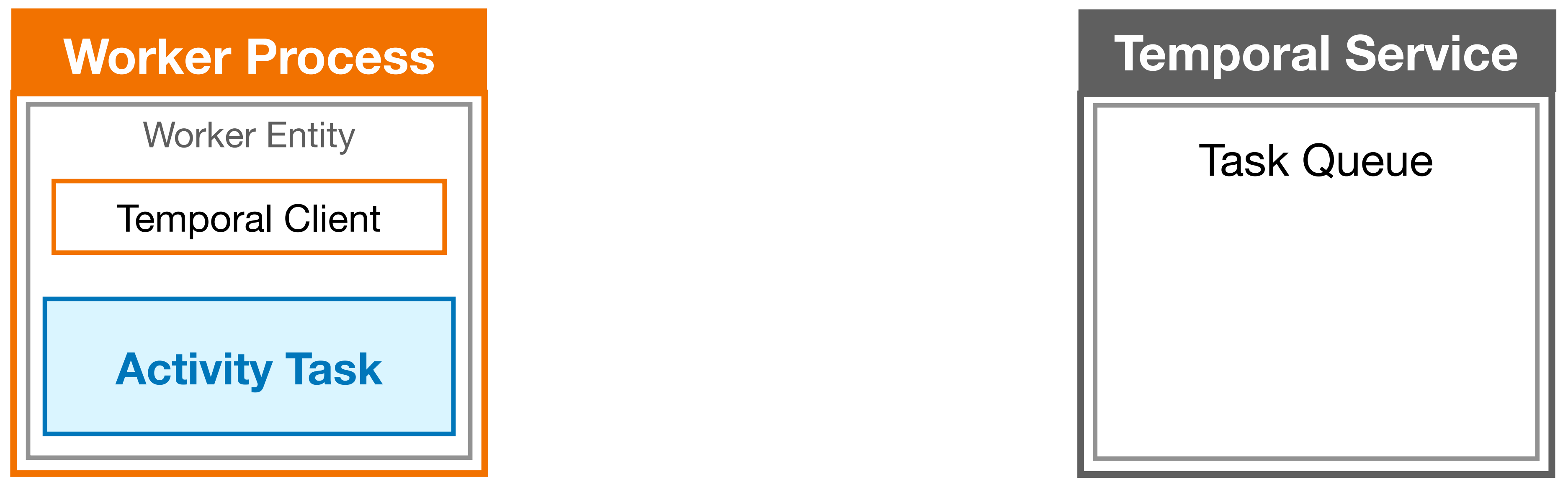
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

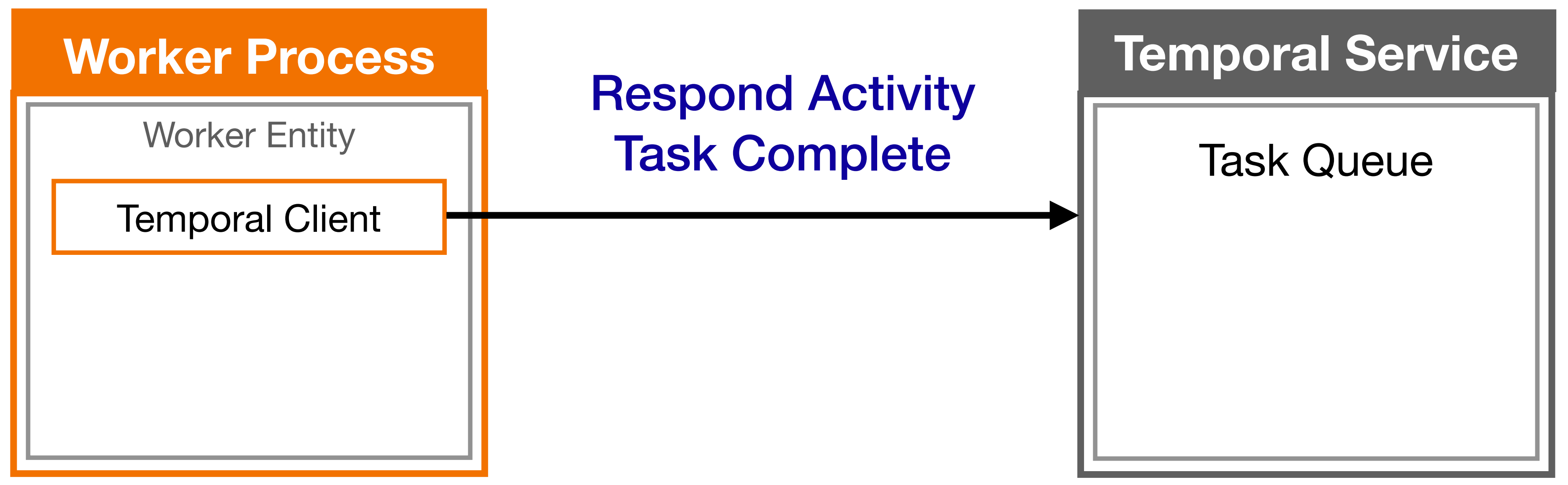
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

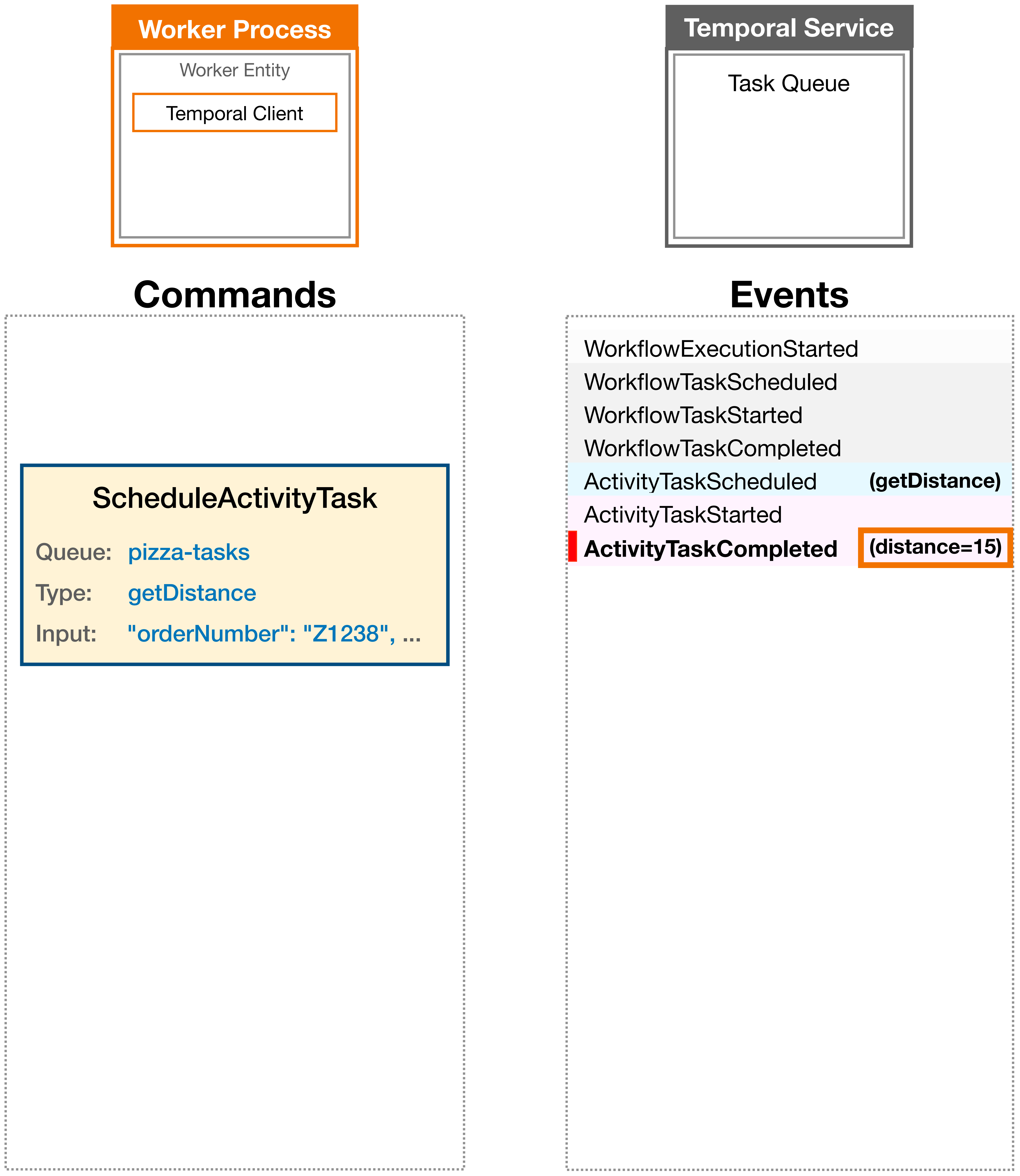
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks

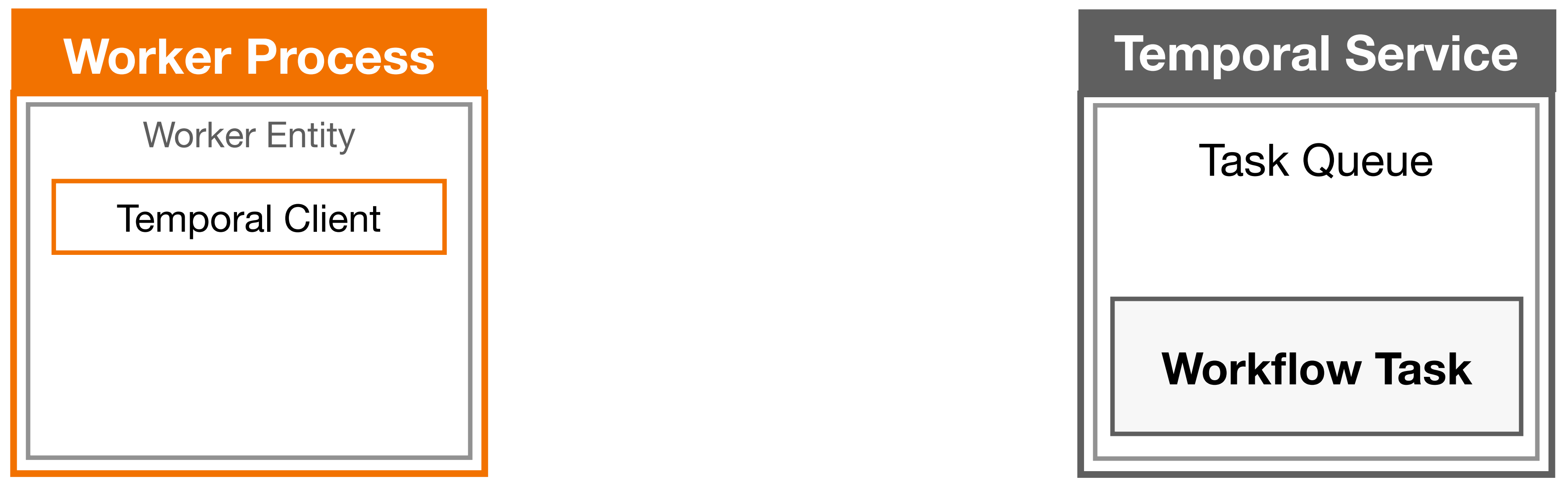
Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks

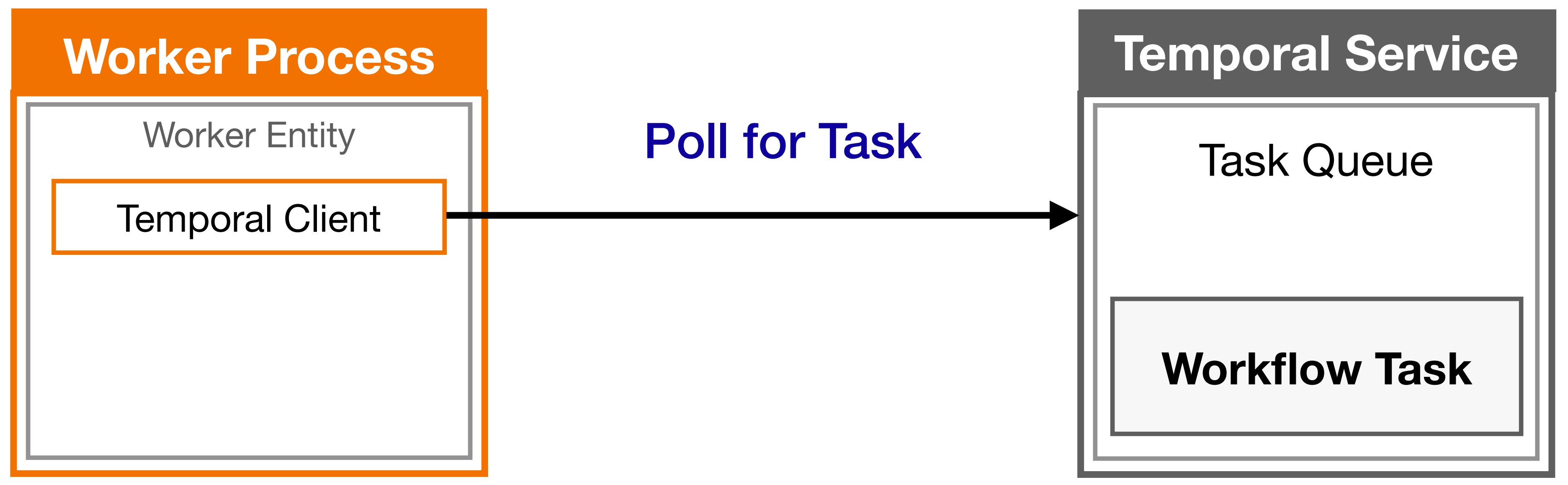
Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

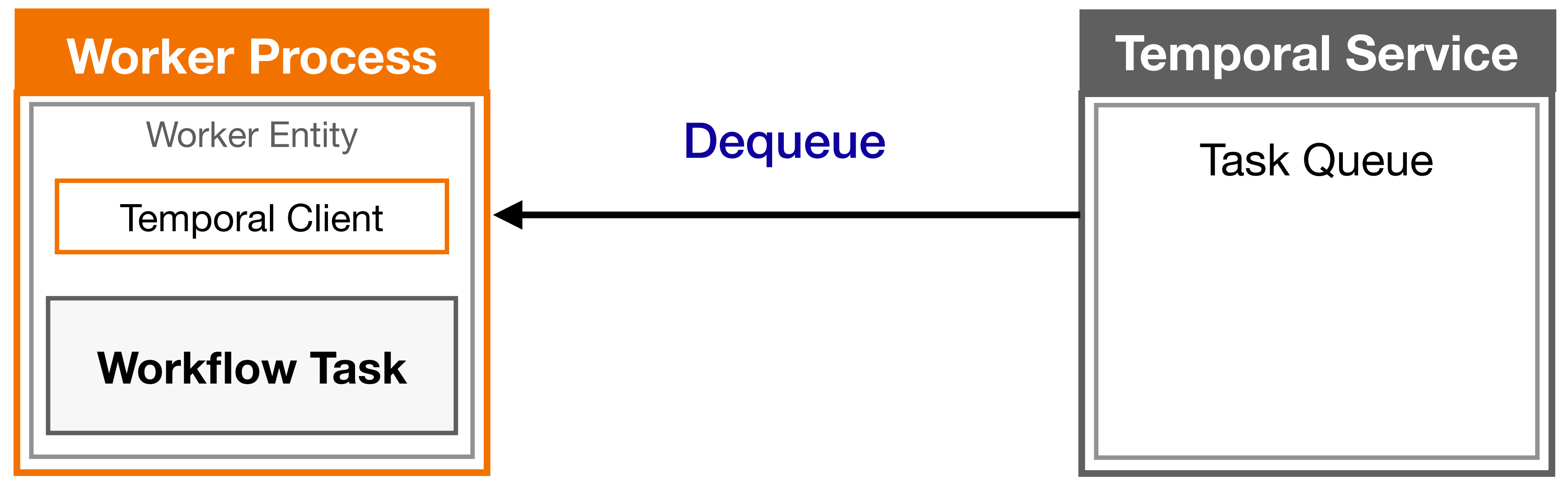
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks

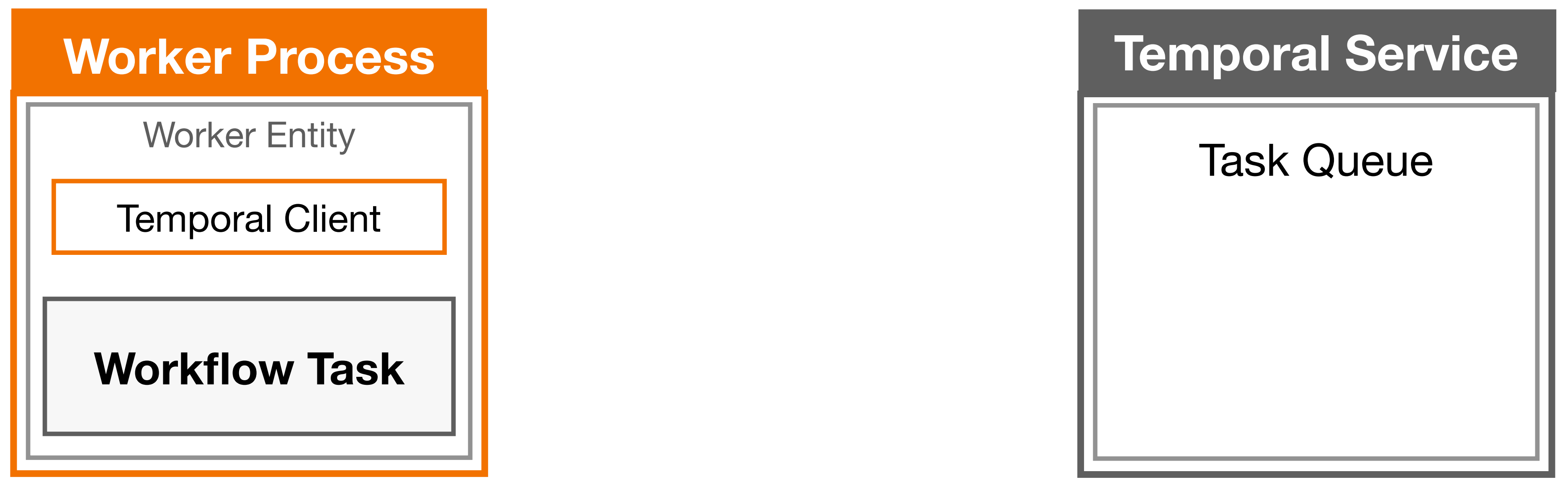
Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

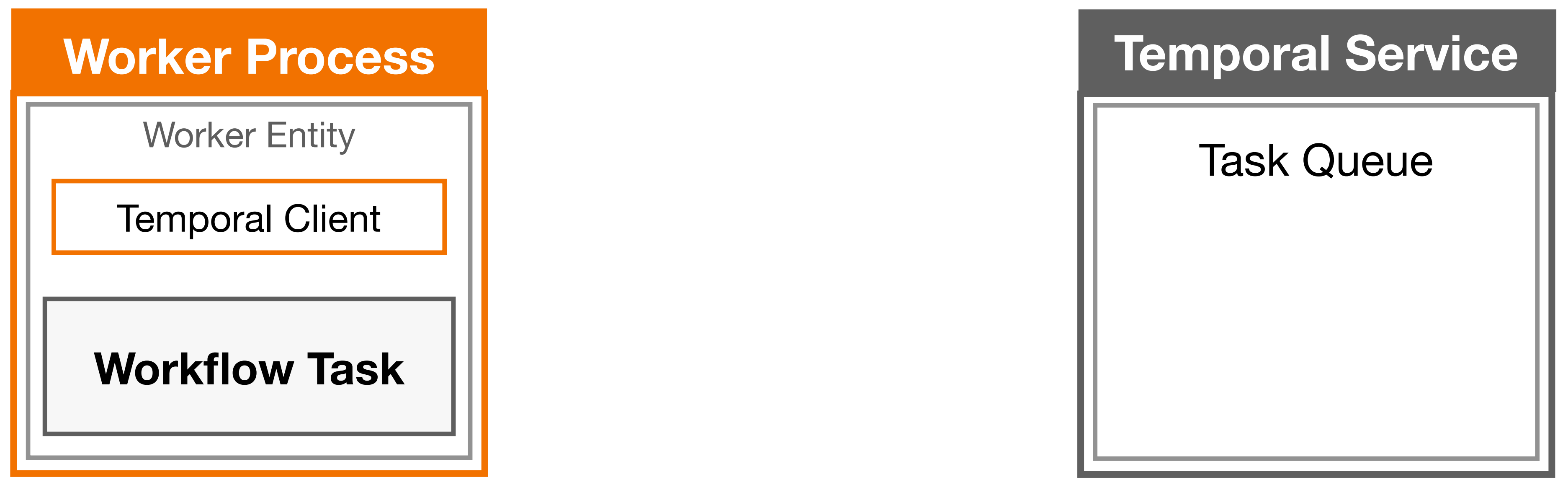
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

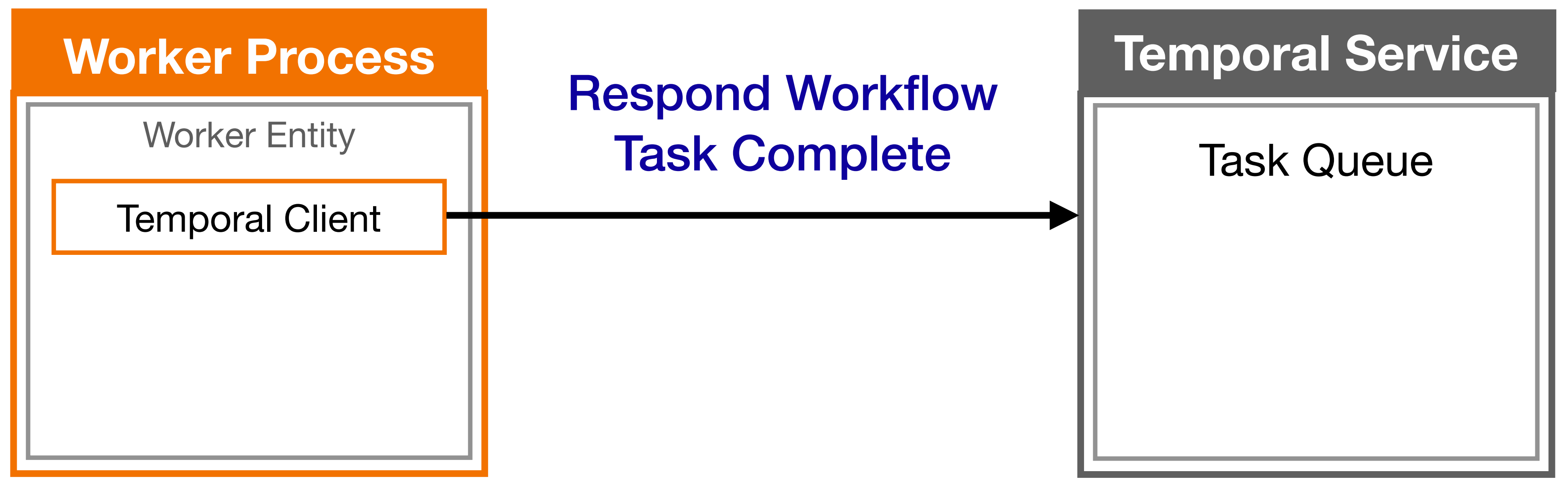
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

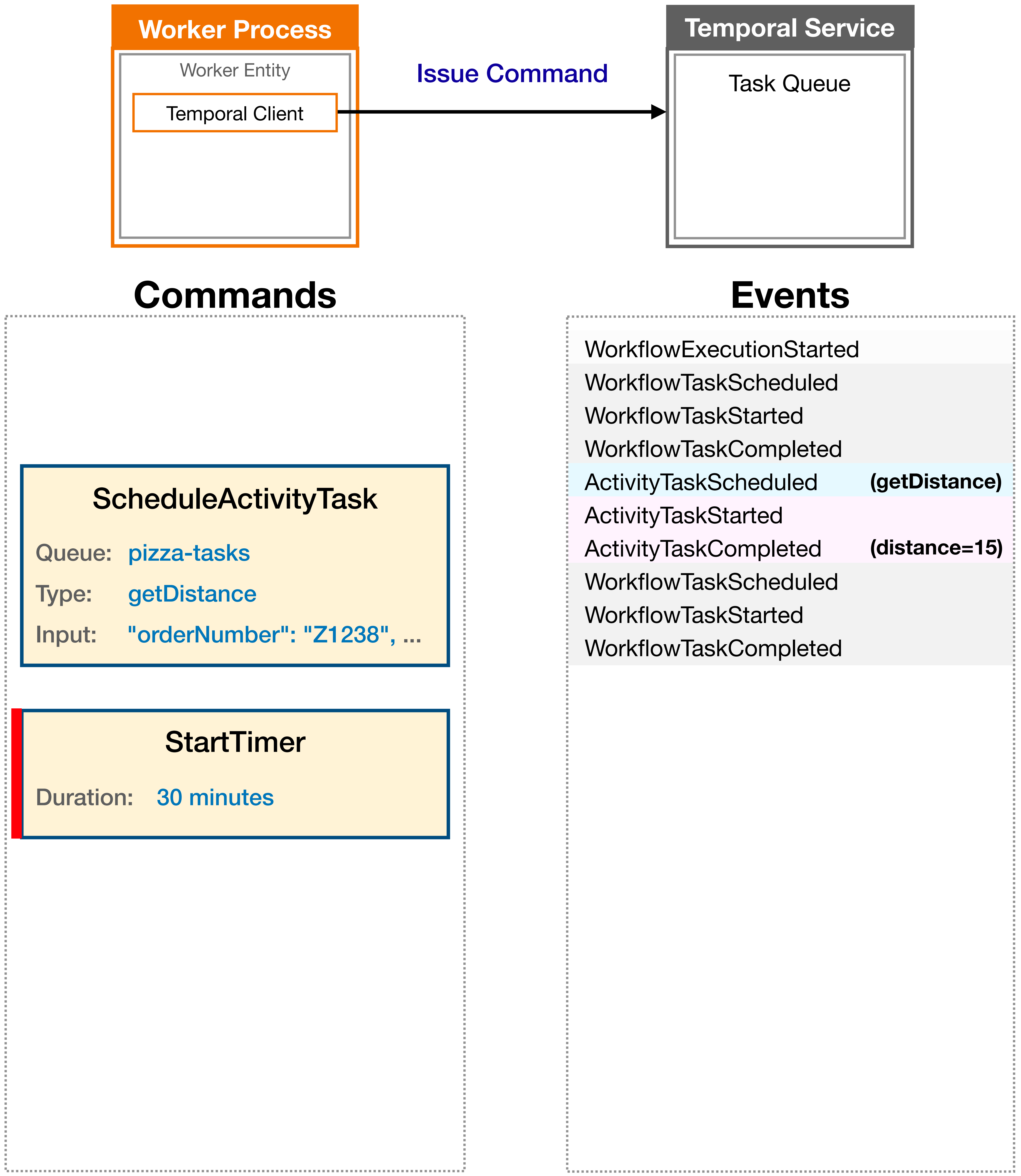
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

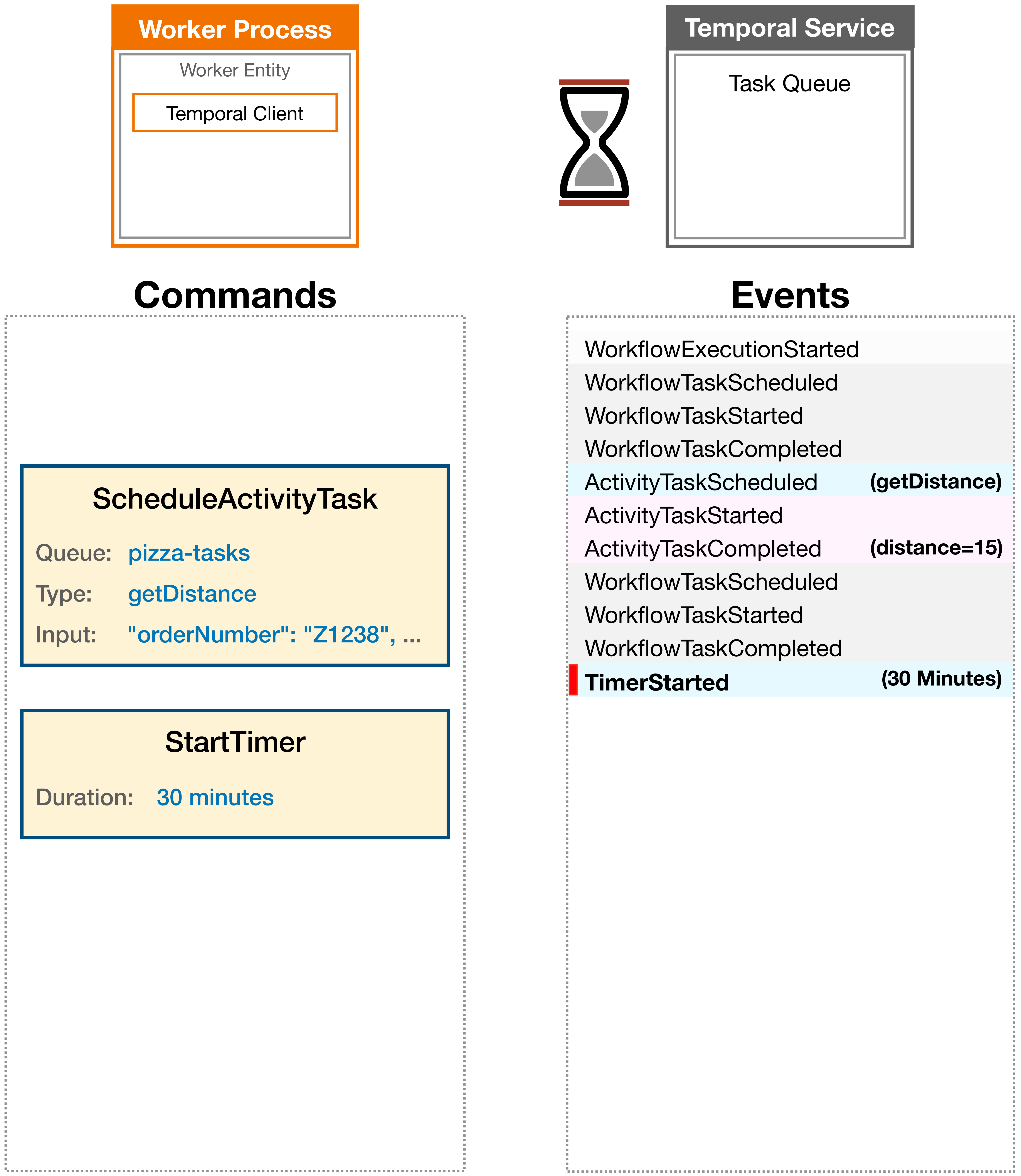
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

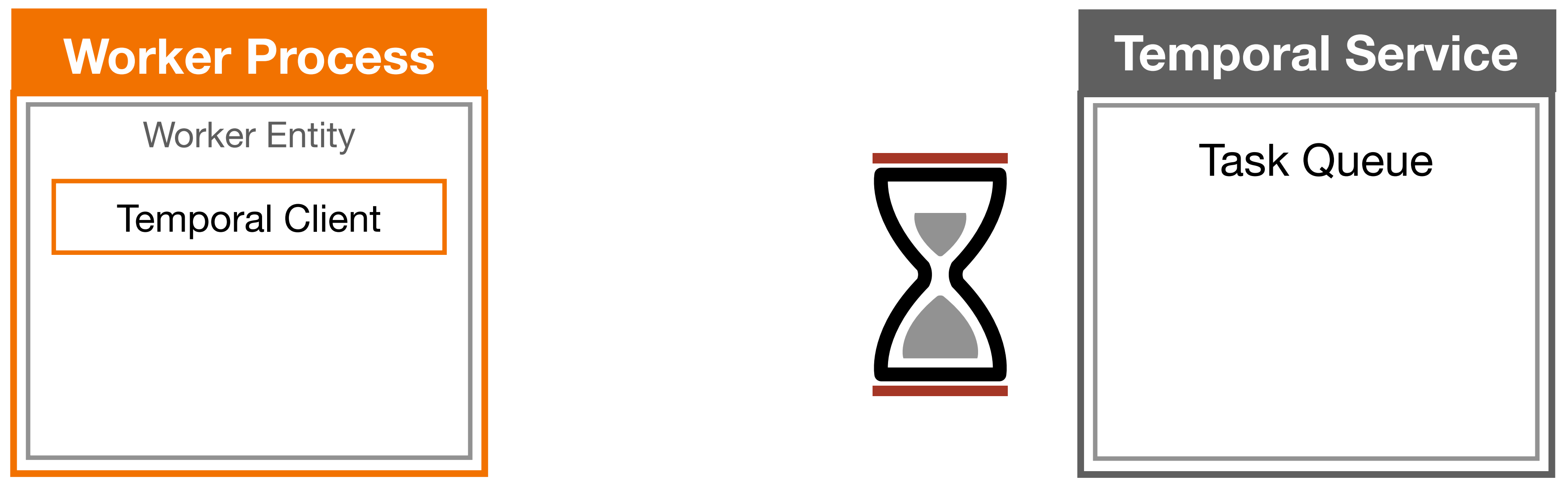
StartTimer

Duration:	30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

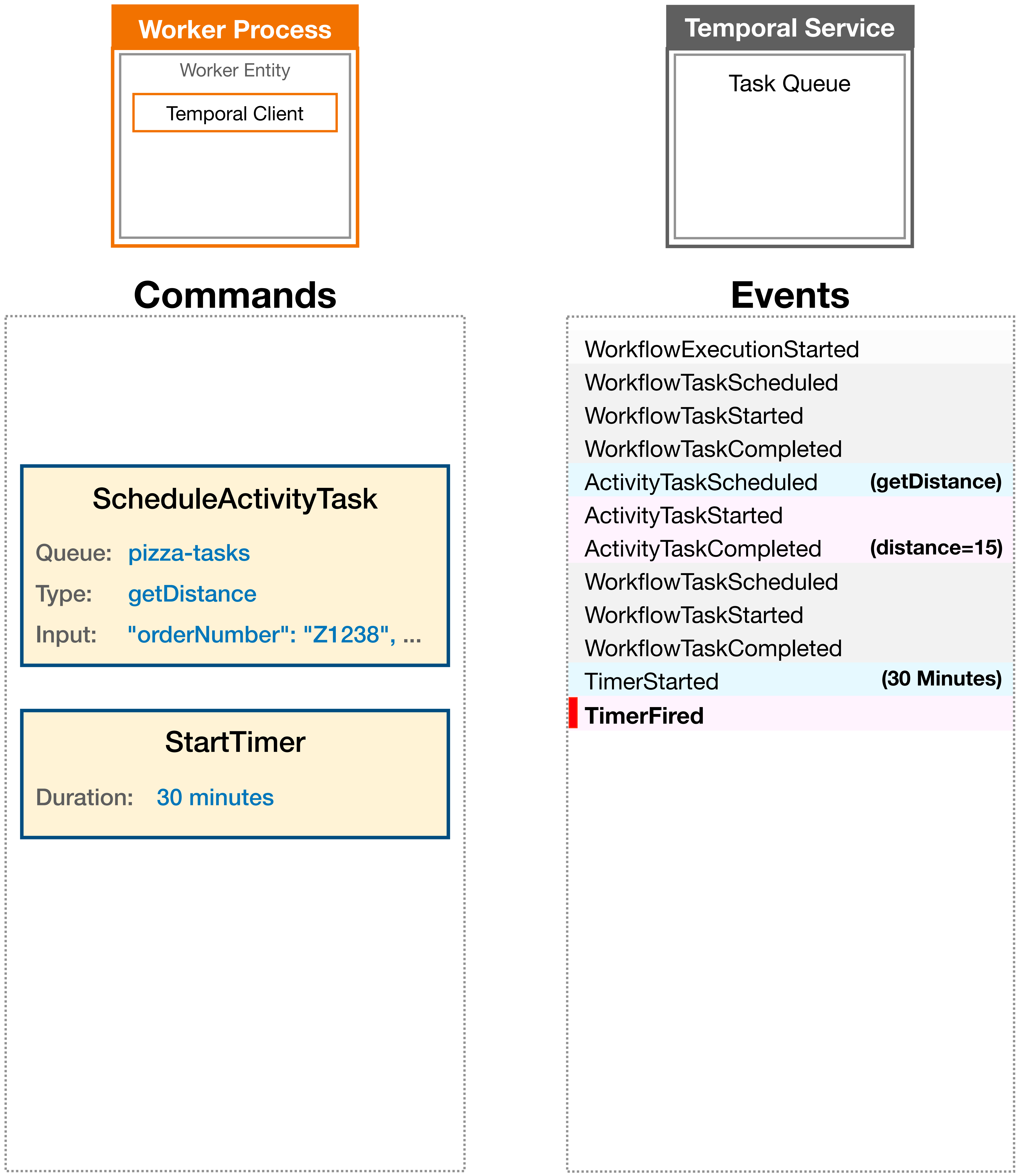
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



ScheduleActivityTask

Queue: pizza-tasks

Type: getDistance

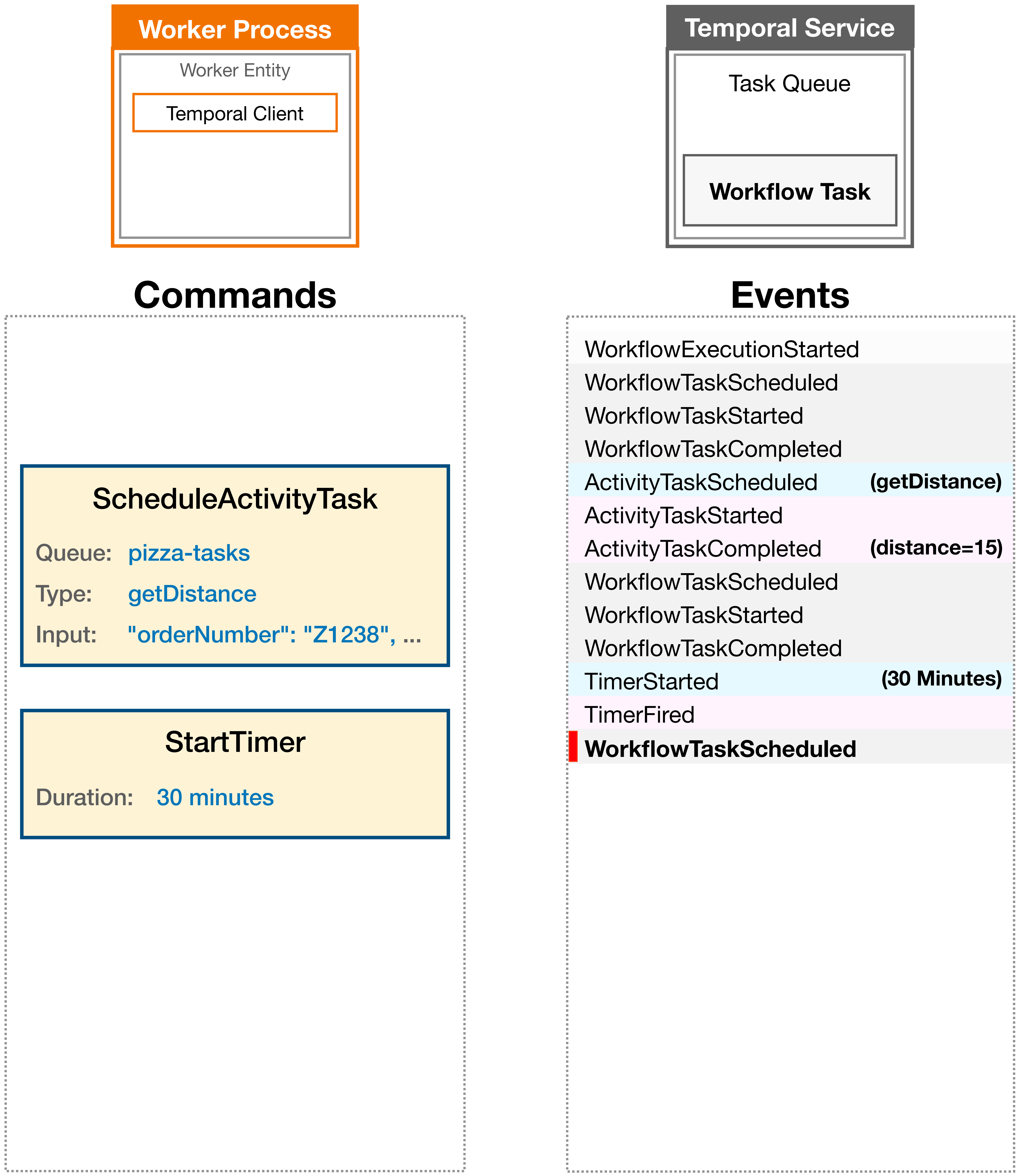
Input: "orderNumber": "Z1238", ...

StartTimer

Duration: 30 minutes

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled        (getDistance)
ActivityTaskStarted
ActivityTaskCompleted        (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted                 (30 Minutes)
TimerFired
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

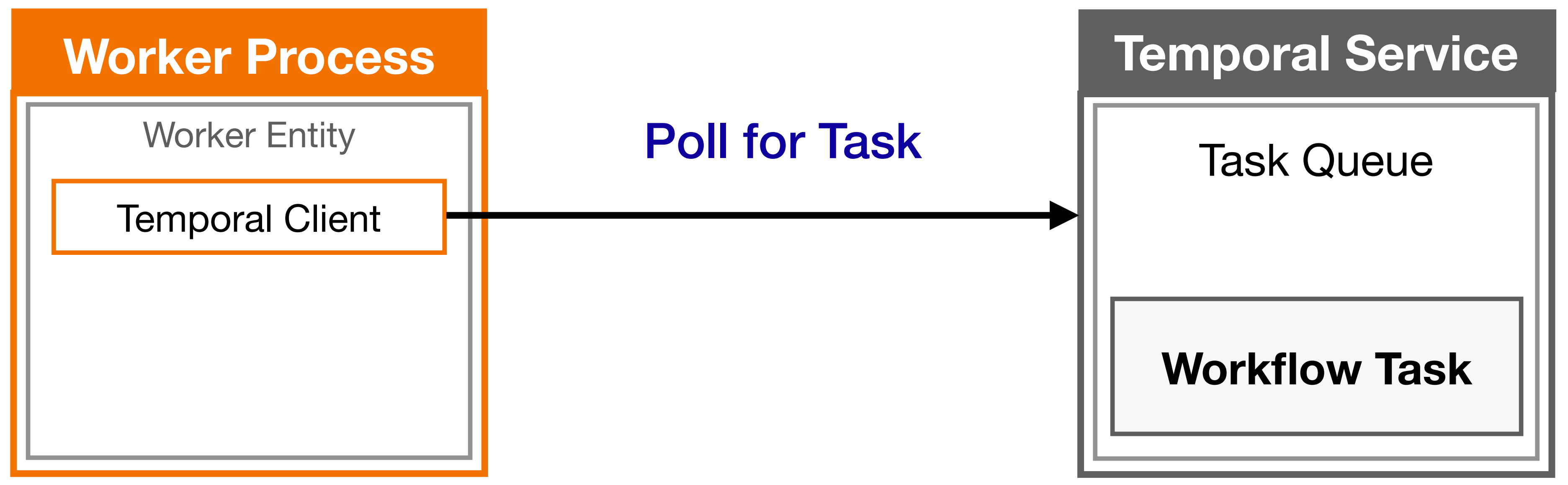
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

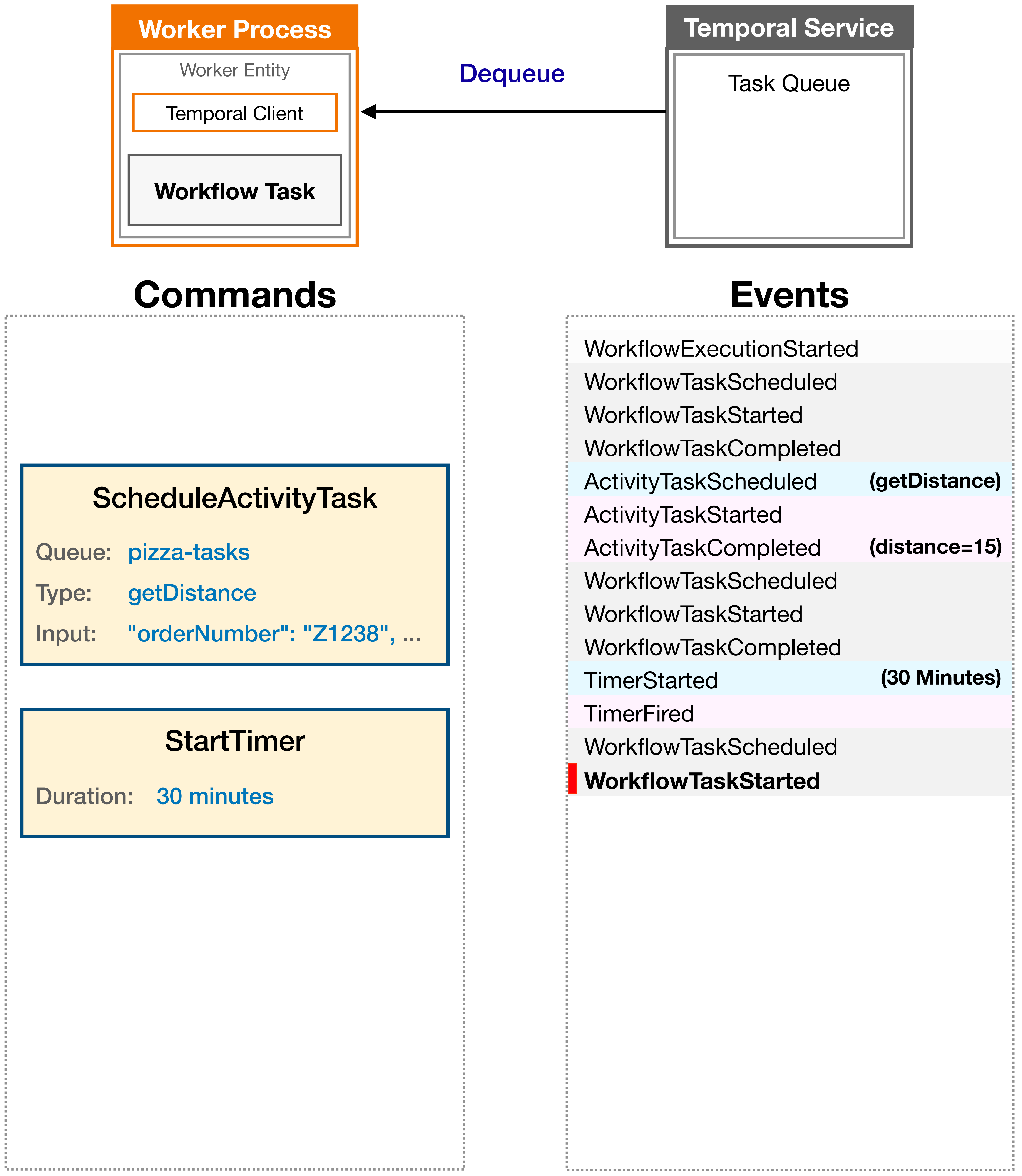
StartTimer

Duration:	30 minutes

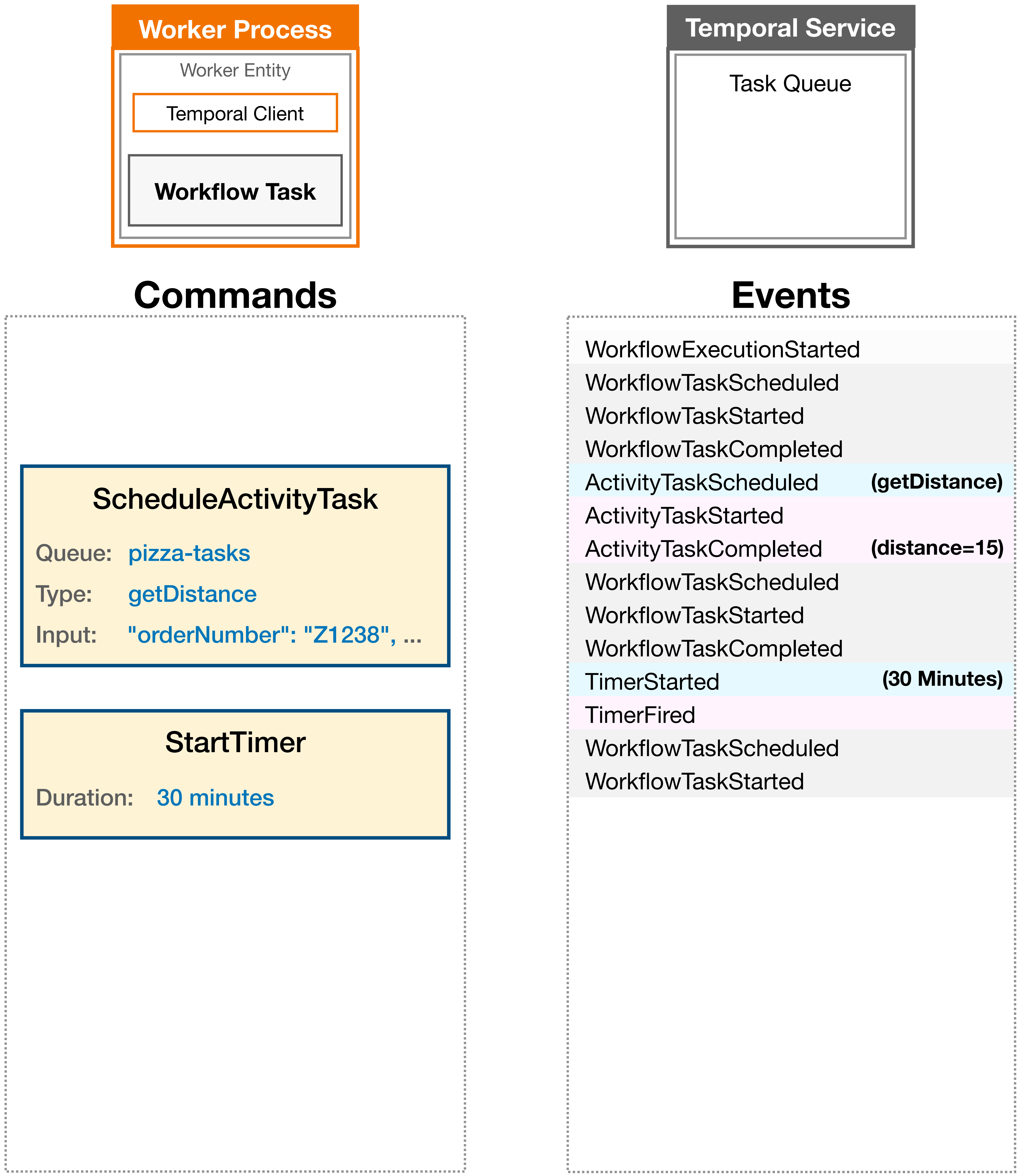
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

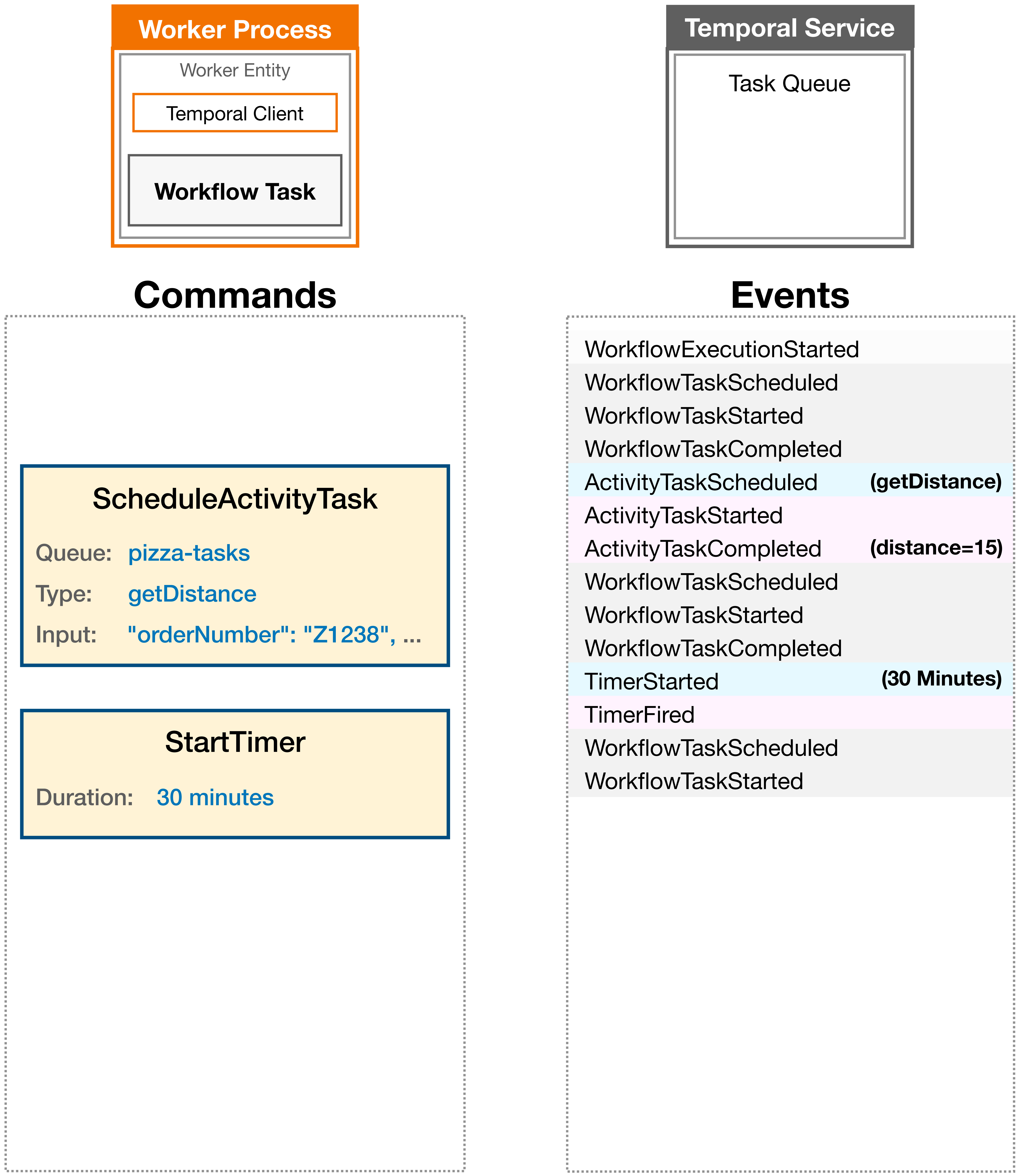


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker crashes here



ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

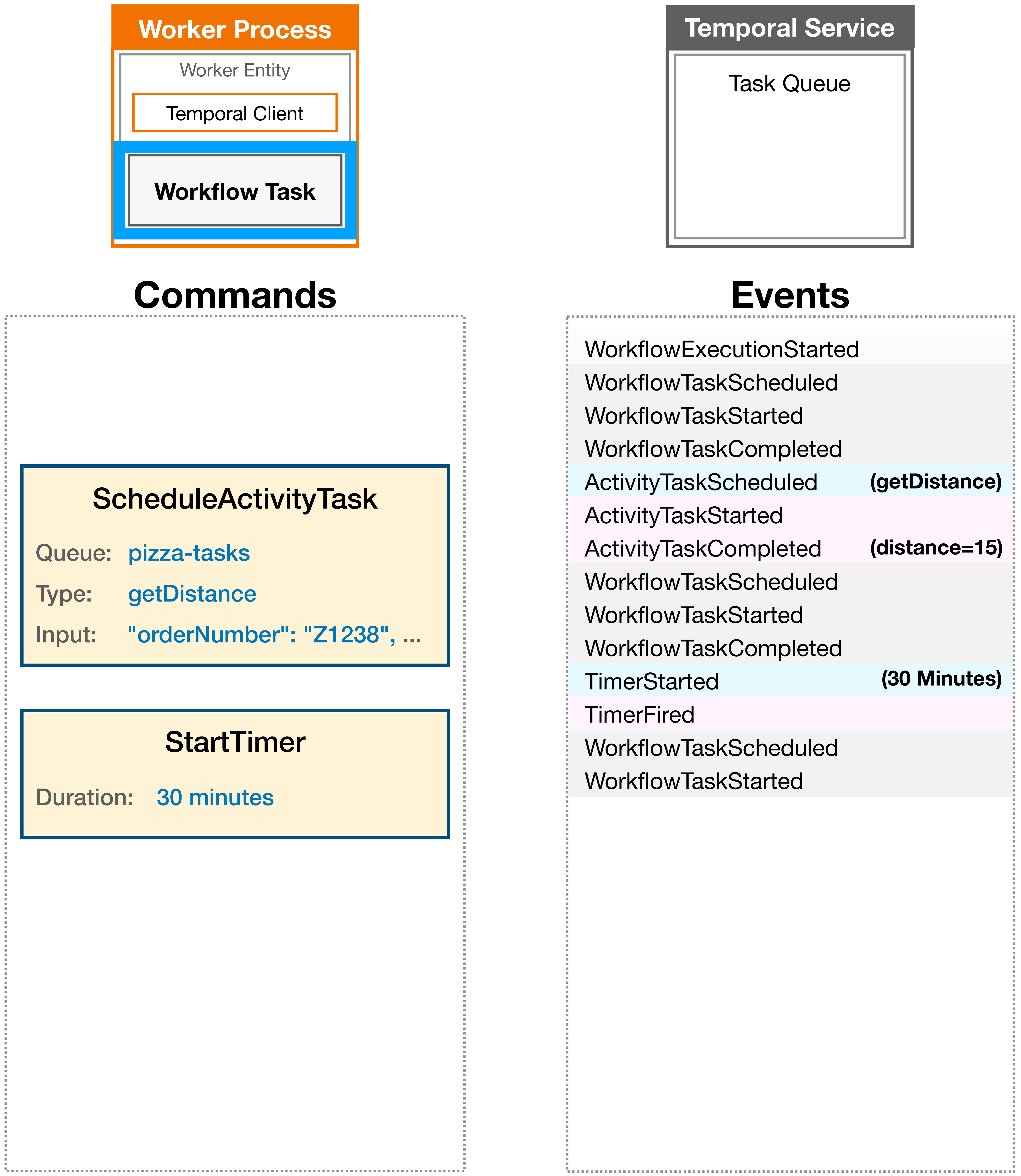
StartTimer

Duration: 30 minutes

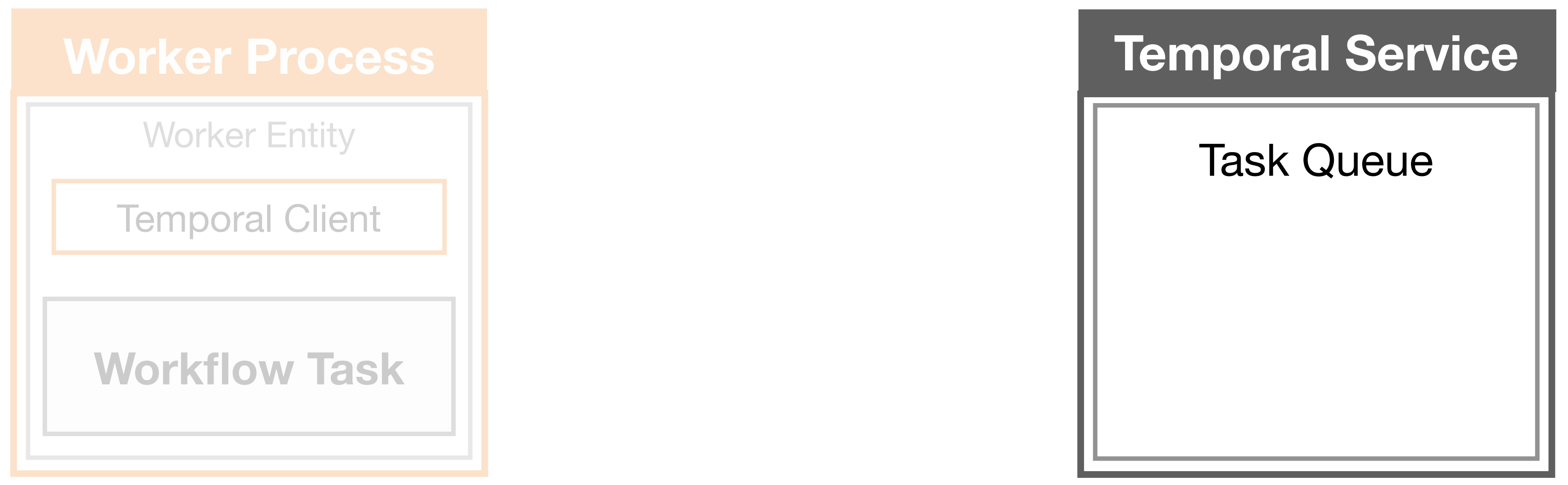
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker crashes here



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

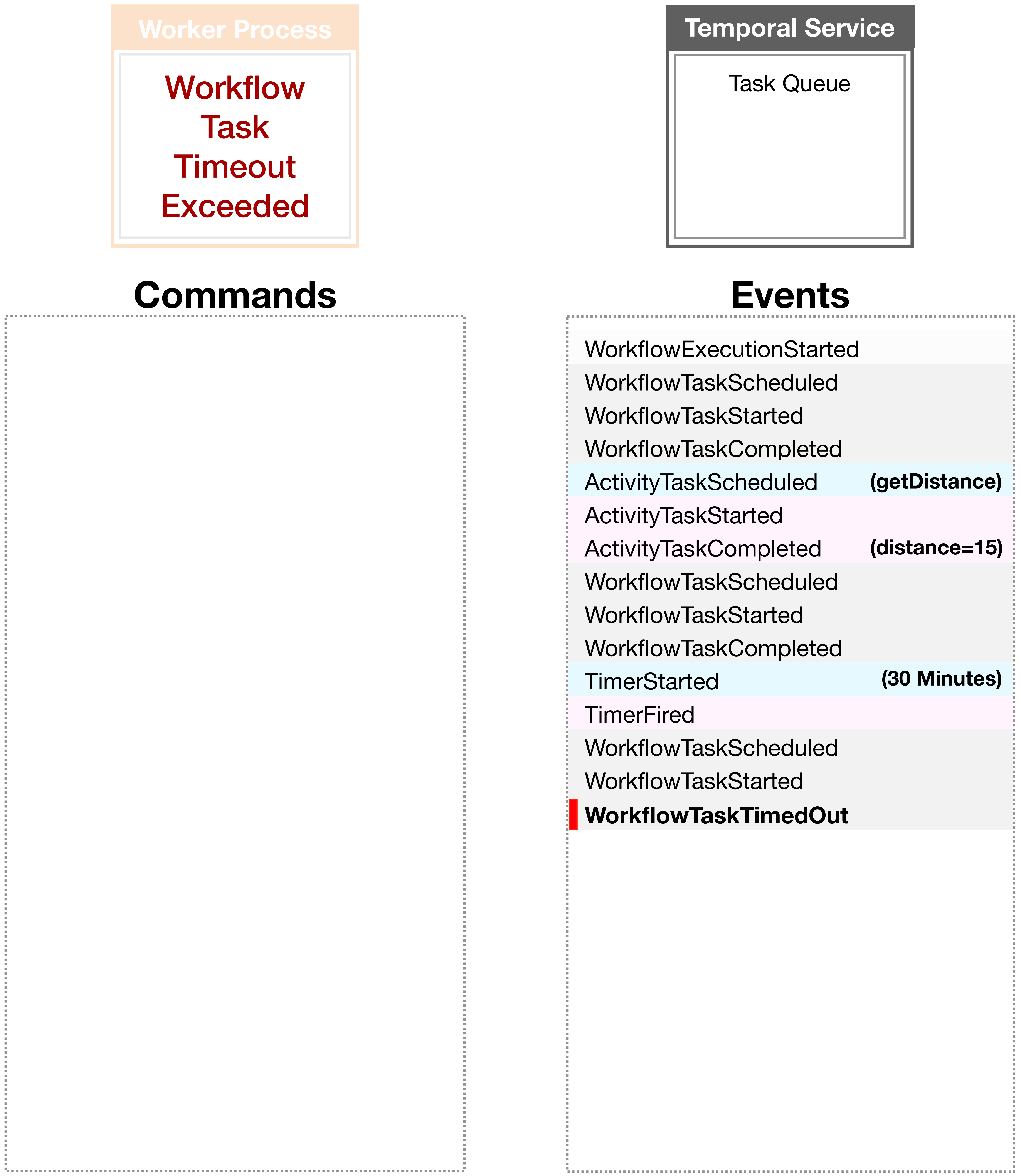


Commands

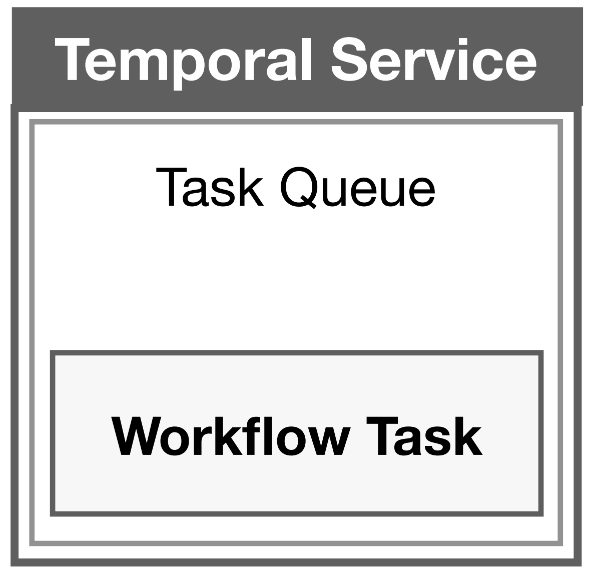
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

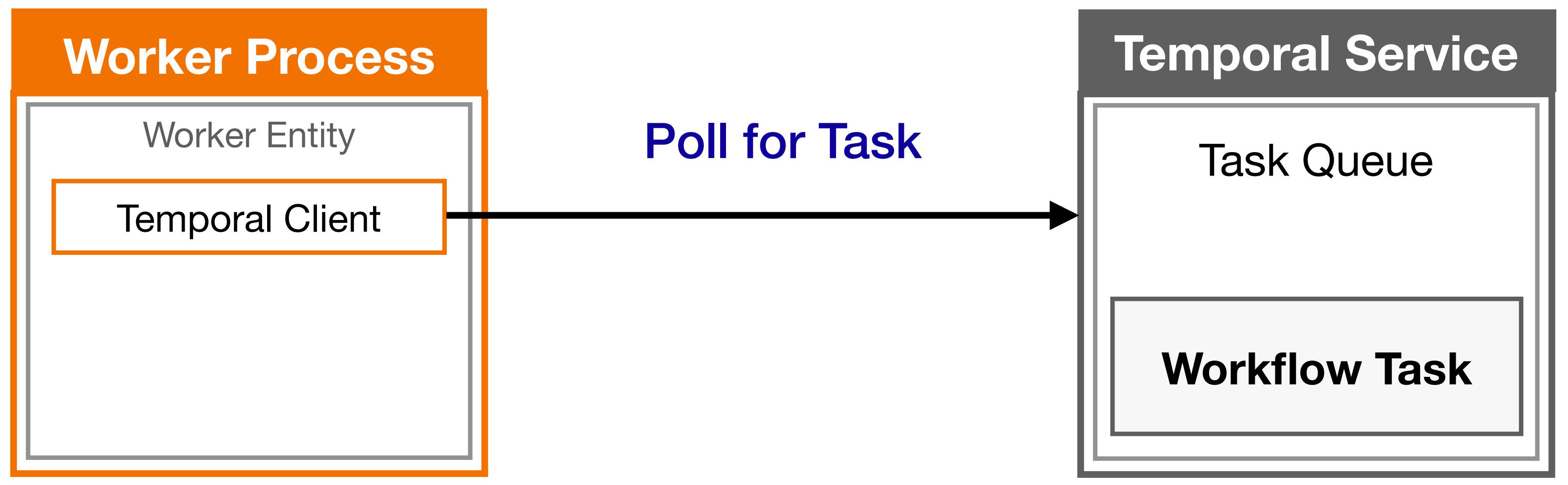


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

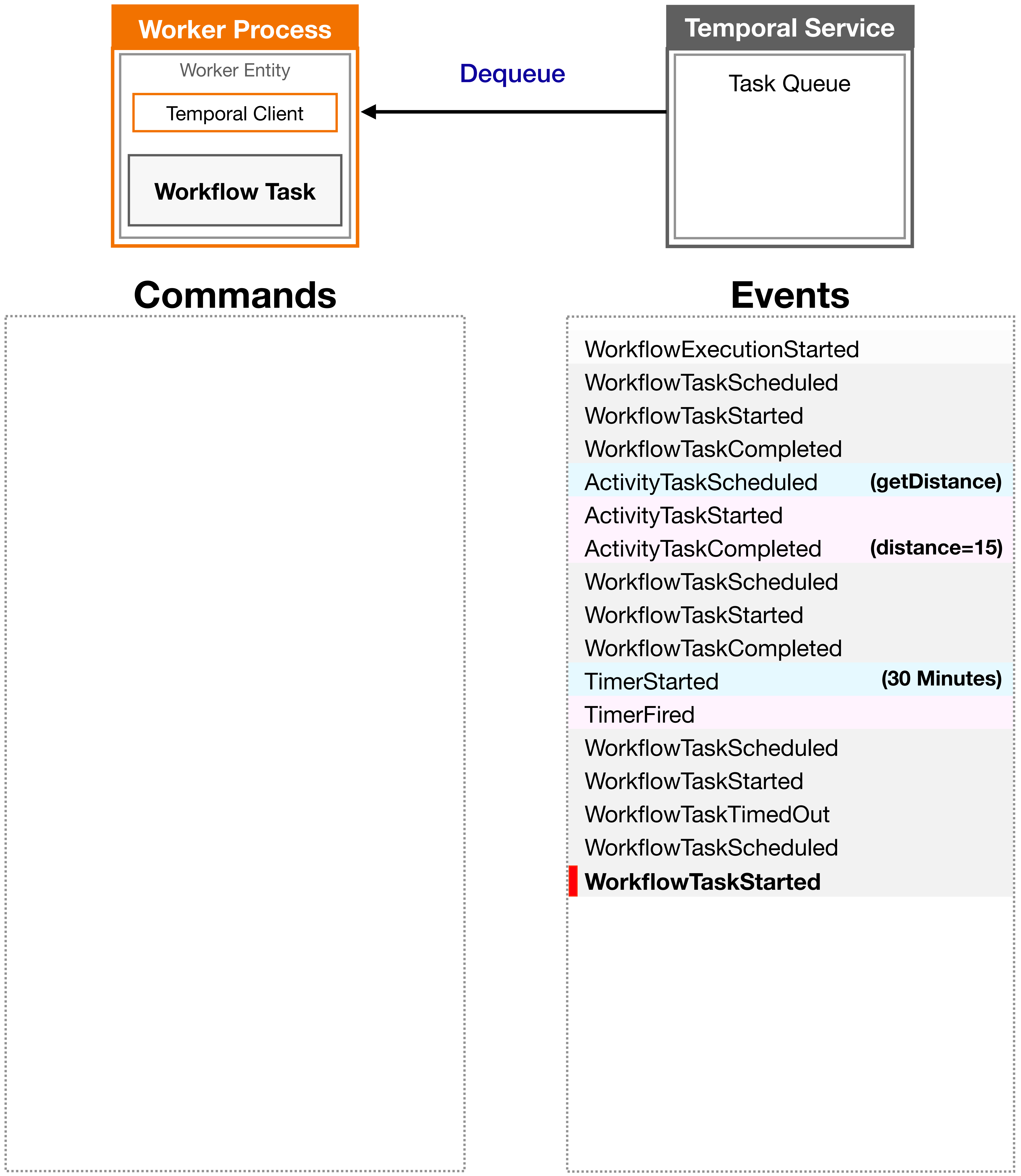


Commands

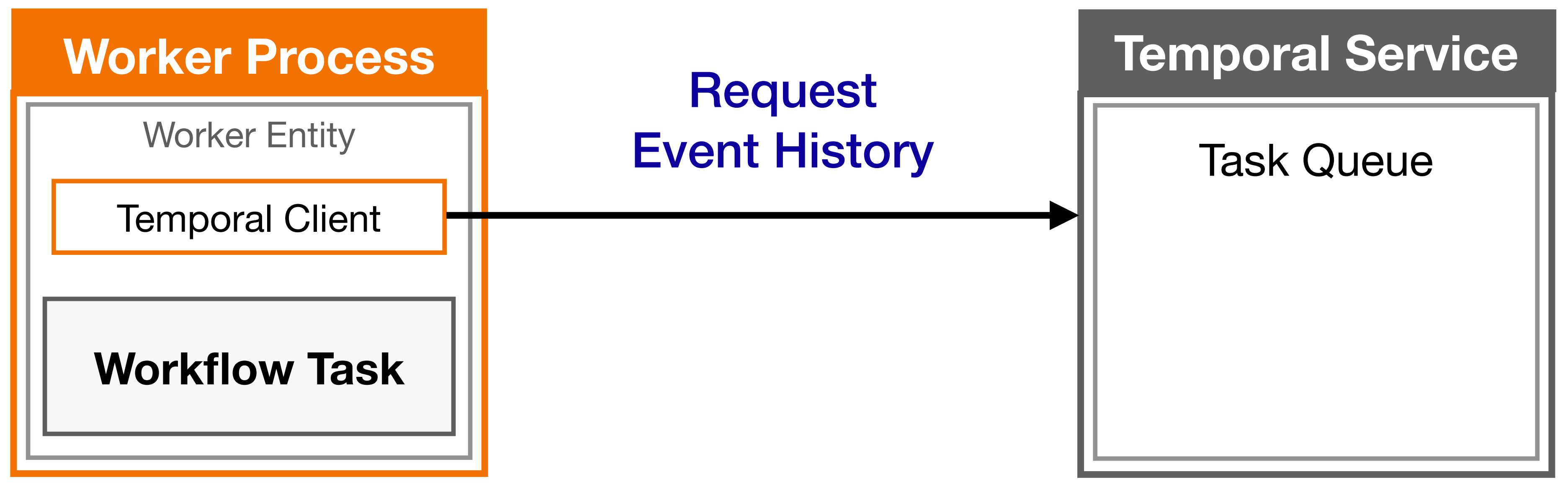
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

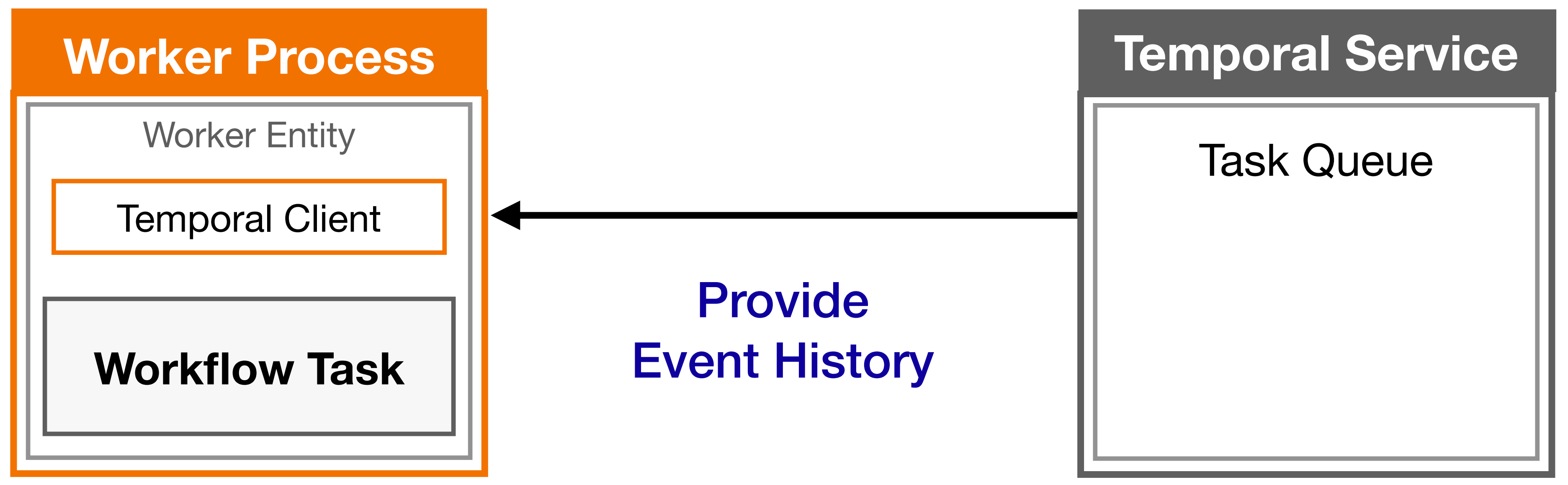


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

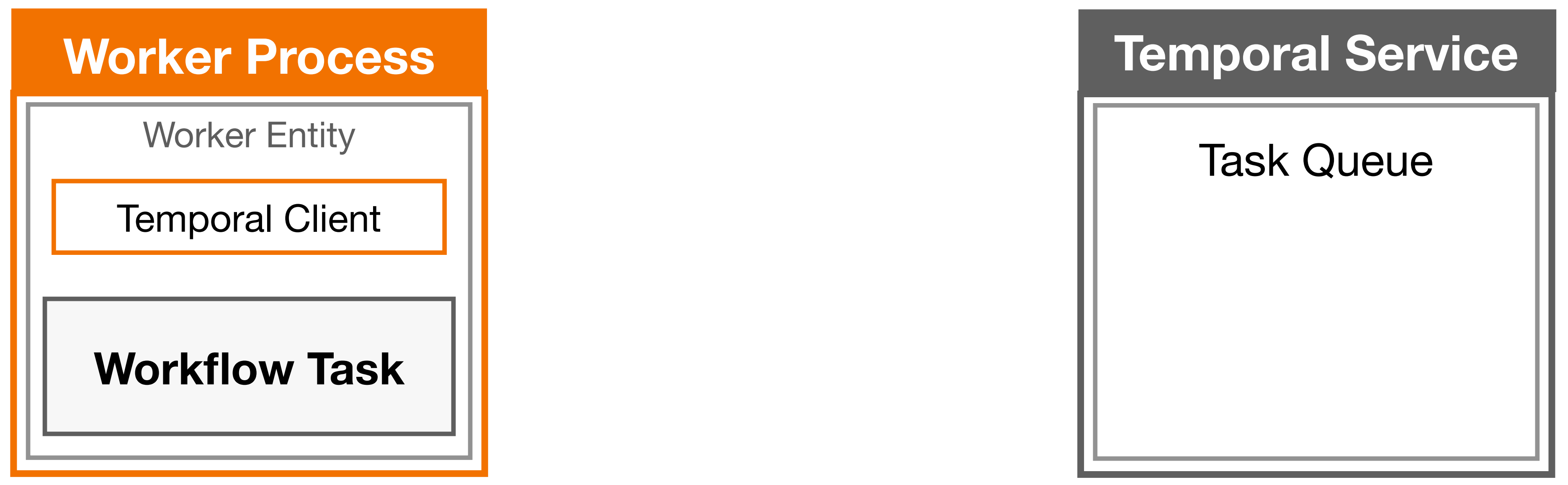


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

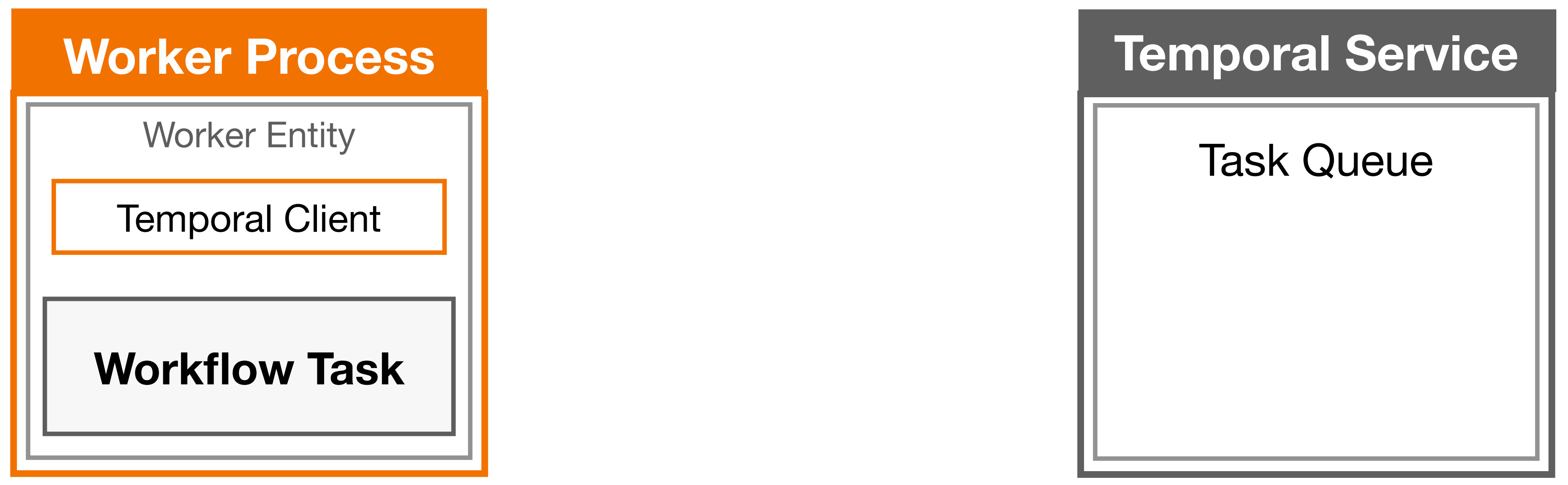


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

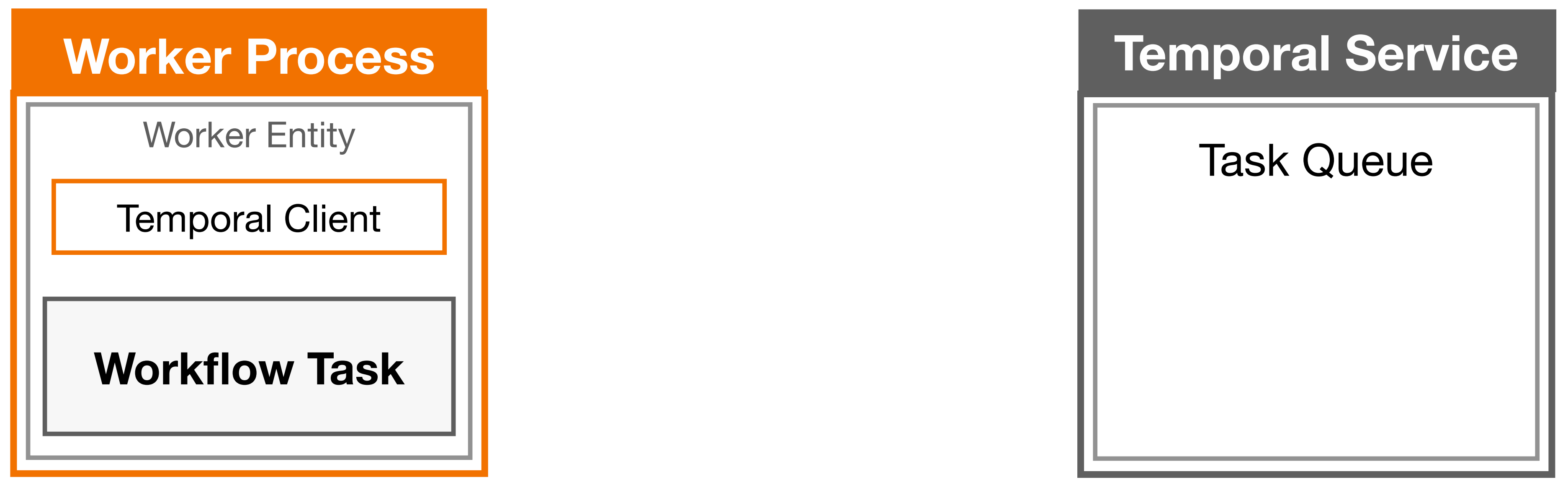


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

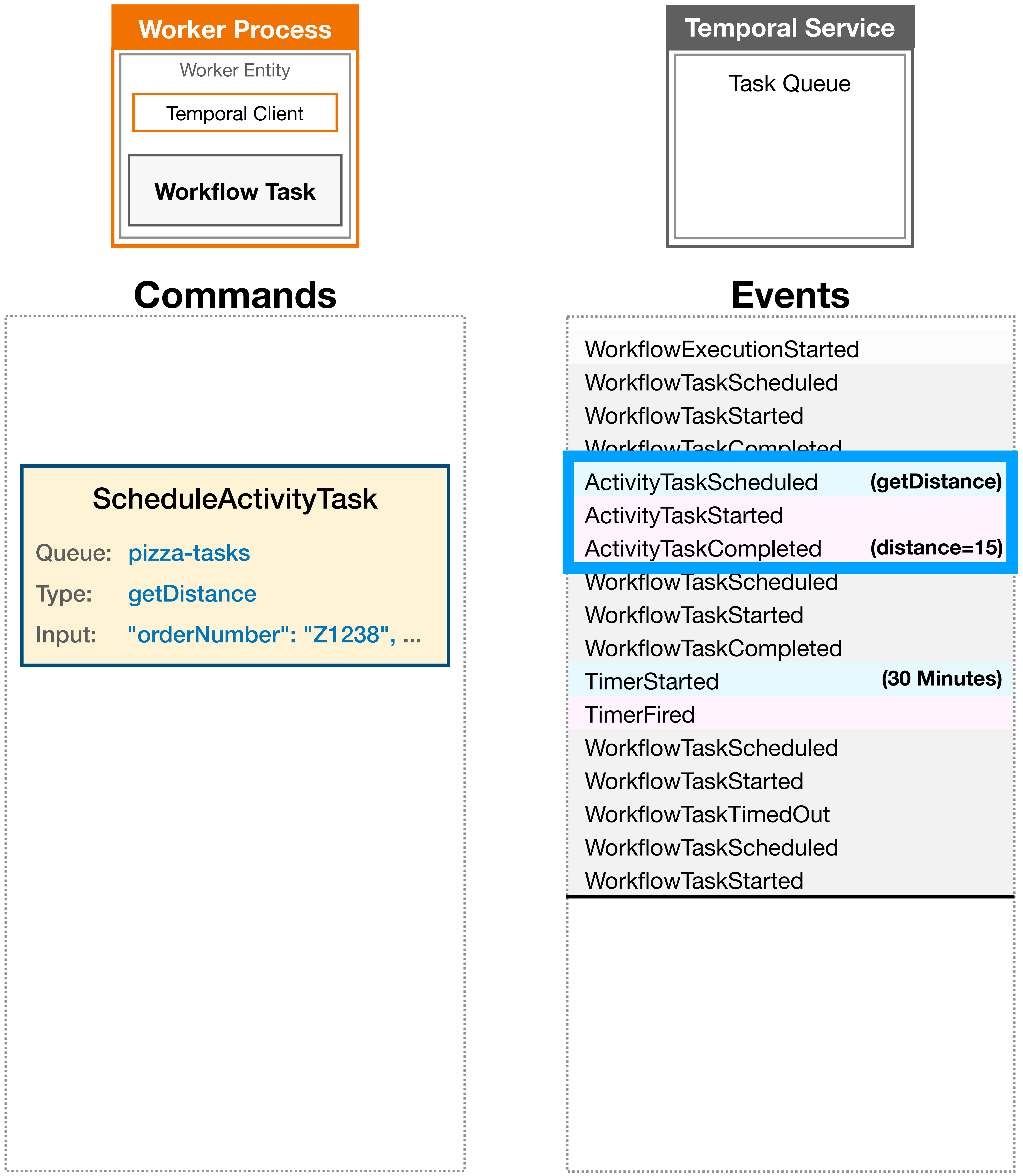


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

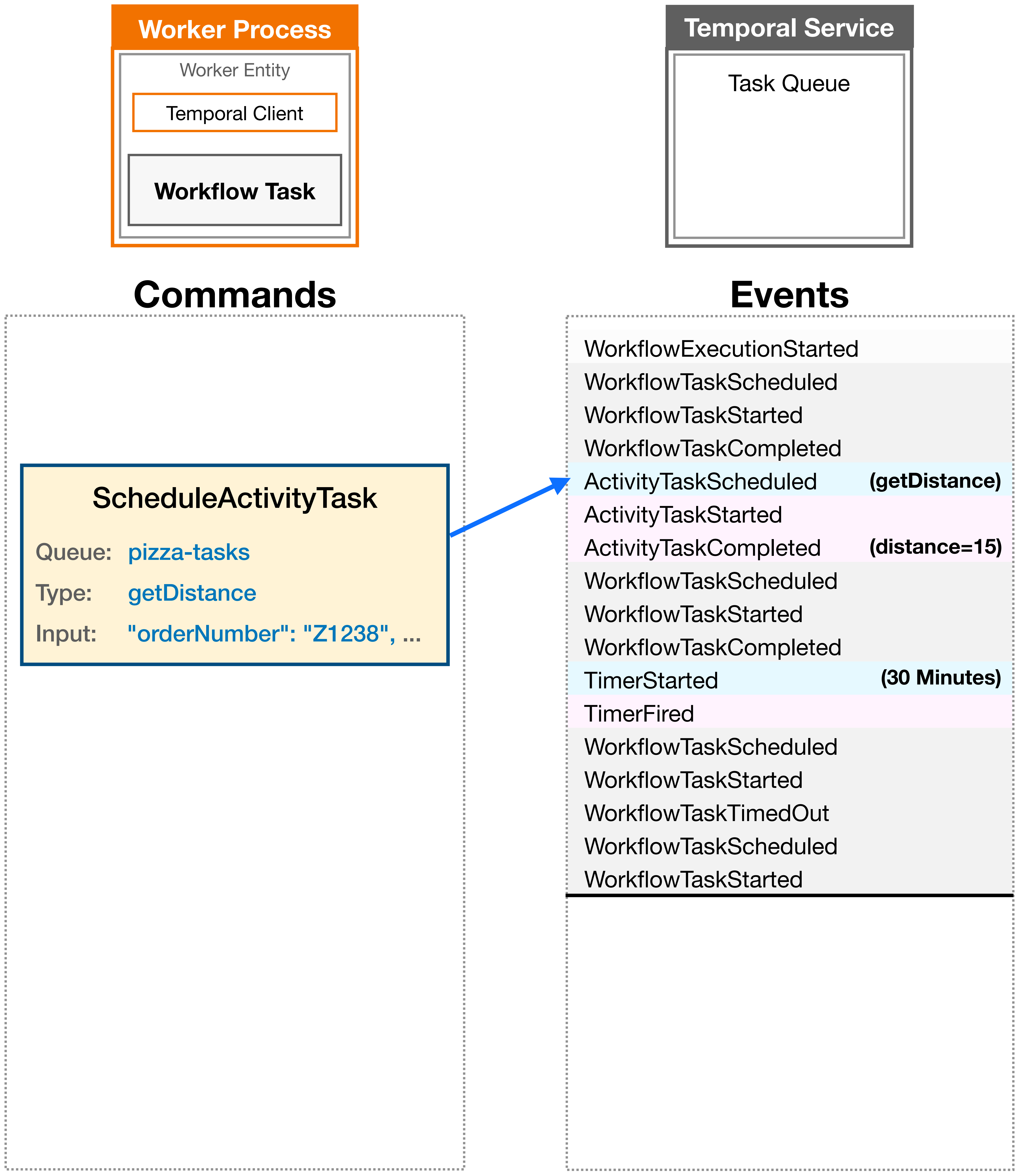


ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled        (getDistance)
ActivityTaskStarted
ActivityTaskCompleted        (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted                 (30 Minutes)
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

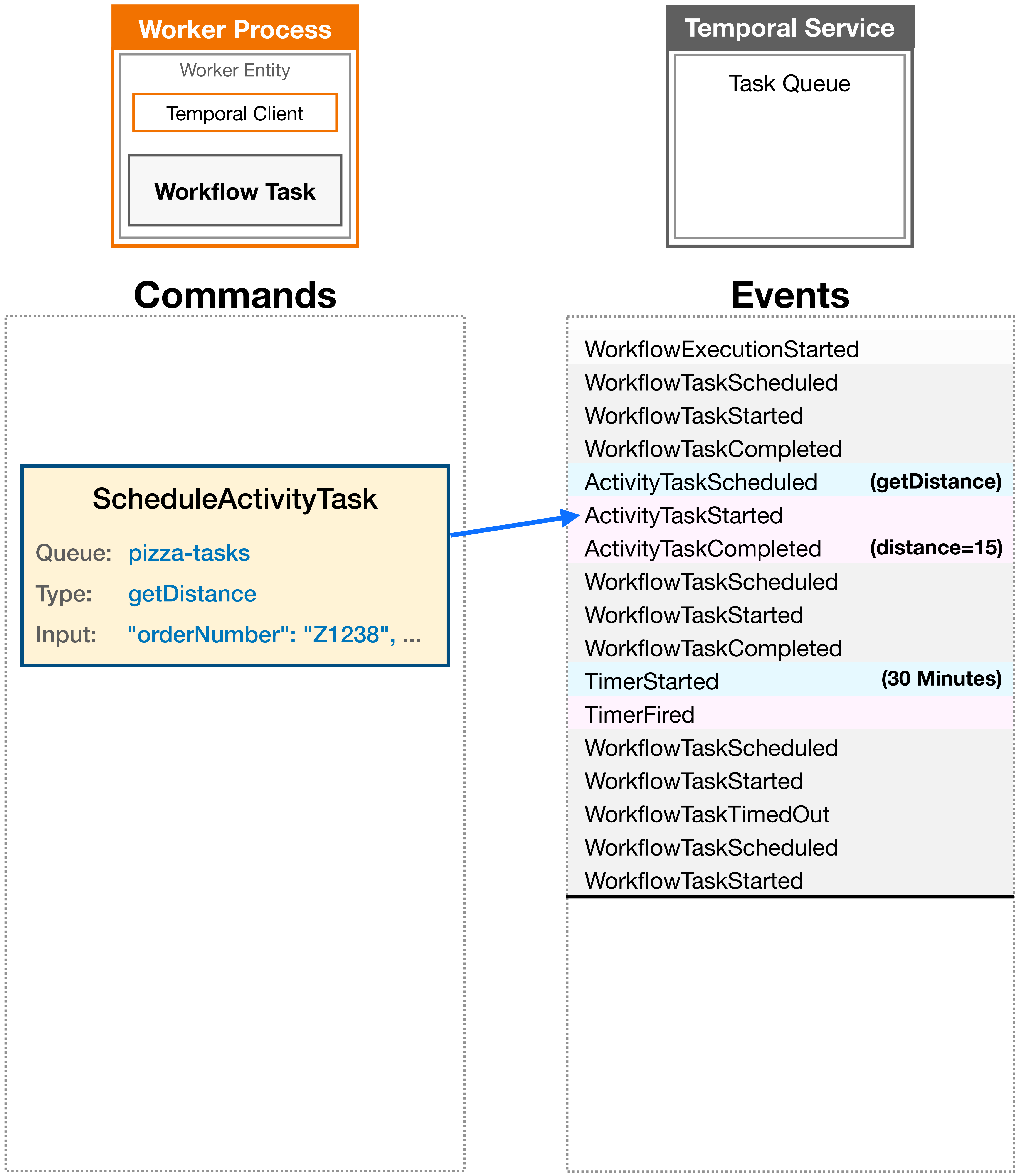
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

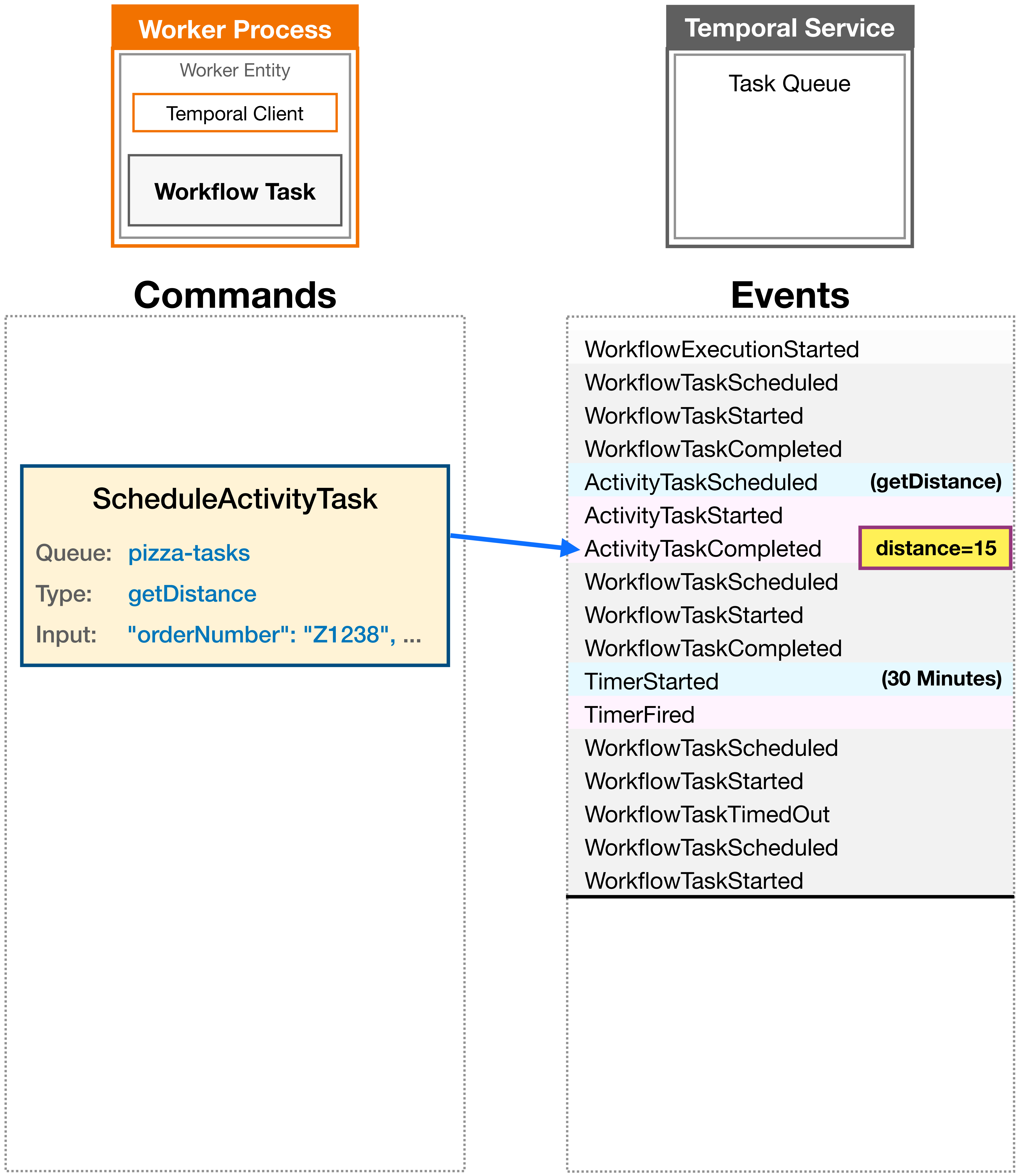
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

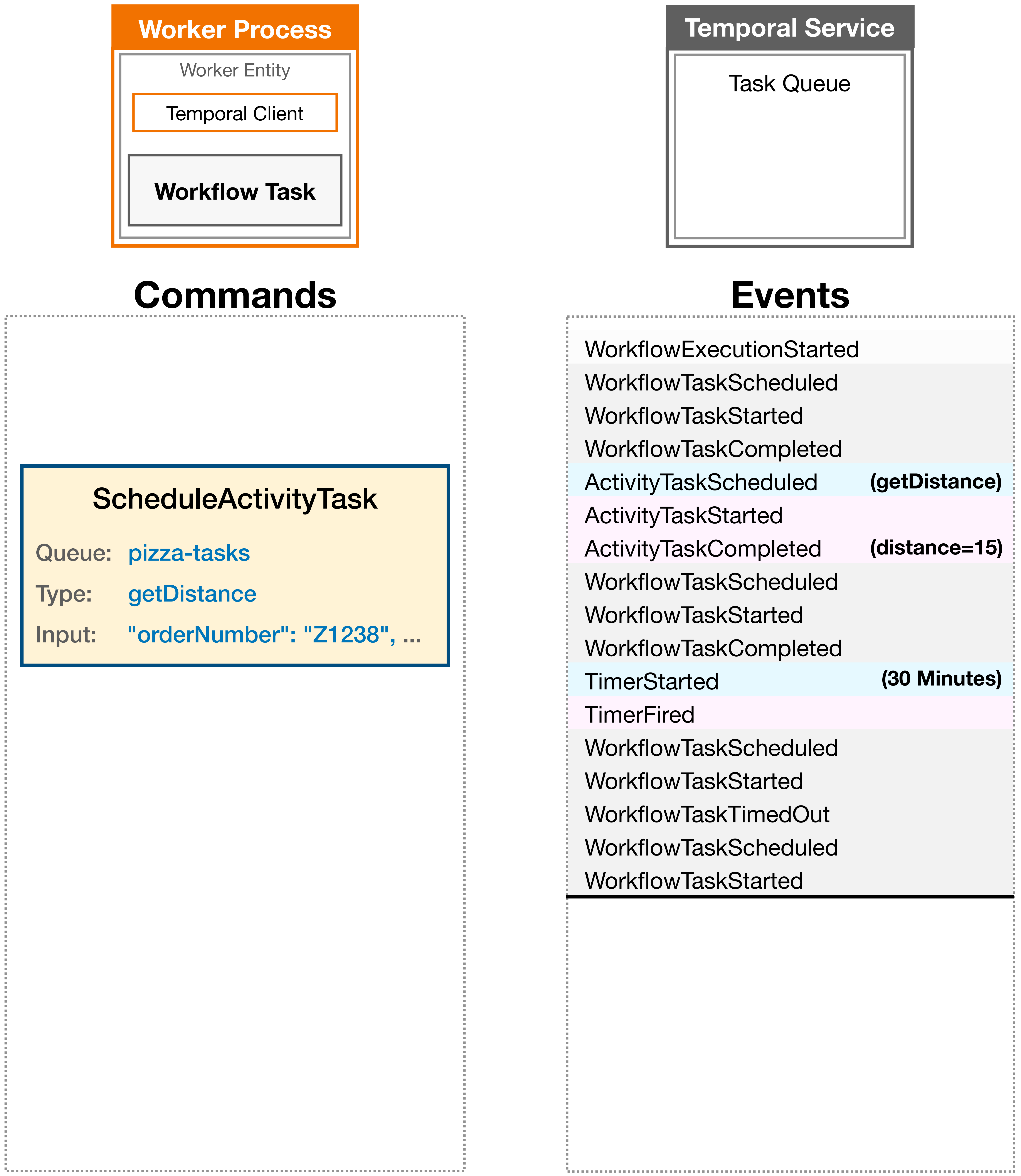


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker assigns 15 to this variable



ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker Process
Worker Entity
Temporal Client
Workflow Task
Temporal Service
Task Queue

Commands

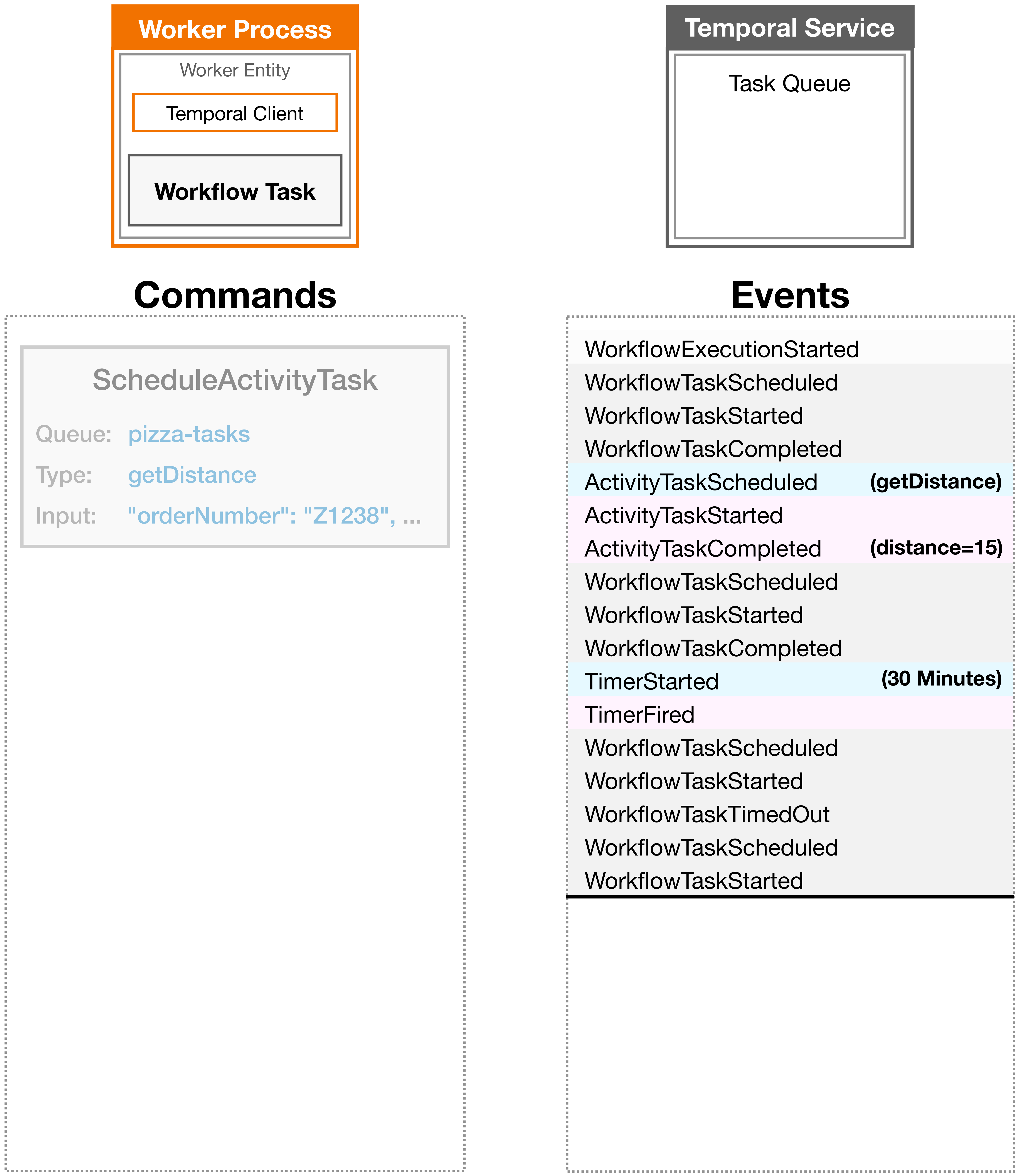
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

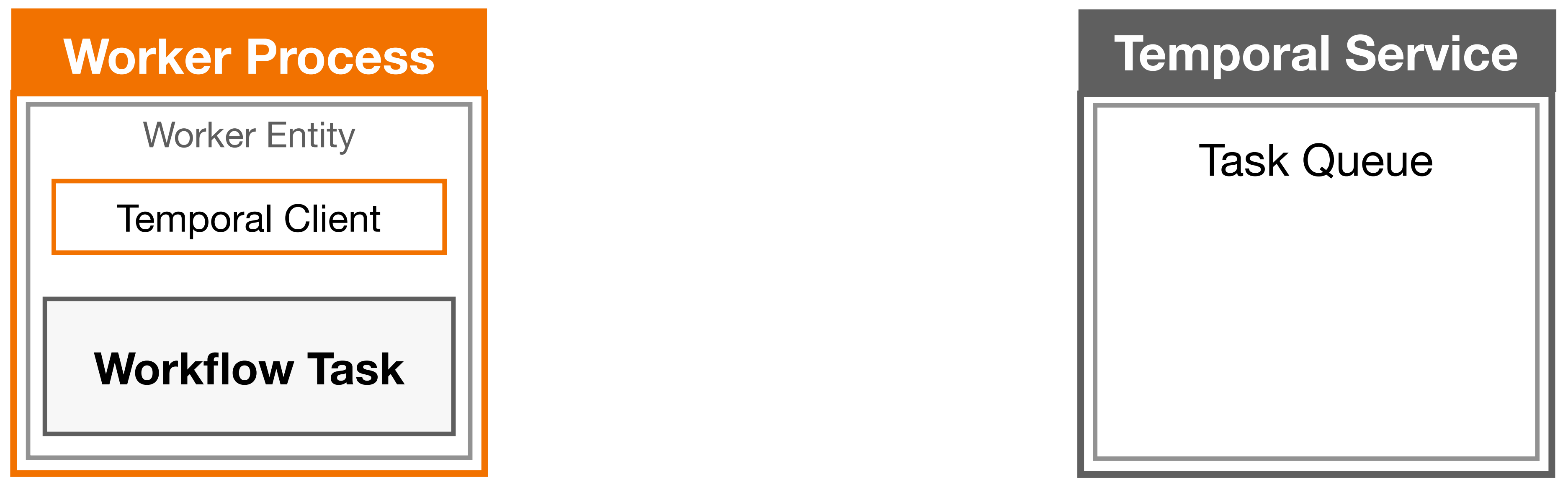
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

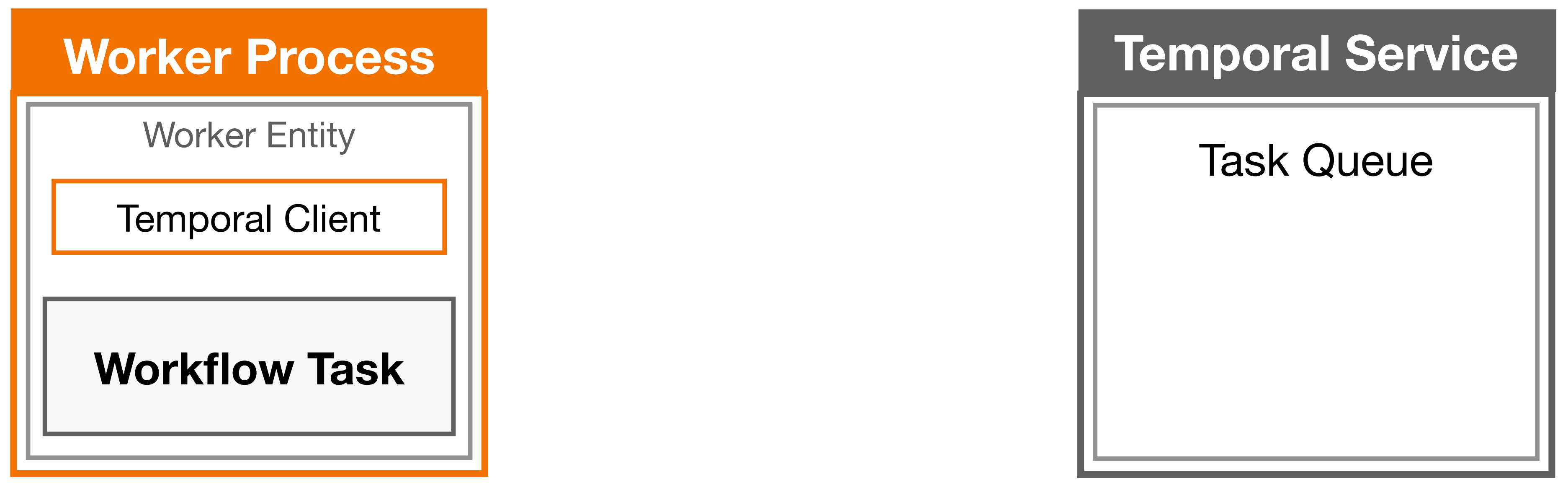
ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

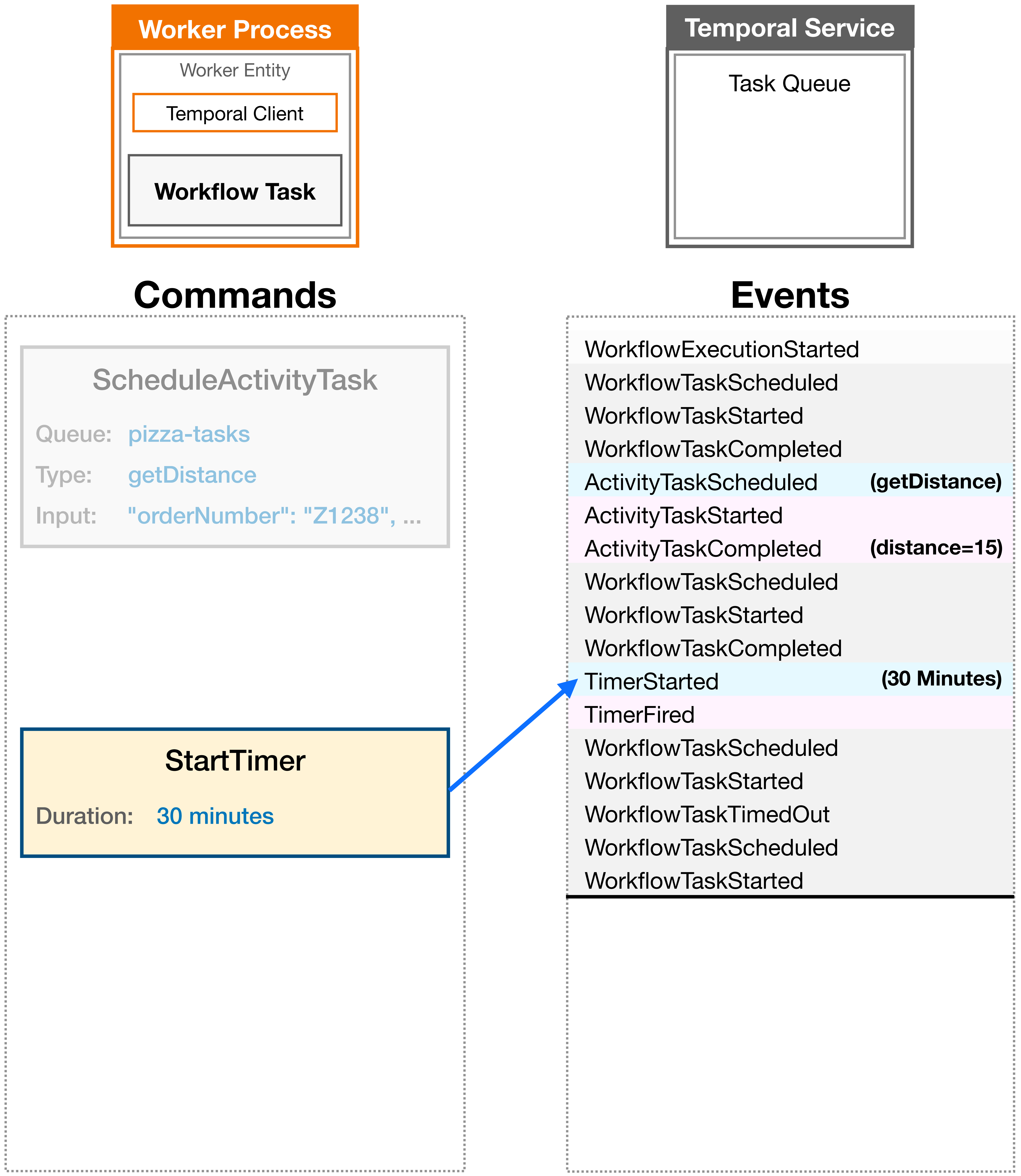
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

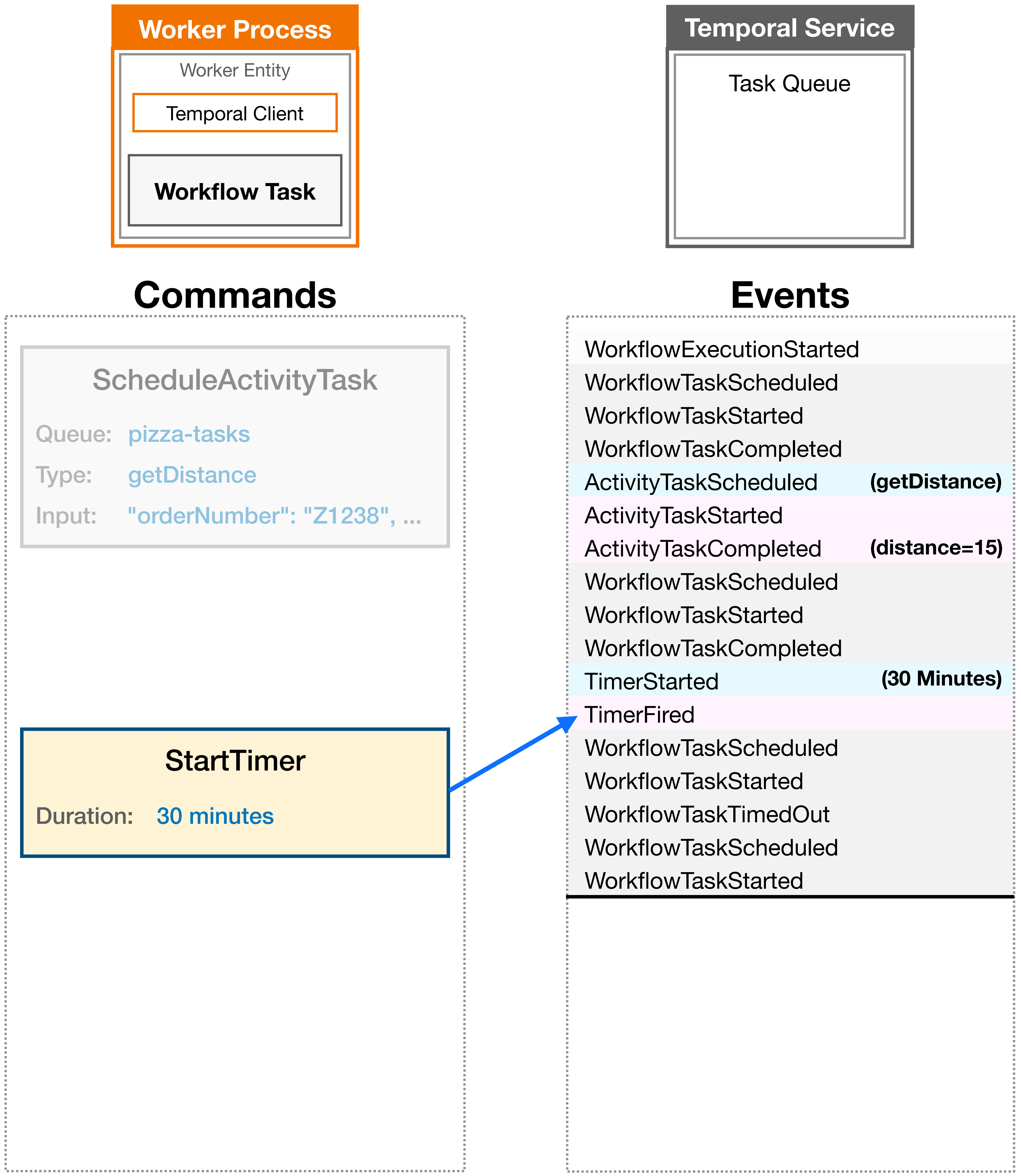
ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

StartTimer
Duration: 30 minutes

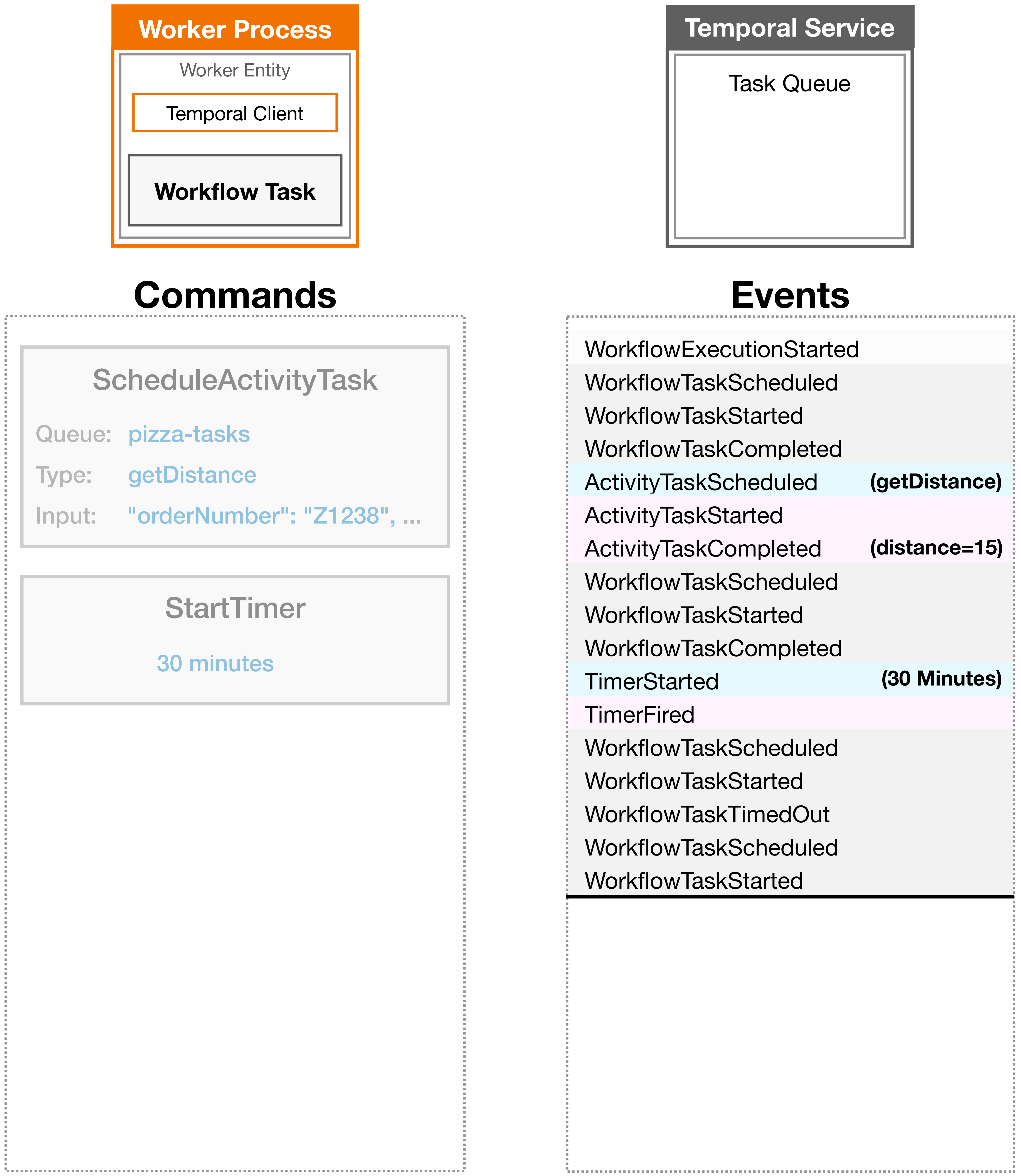
Events

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled          (getDistance)
ActivityTaskStarted
ActivityTaskCompleted          (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted                   (30 Minutes)
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

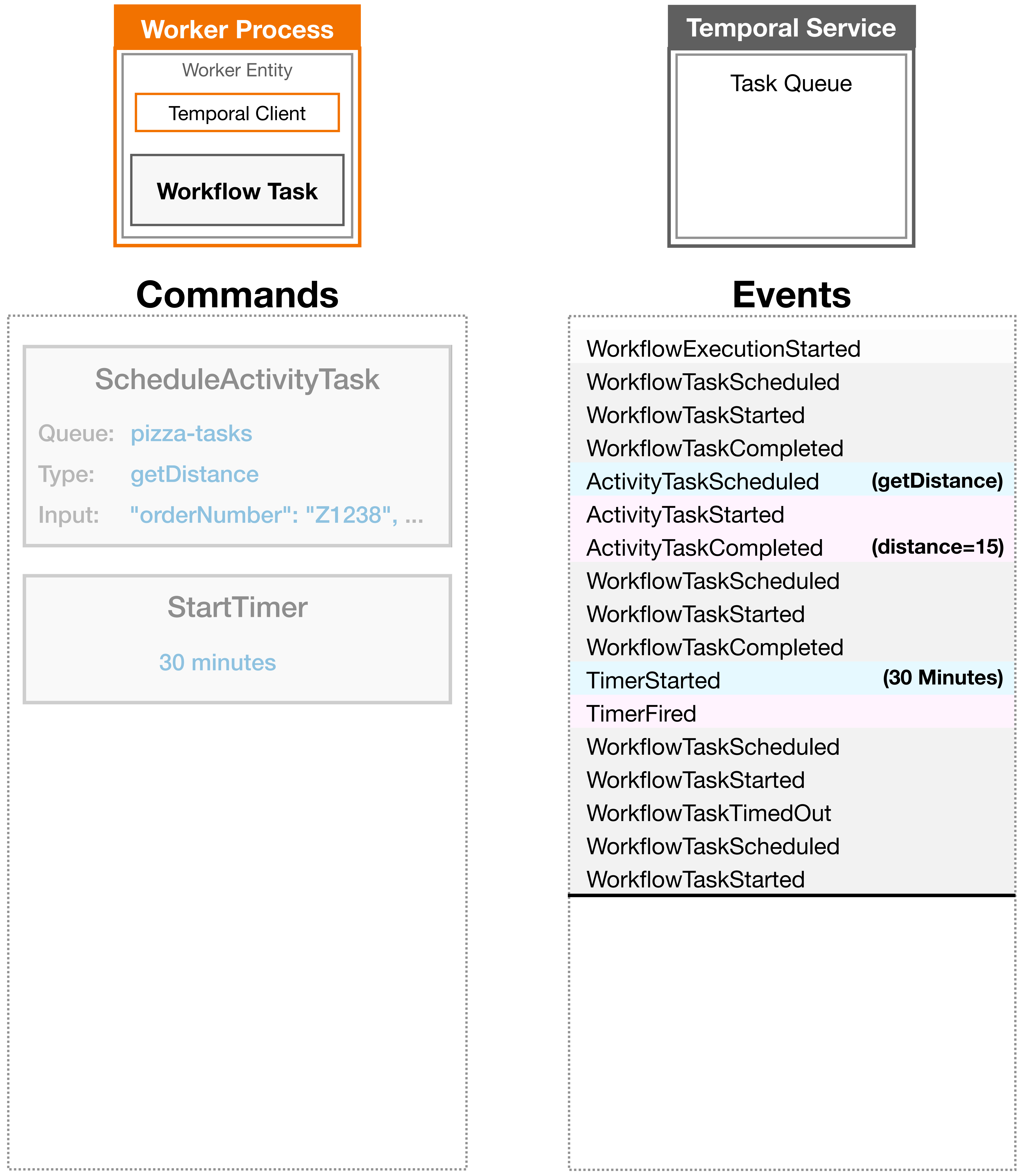
ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

StartTimer
30 minutes

Events

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled        (getDistance)
ActivityTaskStarted
ActivityTaskCompleted        (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted                 (30 Minutes)
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

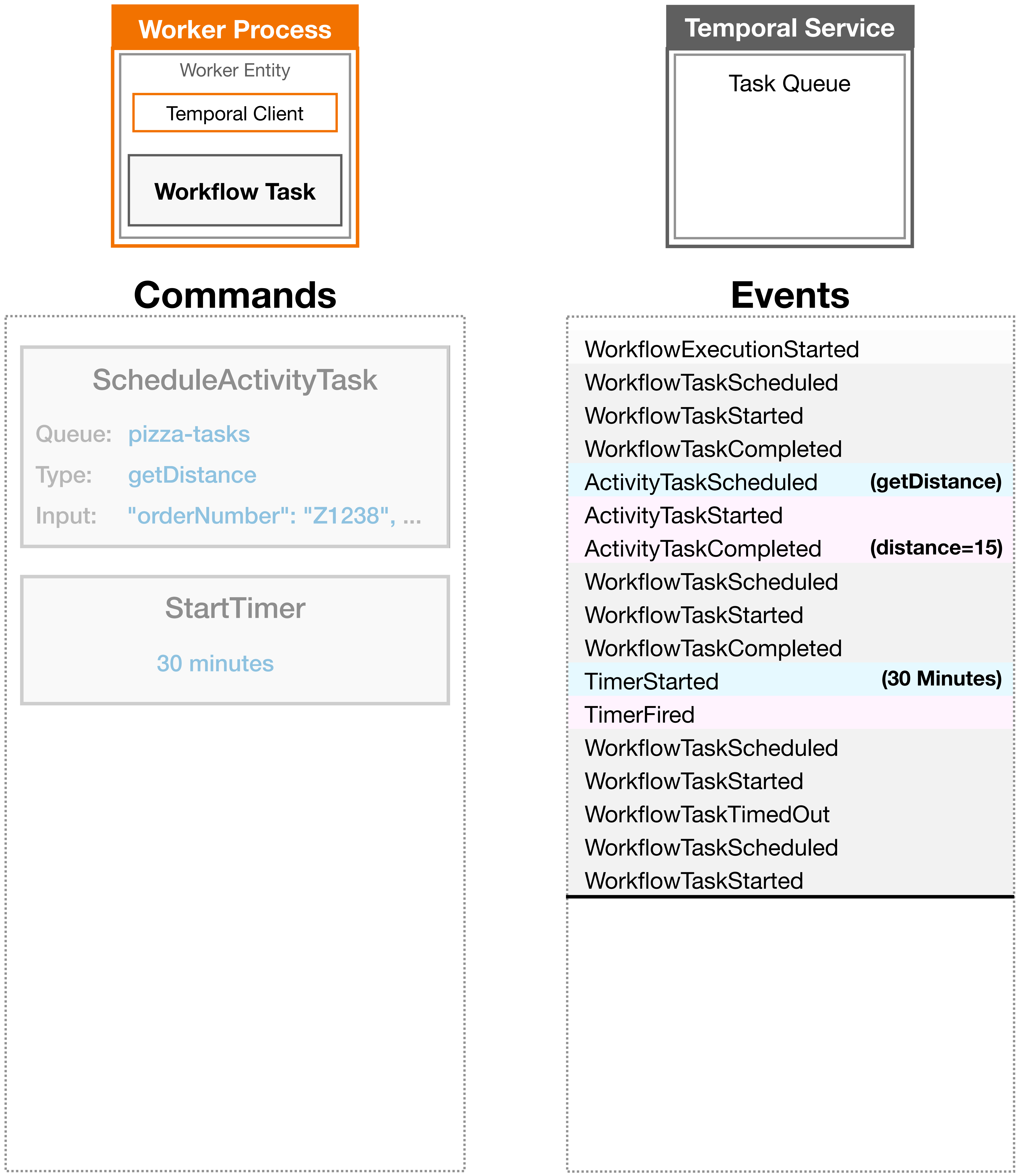
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

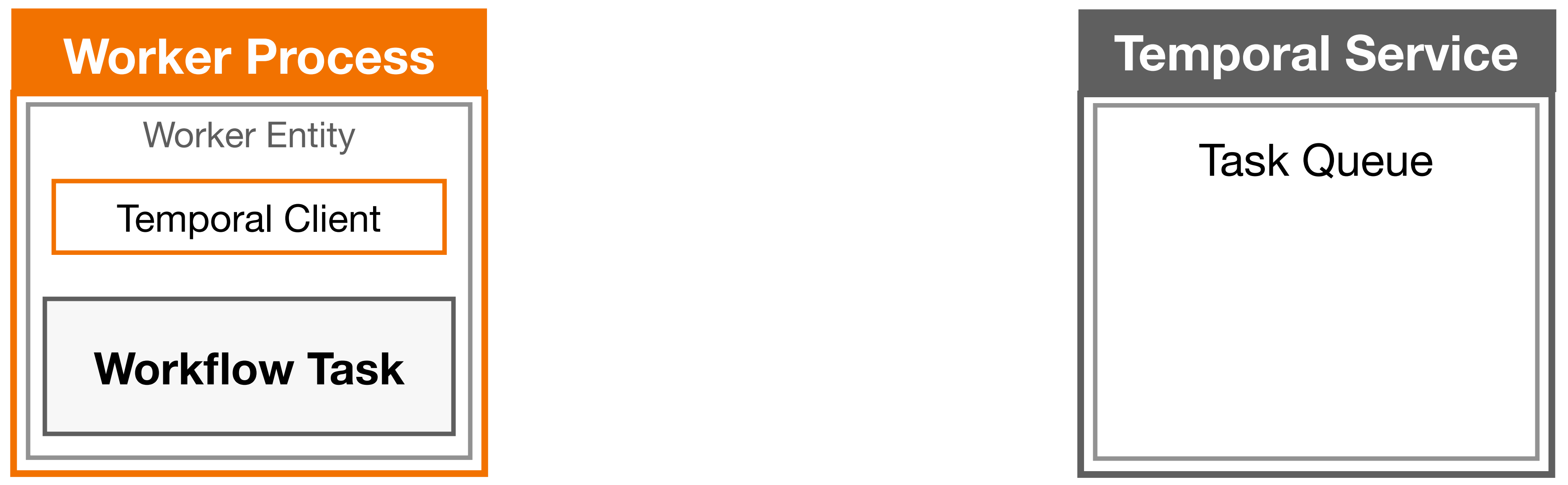
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

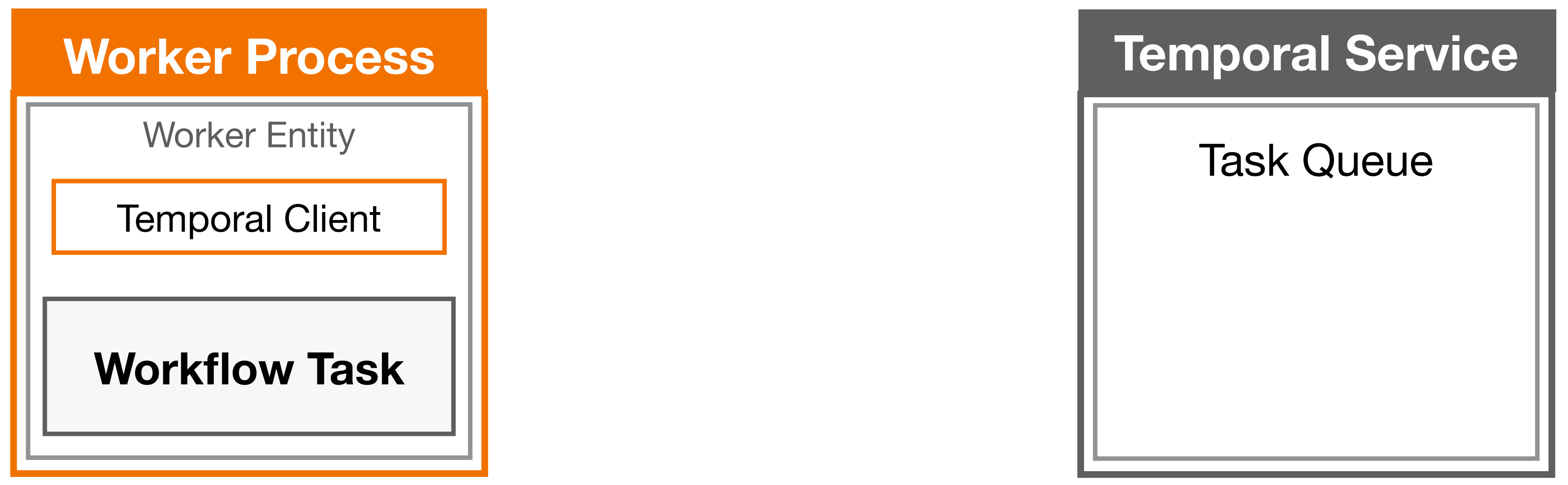
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

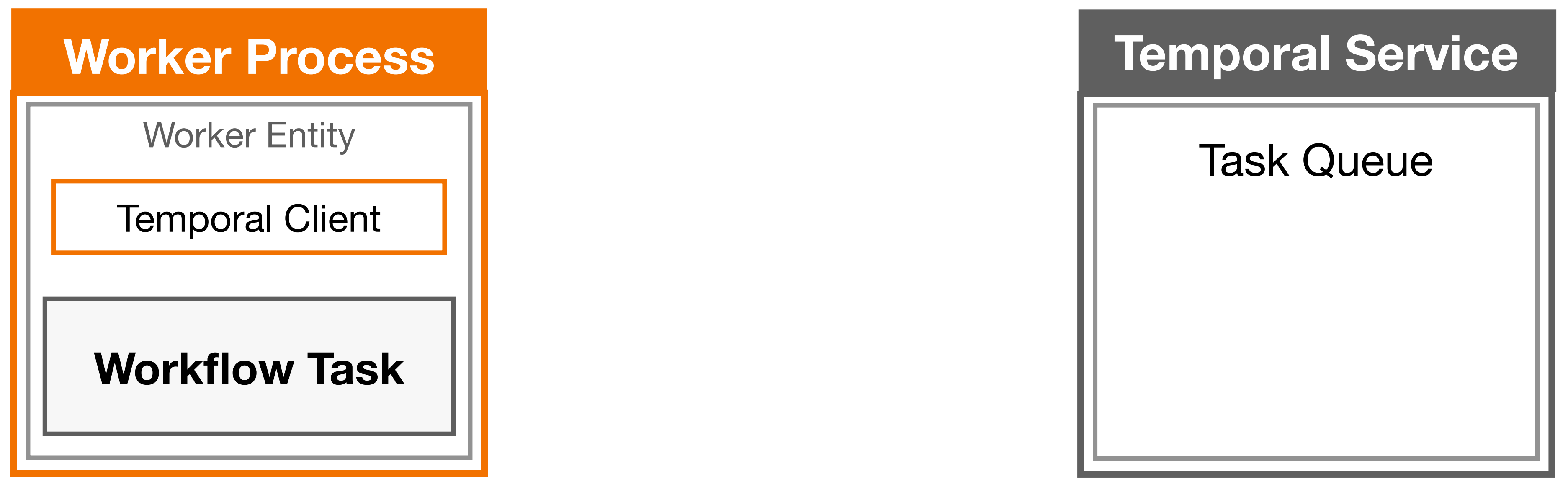
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

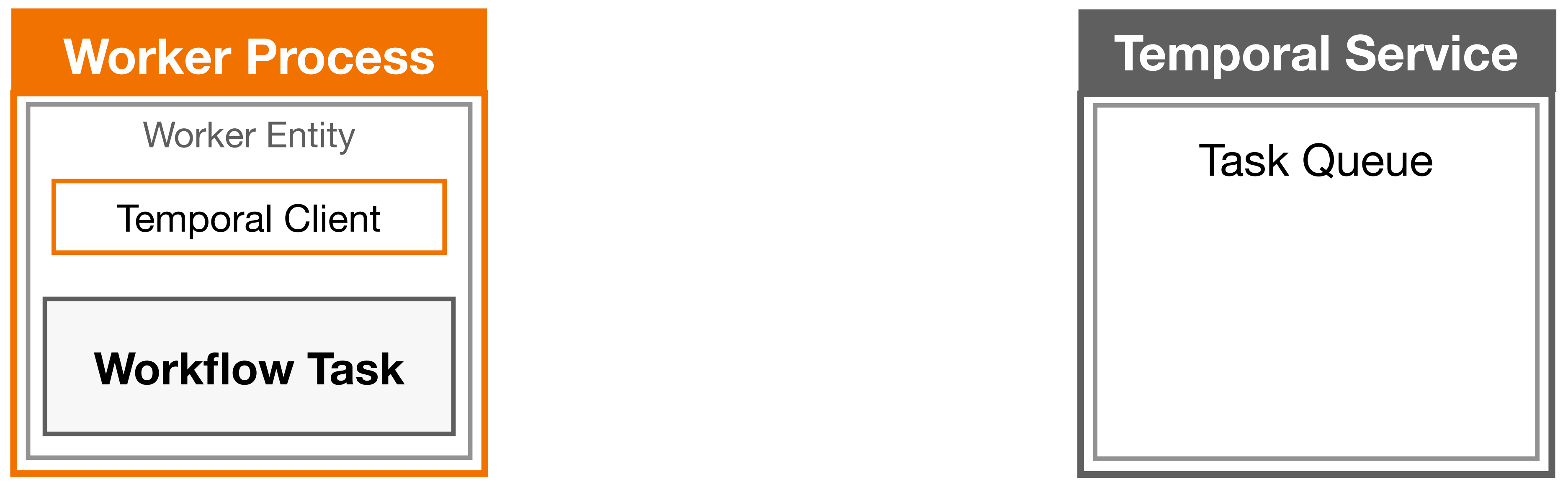
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

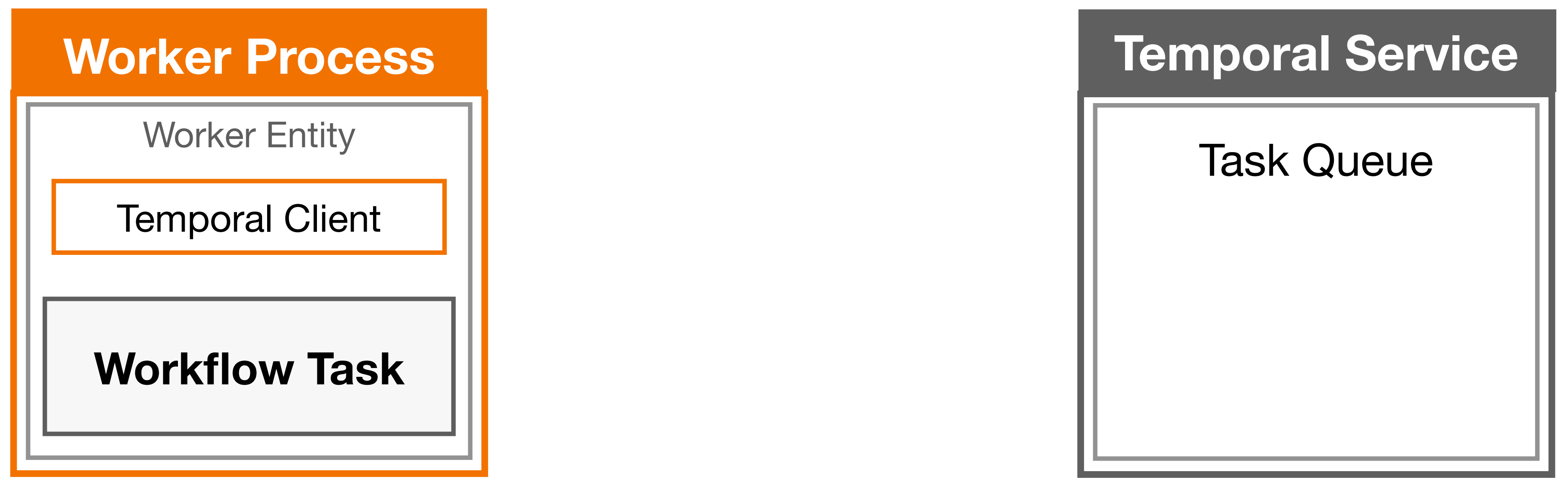
StartTimer

30 minutes

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

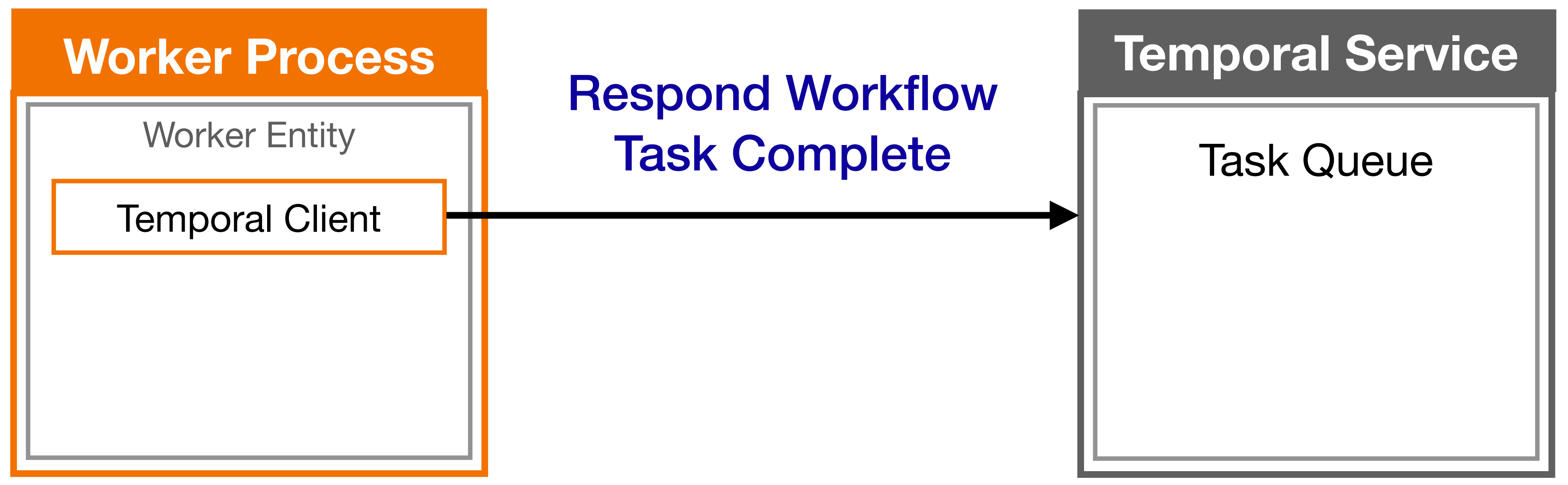
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

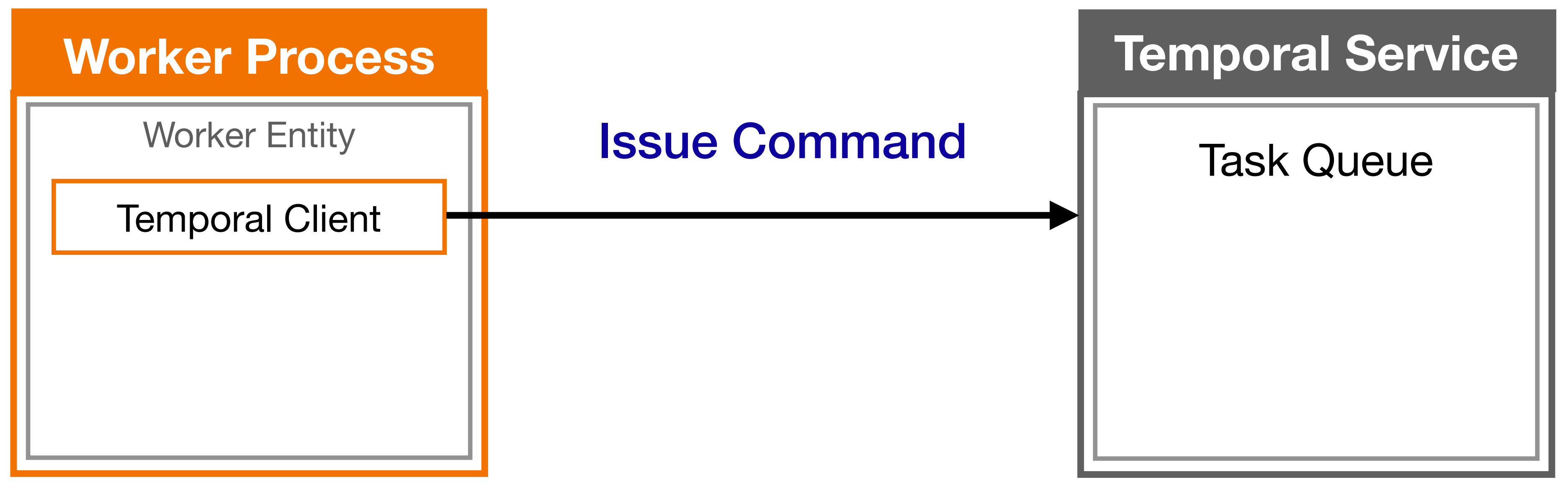
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"orderNumber": "Z1238", ...

StartTimer

30 minutes

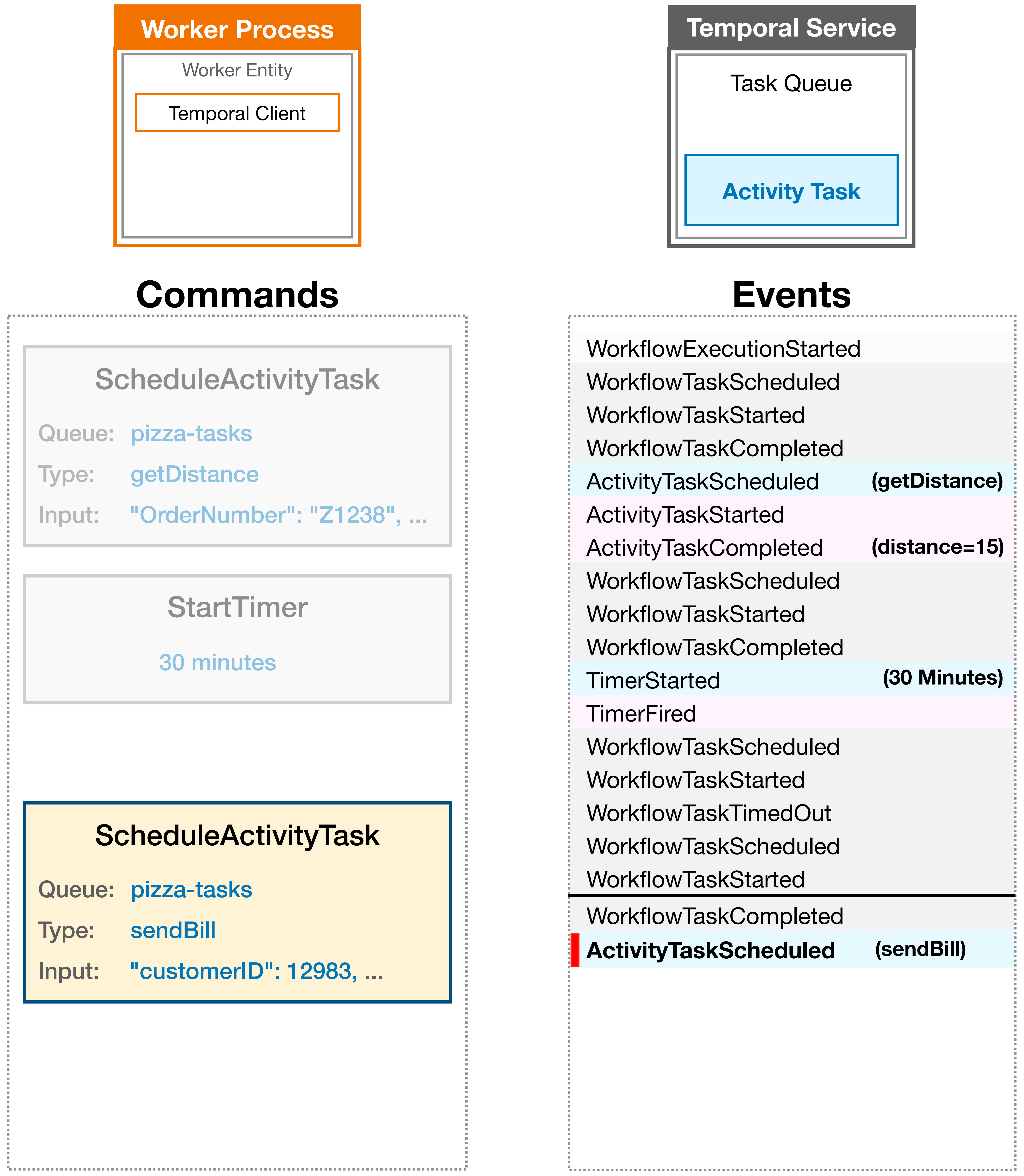
ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

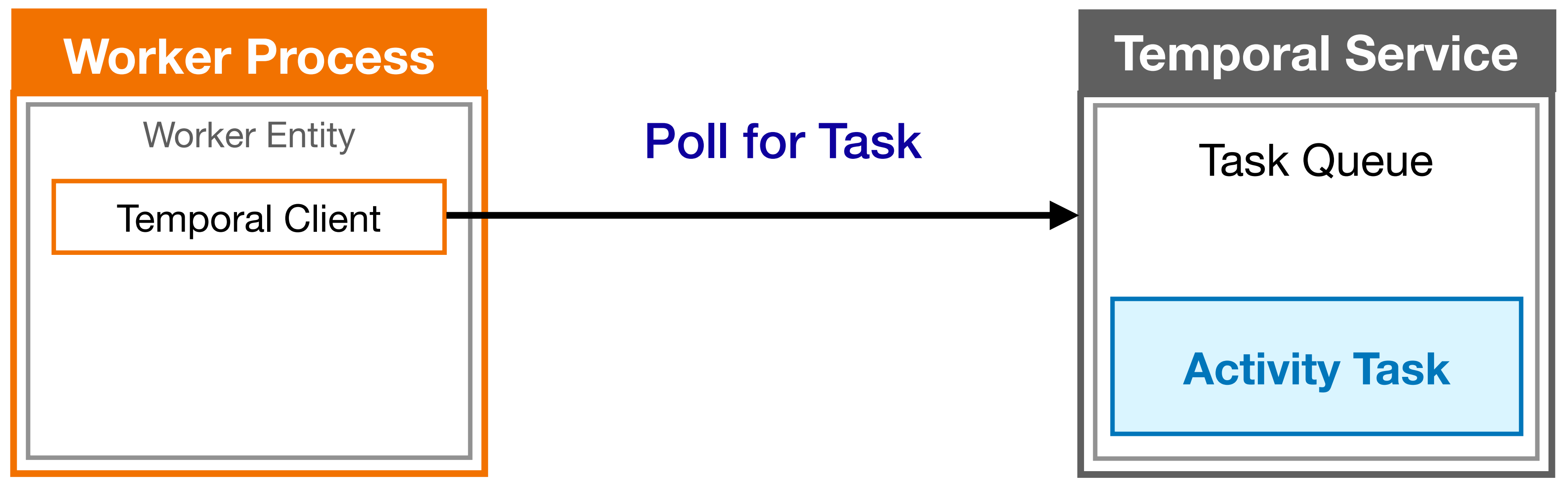
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

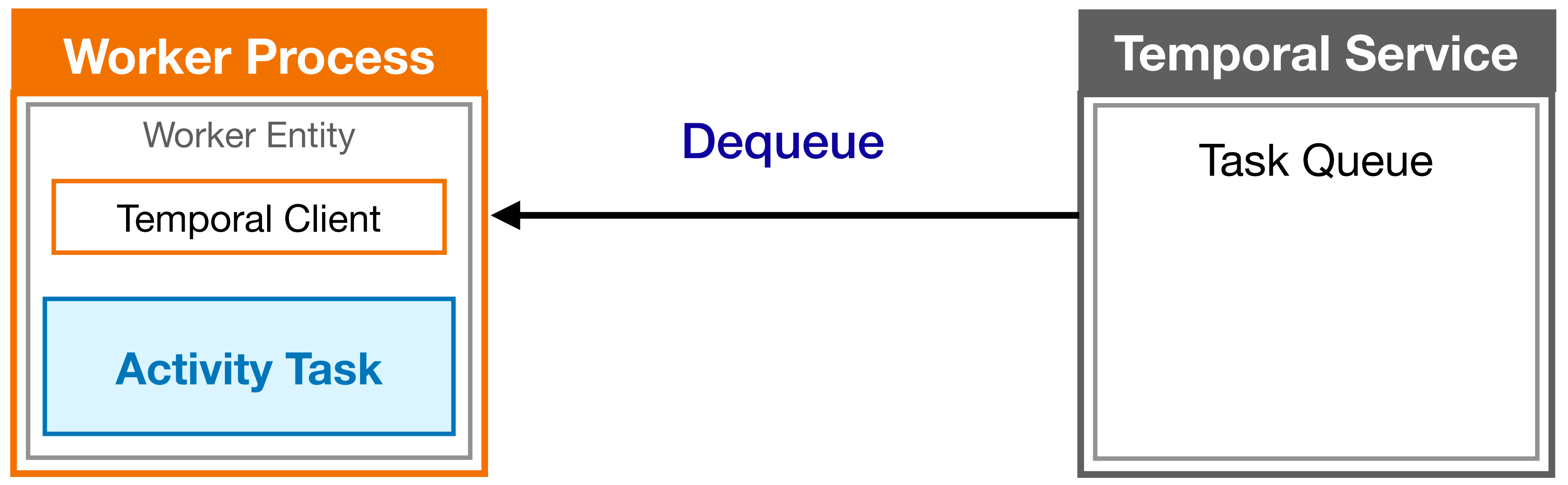
ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

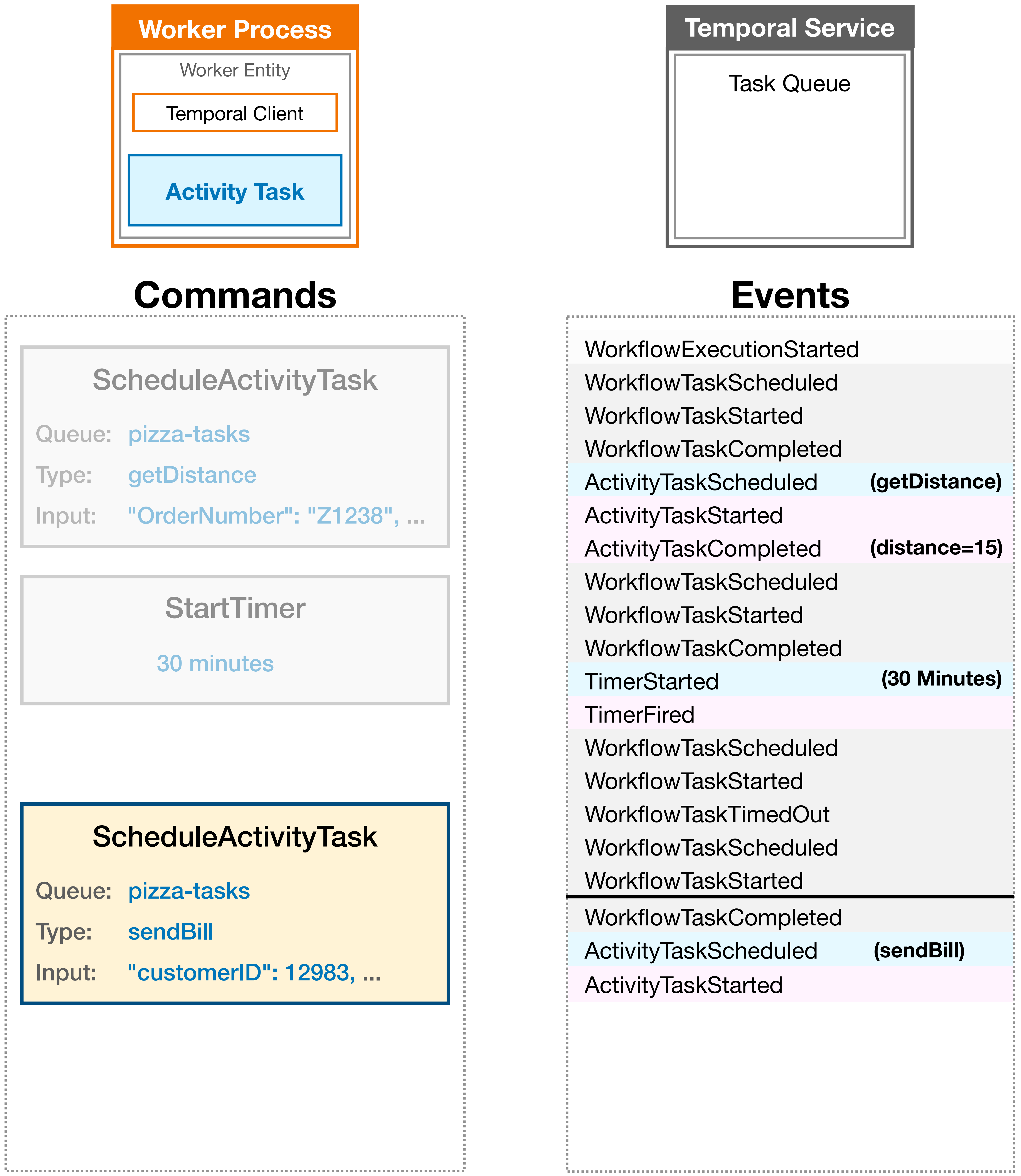
ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



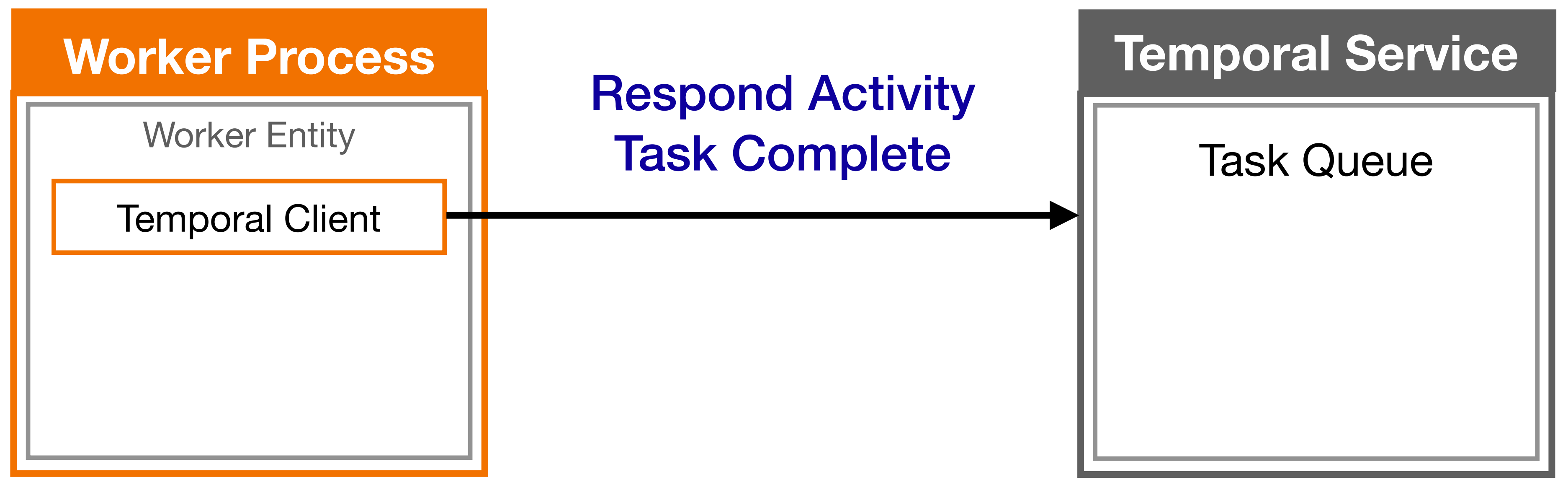
ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled        (getDistance)
ActivityTaskStarted
ActivityTaskCompleted        (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted                 (30 Minutes)
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled        (sendBill)
ActivityTaskStarted
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

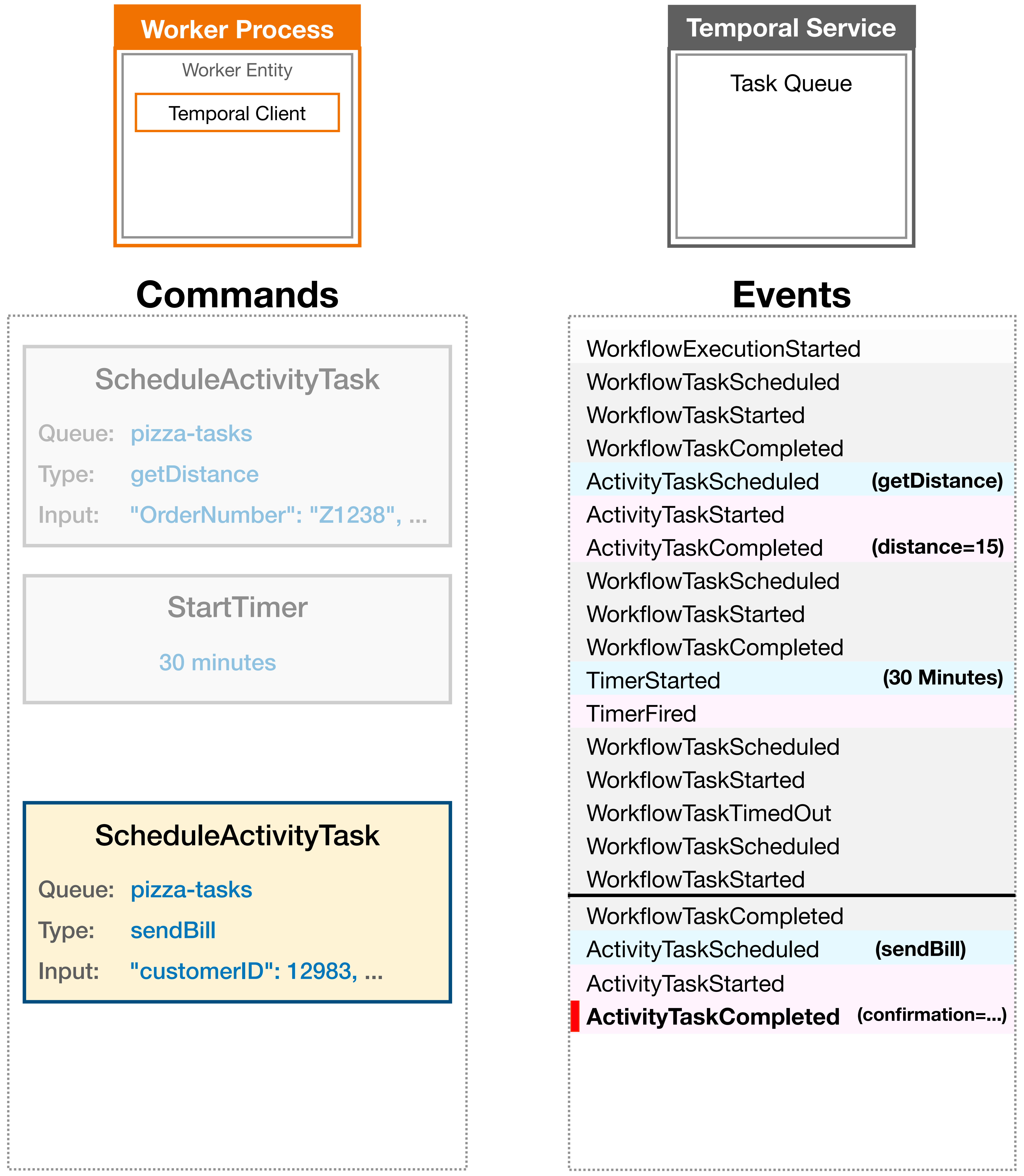
ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

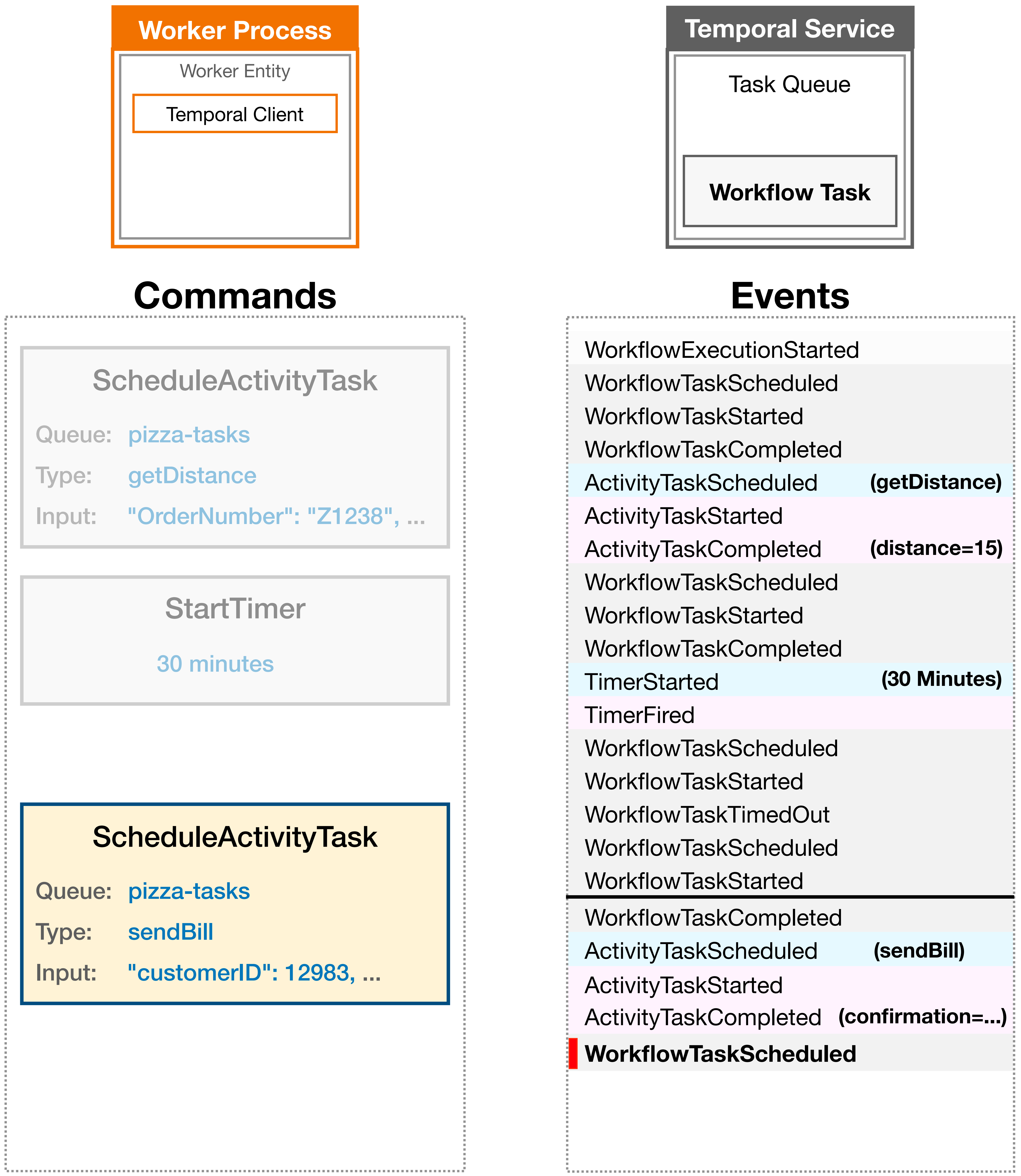
StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

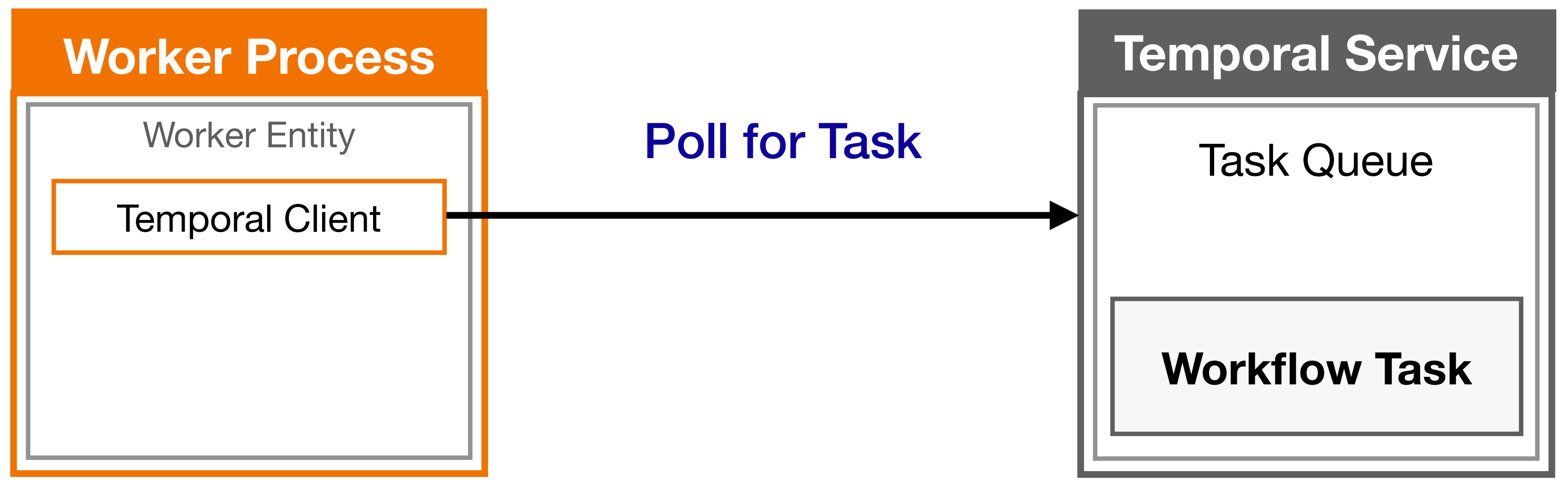
StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

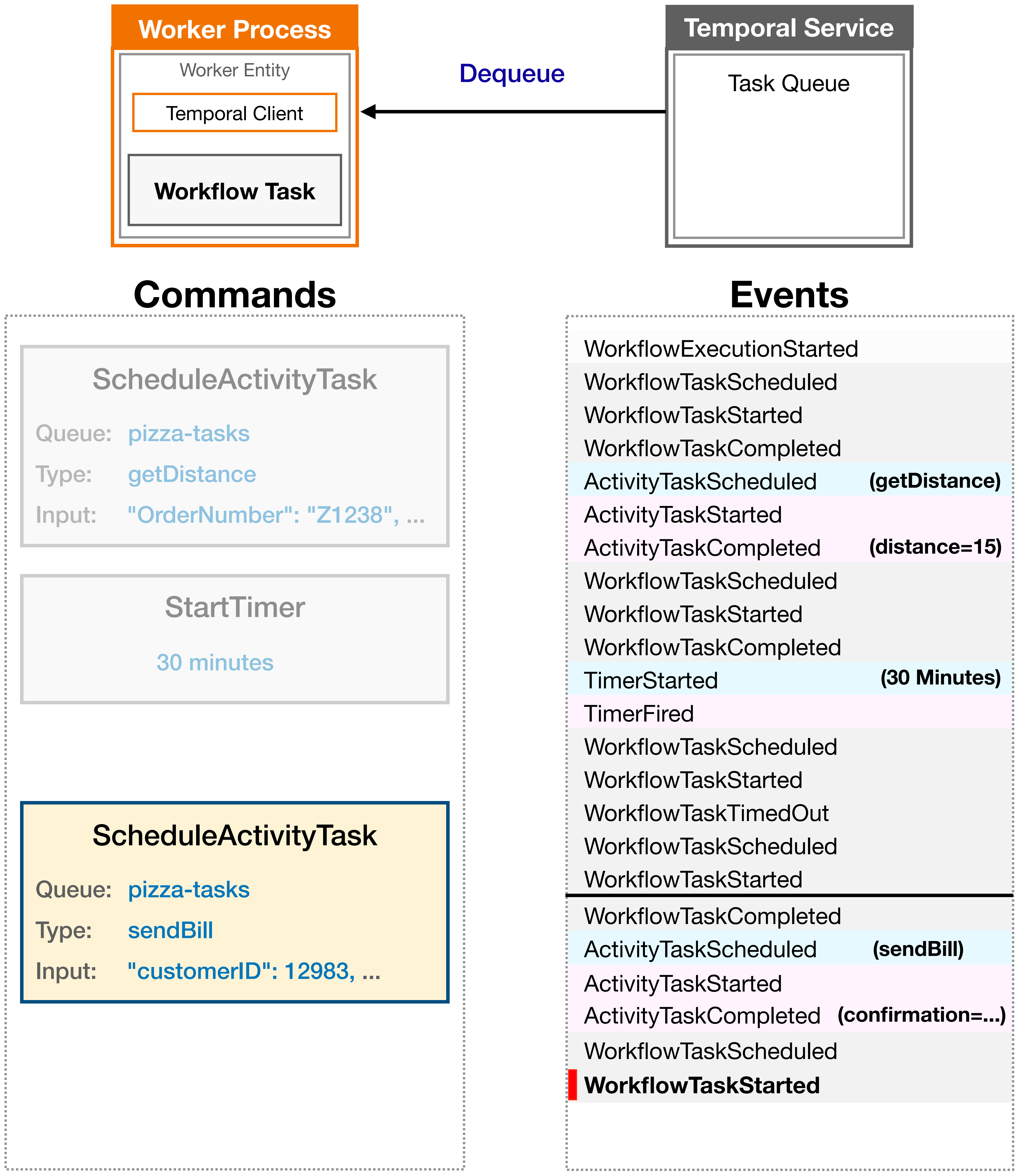
ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

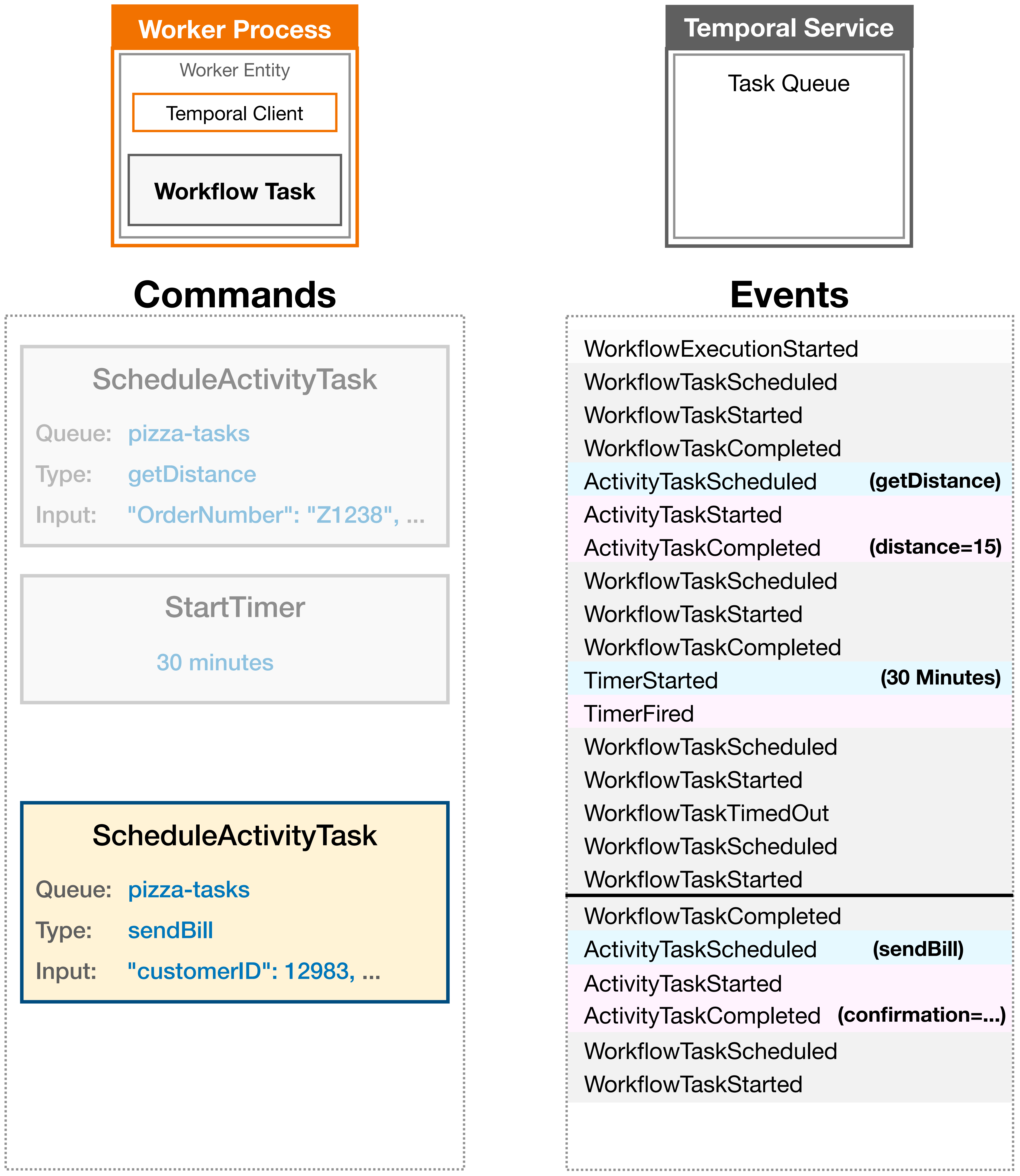
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

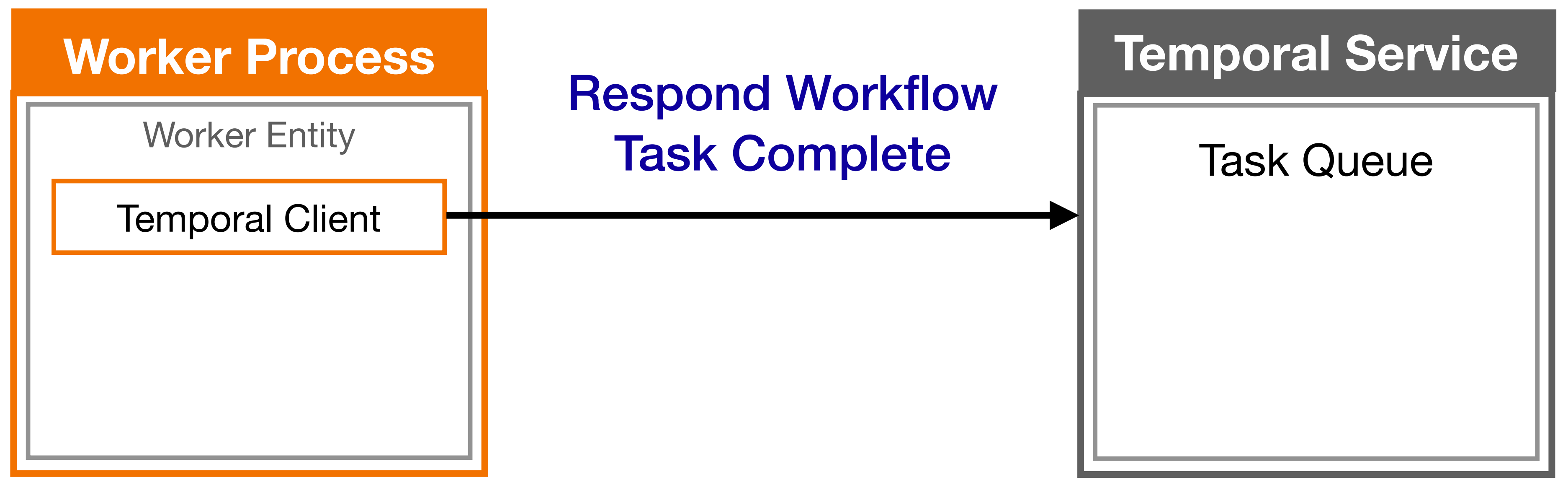
Events

```
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled      (getDistance)
ActivityTaskStarted
ActivityTaskCompleted      (distance=15)
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted               (30 Minutes)
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled      (sendBill)
ActivityTaskStarted
ActivityTaskCompleted      (confirmation=...)
WorkflowTaskScheduled
WorkflowTaskStarted
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker Process
Worker Entity
Temporal Client
Temporal Service
Task Queue

Commands

ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

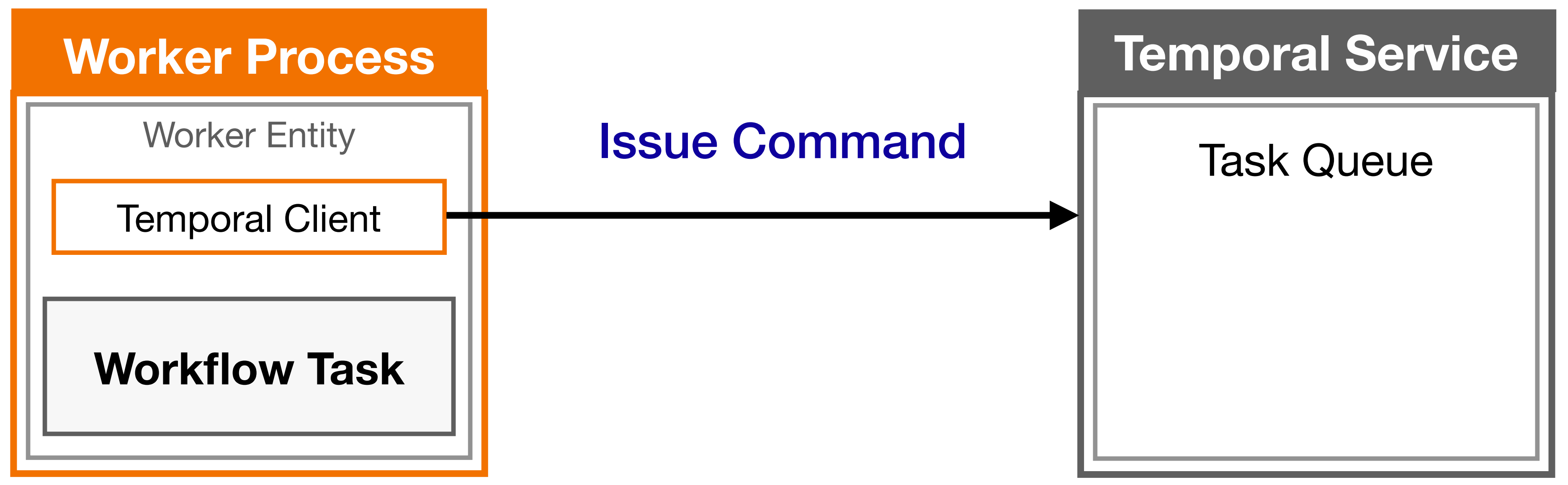
StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue:	pizza-tasks
Type:	getDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue:	pizza-tasks
Type:	sendBill
Input:	"customerID": 12983, ...

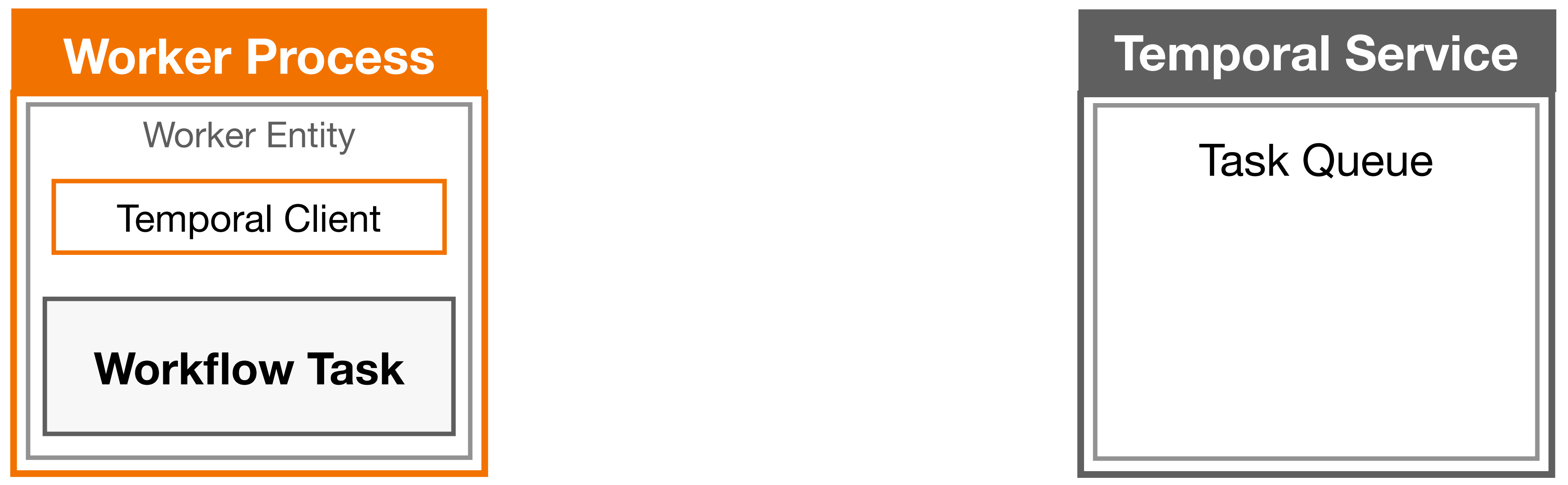
CompleteWorkflowExecution

Result: "confirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: sendBill
Input: "CustomerID": 12983, ...

CompleteWorkflowExecution
Result: "ConfirmationNumber": "TPD-26074139"

Events

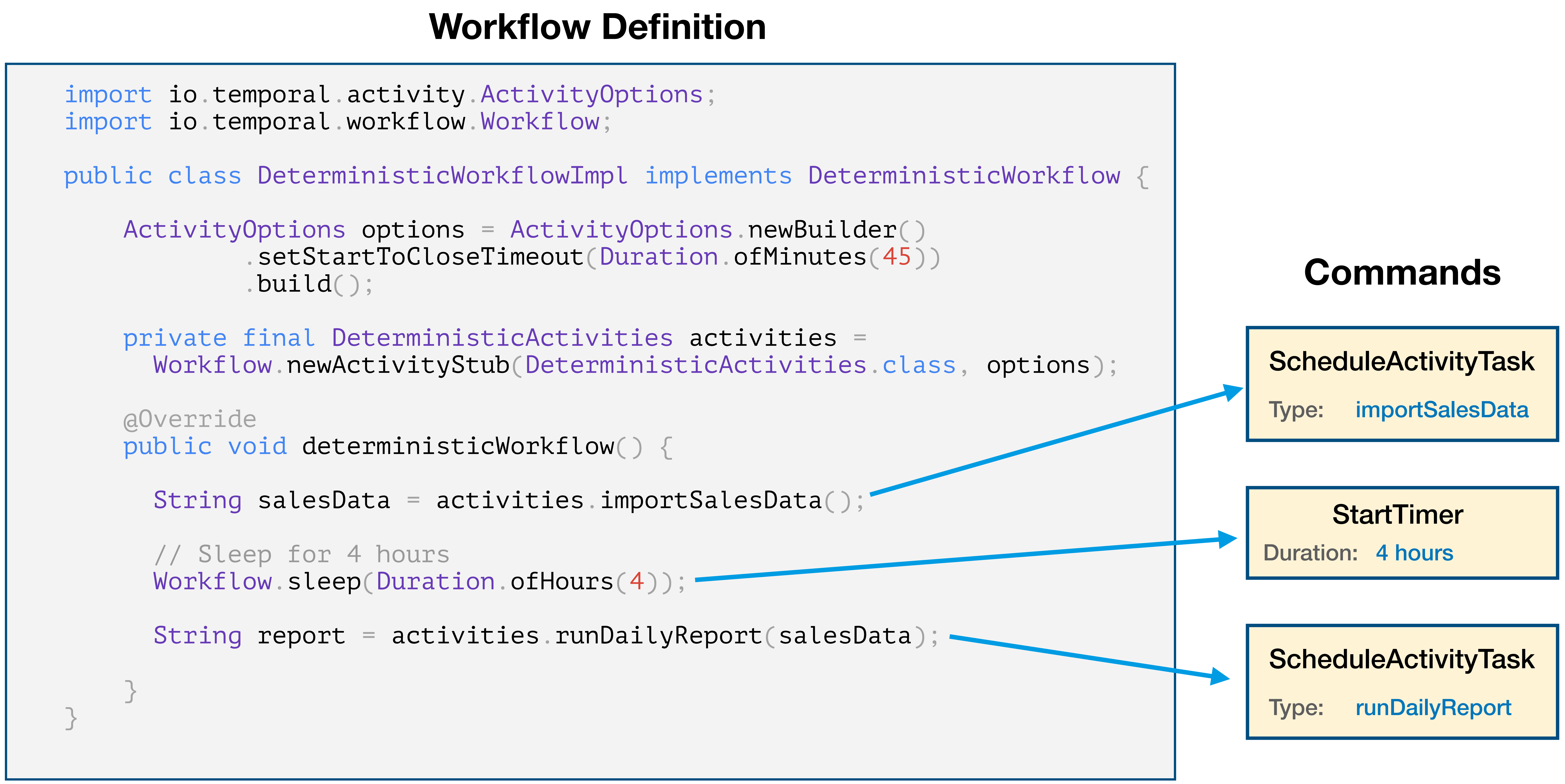
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted

WorkflowTaskCompleted
ActivityTaskScheduled **(sendBill)**
ActivityTaskStarted
ActivityTaskCompleted **(confirmation=...)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
WorkflowExecutionCompleted

Why Temporal Requires Determinism for Workflows

Workflow Definition

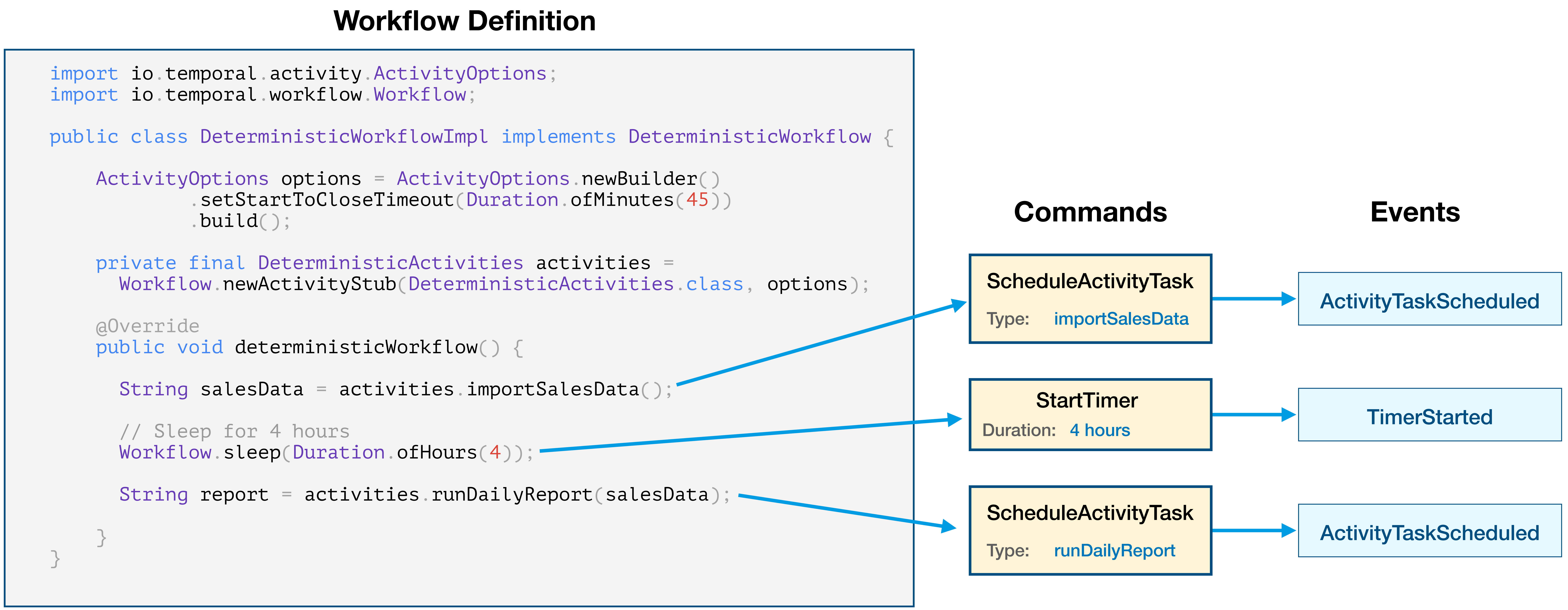
```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```



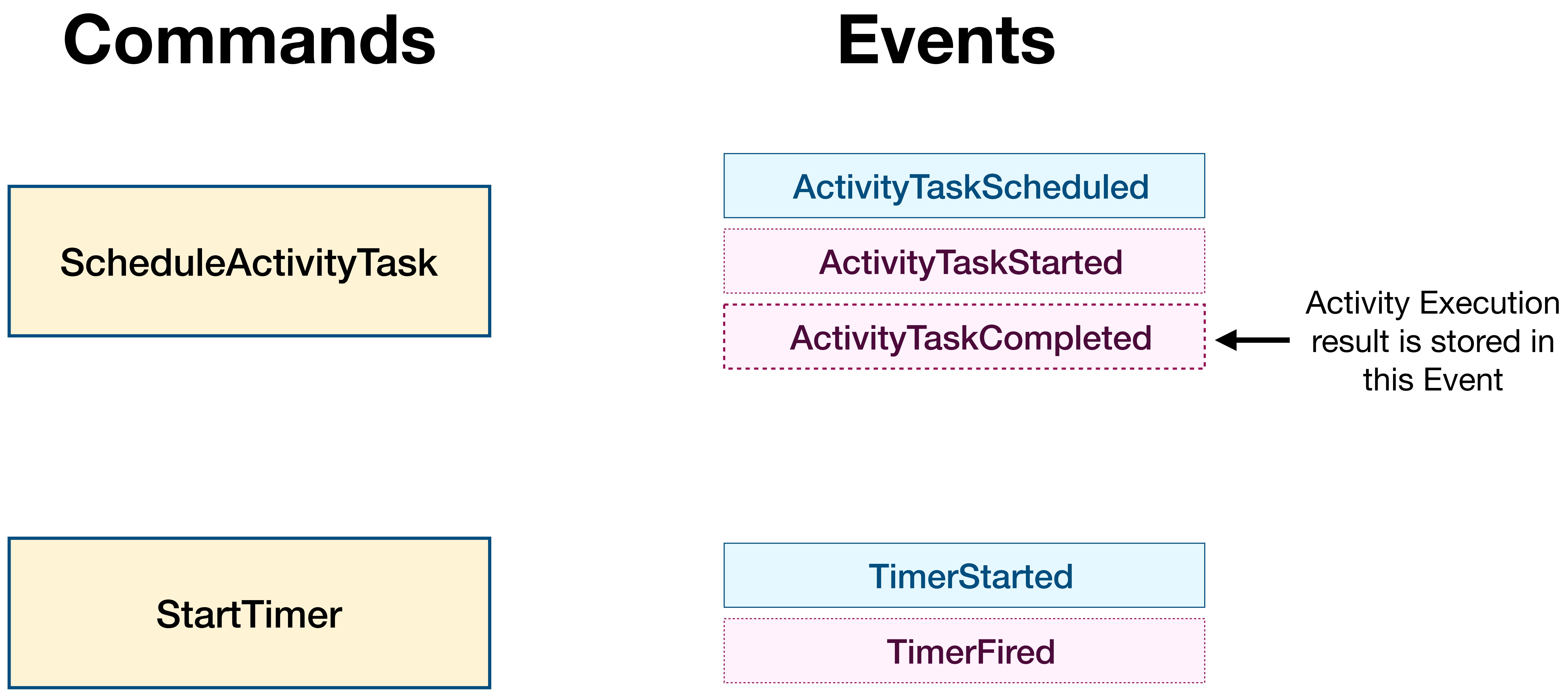
Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Commands



```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```


Commands
Events
ScheduleActivityTask
ActivityTaskScheduled
ActivityTaskStarted
ActivityTaskCompleted
Activity Execution result is stored in this Event
StartTimer
TimerStarted
TimerFired

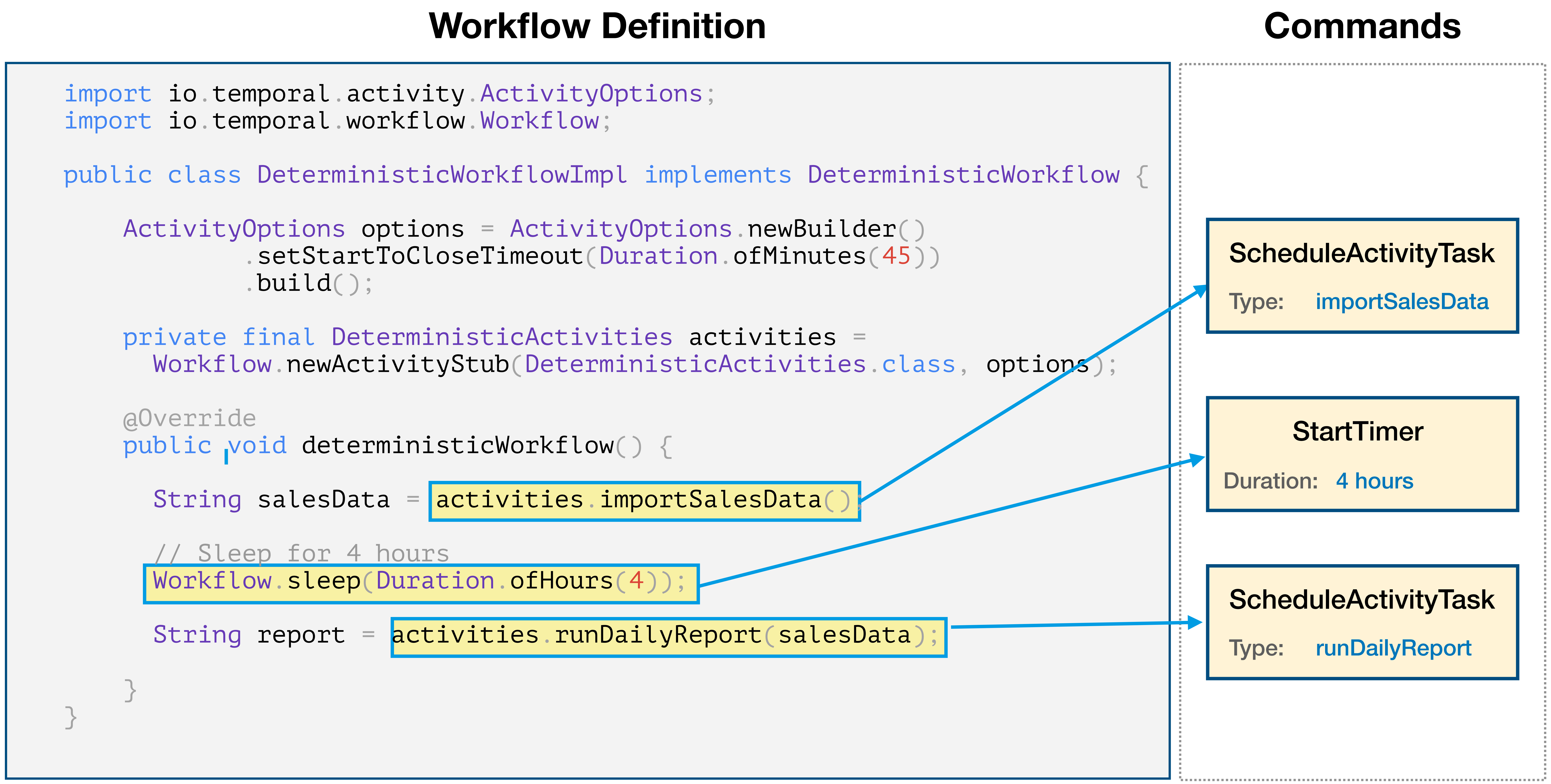
Deterministic Workflows:

- A Workflow is deterministic if every execution of its Workflow Definition:
  - produces the same Commands
  - in the same sequence
  - given the same input

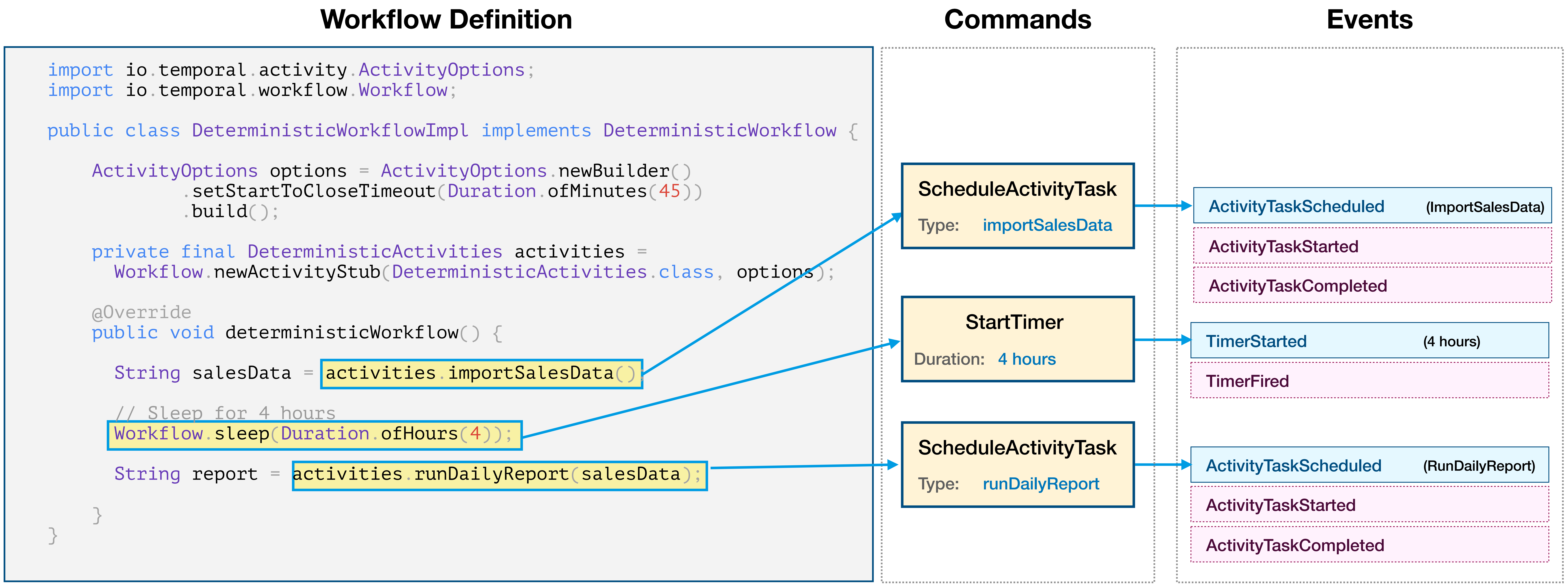
Temporal's ability to guarantee durable execution
of your Workflow depends on deterministic Workflows.

Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

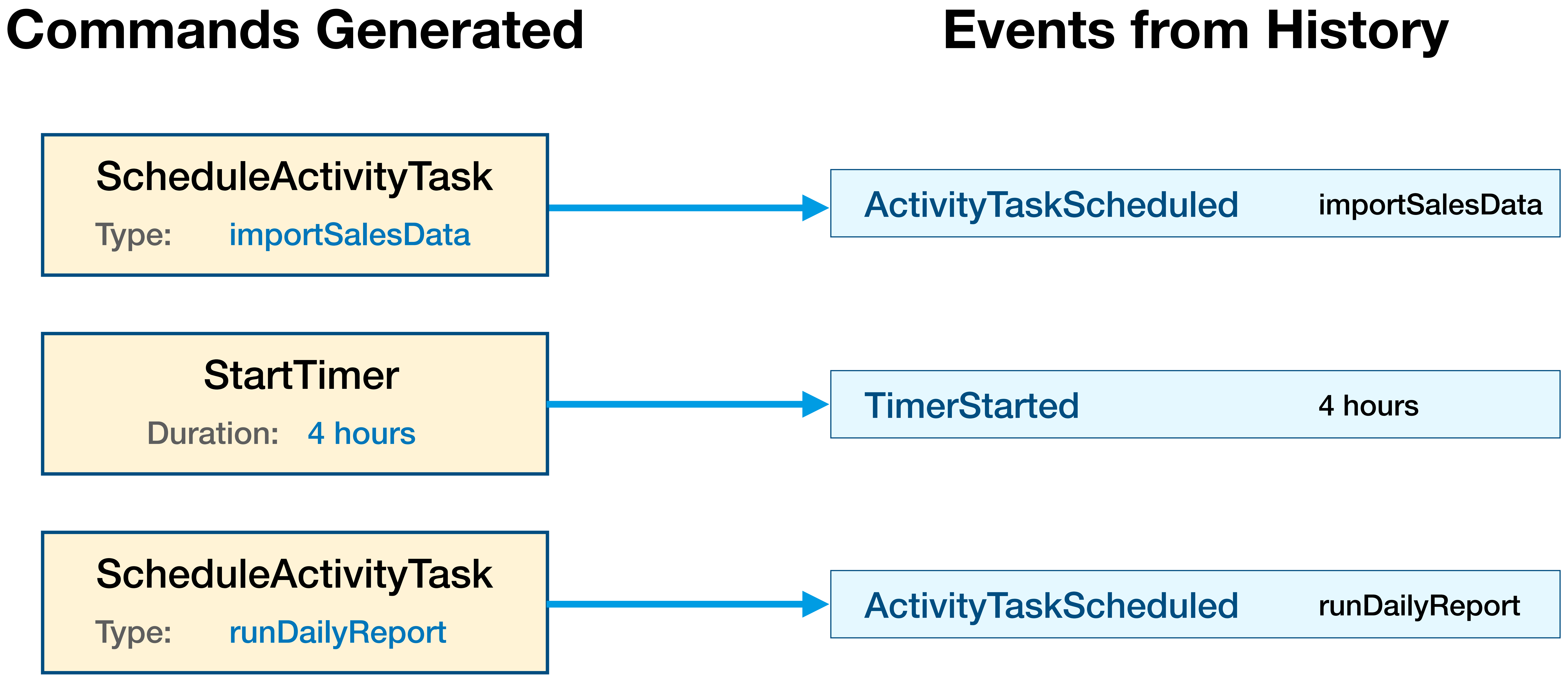


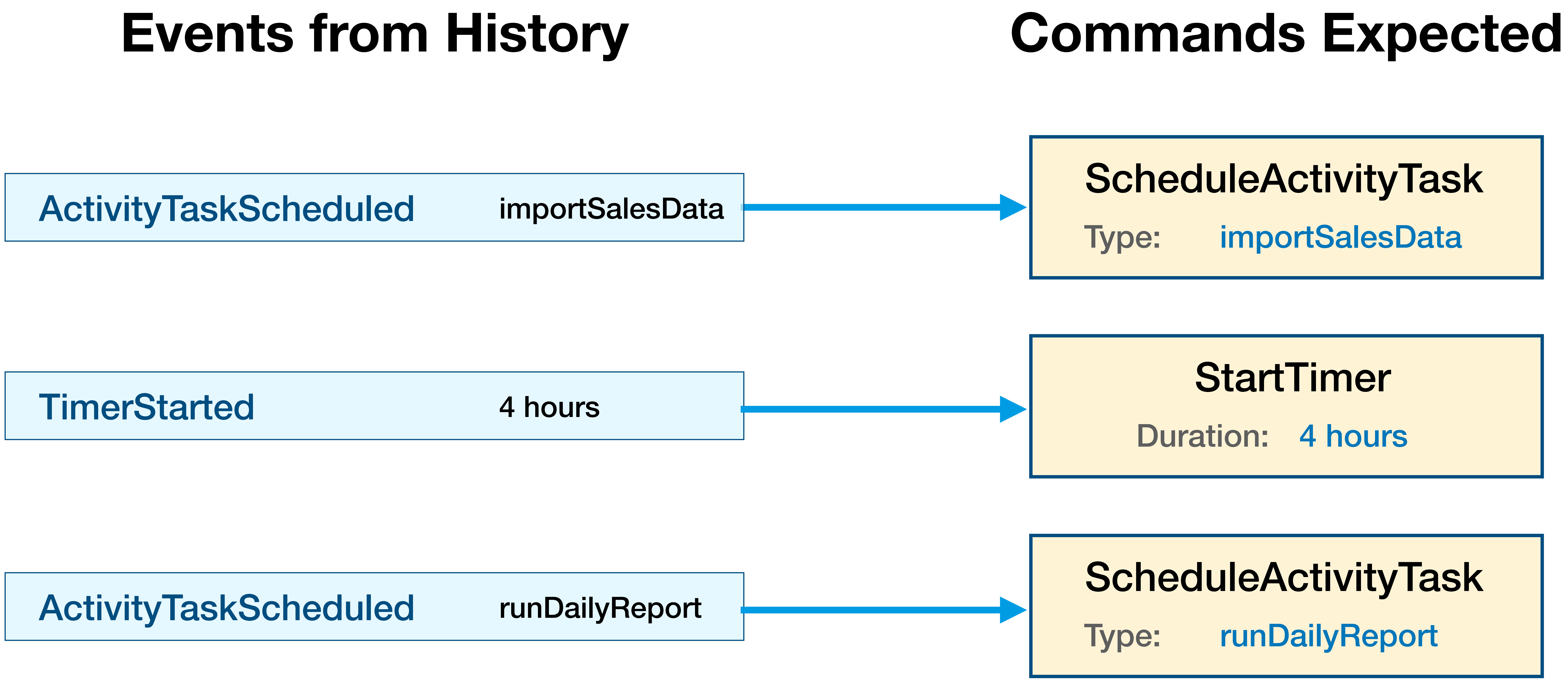
```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```



Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```


Commands Generated
Events from History
ScheduleActivityTask
Type: importSalesData
ActivityTaskScheduled importSalesData
StartTimer
Duration: 4 hours
TimerStarted 4 hours
ScheduleActivityTask
Type: runDailyReport
ActivityTaskScheduled runDailyReport


Events from History
Commands Expected
ActivityTaskScheduled importSalesData
ScheduleActivityTask
Type: importSalesData
TimerStarted 4 hours
StartTimer
Duration: 4 hours
ActivityTaskScheduled runDailyReport
ScheduleActivityTask
Type: runDailyReport

Example of a Non-Deterministic Workflow

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

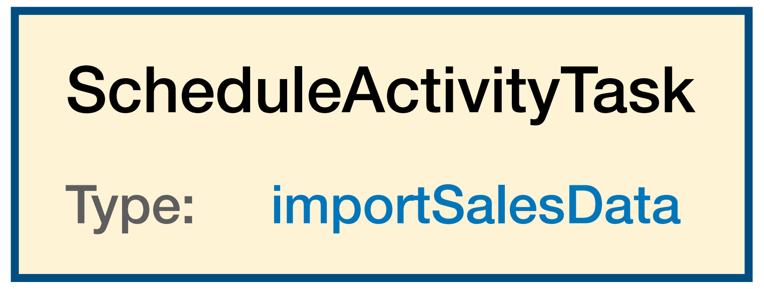
Commands Created

Relevant Events Logged

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created



Relevant Events Logged

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Happens to return 84 during this execution

Commands Created

ScheduleActivityTask
Type: importSalesData

Relevant Events Logged

ActivityTaskScheduled	(importSalesData)
ActivityTaskStarted	
ActivityTaskCompleted	

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

StartTimer
Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

StartTimer
Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Worker crashes here

Commands Created

ScheduleActivityTask
Type: importSalesData

StartTimer
Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

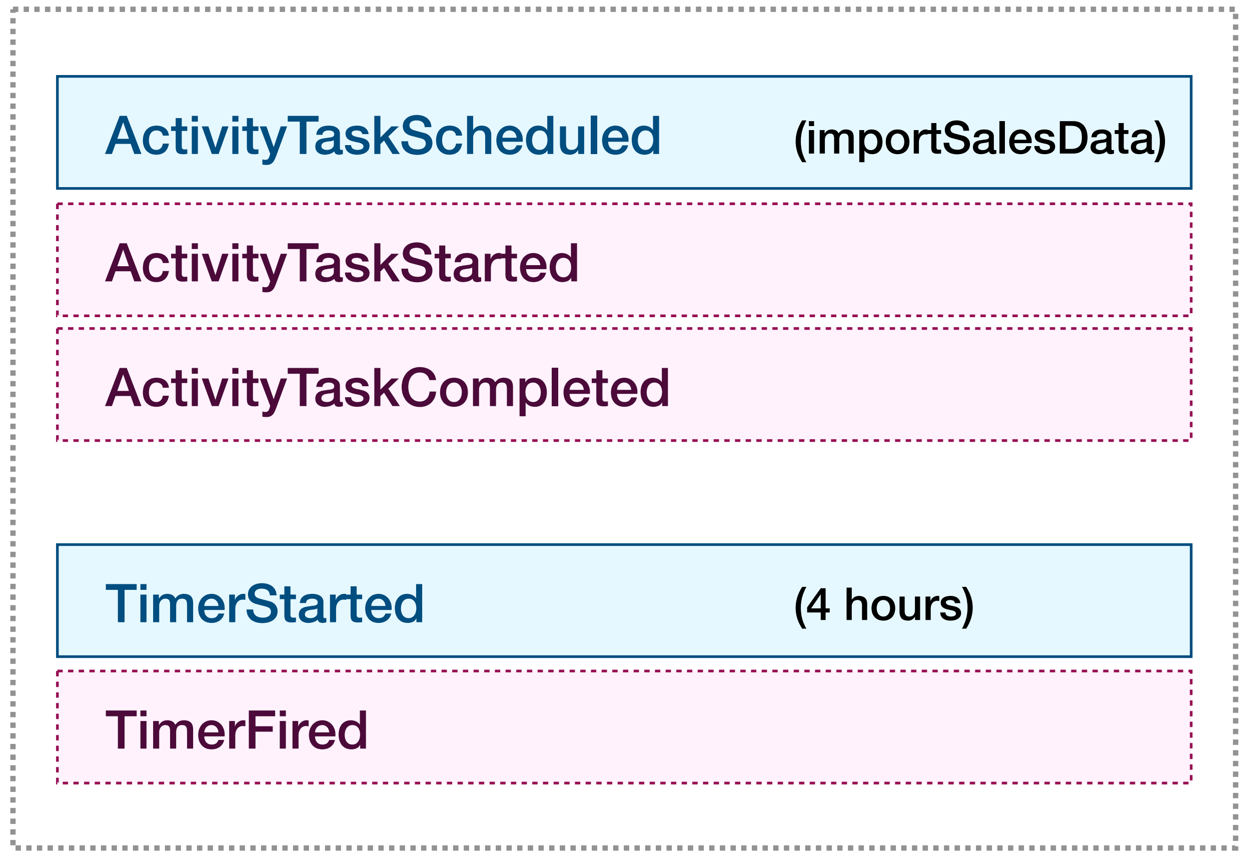
TimerStarted (4 hours)
TimerFired

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

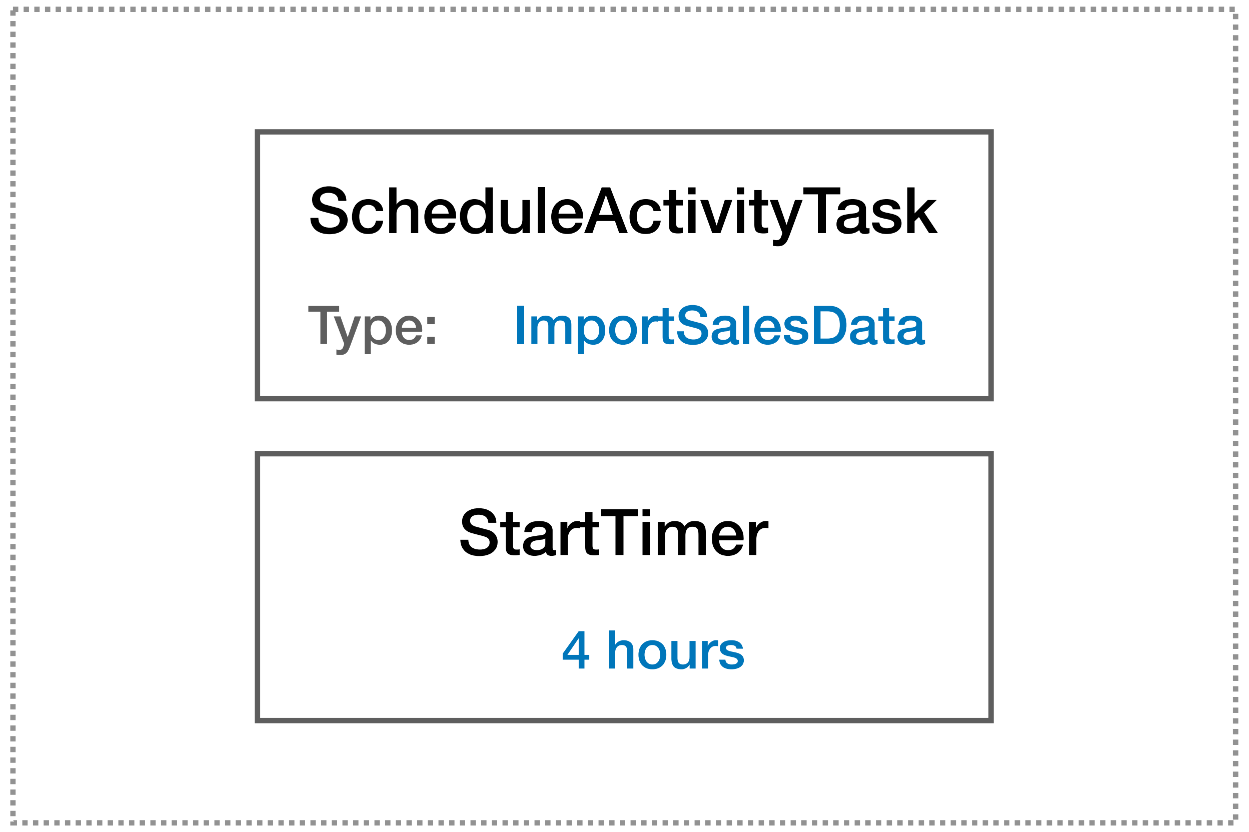
Commands Created

Relevant History Events



ActivityTaskScheduled	(importSalesData)
ActivityTaskStarted	
ActivityTaskCompleted	
TimerStarted	(4 hours)
TimerFired	

Commands Expected (Based on History)



ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

Relevant History Events

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

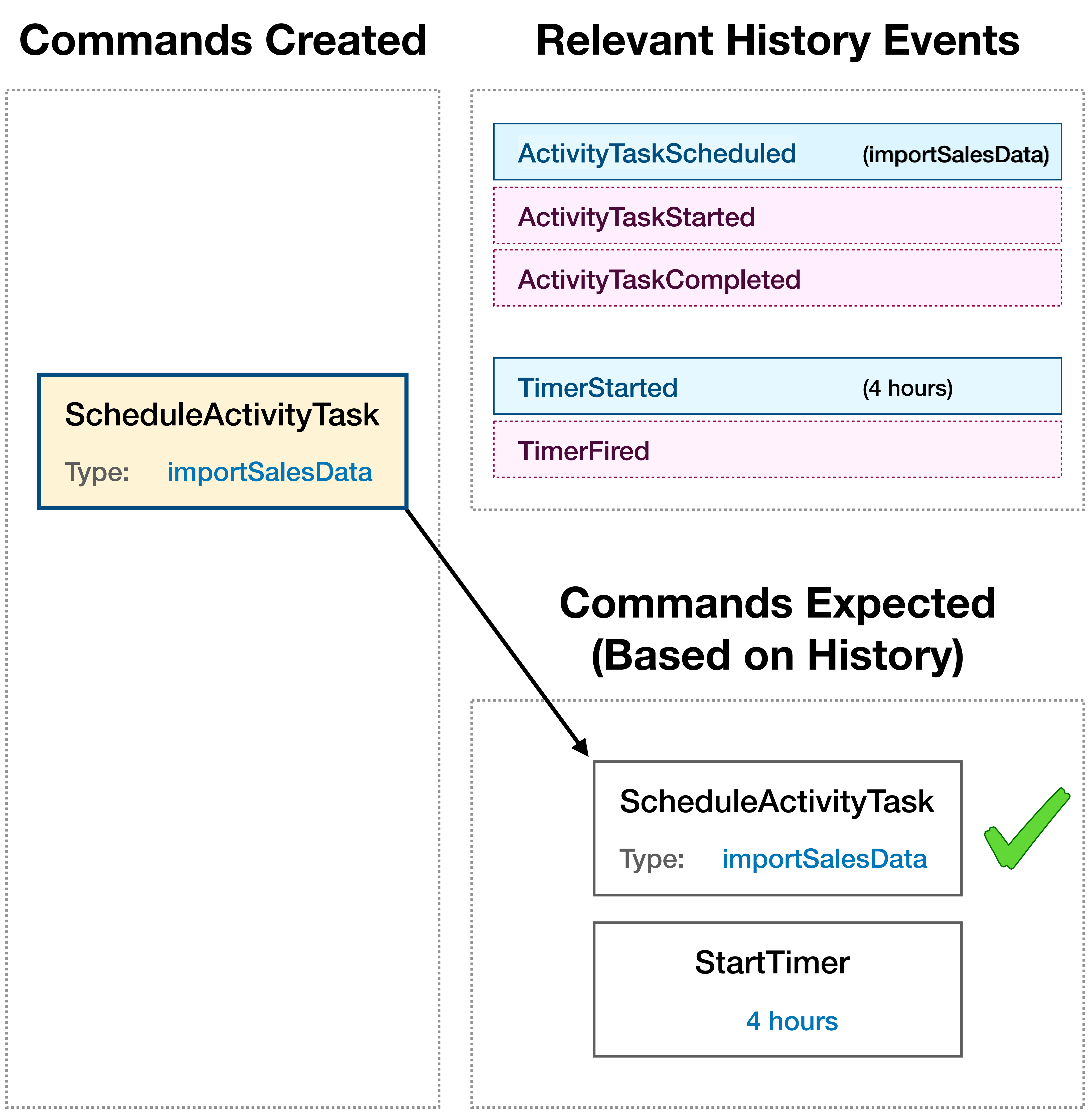
Commands Expected (Based on History)

ScheduleActivityTask
Type: ImportSalesData

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```



A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

Relevant History Events

ActivityTaskScheduled	(importSalesData)
ActivityTaskStarted	
ActivityTaskCompleted	
TimerStarted	(4 hours)
TimerFired	

Commands Expected (Based on History)

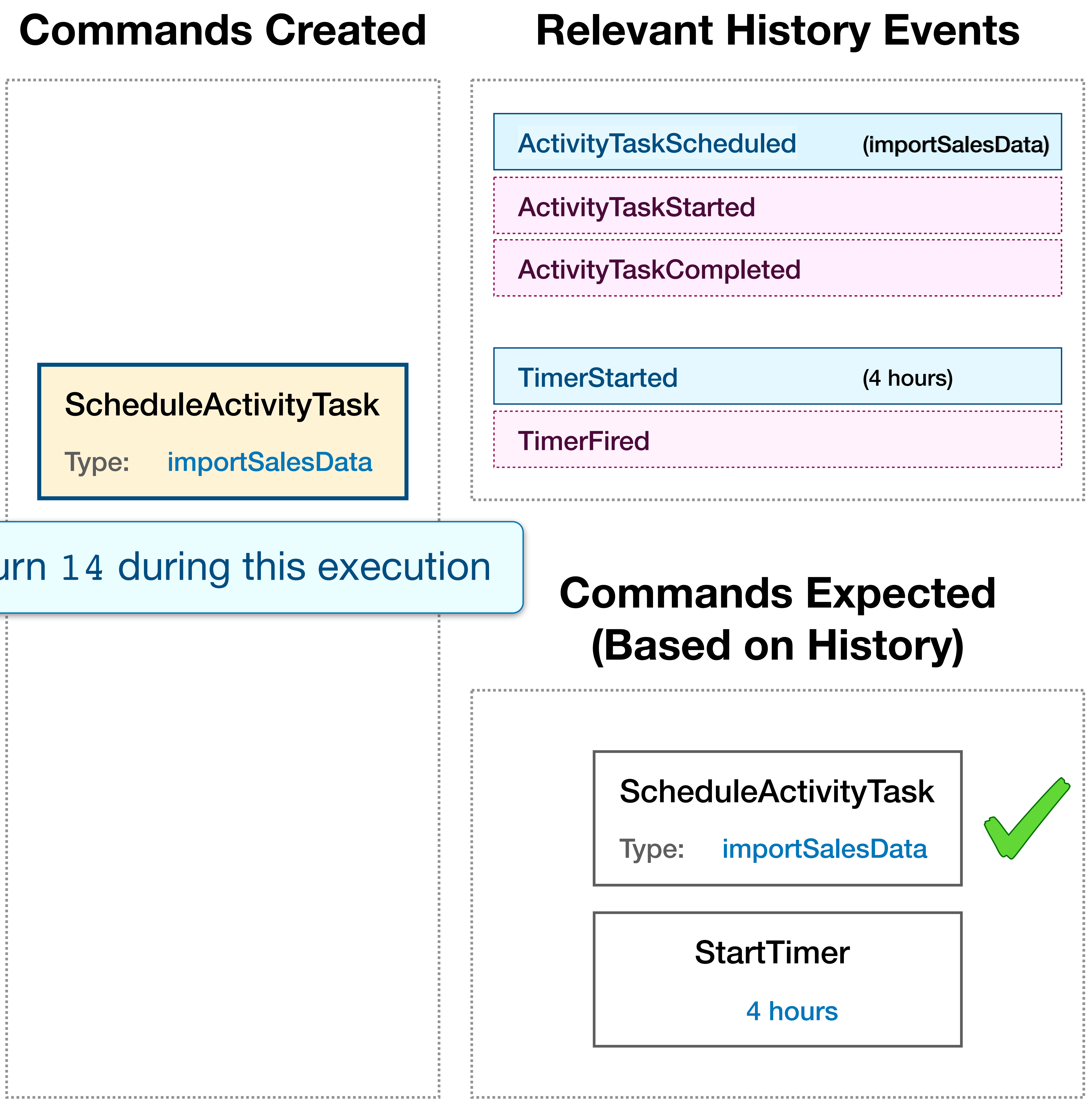
ScheduleActivityTask
Type: importSalesData ✓

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Happens to return 14 during this execution



A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

ScheduleActivityTask
Type: runDailyReport

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

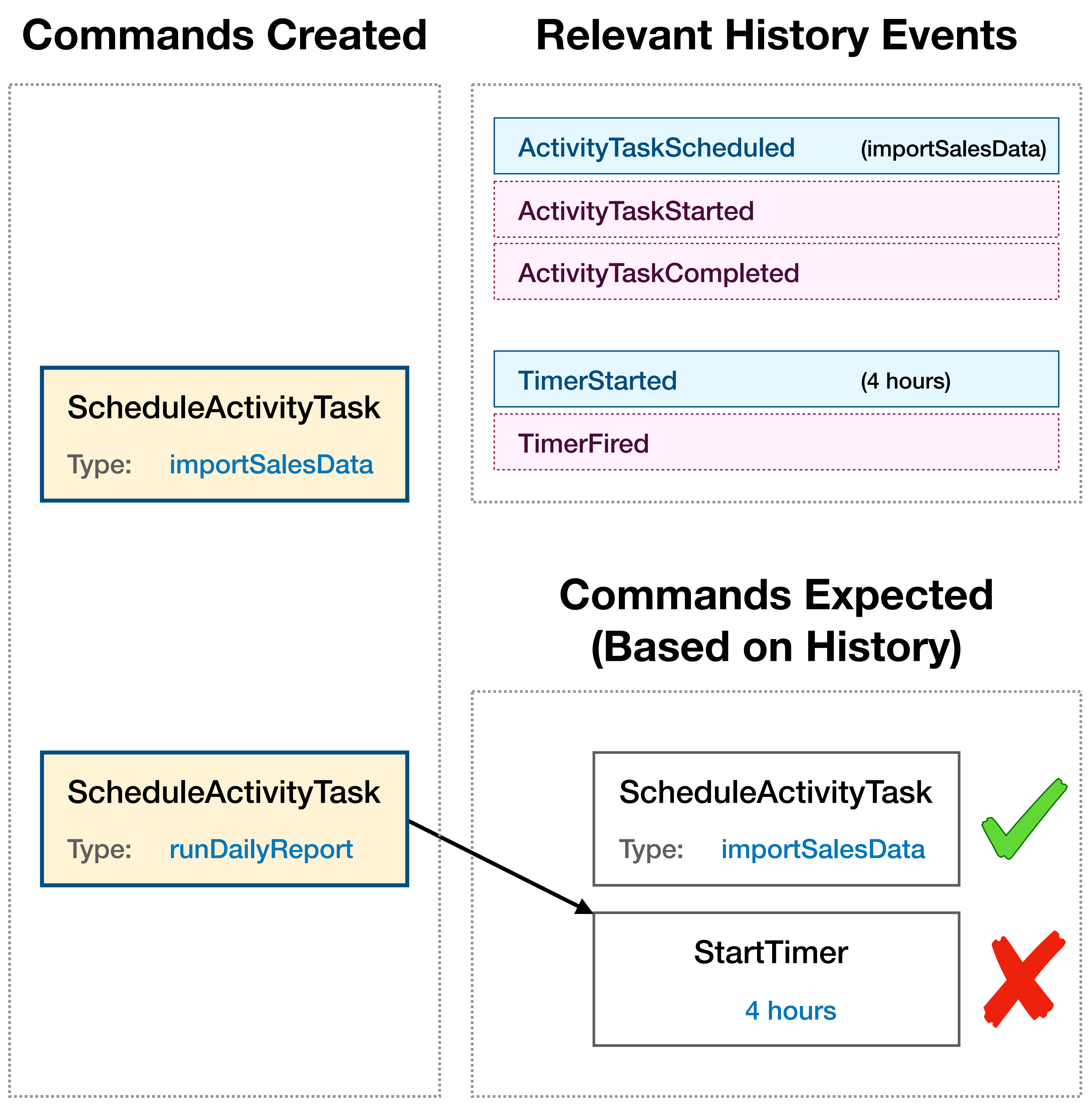
Commands Expected (Based on History)

ScheduleActivityTask ✓
Type: importSalesData

StartTimer
4 hours

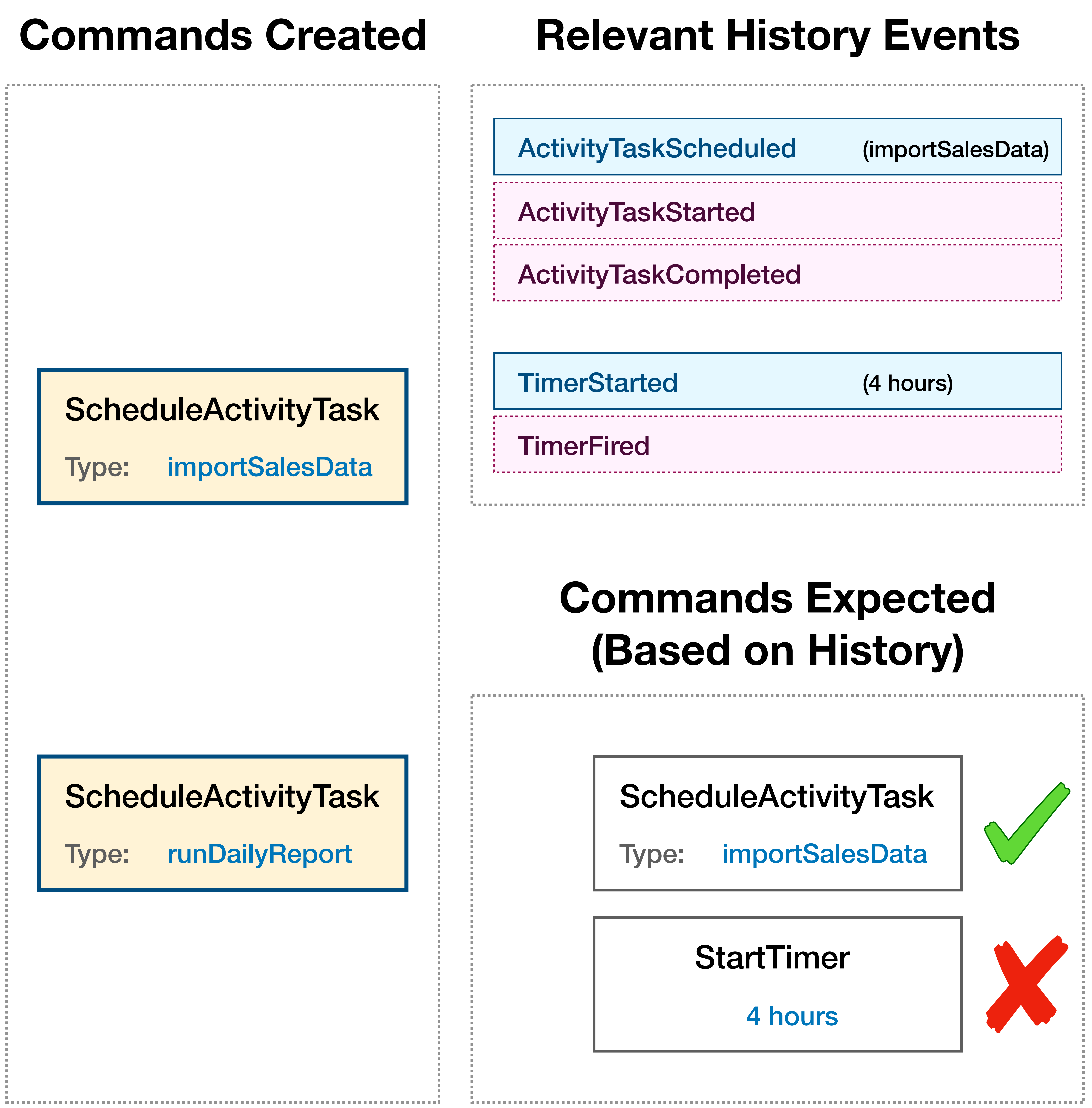
A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```



A Non-Deterministic Workflow Definition

```java
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```



Using random numbers in a Workflow Definition has resulted in Non-Deterministic Error

Each time a particular Workflow Definition is executed with a given input, it must yield exactly the same commands in exactly the same order.

Common Sources of Non-Determinism

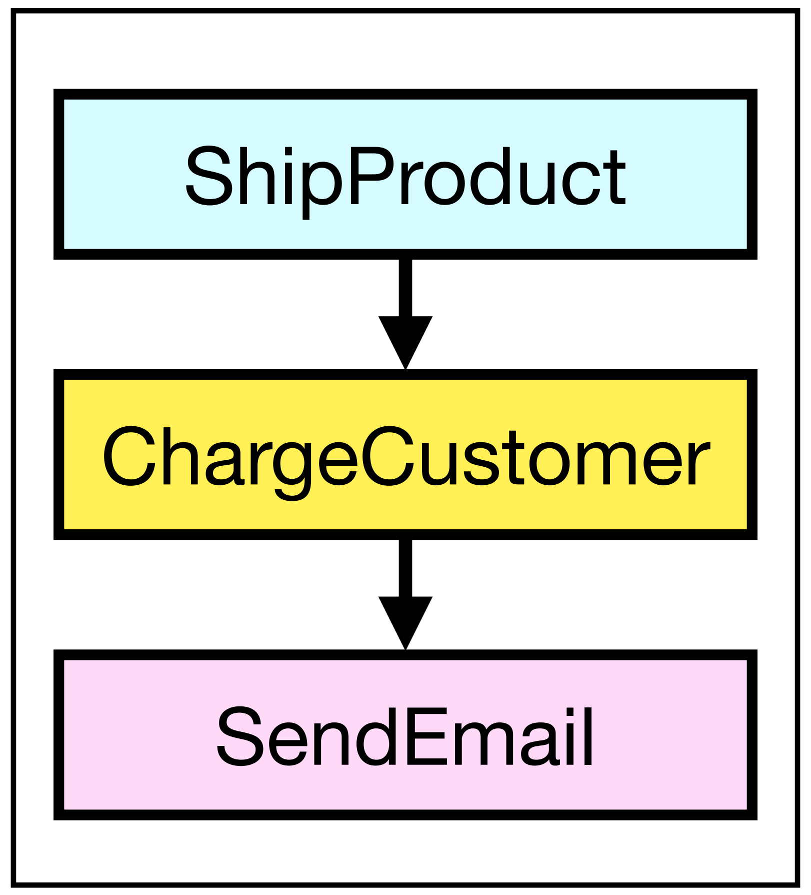
Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**
  - Instead, use Workflow-safe methods such as `Workflow.currentTimeMillis` and `Workflow.sleep`
- **Working directly with threads**
- **Do not iterate over data structures with unknown ordering**

How Workflow Changes Can Lead to Non-Deterministic Errors

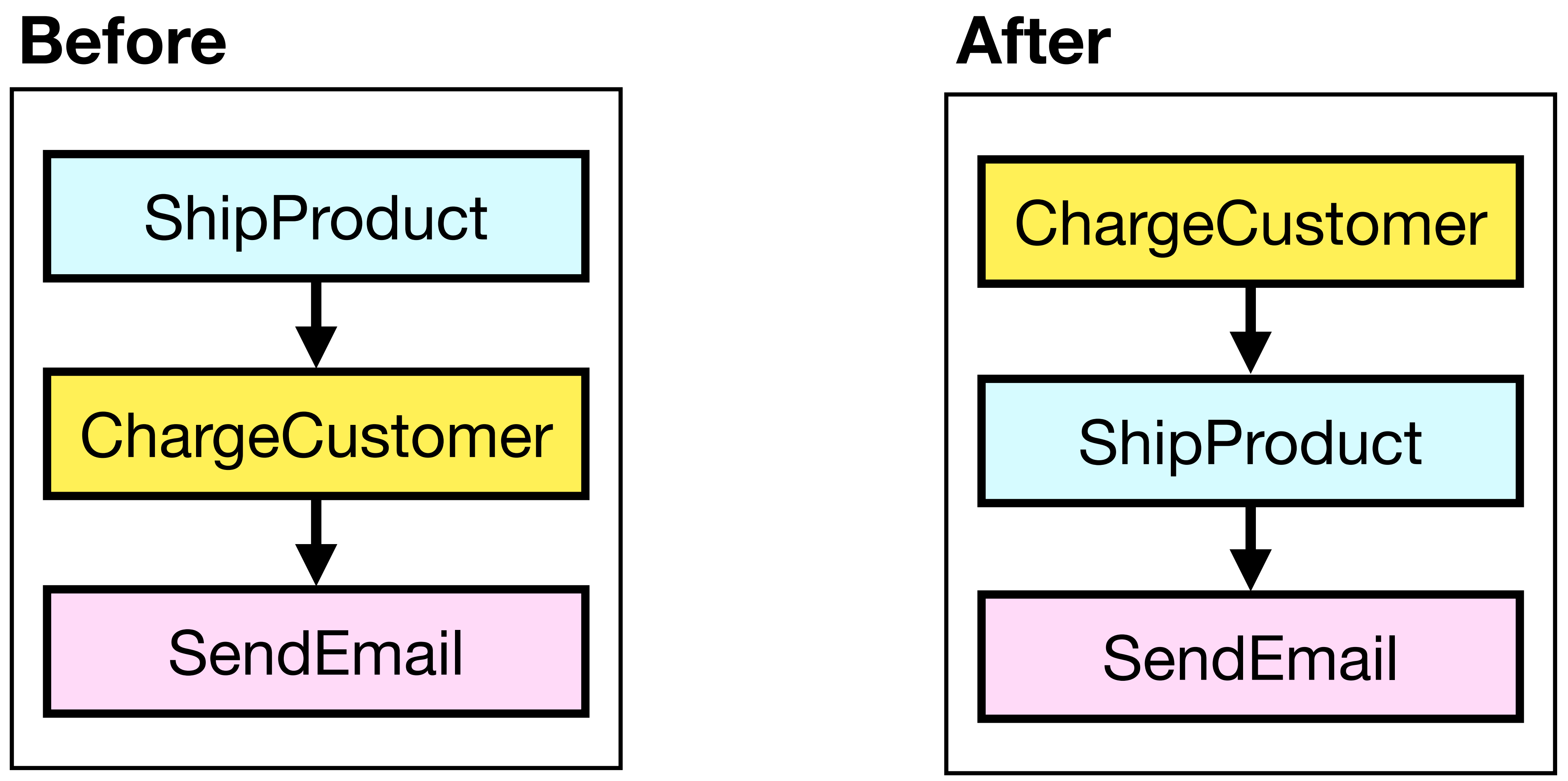
Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario
    - You deploy and execute the following Workflow, which calls three Activities...



Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product



  - You deploy the updated code and restart the Worker(s) so that the change takes effect
- **What happens to the open execution when you restart the Worker?**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
    - Changes to you code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
    - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**
    - Versioning (see documentation for details)
- **How to remediate?**
    - Use Workflow Reset to restart execution to a point before the change was introduced
    - Not always desirable, as any progress made in Activities after the reset point will be lost and re-executed

Resetting A Workflow

- **One way of overcoming a non-deterministic error that has been deployed**
- **Workflows can be reset to a specified point in the history**
- **Can be done via WebUI or CLI**

```
$ temporal workflow reset \
        --workflow-id pizza-workflow-order-XD001 \
        --event-id 4 \
        --reason "Deployed an incompatible change (deleted Activity)"
```

Temporal 102

Validating Correctness of Temporal Application Code

- **The `io.temporal.testing` package provides what you need**
  - Support for JUnit 4 and 5
  - It provides various tools to provide a runtime environment to test your Workflows and Activities
    - `TestWorkflowEnvironment` - Provides a runtime environment, certain aspects of execution work differently to support better testing
      - You can "skip time" so you can test long-running Workflows without Waiting
    - `TestWorkflowExtension` - manages the Temporal test environment and worker lifecycle
    - `TestActivityEnvironment` - Similar to TestWorkflowEnvironment, but for Activities

Testing Activities - Age Estimator

```java
package ageestimationworkflow;

import io.temporal.activity.ActivityInterface;

@ActivityInterface
public interface AgeEstimationActivities {

  int retrieveEstimate(String name);
}
```

```java
package ageestimationworkflow;

// imports omitted for brevity


public class AgeEstimationActivitiesImpl implements AgeEstimationActivities {

  @Override
  public int retrieveEstimate(String name) {

    StringBuilder builder = new StringBuilder();
    ObjectMapper objectMapper = new ObjectMapper();

    String baseUrl = "https://api.agify.io/?name=%s";

    // URL crafting code omitted for brevity

    // HTTP Request code omitted for brevity

    EstimatorResponse response;
   // ObjectMapper code omitted for brevity

    return response.getAge();
  }
}
```

Testing Activities

```java
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import io.temporal.testing.TestActivityEnvironment;

public class AgeEstimationActivitiesTest {

  private TestActivityEnvironment testEnvironment;
  private AgeEstimationActivities activities;

  @BeforeEach
  public void init() {
    testEnvironment = TestActivityEnvironment.newInstance();
    testEnvironment.registerActivitiesImplementations(new AgeEstimationActivitiesImpl());
    activities = testEnvironment.newActivityStub(AgeEstimationActivities.class);
  }

  @AfterEach
  public void destroy() {
    testEnvironment.close();
  }

  @Test
  public void testRetrieveEstimate() {
    int result = activities.retrieveEstimate("Mason");
    assertEquals(38, result);
  }
}
```

Testing Workflows

```java
package ageestimationworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface AgeEstimationWorkflow {

  @WorkflowMethod
  String estimateAge(String name);

}
```

```java
package ageestimationworkflow;

import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;


public class AgeEstimationWorkflowImpl implements AgeEstimationWorkflow {

  ActivityOptions options =ActivityOptions.newBuilder()
        .setStartToCloseTimeout(Duration.ofSeconds(5))
        .build();

  private final AgeEstimationActivities activities =
      Workflow.newActivityStub(AgeEstimationActivities.class, options);

  @Override
  public String estimateAge(String name) {

    int age = activities.retrieveEstimate(name);

    return String.format("%s has an estimated age of %d", name, age);
  }
}
```

Testing Workflows

```java
package ageestimationworkflow;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.RegisterExtension;

import io.temporal.testing.TestWorkflowEnvironment;
import io.temporal.testing.TestWorkflowExtension;
import io.temporal.worker.Worker;

public class AgeEstimationWorkflowTest {

  @RegisterExtension
  public static final TestWorkflowExtension testWorkflowExtension = TestWorkflowExtension
      .newBuilder().setWorkflowTypes(AgeEstimationWorkflowImpl.class).setDoNotStart(true).build();

  @Test
  public void testSuccessfulAgeEstimation(TestWorkflowEnvironment testEnv, Worker worker,
      AgeEstimationWorkflow workflow) {

    worker.registerActivitiesImplementations(new AgeEstimationActivitiesImpl());
    testEnv.start();

    String result = workflow.estimateAge("Betty");

    assertEquals("Betty has an estimated age of 76", result);
  }
}
```

Mocking Activities in Workflow Tests

- **The Workflow test we wrote is an Integration Test!**
  - It invokes an Activity
  - If that Activity required external dependencies (API), that would have needed to be available
  - It's tightly coupled to both
- **Unit test Workflows by mocking Activities**
  - Define new replacement Activities
  - Use the Mockito package to create mocks

Testing Workflows

```java
package ageestimationworkflow;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.RegisterExtension;

import io.temporal.testing.TestWorkflowEnvironment;
import io.temporal.testing.TestWorkflowExtension;
import io.temporal.worker.Worker;

import static org.mockito.Mockito.*;


public class AgeEstimationWorkflowMockTest {

  @RegisterExtension
  public static final TestWorkflowExtension testWorkflowExtension = TestWorkflowExtension.newBuilder()
      .setWorkflowTypes(AgeEstimationWorkflowImpl.class)
      .setDoNotStart(true)
      .build();

  @Test
  public void testSuccessfulAgeEstimation(TestWorkflowEnvironment testEnv, Worker worker, AgeEstimationWorkflow workflow) {

    AgeEstimationActivities mockedActivities = mock(AgeEstimationActivities.class, withSettings().withoutAnnotations());
    when(mockedActivities.retrieveEstimate("Stanislav")).thenReturn(68);

    worker.registerActivitiesImplementations(mockedActivities);
    testEnv.start();

    String result = workflow.estimateAge("Stanislav");

    assertEquals("Stanislav has an estimated age of 68", result);
  }
}
```

Running Tests

```
$ mvn test
```

Exercise #2: Testing the Translation Workflow

- **During this exercise, you will**
  - Write code to execute the Workflow in the test environment
  - Develop a Mock Activity for the translation service call
  - Observe time-skipping in the test environment
  - Write unit tests for the Activity implementation
  - Run the tests from the command line to verify correct behavior
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Review

- Temporal's Java SDK provides support for testing Workflows and Activities with JUnit
- You can test Activities in isolation
- You can test Workflows quickly, even if they have Timers
- You can mock Activities in Workflow tests using Mockito

Temporal 102

Instructor-Led Demo #1

Debugging a Workflow that Does Not Progress

Instructor-Led Demo #2

Interpreting Event History for Workflow Executions

Instructor-Led Demo #3

Terminating a Workflow Execution with the Web UI

Exercise #3: Debugging and Fixing an Activity Failure

- **During this exercise, you will**
  - Start a Worker and run a basic Workflow for processing a pizza order
  - Use the Web UI to find details about the execution
  - Diagnose and fix a latent bug in the Activity Definition
  - Test and deploy the fix
  - Verify that the Workflow now completes successfully
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Temporal 102

Temporal Service is Composed of Four Roles

Frontend

An API Gateway that validates and routes inbound calls

History

Maintains history and moves execution progress forward

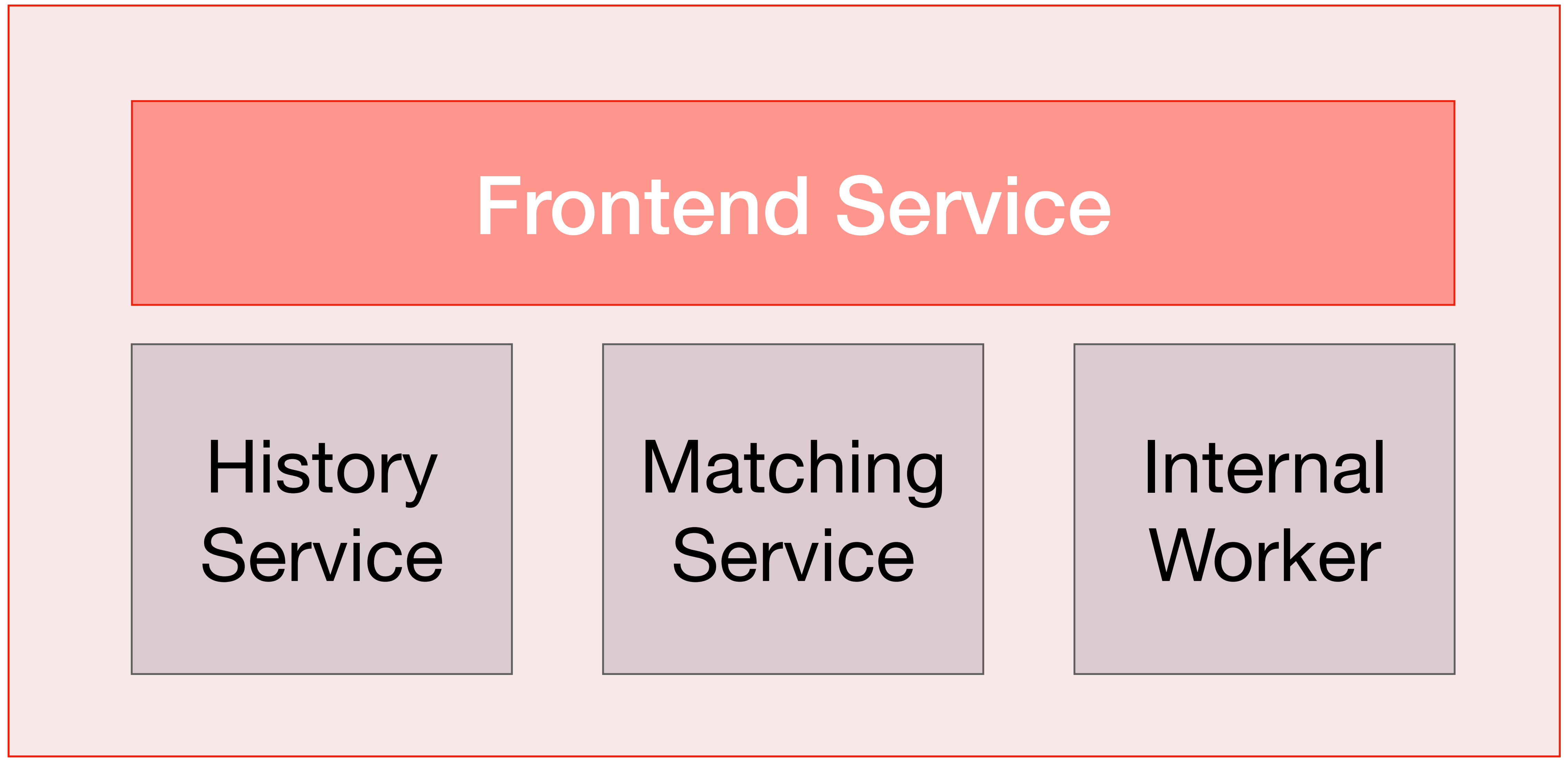
Matching

Hosts Task Queues and matches Workers with Tasks

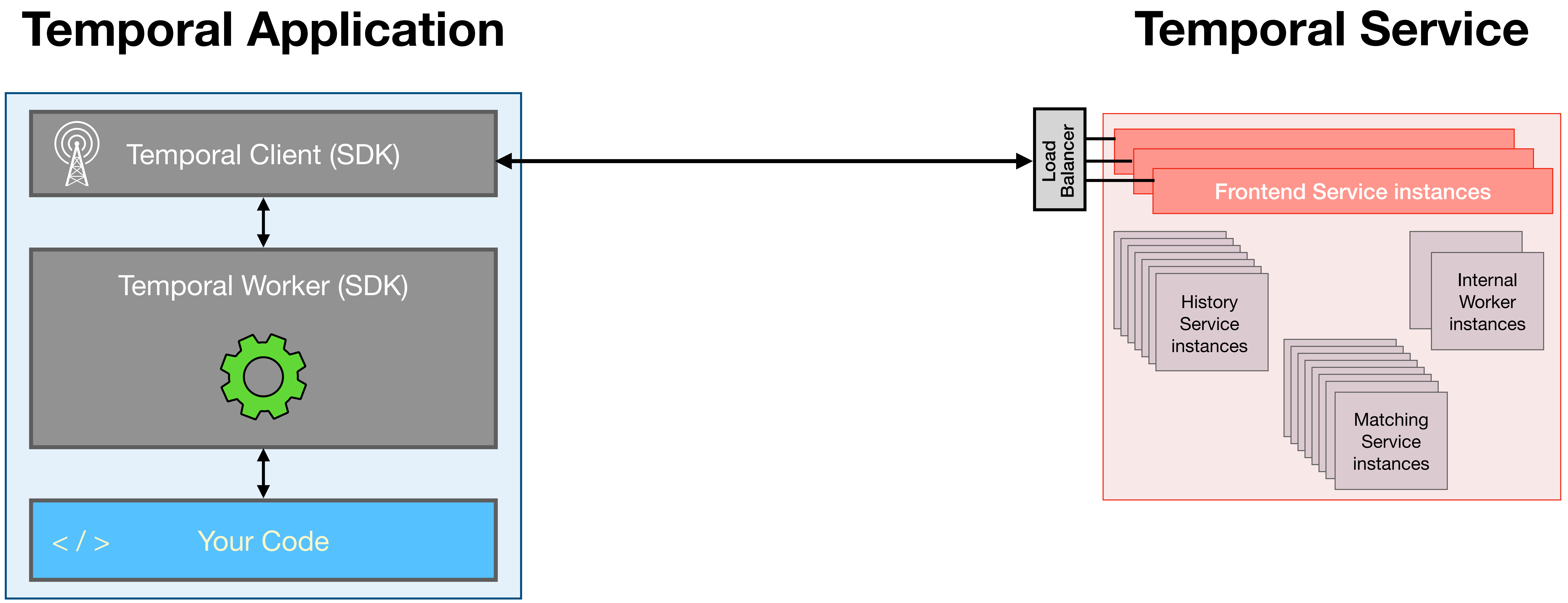
Internal Worker

Runs internal system Workflows, such as those that delete Workflow Execution data after Retention Period elapses

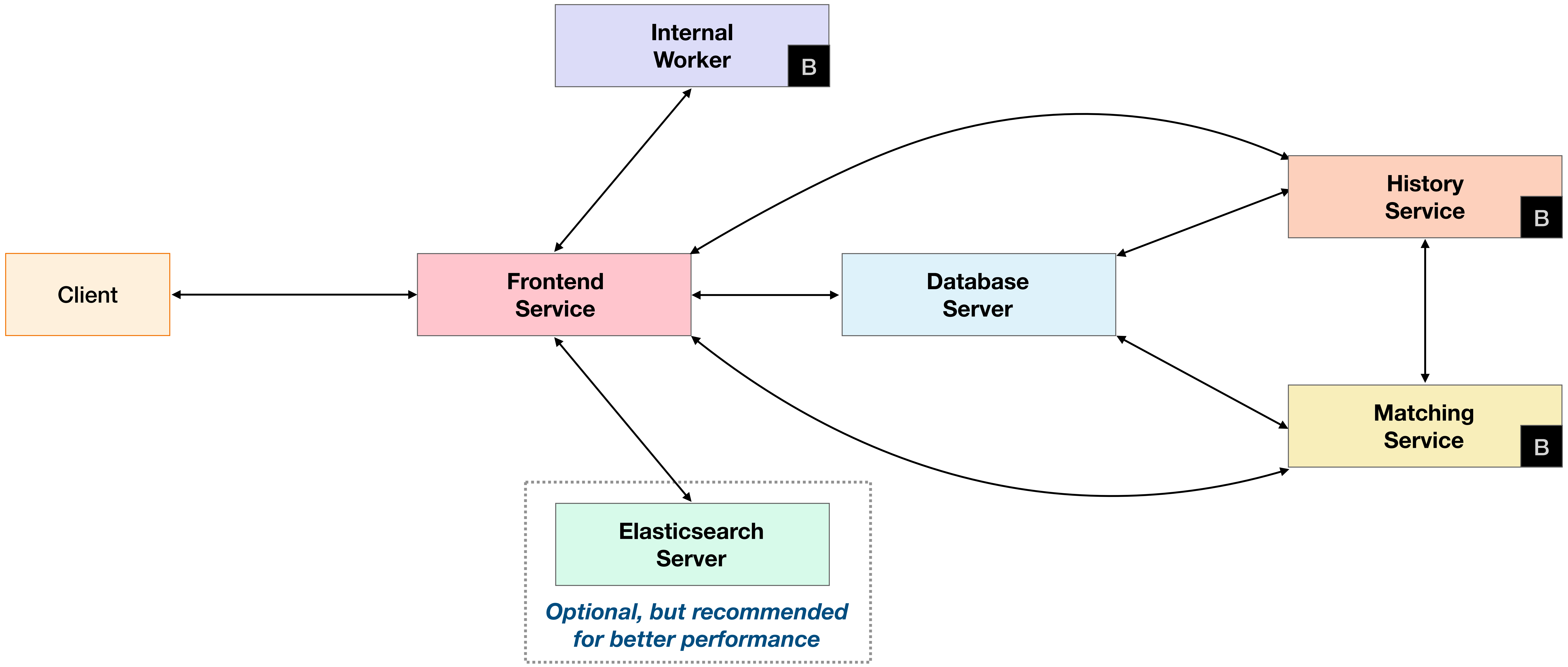
This is distinct from the Worker that executes your application code, which is external to the Temporal Service



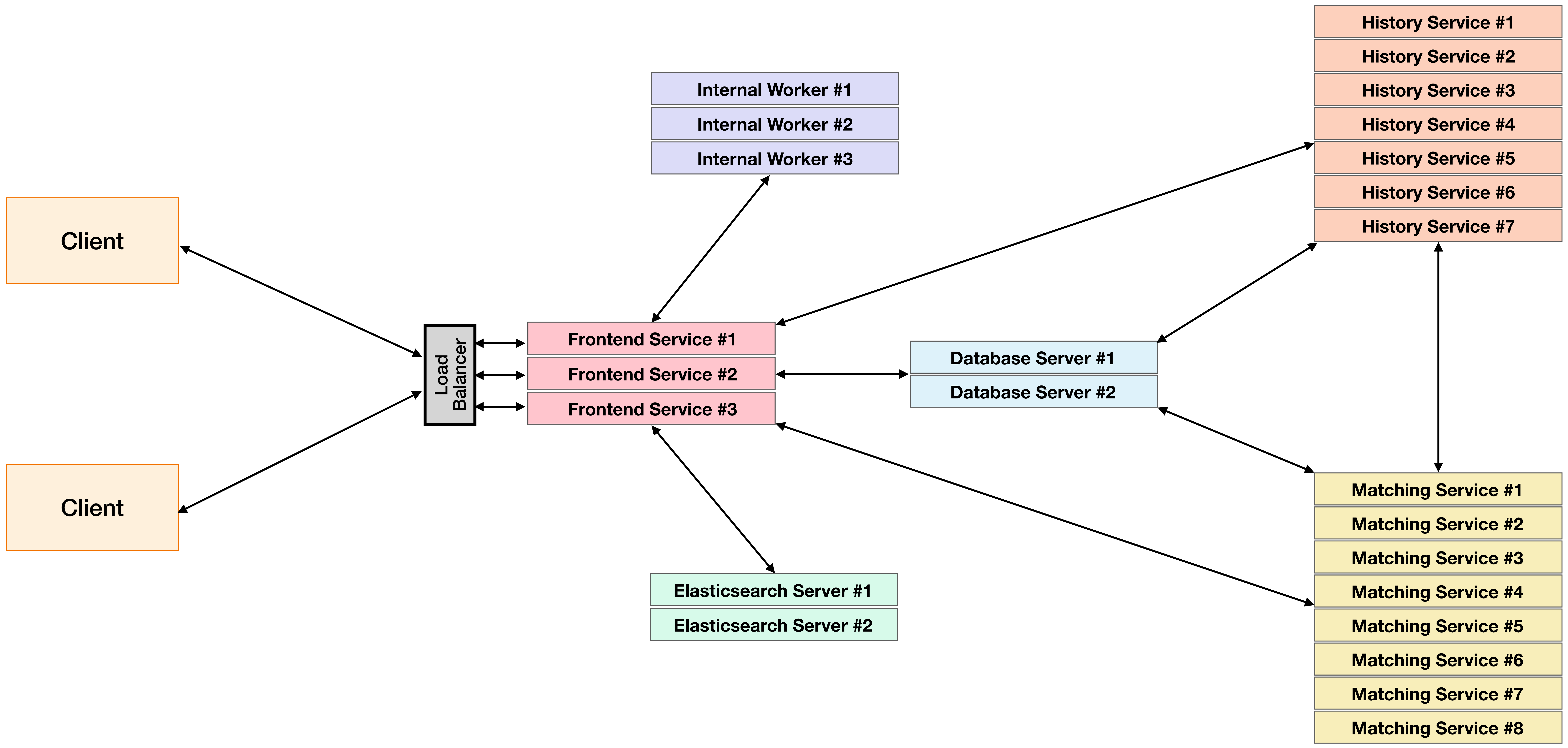
Temporal Service Scalability



Connectivity (Logical)



Connectivity (Physical)



Default Options for a Temporal Client

- **The following code example shows how to create a Temporal Client**
  - This will expect a Frontend Service running on `localhost` at TCP port 7233

```java
import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;

// other code omitted for brevity

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);
```

Customizing a Temporal Client

- **Specify attributes in `WorkflowClientOptions` to configure the Client**
  - **`setTarget()`**: A colon-delimited string containing the hostname and port for the Frontend Service
    - Example: `fe.example.com:7233`
- **Specify attributes in `WorkflowServiceStubs` to configure the gRPC Stubs**
  - **`.setNamespace()`**: A string specifying the namespace to use for requests sent by this Client

Configuring Client for a Non-Local Service

- **This example specifies a namespace, but not parameters needed for TLS**

```java
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.serviceclient.WorkflowServiceStubsOptions;
import io.temporal.client.WorkflowClient;

// other code omitted for brevity
WorkflowServiceStubsOptions stubsOptions = new WorkflowServiceStubsOptions.newBuilder()
                    .setTarget("mycluster.example.com:7233").build();

WorkflowServiceStubs service = WorkflowServiceStubs.newServiceStubs(stubsOptions);

WorkflowClientOptions options = WorkflowClientOptions.newBuilder()
                    .setNamespace("abc");

WorkflowClient client = WorkflowClient.newInstance(service, options);
```

- The options shown above are equivalent to those in the following `temporal` command

```
$ temporal workflow list --address mycluster.example.com:7233 --namespace abc
```

Configuring Client for a Secure Service

- **This example shows Client configuration for a secure non-local cluster**

```java
import io.grpc.netty.shaded.io.netty.handler.ssl.SslContext;
import io.temporal.serviceclient.SimpleSslContextBuilder;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.serviceclient.WorkflowServiceStubsOptions;
import io.temporal.client.WorkflowClient;

//other code omitted for brevity

// Step 1: create the SimpleSslContext
String clientCertFile = "/home/myuser/tls/certificate.pem"
String clientCertPrivateKey = "/home/myuser/tls/private.key"

SslContext sslContext = SimpleSslContextBuilder.forPKCS8(clientCertFile, clientKey).build();

// Step 2: create the WorkflowServiceStubsOptions
WorkflowServiceStubsOptions stubOptions = WorkflowServiceStubsOptions.newBuilder()
   .setSslContext(sslContext)
   .setTarget("mycluster.example.com:7233")
   .build();

// Step 3: create the WorkflowServiceStubs using the SimpleSslContext
WorkflowServiceStubs service = WorkflowServiceStubs.newServiceStubs(stubOptions);

// Step 4: create the WorkflowClientOptions
WorkflowClientOptions options = WorkflowClientOptions.newBuilder()
   .setNamespace("Abc")
   .build();

// Step 5: create the WorkflowClient using the WorkflowServiceStubs and
// WorkflowClientOptions
WorkflowClient client = WorkflowClient.newInstance(service, options);
```

Building a Temporal Application

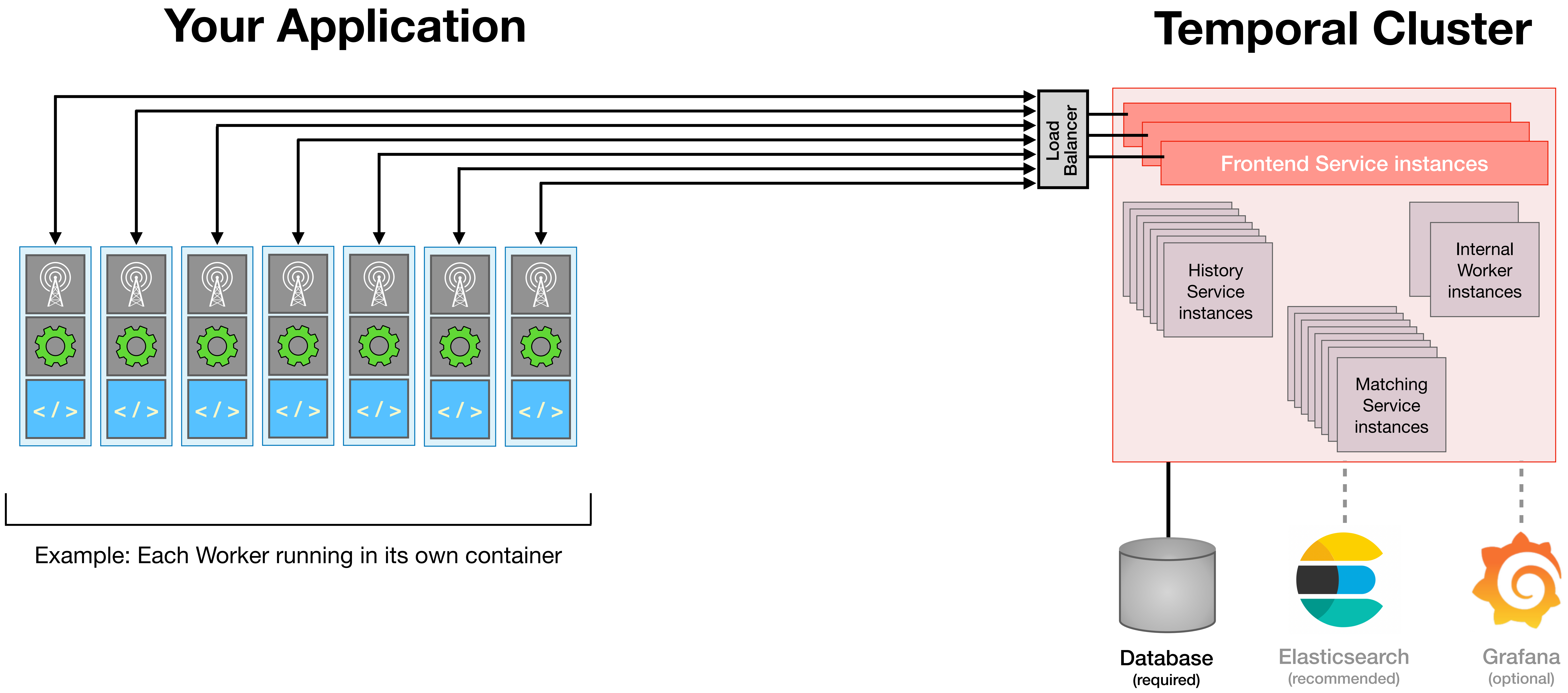
- **Application deployment is usually preceded by a build process**
  - The tools used to do this vary by language, based on the SDK(s) used
  - Temporal does not require the use of any particular tools
  - You can use what is typical for the language or mandated by your organization
- **With the Java SDK, you can build the Worker to create a JAR**
  - The result is what you would deploy and run in production
  - It must contain all dependencies required at runtime

```
$ mvn clean package
```

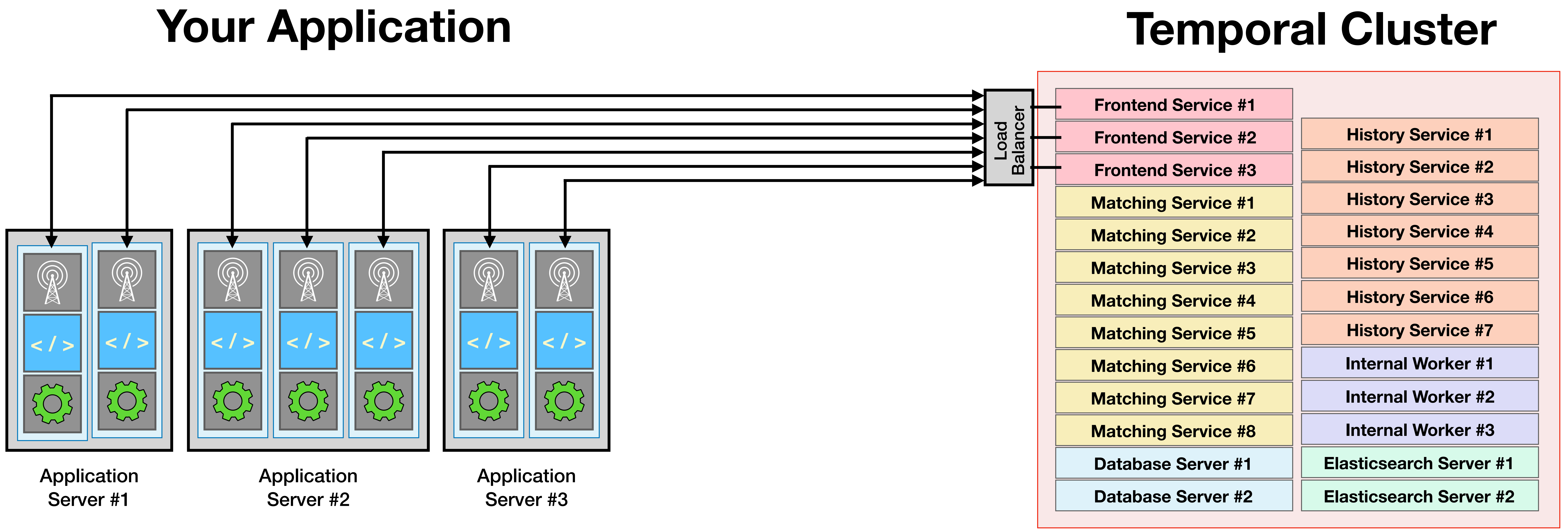
Temporal Application Deployment

- **Once built, you'll deploy the application to production**
  - This will contain your compiled code, plus compile-time dependencies (e.g., Worker, Client, etc.)
  - Ensure any needed dependencies are available at runtime
    - For example, database drivers used by your application
    - For example, the Java runtime or Python interpreter for polyglot Temporal applications
- **Temporal is not opinionated about how or where you deploy the code**
  - Key point: Workers run externally to Temporal Service
  - It's up to you how you run the Workers: bare metal, virtual machines, containers, etc.
  - Let's quickly look at two possible examples

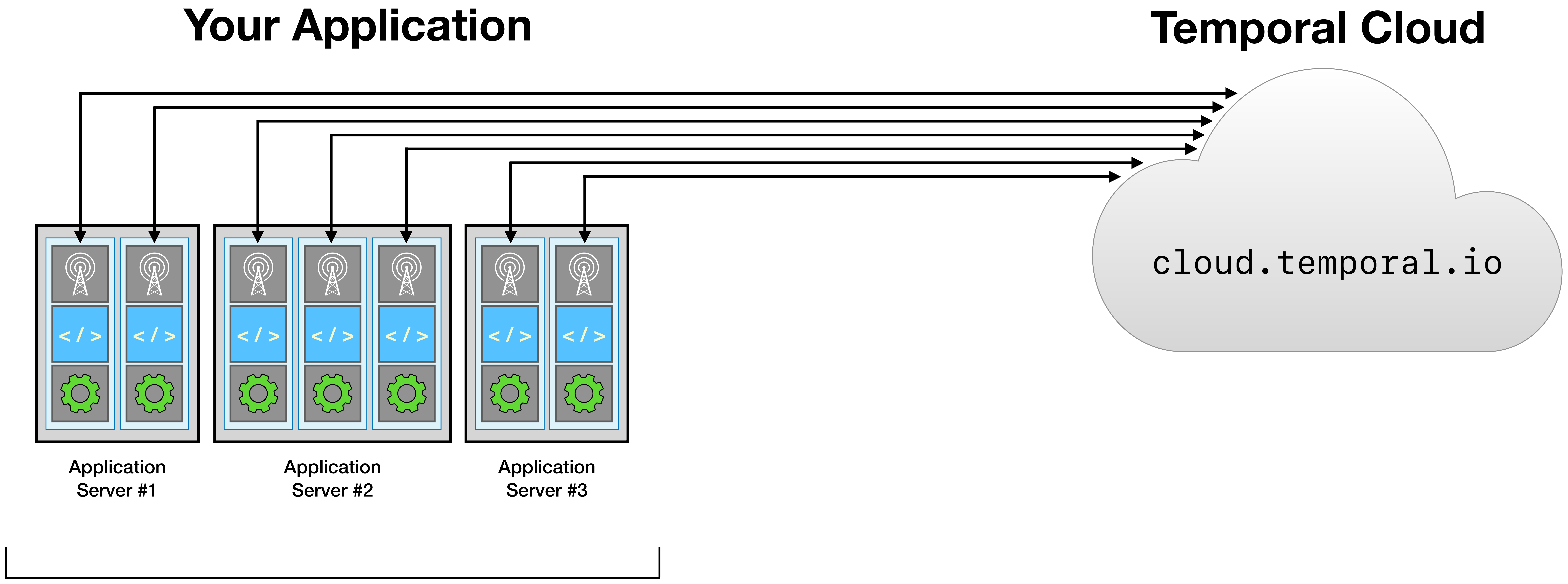
Deployment Scenario #1


Your Application
Temporal Cluster
Load Balancer
Frontend Service instances
History Service instances
Internal Worker instances
Matching Service instances
< / >
Example: Each Worker running in its own container
Database
(required)
Elasticsearch
(recommended)
Grafana
(optional)

Physical View of an Application in Production



Deployment Scenario #2



Example: Multiple Worker Processes distributed across bare metal

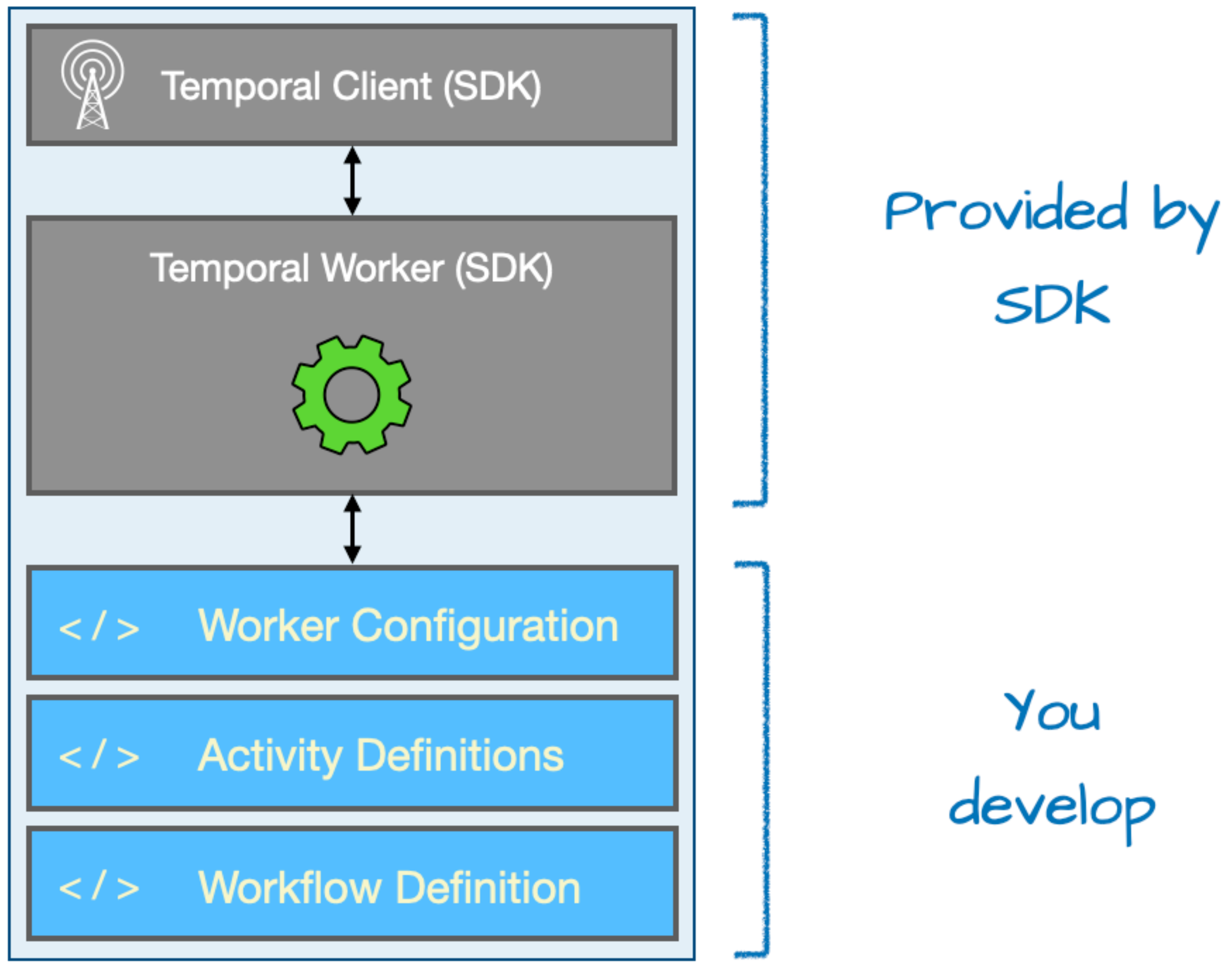
Review

- Temporal Services have four parts:
  - Frontend Service, History Service, Matching Service, and Worker Service
- To connect to a Temporal Service, you can specify the address, the namespace, and provide certificates and keys for mTLS connections
- Use your existing build processes to prepare your app
  - You can bundle Workflows to improve production performance
- Temporal is not opinionated about how or where you deploy the code
  - You run your Workers, Activities, and Workflows on your own servers
  - You can run the Temporal Service on your own servers or you can use Temporal Cloud.

Temporal 102

Essential Points (1)

- **Temporal applications contain code that you develop**
  - Workflow and Activity Definitions, Worker Configuration, etc.
- **Temporal applications also contain SDK-provided code**
  - Such as the implementations of the Worker and Temporal Client
- **Temporal guarantees durable execution of Workflows**
  - If the Worker crashes, another Worker uses History Replay to automatically recreate pre-crash state, then continues execution
  - From the developer perspective, it's as if the crash never even happened



Essential Points (2)

- **Temporal Service perform orchestration via Task Queues**
  - A Worker polls a Task Queue, accepts a Task, executes the code, and reports back with status/results
  - Communication takes place by Workers initiating requests via gRPC to the Frontend Service
  - **Key point**: Execution of the code is external to Temporal Service
- **As Workers run your code, they send Commands to Temporal Service**
  - For example, when encountering calls to Activity Methods or `Workflow.sleep` or when returning a result from the Workflow Definition
- **Commands sent by the Worker lead to Events logged by Temporal Service**

Essential Points (3)

- **The Event History documents the details of a Workflow Execution**
  - It's an ordered append-only list of Events
  - Temporal enforces limits on the size and item count of the Event History
- **Every Event has three attributes in common: ID, timestamp, and type**
  - They will also have additional attributes, which vary by Event Type
  - Examining the Event History and attributes of individual Events can help you debug Workflow Executions

Essential Points (4)

- **A single Workflow Definition can be executed any number of times**
    - Each time potentially having different input data and a different Workflow ID
        - At most, one open Workflow Execution with a given Workflow ID is allowed per Namespace
        - This rule applies to *all* Workflow Executions, not just ones of the same Workflow Type
- **Once started, Workflow Execution enters the Open state**
    - Execution typically alternates between making progress and awaiting a condition
    - When execution concludes, it transitions to the Closed state
    - There are several subtypes of Closed, including Completed, Failed, and Terminated

Essential Points (5)

- **Temporal requires that your Workflow code is deterministic**
    - This constraint is what makes durable execution possible
    - Temporal's definition of determinism: Every execution of a given Workflow Definition must produce an identical sequence of Commands, given the same input
    - Non-deterministic errors can occur because of something inherently non-deterministic in the code
        - Can also occur after deploying a code change that changes the Command sequence, if there were open executions of the same Workflow Type at the time of deployment
- **Activities are used for code that interacts with the outside world**
    - Activity code isn't required to be deterministic (but it should be idempotent)
    - Activities are automatically retried upon failure, according to a configurable Retry Policy

Essential Points (6)

- **Recommended best practices for Temporal app development**
    - Use classes (not individual parameters) as input/output of your Workflow and Activity definitions
    - Be aware of the platform's limits on Event History size and item count
    - Replace non-deterministic code in Workflow Definitions with Workflow-safe counterparts
    - Use Temporal's replay-aware logging API, ideally integrating with a third-party logging package

Essential Points (7)

- **We don't dictate how to build, deploy, or run Temporal applications**
    - Typical advice: Build, deploy, and run as you would any other application in that language
    - However, we recommend running >= 2 Workers per Task Queue (availability/scalability)

Thank you for your time and attention

We welcome your feedback

t.mp/replay25ws