**Inventory System Analysis**

Table of Contents

[Problem Statement 4](#_Toc87177979)

[Objectives and Goals 5](#_Toc87177980)

[Conceptual Model 6](#_Toc87177981)

[System Description 6](#_Toc87177982)

[State Variables 7](#_Toc87177983)

[Set of Events 7](#_Toc87177984)

[Input Variables 8](#_Toc87177985)

[Output Variables 8](#_Toc87177986)

[Specification Model 9](#_Toc87177987)

[State Equations 9](#_Toc87177988)

[Output Equations 10](#_Toc87177989)

[Input Data Modelling 12](#_Toc87177990)

[Computational Model 13](#_Toc87177991)

[Scheduling 13](#_Toc87177992)

[Events 14](#_Toc87177993)

[Inventory Initialization 14](#_Toc87177994)

[Supply Handler 16](#_Toc87177995)

[Demand Handler 17](#_Toc87177996)

[Evaluation Handler 18](#_Toc87177997)

[Termination Handler 19](#_Toc87177998)

[Updating Statistical Variables 20](#_Toc87177999)

An **inventory** stores items that a seller expects to sell. The exact number of items we store will depend on the **demand** for the product. If we do not have enough, then we will **lose sales**. If we store too many, **storage costs** will rise. For lost sales, there is the additional negative impact that customers may choose a different vendor for their next purchase, so there is an effect on **future sales** as well.

At the beginning of each month, we examine our inventory to check how many products we have left, . Consider that there is an arbitrary **threshold**, , and if , we do not request more products from our supplier.

If we find that we do have to request more products, then we order them in a way such that the **maximum capacity** of our inventory, , is exactly matched, i.e. we order products. By doing this, we ensure that, even if there are no sales until the new supply arrives, we will not cross our storage capacity. Note that the new supply does not arrive immediately. In our simulation, we will assume that it takes a **random amount of time** for the new supply to arrive.

In a given day, the number of **customers** that we can get depends on some distribution. For a given customer, if they arrive and demand a certain number of products and we do not have enough, the number of products in the inventory, , could become **negative**. A negative value indicates lost profits.

We are simulating this system mainly to determine two things, firstly, what value of we should maintain, and secondly, what value of we should maintain. We will vary these two values to try to reach the **optimum balance** between inventory storage costs and potential lost profits.

## Problem Statement

We are assuming that there is a **single product** (usually there are multiple though). When demands come from customers, they are **satisfied immediately** if the required number of products are available. The number of **products left** in the inventory is **evaluated periodically** in order to determine if we should request a refill from the supplier. The **arrival** of the new supply has a **variable time**. The system is being studied for a **predefined time** in order to **minimize** the average **inventory costs**.

## Objectives and Goals

We will be studying different **inventory policies**. Our goal is to know the **optimum inventory level**. This is determined by two variables:

1. The maximum inventory capacity,
2. The order placement threshold,

We also want to know the **optimum average inventory cost**. This depends on three variables:

1. The average ordering cost, the cost of getting a new supply from the supplier. This includes any fixed costs, such as delivery vehicles, or variable costs, such as packaging, which depends on the number of products.
2. The average holding cost, the cost associated with storing items.
3. The average shortage cost, the cost associated with lost sales. This refers to the long-term effects of the lost sales. The actual demands will still be met as soon as new supplies arrive.

## Conceptual Model

### System Description

1. As discussed above, the system manages a **single item**.
2. Customers **arrive** at **random intervals** and the **demanded amount** is **variable**. If the demanded amount is available, the demand is **satisfied immediately**. If the demanded amount is unavailable, a **backlog** is kept of the number of items that could not be sold. This allows us to keep track of lost sales. When a new supply arrives, the backlogged demands will be fulfilled before any other sales.
3. The inventory is **evaluated periodically** (every month in our case) at **fixed intervals**. The evaluation determines the need for an **order placement**, which occurs if the available items is less than the **threshold**.
4. If an **order is placed**, the number of items should not exceed the **inventory capacity** and should **minimize unused capacity**, i.e. number of items will be ordered. Order placement can only be triggered by an **evaluation**. If an order is placed, the **ordering cost** needs to be determined.
5. It takes a **random amount of time** for an order to arrive after it has been placed. Once the order has arrived, **backlogged demands** are fulfilled first and the remaining items are added to the **inventory**.
6. The **period of study** is a predetermined number of months.

### State Variables

The only state variable is the **number of items available** at time , the **inventory level**, .

This value may be initialized to a **non-zero value**, in the best case the inventory capacity. It decreases by the demand amount when a demand arrives. It increases by the supply amount when a supply arrives. Its value may be **negative**, in the case of backlogged demands.

The inventory level, , is evaluated at the beginning of each month. If , then an order is placed.

### Set of Events

* **Evaluation** (e) – The evaluation of the inventory level does not directly affect the system state. It determines whether an order needs to be placed and what the order amount will be. Thus, it triggers the **order placement**. The evaluation event occurs at a fixed interval, at the beginning of each month in our case.
* **Supply** (s) – Supply events are when a new supply arrives from the suppliers. These events **increase the inventory level** by the supplied amount. The supply event takes a random amount of time to occur after an order is placed.
* **Demand** (d) – Demand events are when a customer makes a demand. These events **decrease the inventory level** by the demand amount, perhaps even making it **negative**. The demand event interval time is random.
* **Termination** (t) – The termination event is the **end of the simulation**. As such, it only occurs once. It forces the system to stop immediately, **cancelling any pending events**. The termination must be done carefully, so as to ensure it occurs **before any evaluation event** at the exact same time. This is because there is no point to evaluating the system and placing a new order it the system will end immediately.

### Input Variables

* Evaluation Interval
* Demand Arrival Interval
* Supply Arrival Interval
* Simulation Termination Time
* Maximum Inventory Level
* Order Placement Threshold

### Output Variables

* Average Inventory Cost
* Average Ordering Cost, which is a job-average output variable
* Average Holding Cost, which is a time-average output variable
* Average Shortage Cost, which is a time-average output variable

## Specification Model

The specification model mainly deals with **state equations**, **output equations**, **feasible events** and **input data modelling**. For simplicity, we will be considering that the specifications for the input data will be given, meaning input data modelling is not required.

### State Equations

Here, is the amount of supply and is the amount of demand.

The **state space** is

This means that the state variable can have any integer value up to a maximum of .

The **feasible event set** is

This means any event can occur for any value of the state variable.

### Output Equations

The **average ordering cost** is the ordering cost per month. It is the ratio of the total ordering cost to the number of months.

Let denote the **ordering cost** and denote the **total ordering cost**.

Here, is if and otherwise. is the **fixed cost**, is the **variable cost per item** and is the total number of **items ordered**.

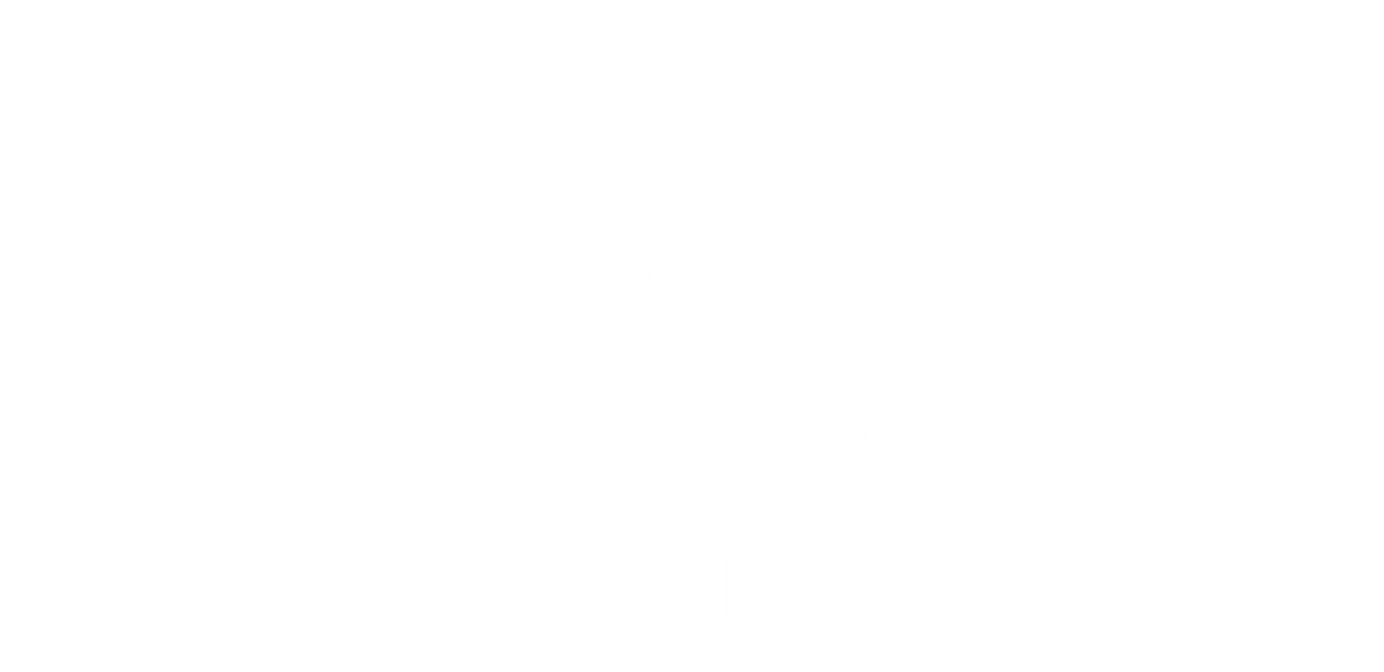
Every time an evaluation is done, is updated.

At the end of the simulation, the average ordering cost is given by:

where is the number of evaluation months.

The **average holding or shortage cost** depends on the value of . A holding cost is incurred if and a shortage cost is incurred if . These quantities clearly depend on the **number of items** in the inventory, since whenever the number of items in the inventory changes, these values can change. Since each event advances the simulated time, these values are **time-average** variables.

To identify the holding cost and the shortage cost, we use the variables and respectively, since the two values depend on being positive and negative respectively. The actual values however, will be positive, i.e. for .



The graph above shows one possible example of how and change over time.

We can calculate as

We can calculate as

In both cases, is the total number of months.

From here, we can calculate the average holding and shortages costs as and respectively, where is the holding cost per item per month and is the backlog cost per item per month.

### Input Data Modelling

For this example, the distributions being used to model the input data is being provided. In an actual simulation, we would have to identify the distribution using empirical data, i.e. from the actual system.

1. The first input data we need is the **evaluation interval**. We already know that this is **deterministic**, and is set to once at the beginning of each month.
2. The second input data is the **demand interval**. Assume that this is a random quantity and is an **IID exponential random variable**. The mean is , meaning there are 10 orders per month on average.
3. The third input data is the **demand amount**. Assume that this is an IID random variable with the following PMF:
4. The **delivery lag** is an IID uniform random variable with parameters and . This means deliveries from the supplies will take between 15 to 30 days to arrive.
5. The **setup cost**, , is fixed at .
6. The **cost per item**, , is fixed at .
7. The **holding cost per item per month**, , is fixed at .
8. The **shortage cost per item per month**, , is fixed at .

## Computational Model

### Scheduling



The first thing to notice is that the overall computational model remains exactly the same. We still need a Simulator class that has a clock variable, an eventList variable, an initialize function and a run function. The run function is still removing events from eventList, updating clock and executing the events until the list is empty. It is just the work in the actual events that will change, since the events, input equations and output equations are different. This is the advantage of discrete event systems.

And yes, there are libraries to handle this part.

### Events

There are four events we need to deal with, the **supply** event, the **demand** event, the **evaluation** event and the **termination** event.

Each of these events will have a corresponding event handler. All of these handlers will be part of the InventorySystem class.

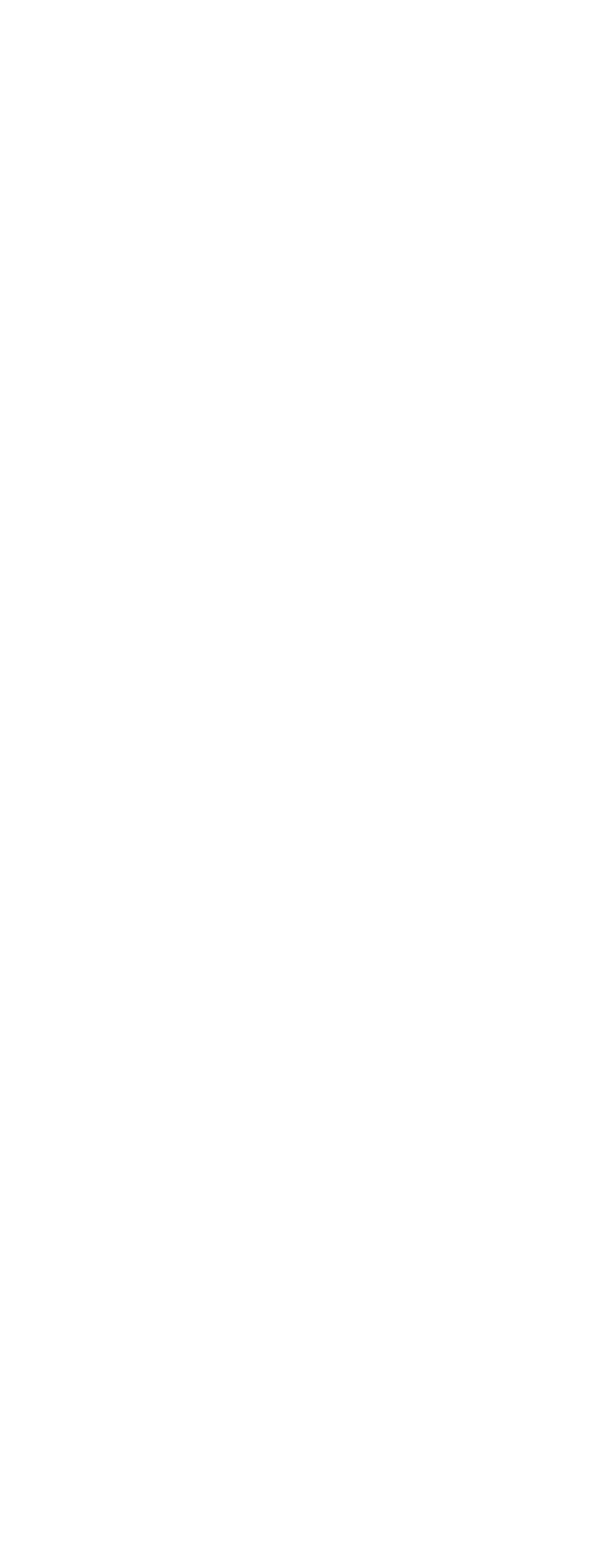
### Inventory Initialization

Before we can start the run function from the Simulator, we need to include at least one event in eventList. This is where the process of initializing the inventory comes in.



1. The first thing to do is to initialize the **inventory level**. We do this by setting . We also set the **termination period**, denoted as in the diagram above.
2. Next, we initialize variables to keep track of the **evaluation event number**, **demand event number** and **supply event number**. These are all initialized to .
3. The third step is to initialize the statistical variables, which we will need to produce the output data. This includes the total **ordering cost**, the area under the **holding** graph and the area under the **shortage** graph.
4. Next, we initialize a variable to keep track of the **time of the last event**. This variable, as seen before, will help us with other calculations. It is initialized to .
5. Finally, we need to schedule a few events.
6. The **evaluation** event is scheduled, since it needs to run at the beginning of each month.
7. The **termination** event is scheduled, with the parameter .
8. The **demand** event is scheduled, which requires the duration after which the event will occur as a parameter. This duration is found using a separate process to calculate a random value based on an **exponential distribution** with the provided mean. This is also a library function.

### Supply Handler

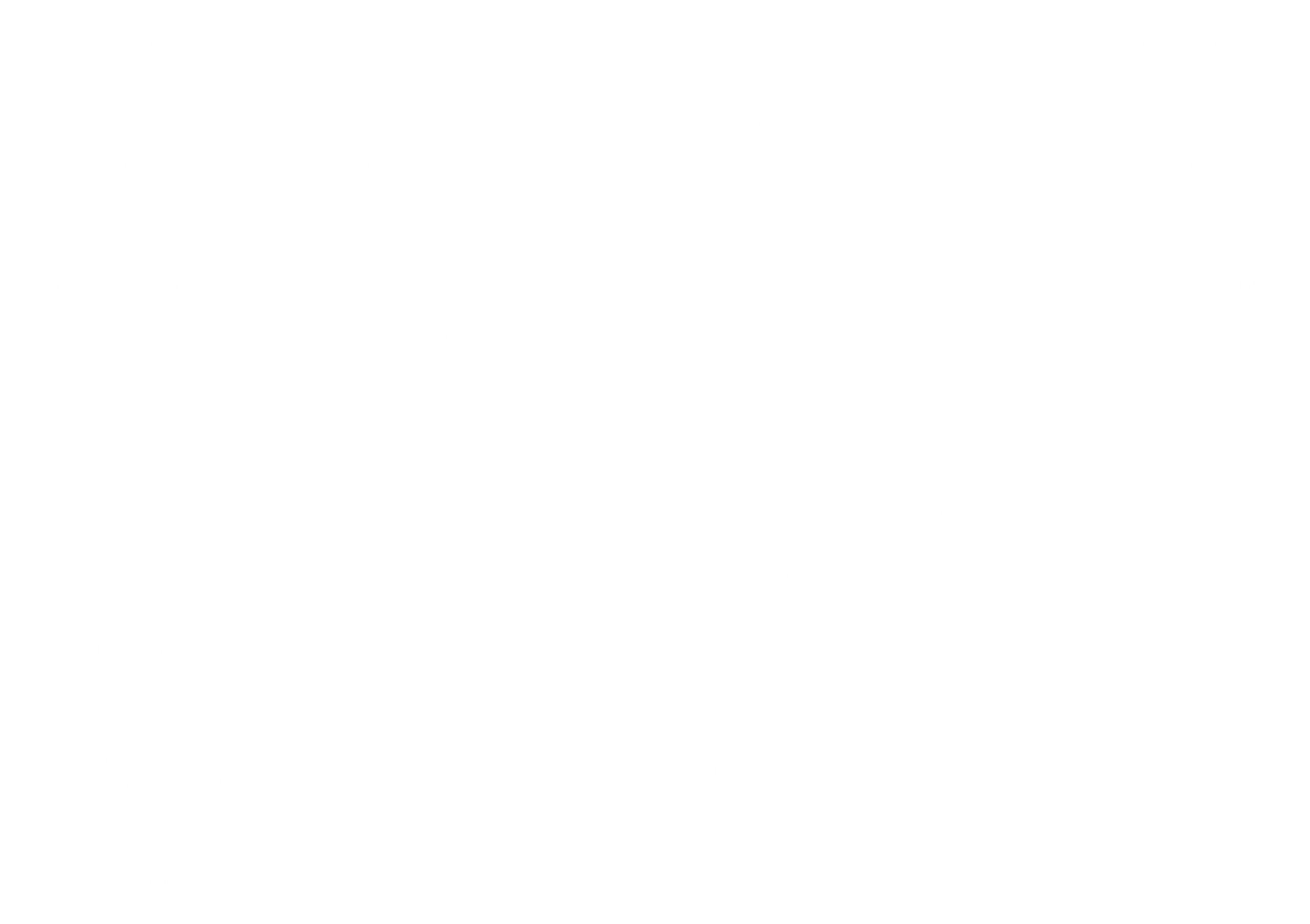


For a **supply** event, the following actions are taken:

1. The **supply event number** is incremented.
2. The **inventory level** is increased by the supply amount.
3. We update the **statistical variables**.
4. We update any **trace files**.

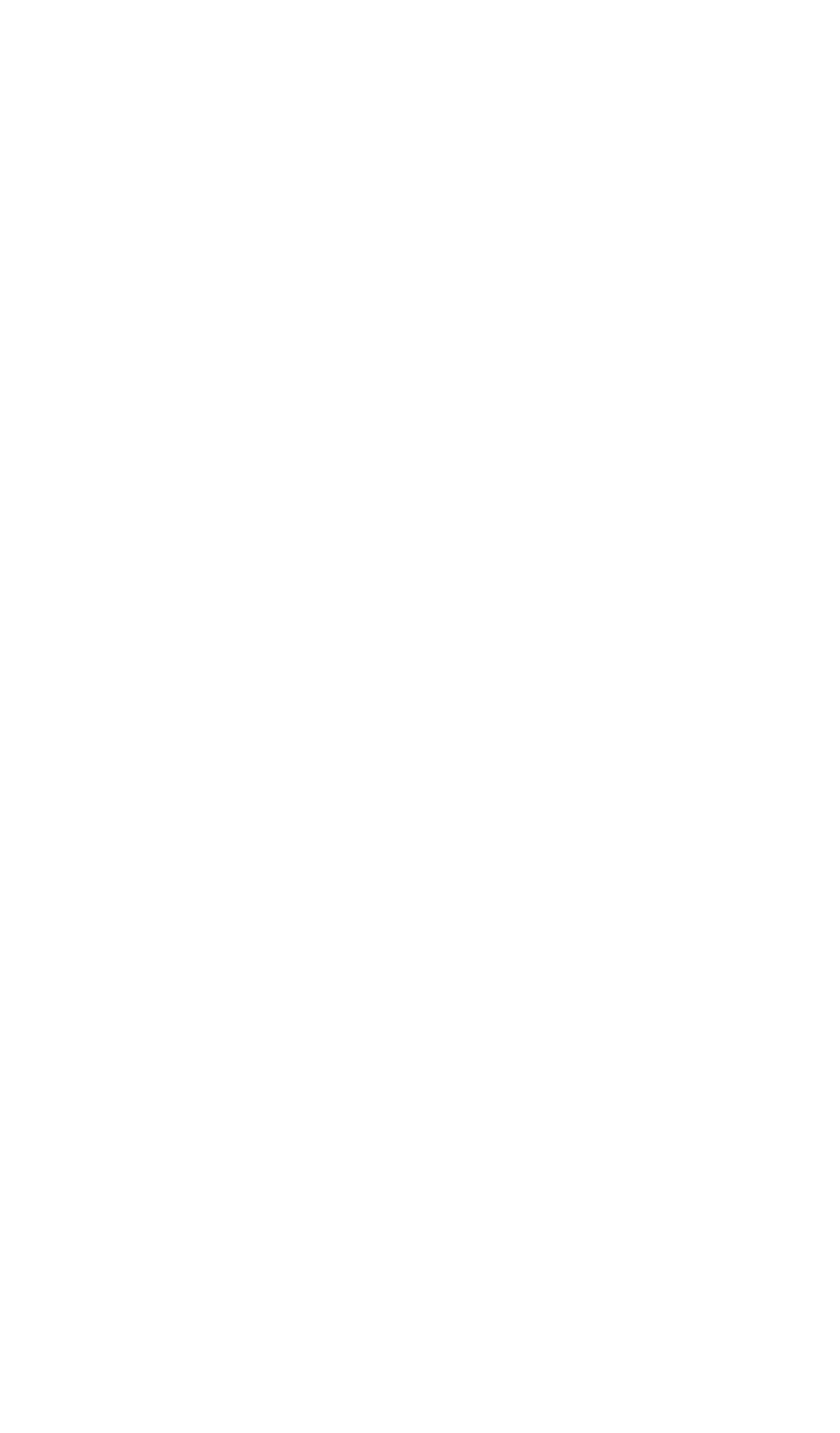
Notice how we do not schedule another supply event.

### Demand Handler



1. The **demand event number** is incremented.
2. The **demand amount** is found using a separate process. This process generates a random value and reports a demand amount based on it.
3. The **inventory level** is decremented by the demand amount.
4. The **statistical variables** and the **trace file** are updated.
5. We schedule another **demand event**. The time after which this event will occur is again found using a separate process that gives a random value based on an exponential distribution with the provided mean value, the exact same process used in the inventory initialization step.

### Evaluation Handler



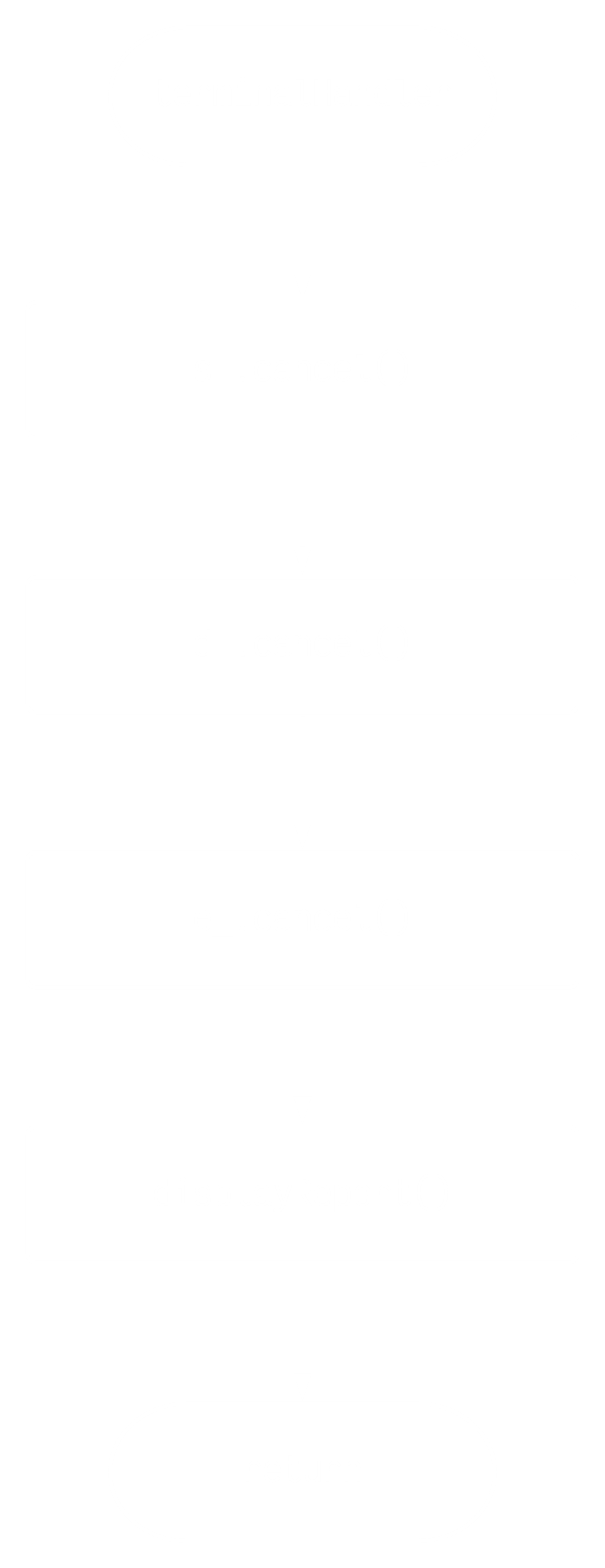
1. Increment the **evaluation event number**.
2. Check whether .

If it is:

* 1. Find the **order amount**, which is .
  2. Find the **order cost**, which is .
  3. Schedule a **supply** event. The time after which this event will occur is found using a separate process that generates a random value based on a **uniform distribution**. This process is a library function.

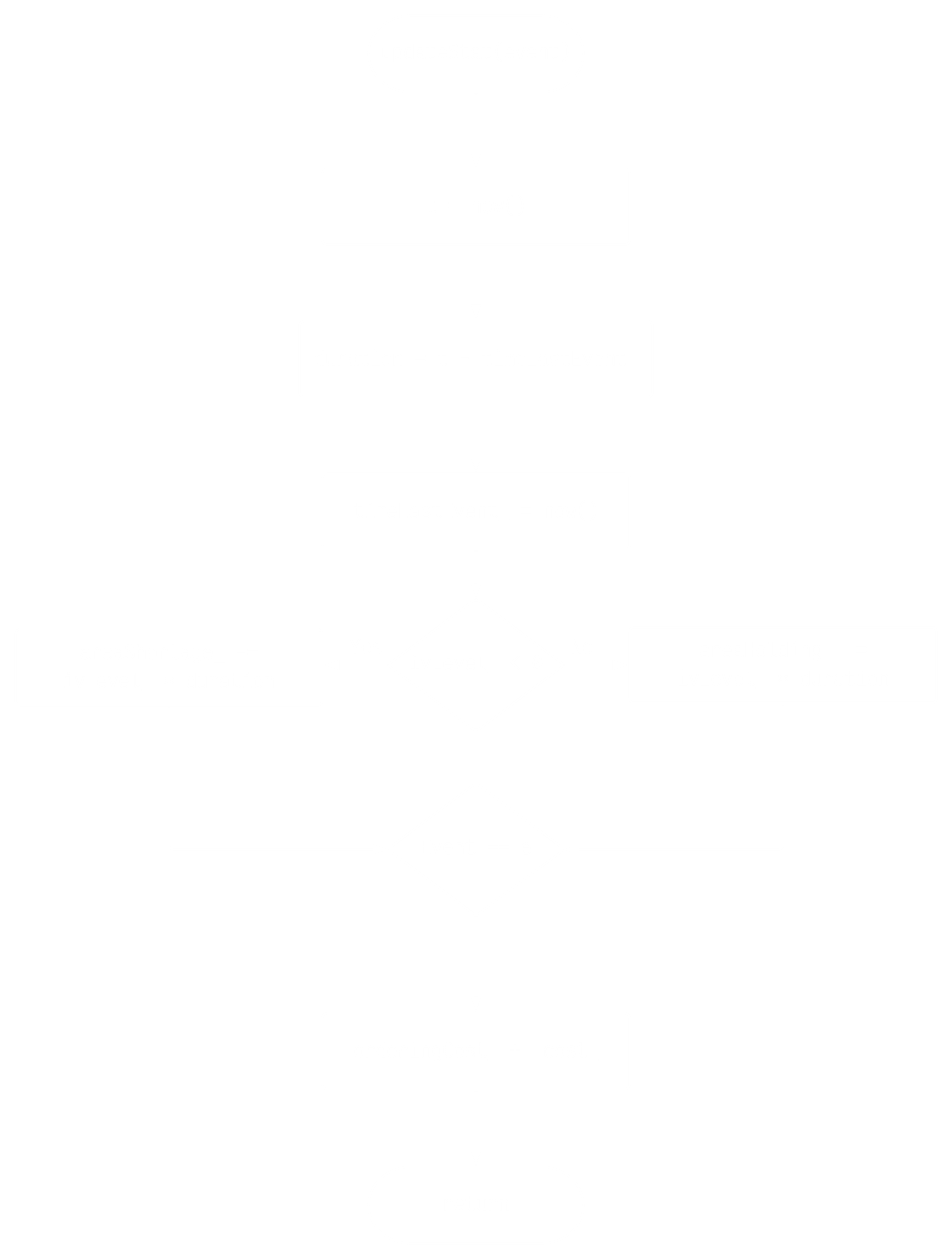
1. Update the **statistical variables**.
2. Update the **trace file**.
3. Schedule the next **evaluation event** with the parameter , since it will occur after exactly one month.

### Termination Handler



The **termination handler** is simple. It **cancels** any **pending events** and then calls a function that displays a **report**.

### Updating Statistical Variables



1. We find the **current time**, .
2. We find the **duration since last event**.
3. We update the **time of last event** to the current time.
4. If the **inventory level** is negative, the **shortage area** is decreased (since the inventory level is negative) by the product of the inventory level and the duration since the last event.
5. If the **inventory level** is positive, the **holding area** is increased by the product of the inventory level and the duration since the last event.
6. The **ordering cost** is updated.