***

Optimizing Motor Oil Supply Chain

-- One Hour Long Project --

performed live in front of you by
a consulting company and its client

Your Options:

A: Sit back, relax, and watch us doing the work

B: Follow the cheat sheet, and build the model at your own pace

(the model barely fits into one hour, we will be going very fast, and it will be virtually impossible to follow us on your laptops and stay in sync – no option C)

THE AnyLogic COMPANY

The Problem: Motor Oil Supply Chain. Overall Structure

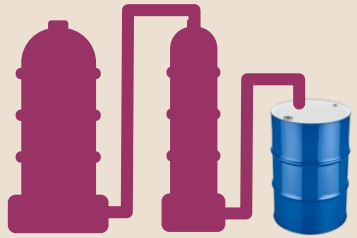
Find out optimal Inventory Policy parameters: s and S ensuring minimum Retailer's inventory and certain QoS

The Problem: Motor Oil Supply Chain. Numbers

[TO SAVE TIME DURING THIS WORKSHOP, WE WILL USE THE DEFAULT VOLUME AND TIME UNITS; **ASSUME ABSTRACT UNITS**]

- Each **Station** consumes oil at a constant rate; the rate however varies from station to station: uniform(0.2, 0.8) vol. units / time unit
- Initial amount of oil at a station also varies, distributed as uniform(5, 20) vol. units; max amount of oil at a station is 20 vol. units
- When the oil level reaches 5 vol. units, a station orders the next barrel from Retailer. Barrel capacity is 15 vol. units

- Upon receiving an order from a station, **Retailer** ships a barrel immediately, if not out of stock. Otherwise, keeps a backlog of orders.
- Implements (s,S) inventory policy, i.e., when the stock falls to s or below, places a replenishment order to restore it to S.
- The storage capacity is 500 barrels, initially the stock equals S.



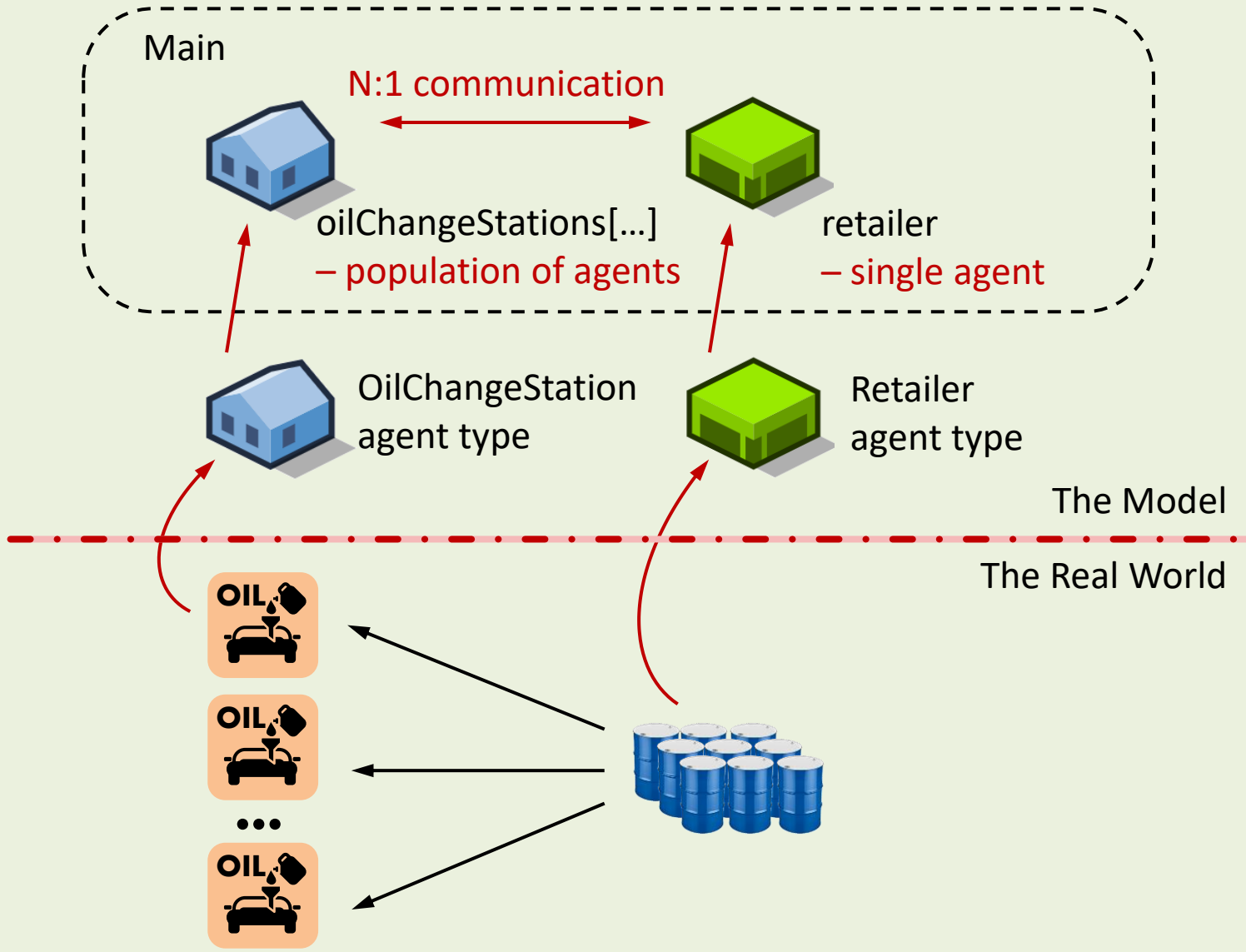
- **Factory** Produces on demand only, one order at a time.
- Total production time consists of the constant order setup time of 2 time units plus item production time of 0.04 time units per each barrel

Our goal: Minimize Retailer's inventory over time while keeping the maximum number of stations out of oil below **1%** at any time.

THE AnyLogic COMPANY

Conceptualization: High-level architecture of the model: **Agent-Based**

- Let's start with the high-level structure, say, the retailer (we have only one) and oil change stations (we have many of those) – and we will model the factory later
- We will need two new agent types: OilChangeStation and Retailer
- We will place instances of both types into the Main agent and then (at a later phase) let them communicate

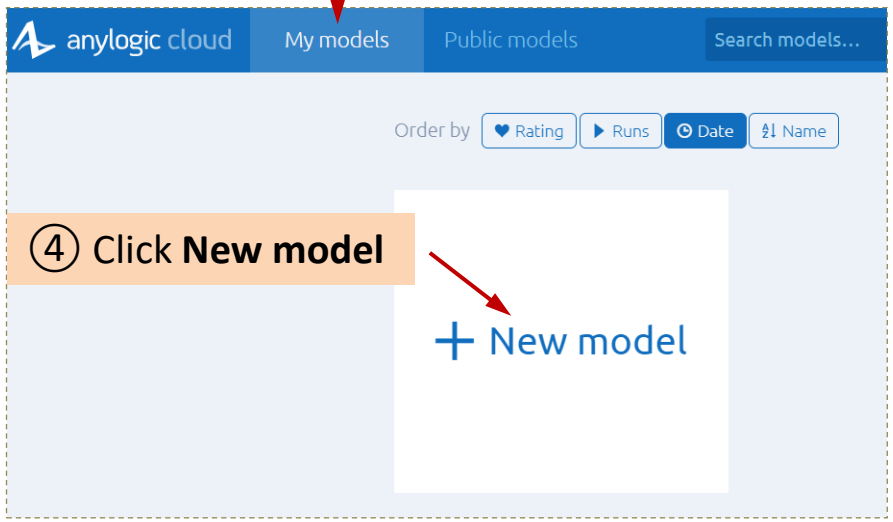


Log in to AnyLogic Cloud and create a new model

① Open https://cloud.anylogic.com

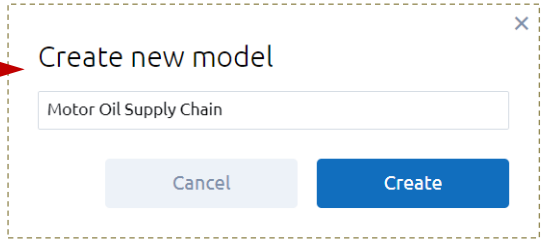
② If you are a Cloud subscriber, log in as usual. If you have signed up for the workshop, use the credentials you received to sign in.

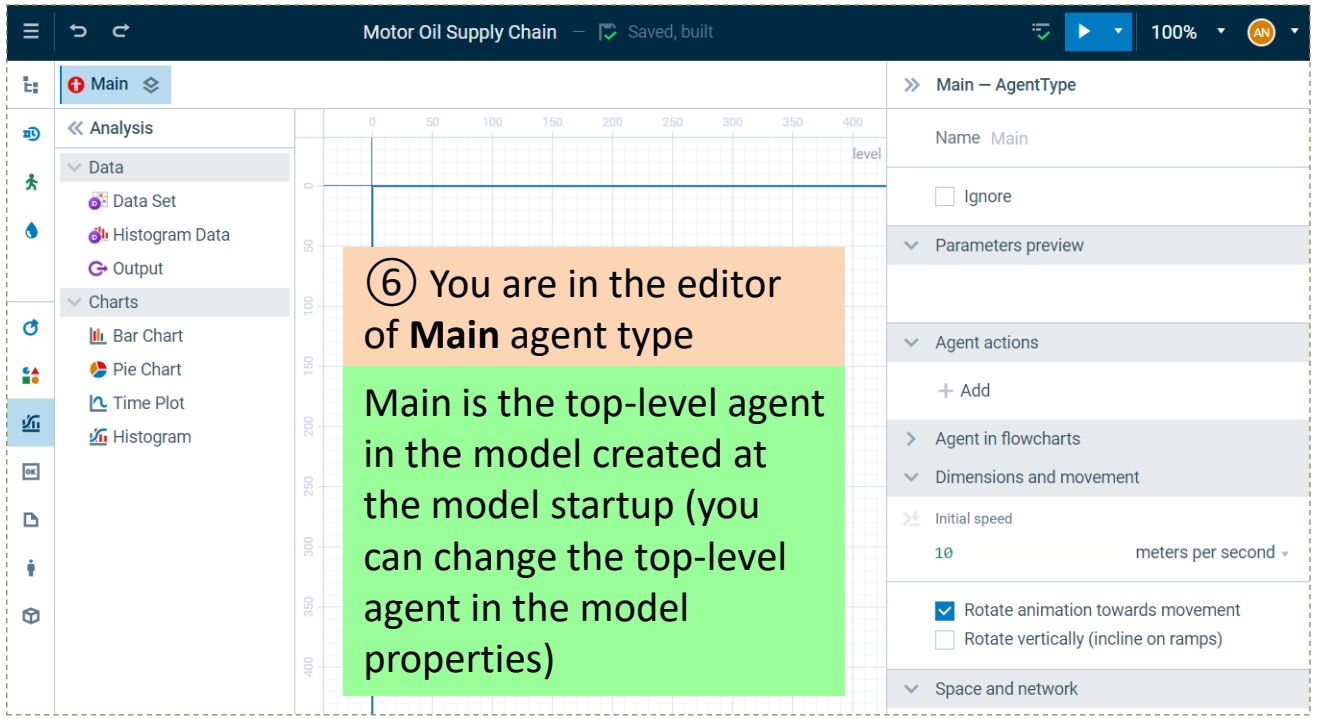
③ Switch to My models



④ Click **New model**

⑤ Name your model "Motor Oil Supply Chain"



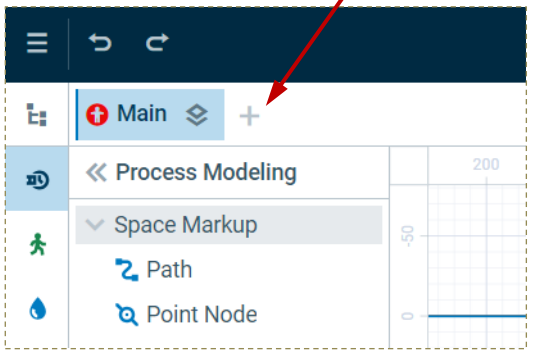


⑥ You are in the editor of **Main** agent type

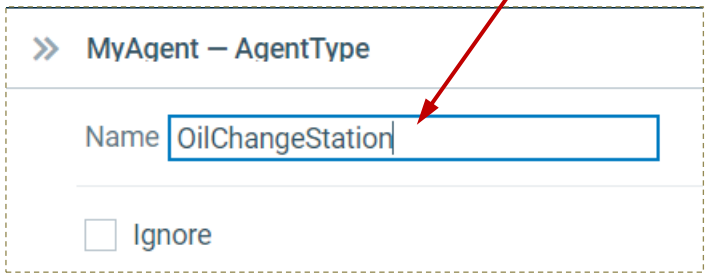
Main is the top-level agent in the model created at the model startup (you can change the top-level agent in the model properties)

Create a new Agent type: OilChangeStation

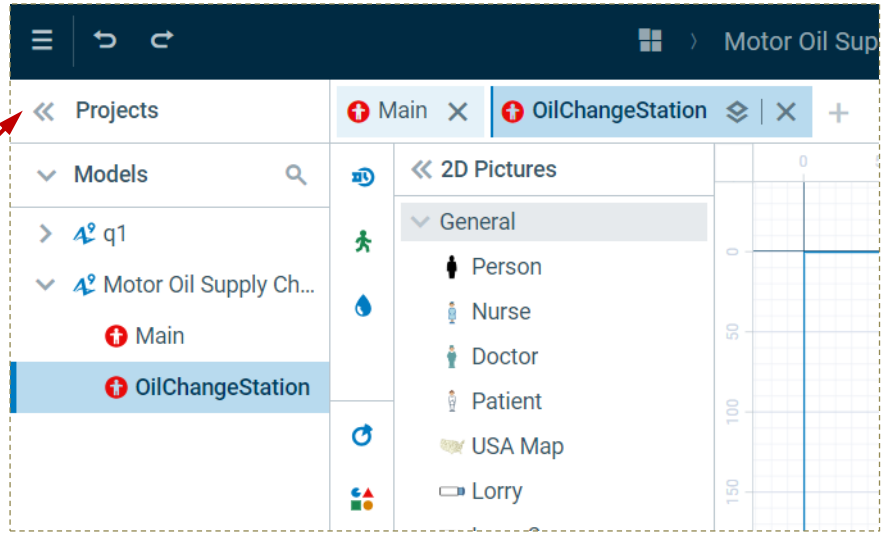
① Click "+" to the right of the **Main** tab to create a new agent type



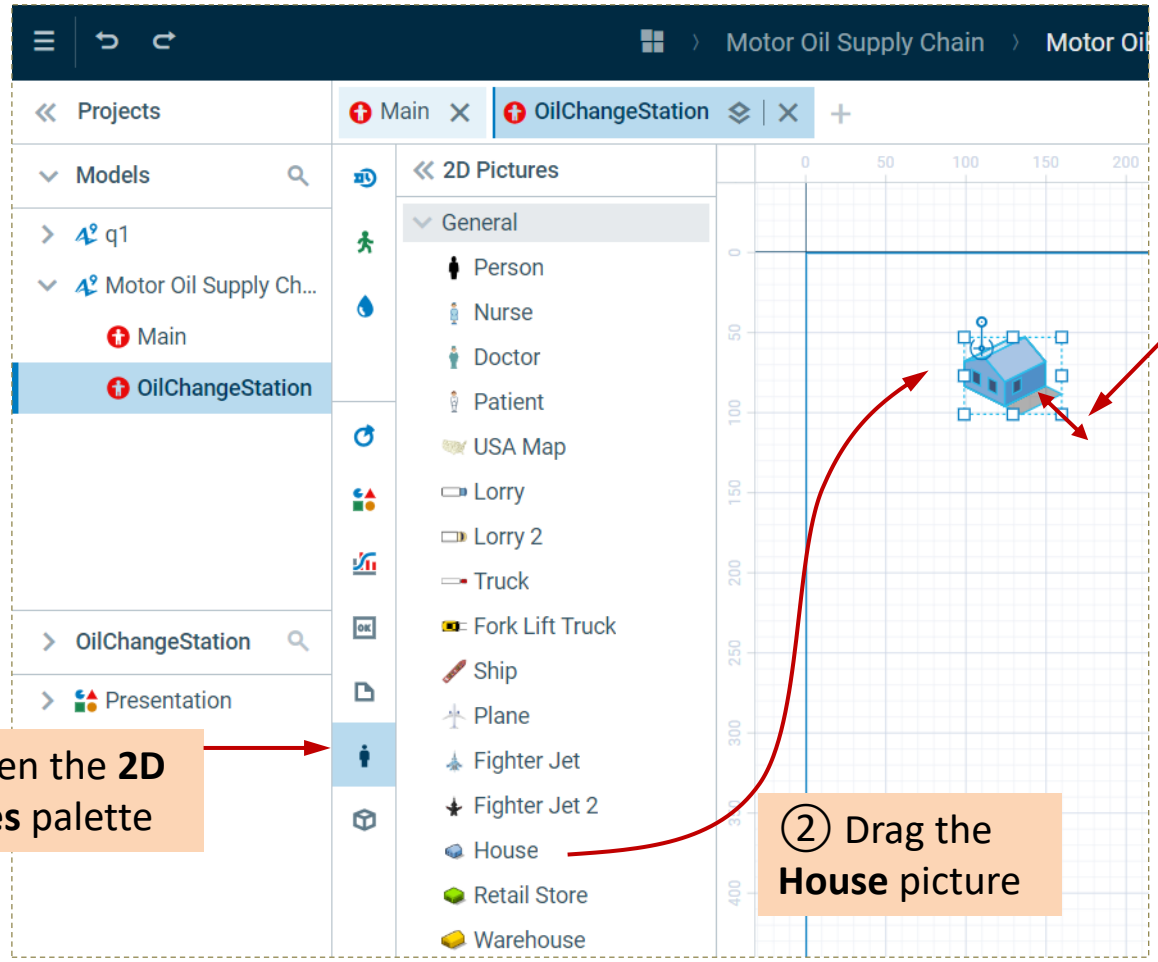
② Rename the newly created agent type to OilChangeStation



③ Open the **Projects** tree to see what agent types are present in the model



Create the icon of OilChangeStation

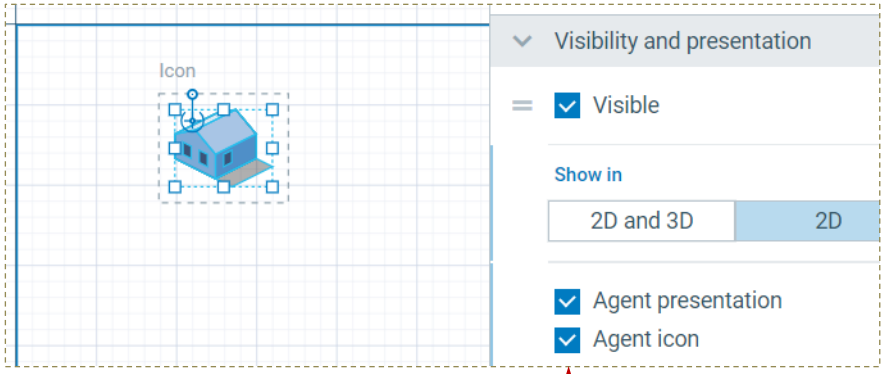


① Open the **2D Pictures** palette

② Drag the **House** picture

③ Enlarge it to approx. 50 x 50 pixels by Shift+Dragging the bottom-right handle

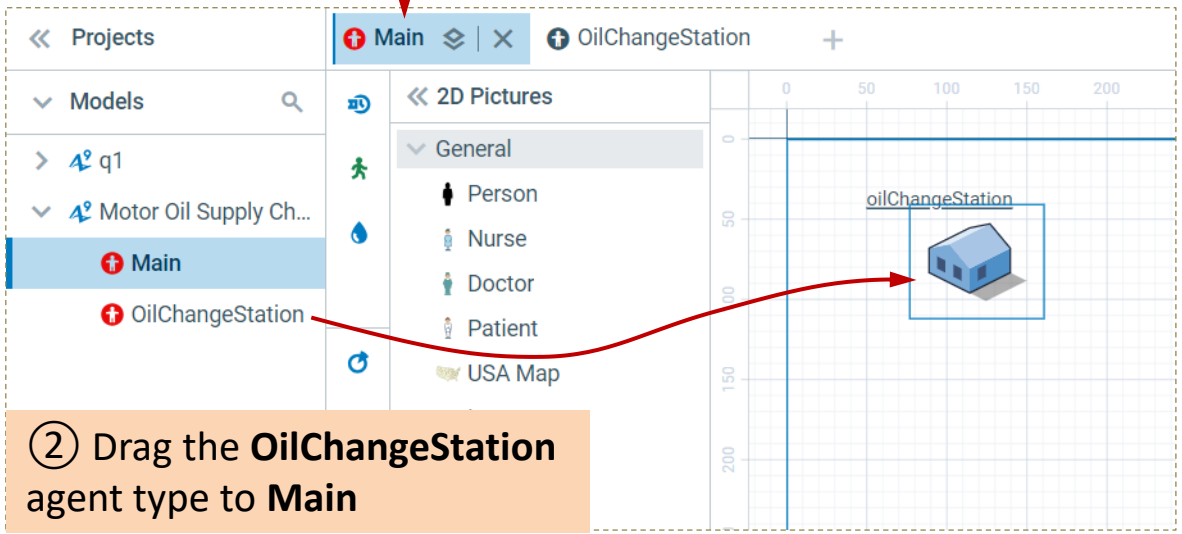
Pressing Shift keeps original aspect ratio



④ In the properties of the picture, make the picture the agent's icon

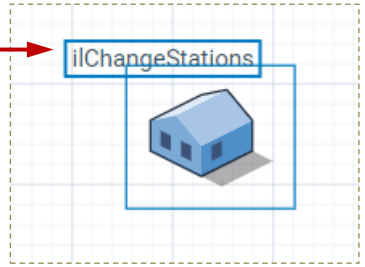
Create a population of oil change stations

① Switch to **Main** agent type

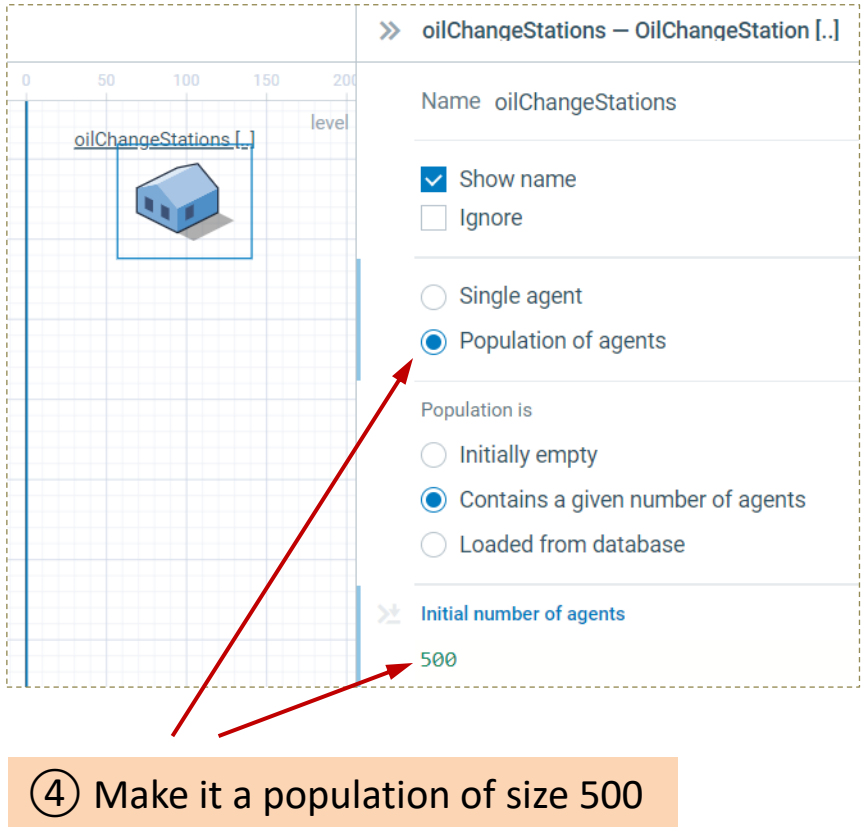


② Drag the **OilChangeStation** agent type to **Main**

③ Double-Click the instance name and rename it to oilChangeStations



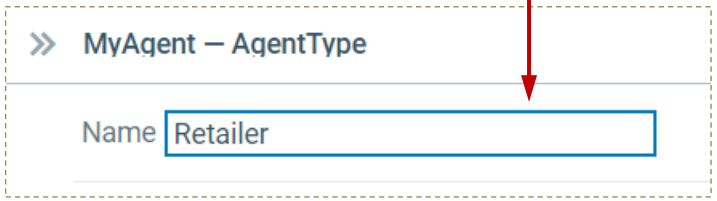
The [..] sign indicates the population

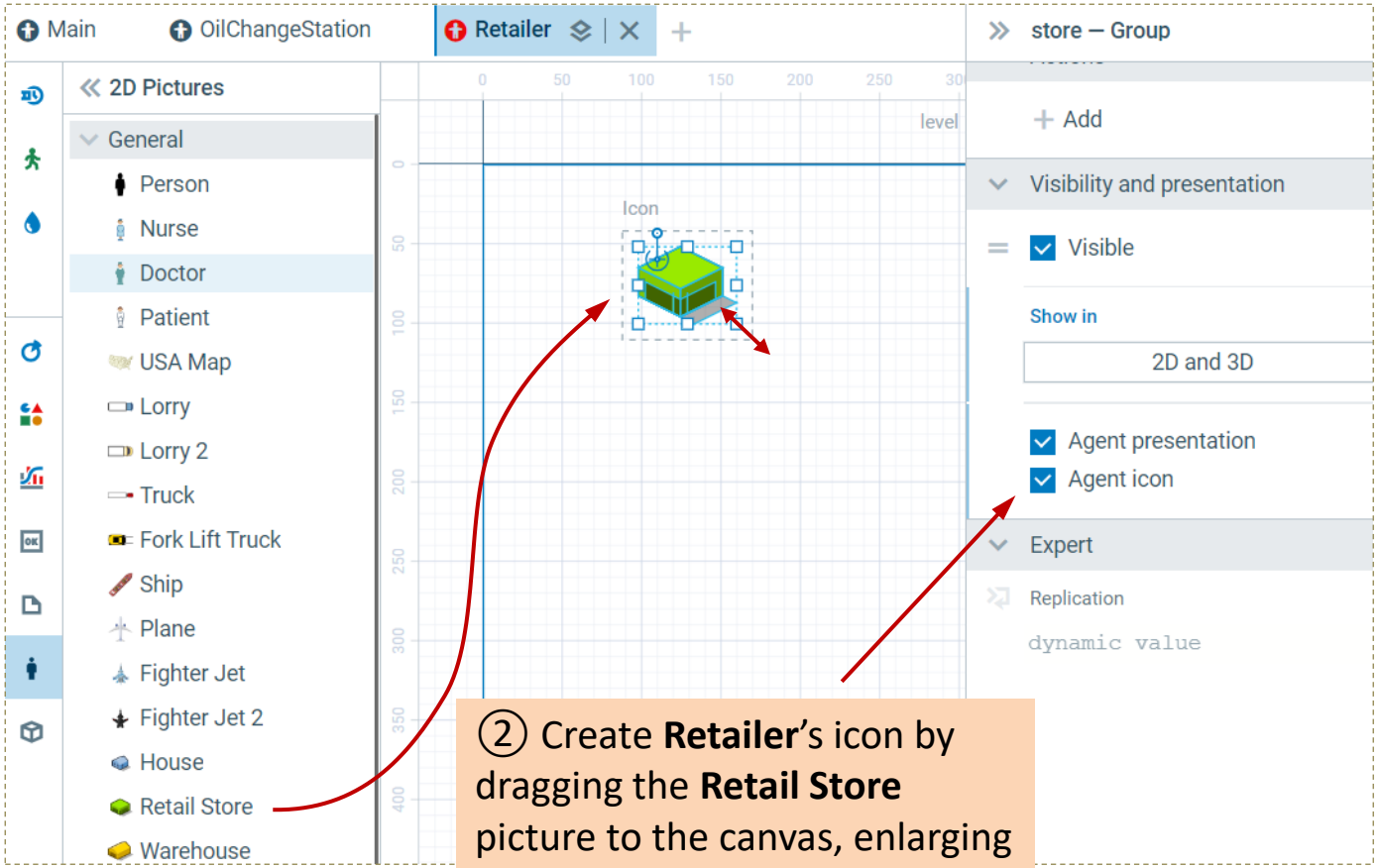


④ Make it a population of size 500

Create a new agent type Retailer and create its icon

① Create a new agent type Retailer by clicking "+" to the right of the agent tabs and changing the name to Retailer

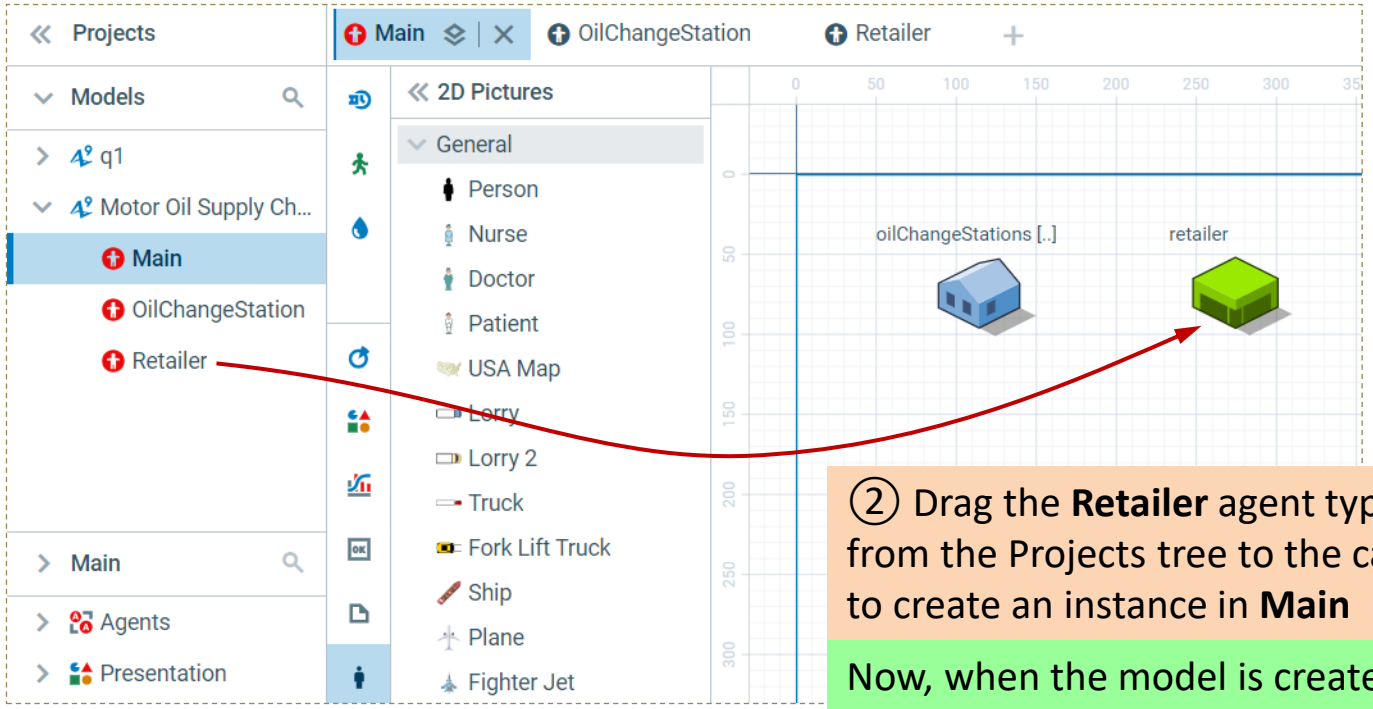




② Create **Retailer**'s icon by dragging the **Retail Store** picture to the canvas, enlarging it, and selecting the **Agent icon** property

Create an instance of Retailer on Main

① Switch to **Main** agent type

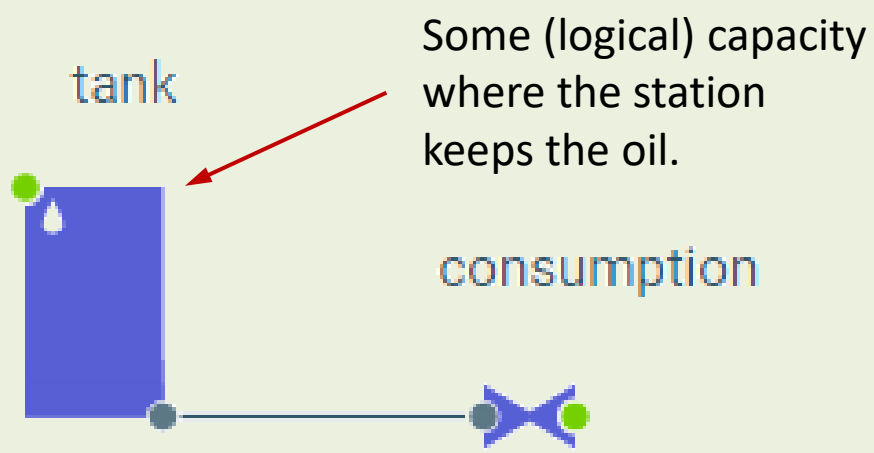


② Drag the **Retailer** agent type from the Projects tree to the canvas to create an instance in **Main**

Now, when the model is created, it will create the top-level agent of type Main and then a population of **OilChangeStation** agents and a single **Retailer** agent inside (or embedded into) **Main**.

Conceptualization: behavior of OilChangeStation: **Fluid Library**

- Let's first focus on oil consumption, we will add oil re-ordering later
- Although in reality consumption happens by portions (per car) and has some daily and weekly patterns, we are allowed to assume it to be at a constant rate: we have not been given any other details (and we actually do not need them, given the scale)
- A very simple flowchart of Fluid Library blocks can easily handle it



The Model

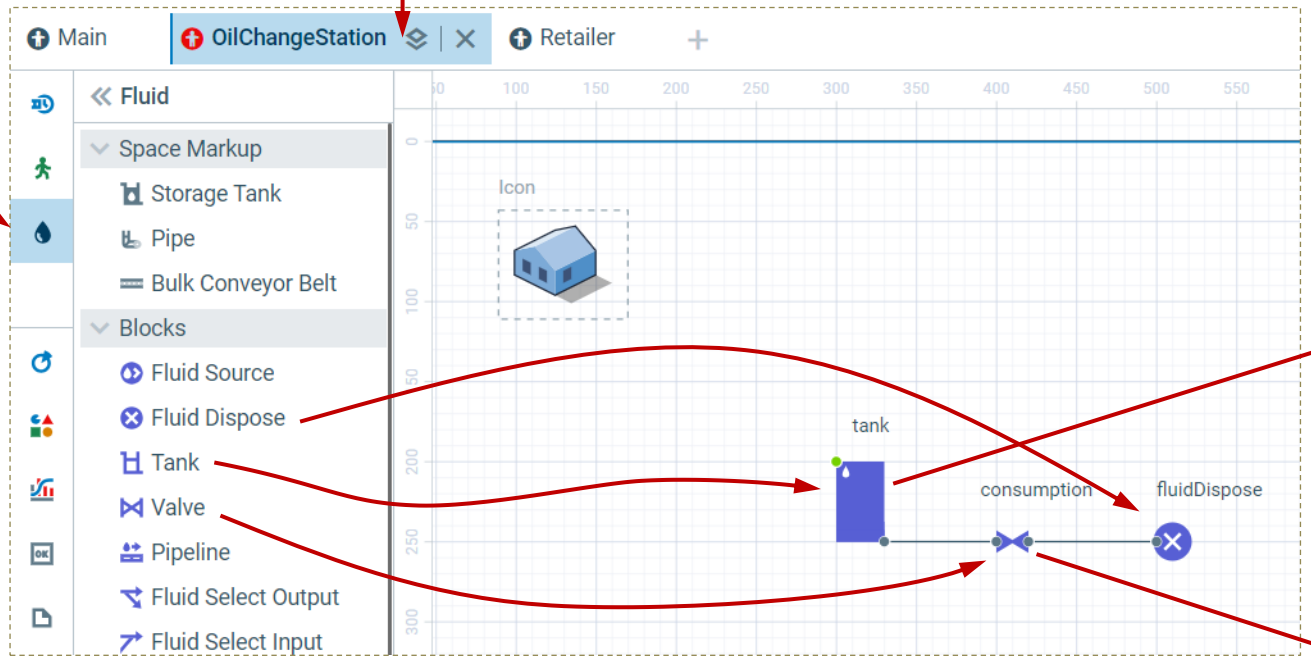
The Real World

- Each station consumes oil at a constant rate; however, the rate varies from station to station: uniform(0.2, 0.8)
- Initial amount of oil at a station also varies, distributed as uniform(5, 20)

Model the behavior of OilChangeStation

① Switch to **OilChangeStation** agent type

② Open the **Fluid** palette



③ Drag these blocks to the canvas

As you drag a block to the right of the other block, they connect automatically. You can also connect ports manually by clicking on them.

④ Name the blocks and set up their parameters:

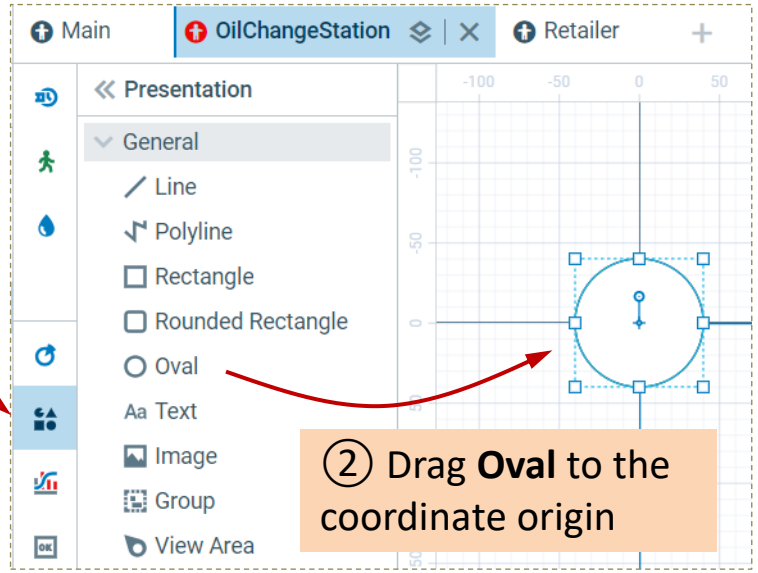
Name: tank
Capacity: 20
Initial amount: uniform(5, 20)

Name: consumption
Rate when open: uniform(0.2, 0.8)

[Remember, to save time, we are using the **default units**, which in this case are cubic meter and seconds]

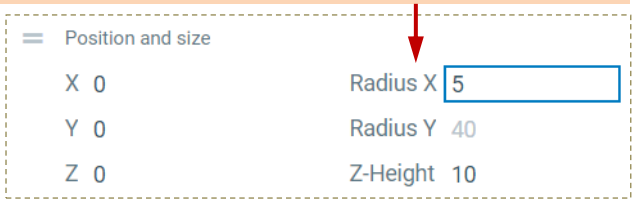
Create animation of OilChangeStation: we will animate the oil level

① Open the **Presentation** palette

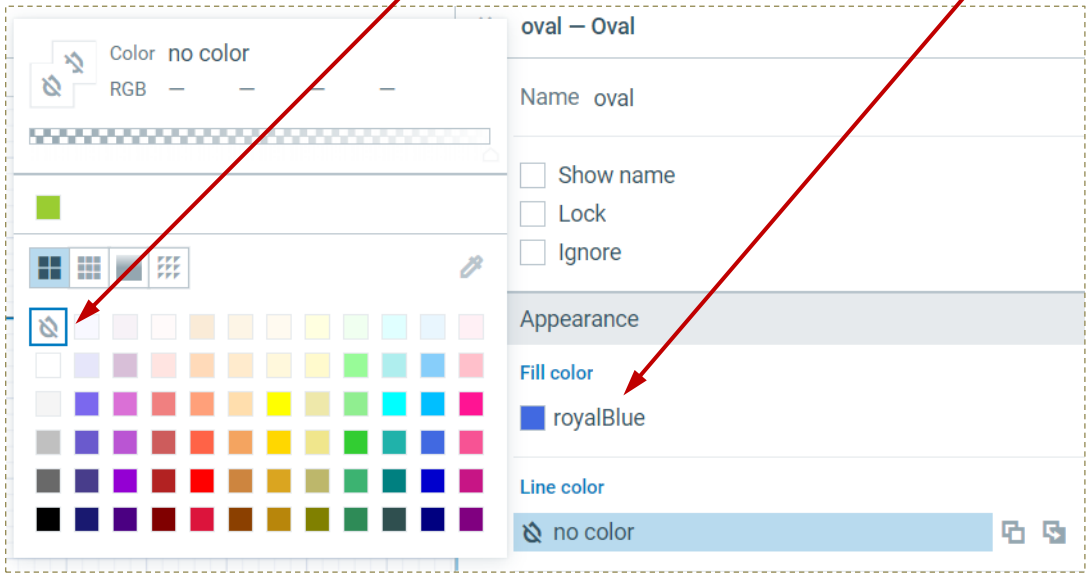


② Drag **Oval** to the coordinate origin

③ In the properties, set the oval radius to 5



④ Set **Line color** to No color and **Fill color** to e.g. royalBlue



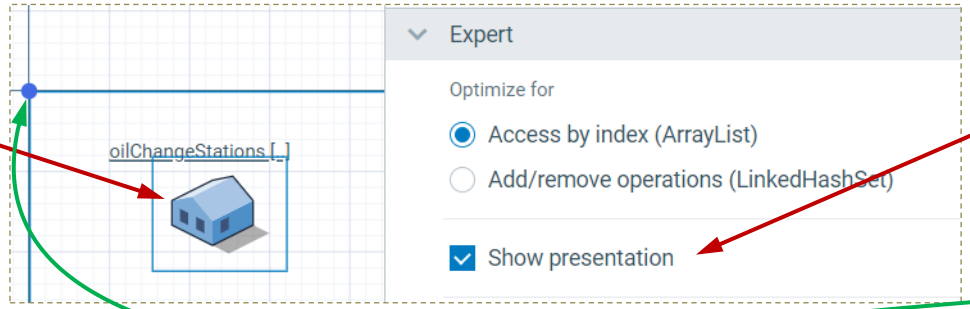
⑤ Click the = icon next to **Fill Color** to display its **Dynamic value** and set it as shown

When the model is running, and animation is shown, the color of the oval will be recalculated dynamically and will change according to this expression

Fill color (dynamic value):
tank.amount() > 5 ? royalBlue :
tank.amount() > 0 ? peru : red

Set up animation of the of oilChangeStations population

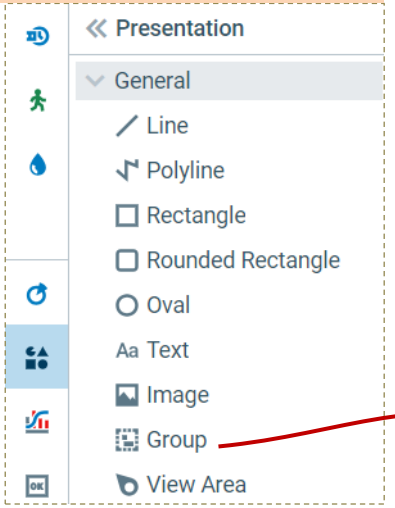
① Switch to **Main** agent type and select **oilChangeStations**



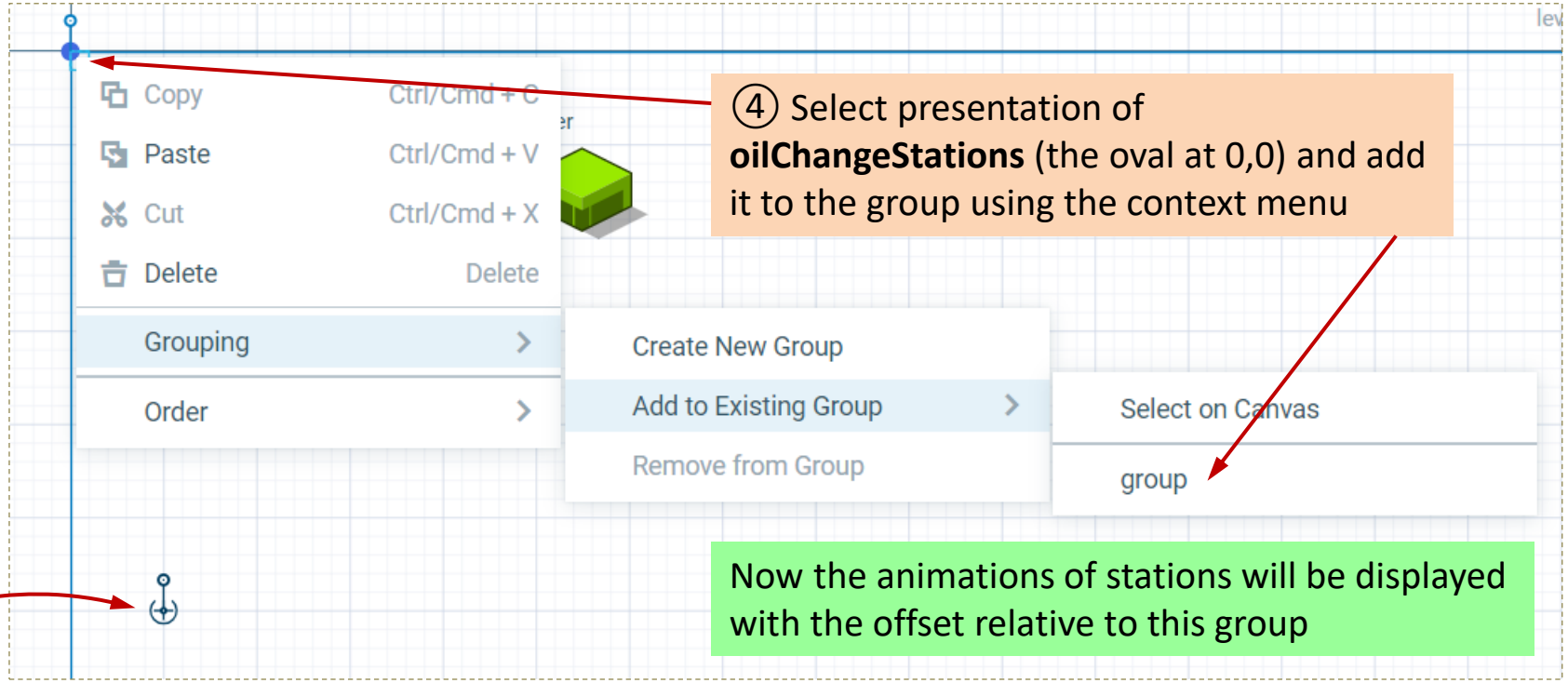
② In the **Expert** section, check **Show Presentation**

The presentation (the oval) then appears at (0,0)

③ From the **Presentation** palette, drag the **Group** to approx. (50,300)



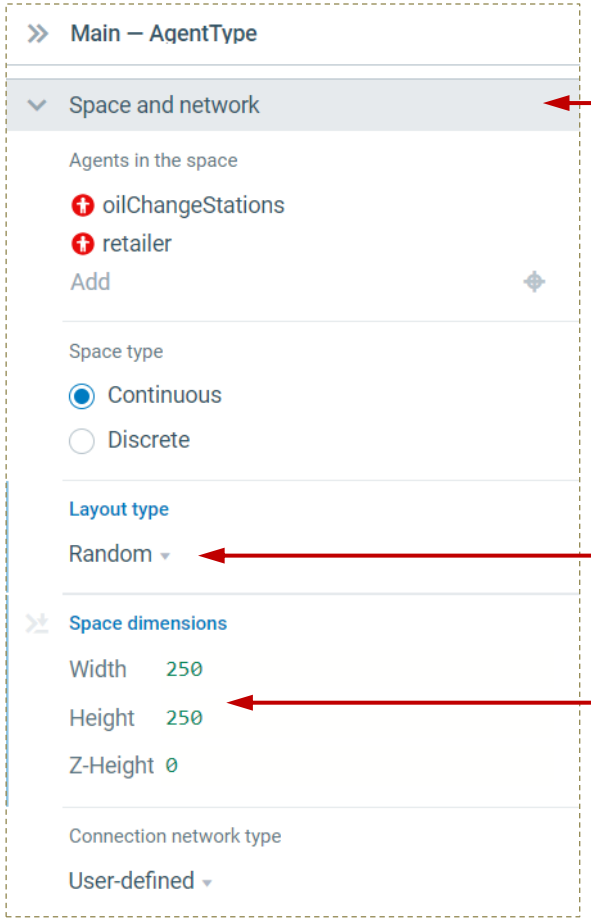
④ Select presentation of **oilChangeStations** (the oval at 0,0) and add it to the group using the context menu



Now the animations of stations will be displayed with the offset relative to this group

Final step before we run the model: set the layout of the population

① Staying in **Main** agent type, click on an empty place on the canvas (to remove any possible selection) to display **Main** agent type properties



② In the properties, go to **Space and network**

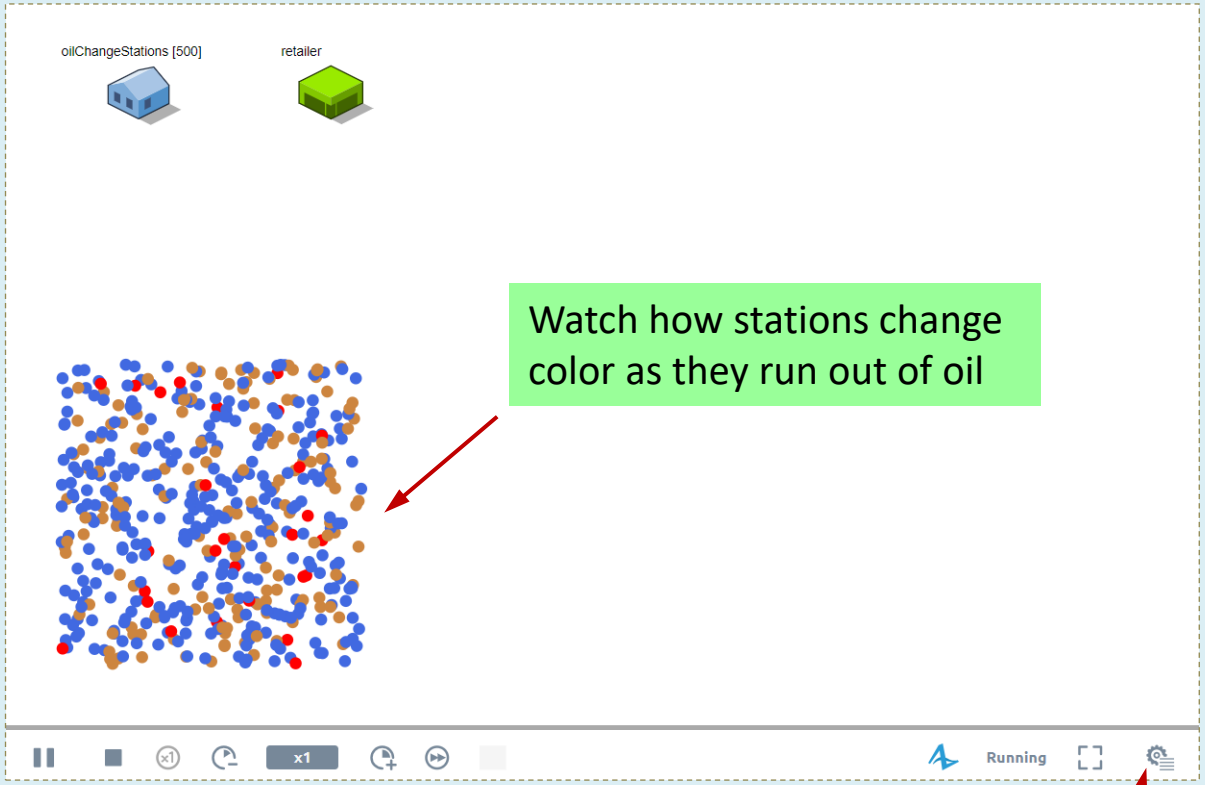
③ Set **Layout type** to Random

④ Set the **Space dimensions** to 250 x 250

This is not a part of the model logic and will (in our case) not affect anyhow the model behavior. We did it entirely for visualization purposes

Run the model to see the population of oil change stations

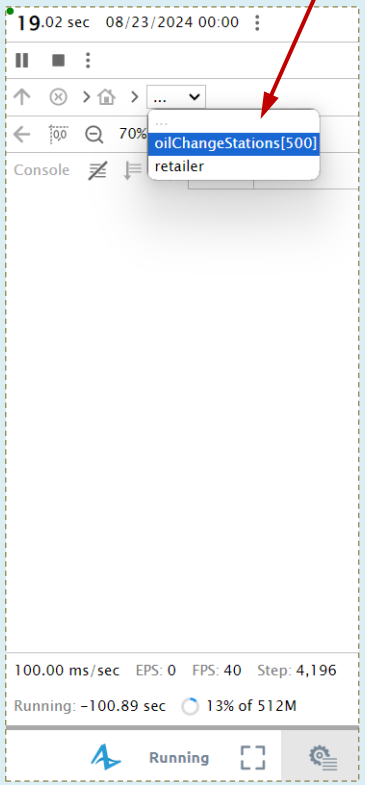
① Press here to run animated simulation over the editor window

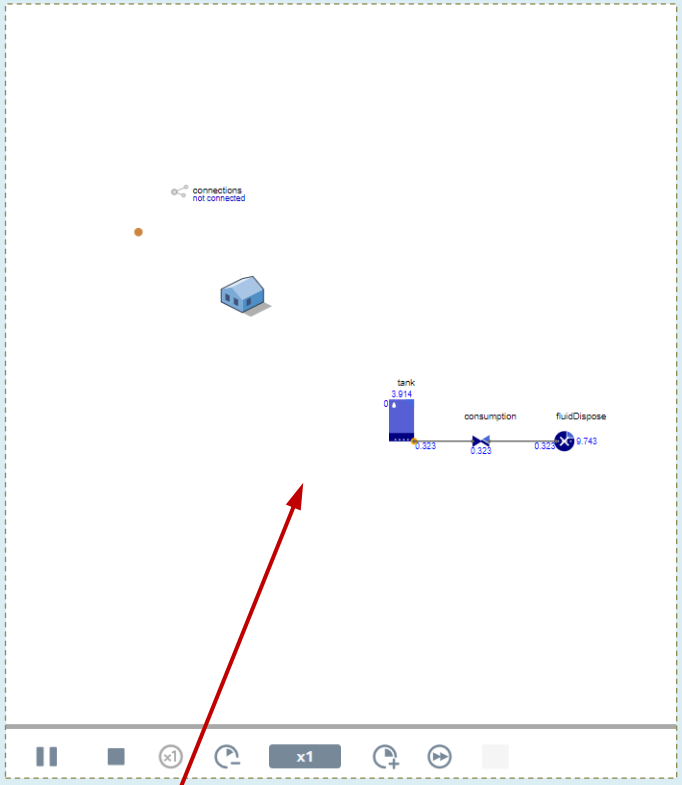


Watch how stations change color as they run out of oil

② You can also open the **Developer panel**

③ Dive into an **oilChangeStation**

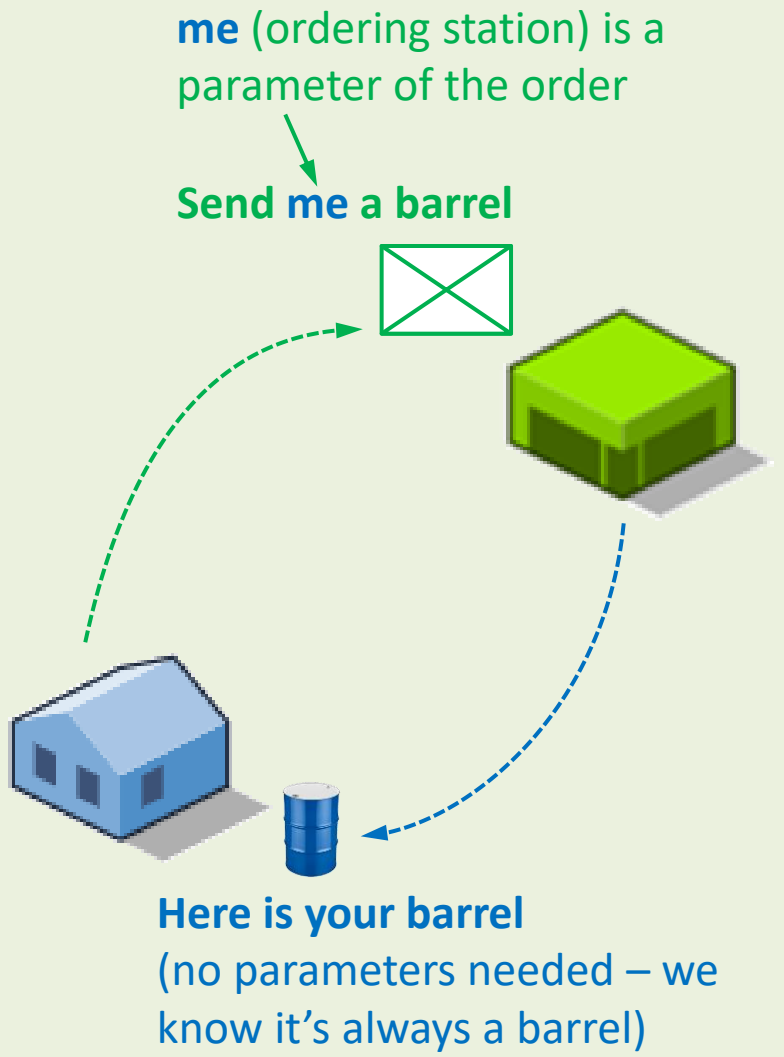




④ And watch what's happening inside

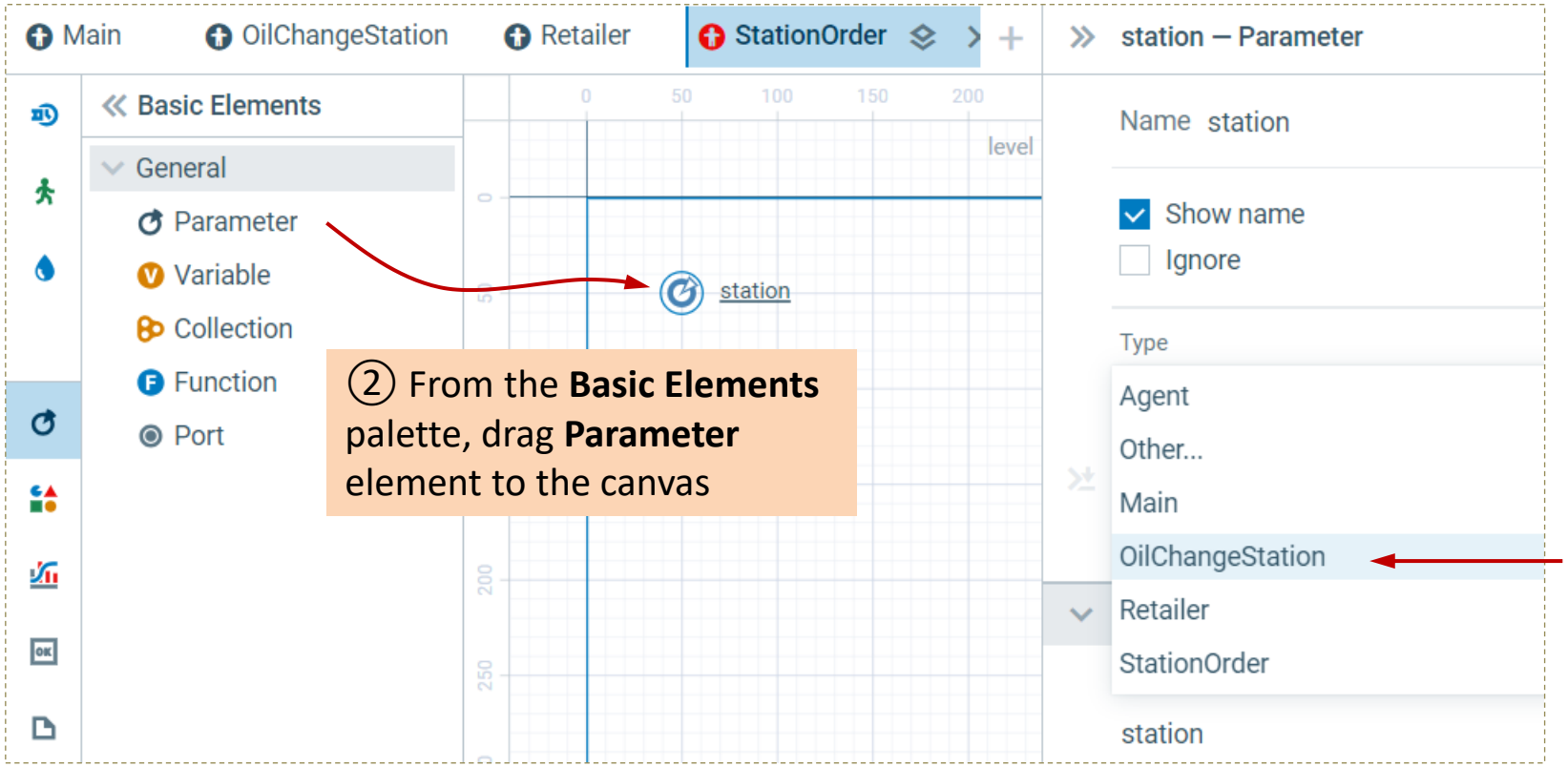
Conceptualization: Communication between the stations and the retailer

- This is a part of agent-based modeling and probably, the trickiest part of the model
- Although motor oil is consumed continuously as a fluid substance, it is ordered and delivered in discrete items (barrels)
- Therefore, we will need discrete communication between the stations and the retailer: a station will send an order to the retailer, and the retailer will send a barrel of oil back
- While there is only one retailer in the model, there are many stations, and the order should somehow identify the station to deliver the oil ("ship to")
- We will use references, or "addresses" to deliver orders and shipments directly to the retailer's and station's entry points



Create a new agent type StationOrder

① Create a new agent type StationOrder



② From the **Basic Elements** palette, drag **Parameter** element to the canvas

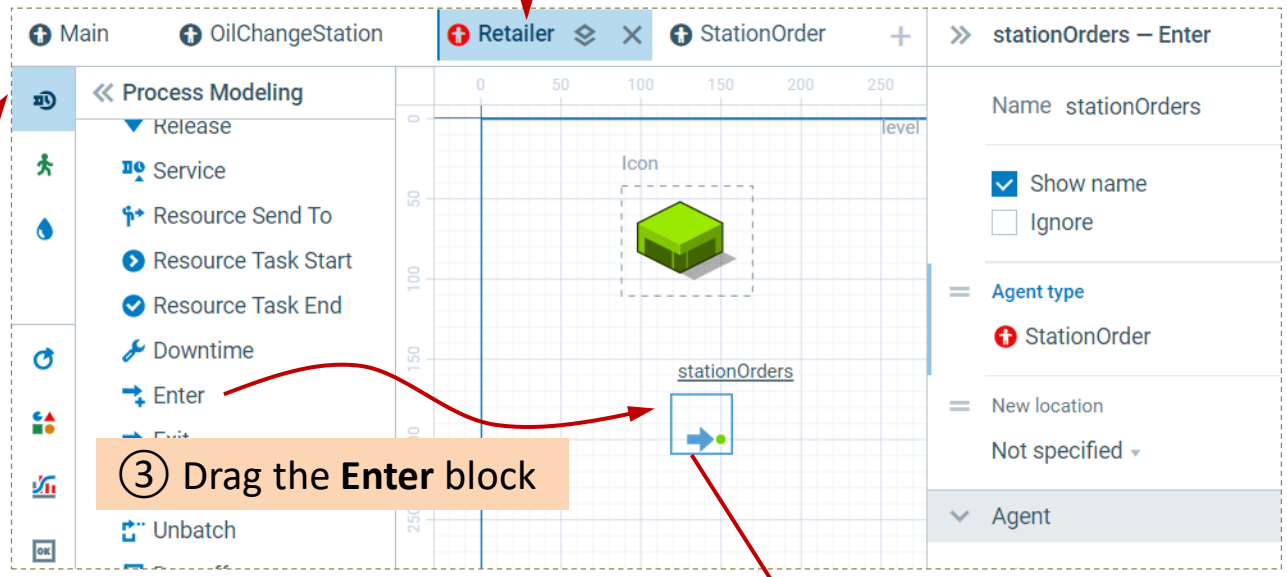
③ Set the parameter type to OilChangeStation (this is an agent type in our model, it can be found in the dropdown list)

In AnyLogic, everything is an agent: a representation of a system with complex behavior such as a warehouse, or a workstation, as well as a simple thing like an entity moving through a process such as document, patient, part, or shipment. In this case, an agent StationOrder is a representation of order holding just one parameter "**shipping address**" and having no behavior.

In the Retailer, create the “order entry point”

① Switch to **Retailer** agent type

② Open the **Process Modeling** palette



③ Drag the **Enter** block

④ Name the block and set its **Agent type** parameter as shown

Name: stationOrders
Agent type: StationOrder

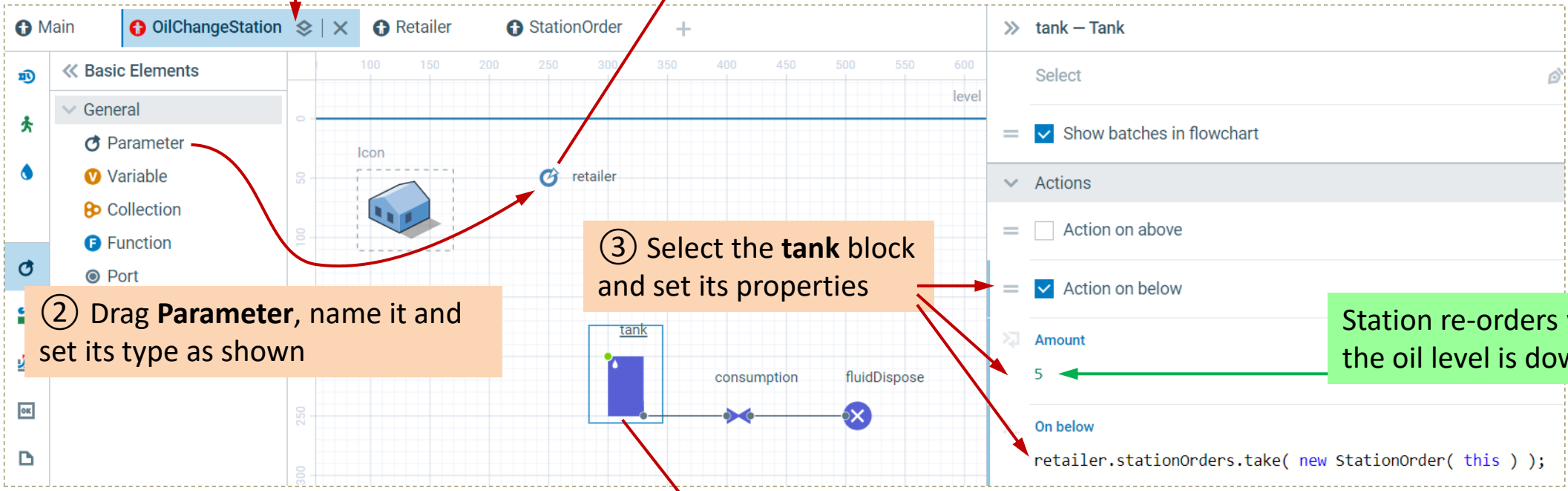
The Enter block here at Retailer is the initial block of a flowchart. By setting the Agent Type here, you inform the flowchart about what type of entities (agents) to expect in the flowchart.

Model Station to Retailer interaction: ordering oil

① Switch to **OilChangeStation** agent type

Name: retailer
Type: Retailer

We will use this as the "retailer's address" known to the station. There are also other ways to access the retailer



② Drag **Parameter**, name it and set its type as shown

③ Select the **tank** block and set its properties

Station re-orders when the oil level is down to 5

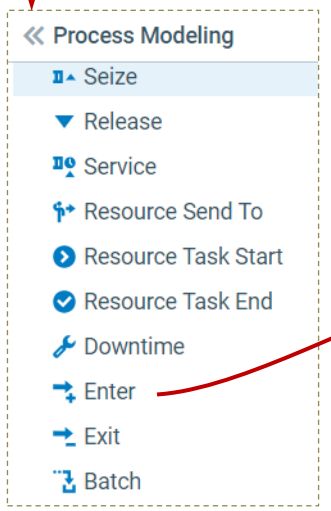
☑ actionOnBelow
On below: retailer.stationOrders.take(new StationOrder(this));

Access retailer's "**order entry point**" we have created

Send there a new order where "**ship to address**" is this station

Model delivery of motor oil to the station (it is always one barrel)

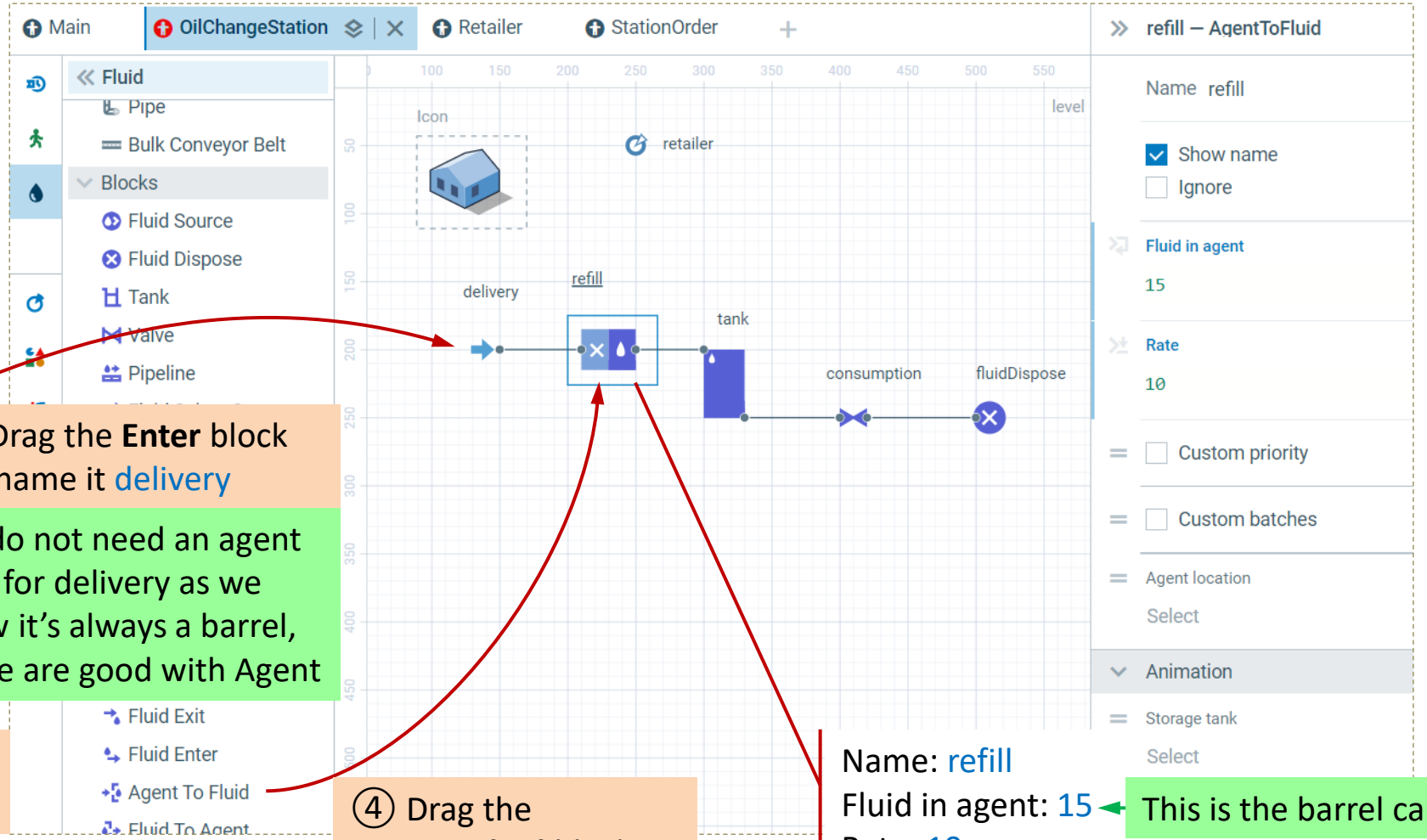
① Open the **Process Modeling** palette



② Drag the **Enter** block and name it delivery

We do not need an agent type for delivery as we know it's always a barrel, so we are good with Agent

③ Open the **Fluid** palette



④ Drag the **AgentToFluid** block, name it and set its properties as shown

Name: refill
Fluid in agent: 15
Rate: 10

This is the barrel capacity

And this is some fast rate of emptying the barrel into the tank (let's assume it happens this way)

Let the stations know the retailer's "address"

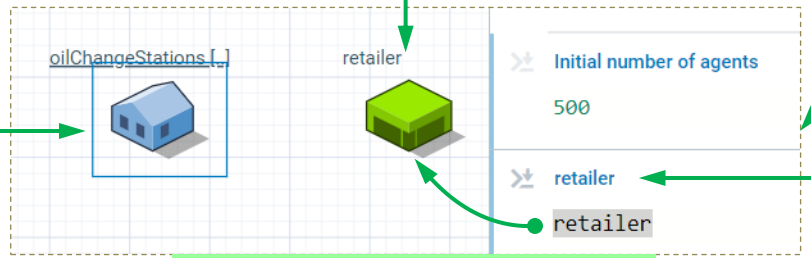
① Switch to **Main** agent type – the top-level agent in the model

② Select the **oilChangeStations** (a population of agents embedded in **Main**) and set its retailer parameter to retailer

[In the screenshot below, there are three "retailers":) Do not get confused, read the comments in green]

This is the retailer

These are the properties of the population of OilChangeStation agents



This is the population of oil change stations (selected)

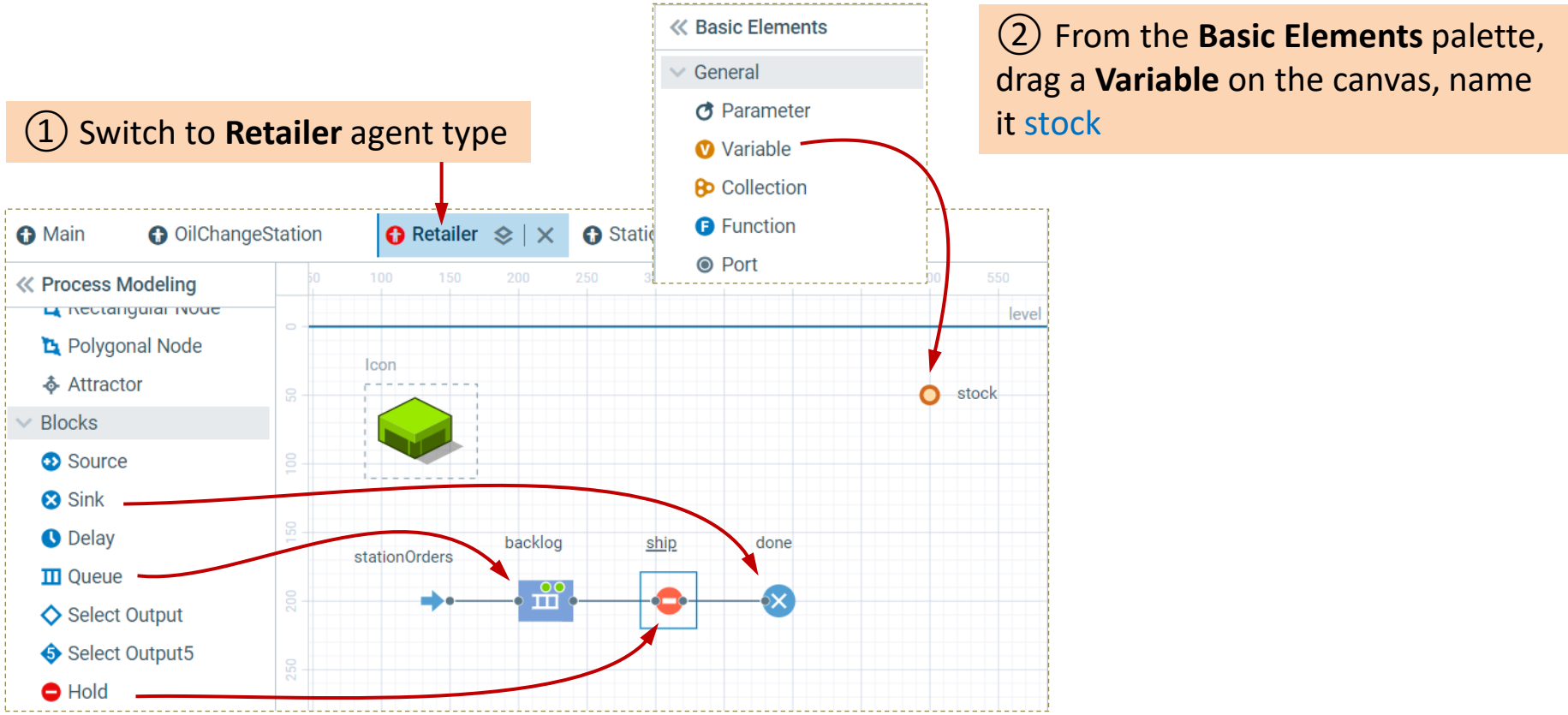
This is a parameter of OilChangeStation

This is a reference to the retailer agent. We set up this parameter to point to the retailer agent also embedded in Main so that now every station now knows the retailer

Model how the retailer ships oil barrels to the stations: blocks

① Switch to **Retailer** agent type

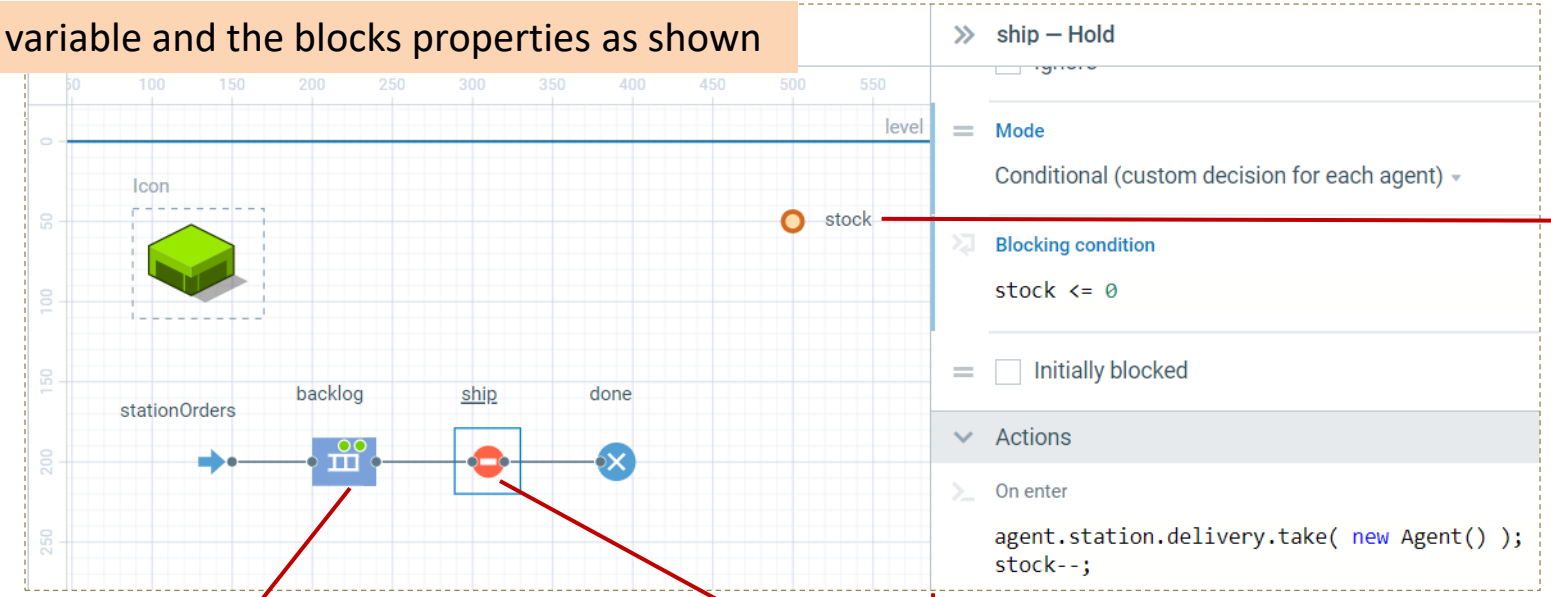
② From the **Basic Elements** palette, drag a **Variable** on the canvas, name it stock



③ From the **Process Modeling** palette, drag the three blocks, connect them, and set their names: backlog, ship, and done

Model how the retailer ships oil barrels to the stations: properties

① Set the variable and the blocks properties as shown



Name: stock
Type: int
Initial value: 500

For now, we will assume maximum stock in the beginnig

Name: backlog
☑ Maximum capacity

Name: ship
Mode: Conditional
Blocking condition: stock <= 0
On enter:
agent.station.delivery.take (new Agent());
stock--;

A barrel – abstract agent

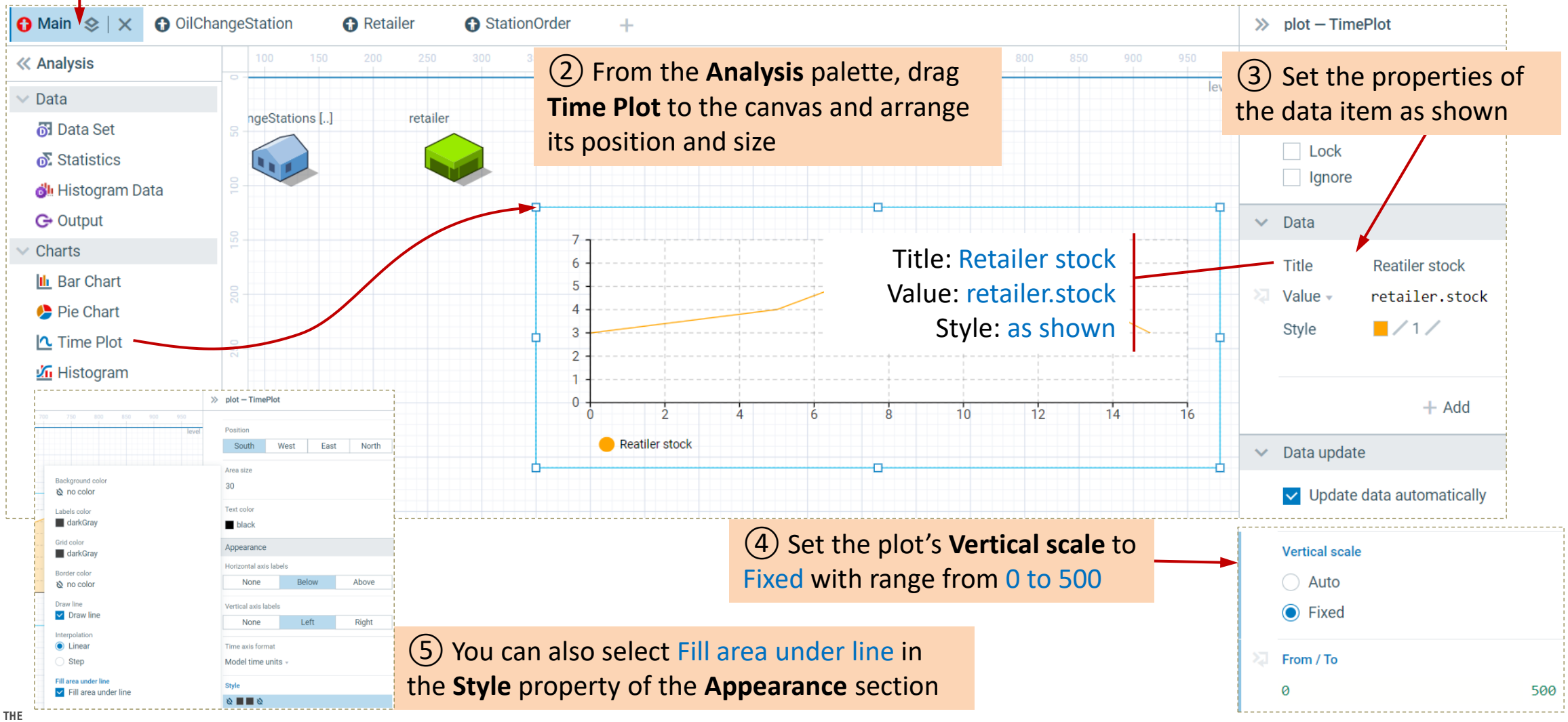
"agent" here is the station's order

Station's "address" (a part of order)

Station's "entry point" for shipments

Last bit before we run the model: add a time plot to watch retailer's stock

① Switch to **Main** agent type



② From the **Analysis** palette, drag **Time Plot** to the canvas and arrange its position and size

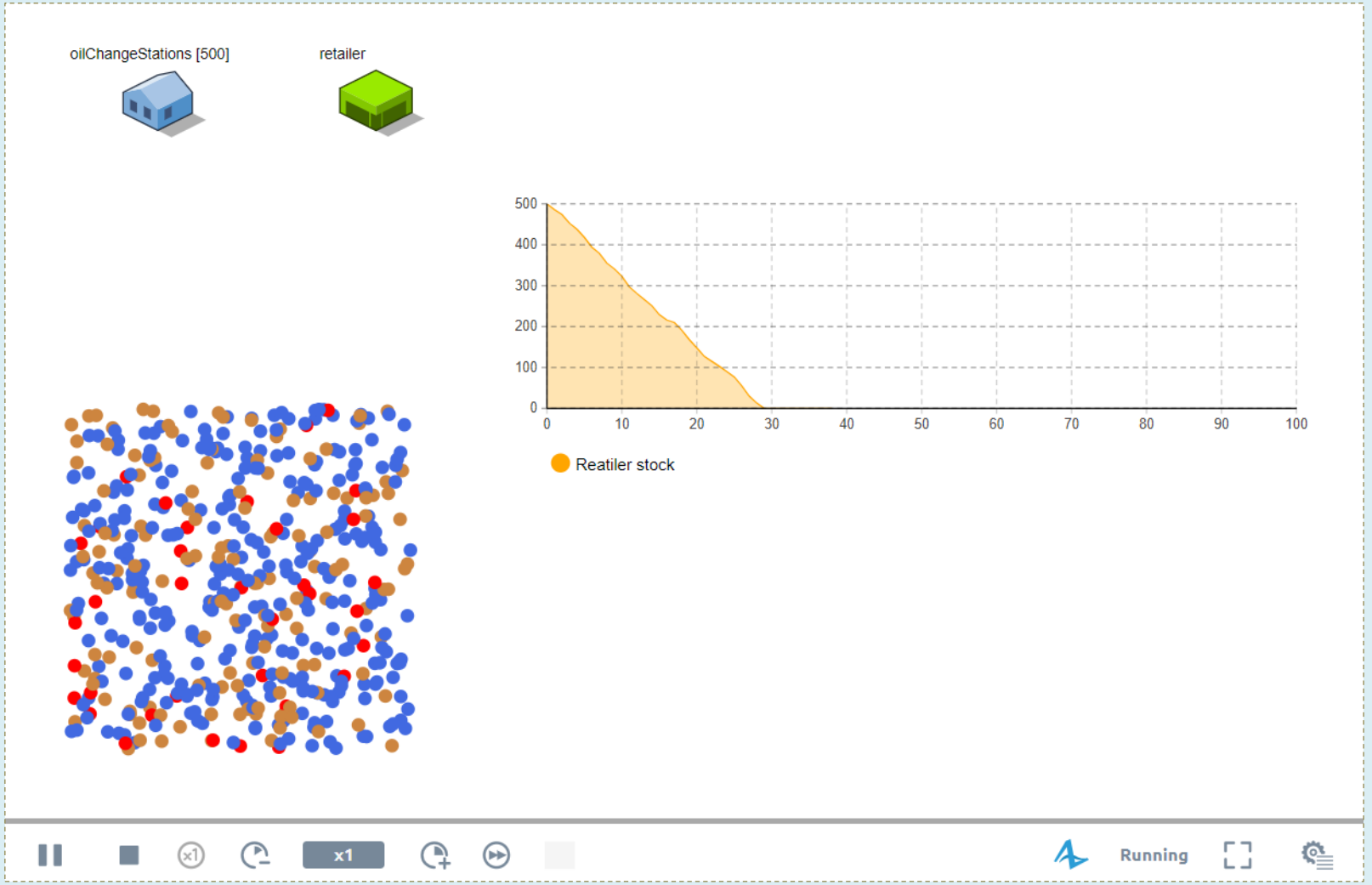
③ Set the properties of the data item as shown

Title: Retailer stock
Value: retailer.stock
Style: as shown

④ Set the plot's **Vertical scale** to Fixed with range from 0 to 500

⑤ You can also select Fill area under line in the **Style** property of the **Appearance** section

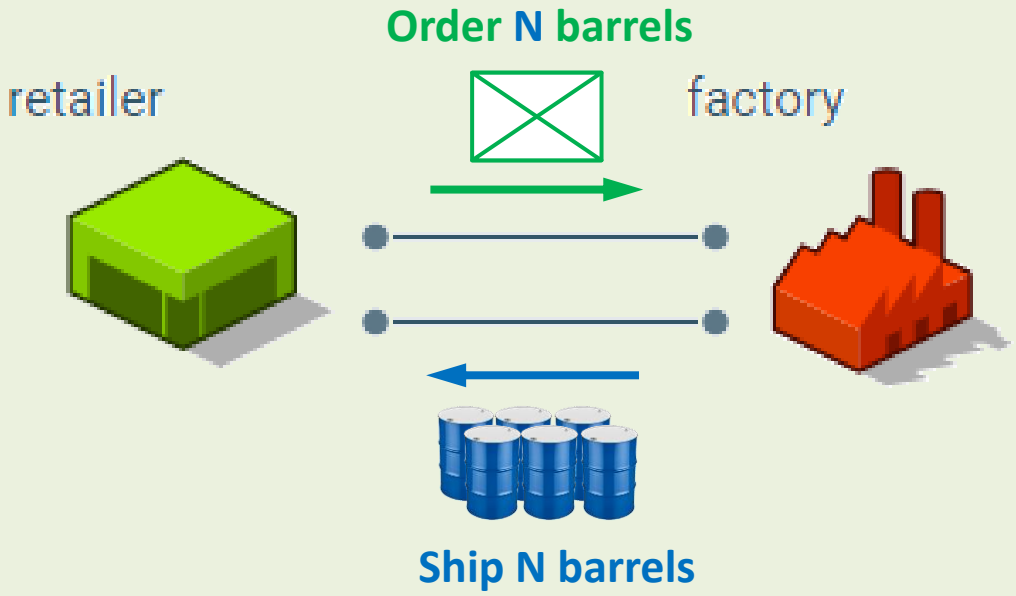
Run the model, watch the retailer's stock



Conceptualization: Interaction between the Retailer and the Factory

- Both the retailer and the factory are communicating objects with certain behaviors
- We could use the same approach to establish communication between them as we did for the stations and the retailer – use references or "addresses"
- However, as there is only one retailer and one factory AND both agents have process flowcharts inside, we can simply connect the two objects graphically using **ports** (and thus avoid writing any code)

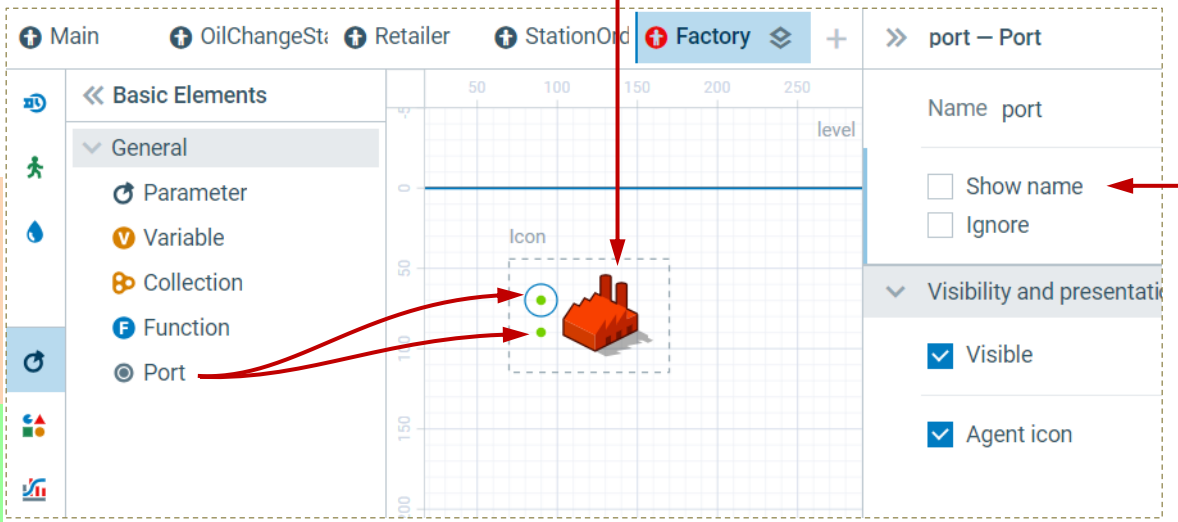
- So basically, we need entities (agents) carrying one integer number back and forth



- And we can be OK in this simple model with just one agent type RetailerOrder sent over these port connections

Create the Factory agent. Add two ports to the Factory icon

① Create a new agent type Factory and use the **Factory** 2D picture for its icon. As you did before, enlarge the picture as shown.



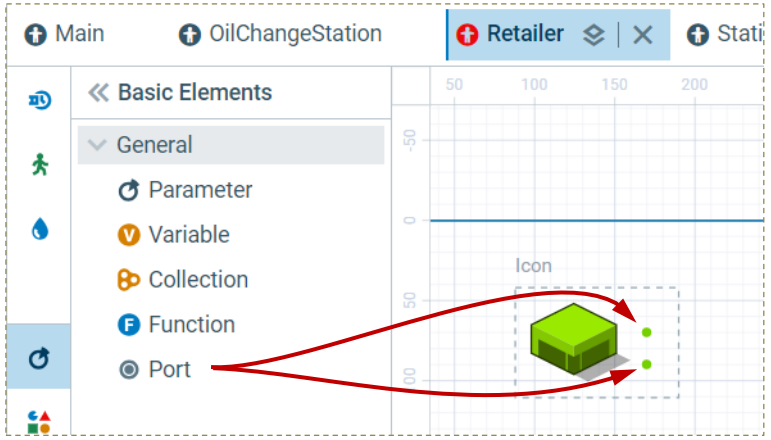
② From the **Basic Elements** palette, add two **Ports** and place them in the grid to the left of the picture as shown.

Ports are a part of agent's icon by default: they are intended to show on the upper level (owner) agent so you can connect them.

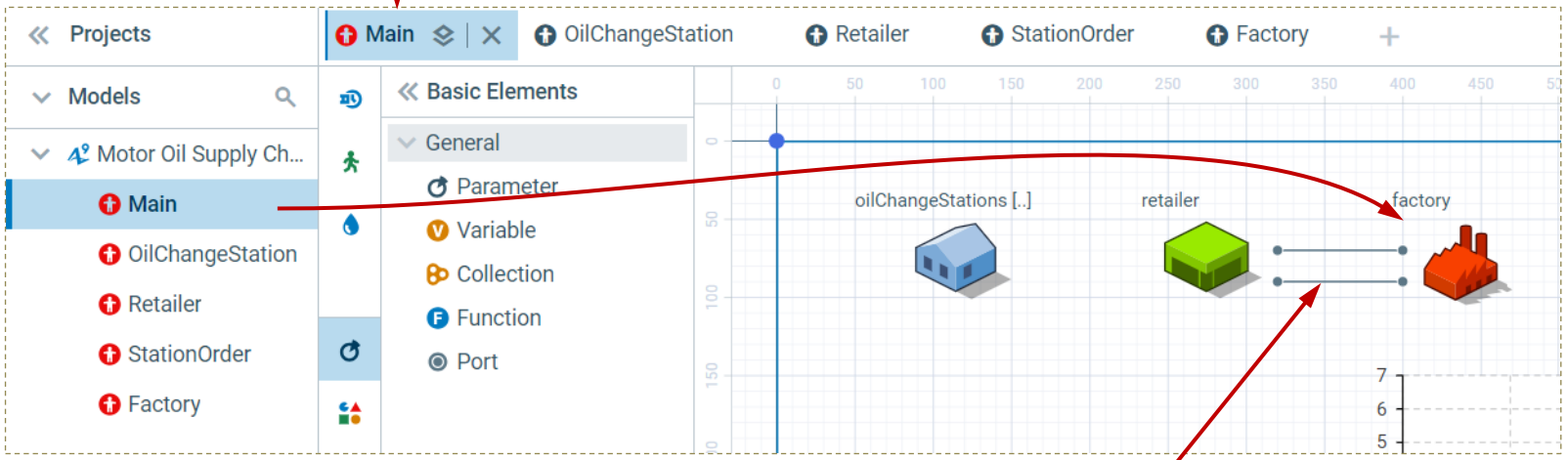
③ Deselect the Show name property of both ports – we do not need to see the names

Add two ports to the Retailer, create an instance of Factory, and connect

① Switch to **Retailer** agent type and add two ports to the right of its icon as shown. Hide port names



② Switch to **Main** agent type



③ From the **Project tree**, drag the **Factory** agent to **Main** and place it to the right of **retailer**

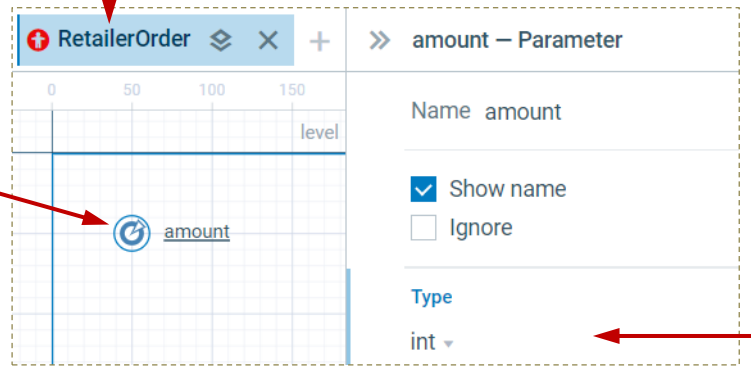
The upper ports of retailer and factory should connect automatically when aligned

④ Connect the bottom ports manually by clicking the port and then the second port. (If the top ports did not connect, connect them, too.)

Create a new agent type RetailerOrder with one parameter: amount

① Create a new agent type RetailerOrder

② From the **Basic Elements** palette, add a **Parameter**, name it amount



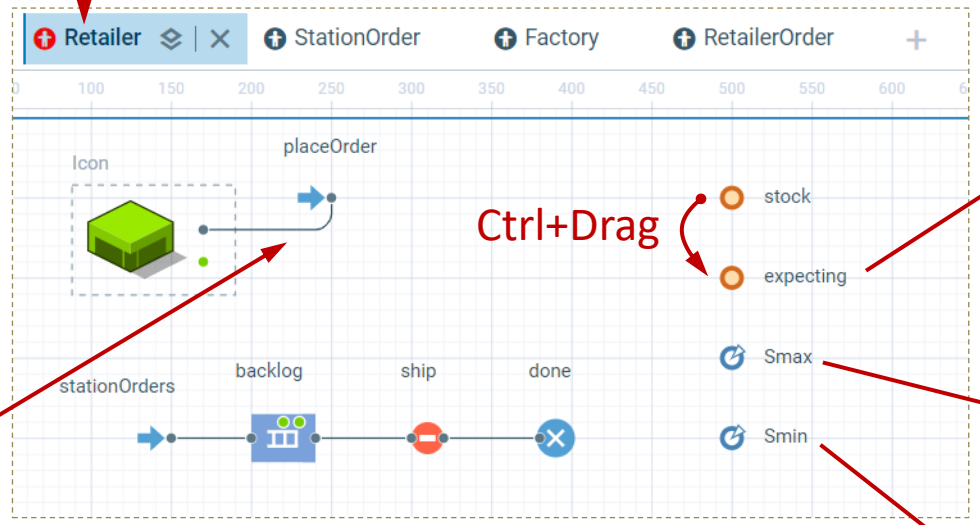
③ Set the parameter type to int

This parameter will obviously represent the number of barrels the retailer is ordering.

Note that, unlike with the StationOrder, here we do not need to identify who is ordering ("shipping address"): there is only one retailer, and it is graphically connected to the factory.

Model the (s, S) inventory policy – variables and parameters

① Switch to **Retailer** agent type



② Ctrl+Drag the stock variable to create a copy. Rename it and change the initial value as shown. (The type will be the same as of the original: int)

Name: expecting
Initial value: 0

The meaning is: this is how much I have ordered but not received yet.

③ Add two parameters, Smin and Smax, both of type int, set their initial values as shown

Name: Smax
Type: int
Default value: 500

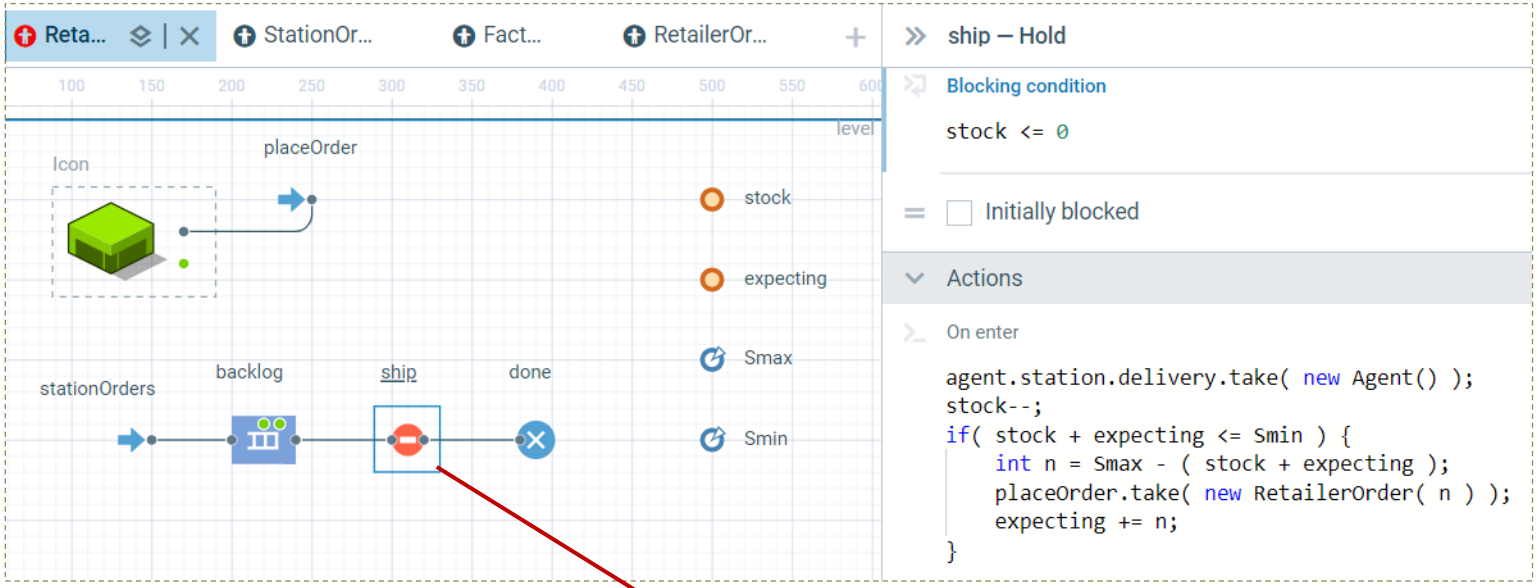
Name: Smin
Type: int
Default value: 100

These are of course the Inventory Policy parameters

④ Set the initial value of stock to Smax

⑤ From the **Process Modeling** palette, add the **Enter** block, name it placeOrder, and connect it to the upper port.

Model the (s, S) inventory policy – the policy



① Select the **ship** block and amend the **On enter** code as shown

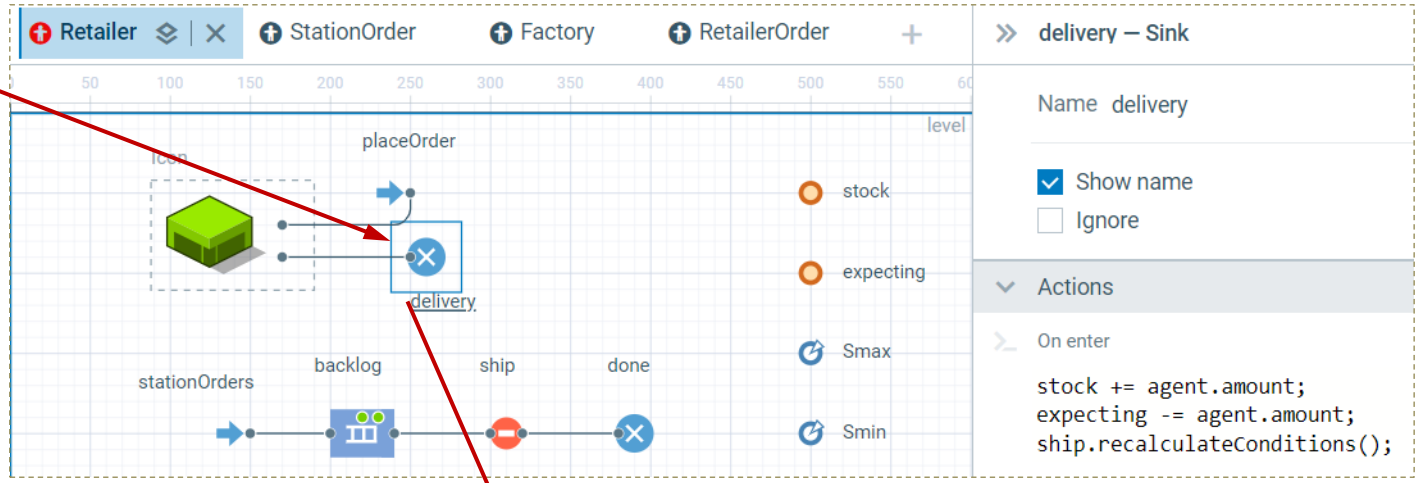
This is where the policy is actually modeled: we ship a barrel, and then, if the stock + whatever is on the way is lower or equal Smin, we order to replenish to Smax. And remember how much we ordered.

On enter:

```
agent.station.delivery.take ( new Agent() );
stock--;
if( stock + expecting <= Smin ) {
   int n = Smax - ( stock + expecting );
   placeOrder.take( new RetailerOrder( n ) );
   expecting += n;
}
```

Model the (s, S) inventory policy – delivery

① From the **Process Modeling** palette, add the **Sink** block and connect it to the lower port. Name it delivery.



② Set the **Agent type** property and add and set **On enter** action as shown

Note that we will be using the same agent type for order and shipment!

Agent type: RetailerOrder

On enter:

```
stock += agent.amount;
expecting -= agent.amount;
ship.recalculateConditions();
```

Add what was delivered

Decrease what is on the way

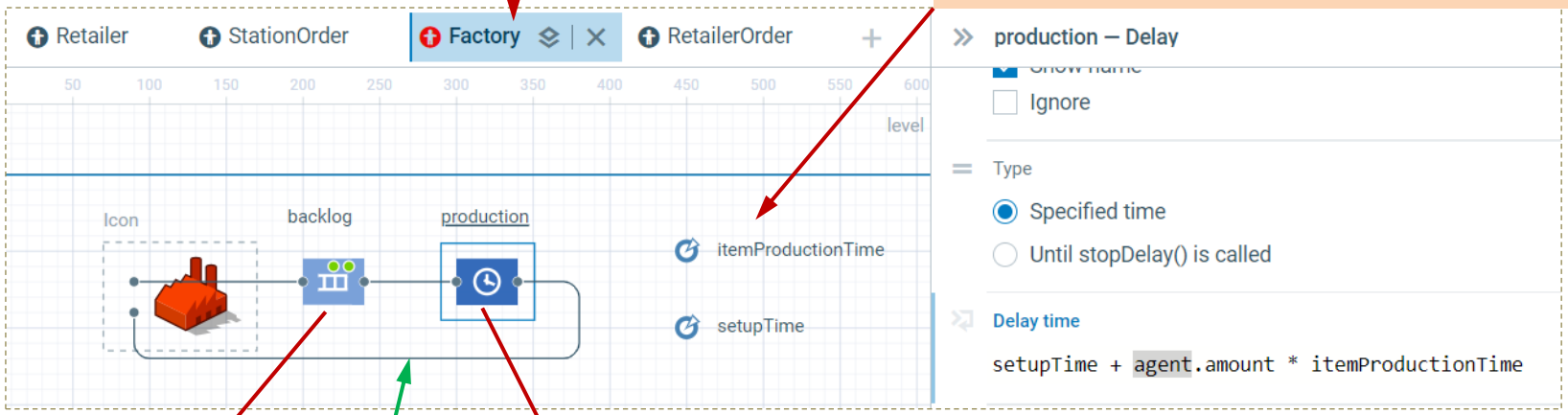
How would the ship block know that the stock variable has been updated? This is how we tell it to recalculate the condition and possible ship more orders to stations

And before we run the model – add the Factory behavior

① Switch to **Factory** agent type

② Add two parameters (leave type double): setupTime with **Default value** 2, and itemProductionTime with **Default value** 0.04

③ From the **Process Modeling** palette, add two blocks: **Queue** and **Delay**, connect the to each other and to the ports, and set their properties as shown



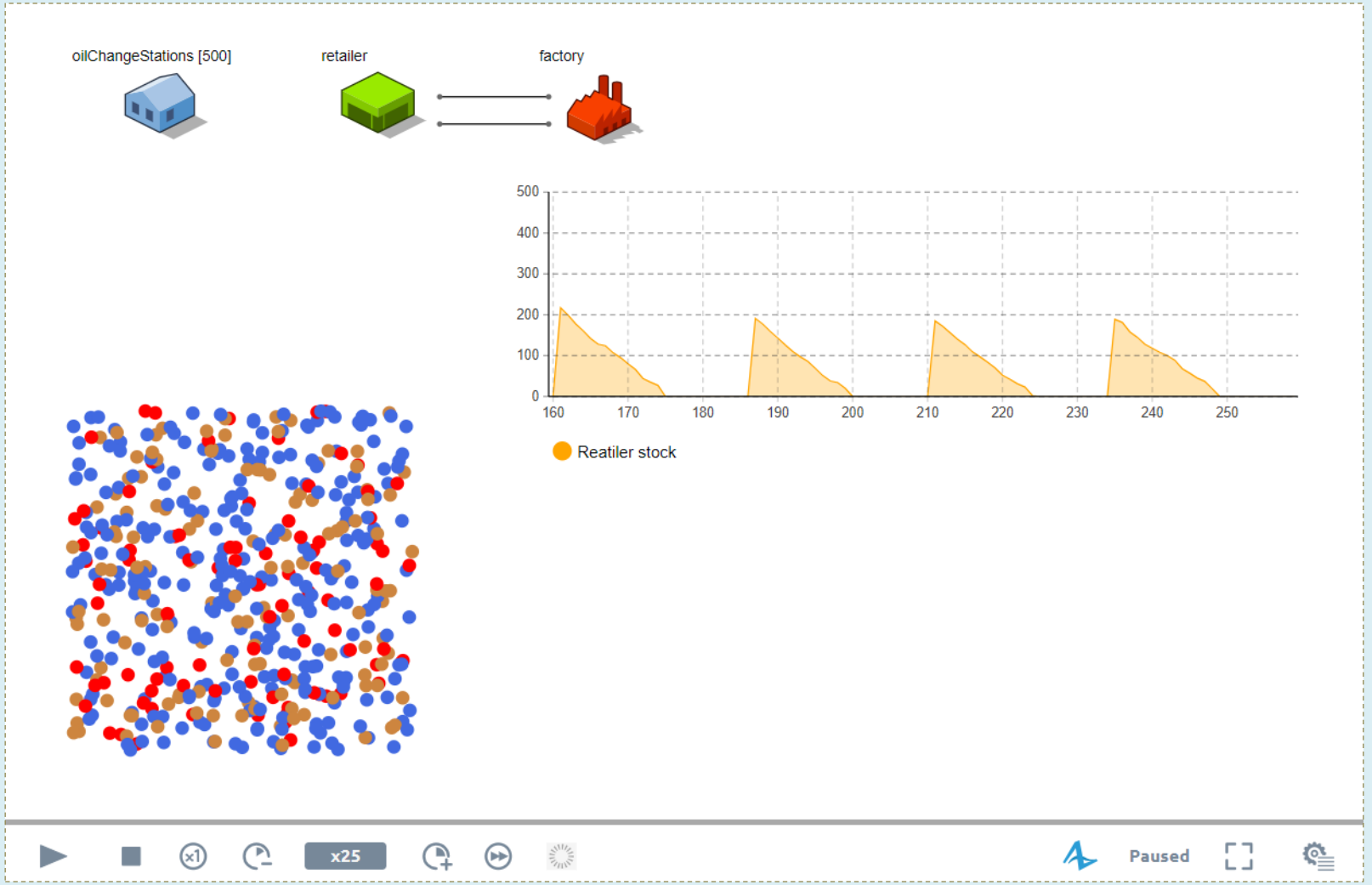
Name: backlog
☑ Maximum capacity
Agent type: RetailerOrder

Name: production
Delay time: setupTime + agent.amount * itemProductionTime

Once again, notice that we are using the **same agent type for order and shipment**!

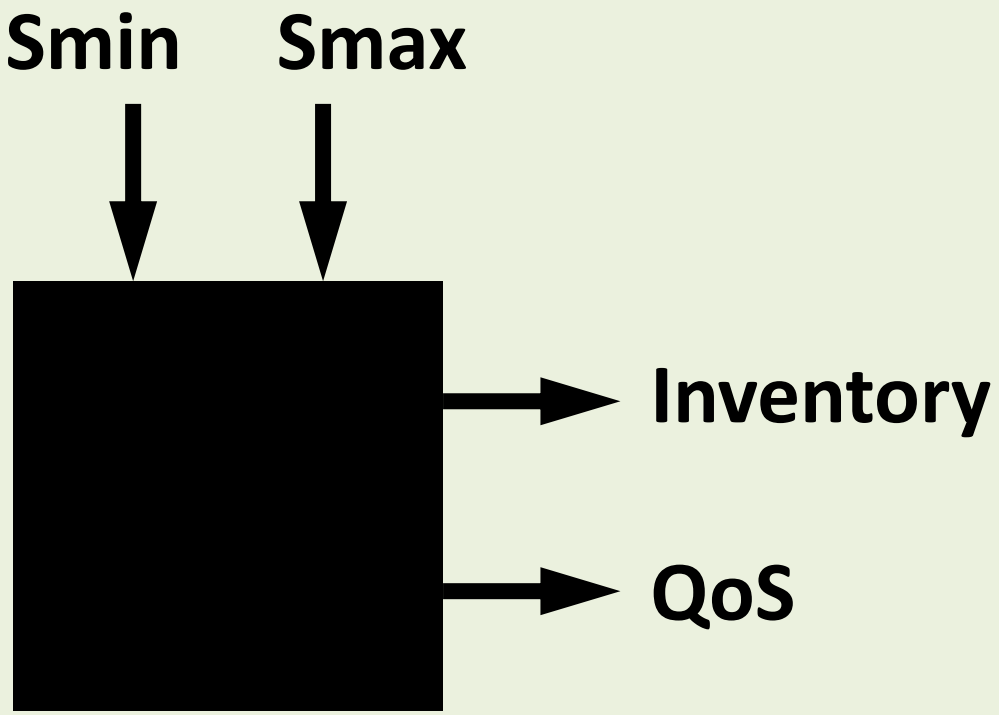
The delay time value is straightforward. But how do we ensure orders are processed one by one and not in parallel? This is **controlled by the Delay block Capacity parameter**, which is 1 by default.

Run the model, watch the retailer's stock pattern



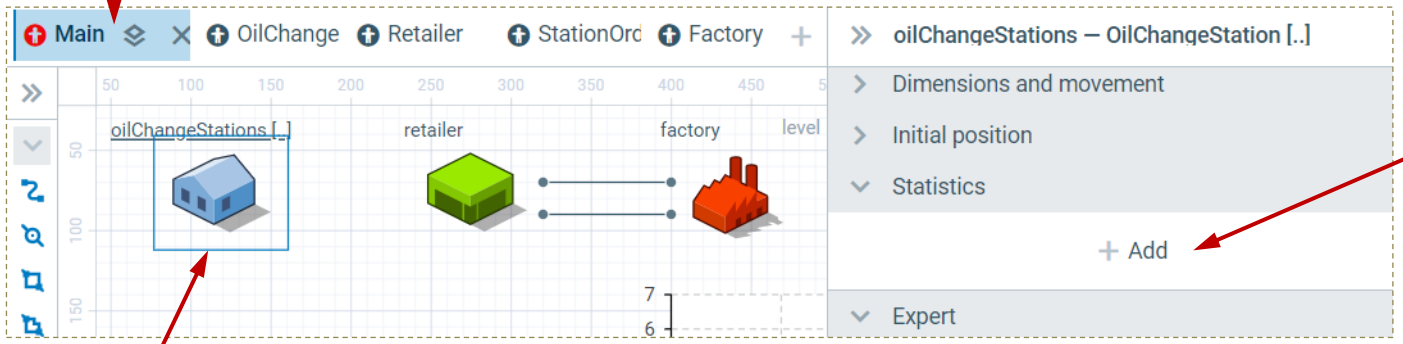
Congratulations! We are done with the model logic! What next?

- Looks like the model now fully reflects the system (the supply chain), subject to the specification given to us by the client and our own assumptions.
- However, we are not ready to solve the problem yet: the model needs metrics:
  - **Inventory**: average retailer's stock over time, and
  - **QoS**: the max percentage of oil change stations out of oil at any given time
- Also, we need to have control over the (s,S) parameters from outside of the model



Create the “number of stations out of oil” (a population statistics)

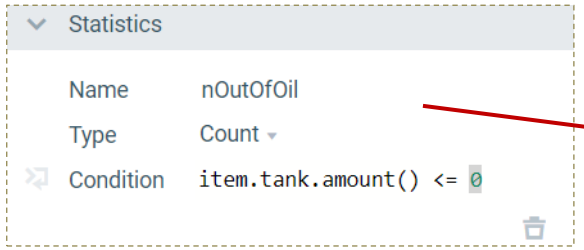
① Switch to **Main** agent type



② Select the **oilChangeStations** population

③ In the **Statistics** section of the properties, click **Add**

This will add a statistics across the entire population



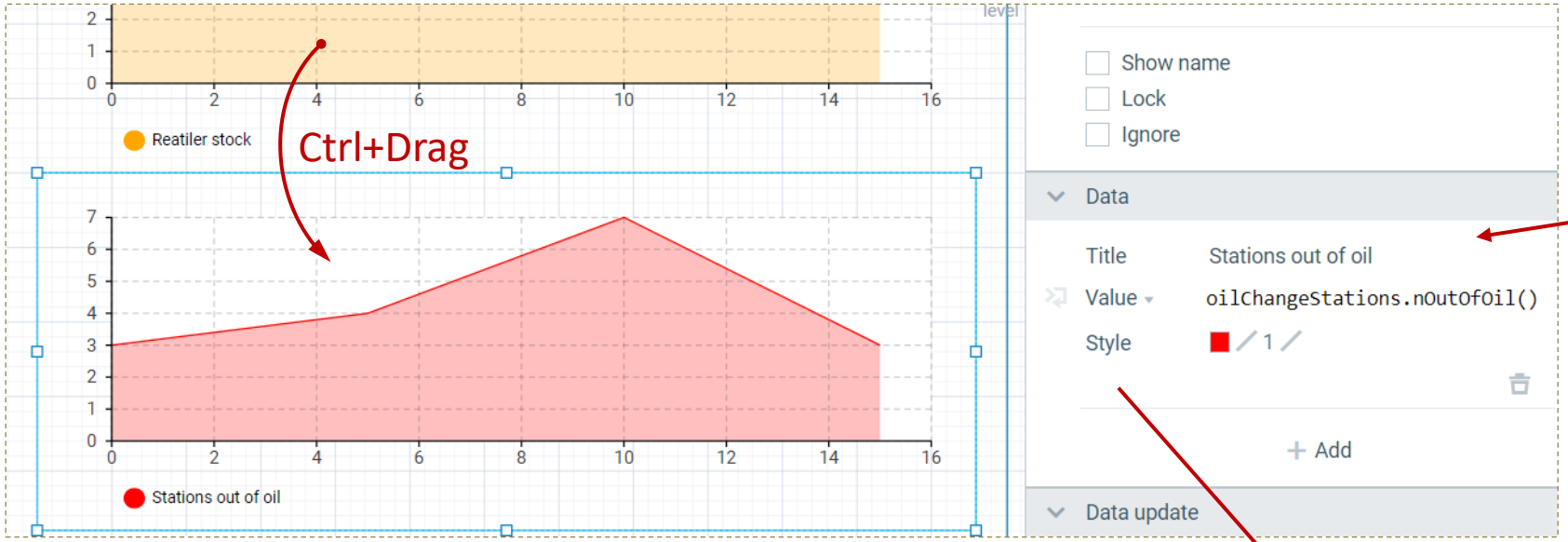
④ Setup the **Statistics** item as shown

Name: nOutOfOil
Type: count
Condition: item.tank.amount() <= 0

Here, “item” refers to the population item, i.e., an oil change station; the rest is straightforward: we are counting the number of stations where amount of oil is <= 0. Adding such statistics means adding a function **nOutOfOil()** that returns the corresponding number. (no statistics will be collected unless you call that function somewhere)

Display the number of stations out of oil over time

① Staying in the **Main** agent, Ctrl+Drag the **Retailer stock** chart to create a copy.



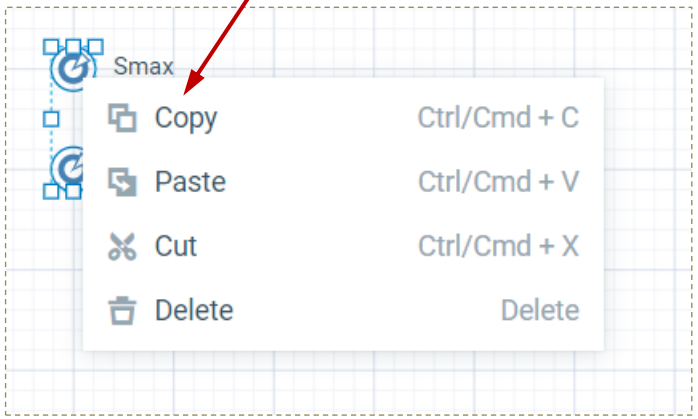
② In the new chart, change the properties of the data item as shown

Title: Stations out of oil
Value: oilChangeStations.nOutOfOil()
Style: Color: red

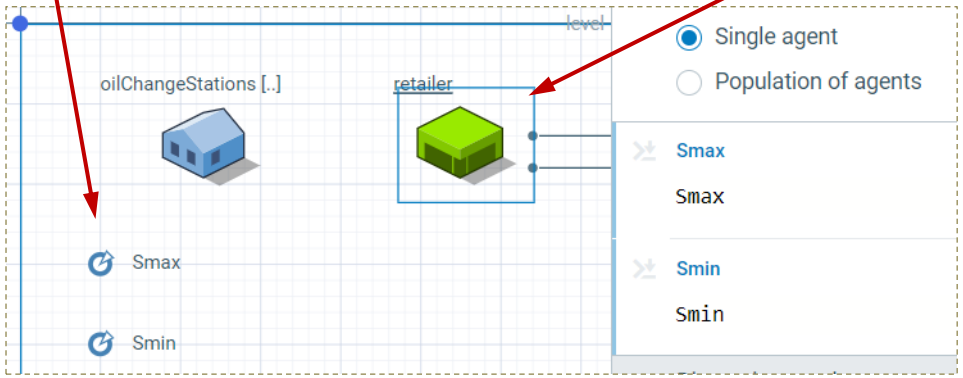
Here, the plot will call the statistics function nOutOfOil() and update itself

And one more thing: we want to have control over the (s,S) parameters

① Switch to **Retailer** agent type. Select **Smin** and **Smax** parameters and copy them using the context menu



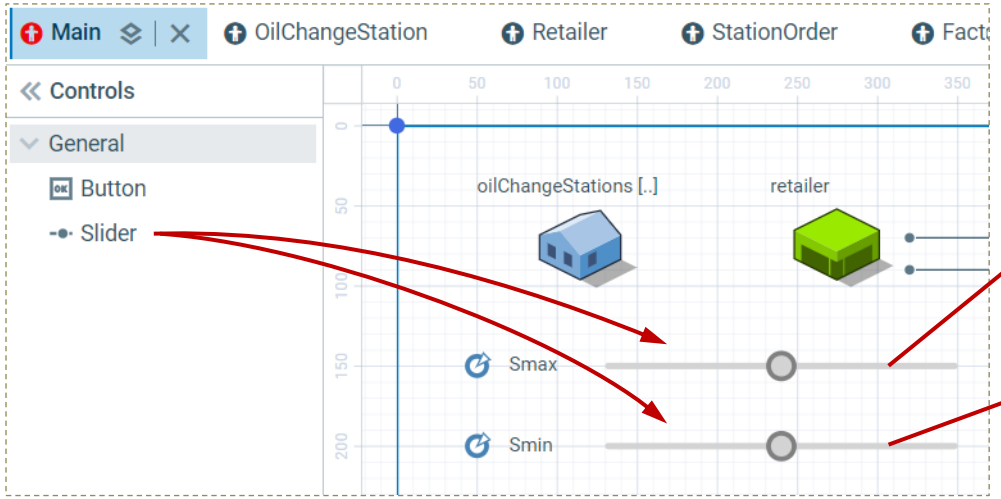
② Switch to **Main** agent type. Paste the parameters as shown



③ Select the **retailer** agent and set its parameters to **Smax** and **Smin** correspondingly as shown

This way we propagated the parameters up to the level of Main, i.e., to the top level of the model

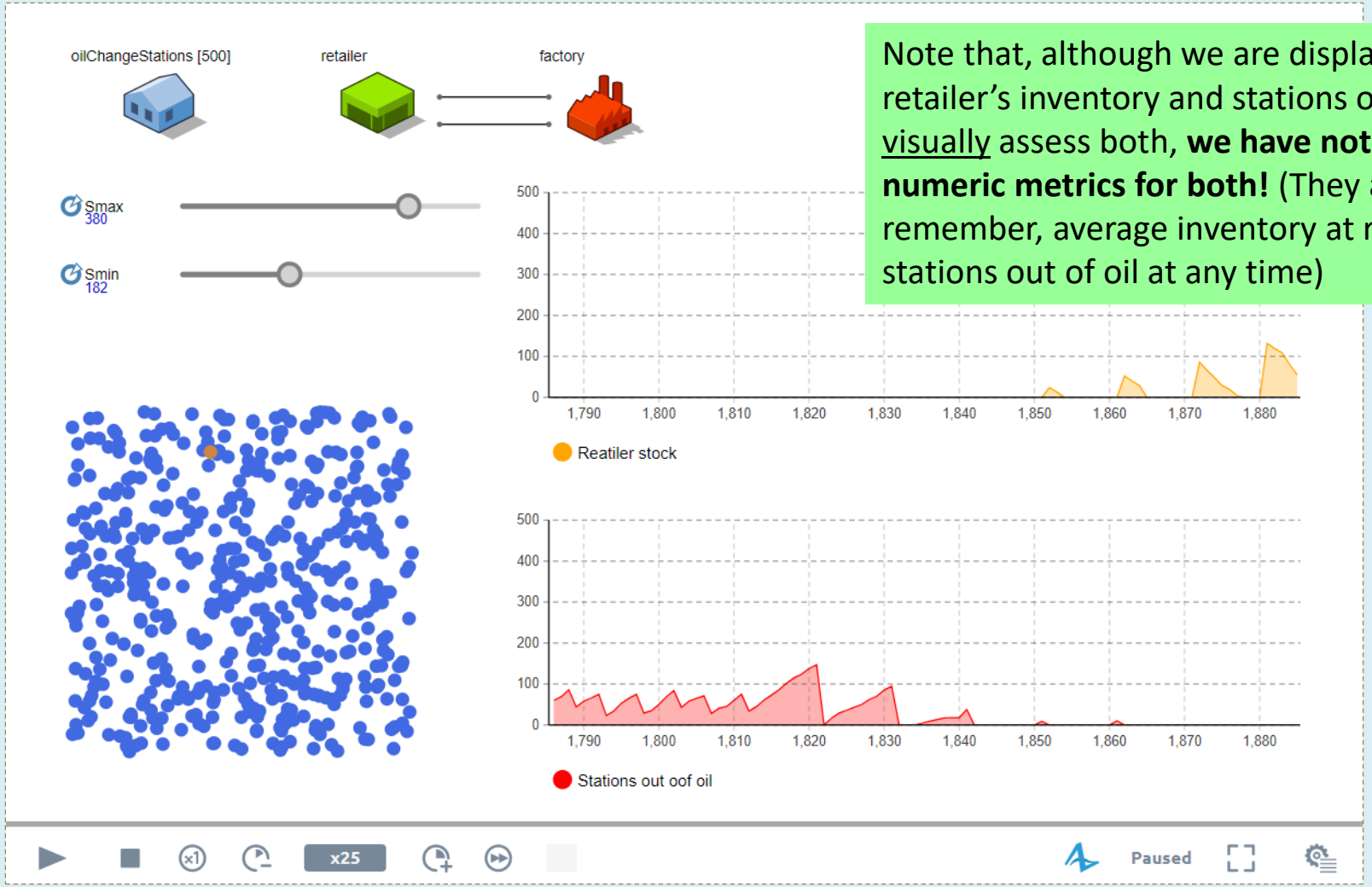
④ From the **Controls** palette, add two **Sliders** and link one of them to Smax, the other one to Smin, and set the range of both sliders to 0..500



☑ Link to: Smax
Maximum value: 500

☑ Link to: Smin
Maximum value: 500

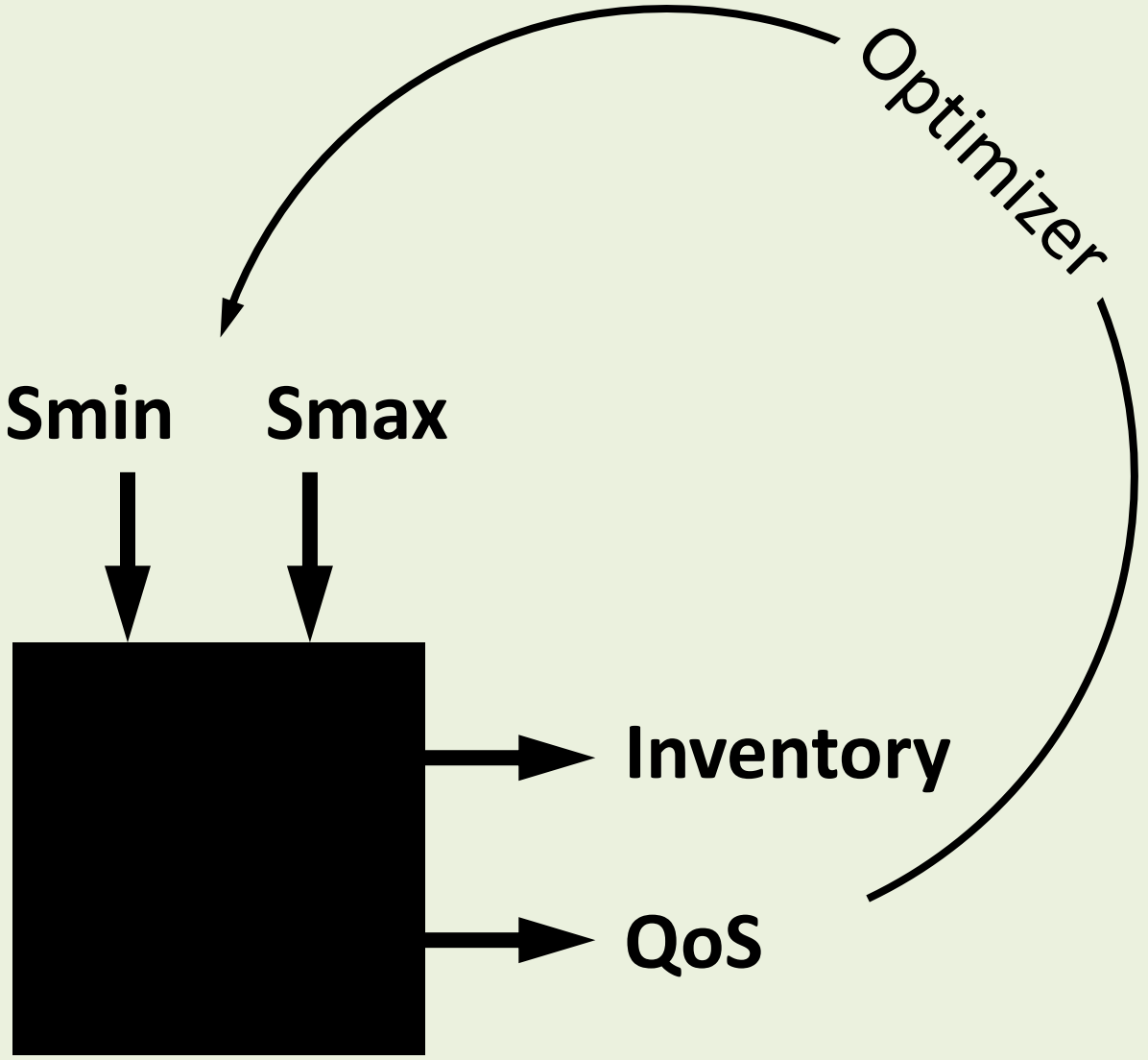
Run the model. Play with Smin and Smax parameters – try to optimize!



Note that, although we are displaying the plots with retailer's inventory and stations out of oil and can visually assess both, **we have not yet created formal numeric metrics for both!** (They are, as you may remember, average inventory at retailer and max % of stations out of oil at any time)

Manual optimization is tricky. But we can automate it!

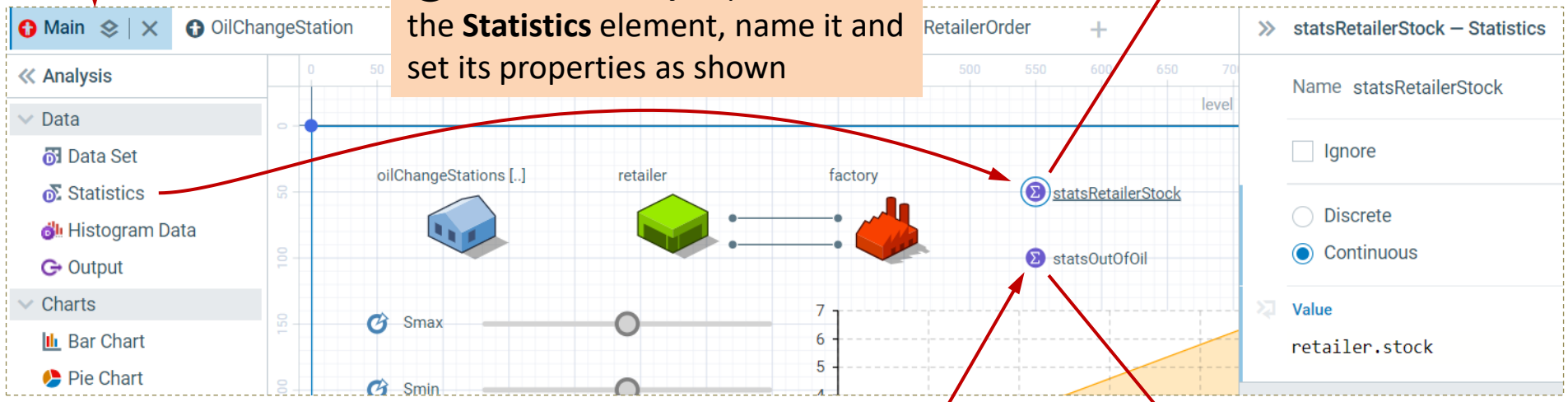
- We will utilize AnyLogic Cloud Optimization experiment to find the best values of Smin and Smax
- We will put the model in the loop with the optimizer
- The optimizer will execute some search strategy in the parameter space and will choose parameters for the next set of runs based on the objectives of the previous runs
- To prepare the model for optimization, we need to finish creating formal metrics!



Create formal numeric metrics. Add statistics

① Switch to **Main** agent type

② From the **Analysis** palette, add the **Statistics** element, name it and set its properties as shown



Name: statsRetailerStock
◉ Continuous
Value: retailer.stock

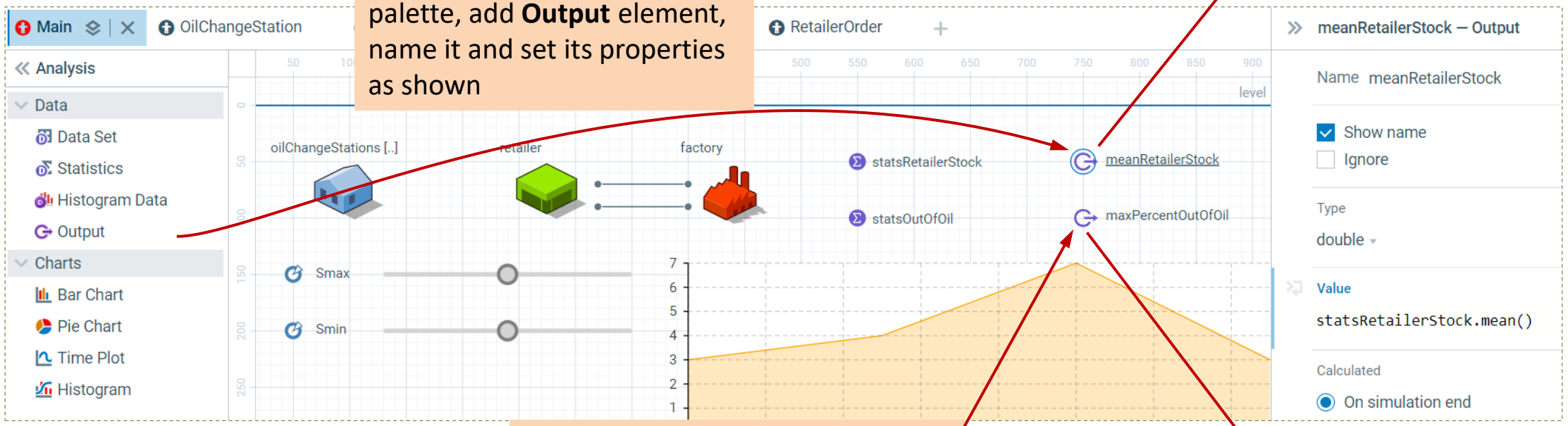
③ Ctrl+Drag the statistics to create a copy. Rename it and set the properties as shown

Name: statsOutOfOil
◉ Continuous
Value: oilChangeStations.nOutOfOil()

The statistics elements will automatically collect samples of data during the simulation run and provide the aggregated info such as mean, minimum, or maximum.

Last step: add outputs

① From the same **Analysis** palette, add **Output** element, name it and set its properties as shown



Name: meanRetailerStock
Value: statsRetailerStock.mean()

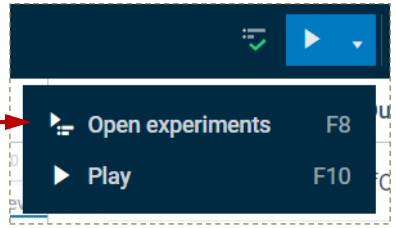
③ Ctrl+Drag the output to create a copy. Rename it and set the properties as shown

Name: maxPercentOutOfOil
Value: statsOutOfOil.max() / 500

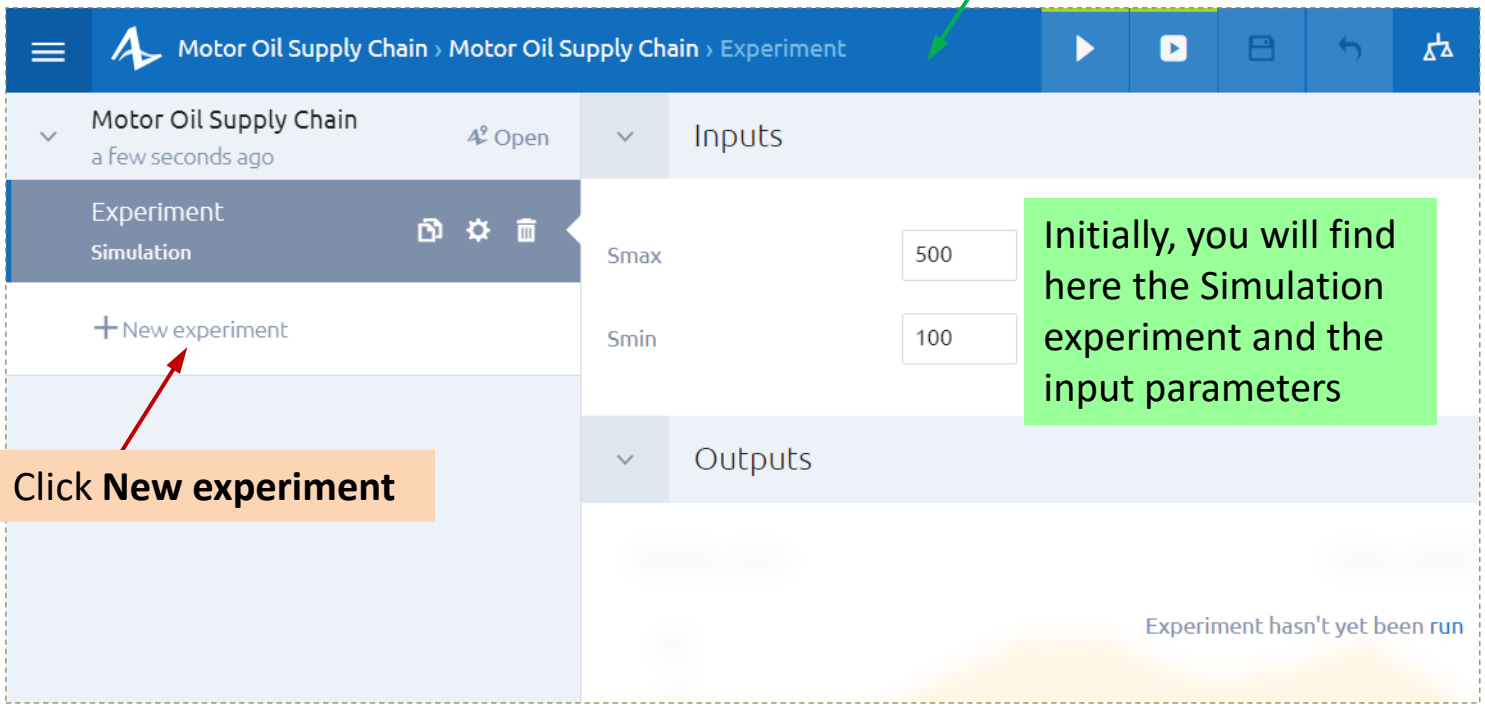
Both output values are calculated at the end of a simulation run (see the properties of the Output element). Therefore, they will get the latest available statistics values covering the whole simulated period. And we will use these metrics in the optimization experiment.

We have all the parameters and metrics! Let's set up optimization

① Open the dropdown list near the **Run** button and choose **Open experiments**



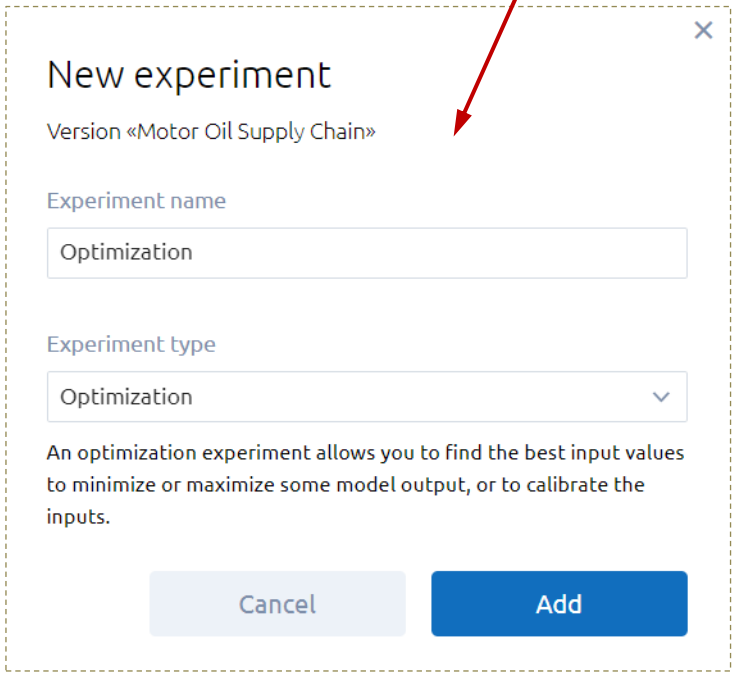
As you do it, the model gets compiled, and you are taken to the AnyLogic Cloud execution environment



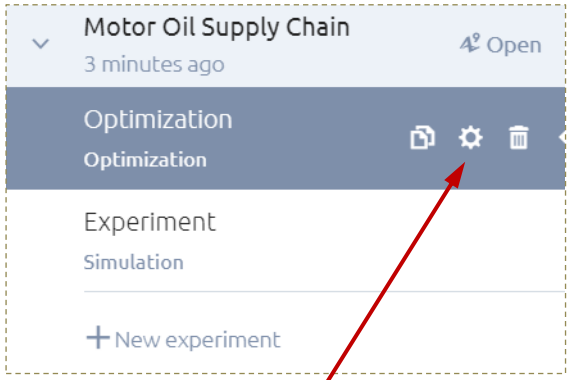
Initially, you will find here the Simulation experiment and the input parameters

② Click **New experiment**

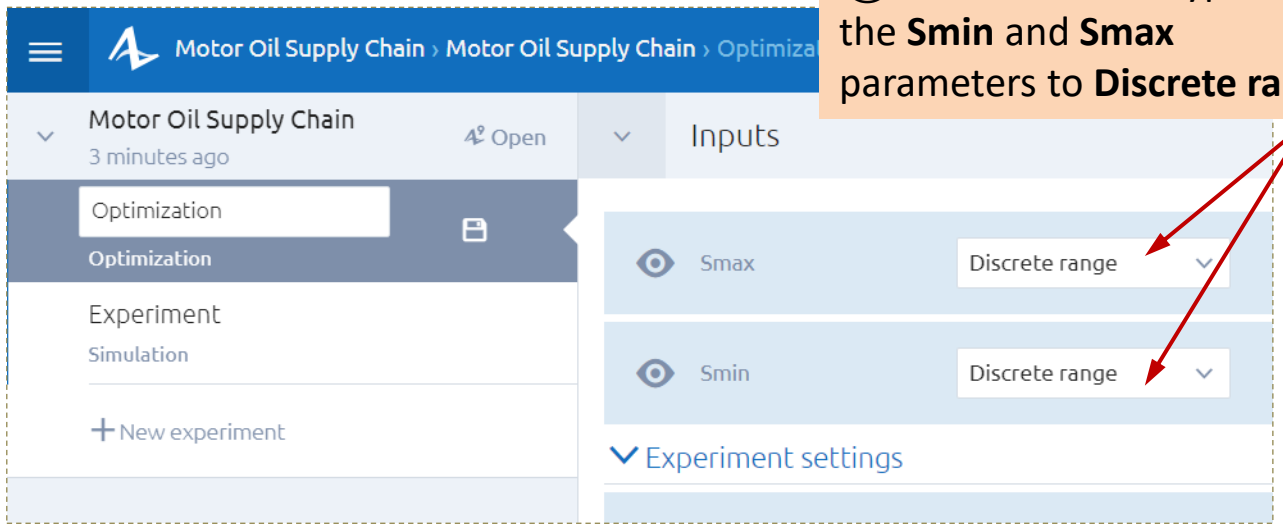
③ Enter the experiment name, e.g., Optimization and choose **Optimization Experiment type**, and click **Add**



Set up optimization inputs

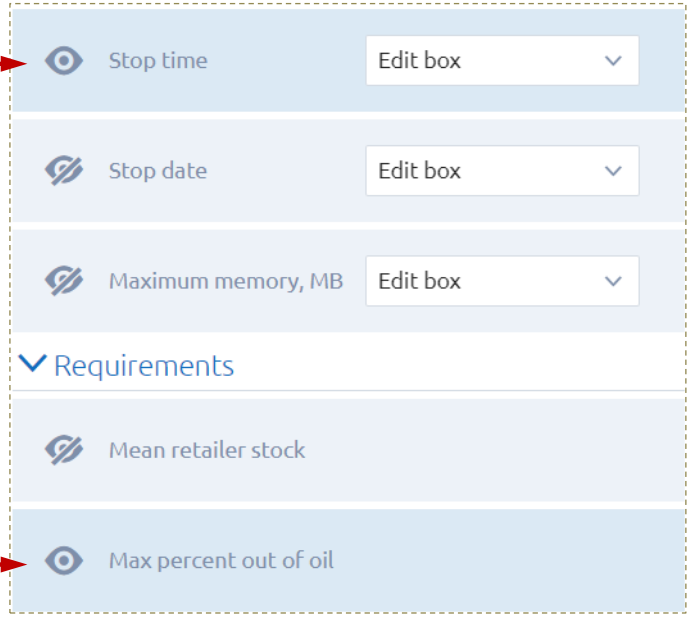


① Click the gear icon of the newly created Optimization experiment to open its dashboard editor



② Set the control type of the **Smin** and **Smax** parameters to **Discrete range**

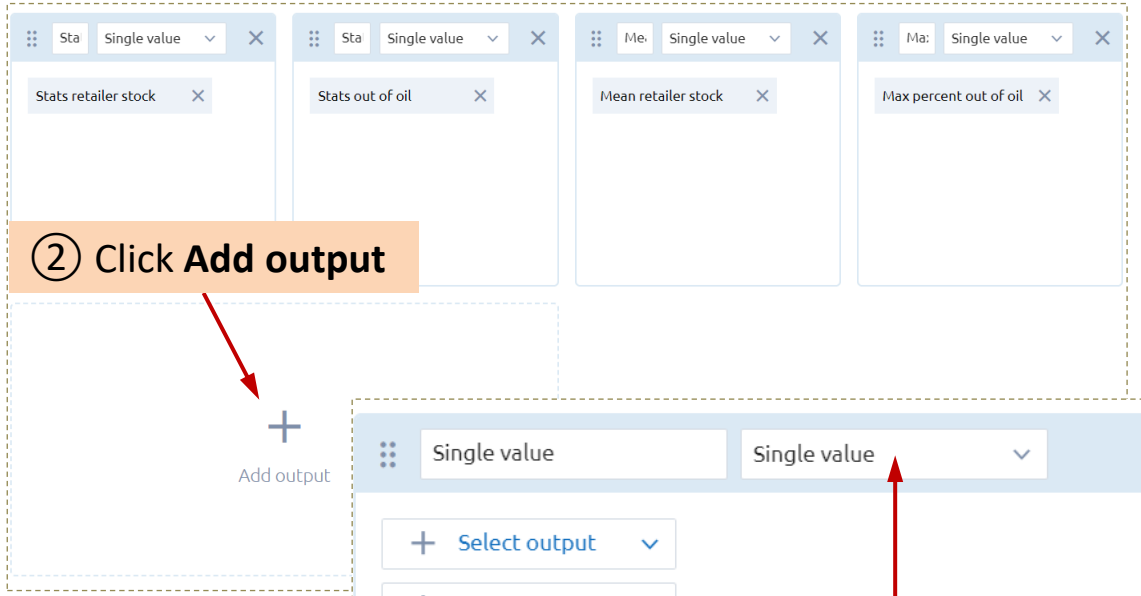
③ Show the **Stop time** – always helpful



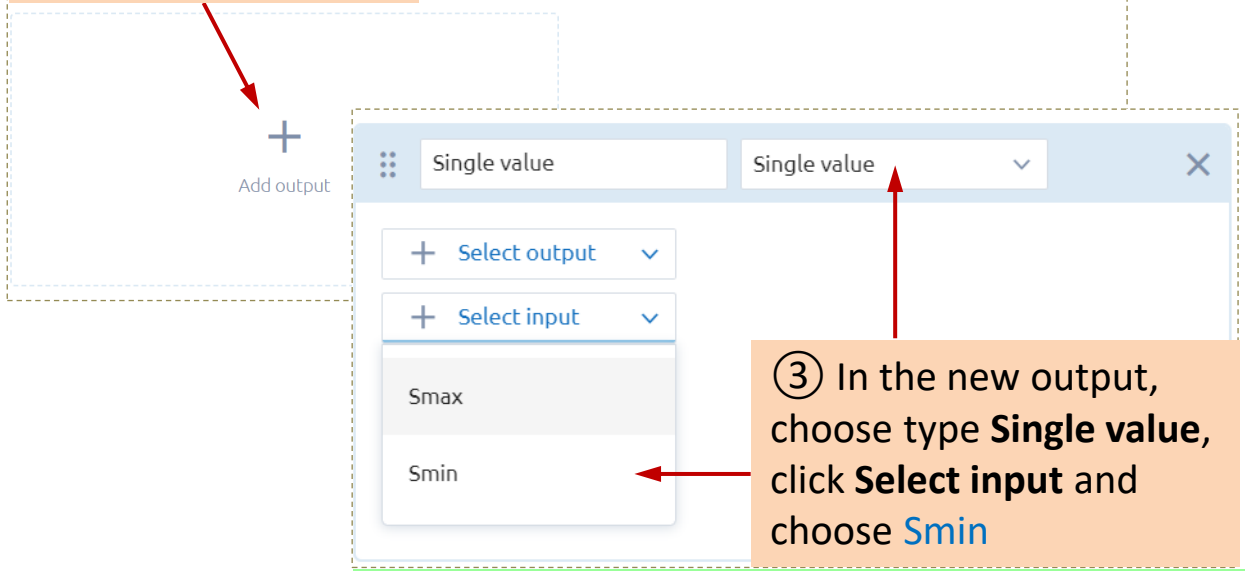
④ Show the **Requirement** for **Max percent out of oil** [explained later]

Set up optimization outputs

① Scroll down the dashboard

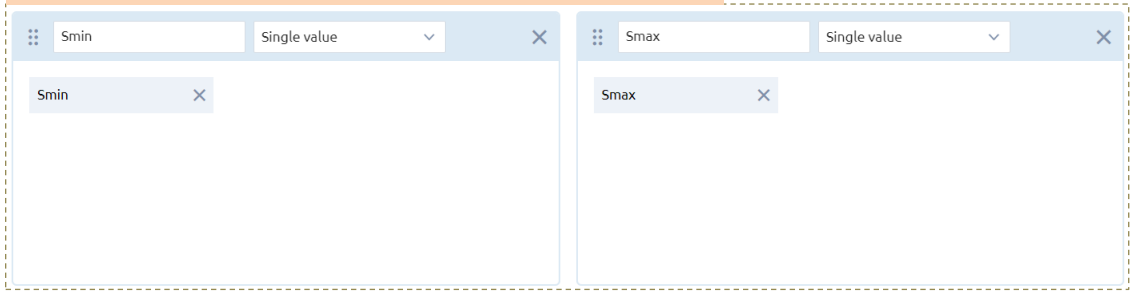


② Click **Add output**



③ In the new output, choose type **Single value**, click **Select input** and choose Smin

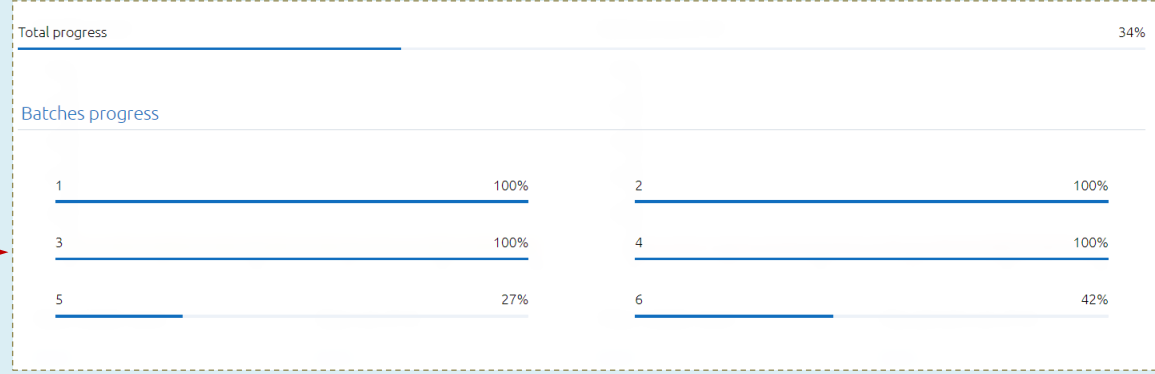
④ Repeat steps 2 and 3 for Smax. You should now have these two items:



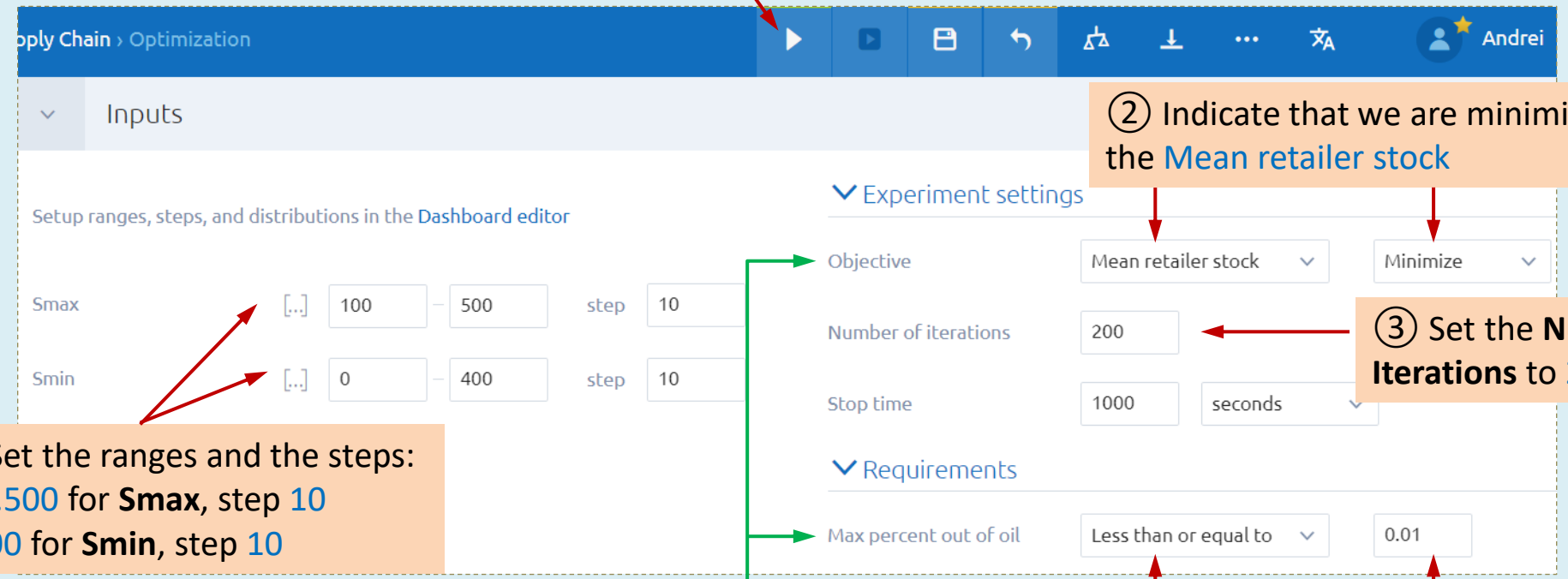
⑤ Save the dashboard

It might seem strange to have **inputs in the Outputs section**, but in case of optimization experiment, optimal inputs are exactly the result of the experiment run

Run optimization



⑤ Run **Optimization** and watch the progress



② Indicate that we are minimizing the Mean retailer stock

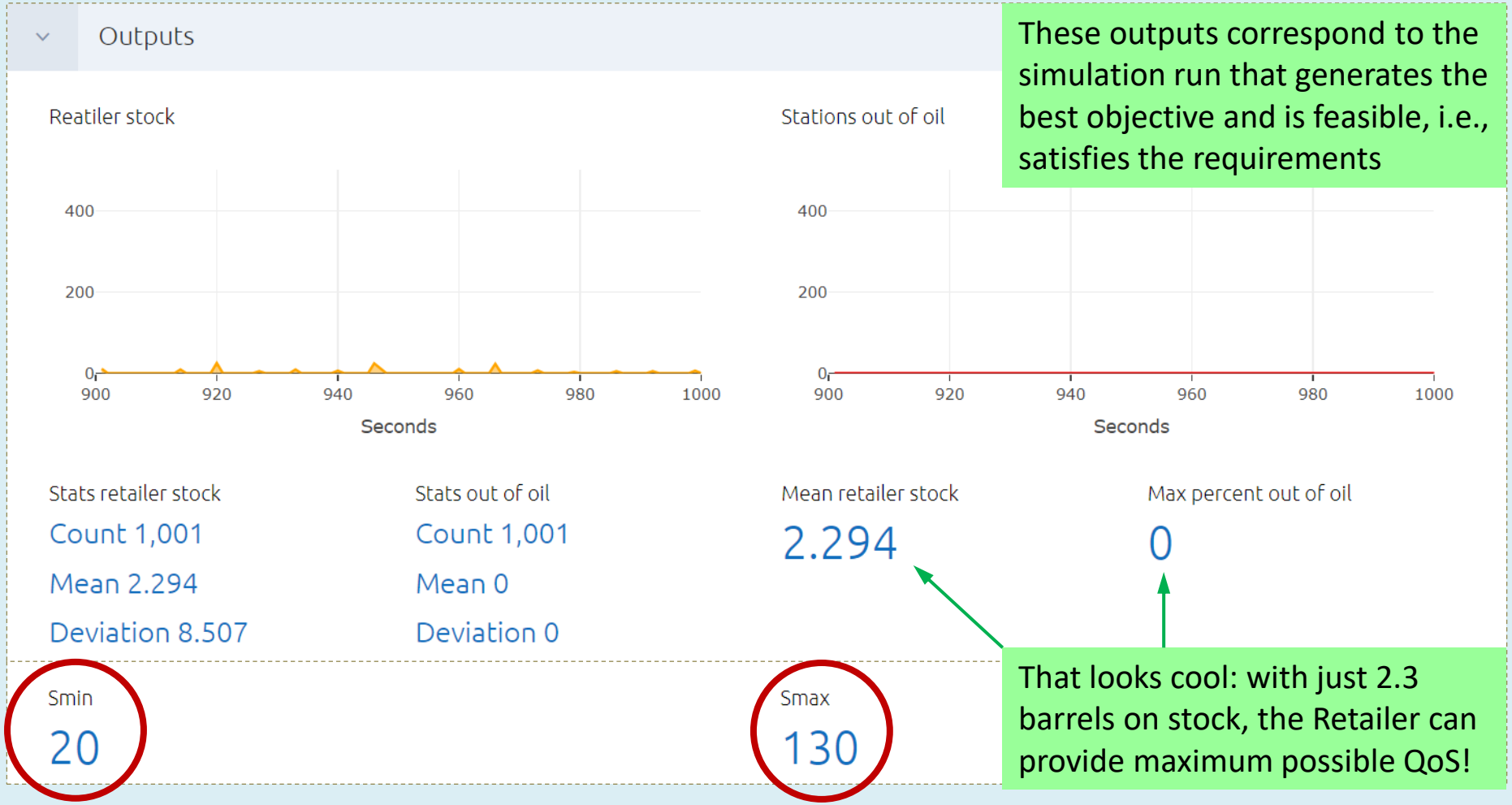
③ Set the **Number of Iterations** to 200

① Set the ranges and the steps:
100..500 for **Smax**, step 10
0..400 for **Smin**, step 10

④ Set the requirement that **Max percent out of oil** is <= 0.01

Having “Retailer stock” as the objective and “% out of oil” as a requirement is natural for our problem: being the retailer, we want to have the stock just as low as possible while for the QoS we have a hard upper bound of 1%.
If we wanted to minimize the “% out of oil” as well, we would have created a **weighted sum objective function**.

As the optimization finishes, you will see the results [yours may be different!]



These outputs correspond to the simulation run that generates the best objective and is feasible, i.e., satisfies the requirements

That looks cool: with just 2.3 barrels on stock, the Retailer can provide maximum possible QoS!

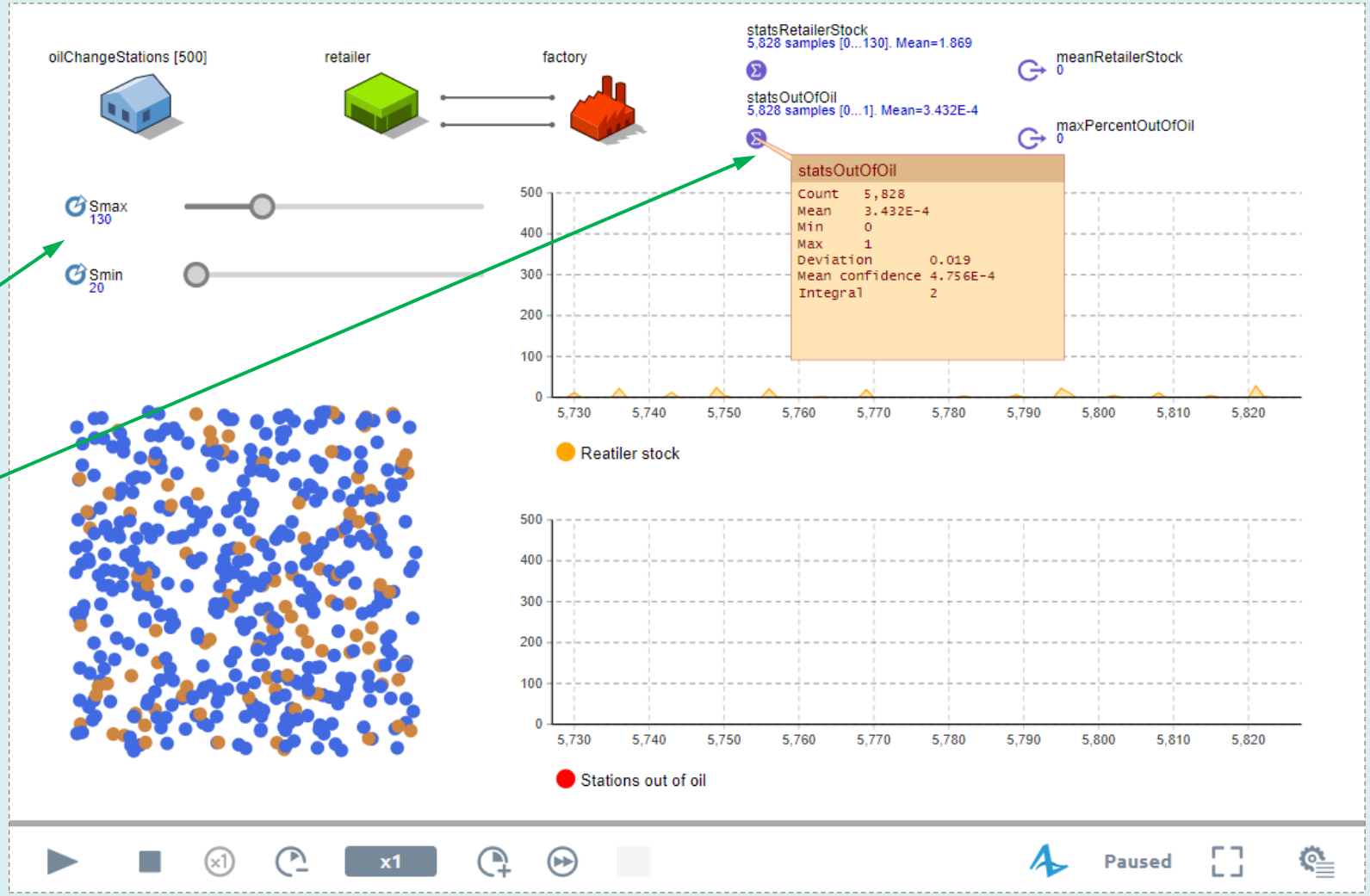
And here is the answer for the client!

Verify the answer. Run simulation with the parameters found

① While in the (completed) Optimization experiment, press **Play** button

An animated run is played with the parameters corresponding to the best feasible objective

Display the Inspect of statsOutOfOil. As you simulate further, you may see some close to zero values



***

That was quite a project, given the timeframe, and you did great!

We hope you enjoyed the AnyLogic 9 UX!

And by the way – the model stays in the Cloud: always ready for more experiments with new inputs and for more edits!

Thank you!

AnyLogic Team