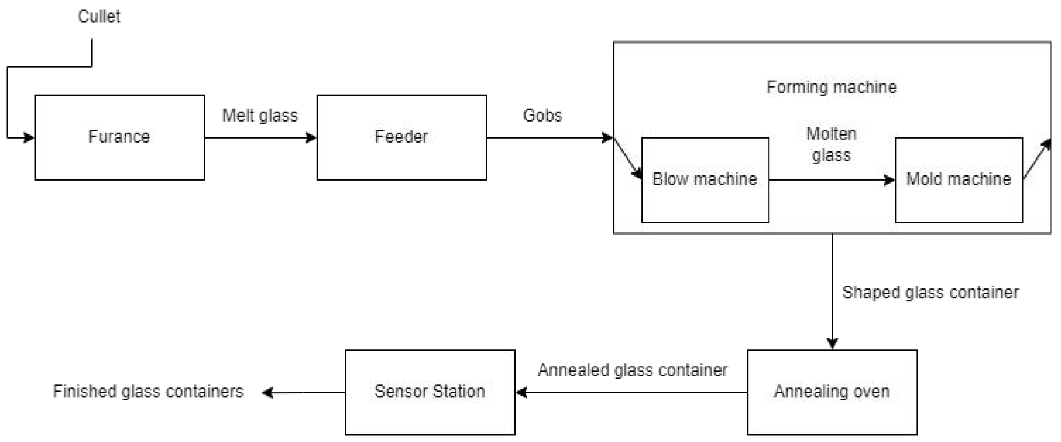
Pipeline and Filter Style

1. Example of application in real life

The factory glass-making line is a good application of the pipeline-filter architecture, and we can think of the broken glass (cullet) from the recycling bin as the "source code" at the beginning. Then, each stage (machine) that processes it can be considered as a different "filter". In the beginning, the first filter, the furnace, melts the cullet and then adds silica sand and colorant to make the glass more transparent. Then, the melted glass is fed into a second filter, the feeder. At the bottom of the feeder, the melted glass is cut into gobs of the same size, and each gob is sent to a different third filter, the forming machine, through the delver machine. Here we will take the simplest two types of forming machines as an example. The first is the blowing machine, which uses compressed air to make the gob form a cavity to initially form the shape of the container. The second is a molding machine, where the molten glass from the blowing machine is placed in a mold and blown and extruded to form the final shape. To make the glass containers stronger, the containers are sent to a fourth filter, the annealing oven, where they are reheated and cooled to produce an annealing layer. finally, the final filter checks the regularity of the containers, and the qualified glass containers (“executable code") is produced.



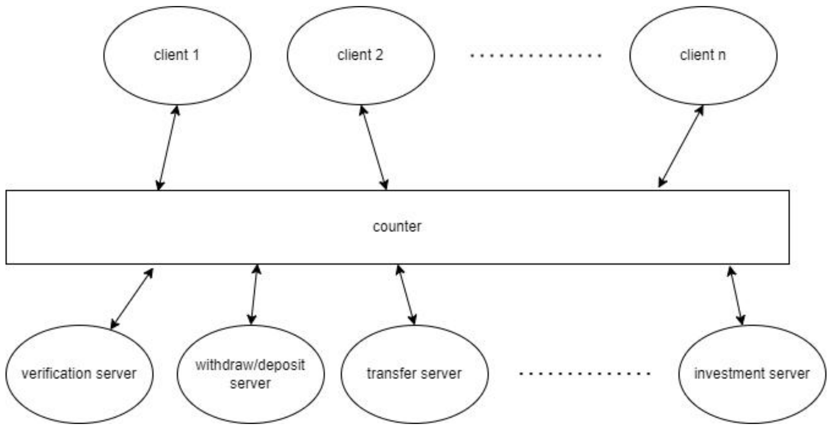
2. Beneficial to the system

Compared to one machine that is responsible for all production steps, the pipeline and filter structure allows the entire glass container production line to be seen at a glance. When there is an abnormality in the production, the results of each filter can be checked, and the problem can be solved. This structure reduces the coupling, i.e., each stage of the machine is independent of the other (the filters do not affect each other). The production line becomes more flexible and easier to maintain. If there is a special need, only the machines can be changed (adding, reducing, or modifying filters in the line) to control the production. Also, the line can be better interfaced with other plant modules, such as packaging lines.

Client and Server Style

Example of application in real life

An example of the client and server is the banking system (in person, not necessarily online banking), including the services of opening/closing accounts, money transfer, withdrawing/depositing cash, cheque ordering, and many others. The client components in the banking system will be the different customers. The server part of the system is the services that the branches provide. The clients communicate with the servers through the bank counters in the branches (network). For instance, if a client Tom wants to withdraw $200, he will go to a branch counter and produce his card to the verification server (hand in card to counter and type password). The verification server will check if this card/password is valid and belongs to Tom and send back the result. Tom will then send a request to the withdrawal server (tell the counter that he wants to withdraw $200). The withdraw server will check that he has $200 in his account, deduct the amount, and produce $200 in cash, which is sent back to Tom via the counter.



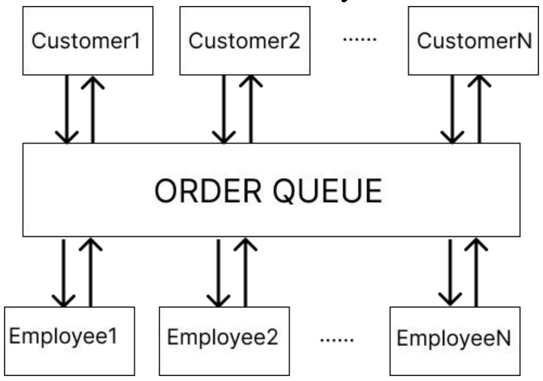
There are several advantages of using this style. Firstly, the distribution of data is straightforward – different counters in different branches could perform the same task, and banking information could be on multiple servers.

Secondly, the clients and server could be on "heterogeneous platforms". For instance, the client may speak French and the accountant may speak English, but the entire system will work fine as long as the counter can communicate with both of them.
Thirdly, it is easy to maintain and upgrade the servers. For instance, if an accountant is sick today, their absence will not affect the behavior of others. On the other hand, if a new bank employee joins the system as a new server, they will not affect existing servers as well.

Lastly, each client is independent, and will not affect each other requiring this service. Although there might be communication among the servers (bank employees), there is no need for the clients to communicate with each other directly. Every request is sent to the server through the network (counter) separately. Thus, the coupling of the system is reduced.

Implicit Invocation Style

Example of application in real life
Suppose that in a bubble tea café, a person is employed to take care of the order queue. When a customer finished ordering, the person is going to record the preferred name given by the customer with that customer's order and put it at the end of the order queue. When the items of an order are ready to serve, the person reads the order out loudly with the associating preferred name. This queuing mechanism in this bubble tea café can be a real-life example of the event-based architecture style, which is a variant of the implicit invocation style. In this example, the person records events in an event stream and the order queue can be considered as the event bus, whereas the customers and employees can be considered as the components. A customer can emit an event by making an order, while an employee can emit an event when finishing preparing the items and let the person to broadcast. All employees are listening on the event bus, while an employee start preparing when he/she get an order from the order queue. All customers are listening on the event bus as well, while a customer consumes the event when he/she hear of a ready-to-serve order with his/her preferred name.



How it is beneficial to the overall system
Firstly, this queuing mechanism can help enhance customer experience. If this architecture style is not being used, a customer would have to regularly check to see if his/her order items are ready. With the help of this, the customer is not tethered to a physical line and only need to pay attention to the event where the order with his/her preferred name is broadcasted. Also, event consumers access the stream at any time, so a customer can check the order queue to manage his/her time.

Secondly, it can reduce the operational costs for the shop by improving efficiency and eliminating the logistical requirements (e.g., barrier expenditures, customer traffic flow) of a physical queue.

Thirdly, the customers are independent from each other, which means they will not affect each other. They do not directly communicate with each other but communicate with the event bus.

Lastly, compared to monolithic solutions, this architecture solution can scale according to the data needs, be it the amount of data or the number of events that need to be processed.