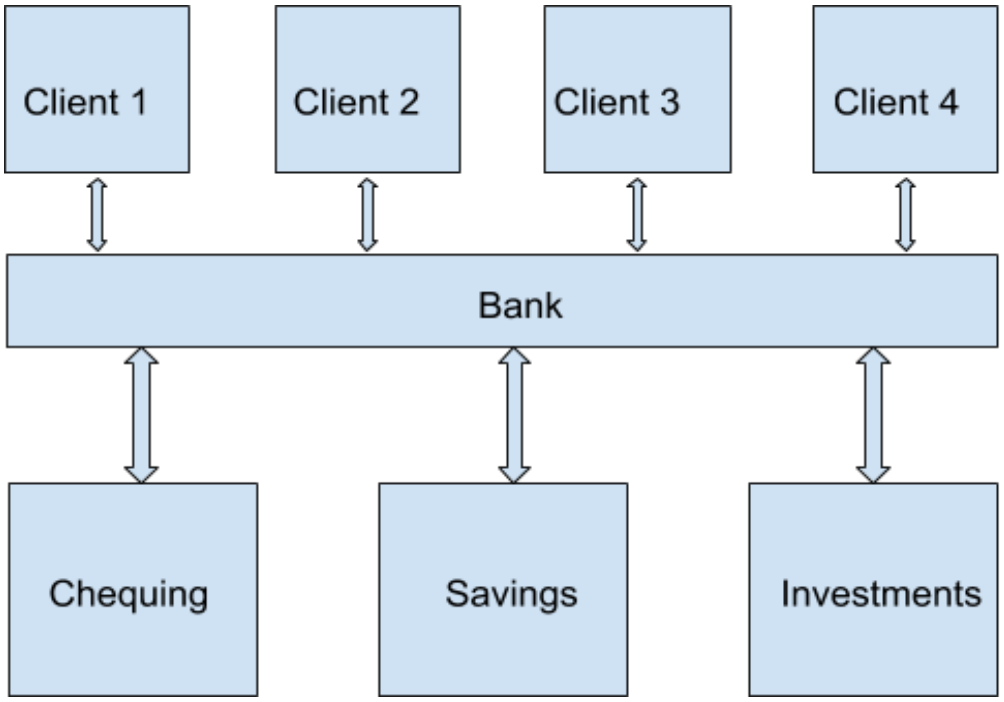
Client-Server Style

A real world non-software example of the client-server architecture style is a bank. In this example, the bank acts as the server and the bank goer as the client. Clients who want to access and/or request modifications on their bank information must contact any of their bank's locations, which is like an interface for them to view the distributed data stored at the bank. Similar to the functionality that multiple servers can provide, the bank can manage multiple standalone accounts (e.g. chequing, savings, etc). For each of these, the bank offers account-specific services such as accessible debit cards for chequing accounts, and interest rates for savings accounts. For all client requests, the query processing is done by the bank.



This style reduces coupling because the server does not need to know how many clients there are and who they are. The servers in this example are the different banking services such as chequing, savings and investment who serve those who visit it, but don't have to know who is visiting them, and only interacts with those who came for its service. In addition, each client doesn't need to know about anything other than their server, ie, the people going to the bank don't need to know anything about the other people; rather, they only need to know the bank they are going to. The clients are decoupled by this architecture as they don't need to know the inner working of the bank, as it doesn't affect them. The server is also decoupled by this architecture as the transaction between one client and the bank will not affect the transaction of another client and the bank. This style makes it easy to add new servers and upgrade existing ones, as implementation is hidden from the client. The architecture is robust enough to handle heterogeneous access points (i.e. various bank locations, ATMs, browser access, etc).

Pipe and Filter Style

A real-world example of a non-software use case of a *pipe and filter* architectural style is a factory assembly line. An assembly line consists of a number of independent assembly or modification stages that all operate on a central, continuous ordered input stream (generally by means of a conveyor belt). The input to each stage consists of the output of the previous stage in the form of the item being constructed. The assembly line was originally developed by Henry Ford to assist in the development of the Model T automobile. When the assembly line was developed it was observed that by splitting the assembly process into multiple, simple phases which could be carried out quickly by a small team of workers, it allowed the overall production process to be accelerated significantly. This concept is completely analogous to a software pipeline design architecture because it involves an ordered set of operations to be carried out on a central item.

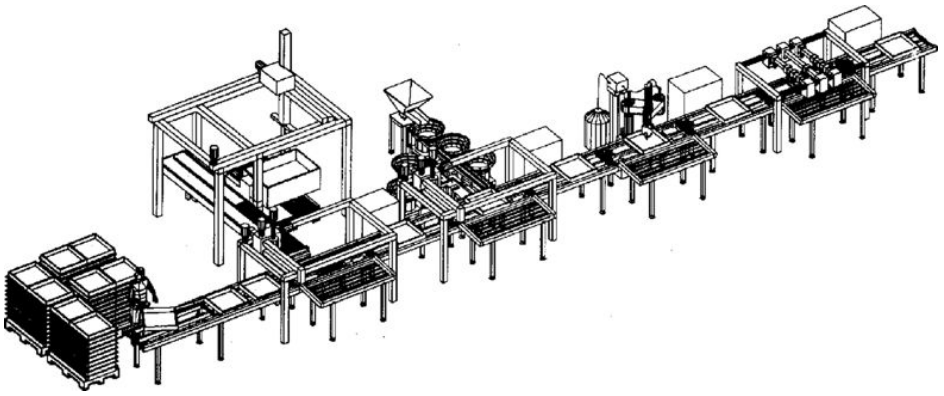


Figure 1: Conveyor belt assembly line [1]

The pipe and filter style allows for a large amount of decoupling in a system by only allowing input to a stage from an upstream stage (and output to a downstream stage). This maintains a strict ordering of operations by only allowing data to flow in a single direction. Each stage is naturally compartmentalized and can be treated as a blackbox with a well defined set of input and output interface. This quality allows for an extremely easy development process as each stage can be developed, maintained, and tested individually and then integrated with the other stages when they are ready.

Another major advantage to pipelining systems is their inherent comprehensibility. The concept of a pipeline is relatively easy for humans to understand and therefore reduces confusion over their designs and implementations. Additionally, because each stage of the pipeline can be viewed as a blackbox, each stage is completely reusable because as long as any system can meet the input and output specifications, the stage can be used.

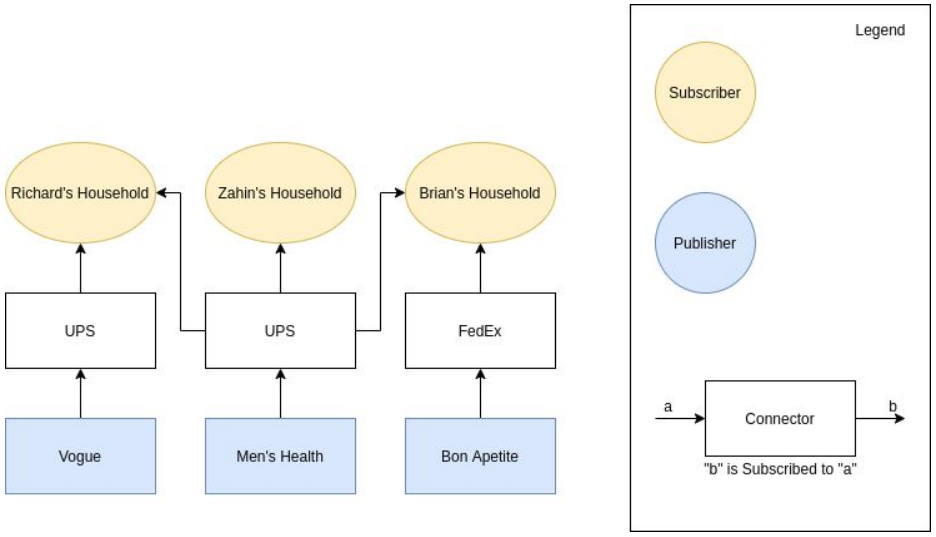
References:

[1] Reinhart, G., 2019. *Assembly Line*. [image] Available at: <https://media.springernature.com/original/springer-static/image/prt%3A978-3-642-20617-7%2F1/MediaObjects/978-3-642-20617-7_1_Part_Fig1-8_HTML.png> [Accessed 3 July 2020].

Publish-Subscribe Style

A real-world example of publish-subscribe style is a physical magazine and their readers. A reader (subscriber component) subscribes to a magazine company (publisher component) to receive magazines at their home, via a postal service (connector). The magazine companies maintain a subscription list of all the readers who pay to subscribe for that magazine, and they send their latest issue to all their readers via a postal service company (ex: UPS, FedEx, Canada Post, etc.). Households directly subscribe/unsubscribe to magazines by communicating directly to the magazine company and not via some third-party.

Using the publish-subscribe style can be beneficial for many reasons. First and foremost, it's an efficient way to send magazines, because it does not rely on users going to a magazine stand to pick up new editions of a magazine. New magazines are sent out to readers as they are made. This architecture also allows a reader to subscribe to multiple magazines (supports reusability) without the need to add complicated infrastructure to support this.



This style reduces coupling because each household need not know what their neighbour subscribes to, and they don’t care about who their magazines send to. Each household doesn’t need to know about new editions of magazines they don’t subscribe to (as opposed to an event bus style). Publishers are also de-coupled with this architecture, one publisher going bankrupt just means their readers will be unsubscribed, it has no impact on any other magazines or any other readers. The subscriber and magazine company are de-coupled because the inner-workings of the magazine does not affect the reader (staff turnover, new article types, etc.). The postal service used by a specific publisher does not need to be static either, a company may get a better deal by switching from UPS to FedEx, and the customer won’t need to adapt. This system allows for future changes like new magazines, new readers, and changes within each of those (even changes like families moving or payee of magazine within a household changing) with ease.