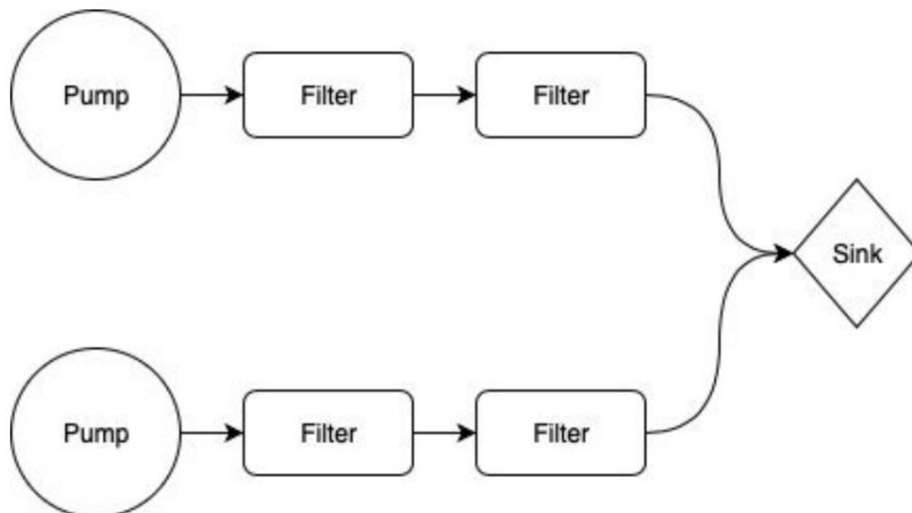
Pipe & Filter Architecture

Definition

- Pipe and filter is a component based architectural style that allows for the deconstructing of monolithic processes into small independent components consisting of Pumps, Filters, and a Sink. Data is passed from pipes through a variety of filters, and ultimately ends up in the sink.
- **Vocabulary for components and connectors:**
    - Pumps are the initial source of data which can be streamed or passed into an initial filter.
    - Filters are independent components that execute a specified task and do not share states with other filters or components. Filters often have buffers to be able to queue work, but are unaware of their upstream/downstream neighbours.
    - Sink or consumer is the target where the final output from a series of filters is collected.

Topological constraints



- Due to the nature of pipe and filter where data is passed through a variety of filters, the design usually must form some uni-directional sequence where the structure lacks cycles. This ensures that there is a clear defined workflow for data, and that the application isn't cycling back to previous filters.
- Although there can be multiple pumps, it's often the case that the structure of pipe and filter application only has one sink to aggregate processed information.
- Filters must not be aware of downstream/upstream providers. They must only communicate through the main communication channels (input/output pipes).

Applicable problems

- The processing required by an application can easily be broken down into a set of independent steps. This allows for modularized content that is not tightly coupled. Each of these steps can be performed by an application and have different scalability requirements.
    - Pipe and filter provides the flexibility of reordering the processing steps performed by an application, and ease with exchanging components for different use cases.
    - Such systems can benefit from distributed processing of such steps across various servers and simplifies the development process for developers as they no longer require an understanding of the entire system.
- Pipe and filter is most applicable in large workflows that can be decomposed into smaller independent components which can be chained and/or run asynchronous.
- The separate components normally wouldn't have any relevant knowledge of previous states, and could be rearranged with respect to functional constraints

Resilience to change

- Pipe and filter has very low resilience to change as each part of the system is componentized, leading to high cohesion and low coupling.
- It would be easy to create new filters and attach new pipes, without impacting the remaining code base. This means it would be trivial to expand a system that adopts this architecture.
- Changing pre-existing aspects of a system can be done with ease as well. With each filter being responsible for its key deliverables, developers do not require fully understanding how each and every piece behaves prior to making changes to a single component.

Negative behaviours

- This pattern requires a large amount of state to be transferred from component to another as no assumptions can be made about the previous process. As a result, any process that requires large amounts of state such that the process becomes inefficient should not utilize this method.
- This architecture is not recommended to use when developing highly dynamic, interactive systems. Filters are independent of another and execute a complete transformation of the input data and pipes do not support interaction, they only act as a stream of data.
- Linearized implementation of pipe and filters can give rise to delays in program execution. If a first filter displays a delay in parsing the input data, this delay will be passed on to the upcoming filters.
- This architecture cannot be used if a program requires a set of steps that cannot be executed independently or concurrently. This refers to highly coupled code.

Supported NFPs

- Scalability: Filters and pipes can be added/replicated to help distribute the load and increase functionality enabling to system to scale horizontally.
- Modularity: Breaking down monolithic systems into modular ones gives users the capability to change the behaviour of individual components without needing to modify large code bases and dependent processes.
- Reliability: If a filter fails, this architecture allows for ease of rescheduling the process to run on another parallel instance of the failed filter, allowing for failover resiliency.
- Efficiency: Filters can run in parallel to each other to help reduce overall processing time.
- Reusability: By having the code componentized into several filters, this results in having high reusability as they can be used in combination with multiple filters to complete various tasks.

Inhibited NFPs

- Dependability: With this model, a single point of failure at a filter would break the entire pipe.
- Simplicity: With the introduction of multiple filters, potentially across multiple servers, such a pattern can introduce significant complexities into a system.

Comparison with other architectures

- Client-server, layered, and pipe and filter architectures are similar in their objective.
    - Client-server can be thought of as a variation of layered architecture with two layers.
    - Pipe and filter only allows unidirectional flow of information, whereas client-server and layered architectures allow bidirectional flow.

Credit:

Students:
M.A.N.S (Aditi Shetty, Stephanie Fok, Nabil Barakati, Mufaddal Jerwalla)
Team Droids (Rohan Dave, Harsh Mistry, Davit Yeghshatyan, Shmuel Kantor)
Instructor and TAs:
Mei Nagappan, Achyudh Ram Keshav Ram, Aswin Vayiravan, Wenhan Zhu