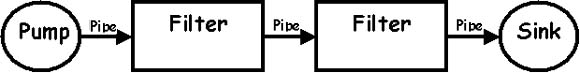
**Pipe-And-Filter**

A very simple, yet powerful architecture, that is also very robust. It consists of any number of components (filters) that transform or filter data, before passing it on via connectors (pipes) to other components. The filters are all working *at the same time*. The architecture is often used as a simple sequence, but it may also be used for very complex structures.

لوله و فیلتر

یک معماری بسیار ساده اما در عین حال قدرتمند ، همچنین بسیار مقاوم است. قبل از انتقال آن از طریق اتصالات (لوله ها) به سایر اجزاء ، از هر تعداد مؤلفه (فیلتر) که داده ها را تبدیل یا فیلتر می کنند تشکیل شده است. فیلترها همه هم زمان کار می کنند. معماری اغلب به عنوان یک ترتیب ساده مورد استفاده قرار می گیرد ، اما ممکن است برای ساختارهای بسیار پیچیده نیز مورد استفاده قرار گیرد.



The **filter** transforms or *filters* the data it receives via the pipes with which it is connected. A filter can have any number of input pipes and any number of output pipes.

The **pipe** is the connector that passes data from one filter to the next. It is a directional stream of data, that is usually implemented by a data buffer to store all data, until the next filter has time to process it.

The **pump** or producer is the data source. It can be a static text file, or a keyboard input device, continously creating new data.

The **sink** or consumer is the data target. It can be a another file, a database, or a computer screen.

فیلتر داده هایی را که از طریق لوله هایی که به آن وصل شده اند ، تبدیل یا فیلتر می کند. یک فیلتر می تواند تعداد لوله های ورودی و تعداد لوله های خروجی را داشته باشد.

لوله اتصالی است که داده ها را از یک فیلتر به فیلتر دیگر منتقل می کند. این یک جریان جهت داده است که معمولاً توسط یک بافر داده برای ذخیره کلیه داده ها اجرا می شود ، تا زمانی که فیلتر بعدی زمان پردازش آن را داشته باشد.

پمپ یا تولید کننده منبع داده است. این می تواند یک فایل متنی استاتیک یا یک دستگاه ورودی صفحه کلید باشد که بطور مداوم داده های جدیدی را ایجاد می کند.

سینک یا مصرف کننده هدف داده است. این می تواند یک فایل دیگر ، یک پایگاه داده یا یک صفحه رایانه باشد.

**Examples**

* Unix programs. The output of one program can be linked to the input of another program.
* Compilers. The consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation.

مثال ها

* برنامه های یونیکس. خروجی یک برنامه را می توان با ورودی یک برنامه دیگر مرتبط کرد.
* کامپایلرها. فیلترهای متوالی ، تجزیه و تحلیل واژگانی ، تجزیه ، تحلیل معنایی و تولید کد را انجام می دهند.

## Where does it come from?

The popularity of the architecture is mainly due to the Unix operating system. It has become popular because Ken Thomson (who created Unix, together with Dennis Ritchie) decided to limit the architecture to a linear pipeline. Using the architecture at all was an idea of Doug McIlroy, their manager at Bell Labs at the time (1972). Both filters (coroutines) and pipes (streams) were not new, but it is not clear to me who designed the architecture of linking the coroutines by streams. As far as I can see, the design was made by Doug McIlroy.

از کجا آمده است؟

محبوبیت معماری بیشتر ناشی از سیستم عامل یونیکس است. این محبوبیت پیدا کرده است زیرا Ken Thomson (که یونیکس را به همراه Dennis Ritchie ایجاد کرد) تصمیم گرفت معماری را به یک خط لوله خطی محدود کند. استفاده از معماری اصلاً ایده ای بود از Doug McIlroy ، مدیر آنها در آزمایشگاههای بل در آن زمان (1972). هر دو فیلتر (coroutines) و لوله (streams) جدید نبودند ، اما برای من مشخص نیست که چه کسی معماری اتصال coroutines را توسط streams طراحی کرده است. تا آنجا که من می بینم ، این طرح توسط Doug McIlroy ساخته شده است.

## When should you use it?

This architecture is great if you have a lot of transformations to perform and you need to be very flexible in using them, yet you want them to be robust.

چه زمانی باید از آن استفاده کنید؟

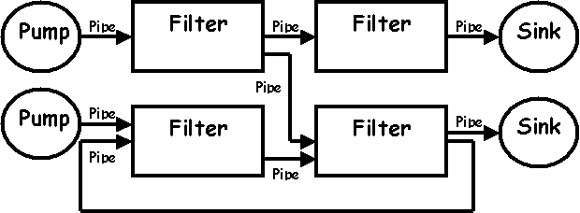
این معماری بسیار عالی است اگر تحولات زیادی را برای انجام آن داشته باشید و در استفاده از آنها باید بسیار انعطاف پذیر باشید ، در عین حال می خواهید آنها قوی باشند.

## How does it work?

The application links together all inputs and outputs of the filters by pipes, then spawns separate threads for each filter to run in. Here's an idea of the relationships that can be created between the different filter processes, through pipes.

چگونه کار می کند؟

این برنامه تمام ورودی ها و خروجی فیلترها را توسط لوله ها به هم پیوند می دهد ، سپس موضوعات جداگانه ای را برای اجرای هر فیلتر ایجاد می کند. در اینجا ایده ای از روابطی که می توان بین فرآیندهای مختلف فیلتر ایجاد شود ، از طریق لوله ها وجود دارد:

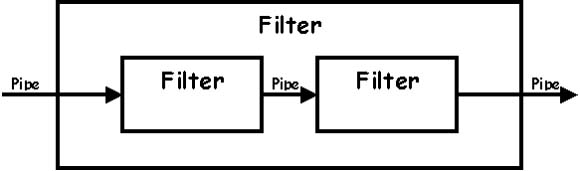


All filters are processes that run (virtually) at the same time. That means, they can run as different threads, coroutines, or be located on different machines entirely. Every pipe connected to a filter has its own role in the function of the filter. So if you connect a pipe, you also need to specify the role it plays in the filter process. The filters should be made so robust that pipes can be added and removed at runtime. Every time the filter performs a step, it reads from its input pipes, performs its function on this data, and places the result on all output pipes. If there is insufficient data in the input pipes, the filter simply waits.

همه فیلترها فرآیندی هستند که بطور همزمان اجرا می شوند. این بدان معناست که آنها می توانند به صورت موضوعات مختلف ، کوروتین ها اجرا شوند یا کاملاً در دستگاه های مختلف واقع شوند. هر لوله متصل به یک فیلتر نقش خود را در عملکرد فیلتر دارد. بنابراین اگر شما یک لوله را وصل می کنید ، باید نقشی که در فرآیند فیلتر نیز ایفا می کند را مشخص کنید. فیلترها باید چنان محکم ساخته شوند که لوله ها در زمان اجرا اضافه شوند و حذف شوند. هر بار فیلتر یک قدم را انجام داد ، از لوله‌های ورودی خود بخواند ، عملکرد خود را روی این داده‌ها انجام دهد و نتیجه را در تمام لوله‌های خروجی قرار دهد. اگر داده های کافی در لوله های ورودی وجود نداشته باشد ، فیلتر به سادگی منتظر می ماند.

The architecture also allows for a recursive technique, whereby a filter itself consists of a pipe-filter sequence:

معماری همچنین یک تکنیک بازگشتی را امکان پذیر می کند ، به موجب آن یک فیلتر خود شامل یک دنباله فیلتر لوله است:



## Problems

* If a filter needs to wait until it has received all data (e.g. a sort filter), its data buffer may overflow, or it may deadlock.
* If the pipes only allow for a single data type (a character or byte) the filters will need to do some parsing. This complicates things and slows them down. If you create different pipes for different datatypes, you cannot link any pipe to any filter.

چالش ها و مسائل

* اگر یک فیلتر باید منتظر بماند تا تمام داده ها را دریافت کند (به عنوان مثال یک فیلتر مرتب سازی) ، بافر داده آن ممکن است سرریز شود ، یا ممکن است بن بست باشد.
* اگر لوله ها فقط به یک نوع داده واحد (یک کاراکتر یا بایت) اجازه دهند ، فیلترها نیاز به تجزیه و تحلیل دارند. این کار مسائل را پیچیده و آرام می کند. اگر لوله های مختلفی را برای داده های مختلف ایجاد می کنید ، نمی توانید هیچ لوله ای را به هر فیلتر وصل کنید.

## Common implementation techniques

* Filters are commonly implemented by separate threads. These may be either hardware or software threads/coroutines.

تکنیک های اجرای مشترک

* فیلترها معمولاً توسط موضوعات جداگانه اجرا می شوند. اینها ممکن است یا موضوعات سخت افزاری یا نرم افزارها باشند.

## Links

* [Course on Software Architectures - Pipe-And-Filter](http://www4.desales.edu/~dlm1/it533/class03/pipe.html)
* [The Evolution of the Unix Time-sharing System](http://cm.bell-labs.com/cm/cs/who/dmr/hist.html)

<http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html>