Data-flow Systems

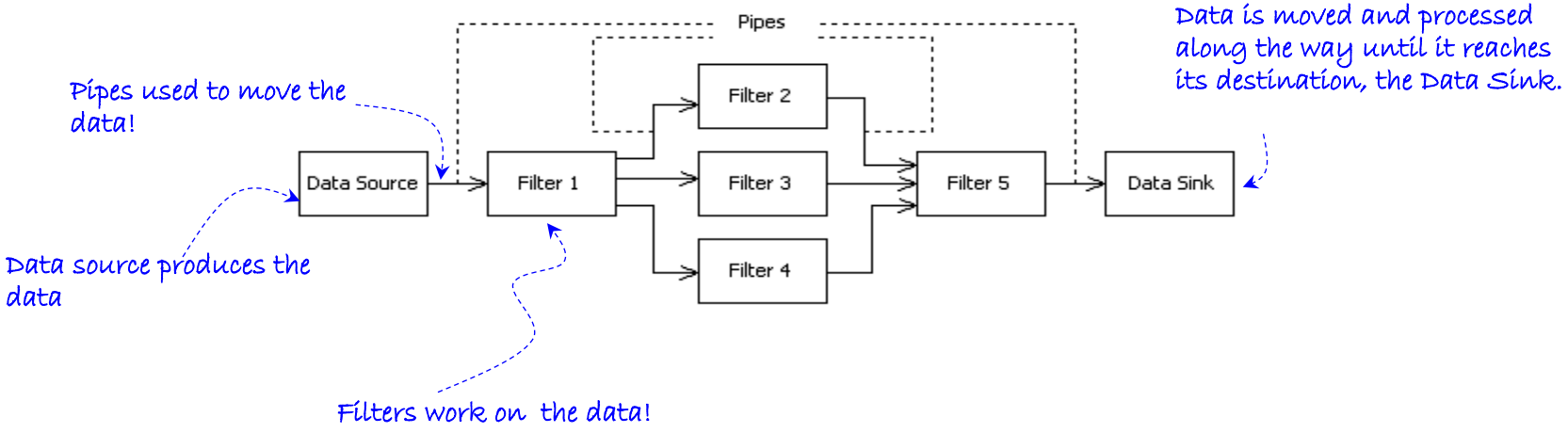
- Data-flow systems are decomposed around the central theme of transporting data (or data streams) and transforming the data along the way to meet application-specific requirements.

- Components perform the data processing and transformations that need to take place before forwarding the data to the next component.
- Connectors implement the data transport mechanisms
  - Intra-process communication
    - Direct function call, etc.
  - Inter-process communication
    - Sockets, pipes, etc.

Pipes-and-Filters Architectural Style

- *Pipes-and-Filters* is an example of an architectural style for data-flow systems.

- Pipes-and-Filters is composed of the following components:
  - Data source
    - Produces the data
  - **Filters**
    - Process the data
  - **Pipes**
    - Provide connections between data source and filter, filter to filter, and filter to data sink.
  - Data Sink
    - Data consumer



Pipes-and-Filters in Unix

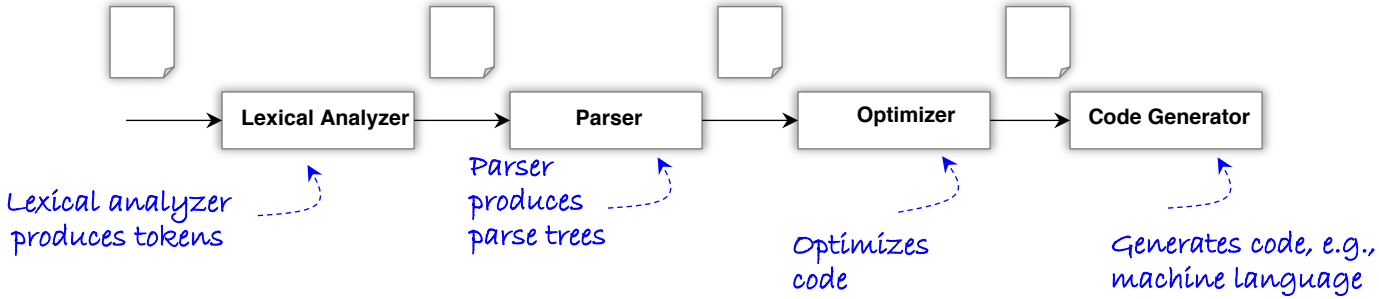
- Create a shell command to *count the number of* `http` *processes running*:

```
$ ps –ef | grep http | wc -l
```

Characteristics of Filters in Unix

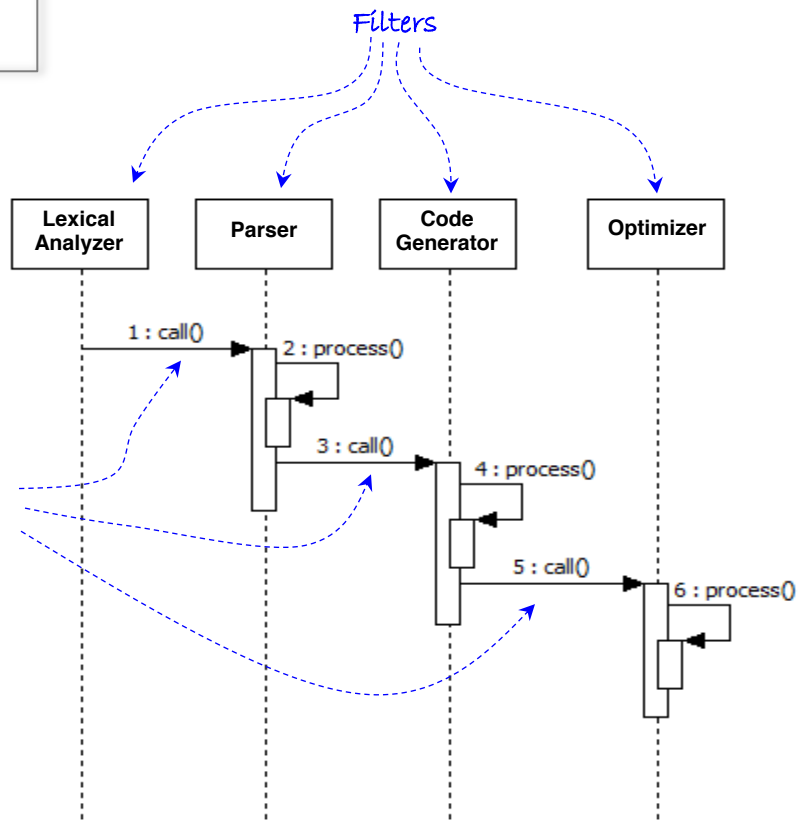
- Filters are independent entities.
- A filter does not share state with other filters.
- Filter does not have knowledge of up- or down- stream filters.
- All data does not need to be processed for next filter to start working.
- Incremental transformation of data by successive filters.
- It is possible to combine filters in any order, although this won’t guarantee desired semantics.

The Architecture of a Compiler



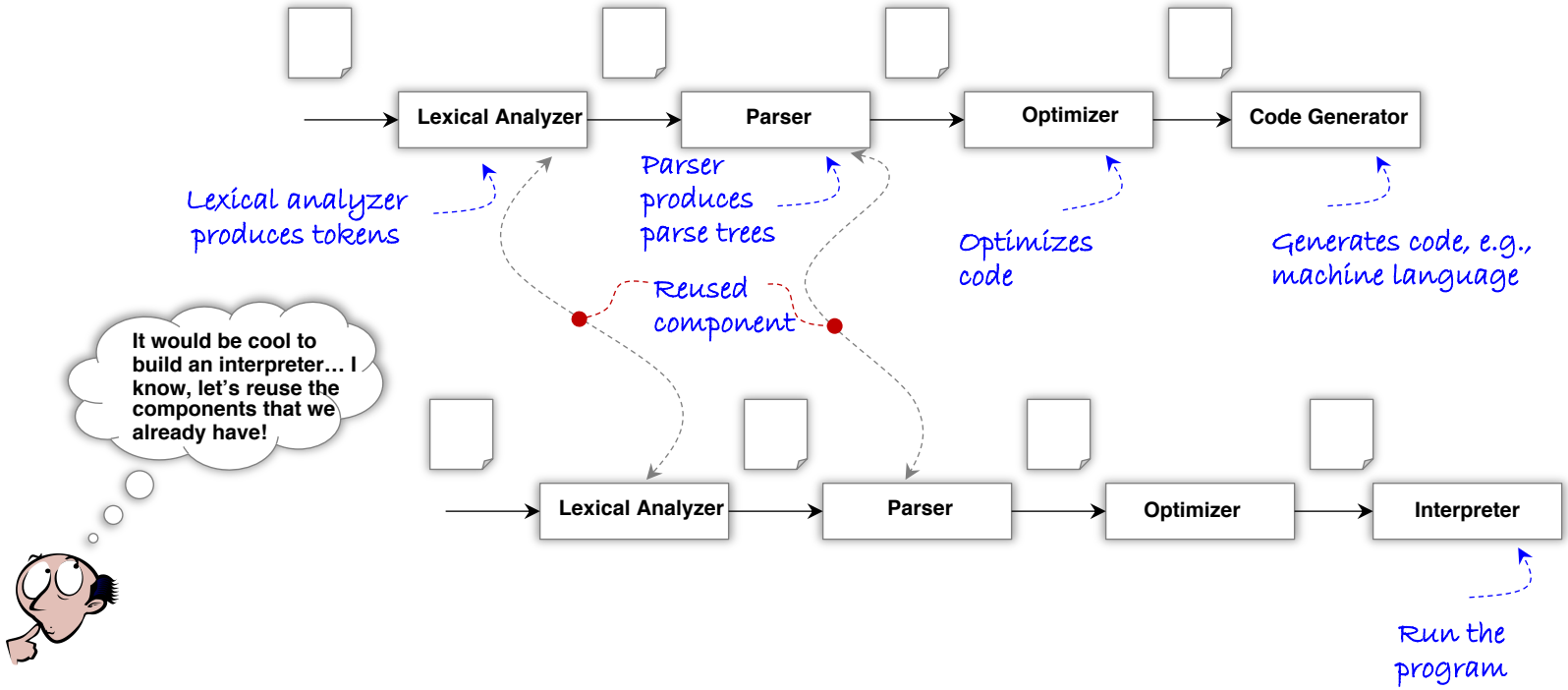
The Architecture of a Compiler (cont'd)

A UML Sequence Diagram



In this example, the "Pipes" are simply function calls!

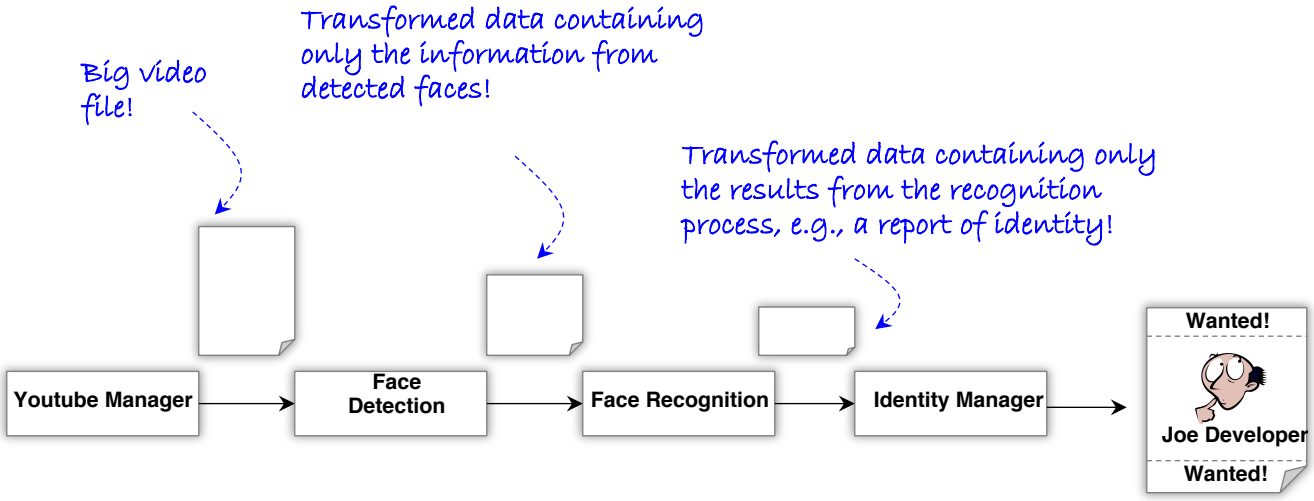
The Architecture of an Interpreter



The Architecture of a Video Processing Application

- Assume we have the following software components that help us identify the identity of an individual in a video clip:
  - A s/w component that accesses videos (with audio) from YouTube (*YoutubeManager*).
  - A s/w component that determines if a face is in the video (*FaceDetection*).
  - A s/w component that detects the identity of the person from the detected face (*FaceRecognition*).
  - A s/w component that produces a report of the results (*IdenityManager*).
- What does a possible architecture look like for a s/w application that processes videos and identifies faces?

The Architecture of a Video Processing Application



Advantages of Pipes-and-Filters

- Easy to understand: system behavior is a succession of component behaviors.
- Filter addition, replacement, and reuse
  - Possible to hook any two filters together.
- Concurrent execution.

Disadvantages of Pipe-and-Filters

- Does not work well with interactive applications.
- If something goes wrong in one of the parts of the pipeline, the entire pipeline fails.
- Excessive parsing and un-parsing may lead to loss of performance.
- Complex data structures to be exchanged between filters may require custom (that is, non-generic) implementation.