Software Design & Architecture

Architectural Styles / Pipe-Filter, Layered, Repository, and others

Pengyu Nie

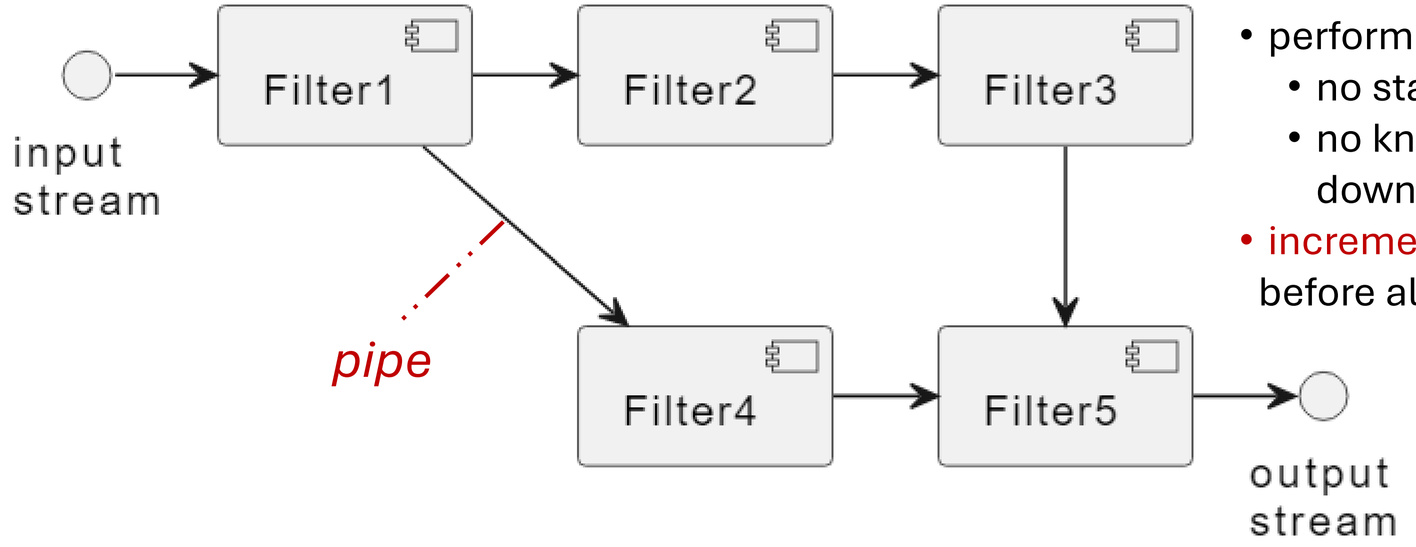
Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

Agenda

- Pipe-filter
- Layered
- Repository
- Implicit invocation (brief)
- Process-control (brief)
- Wrapup architectural styles

Pipe-Filter

- Suitable for applications that require a defined series of independent computations to be performed on ordered data

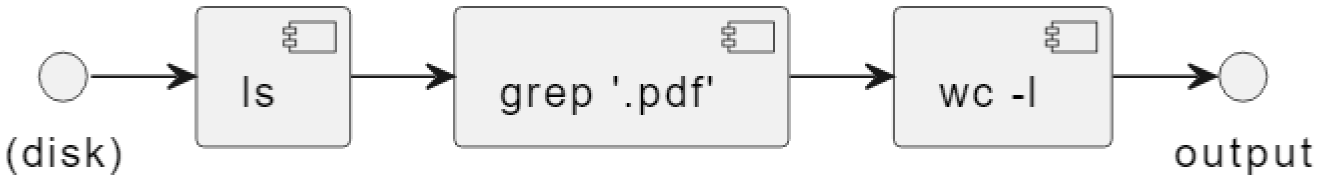


filter

- perform local transformations
  - no state sharing among filters
  - no knowledge about upstream or downstream filters
- incremental processing: outputs begin before all inputs are consumed

Pipe-Filter – Examples

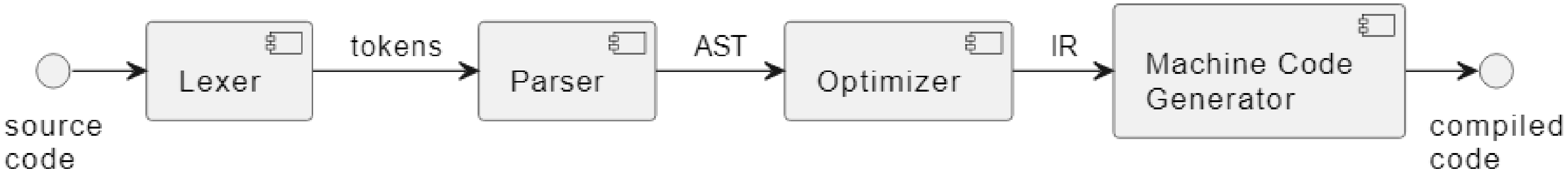
Unix Shell



```
# counting pdf files in current directory
ls | grep ‘.pdf’ | wc -l
```

pipeline variant
linear sequence of filters

Compiler



batch sequential variant
each filter process all inputs before producing any output

Pipe-Filter – Pros and Cons

+ Readability, Maintenability, Reusability

  + filters with the same input/output data format can be used interchangeably

  + filters can be easily replaced or improved

+ Efficiency: naturally support concurrent execution

+ Permit throughput and deadlock analyses

- Complexity

- Efficiency: loss of performance due to (de)serialization

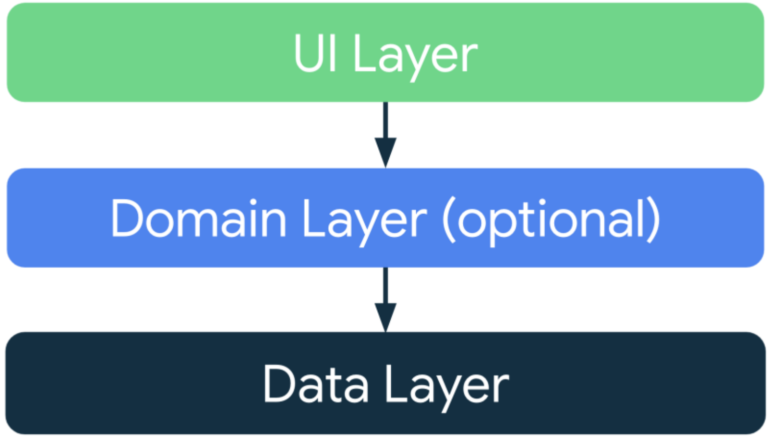
- Not for interactive systems

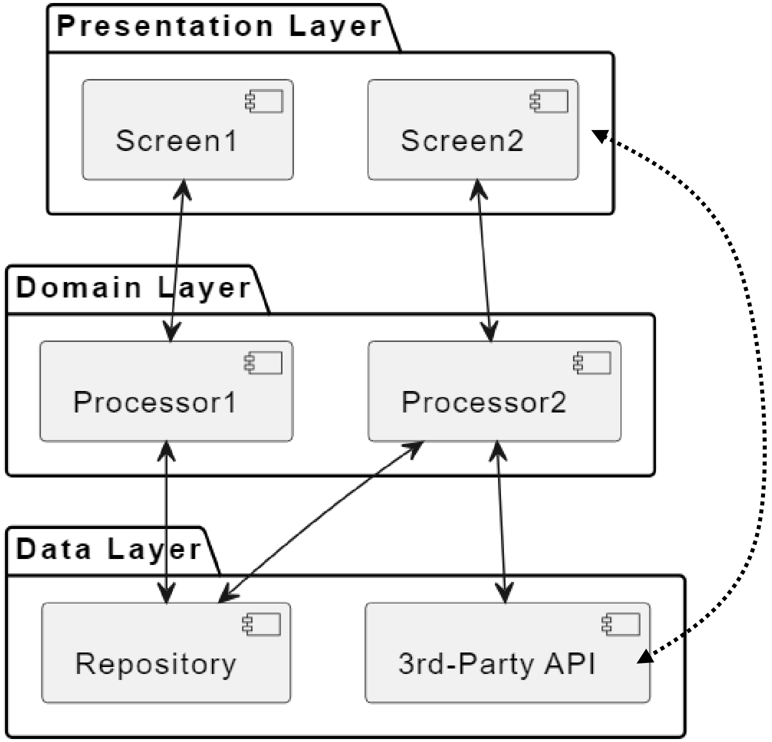
variant: can be improved by making filters less isolated

- sharing cache among filters
- using customized data formats on some pipes

Layered

- Suitable for applications that can be organized into a hierarchy of layers, where each layer may obtain services from a layer above or below it



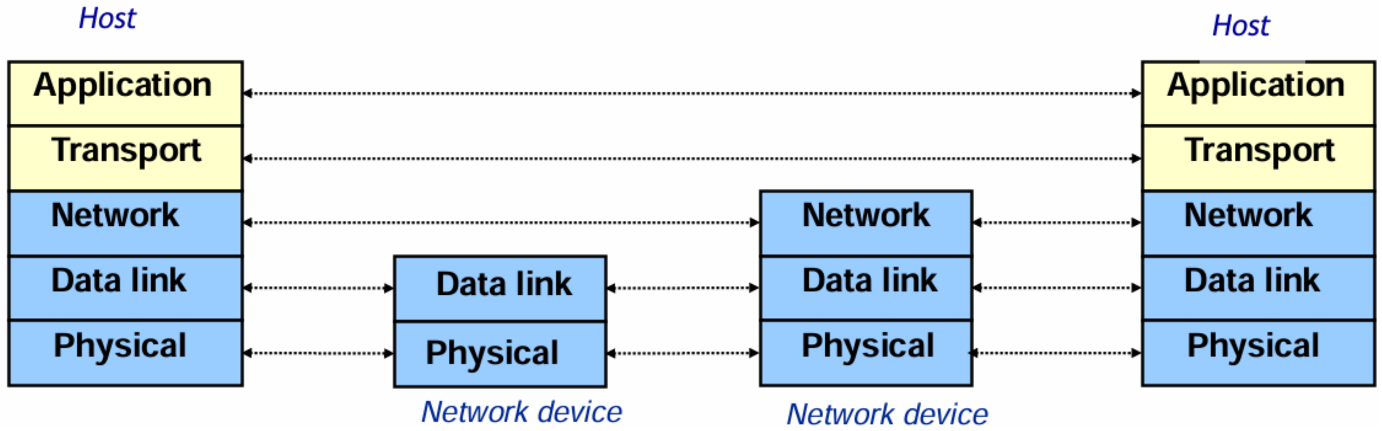


variant: allowing non-adjacent layers to communicate directly (may improve efficiency at the cost of lower readability)

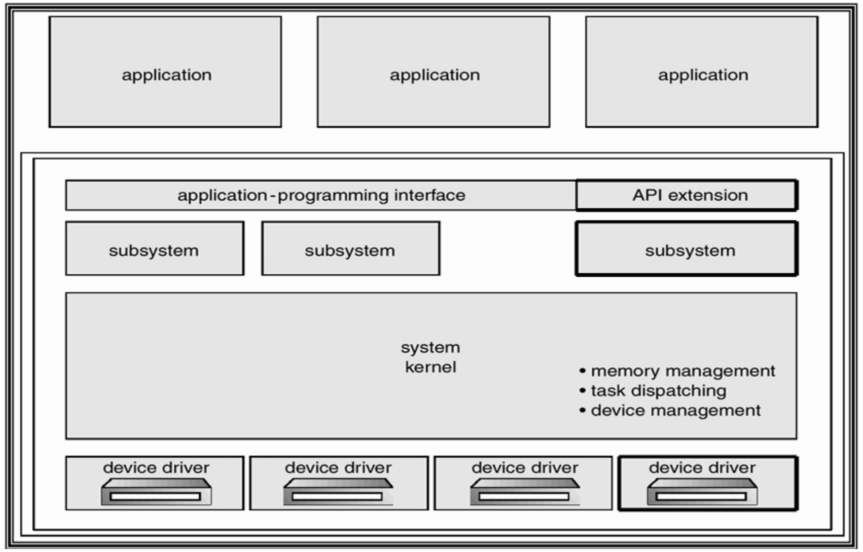
image source: https://developer.android.com/topic/architecture

Layered – (More) Examples

Computer Network



Operating System



Computer network layers: https://student.cs.uwaterloo.ca/~cs456/lectures/lec03.pdf
Operating system layers: https://www.eecg.utoronto.ca/~jacobsen/os/2007s/os-architecture.pdf

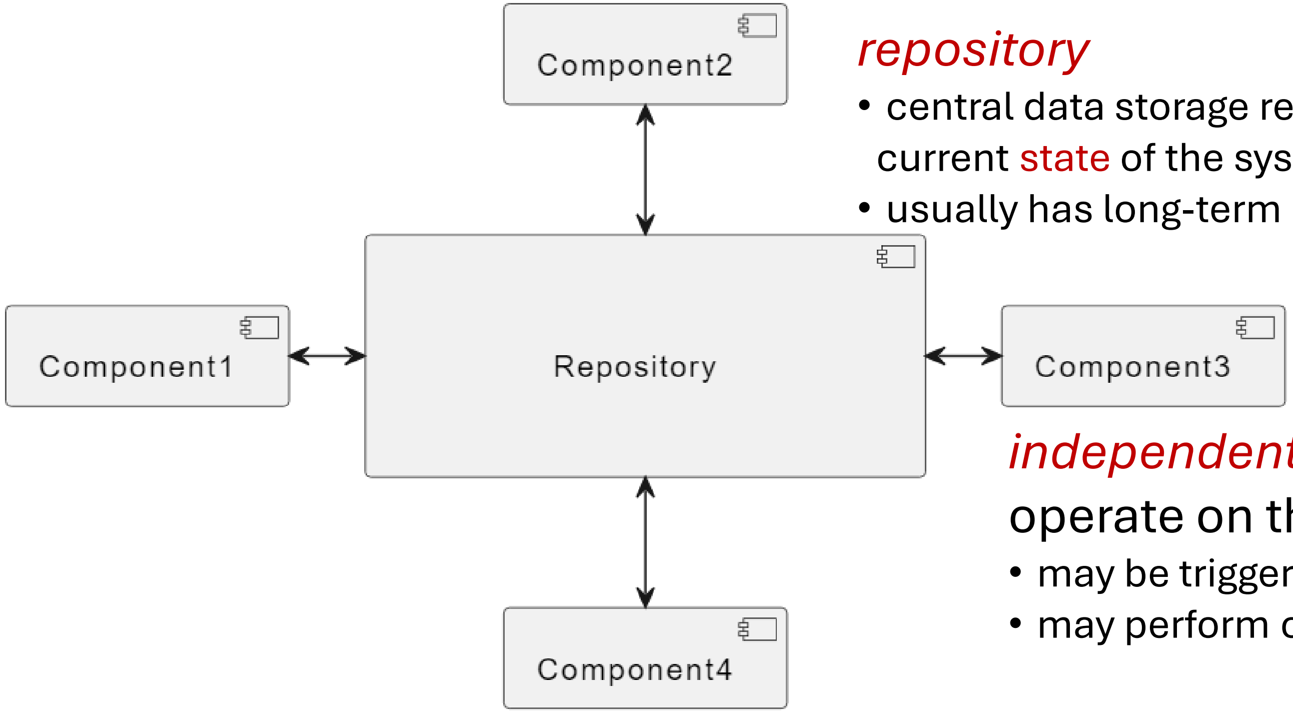
Layered – Pros and Cons

+ Readability, Maintainability, Reusability
    + changes to one layer affects at most two adjacent layers
    + different implementations of the same layer can be used interchangeably
+ Design advantage based on the increasing levels of abstraction

- Not all systems are easily structured in a layered fashion
- Efficiency: performance requirements may force the coupling of high-level functions to their low-level implementations

Repository (aka Data-Centered)

- Suitable for applications in which the central issue is establishing, augmenting, and maintaining a complex central body of information



repository

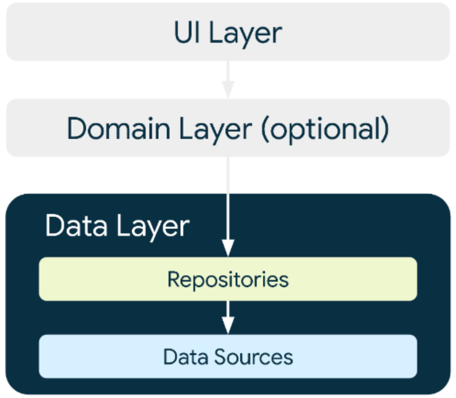
- central data storage representing the current state of the system
- usually has long-term persistence

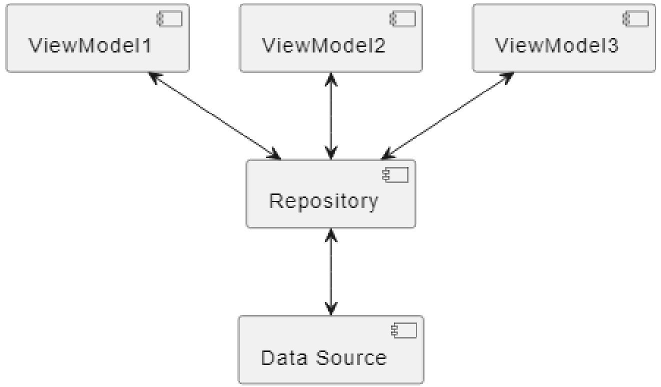
independent components that operate on the repository

- may be triggered by changes on the repository
- may perform concurrent computes and data accesses

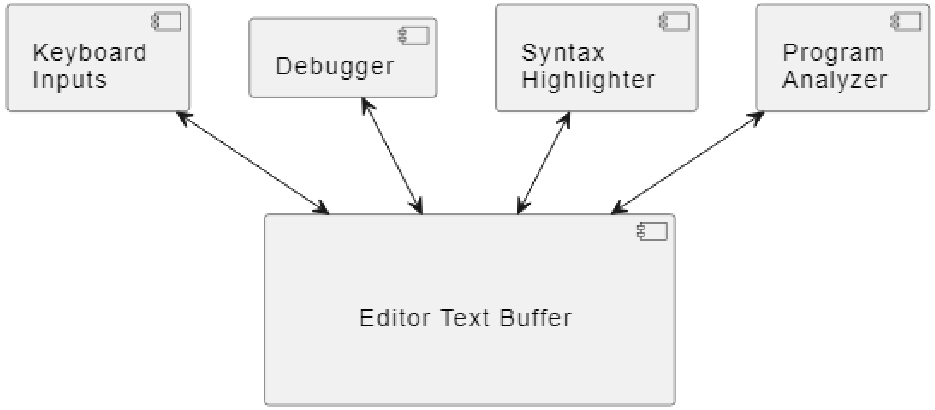
Repository – Examples

Android App Data Layer





Code Editor



more about the repository in Android data layer: https://developer.android.com/topic/architecture/data-layer

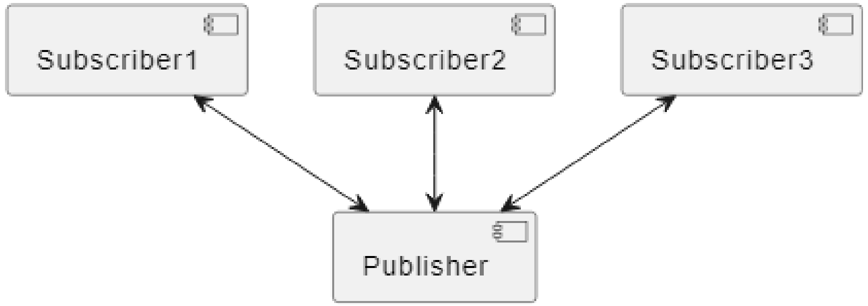
Repository – Pros and Cons

+ Readability, Maintainability, Reusability
    + repository can be reused or shared across different parts of the application
    + components communicate through the repository's sharing model
+ Reliability: centralized data management
+ Efficiency: avoid copies of large amounts of data in multiple components

- Complexity
    - must agree on a data schema a priori (or extra layer of mapping)
- Evolvability: evolving data schema requires changing all components
- Single point of security failure; difficult to distribute data

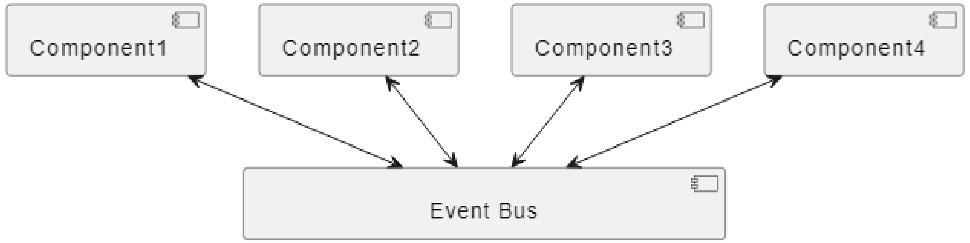
Implicit Invocation

- Suitable for applications where the components producing data do not directly know what other components may consume data



publish-subscribe variant

subscribers register to receive specific messages from publishers
e.g., social media, RSS



event-based variant

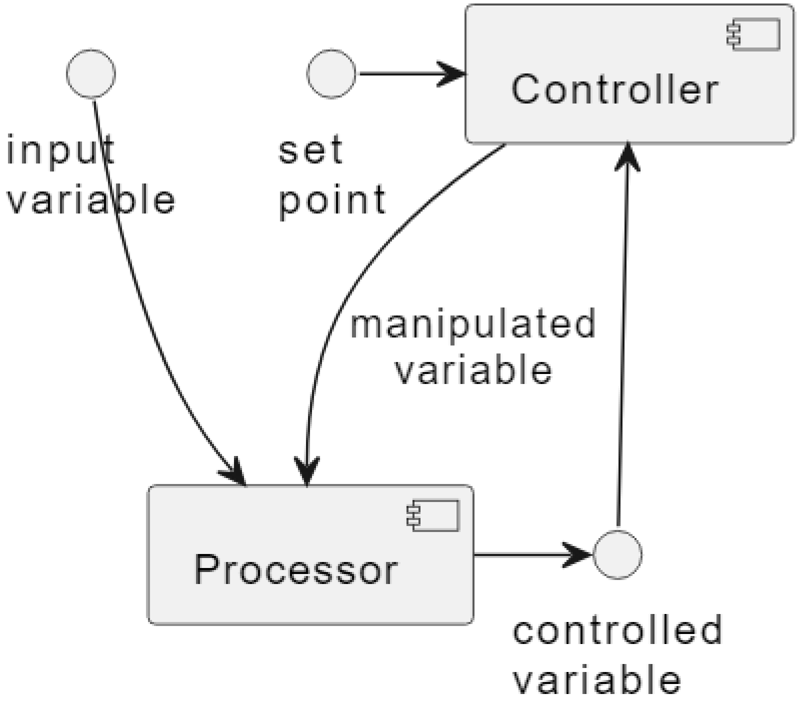
components asynchronously emit and receive events communicated over the event bus
e.g., IDE, GUI events

+ Scalability and flexibility at runtime

- Hidden dependencies; unpredictable execution order

Process-Control (aka Feedback-Control)

- Suitable for applications whose purpose is to maintain specified properties of the outputs of the process at (sufficiently near) given reference values



Examples: temperature controller, autonomous driving

+ Reliability and Robustness: adapt to changing conditions

- Cost for continuous monitoring; latency issues

Architectural Styles Epilogue

MVVM MVC MVP	Server-Client	Pipe-Filter
Layered	Microservices	Implicit Invocation
Repository	Serverless	Process-Control

- Choose architectural styles based on the problem natural and NFRs
- Reference the architecture of famous open-source applications: https://aosabook.org/en/index.html
- The right architecture is the one that addresses the real-world needs, even if that means bending or blending traditional styles

Agenda (recap)

- … architectural styles …

- Take-home exercise:
  What architectural styles are appropriate for your application (except for the obvious ones: MVVM, standard Android app layers)?
  - use UML diagrams to represent your architecture
  - guide your project development

- P3 Iteration 1 Demo this Wednesday, come at your assigned slot!