BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SS ZG653 (RL 10.1): Software Architecture

Pipe and Filter Pattern

Instructor: Prof. Santonu Sarkar

Pipes and Filters

A structure for systems that process a stream of data

Filter

- Has interfaces from which a set of inputs can flow in and a set of outputs can flow out
- processing step is encapsulated in a filter component
- Independent entities
- Does not share state with other filters.
- Does not know the identity to upstream and downstream filters
- All data does not need to be processed for next filter to start working

Pipes

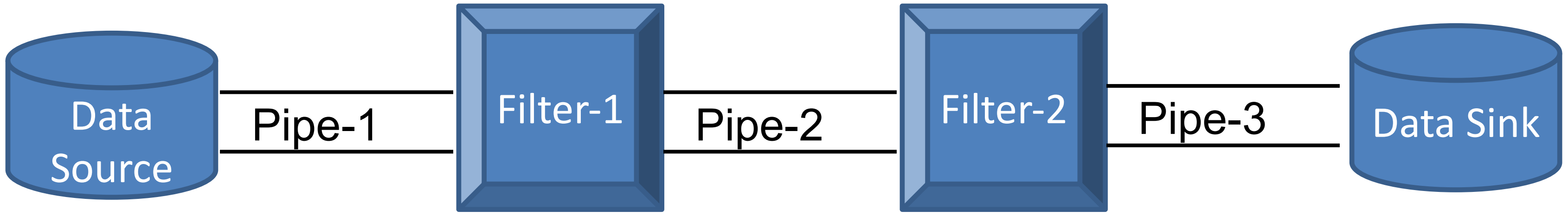
- Data is passed through pipes between adjacent filters
- Stateless data stream
- Source end feeds filter input and sink receives output.

Recombining filters allows you to build families of related systems

Pipes and Filters – 3 part schema

Context	**Processing Data Streams**
Problem	System that must process or transform a stream of input data. Multi-stage operations on data (workflow) Many developers may work on different stages Requirements may change
Forces	• Future enhancements – exchange processing steps or recombination • Reuse desired, hence small processing steps • Non adjacent processing steps do not share information • Different sources of data exist (different sensor data) • Store final result in various ways • Explicit storage of intermediate results should be automatically done • Multiprocessing the steps should be possible
Solution	Pipes and filters – data source to data sink

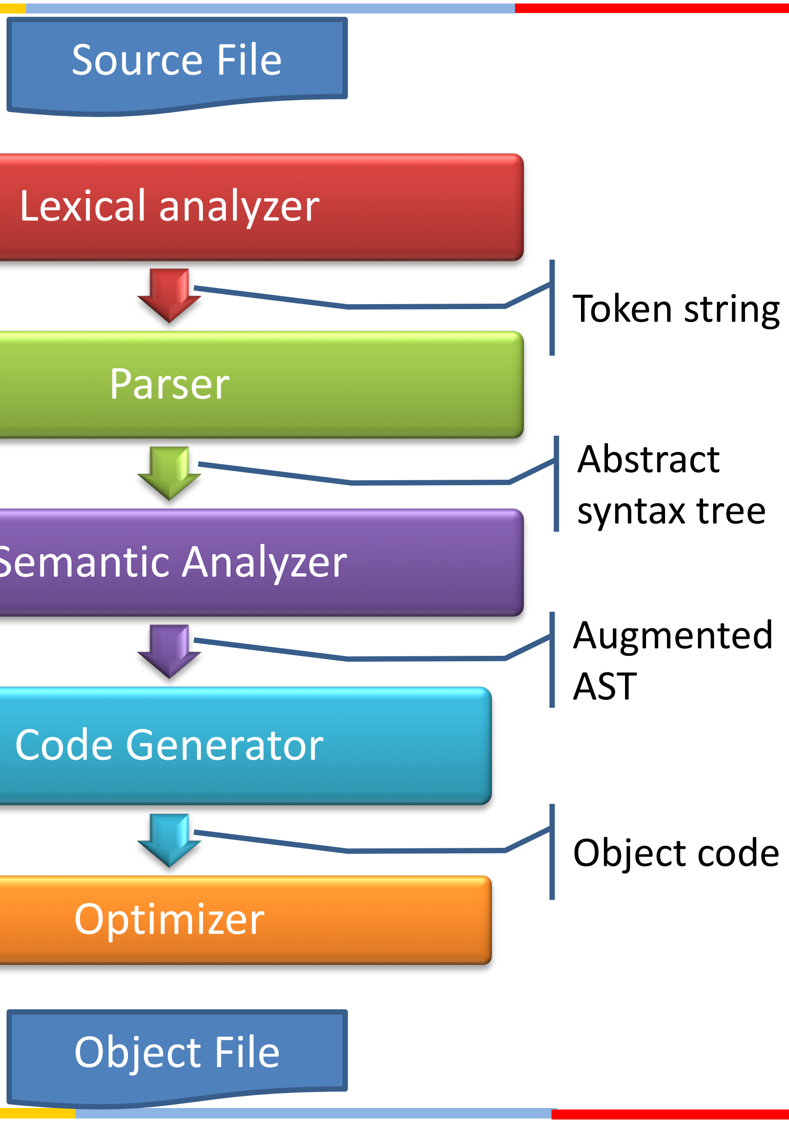
Simple case



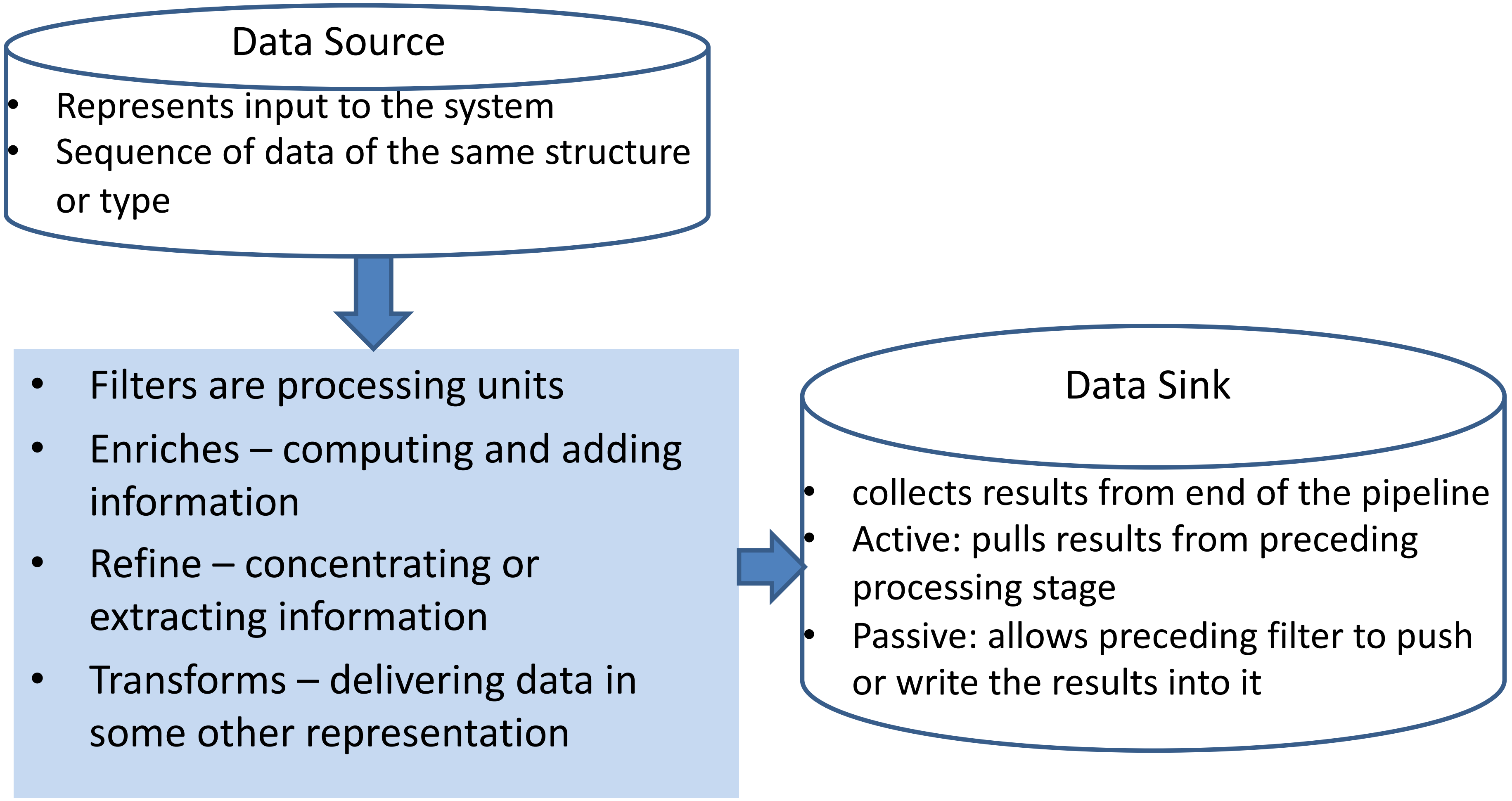
```
ls scores| grep -e July | sort
```

- Data source – file containing scores
- Filters
  - Listing of scores
  - Filtering only July scores
  - Sorting of records

Known Example –Compiler Design



Various Components



Data Source

- Represents input to the system
- Sequence of data of the same structure or type

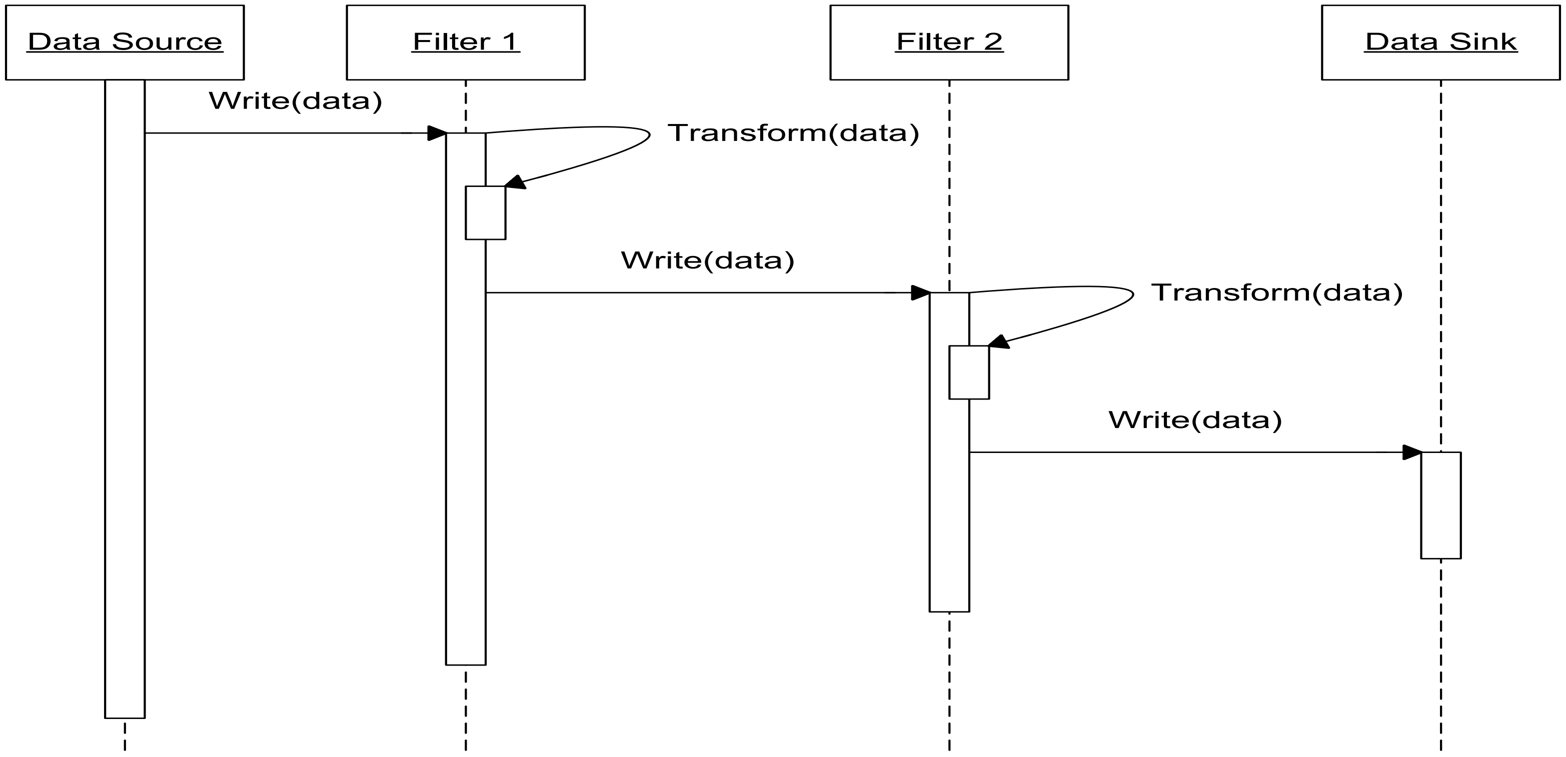
- Filters are processing units
- Enriches – computing and adding information
- Refine – concentrating or extracting information
- Transforms – delivering data in some other representation

Data Sink

- collects results from end of the pipeline
- Active: pulls results from preceding processing stage
- Passive: allows preceding filter to push or write the results into it

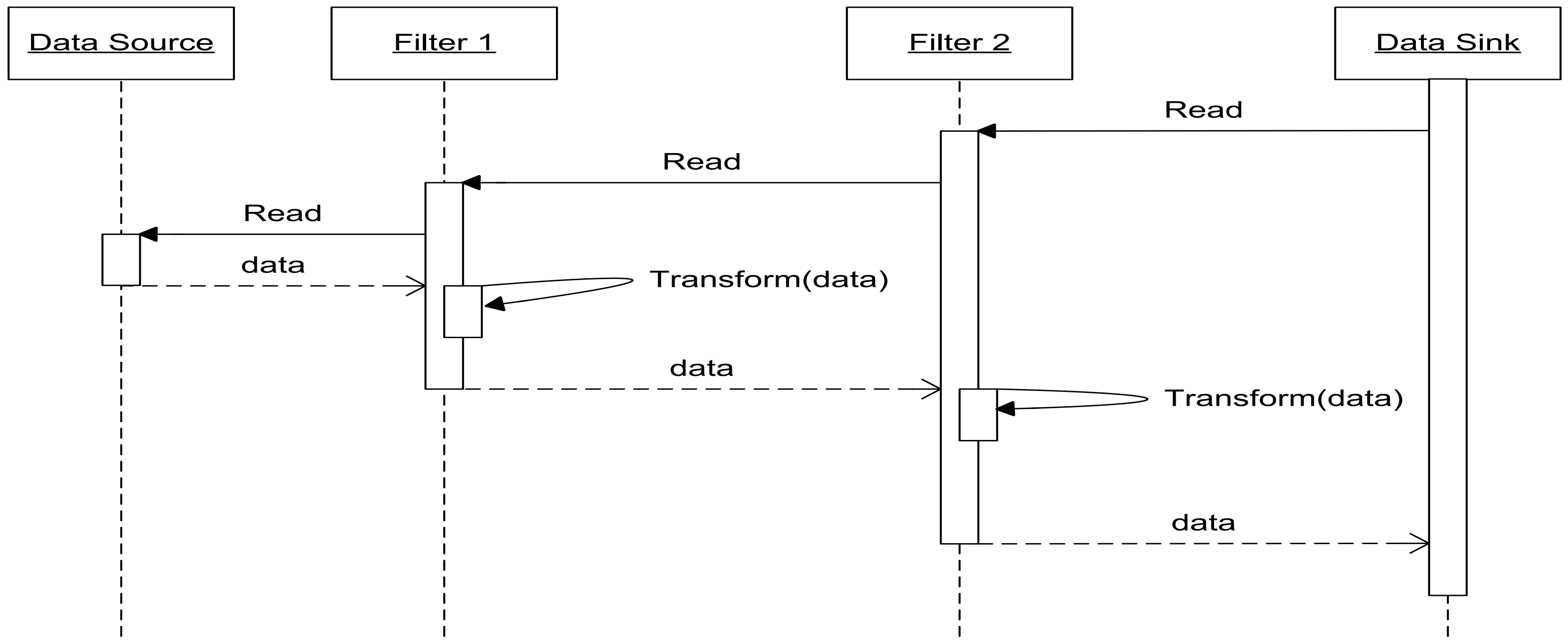
Scenario-1

– Push pipeline [Activity starts with the Data source]
– Filter activity started by writing data to the filters
– Passive Filter [Use direct calls to the adjacent pipeline]



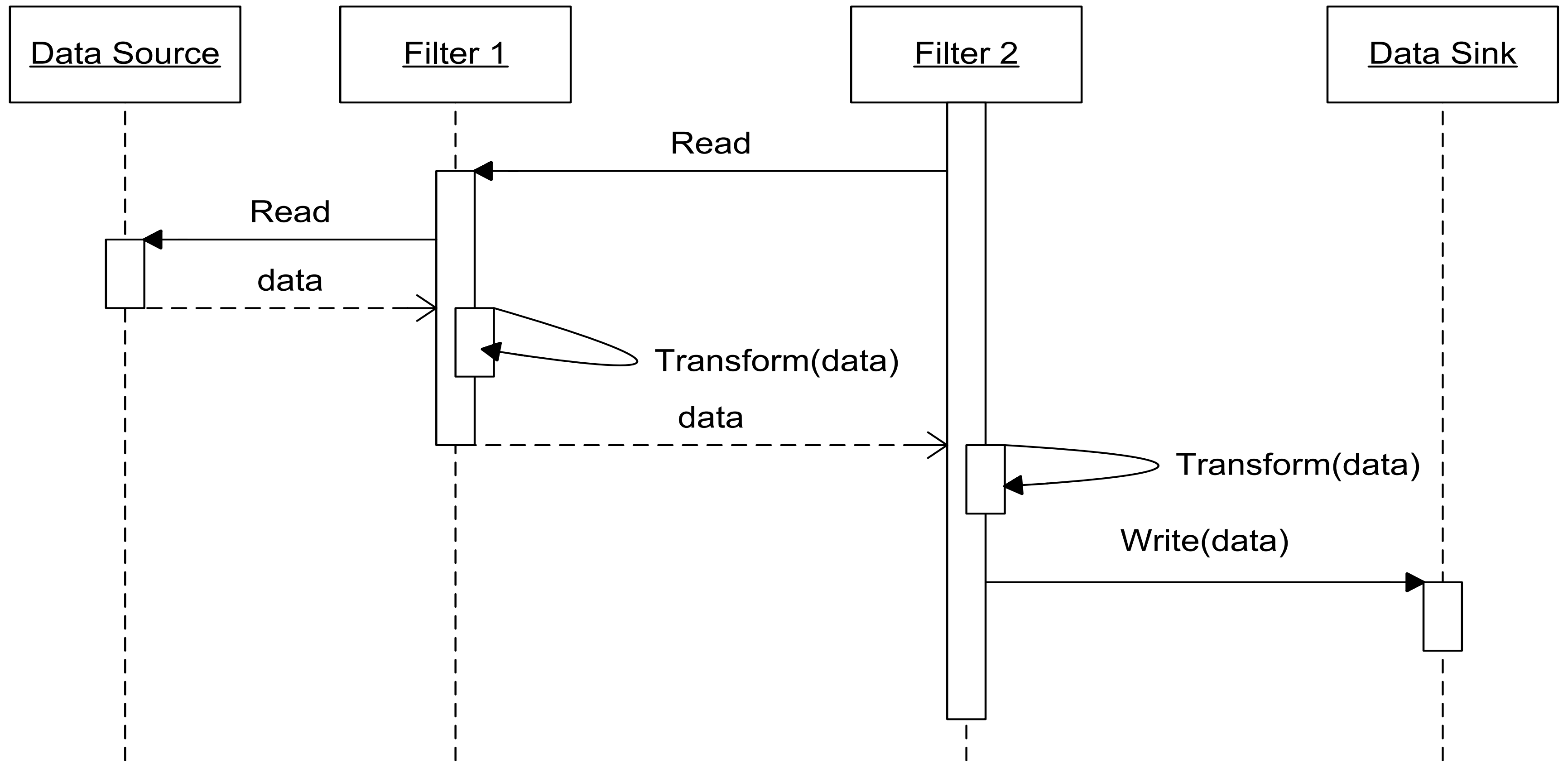
Scenario-2

- Pull pipeline
- Control flow is started by the data sink calling for data



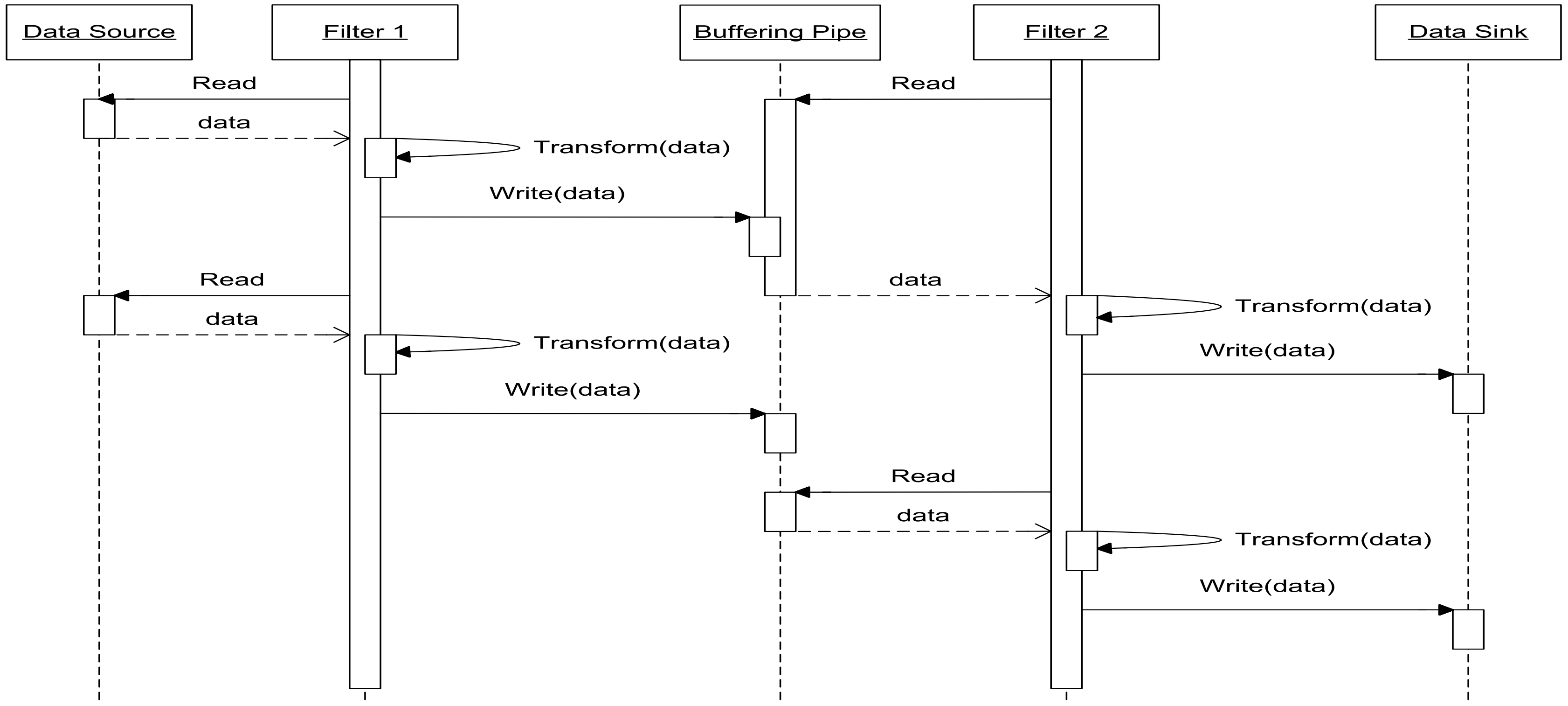
Scenario 3

– Push-pull mixed pipeline



Scenario 4- Multiprocess

- All filters actively pull, compute and push data in a loop
- Each filter runs its own thread of control
- Filters are synchronized by buffering pipe between them



Implementation

#	Steps
1	Divide the system’s task into a sequence of processing stages
2	Define the data format to be passed along each pipe
3	Decide how to implement each pipe connection
4	Design and implement the filters
5	Design the error handling
6	Set up the processing pipeline

Initial Steps

1: Divide the systems tasks into sequence of processing stages

- Each stage must depend on the output of the predecessor
- All stages conceptually connected by data flow

2: Define data format to be passed along each pipe

- Define a uniform format results in the highest flexibility because it makes recombination of filters easy
- Define the end of input marking

Design Pipe and Filter

3. Pipe

- Decision determines active or passive filter
- Using a separate pipe mechanism that synchronises adjacent active filters provide a more flexible solution

4. Filter

- Design Depends on
  - Task it must perform
  - Adjacent pipe
- Active or Passive filters
  - Active filter pulls data from a pipe
  - Passive ones get the data
- Implemented as threads or processes
- Filter reuse
  - Each filter should do one thing well
  - Can read from global or external files for flexible configuration

Final Steps

5: Design error handling

- Never neglect error handling
- No global state shared; error handling hard to address
- Strategies in case of error – depend on domain

6: Setup processing pipeline

- Use of standardised main program
- Use of user inputs or choice

Variants

- Tee and Join pipeline
  - Filters with more then one input and/or more than one output

Benefits

- No intermediate files necessary, but possible
- Filter addition, replacement, and reuse
  - Possible to hook any two filters together
- Rapid prototyping of pipelines
- Concurrent execution
- Certain analyses possible
  - Throughput, latency, deadlock

Liabilities

- Sharing state information is expensive or inflexible
- Data transformation overhead
- Error handling can be a problem
- Does not work well with interactive applications
- Lowest common denominator on data transmission determines the overall throughput

Pipe and Filter in Cloud based Service

- Most PaaS service providers (Amazon, Azure, Google) provides message oriented service orchestration
- Pipe-n-Filter is a common pattern
- Azure
  - The components having worker role are the filters
  - Pipe is the queuing service
- Amazon
  - EC2 instances are filters, communicating via SQS pipes

Thank You

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SS ZG653 (RL 10.2): Software Architecture

Blackboard Architecture

Instructor: Prof. Santonu Sarkar

Context and Problem

- A set of heterogeneous specialized modules which dynamically change their strategies as a response to unpredictable events
  - Non-deterministic strategies
- Problem
  - When there is no deterministic solutions to process raw data, and it is required to interchange algorithms processing some intermediate computation
  - Solutions to partial problems require different representation
  - No predetermined strategy is present to solve a problem (in functional decomposition sequence of activations are more hard-coded)
  - Dealing with uncertain knowledge

Forces

- A complete search of the solution space is not possible
- Different algorithms to be used for partial solutions
- One algorithm uses results of another algorithm
- Input, intermediate data, output can have different representation
- No strict sequence between algorithms, one can run them concurrently if required

Examples

- Speech recognition (HEARSAY project 1980)
- Vehicle identification and tracking
- Robot control (navigation, environment learning, reasoning, destination route planning)
- Modern machine learning algorithms for complex task (Jeopardy challenge)
- Adobe OCR text recognition
- Modern compilers tend to be more Blackboard oriented

Blackboard Pattern

- Two kinds of components
  - Central data structure — blackboard
  - Components operating on the blackboard
- System control is entirely driven by the blackboard state

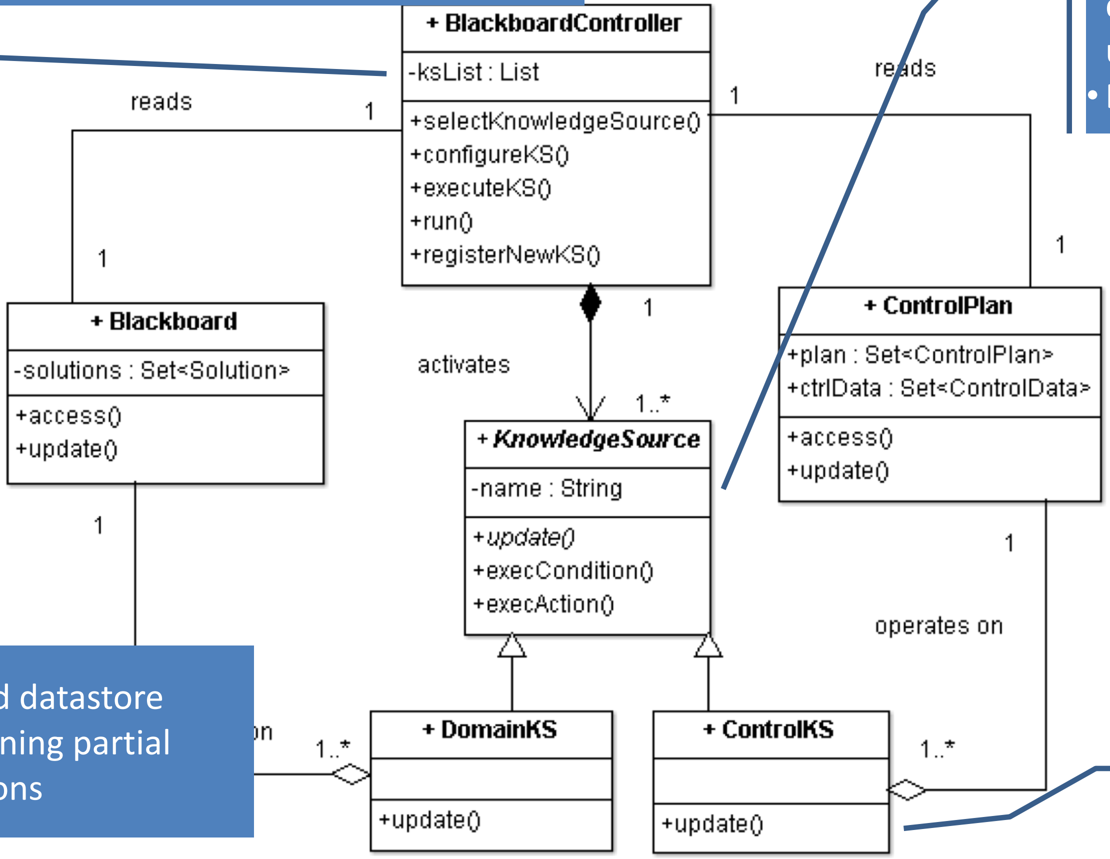
Components of Blackboard

- The blackboard is the shared data structure where solutions are built
- ⍰ The control plan encapsulates information necessary to run the system
  - It is accessed and up dated by control knowledge sources
- DomainKS are concerned with the solving of domain specific problems
- Control KS adapt the current control plan to the current situation
- The control component selects, configures and executes knowledge sources

Solution Structure

Runs in a loop
- Monitors change in blackboard
- Activates next KS
- Selection strategy may depend on ControlData

Highly specialized modules
- Each is different
- Has a set of triggering conditions
- Executable code that retrieves data from the blackboard and updates blackboard
- Don't interact with each other



Shared datastore containing partial solutions

Updates control Data
Such as:
- progress estimation,
- computation cost to execute a KS
- Control plan

Automated Robo Navigation

- Robot’�s high level goal is to visit a set of places as so on as possible
  - The successive sub�goals are
    - to decide on a sequence of places to visit�
    - to compute the best route� and
    - to navigate with a constraint of rapidity�

Benefits

Benefits

- Experimentation- try with different strategies,
- Support for modifiability- each KS is strictly decoupled
- Reuse of KS
- Fault-tolerance even when the data is noisy

Liabilities

- Difficulty in testing
- No good solution guaranteed
- Computational overhead in rejecting wrong solutions
- High development effort
- Concurrent access to blackboard must be synchronized, parallelization is difficult

Thank You