# CITS4401 Software Requirements and Design
# Software architecture

Unit Coordinator: Dr Mehwish Nasim
Based on notes and slides by:
Arran Stewart and Prof. Rachel Cardell Oliver

# Software architecture

# Summary

- What is a software architecture?
- Pipe and Filter
- Repository
- Blackboard
- Layered
- Also: Client Server; Process Control

# Motivation

> As the size of software systems increase, the algorithms and data structures of the computation no longer constitute the major design problems. When systems are constructed from many components, the organization of the overall system – the software architecture – presents a new set of design problems.
>
> — Shaw and Garlan (1994)

# Some historical background

- 1950s machine languages
  - designers recognised certain patterns of execution: procedure invocation, loops, conditionals, arithmetic expressions
- 1960s early high level languages
  - E.g. Fortran
- 1970s abstract data types
  - get the data structures right - the effort will make the development of the rest of the program easier
- 1980s software architecture
  - recognise common, useful system organisations
- 1990s design patterns
  - includes architectural as well as lower level patterns

# Software architecture focus

- gross organisation and global control structure
- protocols for communication, synchronization and data access
- assignment of functionality to design elements
- physical distribution of design elements
- composition of design elements
- scaling and performance
- selection among design alternatives

# Common framework

- All software architectures have 3 common elements:
- **Components**
  - a software architecture consists of a collection of computational components
- **Connectors**
  - the interactions between those components (e.g. procedure call, event broadcast, database queries or pipes)
- **Constraints**
  - A software architecture might have some constraints imposed on it. e.g. topological constraint of having no cycles

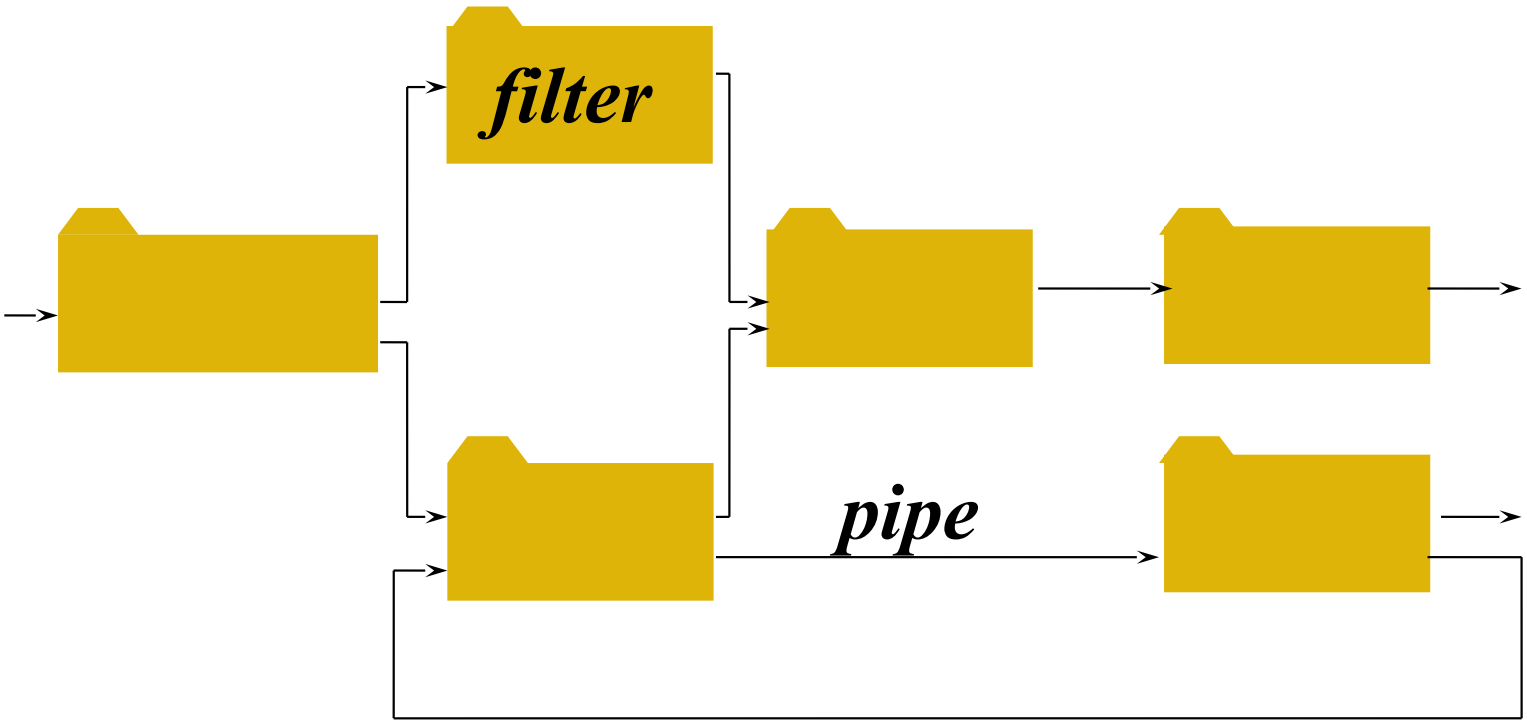
# Questions for a new software architecture

- What is the structural pattern?
- What is the underlying computational model?
- What are the essential invariants of the style?
- What are some common examples of its use?
- What are the advantages and disadvantages of using that style?
- What are some common specializations?

# Why study software architecture?

- recognise common paradigms so you can
  - understand high level relationships among systems
  - build new systems as variations on existing ones
- getting the right architecture is often crucial to the success of a design
  - the wrong architecture can lead to disastrous results
- enables principled choices among design alternatives
- architectural system representation is often essential to analysis and description of high level properties of a complex system

# Pipe and Filter Architecture structural pattern



# Computational Model for Pipe and Filter Architecture

- each component (filter)
  - reads streams of data on its inputs
  - applies a local transformation
  - produces streams of data on its outputs
- computation is incremental
  - output begins before all inputs are consumed
- connecters (pipes)
  - transmit outputs of one filter to inputs of another

# Essential Invariants for Pipe and Filter Architecture

- filters must be independent entities
  - they do not share state with other filters
- filters do not know the identity of other
  - upstream and downstream filters
- filter specifications can
  - restrict what appears on the input pipes or
  - make guarantees about what appears on the output pipes

# Pipe and Filter major specializations

- Pipeline
  - topology restricted to linear sequences of filters i.e. no loops
- Bounded Pipes
  - restrict the amount of data that can reside on a pipe
- Typed Pipes
  - data passed between 2 filters must have a well defined type

# Pipe and Filter Architecture Examples

- Unix shell pipes
  - cat myfile.txt yourfile.txt | sort | more
- Compilers
  - lexical analysis, parsing, semantic analysis, code generation
- also used for
  - image and signal processing
  - functional programming
  - distributed systems

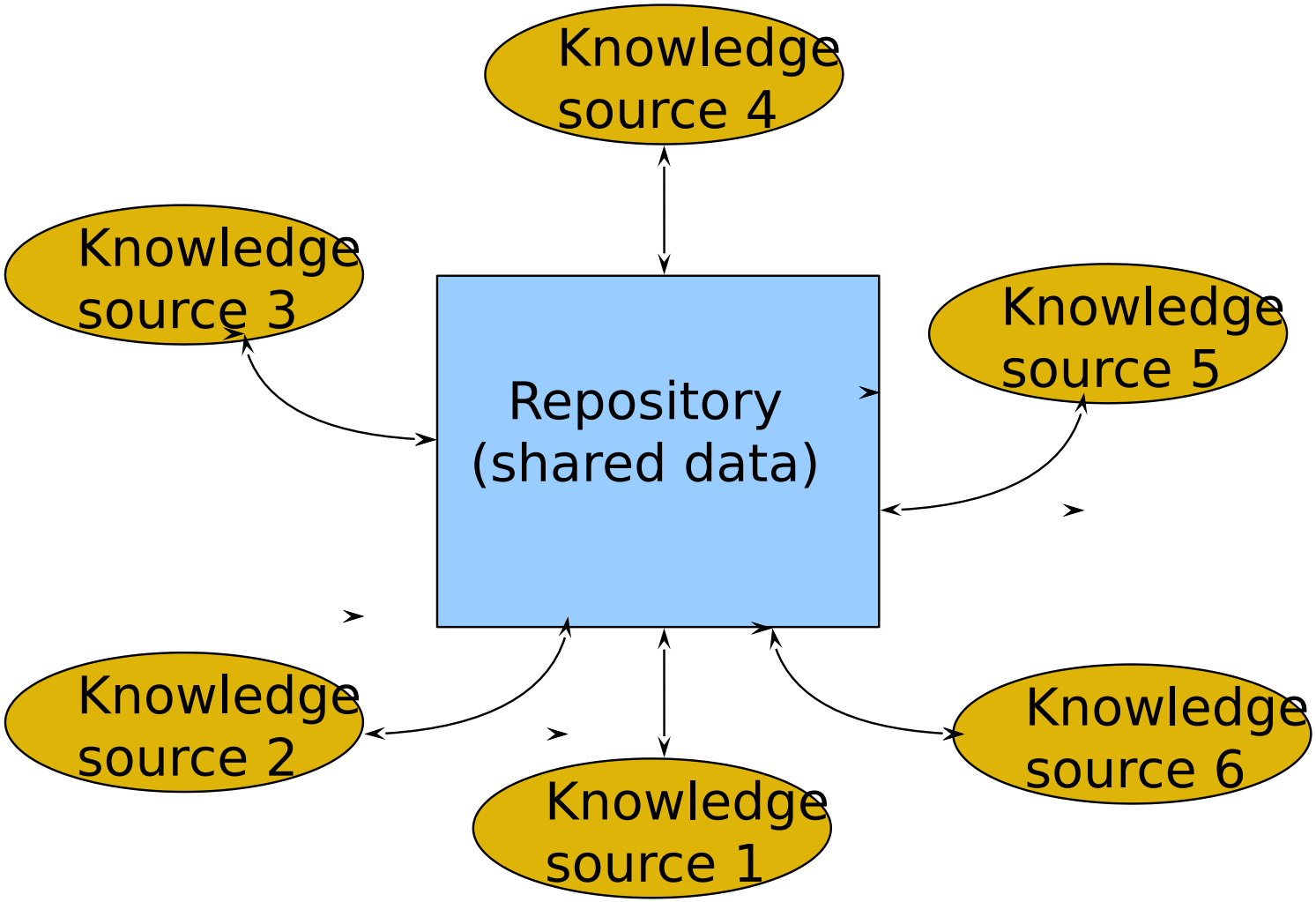
# Pipe and Filter Advantages

- understanding
  - overall input/output is a simple composition of the behaviours of individual filters
- reuse
  - any two filters can be hooked together provided they agree on the data transmitted
- maintenance and enhancement
  - new filters can be added, old filters replaced
- good for analysis
  - e.g. throughput and deadlock analysis
- naturally supports concurrency
  - filters can be executed in parallel with other filters

# Pipe and Filter Disadvantages

- May force batch organisation of processing
  - each filter must provide a complete transformation of input data to output data
- Not good at handling interactive applications
  - E.g. incremental display updates
- May have to maintain correspondences between several separate but related streams
- filters have extra work to parse and unparse the data
- All this leads to loss of overall performance and increased complexity of each filter

# Repository architecture



# Repository Architecture Components

- Subsystems access and modify data from a single data structure
- Subsystems are loosely coupled (interact only through the repository)
- Two distinct types of component
  - central data structure represents current state
  - independent components operate on the data store

# Repository Interactions

- Control flow is dictated by central repository (triggers) or by the subsystems (locks, synchronization primitives)
- Blackboard architecture interactions
  - current state of the central data structure is the main trigger
- Traditional database interactions
  - input stream of transactions triggers selection of processes to execute

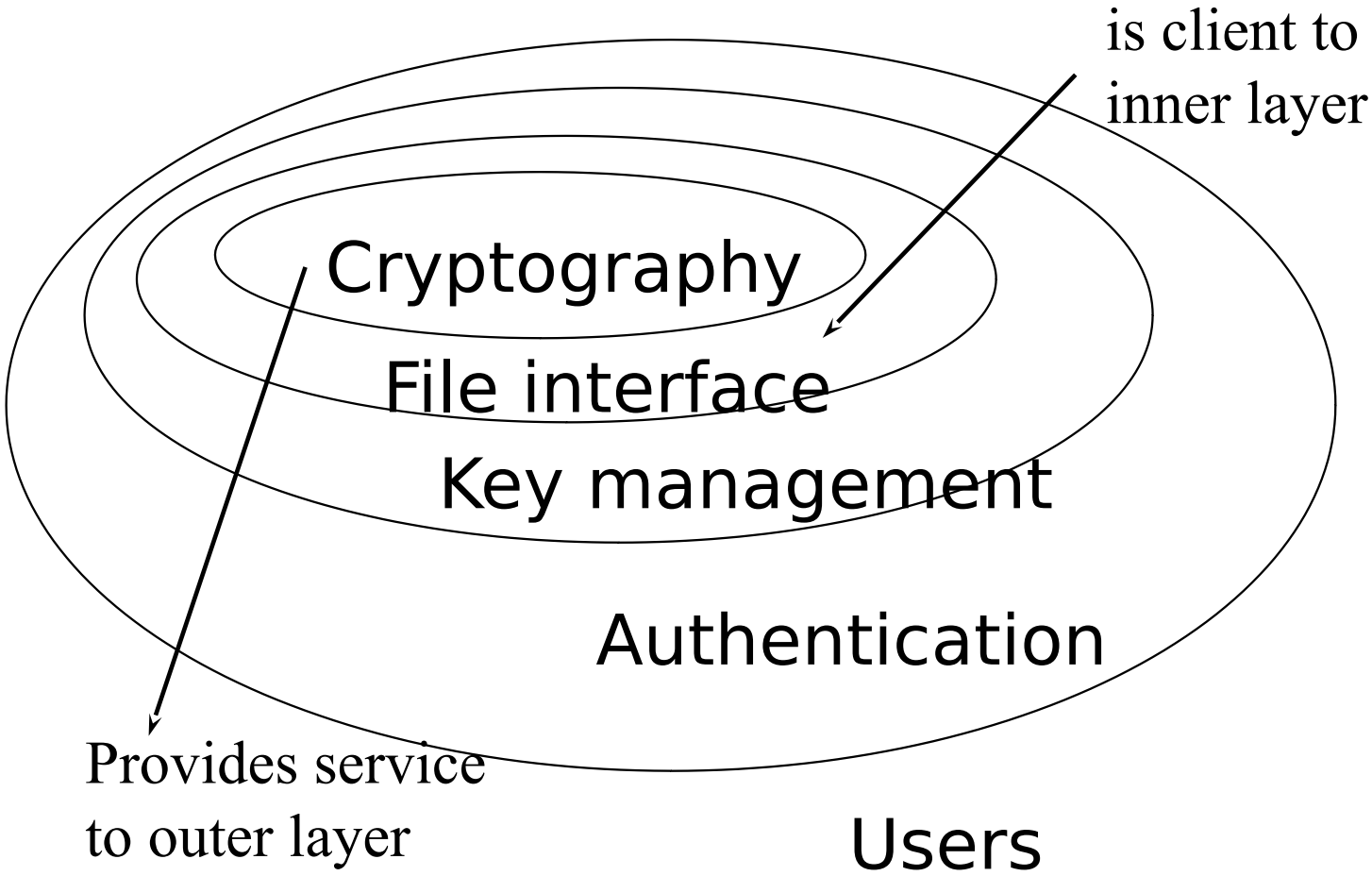
# Blackboard Repositories

- Knowledge sources
  - separate, independent parcels of application-dependent knowledge
- Blackboard data structure
  - problem-solving state data organised in an application-dependent hierarchy
- Control
  - driven by the state of the blackboard

# Examples of Repository systems

- complex interpretations of signal processing
  - E.g. speech and pattern recognition
- shared access to data with loosely coupled agents
- some programming environments
  - collections of tools and shared repository of programs and program fragments
- database management systems

# Layering architecture - structure



# Layered model

- hierarchical organisation: each layer
  - provides a service to the layer above
  - acts as a client to the layer below
- layers may either be
  - partially opaque (some interaction allowed between non-adjacent layers)
  - opaque: inner layers are hidden from all except the adjacent outer layer

# Example

- ISO reference model for Open Systems Interconnection (network protocols)
  - application layer
  - presentation
  - session
  - transport
  - network
  - data link
  - physical

# Layered System Advantages

- Design based on increasing levels of abstraction
  - complex problem becomes sequence of incremental steps
- Supports enhancement
  - changes to one layer affect at most two other layers
- Supports reuse
  - different implementations of the same layer can be used interchangeably provided they support the same interfaces

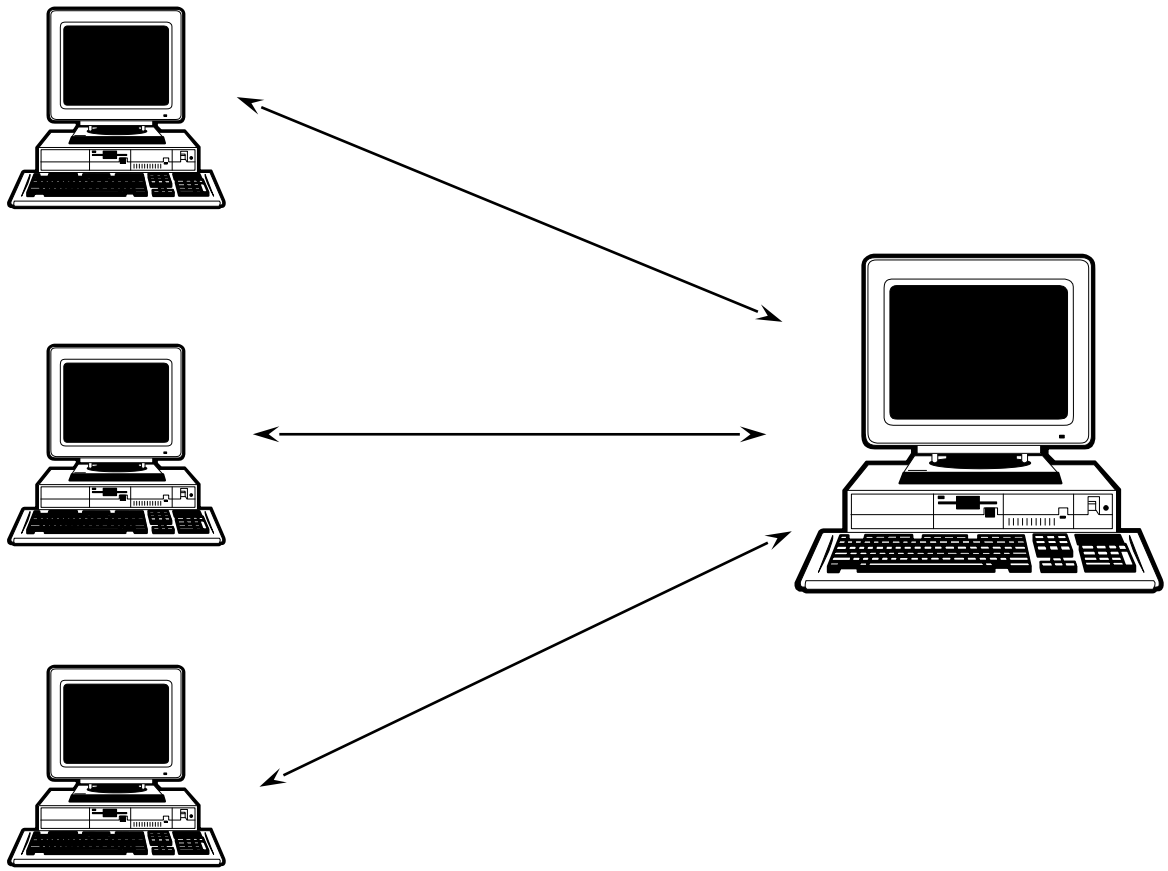
# Layered System Disadvantages

- Not all systems are easily structured in layers
- Performance considerations
  - may need close coupling between logically high-level functions and their low-level implementations
- Can be difficult to find right level of abstraction
  - E.g. protocols which bridge several OSI layers

# More architectures

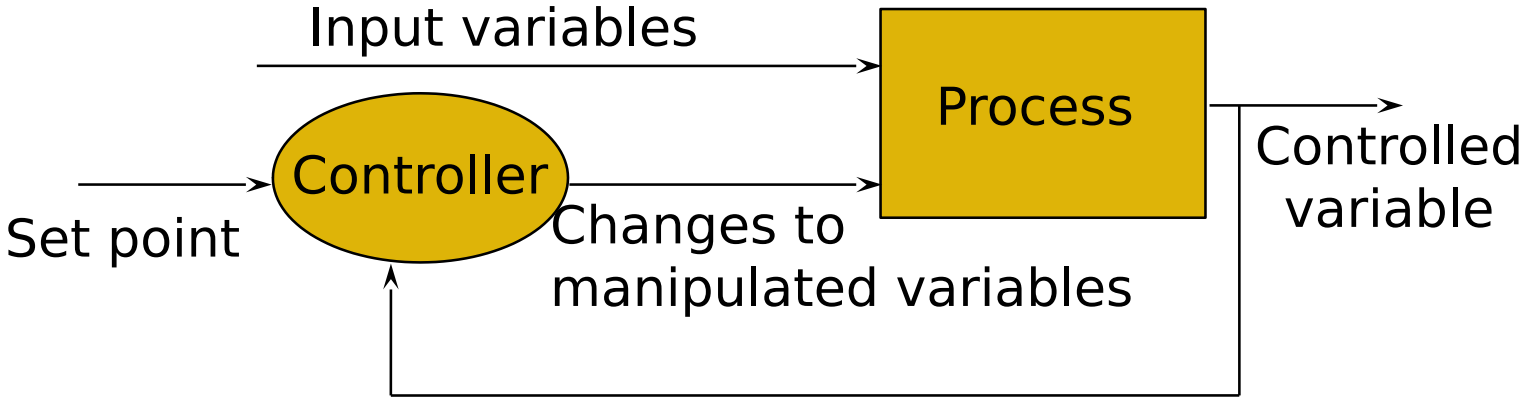
Some other software architectures . . .

# Client and Server architecture

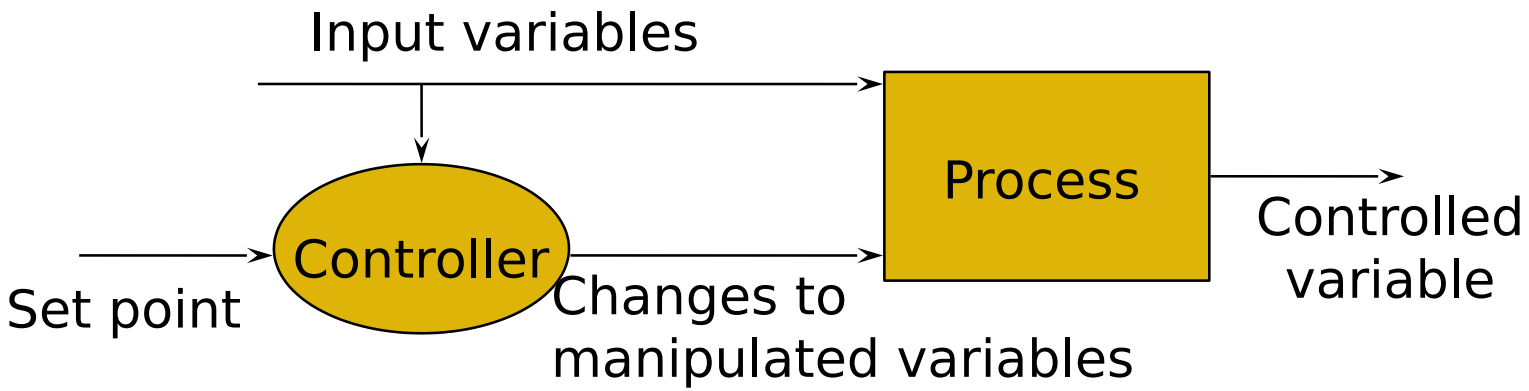


# Process Control architecture

FEEDBACK LOOP:




FEEDFORWARD LOOP:



# Summary

- What is a software architecture?
- Pipe and Filter
- Repository
- Blackboard
- Layered
- Also: Client Server; Process Control

# Recommended reading

- Bruegge and Dutoit, 2010:
  - Section 6.3.5 architectural styles
  - Section 7.4 system design activities
- Garlan, David, and Mary Shaw. "An introduction to software architecture." *Advances in software engineering and knowledge engineering*. 1993. 1-39.
  See LMS attached files.