Software Architecture

ECE-355 Tutorial

Ali Razavi

arazavi@swen.uwaterloo.ca

Introduction

Problems of designing large and complex Software Systems are different from writing Programs.

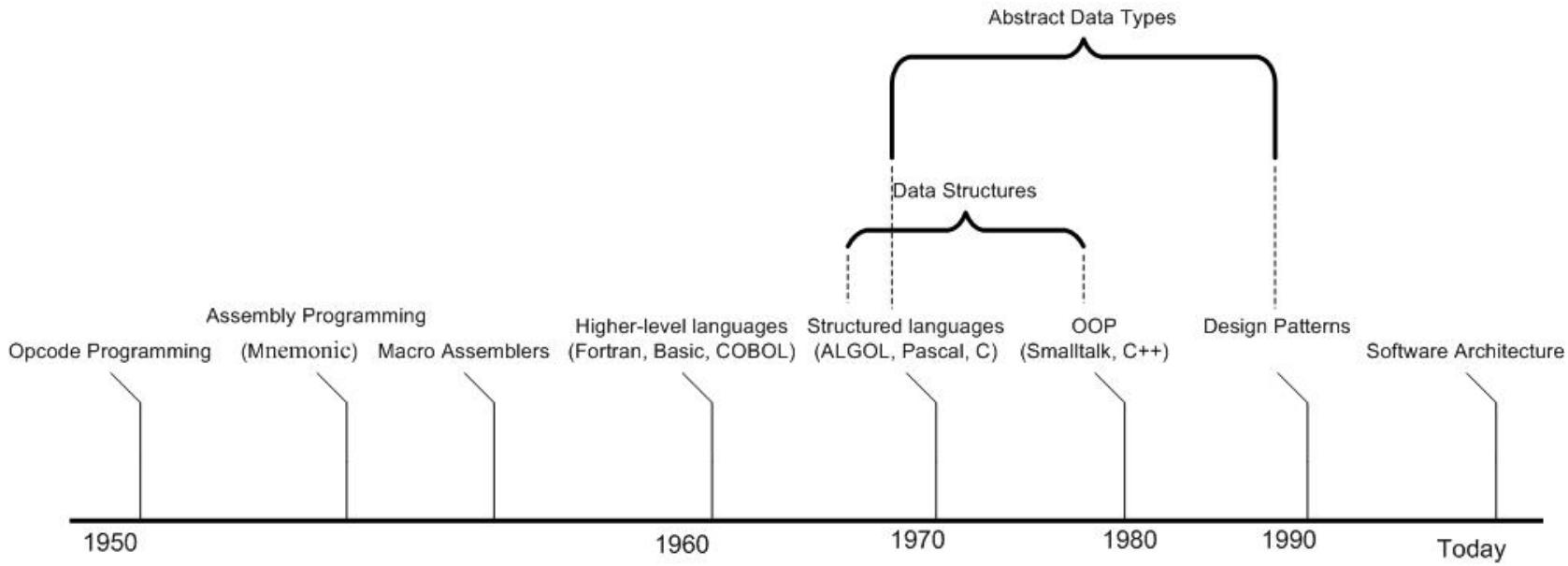
Programs :	Software Systems :
data-structures problems	Architecture
algorithms	Design structure problems
coding style	Programming problems
…	

- Architecture problems : control structures, communication protocols, decomposition to sub-systems
- Design structure problems : Applying appropriate design patterns, avoiding Anti-Patterns, …

Why studying architectural software design is useful ?

- To Build new systems as variation of proven successful architecture of existing system
- Helps in recognizing the best architecture for the new designs and having principled choices among alternatives.
- Helps in analyzing properties of complex systems

Evolution of "Programs" to "Software Architecture"

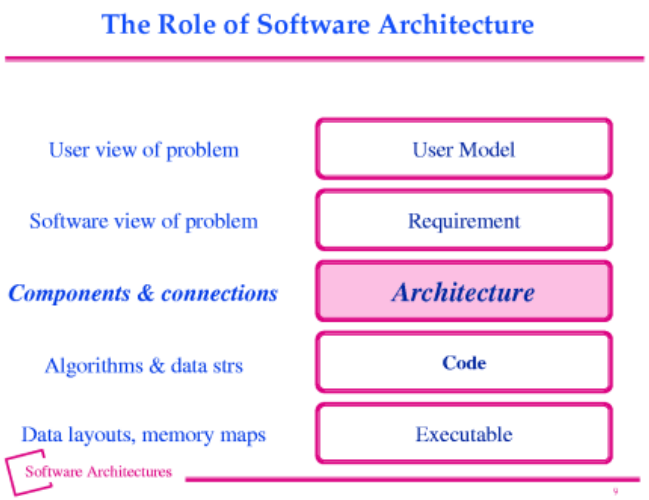


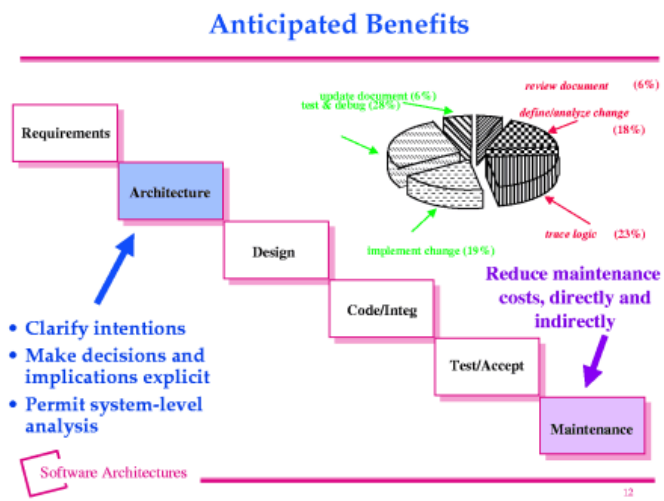
- 1960s intuition of programmers : right data structures can lead to easier software development.
- ADT allows some parts of systems to be developed from the vocabulary of

Data types instead of programming languages.

Software Architecture

- Defines system in terms of components and interactions among components (connectors)
- Shows correspondence between requirements and elements of the constructed system
- Architectural Style defines a <u>family</u> of such systems in terms of **<u>pattern of structural organization</u>**



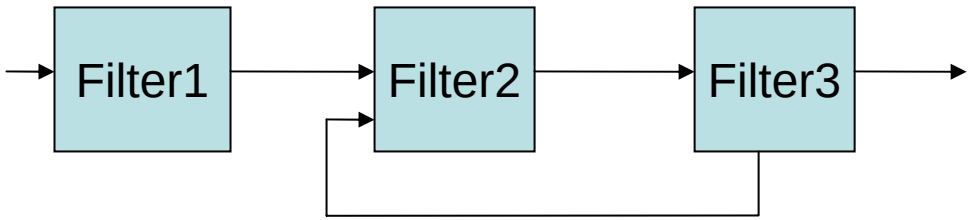


Architectural Styles and Patterns

- Pipe and Filters
- Layered Systems
- Repositories
- Model-View-Controller (MVC)
- Client Server
- Event Driven, Implicit Invocation

Pipe and Filters

- Basic Structure:



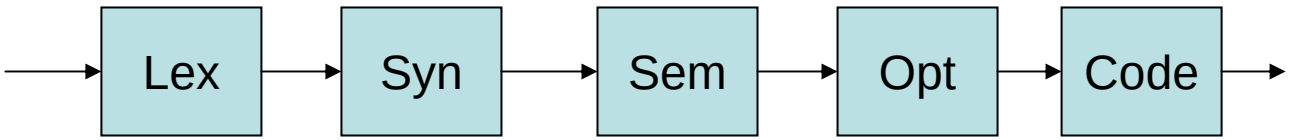
Each component acts like a signal processing **filter** (**transforms** its input stream to an output stream based on set of rules and in standard order). Filters are connected together with pipes
Filters are independent entities (they don't share state with other filters).
Filters do not know the identity of their up and downstream filters.
Common specializations : Pipelines, Bounded pipes, typed pipes, batch sequential systems
Examples : Unix pipes, Compilers, Speech compression systems

Pipe and Filters contd.

- **Advantages** : neat solution, support of reuse, easy maintenance and enhancement of system, analyzable, support of concurrency
- **Drawbacks** **:** not intrinsically interactive, loss of performance, complexity, difficult to handle two related streams

Pipe and Filter example:

- Traditional Compilers: a batch system



Lex: Lexical Analyzer

Syn: Syntax Checker

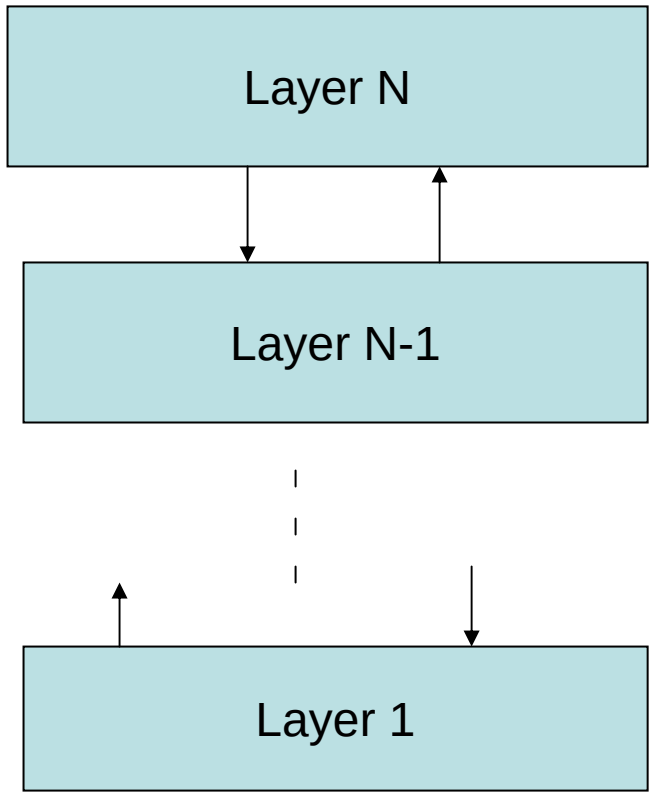
Sem: Semantic Analyzer

Opt: Optimizer

Code: Code Generator

Layered Systems

Style 4 : Layered System
Hierarchical organization, connectors are protocols defined for communication between layers
Examples : Communication protocols, Operating systems
Advantages: high level of abstraction in design (allows problem partitioning), support enhancements, support reuse (example X window system)
Drawbacks: defining layers for some systems is not trivial, Implementation can differ vastly from the model, finding right level of abstraction can be difficult (example : PBX)
Layers vs. Pipelines : two way communications, closer interaction, sometimes considering the identity of data-flow



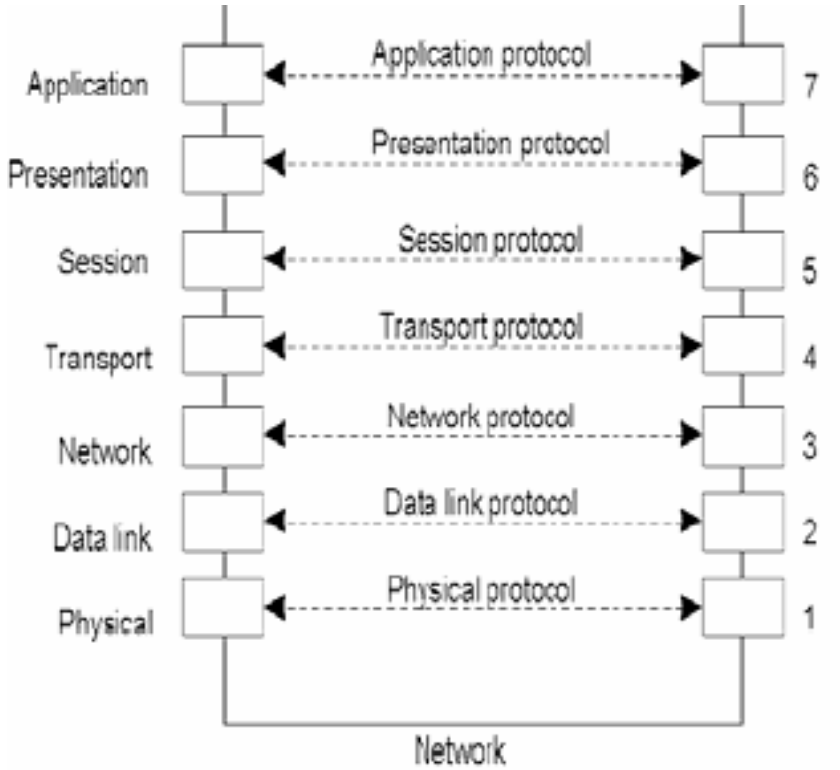
Layered Style example

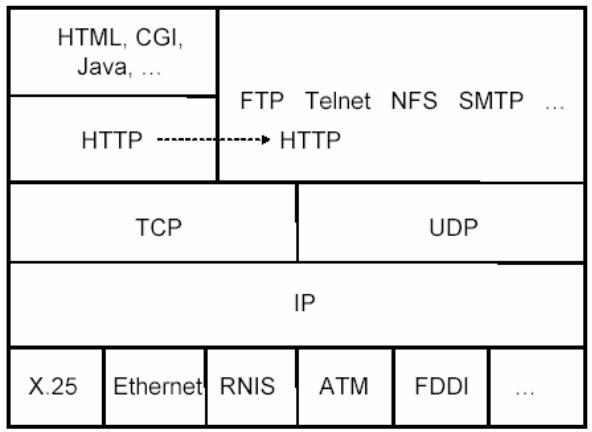
- Operating Systems:
  Mac OS X
- Aqua – User Interface
- Carbon – Procedural API
- Cocoa – Object Oriented API
- Quartz – PDF based graphics system
- Darwin – Unix Based Kernel

Mac OS X system architecture

Layered Systems example

- OSI and TCP/IP Protocol

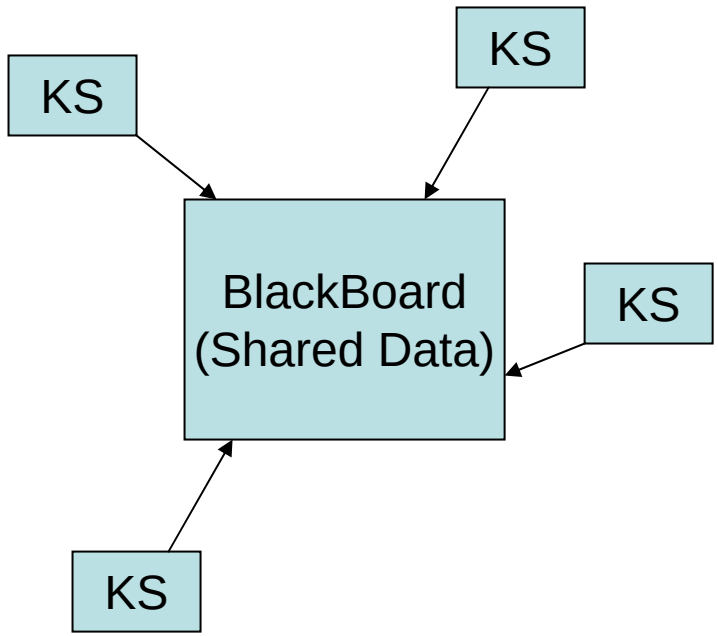




Repositories

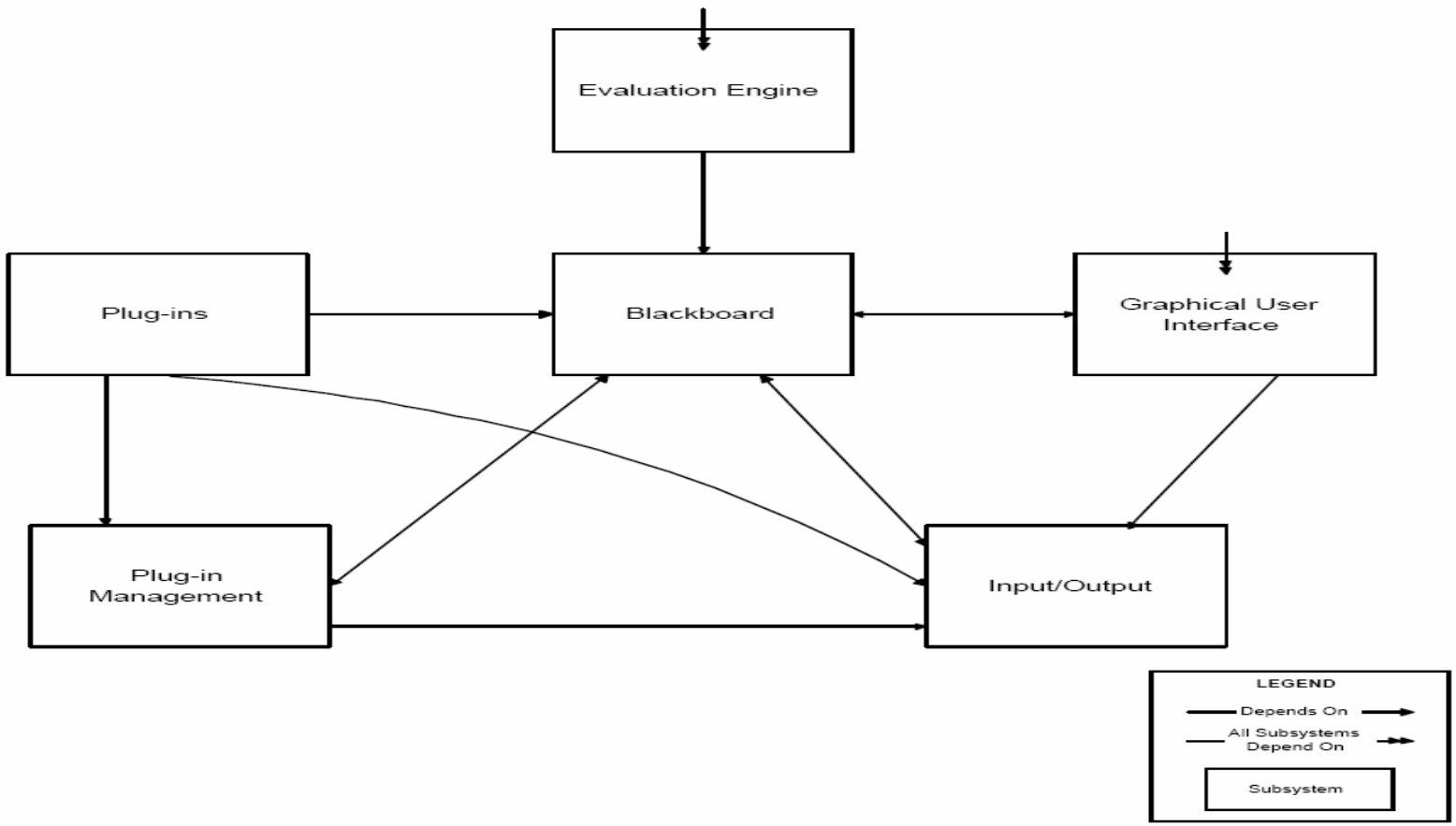
Style 5: Repositories

- Main elements : Central data structures, knowledge sources
- Processes can be triggered by transactions (Data-Base) or by the state of repository(Blackboard)
- Applicable in Speech and Pattern recognition systems.
- KS are usually activated with implicit invocations



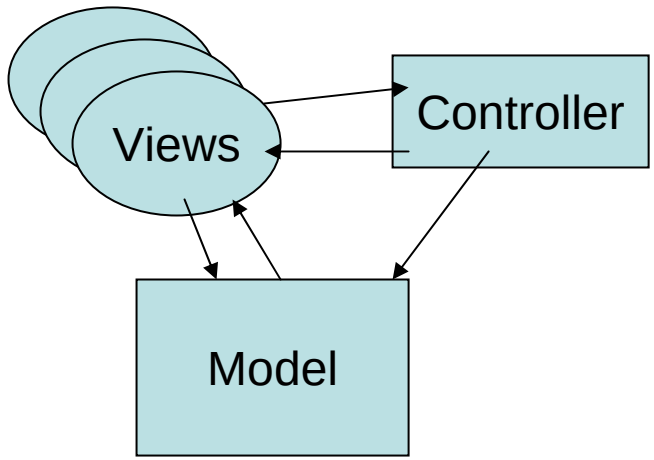
Repository Example

Gnumeric Spreadsheet:



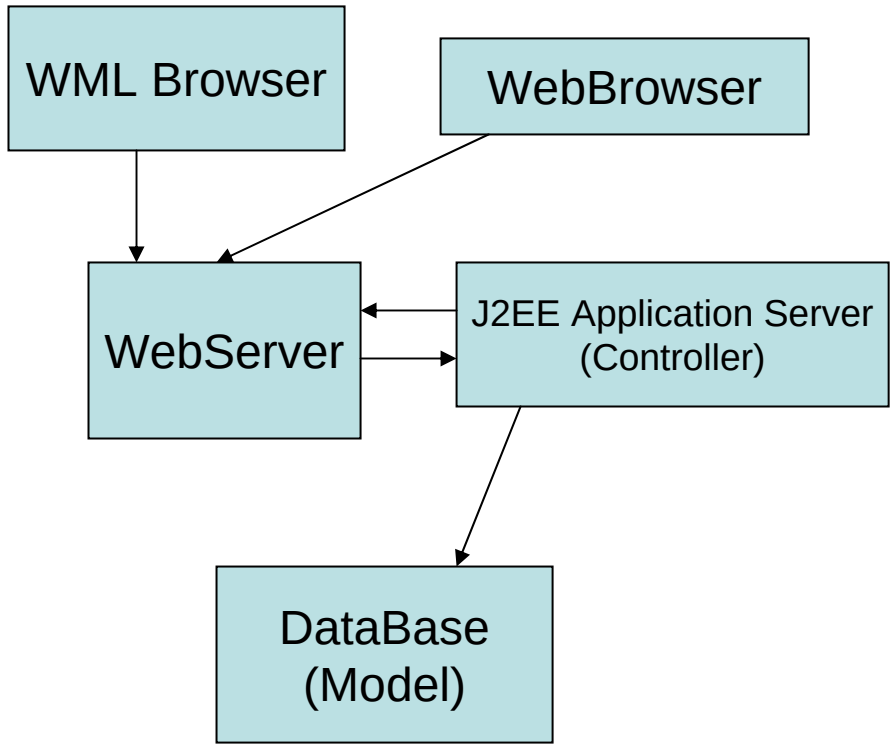
Model View Controller

- For interactive systems:
  - Model : core data
  - Controller: logic of the program
  - View : representation of data



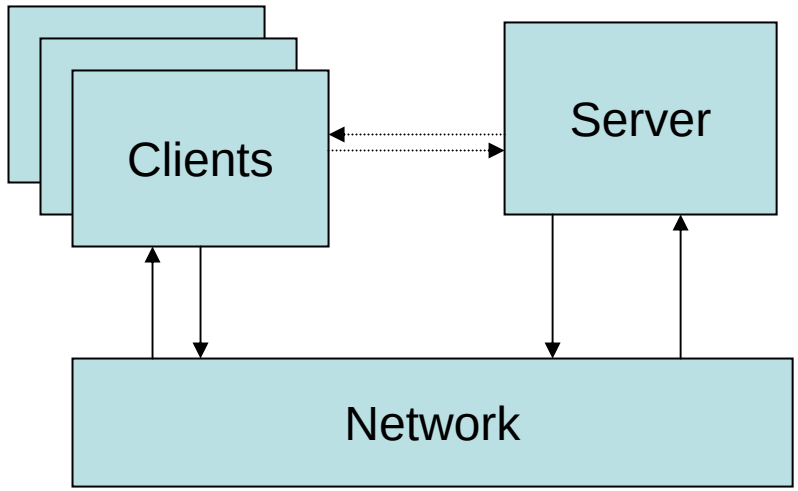
MVC example

- Web Services



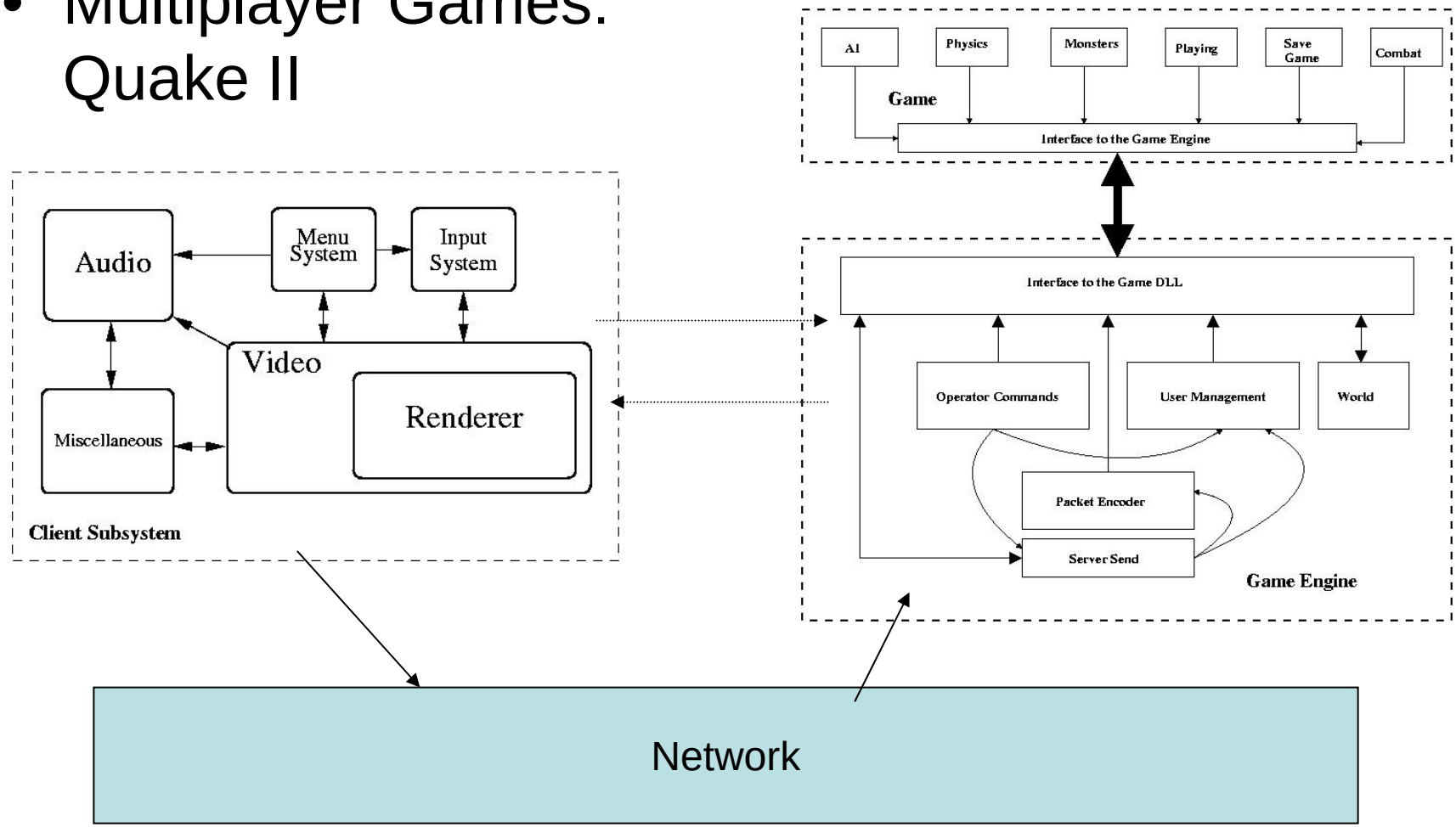
Client Server

- Client and Server
- Communicate via a Network or communication subsystem
- Message based
- Request/Reply



Client Server Example

- Multiplayer Games: Quake II



Implicit Invocation

- Components register their Event Handler procedures, and these will be called implicitly just in case.
- Components in this style are modules with collection of procedures and events in their interface. Connectors are procedure calls (both implicit and explicit)
- *The announcers of events are not aware of the components being affected by those events (No assumption on order and quality of processing for an event)*
- Examples : Win32 GUI (**Don't call me, I will call you**).
- Advantages: Strong support of reuse, easy system evolution
- Drawbacks: preemptive control by system manager, exchange of data, proving correctness is difficult