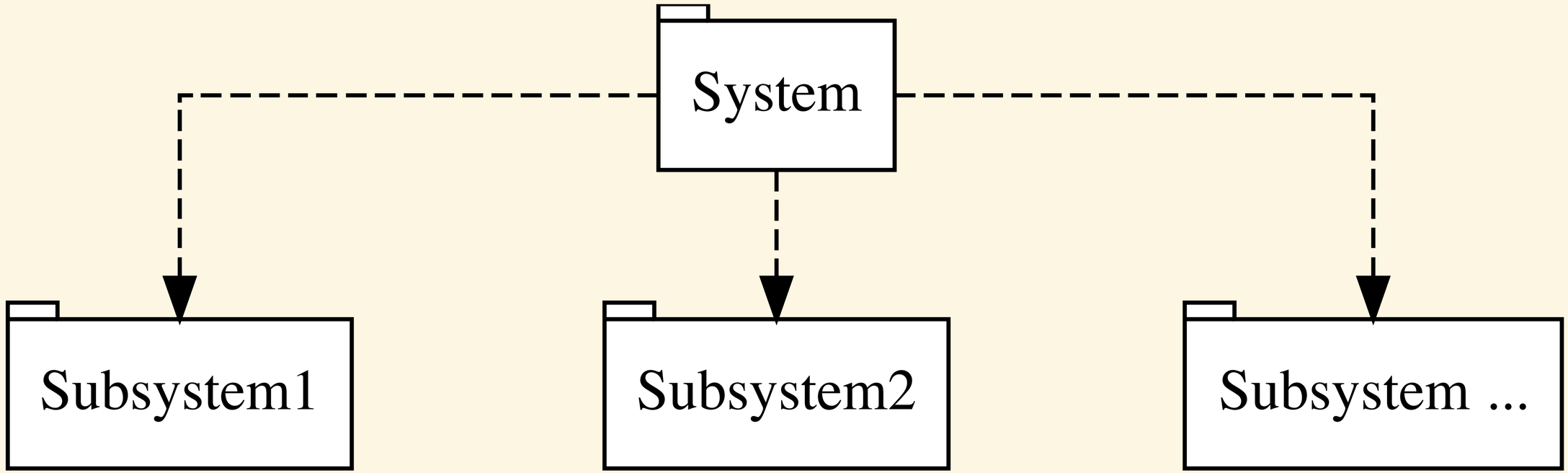
Object-Oriented Programming

Software Architecture

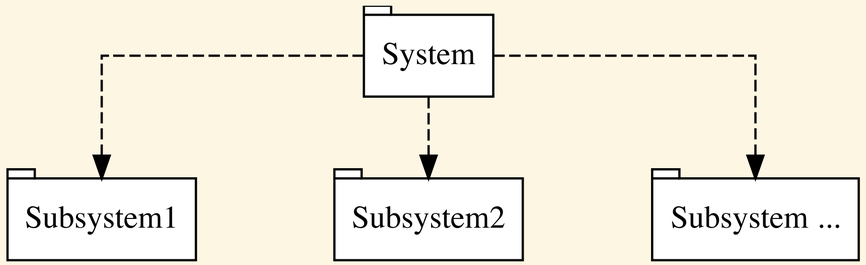
Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

System/Subsystem Design



System/Subsystem Design



- *subsystem* provides a set of *services* to the system

  A set of related operations that share a common purpose

- *subsystem interface* a set of *services* available to other systems

  Application Programmer Interface (API) includes the names of operations, parameters/types, and return types

- System design focuses on defining services

Architectural Style

An architectural style defines a family of systems in terms of a pattern of structural organization

- *components*, e.g., client, server, DB
- *connectors*, e.g., procedure call, pipe, event broadcast
- Consists of:

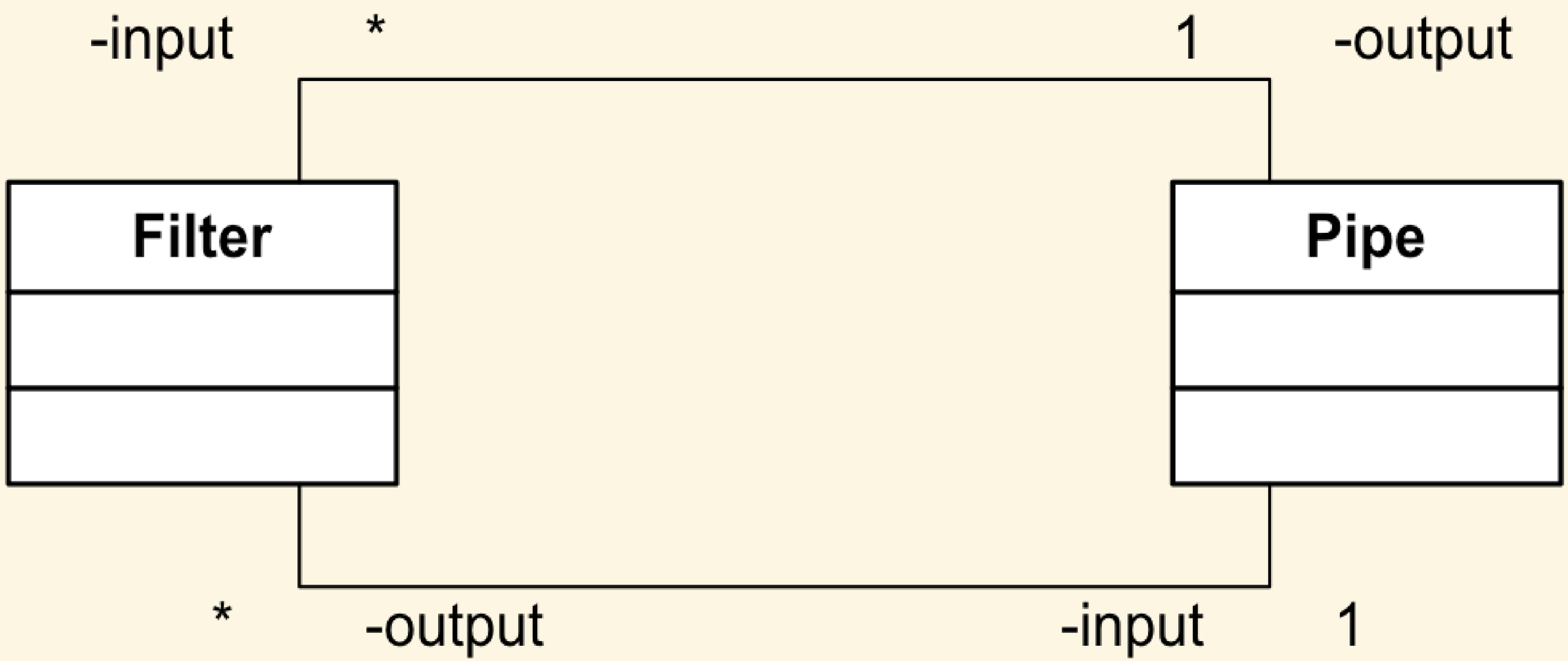
  Vocabulary of *components* and *connectors*

  Constraints on how they are combined

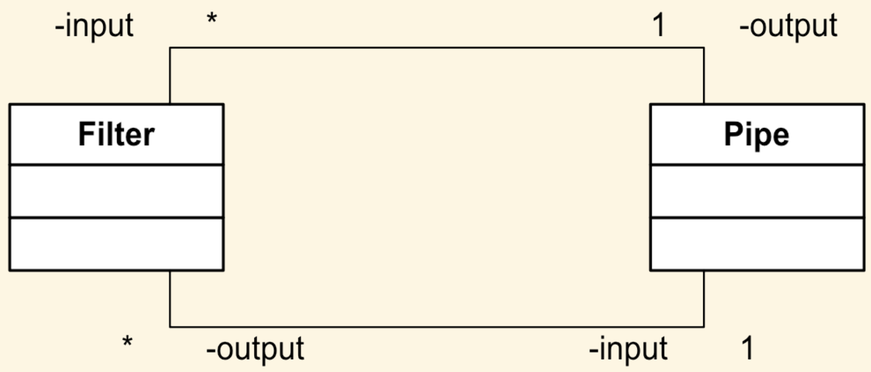
Common Architectural Styles

- Dataflow Systems: *Pipe and Filter*, *Batch Sequential*
- Virtual Machines: *Rule-based Systems*, *Interpreters*
- Repository: *Databases*, *Hypertext Systems*, *Blackboards*
- Independent Components: *Peer-to-Peer*, *Client Server*, *Model/View/Controller*, *Event Driven*
- Call and Return Systems: *Main Program and Subroutines*, *Layered Systems*, *Object-Oriented Systems*

Pipe and Filter Architecture



Pipe and Filter Architecture

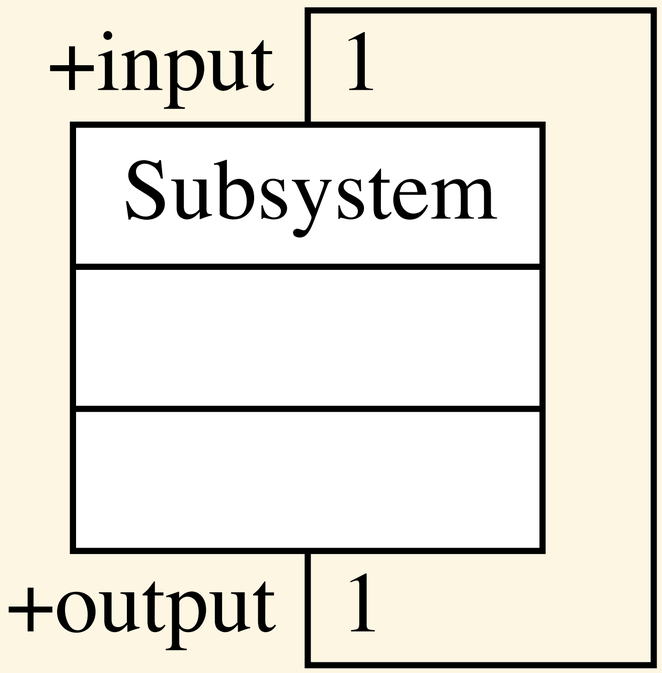


- Subsystems are called *filters* and associations between the filters are called *pipes*
- Filters only know the content and format of data being received and produced; nothing about the other filters in the system
- Filters execute concurrently with synchronization via pipes
- Very reconfigurable
- Transformational systems

Pipe and Filter Example

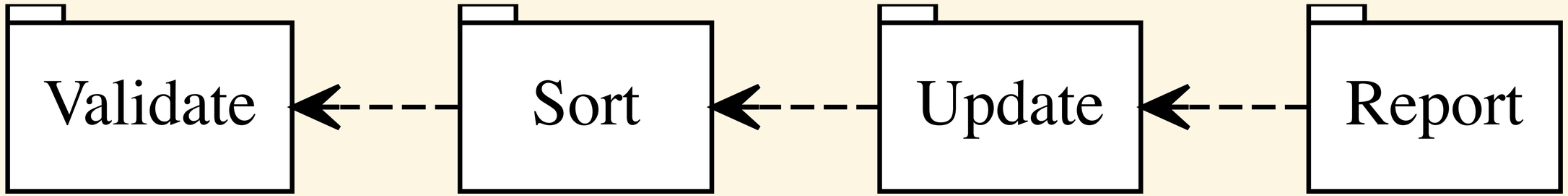
Batch Sequential Architecture

Batch Sequential Architecture

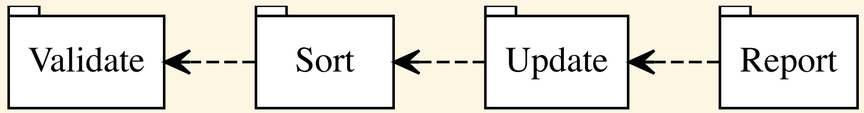


- A small number of large stand-alone subsystems
- Must be executed in a fixed sequential ordering (batch)
- Typically work on large flat files, transforming the file into a new format or a new ordering so the next subsystem can work on the data
- The shared, specific file tightly couples the subsystems
- No real-time feedback, no concurrency

Batch Sequential Example



Batch Sequential Example



- *Validate* must complete and output before *Sort*
- *Sort* must complete and output before *Update*
- *Update* must complete and output before *Report*
- Each subsystem may have a different output format
- Each subsystem is developed based on the previous subsystem's output format
- No concurrency

Layered Architecture

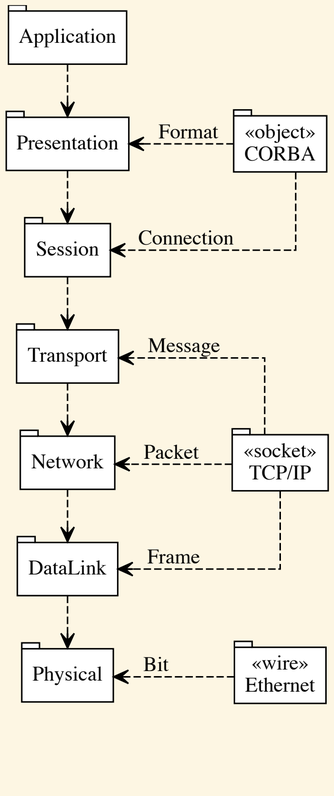
Hierarchical decomposition of a system into subsystems (layers), with each providing a higher level of services provided from lower-level subsystems

- E.g., Presentation Layer

- E.g., Business Layer

- E.g., Data Layer

Closed Architecture: OSI Network Model

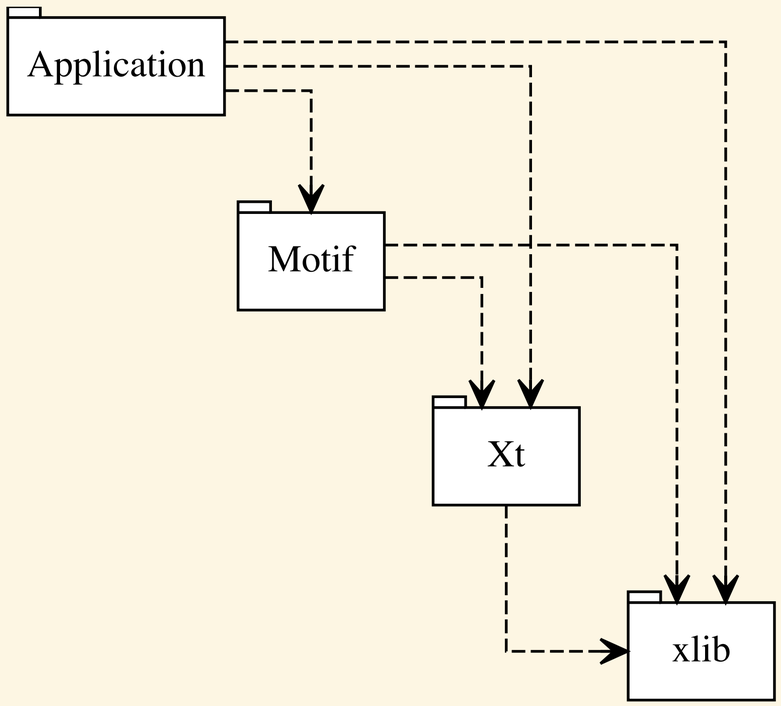


- *closed architecture*

Each layer can only depend on the layer immediately below

- For *data*, it must flow from bottom to top, stopping at every layer

Open Architecture: Motif Library

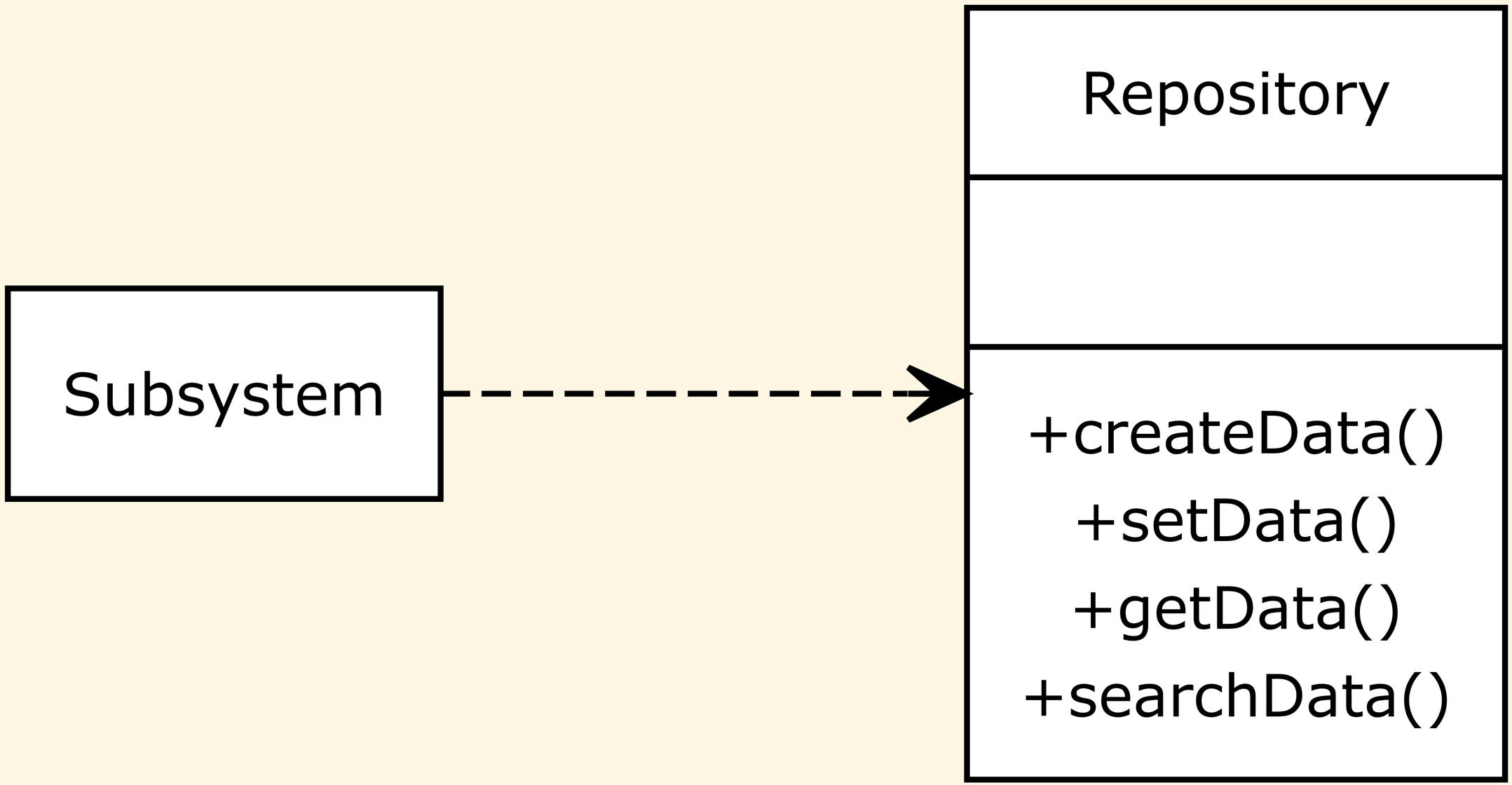


- *open architecture*

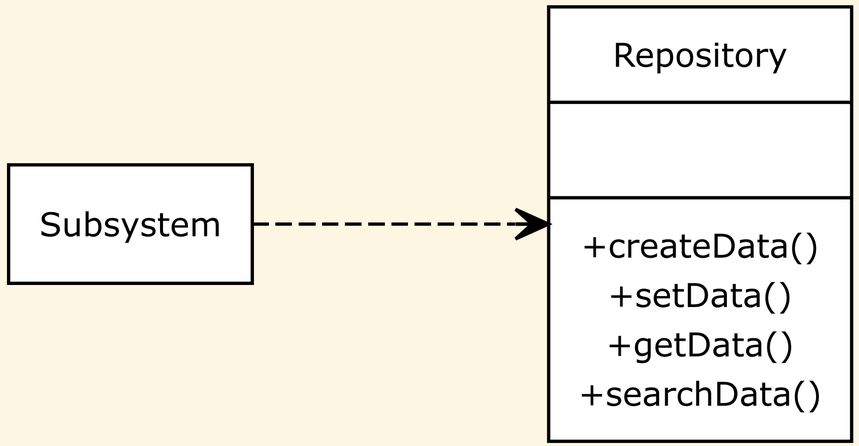
Each layer can access any layer below

- Functionality and can be accessed from any layer below

Repository Architecture



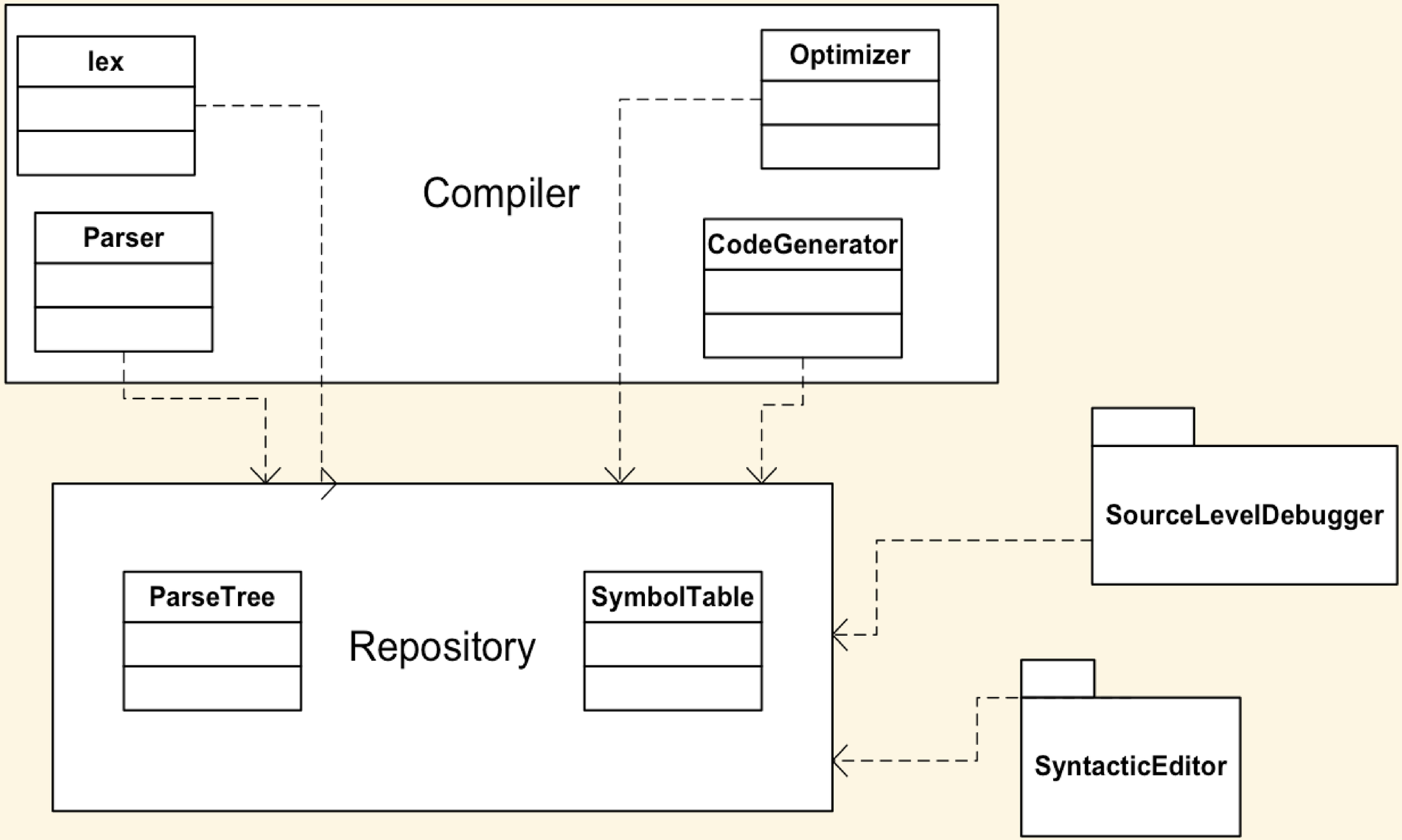
Repository Architecture



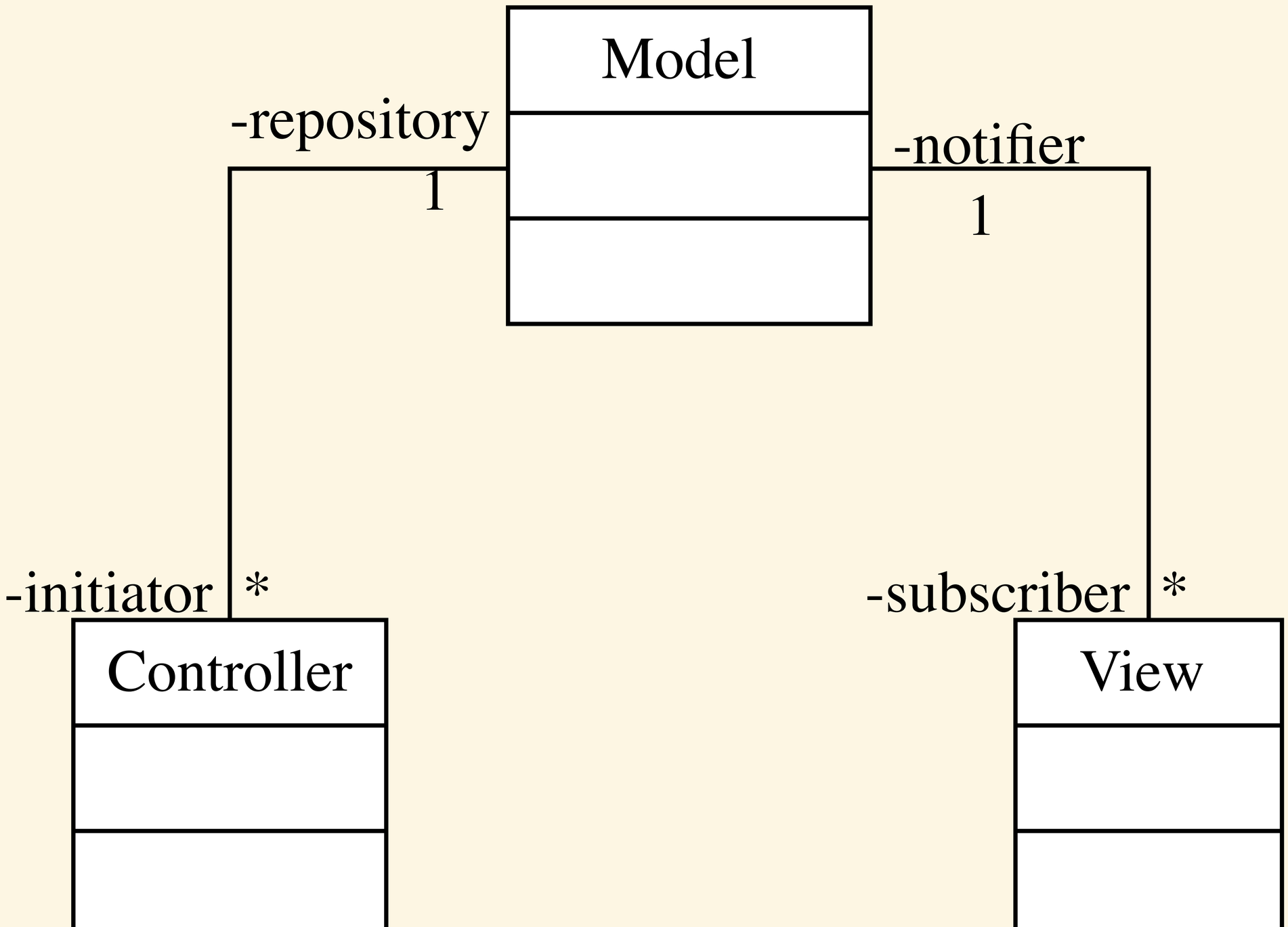
- Subsystems are independent and interact by a central repository

- Examples: Payroll or banking system, Modern IDE/Compiler, Blackboard

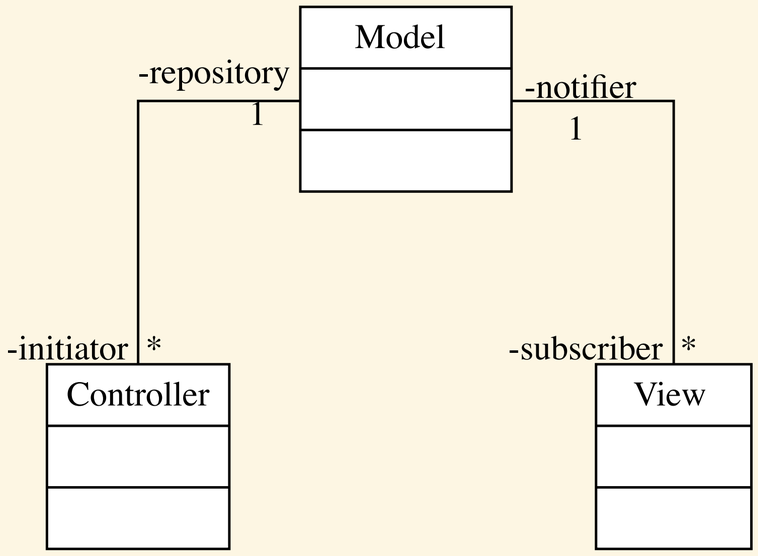
Repository: IDE



Model/View/Controller



Model/View/Controller



- Subsystems

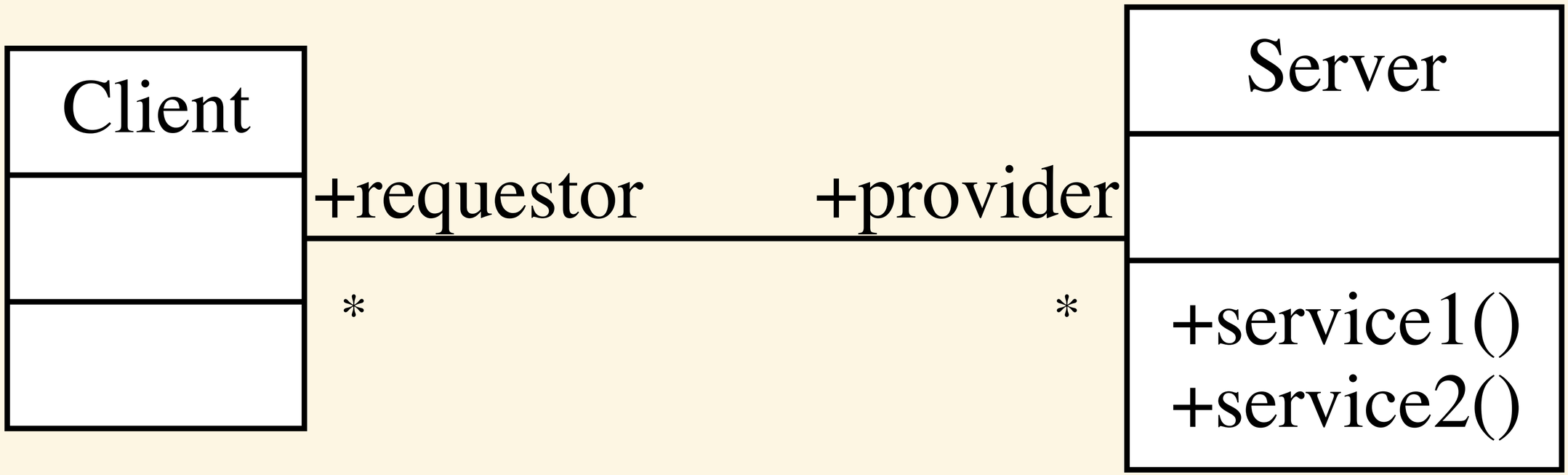
  model subsystems are responsible for maintaining domain knowledge

  view subsystems are for displaying knowledge to the user

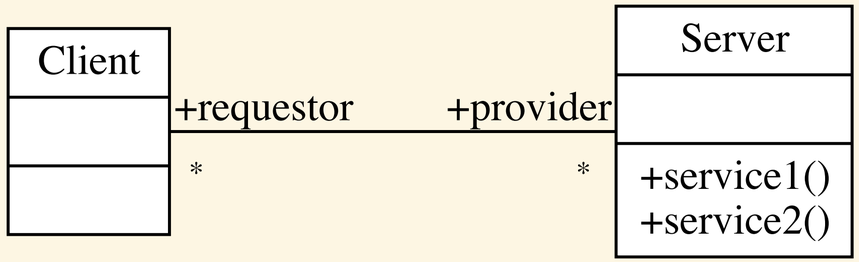
  controller subsystems manage the interactions with the user

- Model subsystems do not depend on view or controllers.
- Changes in model state propagate via a subscribe-notify protocol
- Examples: File system, database

Client/Server Architecture



Client/Server Architecture



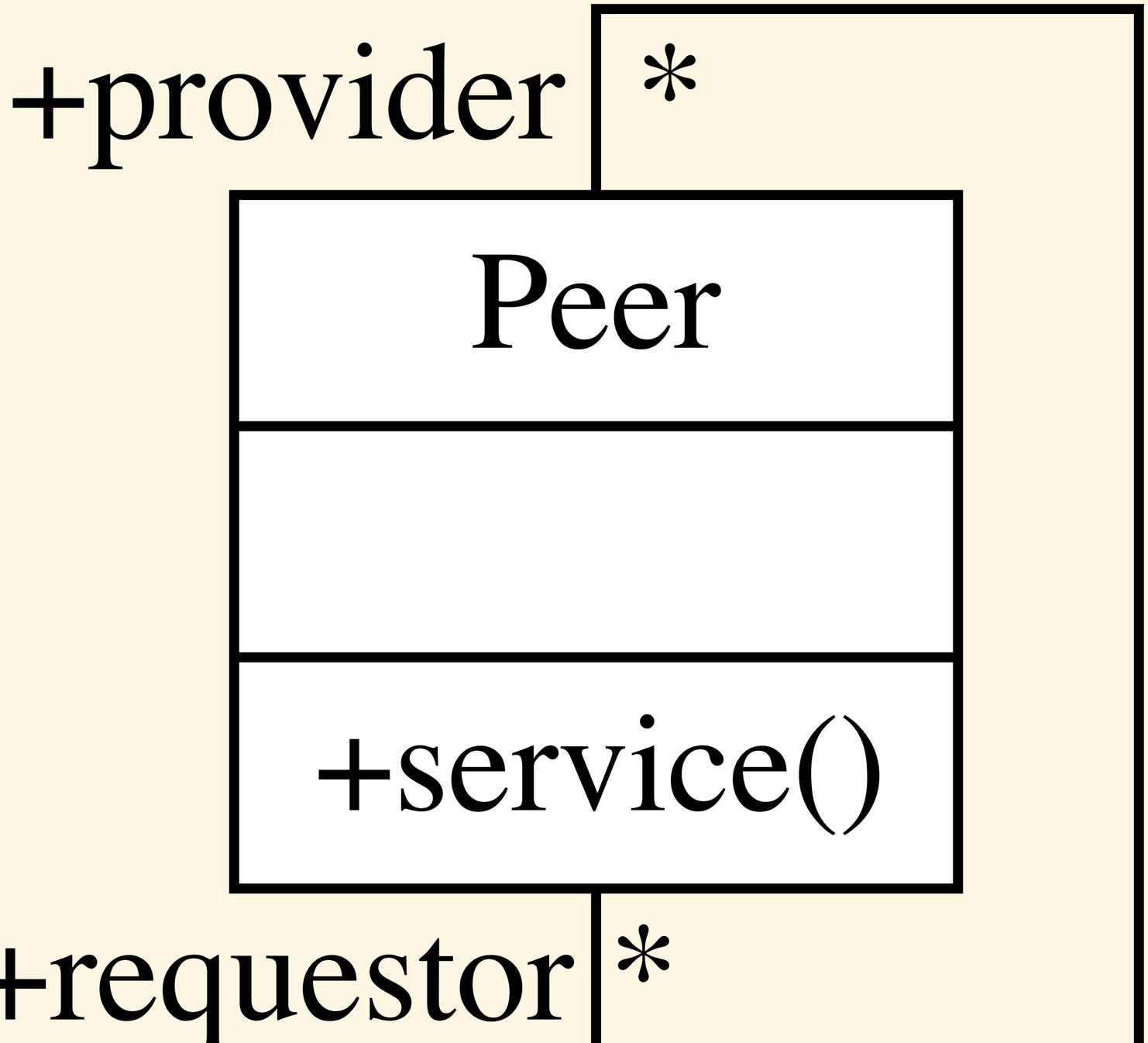
- Subsystems:

  Server provides one or more services to instances of clients

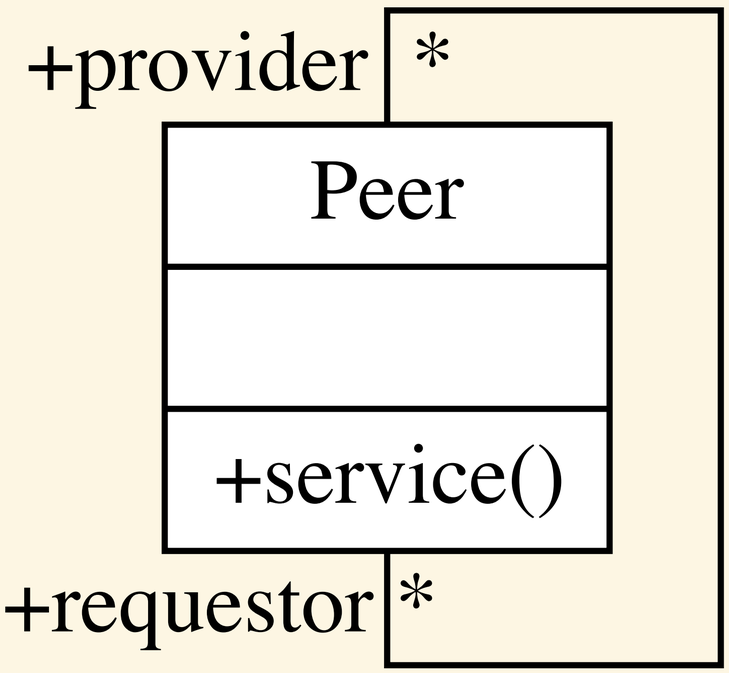
  Clients ask for services, and clients interact with users

- An information system with a central DB is an example

- Web servers (multiple servers)

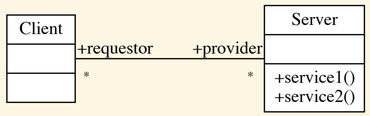

+provider
*
Peer
+service()
+requestor
*

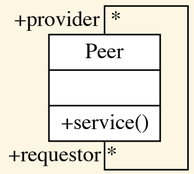
Peer-to-Peer Architecture



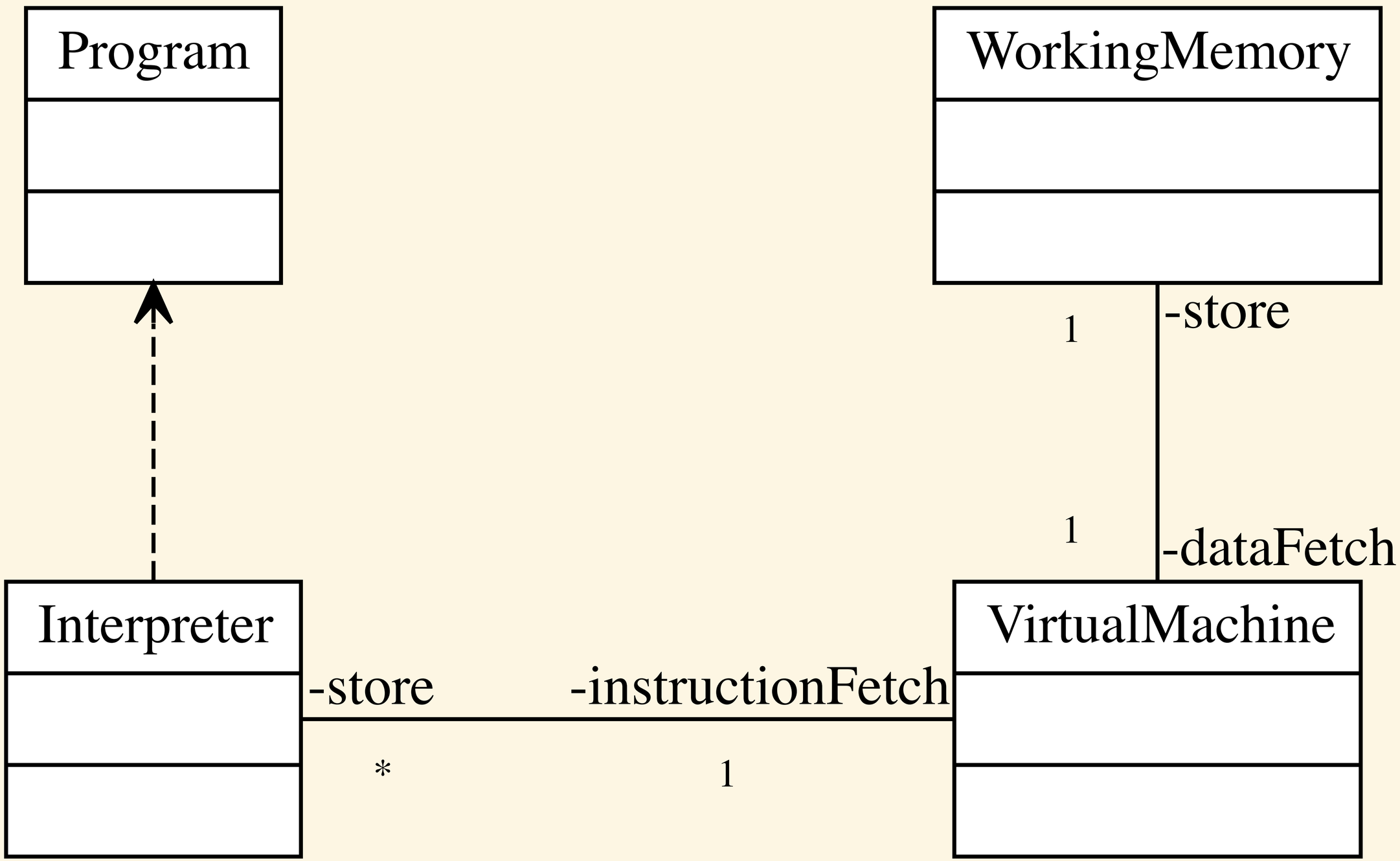
- Generalization of client/server: clients can be servers and vice versa
- The control flow of each subsystem is independent of others except for the synchronization of requests.

Client Server vs. Peer-to-Peer

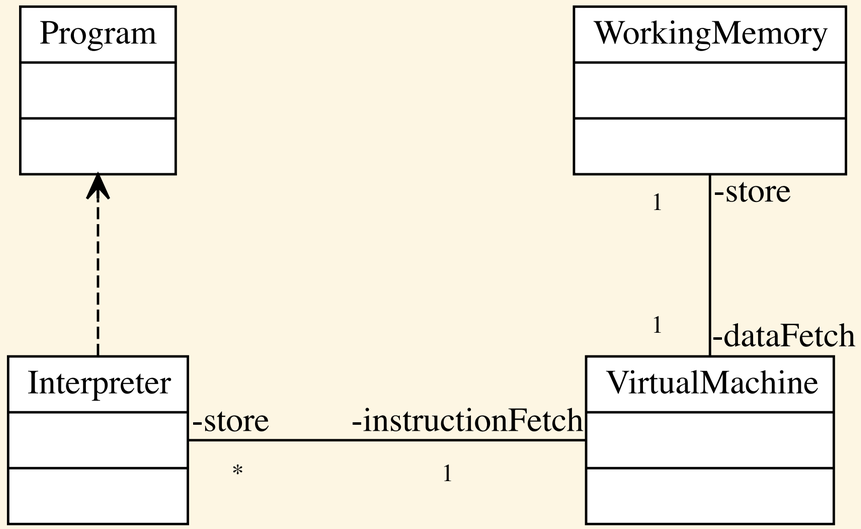




Virtual Machine Architecture Diagram

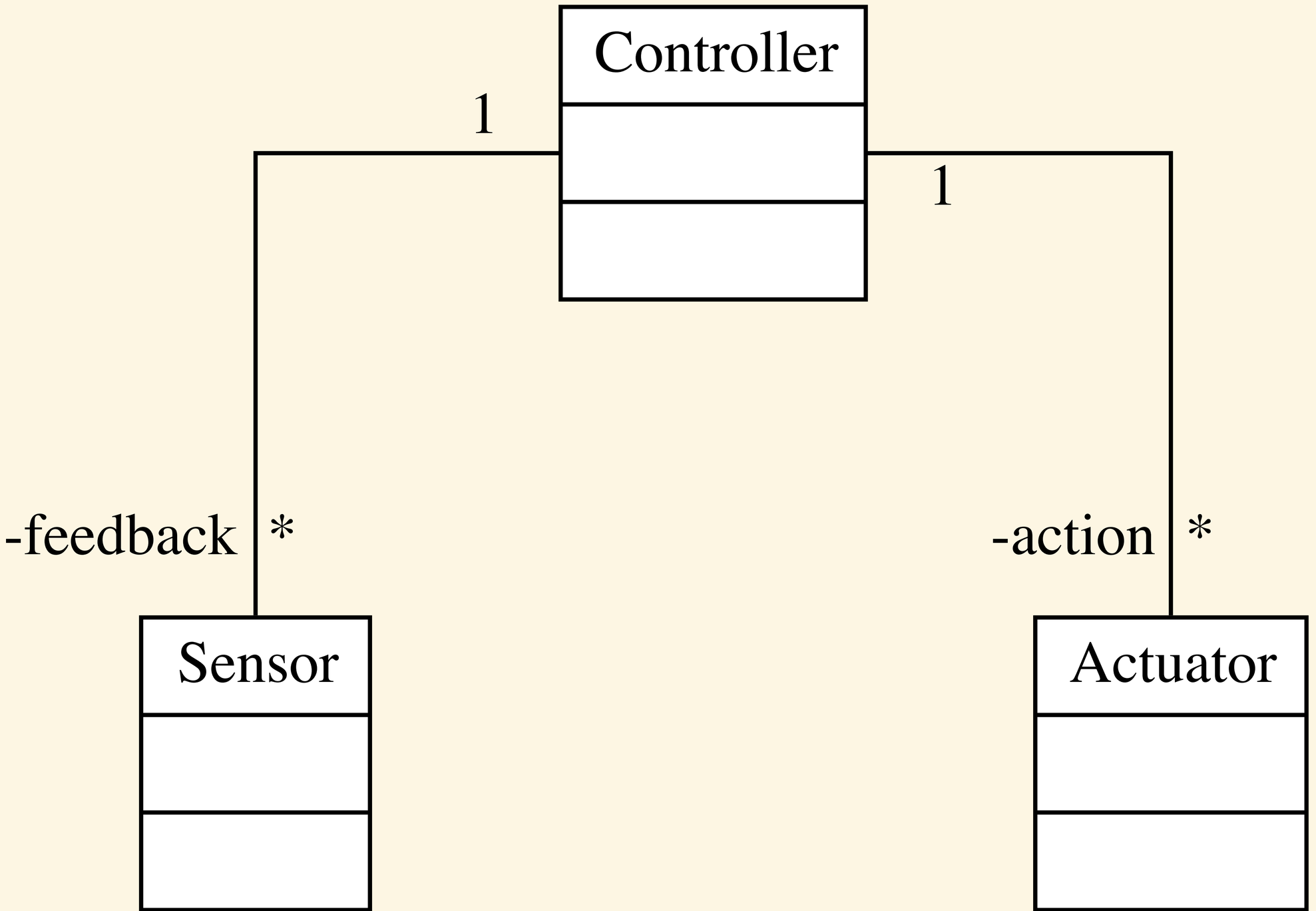


Virtual Machine Architecture

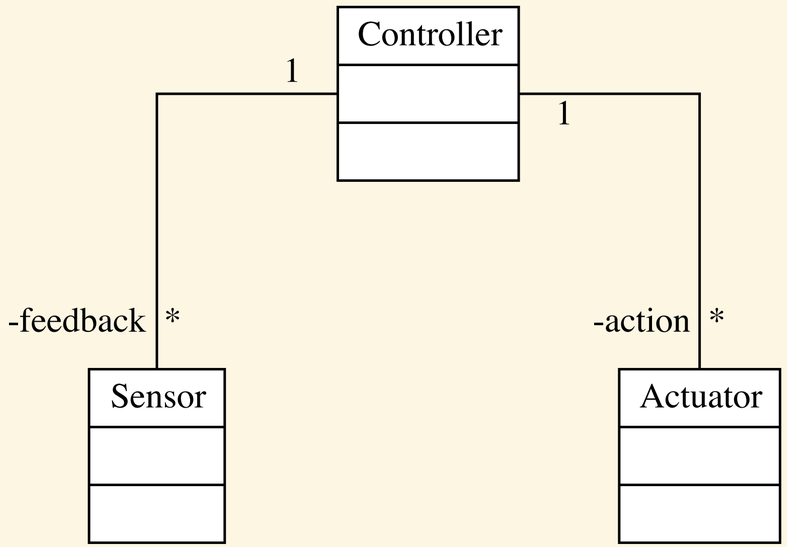


- *Program* Holds a program to execute in a particular language, probably a DSL
- *Interpreter* Knows how to decode the language instruction
- *WorkingMemory* Memory for the VirtualMachine
- *VirtualMachine*
  1. Fetch the instruction
  2. Fetch the data
  3. Execute the instruction
  4. Store the result in the WorkingMemory
  5. Update the Interpreter

Process Control Architecture Diagram

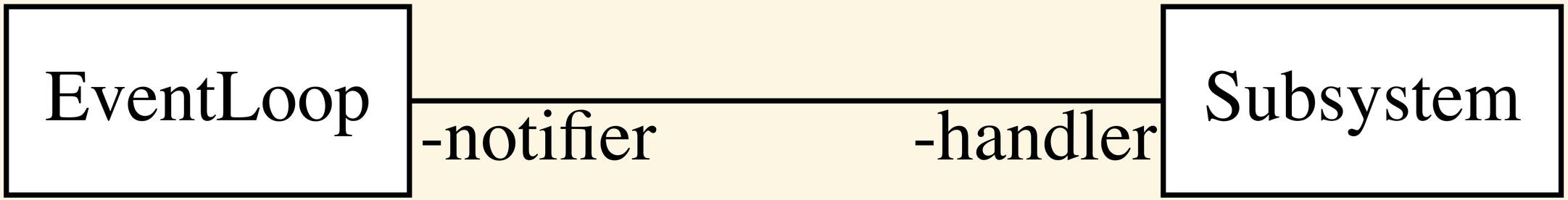


Process Control Architecture

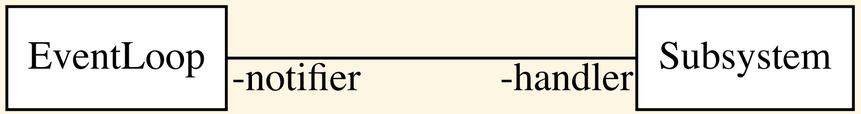


- *Controller* gets *feedback* from a set of *Sensors*
- *Controller* performs *actions* on a set of *Actuators*

Event-Driven Architecture Diagram

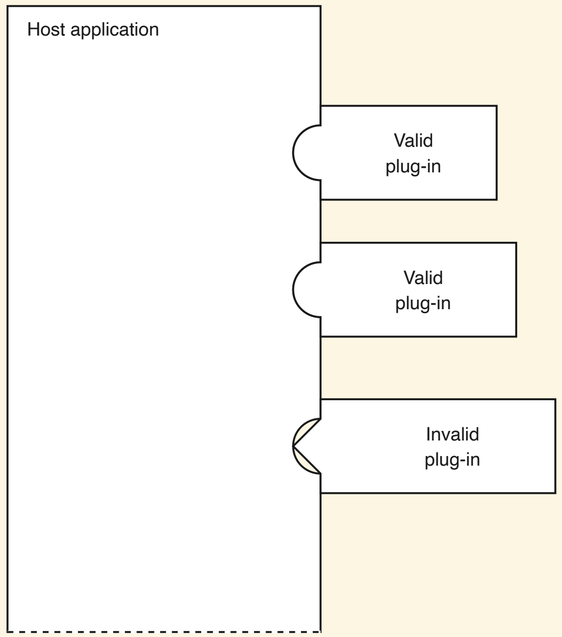


Event-Driven Architecture



- Main *EventLoop* controls *Subsystems*
- *Subsystem* contains *handlers*
- *EventLoop* notifies *Subsystems*

Plug-In Architecture



- *core system*
- *plug-in components*
- Mix and match freely
- E.g., built-in vs. plug-in parsers

Observations

- Each of the architectures controls how you add subsystems to the system
- May define the output and input path
- May define characteristics of the output and input format
- It is difficult to add a subsystem that does not fit the architecture
- Each type may be linked to a particular industry or application type