Design Decomposition

Lecture 10

Overview

Design I: System decomposition

1. Overview of System Design
2. Identify Design Goals
3. Design Initial Subsystem Decomposition

Design II: Refine subsystem decomposition

Design III: Object-level design

Coupling and Cohesion

Goal: Reduction of *complexity while change occurs*

Cohesion measures the dependence among classes

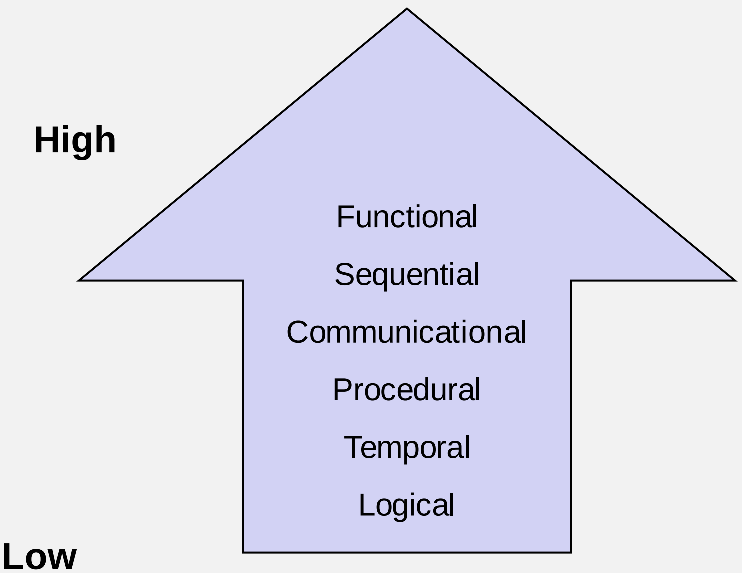
- **High cohesion:** The classes in the subsystem perform similar tasks and are related to each other (via associations)
- **Low cohesion:** Lots of miscellaneous and auxiliary classes, no associations

Coupling measures dependencies between subsystems

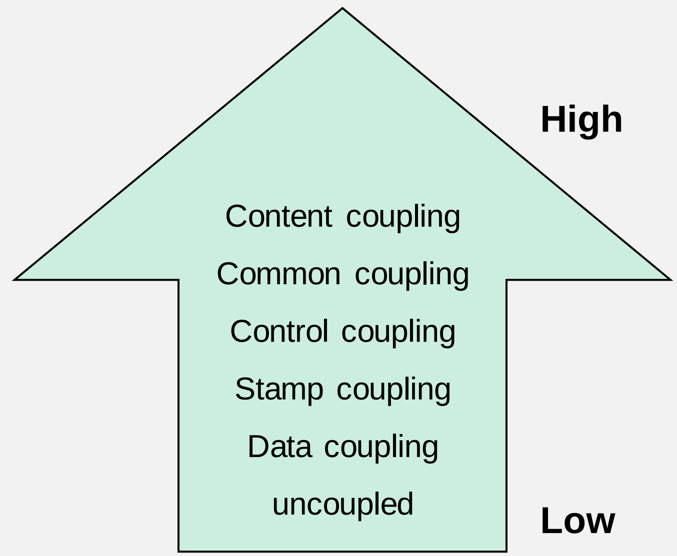
- **High coupling:** Changes to one subsystem will have high impact on the other subsystem (change of model, massive recompilation, etc.)
- **Low coupling:** A change in one subsystem does not affect any other subsystem

Cohesion and Coupling

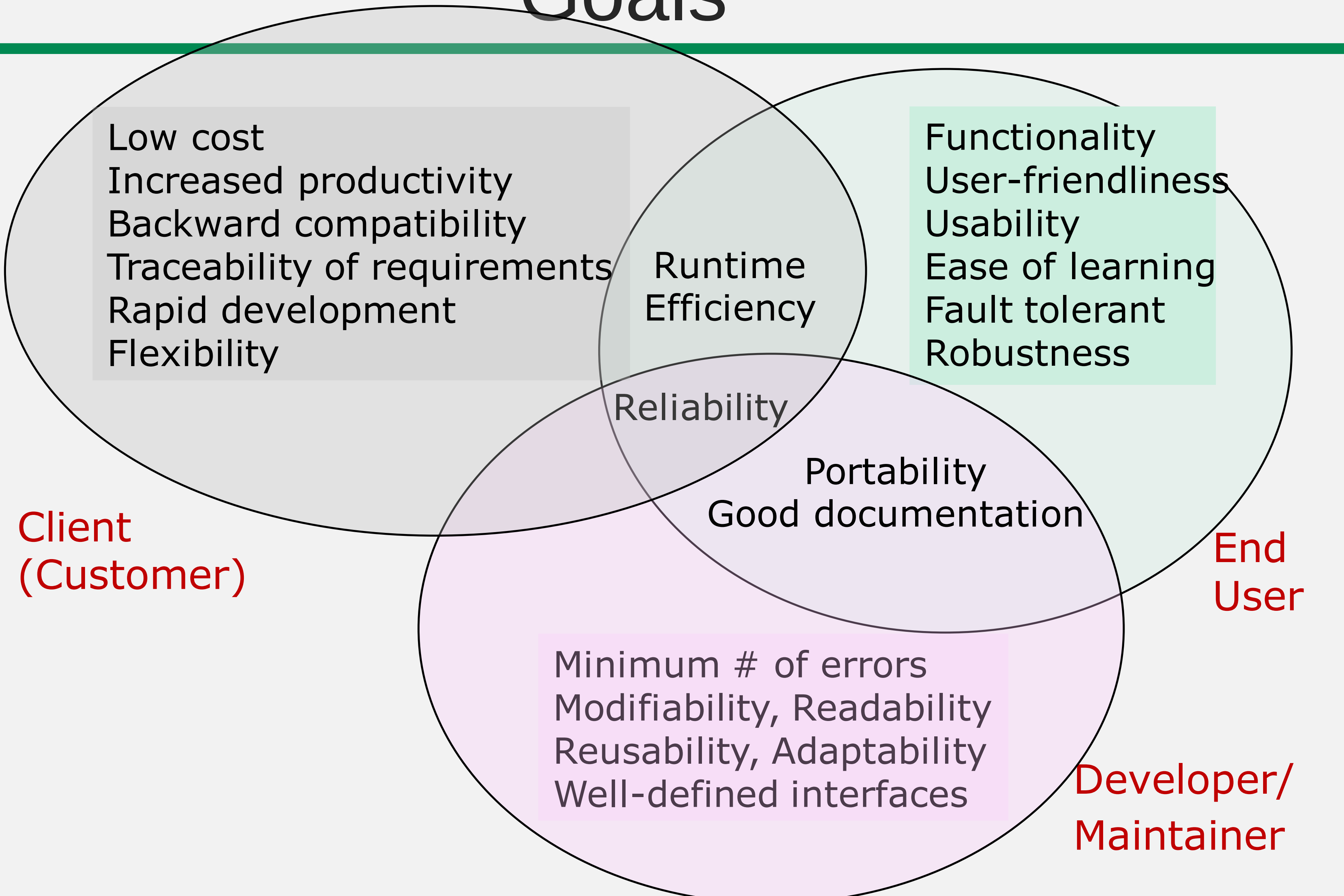
Cohesion



Coupling



Stakeholders have different Design Goals



Subsystem Decomposition

- Subsystem
  - Collection of classes, associations, operations, events and constraints that are closely interrelated with each other
  - The objects and classes from the object model are the "seeds" for the subsystems
  - In UML subsystems are modeled as packages
- Service
  - A set of named operations that share a common purpose
  - The origin ("seed") for services are the use cases from the functional model
- Services are defined during system design.

Choosing Subsystems

Criteria for subsystem selection: Most of the interaction should be within subsystems, rather than across subsystem boundaries (High cohesion).

Does one subsystem always call the other for the service?

Which of the subsystems call each other for service?

Primary Question:

What kind of service is provided by the subsystems (subsystem interface)?

Secondary Question:

Can the subsystems be hierarchically ordered (layered)?

(different layer represent different subsystem)

Subsystems Heurisrics

1. Assign objects identified in one use case into the same subsystem.
2. Create a dedicated subsystem for objects used for moving data among subsystems.
3. Minimize the number of associations crossing subsystem boundaries.
4. All objects in the same subsystem should be functionally related.

Example: TripPlan

Using MyTrip, a driver can plan a trip from a home computer by contacting a trip-planning service on the Web. The trip is saved for later retrieval on the server. The trip-planning service must support more than one driver.

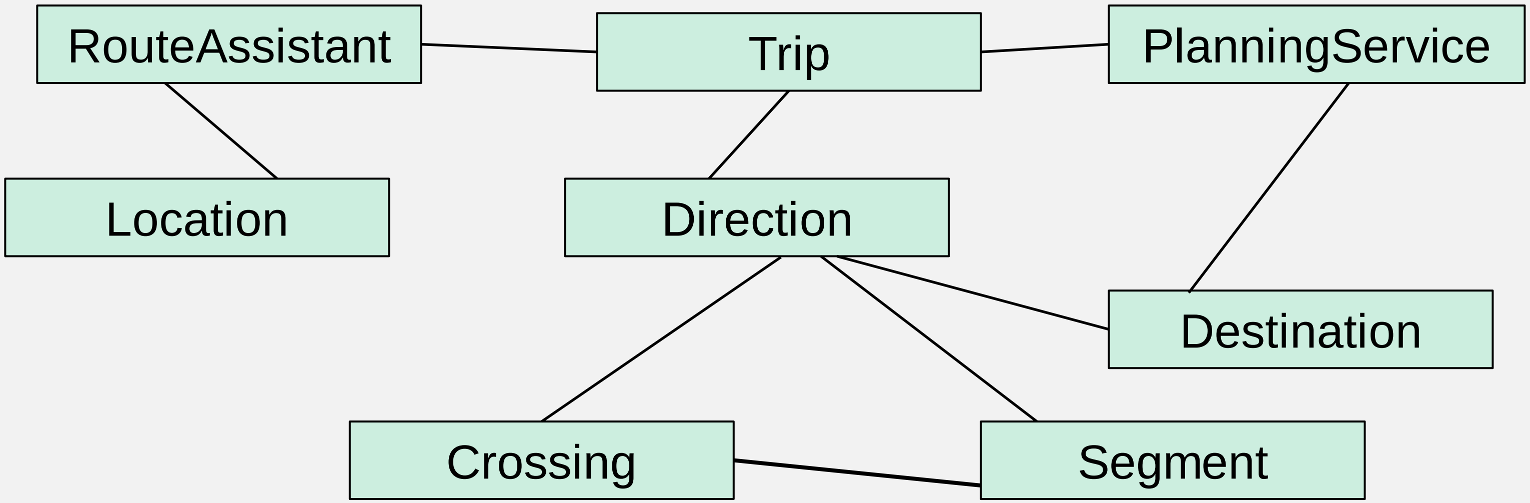
Use Case Name		**Plan Trip**
Flow of Event	1	The Driver activates her computer and logs into the trip-planning Web service.
	2	The Driver enters constraints for a trip as a sequence of destinations.
	3	Based on a database of maps, the planning service computes the shortest way of visiting the destinations in the order specified. The result is a sequence of segments binding a series of crossings and a list of directions.
	4	The Driver can revise the trip by adding or removing destinations.
	5	The Driver saves the planned trip by name in the planning service database for later retrieval.

Example: TripPlan

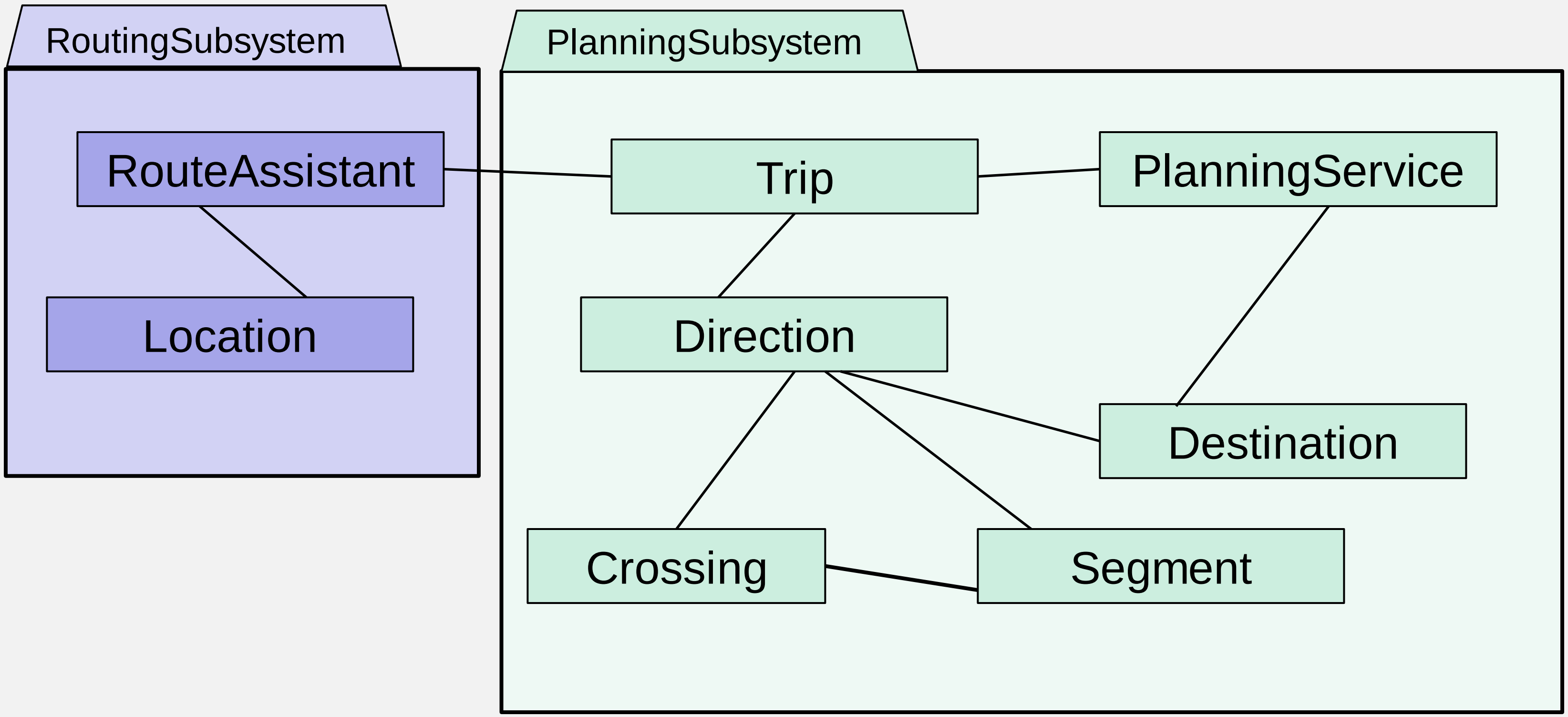
Using MyTrip, a driver can plan a trip from a home computer by contacting a trip-planning service on the Web. The trip is saved for later retrieval on the server. The trip-planning service must support more than one driver.

Use Case Name		**Execute Trip**
Flow of Event	1	The Driver starts her car and logs into the onboard route assistant.
	2	Upon successful login, the Driver specifies the planning service and the name of the trip to be executed.
	3	The onboard route assistant obtains the list of destinations, directions, segments, and crossings from the planning service.
	4	Given the current position, the route assistant provides the driver with the next set of directions.
	5	The Driver arrives to destination and shuts down the route assistant.

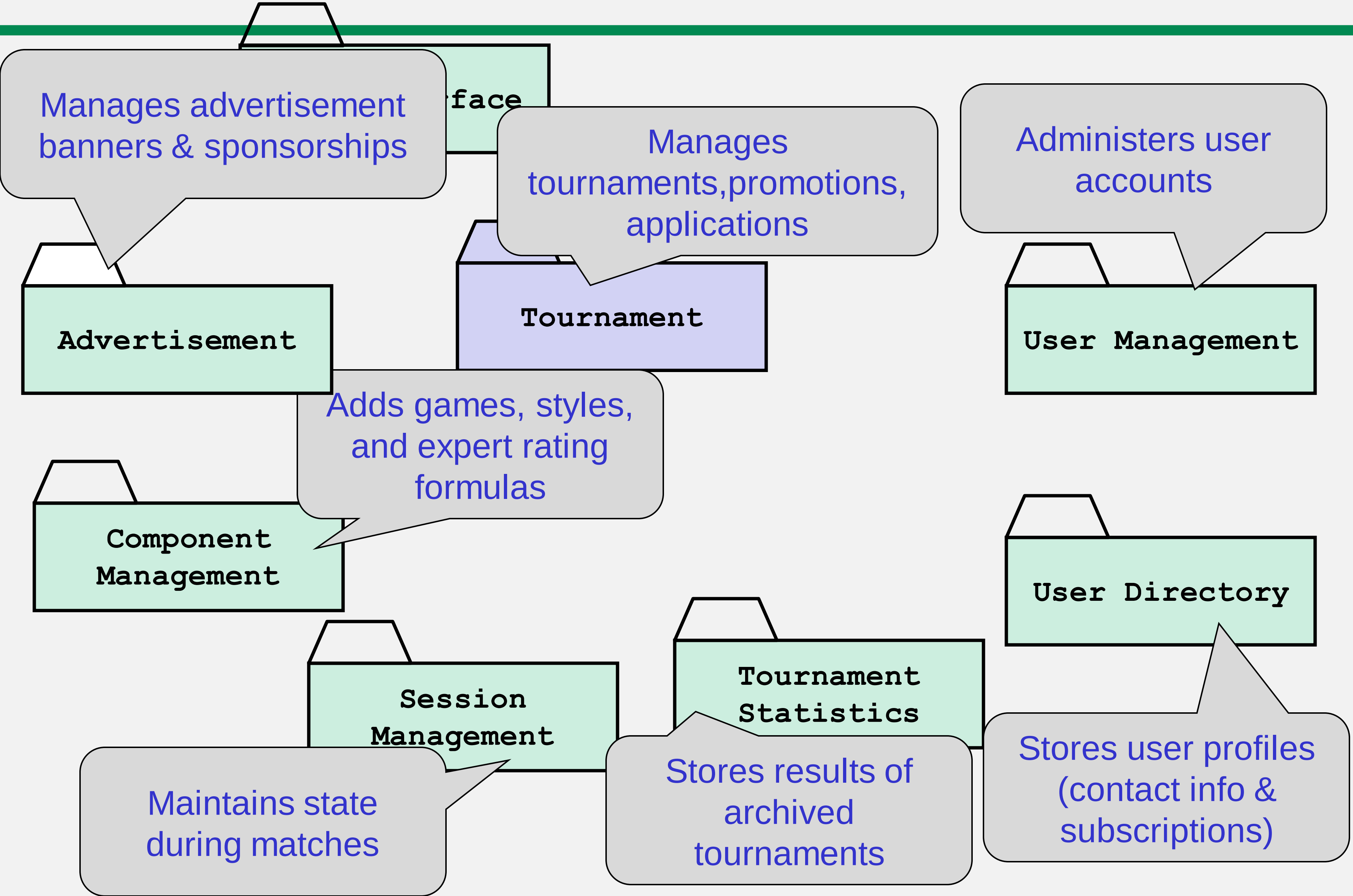
Subsystem Decomposition Example



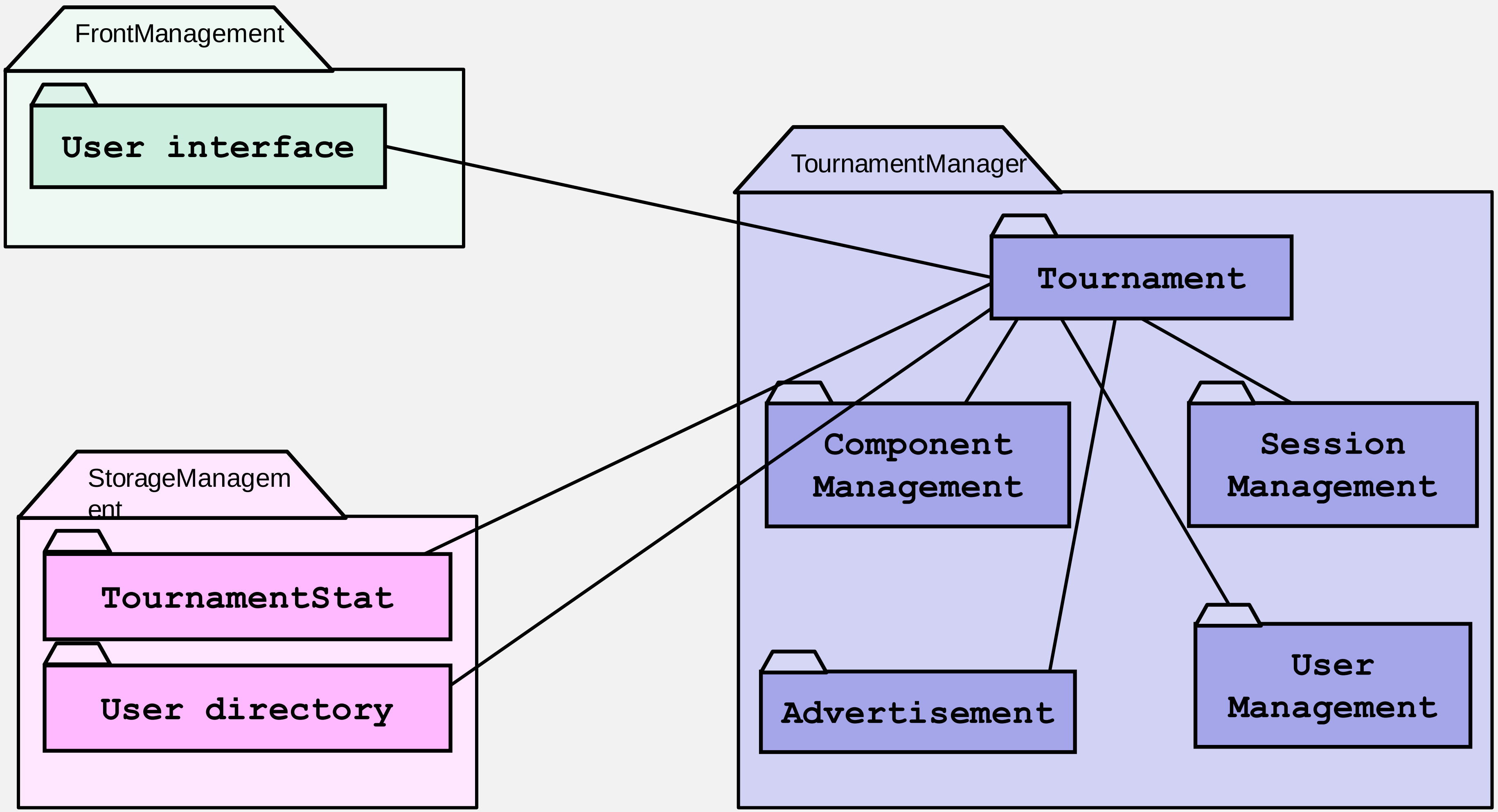
Subsystem Decomposition Example



Example: ARENA Subsystems



Example of a Subsystem Decomposition



Subsystem Interfaces vs API

Subsystem interface: Set of fully typed UML operations

- Specifies the interaction and information flow from and to subsystem boundaries, but not inside the subsystem
- Refinement of service, should be well-defined and small
- *Subsystem interfaces are defined during object design*

Application programmer's interface (API)

- The API is the specification of the subsystem interface in a specific programming language
- APIs are defined during implementation

Example: Notification subsystem

Service provided by Notification Subsystem

- LookupChannel()
- SubscribeToChannel()
- SendNotice()
- UnsubscribeFromChannel()

Subsystem Interface of Notification Subsystem

- Set of fully typed UML operations

API of Notification Subsystem

- Implementation in Java

Subsystem Interface Object

Good design: The subsystem interface object describes *all* the services of the subsystem interface

Subsystem Interface Object

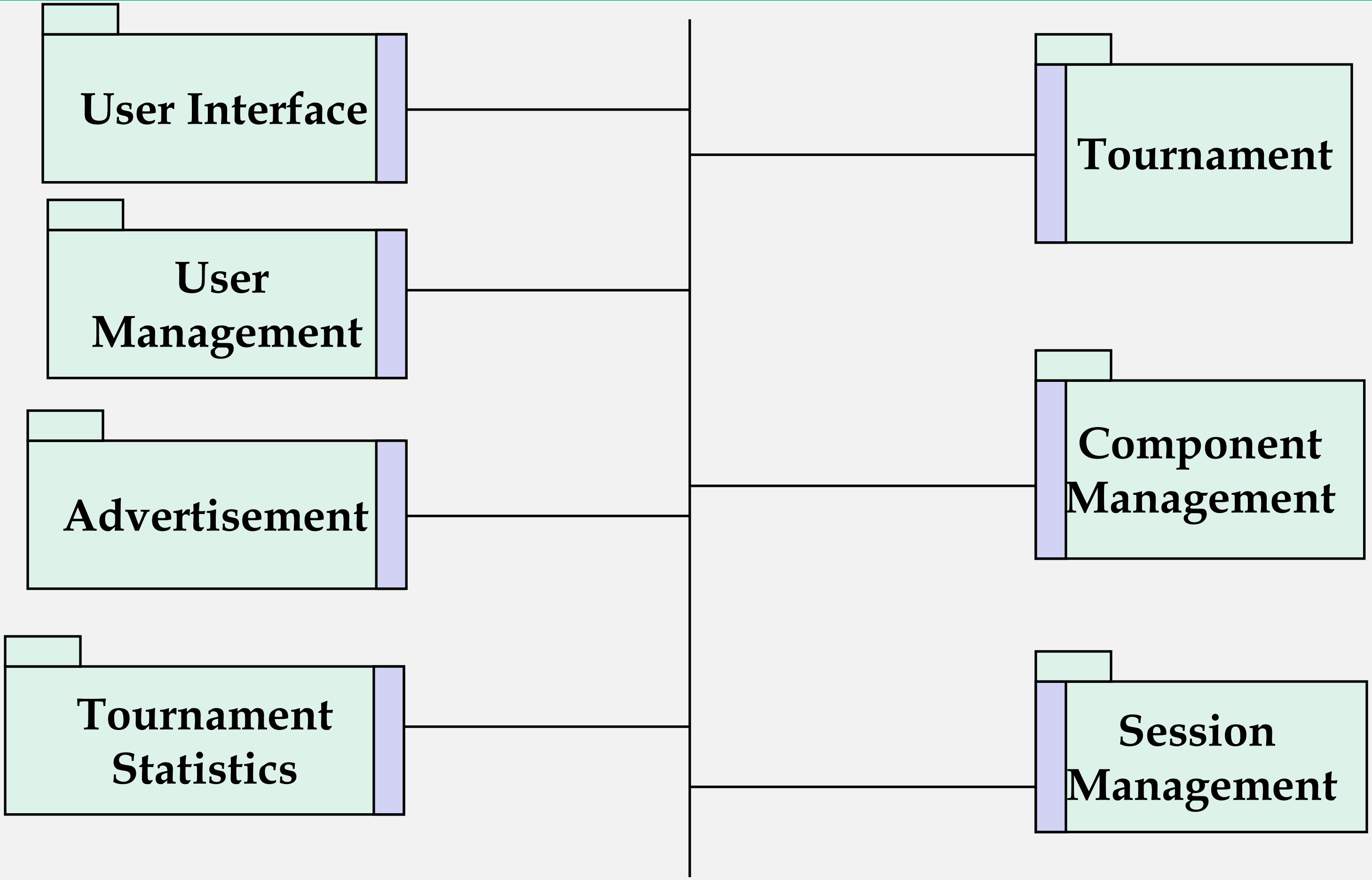
The set of public operations provided by a subsystem

Definition: Subsystem Interface Object

A *Subsystem Interface Object* provides a service

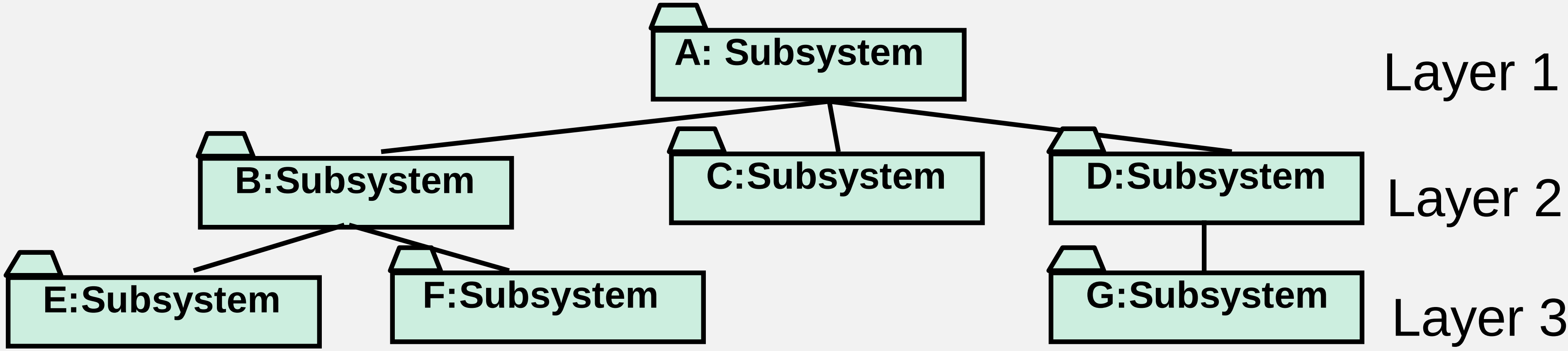
- This is the set of public methods provided by the subsystem
- The Subsystem interface describes all the methods of the subsystem interface object

Good Design: The System as set of Interface Objects



Subsystem Interface Objects

Subsystem Decomposition into Layers



Subsystem Decomposition Heuristics:

No more than 7+/-2 subsystems

More subsystems increase cohesion but also complexity (more services)

No more than 4+/-2 layers, use 3 layers (good)

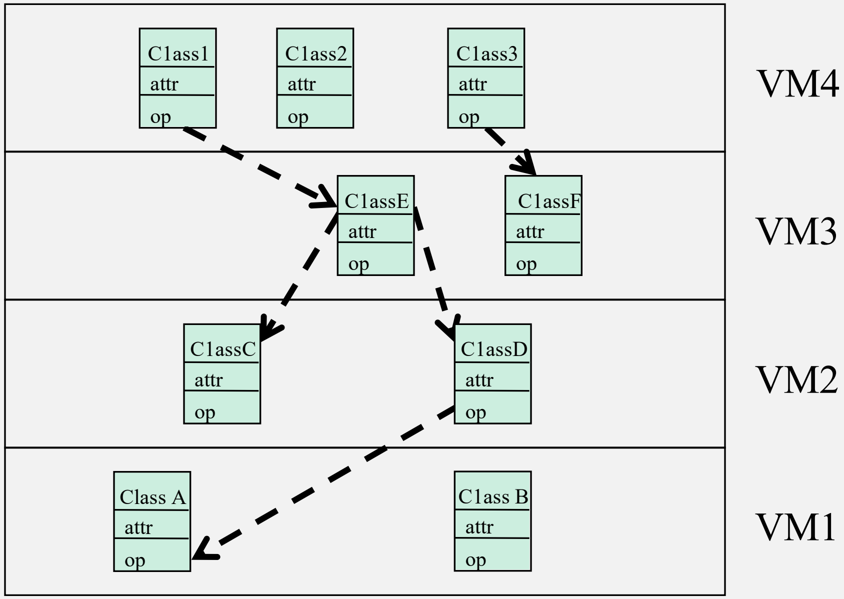
Relationships between Subsystems

- Layer relationship
  - Layer A “Calls” Layer B (runtime)
  - Layer A “Depends on” Layer B (“make” dependency, compile time)
- Partition relationship
  - The subsystem have mutual but not deep knowledge about each other
  - Partition A “Calls” partition B and partition B “Calls” partition A

Opaque Layering

Each virtual machine can only call operations from the layer below

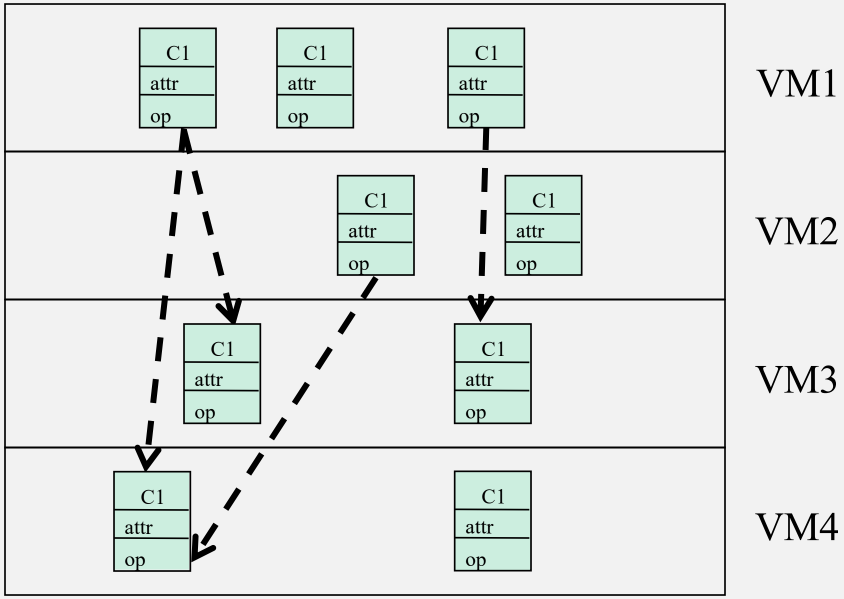
Design goals: Maintainability, flexibility.



Transparent Layering

Each virtual machine can call operations from any layer below

Design goal:
Runtime efficiency



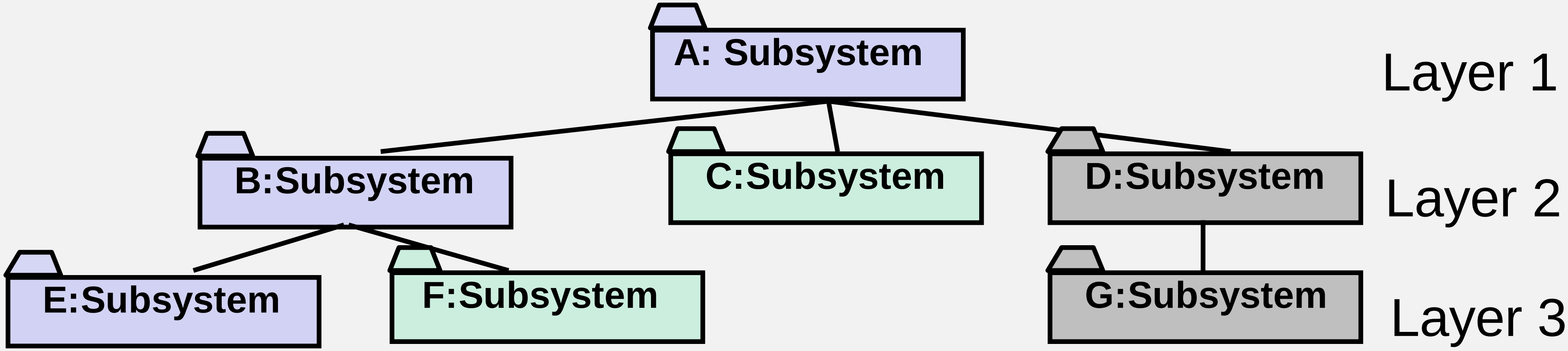
Properties of Layered Systems

Layered systems are *hierarchical*. They are desirable because hierarchy reduces complexity (by low coupling).

Closed architectures are more portable (opaque)

Open architectures are more efficient (transparent)

If a subsystem is a layer, it is often called a virtual slice.

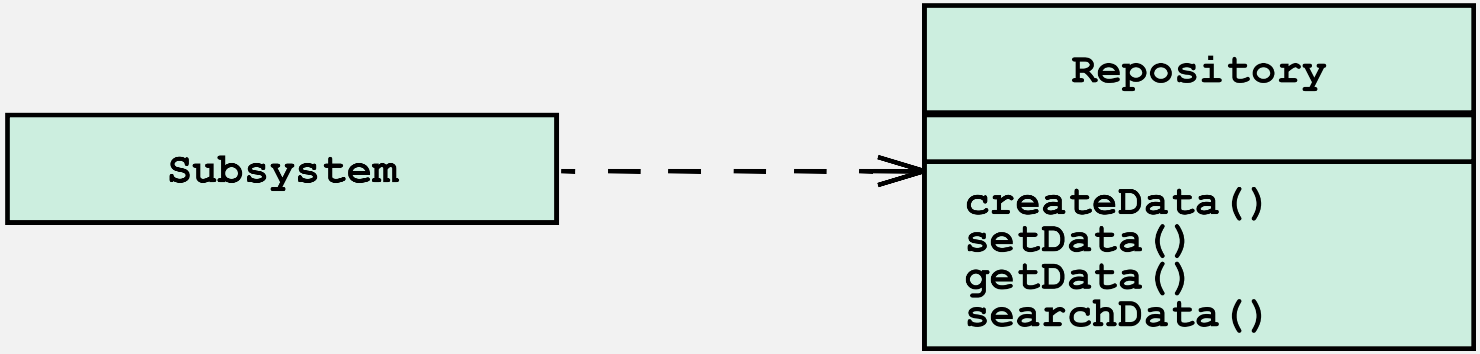


Software Architectural Styles

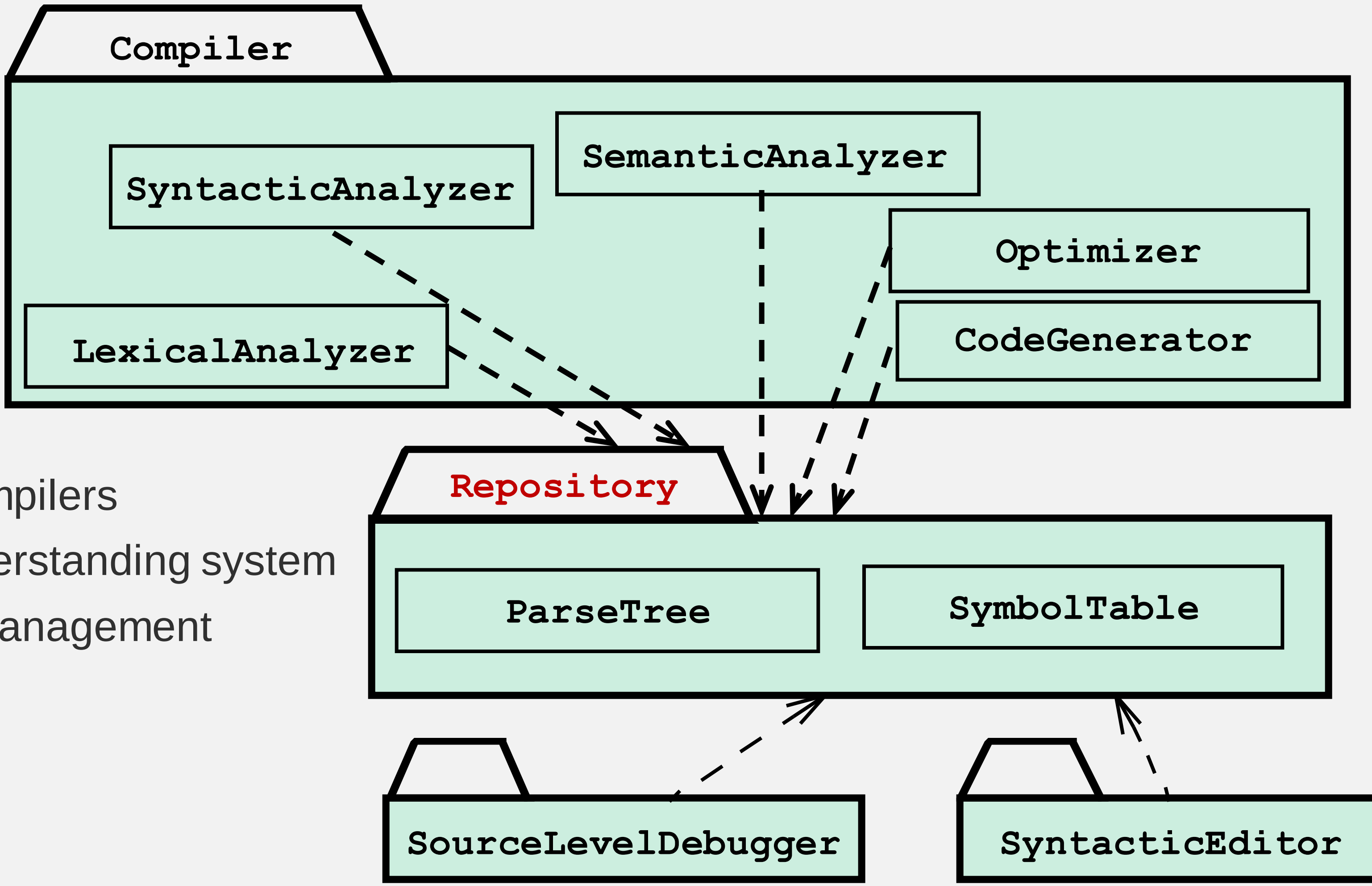
- Subsystem decomposition
  - Identification of subsystems, services, and their relationship to each other.
- Specification of the system decomposition is critical.
- Patterns for software architecture
  - Repository
  - Client/Server
  - Peer-To-Peer
  - Model/View/Controller
  - 3-Tier (4-Tier)

Repository Architectural Style

- Subsystems access and modify data from a single data structure
- Subsystems are loosely coupled (interact only through the repository)
- Control flow is dictated by central repository (triggers) or by the subsystems (locks, synchronization primitives)



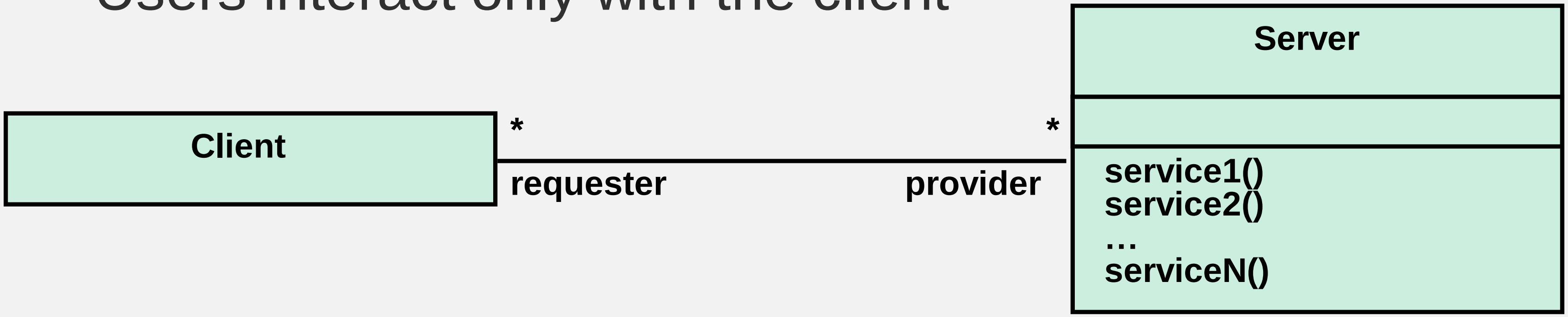
Examples of Repository Architectural Style



- Modern Compilers
- speech understanding system
- Database Management Systems

Client/Server Architectural Style

- One or many servers provides services to instances of subsystems, called clients.
- Client calls on the server, which performs some service and returns the result
  - Client knows the *interface* of the server *(its service)*
  - Server does not need to know the interface of the client
- Response in general immediately
- Users interact only with the client

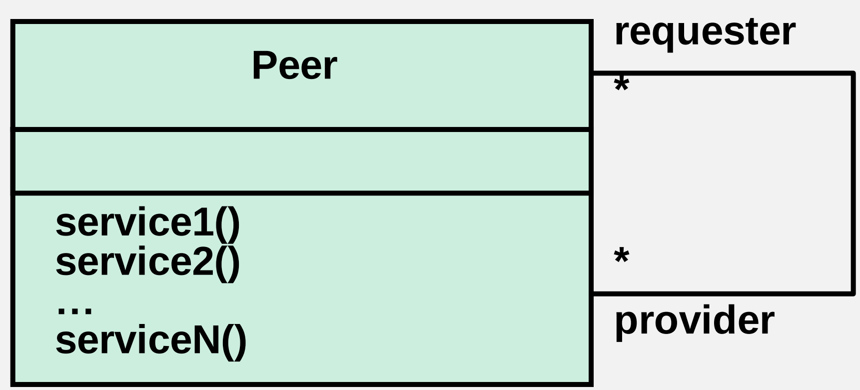


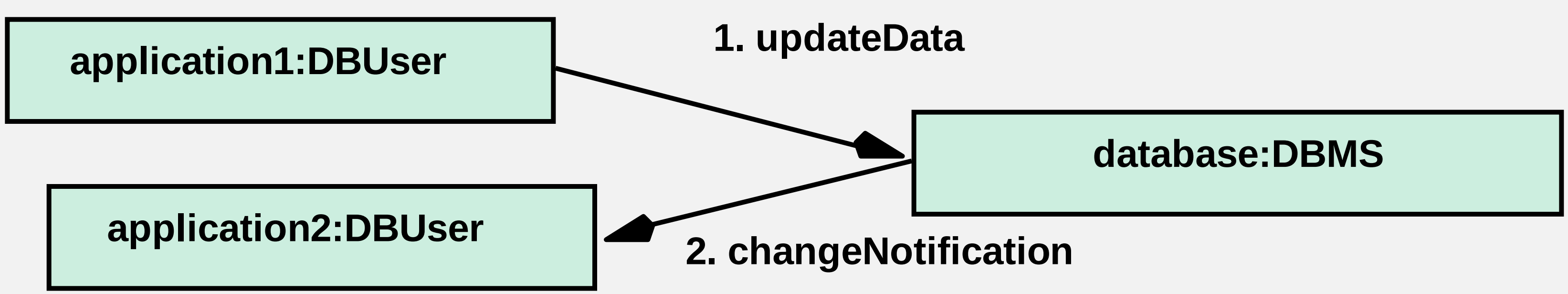
Peer-to-Peer Architectural Style

Generalization of Client/Server Architecture

Clients can be servers and servers can be clients

More difficult because of possibility of deadlocks





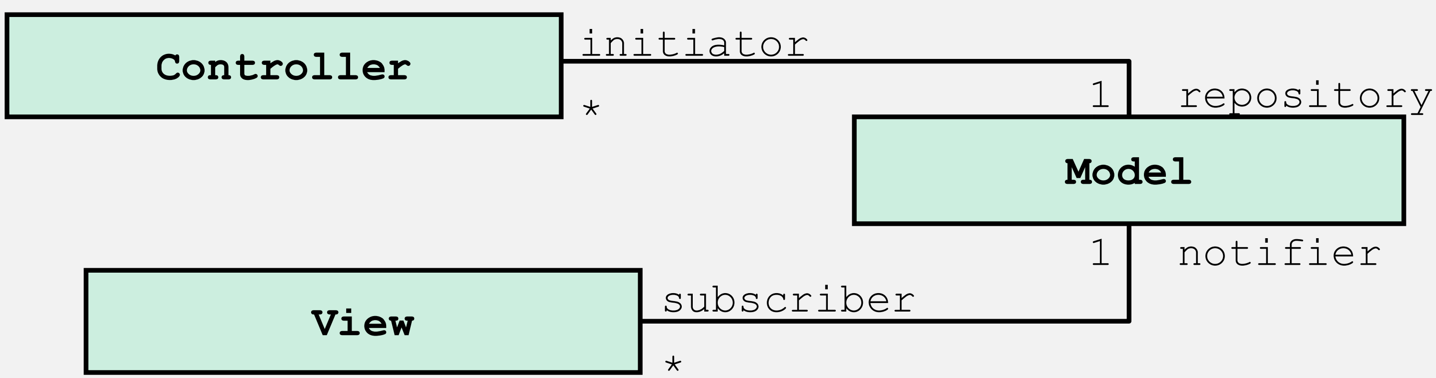
Model/View/Controller

Subsystems are classified into 3 different types

- Model subsystem: Responsible for application domain knowledge
- View subsystem: Responsible for displaying application domain objects to the user
- Controller subsystem: Responsible for sequence of interactions with the user and notifying views of changes in the model.

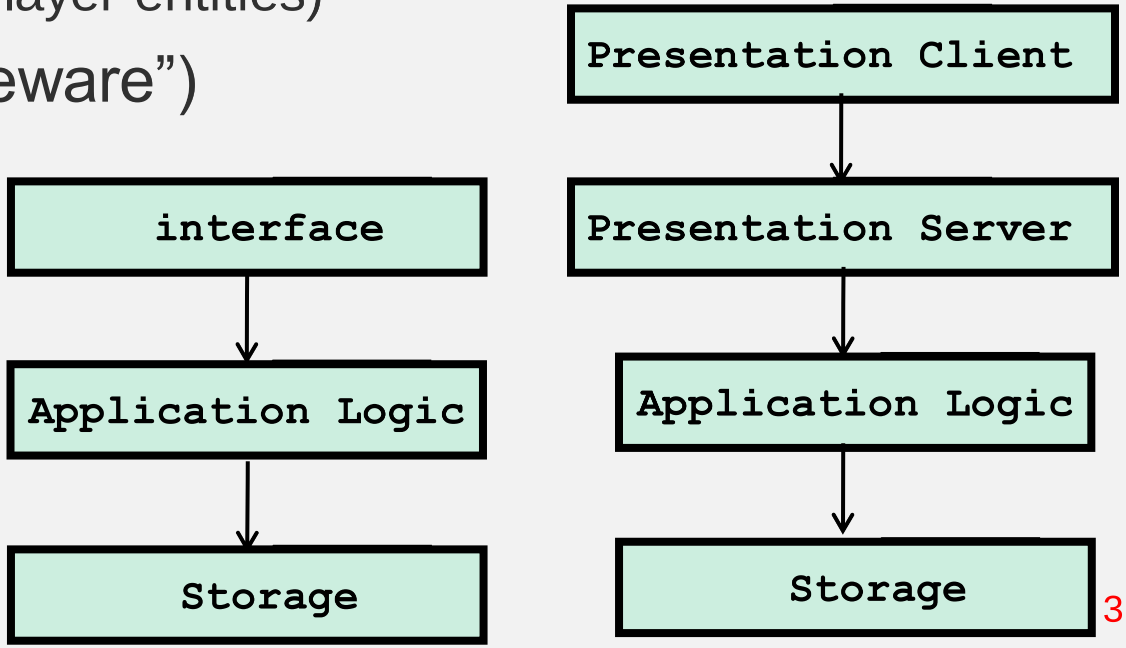
MVC is a special case of a repository architecture:

- Model subsystem implements the central datastructure, the Controller subsystem explicitly dictate the control flow



Three-Tier / Four-Tier

- Interface Layer (boundary objects dealing w. user, *e.g. forms, windows, web pages*,...)
  - Presentation Client Layer (located on user devices, enabling variety of presentation modes, *e.g., desktop, pda, phone*)
  - Presentation Server Layer (located on server, generic versions of client layer entities)
- Logic Layer ("middleware")
- Storage Layer



Summary

System Design

- Reduces the gap between requirements and the (virtual) machine
- Decomposes the overall system into manageable parts

Design Goals Definition

- Describes and prioritizes the qualities that are important for the system
- Defines the value system against which options are evaluated

Subsystem Decomposition

- Results into a set of loosely dependent parts which make up the system

Thank You